FUNDAMENTALS OF COMPUTING AND PROGRAMMING

Second Edition

ABOUT THE AUTHOR

E Balagurusamy, former Vice Chancellor, Anna University, Chennai, is currently Member, Union Public Service Commission, New Delhi. He is a teacher, trainer, and consultant in the fields of Information Technology and Management. He holds an ME (Hons) in Electrical Engineering and PhD in Systems Engineering from the Indian Institute of Technology, Roorkee. His areas of interest include Object-Oriented Software Engineering, Electronic Business, Technology Management, Business Process Re-engineering, and Total Quality Management.

A prolific writer, he has authored a large number of research papers and several books. His best selling books, among others, include:

- *Fundamentals of Computers*
- *Computing Fundamentals and C Programming*
- *Programming in ANSI C*, 5/e
- *Programming in Java*, 4/e
- *Object Oriented Programming with C++,* 5/e
- *Programming in BASIC*, 3/e
- *Programming in C#,* 3/e
- *Numerical Methods*
- *Reliability Engineering*

A recipient of numerous honours and awards, he has been listed in the Directory of Who's Who of Intellectuals and in the Directory of Distinguished Leaders in Education.

FUNDAMENTALS OF COMPUTING AND PROGRAMMING

Second Edition

E Balagurusamy
Chairman
EBG Foundation
Coimbatore

Tata McGraw Hill Education Private Limited
NEW DELHI

McGraw-Hill Offices

New Delhi New York St Louis San Francisco Auckland Bogotá Caracas
Kuala Lumpur Lisbon London Madrid Mexico City Milan Montreal
San Juan Santiago Singapore Sydney Tokyo Toronto

Tata McGraw-Hill

Published by Tata McGraw Hill Education Private Limited,
7 West Patel Nagar, New Delhi 110 008.

Fundamentals of Computing and Programming, 2e

ISBN (13): 978-0-07-107788-0
ISBN (10): 0-07-107788-X

Vice President and Managing Director—MHE: Asia Pacific Region: *Ajay Shukla*
Head—Higher Education Publishing and Marketing: *Vibha Mahajan*

Deputy Manager—Acquisitions (Science, Engineering & Mathematics): *H R Nagaraja*
Sr Editorial Executive—Acquisitions: *Vamsi Deepak Sankar*
Executive—Editorial Services: *Sohini Mukherjee*
Production Executive: *Anuj K Shriwastava*
Sr Production Manager: *Satinder S Baveja*

Marketing Manager—Higher Education: *Vijay Sarathi J*
Sr Product Specialist: *John Mathews*

General Manager—Production: *Rajender P Ghansela*
Assistant General Manager—Production: *B L Dogra*

Typeset at Text-o-Graphics, B1/56, Arawali Apartment, Sector 34, Noida 201 301 and printed at
Print Shop Pvt Ltd., No.4/310, Gandhi Street, Kottivakkam, Old Mahabalipuram Road, Chennai – 600096

Cover Design: Print Shop Pvt Ltd

DZLYYRBFRACDZ

The McGraw·Hill Companies

CONTENTS

PREFACE

The developments in digital electronics and related technologies during the last few decades have ushered in the second Industrial Revolution that is popularly referred to as the Information Revolution. Computer technology plays an ever-increasing role in this new revolution. A sound knowledge of how computers work and how they process data and information has, therefore, become indispensable for anyone who seeks employment not only in the area of IT but also in any other field.

Rightly so, many institutions and universities in India have introduced a subject covering the **Fundamentals of Computing and C Programming** for the undergraduate students. This book is designed primarily to address the topics covered under this subject.

WHY C LANGUAGE?

C is a powerful, flexible and elegantly structured programming language. It is also a machine-independent language. Since it combines the features of a high-level language with the elements of the assembler, it is suitable for both systems and applications programming. C is undoubtedly the most popular and most widely used general-purpose language today.

WHY IS THIS BOOK A WINNER?

This book ensures a smooth and successful transition to being a skilled C programmer. The book uses a simple-to-complex and easy-to-learn approach throughout. The concept of 'learning-by-example' has been stressed everywhere in the book. Each feature of the language is treated in depth followed by a complete program example to illustrate its use. Wherever necessary, concepts are explained pictorially to facilitate better understanding. The book presents a contemporary approach to programming, offering a combination of theory and practice.

PEDAGOGICAL FEATURES

- Bottom-up approach of explaining concepts
- Microsoft Office Software illustrated with screen shots
- Algorithms and Flowcharts conversed extensively

- Codes with Comments provided throughout the book to illustrate the use of various features of the language
- Supplementary Information and Notes that complement but stand apart from the text have been included in special boxes
- Case Studies illustrate common ways C features are put together and also demonstrates real-life applications
- Points-to-Remember and Summary at the end of each chapter helps the reader recollect the topics covered with little effort
- Variety of Review Questions to help test conceptual understanding

HOW IS THE BOOK ORGANIZED?

The book covers the history, evolution, and organization of computers along with the various number systems in **Chapter 1**. **Chapter 2** explains 'Computer Software' and the different steps involved in software development. Evolution of Internet, the basic terminologies and applications of Internet are also covered in Chapter 2. **Chapter 3** delves into problem solving, logic and the standard Microsoft Office Applications. **Chapter 4** introduces the reader to programming using C language with detailed coverage on character-sets, C-Tokens, Constants, Variables, Data-Types, Operators, Expressions, Decision-making and Input-Output operations. Arrays and ordered arrangement of data elements are important to any programming language and have been covered in **Chapter 5** along with Strings, Functions, Structures, Unions and Pointers. This chapter ends with the coverage of Pre-processor Directives and an introduction on How to Develop a C Program.

RESOURCES AVAILABLE ON THE WEB

The McGraw-Hill Online Learning Centre of the book can be accessed at http://www.mhhe.com/balagurusamy/fcpau. The site gives the student an opportunity to explore in greater depth the features and application of the C language. It contains case studies and a few sample C programs are also provided.

Feedback

I welcome any constructive criticism of the book and will be grateful for any appraisal by the readers. Feedback to improve the book will be highly appreciated.

E Balagurusamy

PUBLISHER'S NOTE

Tata McGraw Hill Education looks forward to receiving from teachers and students their valuable views, comments and suggestions for improvements, all of which may be sent to tmh.corefeedback@gmail.com, mentioning the title and author's name in the subject line.

INTRODUCTION TO COMPUTERS

1

CHAPTER OBJECTIVES

In this chapter, we will learn:

1. How do computers input, store and process data to generate the output.
2. The various characteristics of computers.
3. The improvements in functioning of computers in the last few years.
4. The five generations of computers.
5. How to classify computers on the basis of operating principles, applications and size.
6. The basic computer organisation.
7. The various number systems like binary, decimal, hexadecimal, etc.

CHAPTER OUTLINE

1.1 INTRODUCTION

Computers are used for a variety of purposes, starting from simple arithmetic calculations to a very complex data analysis such as weather forecasting. They have become an integral part of man's everyday life. From the end user's standpoint, a computer looks like a simple device that automates the otherwise manual computational tasks. However, when we try to explore the basic anatomy of a computer, we get to learn how it performs both simple and complex tasks in an organised manner with the help of discrete components seamlessly integrated with each other.

We will begin this chapter by explaining the key characteristics of a modern-day computer system. We will then explore how computers have evolved over the last six decades. The evolution of computers has been distinctly divided into five generations. Each of these generations is marked by a technological revolution that made the computers of that era take a big leap from its predecessors.

There is not a single factor that can uniquely categorise the modern-day computers. In this chapter, we will learn how a computer is categorised on the basis of operating principles, applications, and size. Further, we will learn the basic computer organisation; that is, how the various components of a computer interact with each other and work in unison. Finally, we will learn about the various number systems that a computer supports and the techniques that are used to convert data from one number system to another.

1.2 OVERVIEW OF COMPUTERS

A computer is an electronic machine that takes input from the user, processes the given input and generates output in the form of useful information. A computer accepts input in different forms such as data, programs and user reply. Data refer to the raw details that need to be processed to generate some useful information. Programs refer to the set of instructions that can be executed by the computer in a sequential or non-sequential manner. User reply is the input provided by the user in response to a question asked by the computer. The main task of a computer system is to process the given input of any type in an efficient manner. Therefore, the computer is also known by various other names such as data processing unit, data processor and data processing system.

A computer includes various devices that function as an integrated system to perform several tasks described above. These devices are:

- **Central Processing Unit (CPU)** It is the processor of the computer that is responsible for controlling and executing instructions in the computer. It is considered as the most significant component of the computer. It is the "brain" of the computer.
- **Monitor** It is a screen, which displays information in visual form, after receiving the video signals from the computer.
- **Keyboard and Mouse** These are the devices, which are used by the computer, for receiving input from the user.

Figure 1.1 shows the various components of a computer.

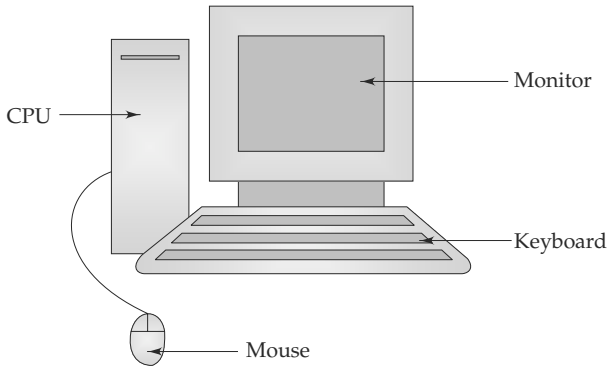


Fig. 1.1 The components of computer

The unique capabilities and characteristics of a computer have made it very popular among its various users, including engineers, managers, accountants, teachers, students, etc.

1.3 APPLICATIONS OF COMPUTERS

Today, computers are used in almost every sphere of life such as education, communication and banking. The users from different locations can easily and quickly communicate with each other with the help of computers. The use of computers has reduced the paper work to a large extent. Thus, computers have become a basic need to perform various tasks in our day-to-day life. The various application areas of computers are as follows:

- **Education** Computers are used in schools and colleges to teach students in a better and easy way. The students can get more information about a specific topic or subject using the Internet. Computers help in easy learning by creating presentations on a specific topic. Today, students can fill their application forms and give their exams online that facilitates distance education.
- **Business** Computers are used in different types of businesses to store a large amount of information in the form of a database. Using computers, business meetings can be held between people sitting at remote locations through web conferencing. Buyers and sellers can conduct business online through the use of computers and Internet.
- **Communication** Computers that are connected with each other through Internet can be used to transfer data to and from other computers. In order to establish communication between two users, e-mail is one of the most common mediums that is used. Through e-mail users can send/receive text messages, graphic messages and file attachments.
- **Science** Computers are used by various scientists for the purpose of research and development. They generally make use of computer for research and analysis of new theories. With the help of computers, scientists are moving towards the possibility of predicting natural disasters such as earthquake and tsunami.

- **Engineering** Computers are used by engineers for the creation of complex drawings and designs while working in different fields like automobiles and construction.
- **Entertainment** Computers are used in the entertainment industry for creating graphics and animations. There are various free as well as proprietary graphics software available for creating graphics and animations.
- **Banking** Now days, computers are being increasingly used for online banking. Through online banking, the users or customers can transfer and receive money by using computers and Internet. Some banks also provide the facility of online bill payment through their websites.
- **Health** Computers are used by doctors to diagnose various kinds of diseases and ailments. Several analog and digital devices are connected with computers enabling the doctors to monitor the condition of a patient and view the internal organs of the body. Further, bioinformatics has evolved as an altogether new science that deals with the application of information technology in the field of molecular biology.

1.4 CHARACTERISTICS OF COMPUTERS

The characteristics and capabilities of a modern digital computer include, among others:

- **Speed** A computer is a fast electronic device that can solve large and complex problems in few seconds. The speed of a computer generally depends upon its hardware configuration.
- **Storage capacity** A computer can store huge amount of data in its different storage components in many different formats. The storage area of a computer system is generally divided into two categories—main memory and secondary storage.
- **Accuracy** A computer carries out calculations with great accuracy. The accuracy achieved by a computer depends upon its hardware configuration and the instructions.
- **Reliability** A computer produces results without any error. Most of the errors generated in the computer are human errors that are created by the user itself. Therefore, they are very trustworthy machines.
- **Versatility** Computers are versatile machines. They can perform many different tasks and can be used for many different purposes.
- **Diligence** Computers can perform repetitive calculations any number of times with the same accuracy.

Computers do not suffer from human traits, such as tiredness, fatigue, lack of concentration, etc. Although computers are highly reliable and versatile machines, they do possess certain limitations. Since computers are capable of doing only what they are instructed to do, any wrong instruction (or faulty logic) or any wrong data may result in erroneous output. This is popularly known as "Garbage-In, Garbage-Out" (GIGO).

A computer is a dumb machine and therefore lacks "common sense". Anything it does is a result of human instructions. It carries out instructions as long as it can understand them,

no matter whether they are right or wrong. Although computers can be instructed to make certain decisions based on mathematical or logical equations, they cannot make decisions in situations where qualitative considerations are involved.

1.5 EVOLUTION OF COMPUTERS

In ancient times, people used different mechanical devices and methods for performing computing operations. However, these devices and methods used for calculations were not very fast and accurate. This fact led to the invention of a computer. The computer was developed to produce accurate results at a very fast speed. Since its invention, the computer has gone through several phases of technological developments. We can understand these developments by just looking at the history of computers. Before the invention of any type of calculating device, people carried out simple arithmetic calculations, such as addition and subtraction on their fingers. This method of counting is still preferred in schools as it teaches children how to count. In ancient times, people also used stones for representing numbers and carrying out simple calculations. These stones were then kept at a place that was suitable for adding and subtracting more stones. In this manner, people performed simple arithmetic calculations. However, the use of stones did not constitute the only method of performing calculation at that time. People also used other devices—such as notches in a stick and knots in a rope—for carrying out simple calculations. However, the purpose of each device was to represent numbers. Some of the early computing devices were manually operated, while the later computing devices were completely automated.

1.5.1 Manual Computing Devices

The idea of using stones for representing numbers and putting them at a place for performing simple calculations led to the invention of a device called sand table. A sand table was a device that arranged stones in three channels in the sand. Each channel could have a maximum of 10 stones. The addition operation was performed on this device by incrementing the count of right hand channel by one and by adding one stone in it. As soon as the right hand channel reached its maximum capacity, the stones were removed from that channel and one stone was added to the left hand channel. Figure 1.2 shows the idea of sand table used for the purpose of calculations.

The idea of sand table led to the development of a fast calculating device of that time, which was known as abacus. Unlike the sand table, the abacus replaced the sand frame with a wooden frame, the grooves with wires and the stones with beads. An abacus was also known as a counting frame and became popular among the people in Asia Minor around 5000 years back. This device is still in use in many parts of the world. In this device, the wooden frame consists of many wires, with beads sliding on them. The user of an abacus can perform arithmetic operations by sliding the beads on the wires by hand. Figure 1.3 shows an abacus consisting of beads on different wires of a wooden frame.

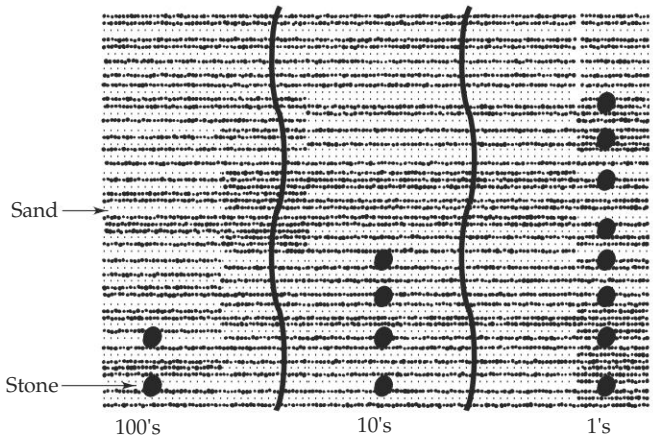


Fig. 1.2 **A sand table**

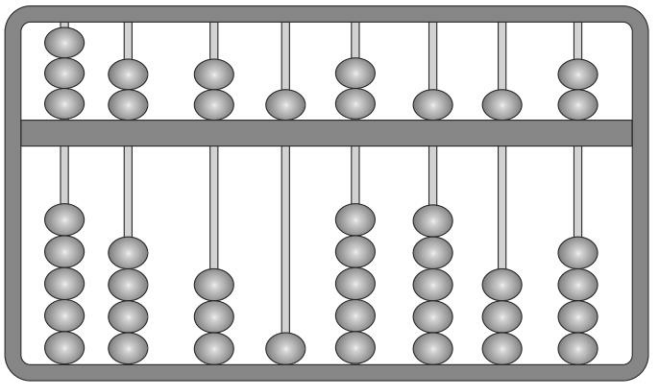


Fig. 1.3 **An abacus**

Another complicated manual computing device called napier bones was developed by John Napier in the year 1614. This device was specially designed for the multiplication and quotient of numbers. Napier bones consisted of a board whose left edge was divided into 9 squares. These 9 squares were used to hold the numbers from 1 to 9. It also consisted of 10 rods, which were made up of strips of ivory bones. The multiplication of two numbers with Napier bones could be performed in a faster manner, if one of the numbers involved in multiplication was of a single digit only. Figure 1.4 shows the arrangement of bones for the multiplication of two numbers—one is of four digits and the other of one digit.

Figure 1.4 shows the process of multiplying the number 5437 with any other number of a single digit. For instance, suppose we want to multiply 5437 with 6. The computation process with this device starts with the rightmost bone and proceeds towards the left bones. The last digit in the 6th row of the 7-bone is 2, so the rightmost digit of the multiplication output is 2. After this, add the two adjacent numbers in the same row forming the parallelogram, which are 8 and 4. The addition of these two numbers is 12, so the next rightmost digit of the

1	5	4	3	7
2	10	08	6	14
3	15	12	09	21
4	20	16	12	28
5	25	20	15	35
6	30	24	18	42
7	35	28	21	49
8	40	32	24	56
9	45	36	27	63

Fig. 1.4 **The napier bones**

multiplication output is 2. Now, we have obtained 22 with a carry 1. Similarly, add the next two adjacent numbers and the carry to obtain the digit 6. At this stage, we have obtained 622 with no carry. We can proceed like this to obtain the final answer as 32622. The idea of using bones to carry out the multiplication of numbers was modified by Edmund Gunter in 1620 to produce a device known as slide rule. This device consisted of two sets of graduated scales, which could slide over each other. The slide rule was developed not only for performing multiplication and division of numbers, but also for various scientific functions, such as logarithms, trigonometry, roots, etc. Apart from these manual computing devices, many other devices were also developed for computation purposes. Some of these devices were pascaline, stepped reckoner, punch card system, etc. Pascaline was a calculator developed by Blaise Pascal in 1642. It was also known as a numerical wheel calculator. This device contained a set of toothed wheels that could be operated by hand. Pascaline was designed to handle numbers up to 999,999.999. Pascaline was further improved by German mathematician, Gottfried Wilhem Von Leibriz to produce a device, called stepped reckoner. Stepped reckoner was able to perform the multiplication and division of numbers as well as calculation of the square root of a number. Figure 1.5 shows an illustration of how computing devices have evolved over a period of time:

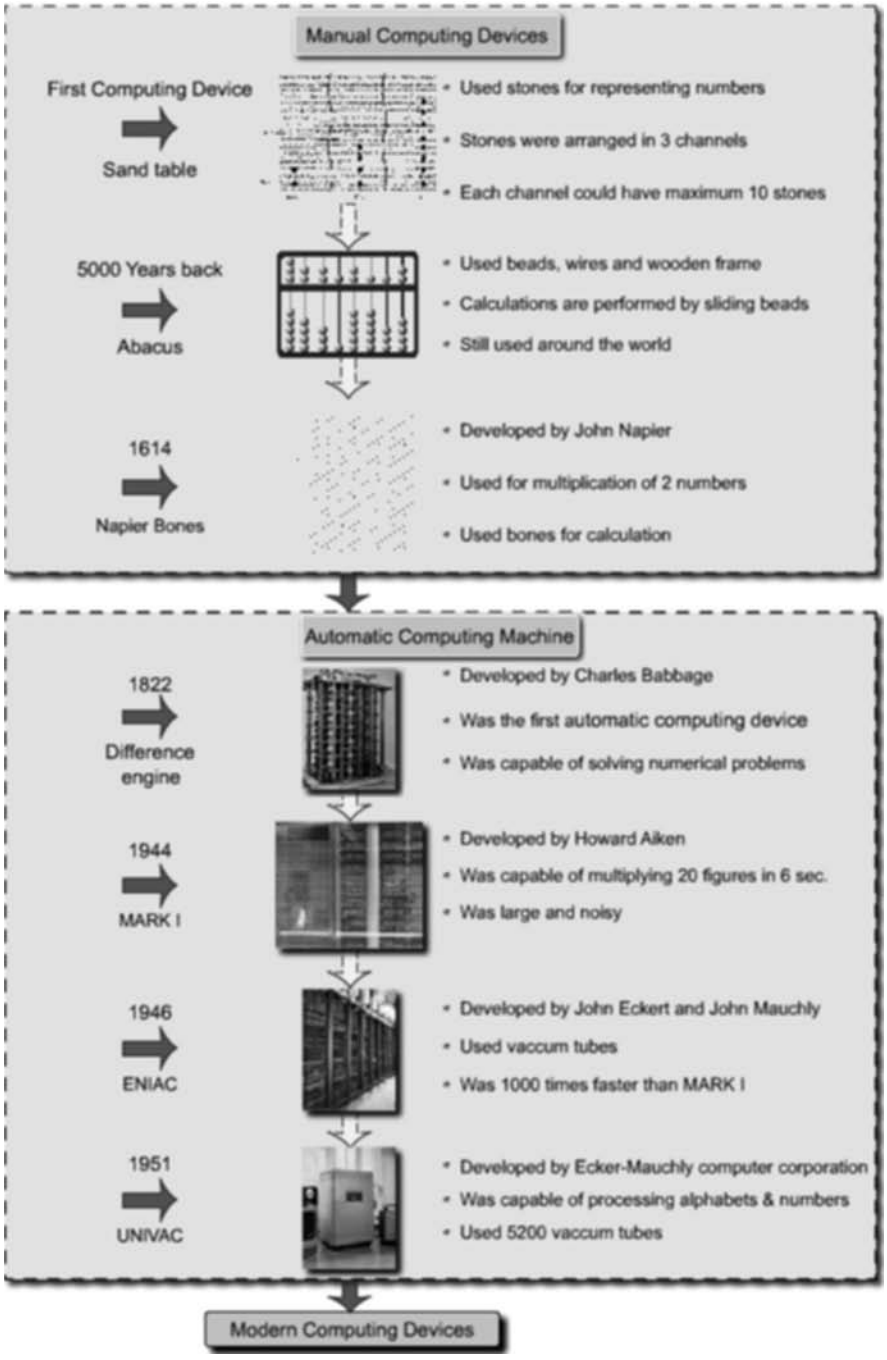


Fig. 1.5 Evolution of computing devices

1.5.2 Automated Computing Devices

Charles Babbage, a professor of mathematics at the Cambridge University, made some worthwhile efforts towards automatic computing. He is also considered to be the father of modern computer. In 1812, Charles Babbage decided to automate the repeated series of steps needed in tabulating various functions, such as polynomial, logarithmic and trigonometric. In 1822, he presented a working model of his concept with the help of an automatic mechanical computing machine. He named the automatic mechanical computing machine as difference engine. In 1823, Babbage made it more automatic by providing the feature of printing the tabulated results. Babbage did not stop here and started working on developing the analytical engine. The analytical engine was considered as the completely automatic, general-purpose programmable digital computer. The analytical engine was the first device that used all the features of a modern digital computer, which include an input unit, an output unit, a storage unit, a processor and a control unit. This engine was designed to perform various mathematical operations by getting two sets of inputs from the user. The first set of input is a program that contains a set of instructions to operate on the data. The other set of input contains the list of variables used in the program or data. The analytical engine built by Babbage in 1833 was digital, programmable and automatic. However, it was a slow engine that took almost 3 minutes to multiply two numbers of twenty figures each.

In 1937, an American mathematician, Howard Aiken designed MARK I and completed it in the year 1944. MARK I was one of the well-known early computers that could perform the multiplication of two numbers of twenty figures in just 6 seconds. Hence, as compared to the analytical engine, MARK I performed calculations at a much faster speed. However, this computer was also not considered very fast from the user's point of view because it printed the results of calculations at the rate of one result per 5 seconds. Also, MARK I computer was noisy and large in size. In the year 1944, a British mathematician, Alan Mathison developed the first pure electronic digital programmable computer. This computer was known as Colossus. Colossus was a special-purpose electronic device that used the vacuum tube technology in performing different operations. It was designed to perform only some specific functions.

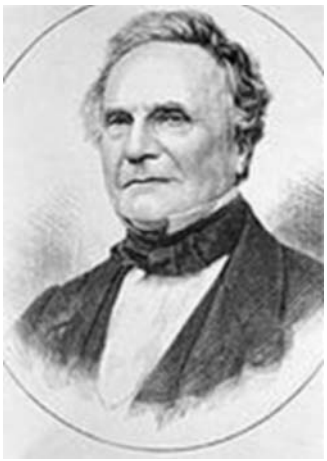


Fig. 1.6 Charles Babbage

Fig. 1.7 Difference Engine

Fig. 1.8 Howard Aiken

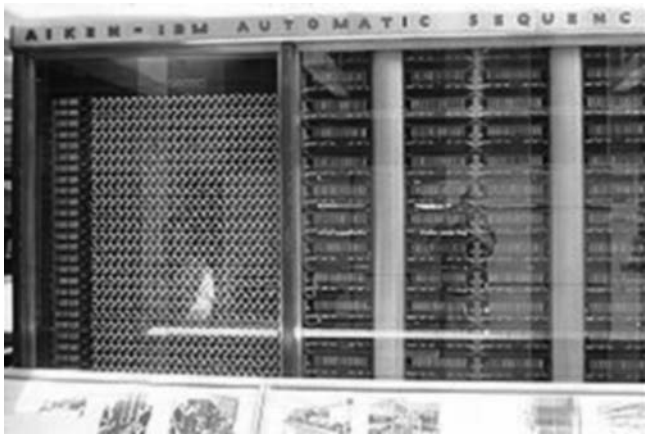


Fig. 1.9 **MARK 1**

The Electronic Numerical Integrator And Calculator (ENIAC) was another general-purpose electronic digital computer developed at the Moore School of Engineering of the University of Pennsylvania by John Eckert, John Mauchly and their team in the year 1946. This computer also used the vacuum tube technology in constructing the basic circuits. It was a general-purpose computer that was capable of solving all types of computing problems. It included all the features and components of a modern digital computer. The internal hardware structure of ENIAC included 17,468 vacuum tubes, 1,500 relays, 70,000 registers, 7,200 crystal diodes and 10,000 capacitors. It was a bulky computer and operated at 1000 times more speed than that of MARK I computer. ENIAC was designed to perform simple arithmetic operations as well as some advanced operations, such as separating the sign of a number and comparing different numbers to check whether they are equal or not. The computer used the decimal number system for representing and processing values.

Fig. 1.10 **John Eckert**

Fig. 1.11 **ENIAC**

In 1949, another electronic computer that used the binary number system for representing and processing values was introduced. This computer was known as Electronic Discrete Variable Automatic Computer (EDVAC). EDVAC was also invented by John Eckertt and John Mauchly and was considered as the successor of ENIAC. EDVAC was the first computer that worked on the principle of stored program. The stored program computer considers the programs and data stored in the memory as a string of binary numbers. Therefore, programs and data stored in the memory are indistinguishable inputs for the computer. The different hardware components of EDVAC were magnetic tape, control unit, dispatcher unit, processor, timer, dual memory and three temporary tanks to hold a single word.

Electronic Delay Storage Automatic Calculator (EDSAC) was another early British electronic computer developed by Maurice Wilkes and his team at the University of Cambridge Mathematical Laboratory in 1949. It also used the vacuum tube technology in constructing the basic logic circuits and mercury delay lines for constructing the memory of a computer. The typical input and output unit of this computer system was punch card and teleprinter respectively. These computer systems were only able to carry out 650 instructions per second. Therefore, these computers were not considered as fast computing devices. During 1950s, Eckert-Mauchly Computer Corporation, a company of John Eckertt and John Mauchly, made some serious efforts in the field of automated computing. In 1951, the company invented the first commercial computer that was known as Universal Automatic Computer (UNIVAC). This computer was a bulky computer that used 5200 vacuum tubes for constructing the basic logic circuits. The mercury data lines were used to construct the memory for storing data and programs. UNIVAC was able to process numbers as well as alphabetic characters in an efficient manner. The important feature of UNIVAC—that made it unique among other well-known early computers—was that it provided separate processes for handling input/output and processing functions.

Fig. 1.12 UNIVAC

1.6 COMPUTER GENERATIONS

Over the years, various computing devices were invented that enabled people to solve different types of problems. All these computing devices can be classified into several generations. These generations refer to the phases of improvement made to different computing devices. The different phases of improvement made to computing devices resulted in a small, cheap, fast, reliable and productive computer. The technological development in the field of computers not only refers to the improvements made to the hardware technologies, but also the improvements made to the software technologies. The history of computer development is often discussed in terms of different generation of computers, as listed below.

- First generation computers
- Second generation computers
- Third generation computers

- Fourth generation computers
- Fifth generation computers

1.6.1 First Generation Computers

The first generation computers were employed during the period 1940–1956. These computers used the vacuum tubes technology for calculation as well as for storage and control purposes. Therefore, these computers were also known as vacuum tubes or thermo ionic valves based machines. Figure 1.13 shows the vacuum tube used in first generation computers.

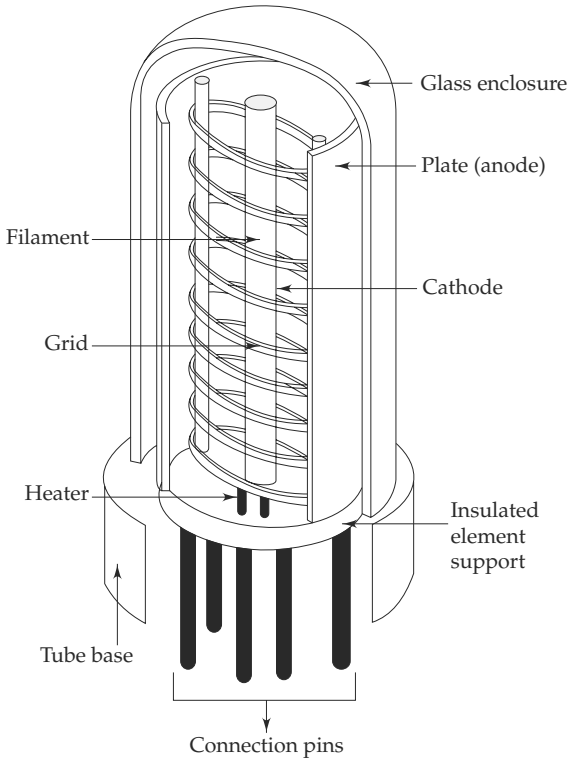


Fig. 1.13 **A vacuum tube**

A vacuum tube is made up of glass and contains filaments inside it. The filaments when heated, generate electrons which eventually help in the amplification and deamplification of electronic signals. The input and output medium for first generation computers was the punched card and printout respectively. Some examples of first generation computers are ENIAC, EDVAC, EDSAC and UNIVAC.

The following were the two major advantages of first generation computer systems:

- These computers were the fastest computing devices of their time.
- These computers were able to execute complex mathematical problems in an efficient manner.

The above two advantages of first generation computers were not sufficient enough to make them popular among the users. The first generation computers had many disadvantages associated with them; some of them are mentioned below:

- The functioning of these computers depended on the machine language. A machine language is a language in which all the values are represented in the form of 0s and 1s. Therefore, these computers were not very easy to program.
- They were generally designed as special-purpose computers. Therefore, they were not very flexible in running different types of applications.
- The use of vacuum tube technology made these computers very large and bulky. Due to their large size, it was not an easy task to install them properly.

- They were not easily transferable from one place to another due to their huge size and also required to be kept in cool places.
- They were single tasking because they could execute only one program at a time and hence, were not very productive.
- They generated huge amount of heat and hence were prone to hardware faults. Hence, they were not considered as reliable and required proper maintenance at regular intervals.

1.6.2 Second Generation Computers

The second generation computers were employed during the period 1956–1963. The main characteristic of these computers was the use of transistors in place of vacuum tubes in building the basic logic circuits. The transistor was invented by Shockley, Brattain and Bardeen, in 1947, for which they won the Nobel Prize. A transistor is a semiconductor device that is used to increase the power of the incoming signals by preserving the shape of the original signal. It has three connections, which are emitter (E), base (B) and collector (C). The base of the transistor is the gate through which the signal, needed to be amplified, is sent. The signal sent through the base of the transistor is, generally, a small flow of electrons. Therefore, the base terminal also acts as the input gate for the transistor. The collector of the transistor is used to collect the amplified signal. The emitter of the transistor acts as the output gate for emitting the amplified signal to the external environment. Figure 1.14 shows the transistor used to manufacture circuitry of second generation computers.

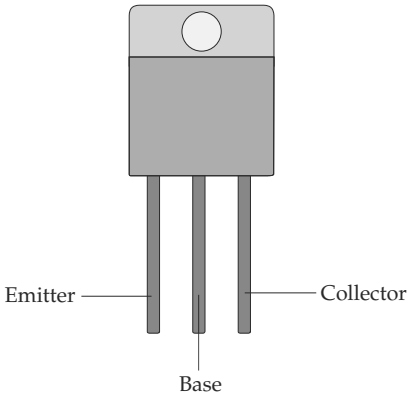


Fig. 1.14 A transistor

The use of transistor technology helped in improving the performance of computers to a large extent. The transistor was a superior technology over vacuum tubes. Transistors used in second generation computers were smaller, faster, cheaper and generated less heat than vacuum tubes used in first generation computers. Transistors were also lightweight electronic devices that required very less power during their operation. These characteristic features of transistors made the second generation computers smaller, faster, cheaper, more efficient, more productive and more reliable, as compared to the first generation computers. Printers, secondary storage and operating system technology were also invented during this era. However, these computers still relied on punched card and printout for carrying out their input/output operations. Another major technological development made to these computers was the replacement of the machine language with the assembly language. Assembly language is a low-level language that allows the programmer to use simple English words—called mnemonics—to represent different instructions in a program. Some examples of second generation computers are PDP-8, IBM 1401 and IBM 7090.

The following were the advantages of second generation computers:

- They were the fastest computing devices of their time.
- They were easy to program because of the use of assembly language.
- They could be transferred from one place to other very easily because they were small and lightweight computing devices.
- They required very less power in carrying out their operations.
- They were more reliable as compared to first generation computers and hence, did not require maintenance at regular intervals of time.

The following were the limitations of second generation computers:

- The input and output media for these computers were not improved to a considerable extent.
- They were required to be placed in air-conditioned places.
- The cost of these computers was very high and they were beyond the reach of home users.
- They were special-purpose computers and could execute only specific applications.

1.6.3 Third Generation Computers

The third generation computers were employed during the period 1964–1975. The major characteristic feature of third generation computer systems was the use of Integrated Circuits (ICs). The IC technology was also known as microelectronics technology. ICs are the circuits that combine various electronic components, such as transistors, resistors, capacitors, etc. onto a single small silicon chip. The first IC was developed by Jack Kilby and Robert Noyce in the year 1958. Figure 1.15 shows a typical IC chip used for manufacturing third generation computers.

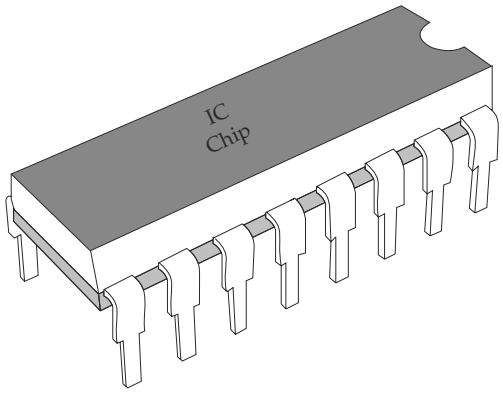


Fig. 1.15 An integrated circuit

ICs were superior to vacuum tubes and transistors in terms of cost and performance. The cost of ICs was very low and the performance was very high because all the electronic components were arranged very close to each other. They also required very low power for performing their operations. Therefore, the use of ICs in third generation computers made them smaller, faster, more efficient and more reliable than the first and second generation of computers. Some examples of third generation computers are NCR 395, B6500, IBM 370, PDP 11 and CDC 7600.

The following were the merits of the third generation computers:

- They were the fastest computing devices as compared with first and second generation of computers. The computational time for these computers was also reduced to great extent. The computational time for these computers was usually measured in nanoseconds.

- They were very productive because of their small computational time.
- They were easily transportable from one place to another because of their small size.
- They used high-level languages. A high-level language is a computer programming language that is independent of the machine details. Hence, the programmer finds it very easy to use them. The programs written in these languages on one computer can be easily executed on some other computer.
- They could be installed very easily and required less space for their installation.
- They were able to execute any type of application, such as business and scientific applications. Hence, the third generation computers were also considered as general-purpose computers.
- They were more reliable and required less frequent maintenance schedules.

Some of the disadvantages of third generation computers were:

- The storage capacity of these computers was still very small.
- The performance of these computers degraded while executing large applications, involving complex computations because of the small storage capacity.
- The cost of these computers was very high.
- They were still required to be placed in air-conditioned places.

1.6.4 Fourth Generation Computers

The fourth generation computers were employed during 1975–1989. The invention of Large Scale Integration (LSI) technology and Very Large Scale Integration (VLSI) technology led to the development of fourth generation computers. However, these computers still used the IC technology to build the basic circuits. The LSI technology allowed thousands of transistors to be fitted onto one small silicon chip. On the other hand, the VLSI technology allowed hundreds of thousands of transistors to be fitted onto a single chip. As a result, the manufacturers were able to reduce the size of the computers and make them cheaper as compared to the other generation of computers.

The progress in LSI and VLSI technologies led to the development of the microprocessor, which became the major characteristic feature of the fourth generation computers. A microprocessor incorporates various components of a computer—such as CPU, memory and Input/Output (I/O) controls—onto a single chip. The computers in this generation were designed to have a microprocessor, some additional storage chips and support circuitry. Some popular later microprocessors include Intel 386, Intel 486 and Pentium. Figure 1.16 shows the Intel P4004 microprocessor chip developed in 1971.

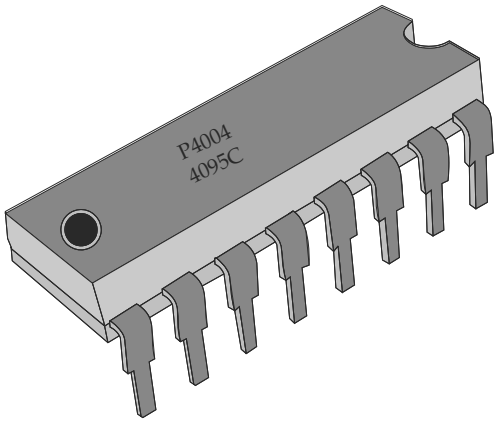


Fig. 1.16 **The Intel P4004 microprocessor chip**

The term Personal Computer (PC) became known to the people during this era. The term PC refers to a computer that is designed to be used by an individual. Since the size and cost of the computer was decreased to a considerable extent in this period, people started using these computers for their personal work too. The storage technologies used in the fourth generation computers were also improved and they started using static and dynamic Random Access Memory (RAM). The advantage of using this type of memory was that it allowed the computers to access the stored information at a rapid pace and hence helped in increasing the productivity and performance of the computers. Some of the examples of fourth generation computers are IBM PC, IBM PC/AT, Apple and CRAY-1.

The use of LSI and VLSI technologies made the fourth generation computers small, cheap, compact and powerful. Apart from these technologies, the fourth generation computers also included the following developments:

- Development of Graphical User Interfaces (GUIs)
- Development of new operating systems
- Invention of various secondary storage and I/O devices
- Development of Local Area Network (LAN)

Some of the advantages of fourth generation computers were:

- The use of LSI, VLSI and semiconductor technologies made these computers very powerful in terms of their processing speed and access time.
- The storage capacity of these computers was very large and faster, and hence, they were very productive and highly optimised.
- They were highly reliable and required very less maintenance.
- They provided a user-friendly environment while working because of the development of GUIs and interactive I/O devices.
- The programs written on these computers were highly portable because of the use of high-level languages.
- They were very versatile and suitable for every type of application.
- They required very less power to operate.

Some of the problems associated with fourth generation computers were:

- The soldering of LSI and VLSI chips on the wiring board was not an easy task and required complicated technologies to bind these chips on the wiring board.
- The working of these computers is still dependent on the instructions given by the programmer.

1.6.5 Fifth Generation Computers

The different types of modern digital computers come under the category of fifth generation computers. The fifth generation computers are based on the Ultra Large Scale Integration (ULSI) technology that allows almost ten million electronic components to be fabricated on one small chip. The ULSI technology helps in increasing the power and speed of the microprocessor

chips and the capacity of primary and secondary storage devices to a great extent. As a result, the fifth generation computers are faster, cheaper and more efficient, as compared to the fourth generation computers. Some of the improvements or developments made during this generation of computers are:

- Development of various portable computers such as laptop, pocket computer, Personal Digital Assistant (PDA), etc.
- Development of Parallel Processors.
- Development of centralised computers called servers.
- Invention of optical disk technology.
- Invention of the Internet and its different services.

Some of the advantages of fifth generation computers are:

- They are the fastest and powerful computers till date.
- They are able to execute a large number of applications at the same time and that too at a very high speed.
- The use of ULSI technology helps in decreasing the size of these computers to a large extent. Some of the fifth generation computers are so small in size that they can be used while traveling.
- The users of these computers find it very comfortable to use them because of the several additional multimedia features.
- They are versatile for communications and resource sharing.

The fifth generation computers are highly appreciated by their users because of their several advantages. However, the major disadvantage of the fifth generation computers is that they are not provided with an intelligent program that could guide them in performing different operations. Nowadays, scientists are making serious efforts in this field, and artificial intelligence and expert system applications are the results of these efforts. Figure 1.17 shows a snapshot of the evolution of computers over different generations.

1.7 CLASSIFICATION OF COMPUTERS

There are different types of computers available these days. The function of each type of computer is to process data and provide some output to the users. However, the methods or techniques used by these computers to process and handle the data may be different. We can classify computers according to the following three criteria:

- Based on operating principles
- Based on applications
- Based on size and capability

1.7.1 Based on Operating Principles

On the basis of operations performed and methods used to store and process data and information, computers can be classified into the following categories:

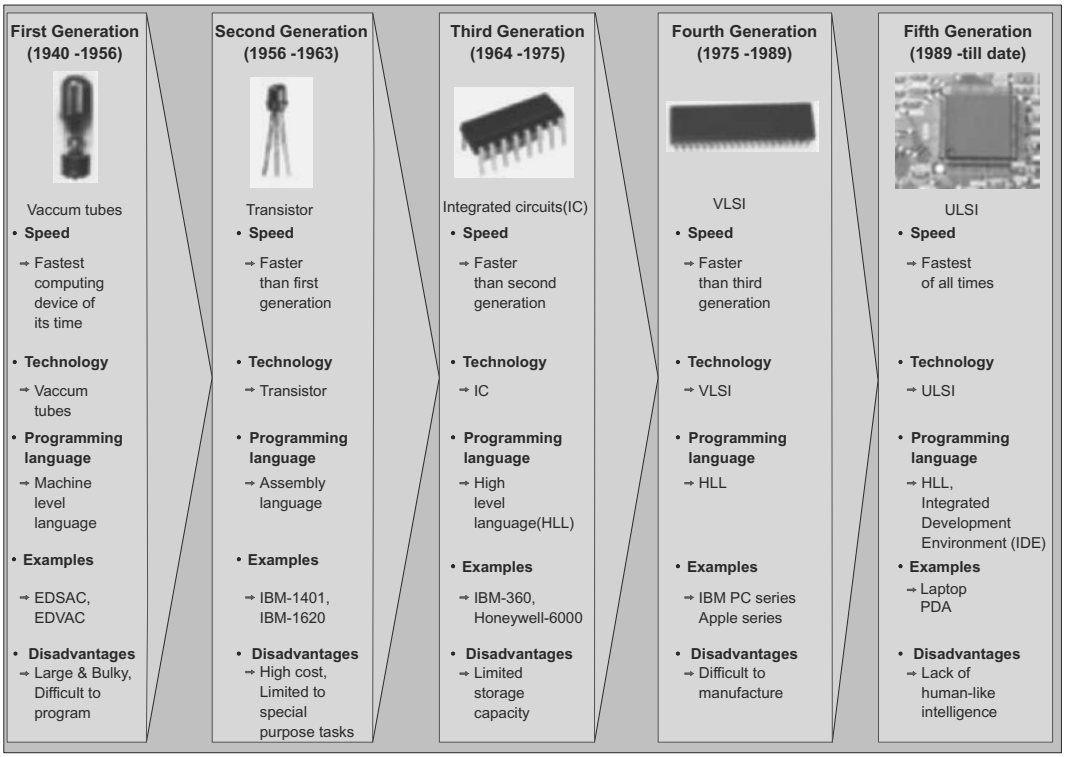


Fig. 1.17 Genertaion of computers—a snapshot

- Analog computers
- Digital computers
- Hybrid computers

Analog computers The analog computers represent data in the form of continuous electrical signals having a specific magnitude. These computers are very fast in their operation and allow several other operations to be carried out at the same time. However, the results produced by these computers are not very accurate. Therefore, the analog computers are widely used in applications in which the accuracy of results is not a major concern. They are powerful tools to solve differential equations.

The electronic circuit employed in modern analog computers is generally an Operational

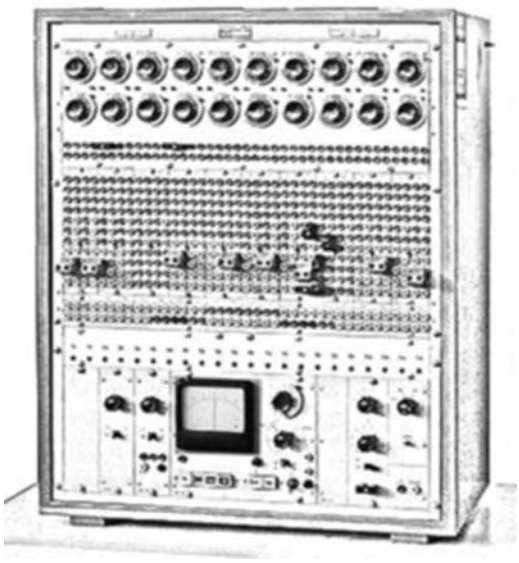


Fig. 1.18 Analog computer

Amplifier (Op-Amp). It is made up of semiconductor integrated circuits. The three different characteristic features of Op-Amps are:

- They have large voltage gain. The voltage gain of an amplifier is defined as the ratio of the output voltage to the input voltage.
- They have infinite input resistance. The input resistance is defined as the ratio of change in the input voltage to the change in input current.
- They have zero output resistance. The output resistance is the nominal resistance measured with no load.

Figure 1.19 shows the basic circuit used in analog computers.

In Fig. 1.19, the triangle represents an amplifier that is used to invert the incoming signal. If the incoming signal is a positive signal, then it will be inverted into a negative output signal. Similarly, if the incoming signal is a negative signal, then it will be inverted into a positive output signal. R_f and R_{in} are used to represent the feedback resistor and the input resistor respectively.

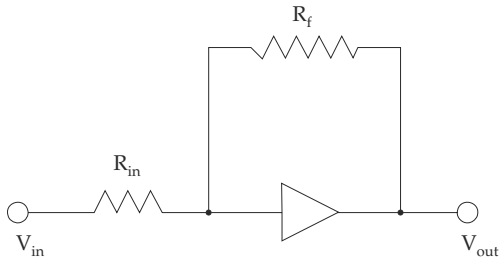


Fig. 1.19 **Integrated circuit of an operational amplifier**

Digital computers The digital computer, also known as the digital information processing system, is a type of computer that stores and processes data in the digital form. Therefore, each type of data is usually stored in these computers in terms of 0s and 1s. The output produced by these computers is also in the digital form. The digital computers are also capable of processing the analog data. However, the analog data should be first converted to the digital form, before being processed by these computers. Similarly, if we want the output in the analog form, then the digital information produced by these computers should be first converted to an analog form. These conversions are generally carried out by the in-built components of digital computers.

Fig. 1.20 **Digital computer**

Digital computers are generally faster and more reliable than the analog computer systems and provide more accurate results. The computer used by a home user is a typical example of a digital computer. The digital computers are also employed in colleges, universities and small- and medium-sized businesses. The different hardware components of a digital computer are an Arithmetic Logic Unit (ALU), a Control Unit (CU), a memory unit and I/O units. The ALU of a digital computer is used to perform various arithmetic operations, such as addition, subtraction, multiplication and division and various logic operations such as AND, OR, NOT, etc. CU helps in directing the operations of ALU. The memory unit is used to store the data on temporary or permanent basis. The input units are used to enter the data into the computer and the output units are used to display the information generated by the computer to the user.

Hybrid computers The hybrid computer is a combination of analog computer and digital computer because it encompasses the best features of both these computers. Therefore, the hardware components of hybrid computers are usually the mixture of analog and digital components. These features make the hybrid computers very fast, efficient and reliable. In these computers, data is generally measured and processed in the form of electrical signals and is stored with the help of digital components. However, these computers can also be used to perform various types of logical operations.

Fig. 1.21 Hybrid computer

The input accepted by the hybrid computers is a continuously varying input signal. This input signal is then converted by them into a set of discrete values for performing different operations. These computers prove to be very cost-effective in performing complex simulations. The hybrid computers are also less expensive than the digital computers.

The computer used in hospitals to measure the heartbeat of a patient is a very good example of a hybrid computer. Apart from this, the hybrid computers are also used in scientific applications, various engineering fields and in controlling business processes.

1.7.2 Based on Applications

Different computers are designed for different purposes so that they can perform their tasks according to their capabilities. On the basis of different applications or purposes, computers can be classified into the following categories:

General-purpose computers They are designed in such a manner that they can work in all environments. The general-purpose computers are versatile and can store a number of programs meant for performing distinct tasks. However, the general-purpose computers are not efficient and consume a large amount of time in generating the result.

Special-purpose computers They are designed in such a manner that they can perform only a specified task. The special-purpose computers are not versatile and their speed and memory size depend on the task that is to be performed. These computers are less expensive as they do not contain any redundant information. The special-purpose computers are efficient and consume less amount of time in generating the result.

1.7.3 Based on Size and Capability

Computers differ from each other in terms of their shape, size and weights. Each type of computer performs some unique functions and can be employed in the fields suited for them. These computers also differ in terms of processing speed. Some of them are of moderate

speed, whereas some others operate at a very fast speed. On the basis of size and capability, computers can be classified into the following categories:

- Microcomputers
- Mini computers
- Mainframe computers
- Super computers

Microcomputers A microcomputer is a small and cheap digital computer that is designed to be used by individuals. It is built around a microprocessor, a storage unit and an I/O channel. Apart from these components, the other parts that a microcomputer includes are power supply, connecting cables, keyboard, mouse, printer and scanner. These computers also include several software programs such as operating system, system software and utility software. The micro computers are generally available in the form of PCs, workstations and notebook computers. Figure 1.23 shows the block diagram of a microcomputer.

Fig. 1.22 **Microcomputer**

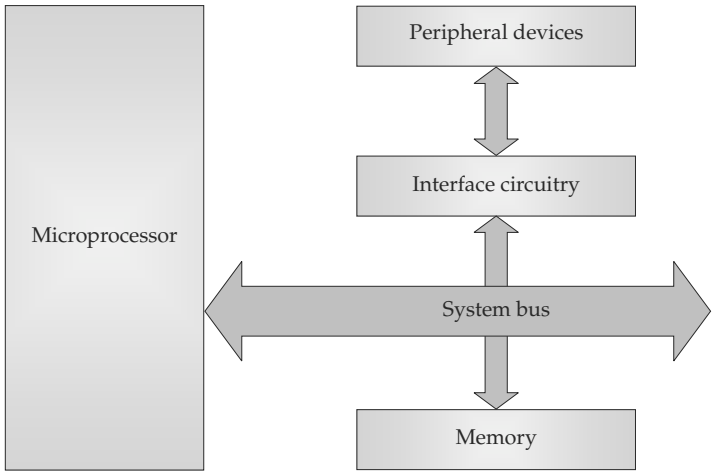


Fig. 1.23 **The block diagram of a micro computer**

- **Microprocessor** It is the heart of the microcomputer. It incorporates all the functions of a CPU onto a single IC in a microcomputer. The basic units of microprocessor are ALU, register unit and CU. ALU is used to perform various arithmetic and logic operations. The register unit is used to store the data and instructions temporarily needed by the ALU. The various registers used by a microcomputer are Accumulator (AC), program control register, I/O register, instruction register, Memory Address Register (MAR) and Memory Buffer Register (MBR). CU is used to manage and direct the operations performed by the microcomputer.

- **Memory** It is used to store the data and instructions on temporary or permanent basis. A microcomputer generally employs two types of memories, i.e., primary memory and secondary memory. Primary memory, also called main memory, is used to store the data and instructions temporarily. It stores only those instructions and data that are needed by the microprocessor of the computer for processing. The secondary memory, also called auxiliary memory, is used to store data and instructions permanently. Magnetic disks and magnetic tapes are some of the examples of secondary storage.
- **Peripheral devices** They are generally the input and output devices attached to the computer. The various input devices—such as keyboard and mouse—are used to enter program and data into the computer before performing any kind of operation. They are used to transfer data and instructions from the external environment into the computer. The various output devices—such as monitor and printer—are used to display the results computed by the computer to the user. The major function performed by the output devices is to convert the binary result computed by the computer into a form that can be easily understood by the users.
- **System bus** It is also referred to as the frontside bus, memory bus, local bus or host bus. The system bus in the micro computer is used to connect microprocessor, memory and peripheral devices into a single unit. The system bus is a collective name given to address, data and control bus. The address bus is a unidirectional bus that is used to identify a peripheral device or a memory location. The data bus is a bidirectional bus that is used to transfer data among microprocessor, memory and peripheral devices of the computer. The control bus is used by the microprocessor to send control signals to the various devices within the computer.

Depending on the size, the microcomputer can be further classified into the following types:

- **Desktop computer** It is also known as PC. The desktop computer systems are designed to be used by an individual at a single location. The typical components of a desktop computer are keyboard, mouse, monitor, hard disk storage, peripheral devices and a system unit. These computers are very cheap and an individual can easily purchase them for home or business use. The different manufacturers of desktop computers are Apple, IBM, Dell and Hewlett-Packard (HP).
- **Laptop computer** It is a portable computer that can be taken from one place to another at any time very easily. It is also known as notebook computer, notepad or mobile computer. The laptop computer is a small-size computer that incorporates all the features of a typical desktop computer. These computers are provided with a rechargeable battery that removes the need of continuous external power supply. However, these computer systems are more expensive than desktop computers. The different manufacturers of laptop computers are Acer, Apple, Panasonic, Sony and HP.
- **Hand-held computer** It is also known as PDA (Personal Digital Assistant), converged device, palmtop or mobile device. The hand-held computer is a very small-size computer that can be kept in the pocket. It generally has a very small display screen

and the input device for these computers is a pen or an electronic stylus. The storage capacity of hand-held computers is not very large. They generally use small cards to store data and programs instead of disk drives. Therefore, they are less powerful as compared to the desktop and laptop computers. The different examples of hand-held computers are Apple Newton, Casio Cassiopeia, Franklin eBookMan, etc.

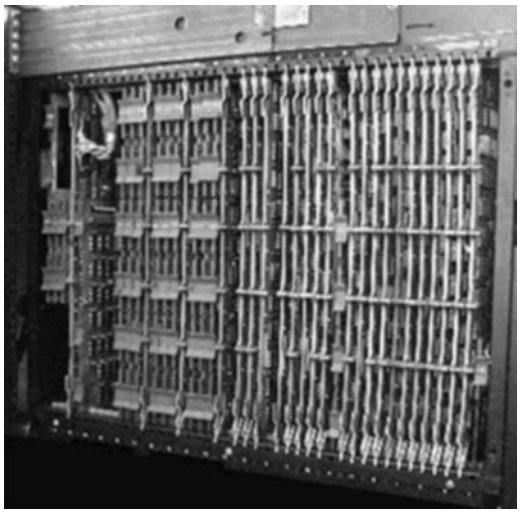


Fig. 1.24 Mini computer

Mini computers A mini computer was first introduced in the year 1960 by Digital Equipment Corporation (DEC). They were called mini computers because of their smaller size than the other computers of those times. They can handle more data and more input and output than micro computers. Mini computers are less powerful than mainframe computers but more powerful than micro computers. Therefore, they are also referred to as the midrange computers. They are able to cater to the needs of multiple users at a single instant of time. The number of users supported by mini computers may range between 4 and 200. These computers are generally designed for small and medium-sized business environments.

Mini computers are generally used in business environments as the centralised computer or the network server. After implementing the mini computer as the network server, hundreds of desktop computers can be connected to it. Mini computers can also be used as the web servers that can handle thousands of transactions in a day. These computers are less expensive than mainframe computers and hence suitable for those organisations that cannot afford high-priced servers. The different examples of mini computers are PDP 11, IBM (8000 series), VAX 7500, etc.

Fig. 1.25 Mainframe computer

Mainframe computers A mainframe computer is a very large computer that is employed by large business organisations for handling major applications, such as financial transaction processing, Enterprise Resource Planning (ERP), industry and consumer statistics, and census. They are capable of handling almost millions of records in a day. The mainframe computers can also be used as the centralised computers with several user terminals connected to it. The mainframe computers are actually considered as the predecessor of servers. These computers are bigger and more expensive than other computers. The implementation of mainframe computers also requires large space with a closely monitored humidity and temperature

levels. These computers are termed as mainframe because all the hardware units are arranged into a frame. The different manufacturers of mainframe computers are IBM, Amdahl, Hitachi, etc. Examples of mainframe computers are IBM 3000, VAX 8000 and CDC 6600.

The mainframe computers can maintain large databases that can be accessed by remote users with a simple terminal. Therefore, they are also known as super servers or database servers. The processing speed of these computers is generally optimised by employing more than one microprocessor to execute millions of instructions per second. The mainframe computers also have large capacity of primary and secondary storage as compared with other types of computers.

Some of the characteristic features of mainframe computers are:

- A typical mainframe computer generally has a maximum of 16 microprocessors. However, some modern mainframe computers can have more than 16 microprocessors.
- The RAM capacity of these computers lies between 128 MB and 8 GB.
- They are able to run multiple operating systems, and therefore, termed 'virtual machines'.
- They have different cabinets for primary storage, secondary storage and I/O units.
- They can handle huge amount of I/O operations at the same time.

Fig. 1.26 Supercomputer

Super computers A super computer is the fastest type of computer that can perform complex operations at a very high speed. The super computers were first presented in the year 1960 by Seymour Cray at Control Data Corporation (CDC). They are more expensive than the other categories of computers and are specially designed for the applications in which large number of complex calculations have to be carried out to get the desired output. The main reason behind the fast speed of super computers is that they are designed only to execute small number of programs at a time rather than many programs simultaneously. Some of the manufacturers of super computers are IBM, Silicon Graphics, Fujitsu, Intel, etc. Examples of Super Computers are CRAY 3, Cyber 205, NEC SX-3 and PARAM from India.

The various application areas of super computers are:

- Weather forecasting
- Animated graphics
- Fluid mechanics
- Nuclear energy research
- Petroleum exploration

Super computers are manufactured with no special hardware. Like the typical computer, they have CPU and memory as their major components. However, the CPU of super computer operates at faster speed, as compared to the other categories of computers. Super computers

are the fastest computers because they employ thousands of processors, hundreds of gigabytes of RAM and thousands of gigabytes of secondary storage.

The designers of super computers use two different methods for optimising their performance. These methods are pipelining and parallelism. Pipelining is a technique that allows the microprocessors to execute the second instruction before the execution of the first instruction is completed, whereas parallelism allows the microprocessors to execute several instructions at the same time. In this type of computing, a large and complex problem is first divided into smaller problems, that are solved concurrently by the microprocessor of the computer.

1.8 BASIC COMPUTER ORGANISATION

The basic computer organisation involves the interfacing of different units of the computer and various operations performed between these units. The basic computer organisation explains the way in which different units of computer are interconnected with each other and controlled. Some of the basic units of computer organisation are:

- Input Unit
- Memory Unit
- CPU
- Output Unit

Figure 1.27 shows the basic computer organisation.

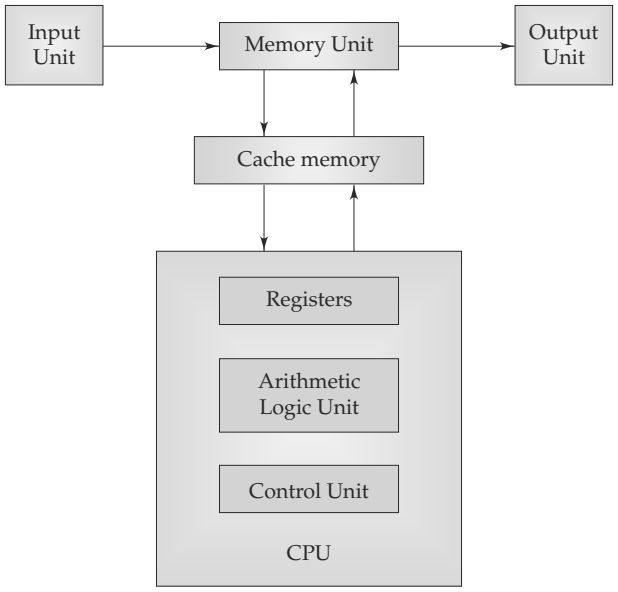


Fig. 1.27 The block diagram of a computer system

1.8.1 Input Unit

An input unit is an electronic device, which is used to feed input data and control signals to a computer. It is also known as input device. Input devices are connected to the computer system using cables. The most commonly used input devices among others are:

- Keyboard
- Mouse
- Scanner

Keyboard A standard keyboard includes alphanumeric keys, function keys, modifier keys, cursor movement keys, spacebar, escape key, numeric keypad, and some special keys, such as Page Up, Page Down, Home, Insert, Delete and End. The alphanumeric keys include the number keys and the alphabet keys. The function keys are the keys that help perform a specific task such as searching a file or refreshing a Web page. The modifier keys such as Shift and Control keys modify the casing style of a character or symbol. The cursor movement keys include up, down, left and right keys, and are used to modify the direction of the cursor on the screen. The spacebar key shifts the cursor to the right by one position. The numeric keypad uses separate keypads for numbers and mathematical operators. A keyboard is shown in Fig. 1.28.

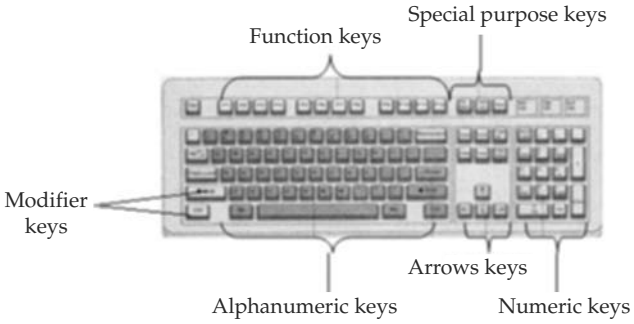


Fig. 1.28 Keyboard

Mouse The mouse allows the user to select elements on the screen, such as tools, icons, and buttons, by pointing and clicking them. We can also use a mouse to draw and paint on the screen of the computer system. The mouse is also known as a pointing device because it helps change the position of the pointer or cursor on the screen.

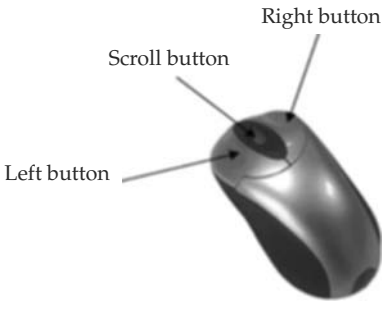


Fig. 1.29 Mouse

The mouse consists of two buttons, a wheel at the top and a ball at the bottom of the mouse. When the ball moves, the cursor on the screen moves in the direction in which the ball rotates. The left button of the mouse is used to select an element and the right button, when clicked, displays the special options such as open and explore and shortcut menus. The wheel is used to scroll down in a document or a Web page. A mouse is shown in Fig. 1.29.

Scanner A scanner is an input device that converts documents and images as the digitised images understandable by the computer system. The digitised images can be produced as black and white images, gray images, or coloured images. In case of coloured images, an image is considered as a collection of dots with each dot representing a combination of red, green, and blue colours, varying in proportions. The proportions of red, green, and blue colours assigned to a dot are together called as colour description. The scanner uses the colour description of the dots to produce a digitised image. Figure 1.30 shows a scanner.

Fig. 1.30 Scanner

1.8.2 Memory Unit

The memory unit of a computer is used to store data, instructions for processing data, intermediate results of processing and the final processed information. The memory units of a computer are classified as primary memory and secondary memory. Figure 1.31 shows the memory categorization in a computer system.

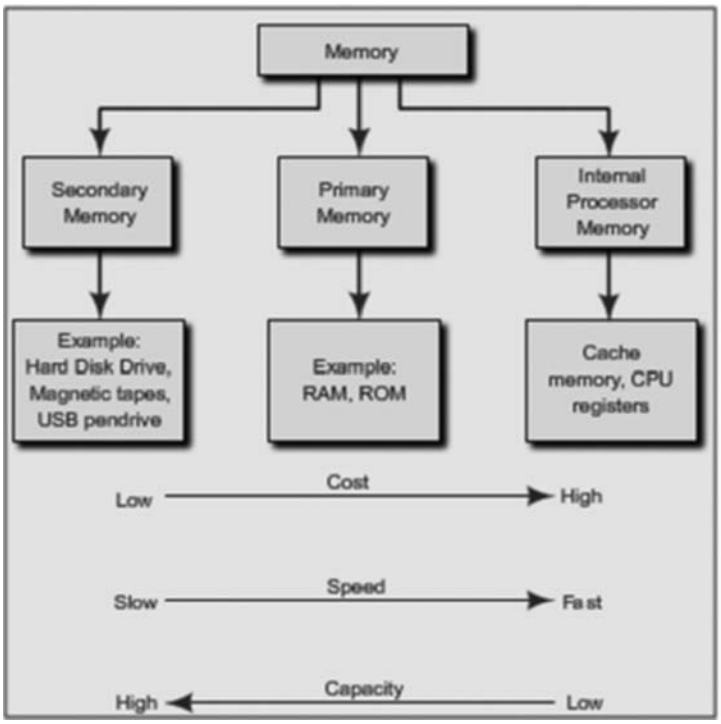


Fig. 1.31 Categorization of Memory Devices

Primary Memory The primary memory is available in the computer as a built-in unit of the computer. The primary memory is represented as a set of locations with each location occupying 8 bits. Each bit in the memory is identified by a unique address. The data is stored in the machine-understandable binary form in these memory locations. The commonly used primary memories are:

- **ROM** ROM represents Read Only Memory that stores data and instructions, even when the computer is turned off. It is the permanent memory of the computer where the contents cannot be modified by an end user. ROM is a chip that is inserted into the motherboard. It is generally used to store the Basic Input/Output system (BIOS), which performs the Power On Self Test (POST).
- **RAM** RAM is the read/write memory unit in which the information is retained only as long as there is a regular power supply. When the power supply is interrupted or switched off, the information stored in the RAM is lost. RAM is a volatile memory that temporarily stores data and applications as long as they are in use. When the use of data or the application is over, the content in RAM is erased.

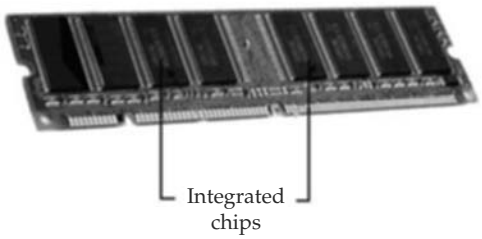


Fig. 1.32 RAM

- **Cache memory** Cache memory is used to store the data and the related application that was last processed by the CPU. When the processor performs processing, it first searches the cache memory and then the RAM, for an instruction. The cache memory is always placed between CPU and the main memory of the computer system.

Table 1.1 depicts some of the key differences between RAM and ROM:

TABLE 1.1

RAM	ROM
It is a read/write memory	It is a read only memory
It is volatile storage device	It is a permanent storage device
Data is erased as soon as power supply is turned off	Data remains stored even after power supply has been turned off
It is used as the main memory of a computer system	It is used to store Basic input output system (BIOS).

Secondary Memory Secondary memory represents the external storage devices that are connected to the computer. They provide a non-volatile memory source used to store information that is not in use currently. A storage device is either located in the CPU casing of the computer or is connected externally to the computer. The secondary storage devices can be classified as:

- **Magnetic storage device** The magnetic storage devices store information that can be read, erased and rewritten a number of times. These include floppy disk, hard disk and magnetic tapes.

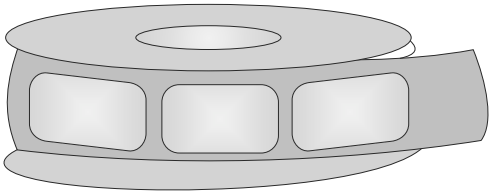


Fig. 1.33 **Magnetic tape**

- **Optical storage device** The optical storage devices are secondary storage devices that use laser beams to read the stored data. These include CD-ROM, rewritable compact disk (CD-RW), and digital video disks with read only memory (DVD-ROM).

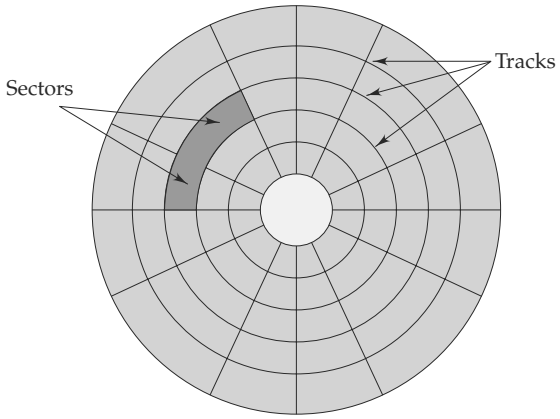


Fig. 1.34 **Magnetic disk**

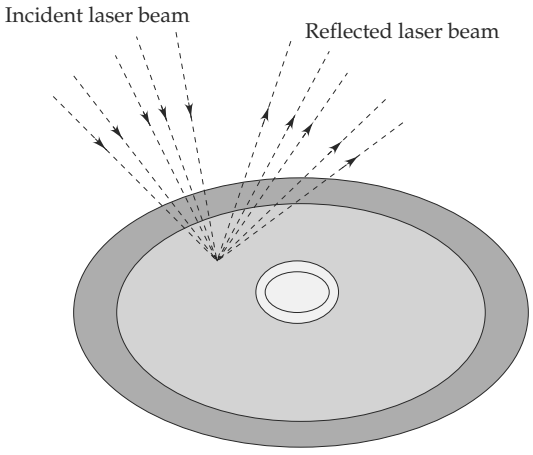


Fig. 1.35 **Optical Disk**

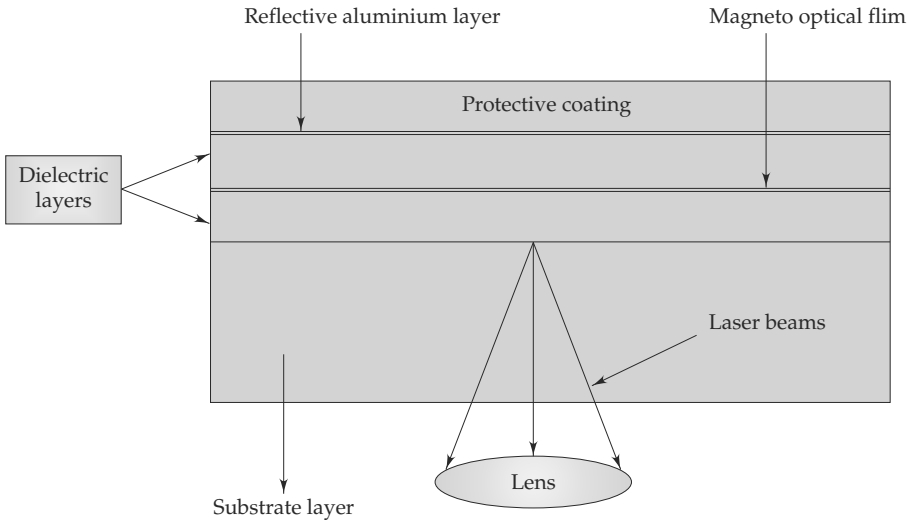


Fig. 1.36 **Magneto-optical disk**

- **Magneto-optical storage device** The magneto-optical devices are generally used to store information, such as large programs, files and backup data. The end user can modify the information stored in magneto-optical storage devices multiple times. These devices provide higher storage capacity as they use laser beams and magnets for reading and writing data to the device. Examples of magneto-optical devices include Sony MiniDisc, Maxoptix T5-2600, etc.
- **Universal serial bus (USB) drive** USB drive or commonly known as pen drive is a removable storage device that is interfaced on the USB port of a computer system. It is pretty fast and compact in comparison to other storage devices like CD and floppy disk. One of the most important advantages of a USB drive is that it is larger in capacity as compared to other removable storage devices. Off late, it has become very popular amongst computer users.

Fig. 1.37 **USB drive**

1.8.3 CPU

The function of any computer system revolves around a central component known as CPU. The CPU, which is popularly referred as the "brain" of the computer, is responsible for processing the data inside the computer system. It is also responsible for controlling all other components of the system. The main operations of the CPU include four phases:

- Fetching instructions from the memory
- Decoding the instructions to decide what operations are to be performed

- Executing the instructions
- Storing the results back in the memory

The three main components of CPU are:

- Arithmeitc and Logic Unit (ALU)
- Control Unit (CU)
- Registers

ALU ALU is a part of the CPU that performs arithmetic and logical operations on the data. The arithmetic operations can be addition, subtraction, multiplication or division. The multiplication and division operations are usually implemented by the ALU as the repetitive process of addition and subtraction operations respectively. It takes input in the form of an instruction that contains an opcode, operands and the format code. The opcode specifies the operation to be performed and the operands specify the data on which operation is to be performed. The format code suggests the format of the operands, such as fixed-point or floating-point. The output of ALU contains the result of the operation and the status of the result, whether it is final or not. The output is stored in a storage register by the ALU. Register is a small storage area inside the CPU from where data is retrieved faster than any other storage area. It also performs 16 different types of logical operations. The various logical operations include greater than (>), less than (<), equal to (=), not equal to (≠), shift left, shift right, etc. It makes use of various logic gates, such as AND, OR, NOR, etc. for performing the logical operations on the data.

CU CU is an important component of CPU that controls the flow of data and information. It maintains the sequence of operations being performed by the CPU. It fetches an instruction from the storage area, decodes the instruction and transmits the corresponding signals to the ALU and the storage registers. CU guides the ALU about the operations that are to be performed and also suggests the I/O devices to which the data is to be communicated. CU uses a program counter register for retrieving the next instruction that is to be executed. It also uses a status register for handling conditions such as overflow of data.

Registers Central processing unit contains a few, special-purpose, temporary storage units known as registers. They are high-speed memory locations used for holding instructions, data and intermediate results that are currently being processed. A processor can have different types of registers to hold different types of information. They include, among others:

- Program Counter (PC) to keep track of the next instruction to be executed.
- Instruction Register (IR) to hold instructions to be decoded by the control unit.
- Memory Address Register (MAR) to hold the address of the next location in the memory to be accessed.
- Memory Buffer Register (MBR) for storing data received from or sent to CPU.
- Memory Data Register (MDR) for storing operands and data.
- Accumulator (ACC) for storing the results produced by arithmetic and logic units.

Many computers employ additional registers for implementing various other requirements. The number and sizes of registers, therefore, vary from processor to processor. An effective implementation of registers can increase considerably the speed of the processor.

Figure 1.38 shows a typical block diagram of computer system, illustrating the arrangement of CPU with the input and output units as well as the memory of the computer system.

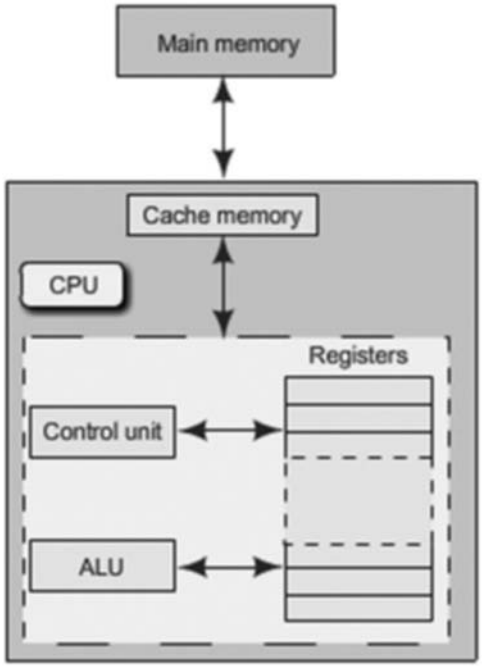


Fig. 1.38 Illustration of CPU and memory

1.8.4 Output Unit

Output unit is an electronic device, which is used to communicate the output obtained after processing a specific task, to the user. The data processed by the CPU, is made available to the end user by the output devices. The most commonly used output devices are:

- Monitor
- Printer
- Speaker

Monitor A monitor is the most commonly used output device that produces visual displays generated by the computer. The monitor, also known as a screen, is connected as an external device using cables or connected either as a part of the CPU case. The monitor connected using cables, is connected to the video card placed on the expansion slot of the motherboard. The display device is used for visual presentation of textual and graphical information. The

monitors can be classified as Cathode Ray Tube (CRT) monitors or Liquid Crystal Display (LCD) monitors. The CRT monitors are large, occupy more space in the computer, whereas LCD monitors are thin, light weighted, and occupy lesser space. Both the monitors are available as monochrome, gray scale and colour models. However, the quality of the visual display produced by the CRT is better than that produced by the LCD.

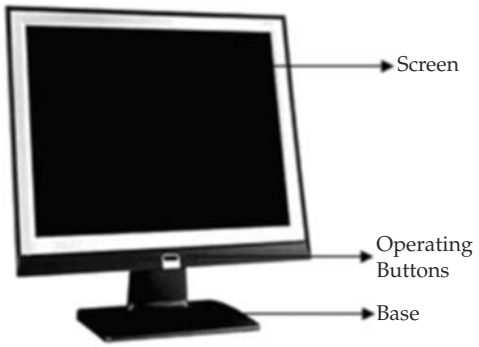


Fig. 1.39 A Monitor

The inner side of the screen of the CRT contains the red, green, and blue phosphors. When a beam of electrons strike the screen, the beam strikes the red, green and blue phosphors on the screen and irradiates it to produce the image. The process repeats itself for a change in the image, thus refreshing the changing image. To change the colour displayed by the monitor, the intensity of the beam striking the screen is varied. If the rate at which the screen gets refreshed is large, then the screen starts flickering, when the images are refreshed.

The LCD monitor is a thin display device that consists of a number of colour or monochrome pixels arrayed in front of a light source or reflector. LCD monitors consume a very small amount of electric power.

A monitor can be characterised by its monitor size and resolution. The monitor size is the length of the screen that is measured diagonally. The resolution of the screen is expressed as the number of picture elements or pixels of the screen. The resolution of the monitor is also called the dot pitch. The monitor with a higher resolution produces a clearer image.

Printer The printer is an output device that transfers the text displayed on the screen, onto paper sheets that can be used by the end user. The various types of printers used in the market are generally categorised as dot matrix printers, inkjet printers, and laser printers. Dot matrix printers are commonly used in low quality and high volume applications like invoice printing, cash registers, etc. However, inkjet printers are slower than dot matrix printers and generate high-quality photographic prints. Since laser printers consist of microprocessor, ROM and RAM, they can produce high-quality prints in quicker time without being connected to a computer.

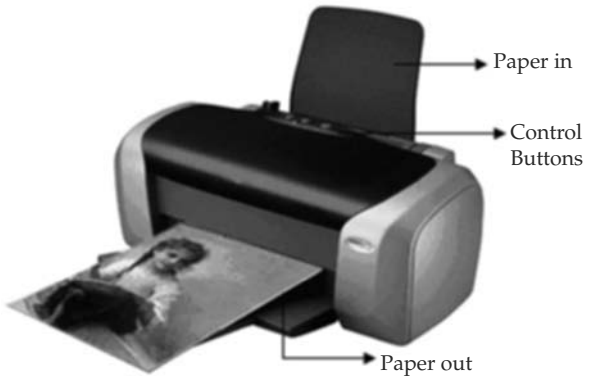


Fig. 1.40 A Printer

The printer is an output device that is used to produce a hard copy of the electronic text displayed on the screen, in the form of paper sheets that can be used by the end user. It is an

external device that is connected to the computer system using cables. The computer needs to convert the document that is to be printed to data that is understandable by the printer. The printer driver software or the print driver software is used to convert a document to a form understandable by the printer. When the computer components are upgraded, the upgraded printer driver software needs to be installed on the computer.

The performance of a printer is measured in terms of dots per inch (DPI) and pages per minute (PPM) produced by the printer. The greater the DPI parameter of a printer, the better is the quality of the output generated by it. The higher PPM represents higher efficiency of the printer.

Speaker The speaker is an electromechanical transducer that converts an electrical signal into sound. They are attached to a computer as output devices, to provide audio output, such as warning sounds and Internet audios. We can have built-in speakers or attached speakers in a computer to warn end users with error audio messages and alerts. The audio drivers need to be installed in the computer to produce the audio output. The sound card being used in the computer system decides the quality of audio that we listen using music CDs or over the Internet. The computer speakers vary widely in terms of quality and price. The sophisticated computer speakers may have a subwoofer unit, to enhance bass output.

Fig. 1.41 Speakers

1.9 NUMBER SYSTEM AND COMPUTER CODES

A computer is a digital system that stores and processes different types of data in the form of 0s and 1s. The different types of data handled by a computer system include numbers, alphabets and some special characters. Therefore, there is a need to change the data entered by the users into a form that the computer system can understand and process. Different types of codes have been developed and used to represent the data entered by the users in the binary format. The binary system represents each type of data in terms of binary digits—0s and 1s. Since these codes convert the data into the binary from, the computer codes are also referred as binary codes.

The decimal system is not the only number system used by computer users. Computer professionals use different number systems according to their requirements to communicate with the computer system. Therefore, before understanding the various computer codes, we need to understand the concept of number systems. All the number systems used by computer professionals to interact with computer systems come under the category of positional number system. The positional number system is a number system in which numbers are represented using some symbols called digits and the values of these numbers can be determined by taking the position of digits into consideration. The different number systems, which come under the category of positional number system, are as follows:

- Decimal system
- Binary system
- Hexadecimal system
- Octal system

1.10 DECIMAL SYSTEM

The decimal system is a positional number system that uses 10 as a base to represent different values. Therefore, this number system is also known as base 10 number system. In this system, 10 symbols are available for representing the values. These symbols include the digits from 0 to 9. The decimal system can be used to represent both the integer as well as floating point values. The floating point values are generally represented in this system by using a period called decimal point. The value of any number represented in the decimal system can be determined by first multiplying the weight associated with each digit in the given number with the digit itself and then adding all these values produced as a result of multiplication operation. The weight associated with any digit depends upon the position of the digit itself in the given number. The most common method to determine the weight of any digit in any number system is to raise the base of the number system to a power that initially starts with a 0 and then increases by 1 as we move from right to left in the given number. To understand this concept, let us consider the following floating point number represented in the decimal system:

Decimal point

↓

6 5 4 3 . 1 2 4

In the above example, the value 6543, which comes before the decimal point, is called integer value and the value 124, which comes after the decimal point, is called fraction value.

$6 \times 10^3 \times 5 \times 10^2 + 4 \times 10^1 + 3 \times 10^0 + 1 \times 10^{-1} + 2 \times 10^{-2} + 4 \times 10^{-3}$

$= 6000 + 500 + 40 + 3 + 0.1 + 0.02 + 0.004$

$= 6543.124$

1.11 BINARY SYSTEM

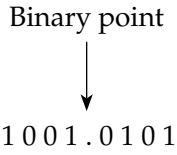
The binary number system uses base 2 to represent different values. Therefore, the binary system is also known as base-2 system. As this system uses base 2, only two symbols are available for representing the different values in this system.

The following are some of the technical terms used in binary system:

- **Bit.** It is the smallest unit of information used in a computer system. It can either have the value 0 or 1. Derived from the words Binary *digIT*.
- **Nibble**. It is a combination of 4 bits.

- **Byte.** It is a combination of 8 bits. Derived from words 'by eight'.
- **Word.** It is a combination of 16 bits.
- **Double word.** It is a combination of 32 bits.
- **Kilobyte (KB).** It is used to represent the 1024 bytes of information.
- **Megabyte (MB).** It is used to represent the 1024 KBs of information.
- **Gigabyte (GB).** It is used to represent the 1024 MBs of information.

In the binary system, the weight of any bit can be determined by raising 2 to a power equivalent to the position of bit in the number. To understand this concept, let us consider the following binary number:



In the binary system, the point used to separate the integer and the fraction part of a number is known as binary point. Like the decimal system, the powers to the base increases by 1 towards the left for the integer part and decreases by 1 towards the right for the fraction part. The value of the given binary number can be determined as the sum of the products of the bits multiplied by the weight of the bit itself. Therefore, the value of the binary number 101001.0101 can be obtained as:

$1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4}$

$= 32 + 8 + 1 + 0.25 + 0.0625$

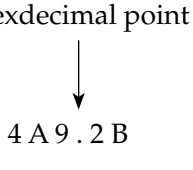
$= 41.3125$

The binary number 101001.0101 represents the decimal value 41.3125.

1.12 HEXADECIMAL SYSTEM

The hexadecimal system is a positional number system that uses base 16 to represent different values. Therefore, this number system is known as base-16 system. As this system uses base 16, 16 symbols are available for representing the values in this system. These symbols are the digits 0–9 and the letters A, B, C, D, E and F. The digits 0–9 are used to represent the decimal values 0 through 9 and the letters A, B, C, D, E and F are used to represent the decimal values 10 through 15.

The weight associated with each symbol in the given hexadecimal number can be determined by raising 16 to a power equivalent to the position of the digit in the number. To understand this concept, let us consider the following hexadecimal number:



In the hexadecimal system, the point used to separate the integer and the fraction part of a number is known as hexadecimal point. The value of the hexadecimal number can also be determined as the sum of the products of the symbol multiplied by the weight of the symbol itself. Therefore, the value of the given hexadecimal number is:

$= 4 \times 16^2 + 10 \times 16^1 + 9 \times 16^0 + 2 \times 16^{-1} + 11 \times 16^{-2}$

$= 1024 + 160 + 9 + 0.125 + 0.0429$

$= 1193 + 0.1679$

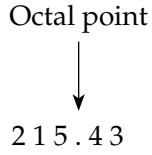
$= 1193.1679$

The hexadecimal number 4A9. 2B represents the decimal value 1193.1679.

1.13 OCTAL SYSTEM

The octal system is the positional number system that uses base 8 to represent different values. Therefore, this number system is also known as base-8 system. As this system uses base 8, eight symbols are available for representing the values in this system. These symbols are the digits 0 to 7.

The weight associated with each digit in the given octal number can be determined by raising 8 to a power equivalent to the position of digit in the number. To understand this concept, let us consider the following octal number:



In octal system, the point used to separate the integer and the fraction part of a number is known as octal point. Using these place values, we can now determine the value of the given octal number as:

$2 \times 8^2 + 1 \times 8^1 + 5 \times 8^0 + 4 \times 8^{-1} + 3 \times 8^{-2}$

$= 128 + 8 + 5 + 0.5 + 0.0469$

$= 141 + 0.5469$

$= 141.5469$

The octal number 215.43 represents the decimal value 141.5469.

1.14 4-BIT BINARY CODED DECIMAL (BCD) SYSTEMS

The BCD system is employed by computer systems to encode the decimal number into its equivalent binary number. This is generally accomplished by encoding each digit of the decimal number into its equivalent binary sequence. The main advantage of BCD system is that it is a fast and efficient system to convert the decimal numbers into binary numbers as compared to the pure binary system. However, the implementation of this coding system requires a lot of circuits that makes the design of the computer systems very complicated.

The 4-bit BCD system is usually employed by the computer systems to represent and process numerical data only. In the 4-bit BCD system, each digit of the decimal number is encoded to its corresponding 4-bit binary sequence. The two most popular 4-bit BCD systems are:

- Weighted 4-bit BCD code
- Excess – 3 (XS – 3) BCD code

1.14.1 Weighted 4-Bit BCD Code

The weighted 4-bit BCD code is more commonly known as 8421 weighted code. It is called weighted code because it encodes the decimal system into binary system by using the concept of positional weighting into consideration. In this code, each decimal digit is encoded into its 4-bit binary number in which the bits from left to right have the weights 8, 4, 2, and 1 respectively. Table 1.2 lists the weighted 4-bit BCD code for decimal digits 0 through 9.

TABLE 1.2 Weighted 4-bit BCD codes

Decimal digits	Weighted 4-bit BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001


NOTE: Apart from 8421, some other weighted BCD codes are 4221, 2421 and 5211.

EXAMPLE 1.1 *Represent the decimal number 5327 in 8421 BCD code.*

Solution The given decimal number is 5327.

The corresponding 4-bit 8421 BCD representation of decimal digit 5 is 0101.
The corresponding 4-bit 8421 BCD representation of decimal digit 3 is 0011.
The corresponding 4-bit 8421 BCD representation of decimal digit 2 is 0010.
The corresponding 4-bit 8421 BCD representation of decimal digit 7 is 0111.
Therefore, the 8421 BCD representation of decimal number 5327 is 0101 0011 0010 0111.

EXAMPLE 1.2 *Determine the decimal number corresponding to the 8421 BCD code 01001001.*

Solution The given 8421 BCD code is 01001001.

To determine the equivalent decimal number, simply divide the 8421 BCD code into sets of 4-bit binary digits as:

0100 1001

The decimal number corresponding to the binary digits 0100 is 4.

The decimal number corresponding to the binary digits 1001 is 9.

Therefore, the decimal number equivalent to 8421 BCD code 0100 1001 is 49.

1.14.2 Excess-3 BCD Code

The Excess-3 (XS-3) BCD code does not use the principle of positional weights into consideration while converting the decimal numbers to 4-bit BCD system. Therefore, we can say that this code is a non-weighted BCD code. The function of XS-3 code is to transform the decimal numbers into their corresponding 4-bit BCD code. In this code, the decimal number is transformed to the 4-bit BCD code by first adding 3 to all the digits of the number and then converting the excess digits, so obtained, into their corresponding 8421 BCD code. Therefore, we can say that the XS-3 code is strongly related with 8421 BCD code in its functioning. Table 1.3 lists the XS-3 BCD code for decimal digits 0 through 9.

TABLE 1.3 Excess–3 BCD codes

Decimal digits	Excess-3 BCD code
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

NOTE: Apart from XS-3 code, the other nonweighted BCD code is 4-bit Gray code.

EXAMPLE 1.3 *Convert the decimal number 85 to XS–3 BCD code.*

Solution The given decimal number is 85.

Now, add 3 to each digit of the given decimal number as:

8 + 3 = 11

5 + 3 = 8

The corresponding 4-bit 8421 BCD representation of the decimal digit 11 is 1011.

The corresponding 4-bit 8421 BCD representation of the decimal digit 8 is 1000.

Therefore, the XS–3 BCD representation of the decimal number 85 is 1011 1000.

EXAMPLE 1.4 *Represent the decimal number 173 in XS-3 BCD code.*

Solution The given decimal number is 173.

Now, add 3 to each digit of the given decimal number as:

1 + 3 = 4

7 + 3 =10

3 + 3 = 6

The corresponding 4-bit 8421 BCD representation of the decimal digit 4 is 0100.

The corresponding 4-bit 8421 BCD representation of the decimal digit 10 is 1010.

The corresponding 4-bit 8421 BCD representation of the decimal digit 6 is 0110.

Therefore, the XS-3 BCD representation of the decimal number 173 is 0100 1010 0110.

NOTE: 4-bit BCD systems are inadequate for representing and handling nonnumeric data. For this purpose, 6-bit BCD and 8-bit BCD systems have been developed.

1.15 8-BIT BCD SYSTEMS

The 8-bit BCD systems were developed to overcome the limitations of 6-bit BCD systems. The 6-bit BCD systems can handle numeric as well as nonnumeric data but with few special characters. The 8-bit BCD systems can handle numeric as well as nonnumeric data with almost all the special characters such as +, - , *, /, @, $, etc. Therefore, the various codes under the category of 8-bit BCD systems are also known as alphanumeric codes. The three most popular 8-bit BCD codes are:

- Extended Binary Coded Decimal Interchange Code (EBCDIC)
- American Standard Code for Information Interchange (ASCII)
- Gray code

1.15.1 EBCDIC Code

The EBCDIC code is an 8-bit alphanumeric code that was developed by IBM to represent alphabets, decimal digits and special characters, including control characters. Control characters are the special characters that are used to perform a specific function. For example, the control character FF is used to feed the next page into the printer or eject the current page from the printer. The EBCDIC codes are generally the decimal and the hexadecimal representation of different characters. This code is rarely used by non IBM-compatible computer systems. Table 1.4 lists some important EBCDIC characters and their corresponding decimal and hexadecimal representation.

TABLE 1.4 EBCDIC codes

Characters	Decimal representation	Hexadecimal representation
NUL	0	00
SOH	1	01
STX	2	02
ETX	3	03
HT	5	05
DEL	7	07
VT	11	0B
FF	12	0C
CR	13	0D
SO	14	0E
SI	15	0F
DLE	16	10
IUS	31	1F
ESC	39	27
BEL	47	2F
SUB	63	3F
[	74	4A
.	75	4B
<	76	4C
(	77	4D
+	78	4E
&	80	50
$	91	5B
*	92	5C
-	96	60
/	97	61
%	108	6C

?	111	6F
=	126	7E
A – i	129 – 137	81 – 89
j – r	145 – 153	91 – 99
s – z	162 – 169	A2 – A9
A – I	193 – 201	C1 – C9
J – R	209 – 217	D1 – D9
S – Z	226 – 233	E2 – E9
0 – 9	240 – 249	F0 – F9

1.15.2 ASCII Code

The ASCII code is pronounced as ASKEE and is used for the same purpose for which the EBCDIC code is used. However, this code is more popular than EBCDIC code as unlike the EBCDIC code, this code can be implemented by most of the non-IBM computer systems. Initially, this code was developed as a 7-bit BCD code to handle 128 characters but later it was modified to an 8-bit code. We can check the value of any ASCII code by just holding down the Alt key and typing the ASCII code. For example, when we hold down the Alt key and type 66 from the keyboard, then the character B appears on the screen. This shows that the ASCII decimal code 66 represents the character B. Table 1.5 lists some important ASCII codes and their corresponding decimal and hexadecimal representations.

TABLE 1.5 ASCII codes

Characters	Decimal representation	Hexadecimal representation
NUL	0	0
SOH	1	1
STX	2	2
ETX	3	3
EOT	4	4
ENQ	5	5
ACK	6	6
BEL	7	7
BS	8	8
HT	9	9
CAN	24	18
SUB	26	1A
ESC	27	1B
RS	30	1E
US	31	1F

!	33	21
#	35	23
$	36	24
%	37	25
&	38	26
*	42	2A
+	43	2B
/	47	2F
0 – 9	48 – 57	30 – 39
<	60	3C
=	61	3D
>	62	3E
?	63	3F
A – I	65 – 73	41 – 49
J – O	74 – 79	4A – 4F
P – Z	80 – 90	50 – 5A
a – i	97 – 105	61 – 69
j – o	106 – 111	6A – 6F
p – z	112 – 122	70 – 7A

1.15.3 Gray Code

Gray code is another important code that is also used to convert the decimal number into an 8-bit binary sequence. However, this conversion is carried in a manner that the contiguous digits of the decimal number differ from each other by one bit only. Table 1.6 lists the 8-bit Gray code for decimal numbers 0 through 9.

TABLE 1.6 8-Bit Gray code

Decimal number	8-Bit Gray code
0	00000000
1	00000001
2	00000011
3	00000010
4	00000110
5	00000111
6	00001111
7	00001011
8	00001001
9	00001101

We can convert the Gray coded number to its binary equivalent by remembering the following two major rules:

- The Most Significant Bit (MSB) of the Gray coded number and the equivalent binary number is always the same.
- The next-to-most significant bit of the binary number can be determined by adding the MSB of the binary number to the next-to-most significant bit of the gray coded number.

EXAMPLE 1.5 *Convert the Gray coded number 11010011 to its binary equivalent.*

Solution The given Gray coded number is 11010011.

The following table lists the steps showing the conversion of the Gray coded number into its binary equivalent:

S No.	Gray coded digit	Binary addition operation	Binary digit
1	1		1
2	1	1 + 1	0
3	0	0 + 0	0
4	1	1 + 0	1
5	0	0 + 1	1
6	0	0 + 1	1
7	1	1 + 1	0
8	1	1 + 0	1

Hence, the binary equivalent of Gray coded number 11010011 is 10011101.

We can also convert a number represented in the binary form to Gray code representation. For carrying out this conversion, we need to remember the following two rules:

- The Most Significant Digit (MSD) of the binary number and the gray coded number is always the same.
- The next MSD of the gray coded number can be obtained by adding the subsequent pair of bits of the binary number starting form the left.

Note: *We need to ignore the carry, if it is generated while adding the subsequent pairs of bits of the binary number.*

EXAMPLE 1.6 *Convert the binary number 10100011 to its equivalent Gray coded number.*

Solution The given binary number is 10100011.

The following table lists the steps showing the conversion of binary number to its equivalent Gray coded number:

S.No.	Binary digit	Binary addition operation	Gray coded digit
1	1		1
2	0	1 + 0	1
3	1	0 + 1	1
4	0	1 + 0	1
5	0	0 + 0	0
6	0	0 + 0	0
7	1	0 + 1	1
8	1	1 + 1	0

Hence, the Gray coded equivalent of the binary number 10100011 is 11110010.

1.16 16-BIT UNICODE

The 16-bit Unicode is an International 16-bit character set that contains a maximum of 216 = 65,536 different characters. These characters are sufficient to represent almost all the technical and special symbols used by the major languages of the world. The 16-bit Unicode, (also called 16-bit universal character set), encodes the different characters by assigning them a unique value. In computer terminology, this unique value is referred as code point. The code assigned to each character of different languages is universal and can be used on any platform without any modification. Therefore, we can say that the 16-bit Unicode allows the computer systems to deal with almost all the characters belonging to different languages used in the world.

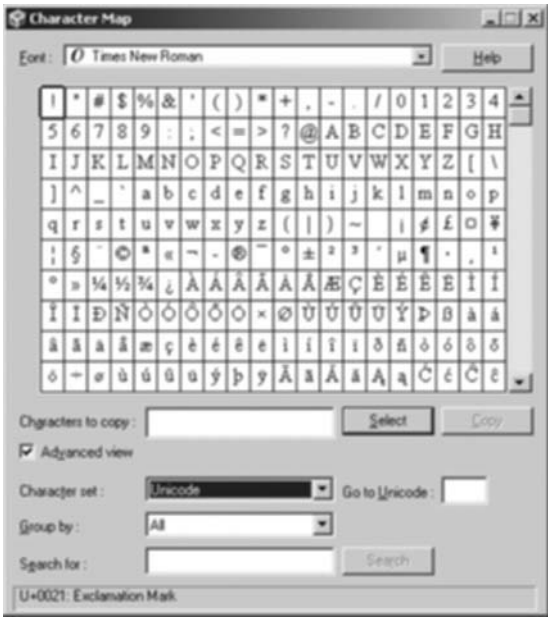


Fig. 1.42 **The character map window**

The 16-bit Unicode is a character code that is supported by almost all the operating systems such as MS Windows, Linux and Mac OS X. For example, MS Windows operating system allows the use of all the Unicode characters through an accessory called Character Map. Figure 1.42 shows the user interface of the character map.

Using the Character Map window, we can select any of the Unicode characters and copy it to the clipboard. After copying it to the clipboard, we can use the selected Unicode character in any application running under MS Windows operating system.

1.17 CONVERSION OF NUMBERS

The computer systems accept data in decimal form, whereas data is stored and processed in binary form. Therefore, it becomes necessary to convert the numbers represented in one system into the numbers represented in another system. The different types of number system conversions can be divided into the following major categories:

- Non-decimal to decimal
- Decimal to non-decimal
- Octal to hexadecimal

1.17.1 Non-Decimal to Decimal

The non-decimal to decimal conversions can be implemented by taking the concept of place values into consideration. The non-decimal to decimal conversion includes the following number system conversions:

- Binary to decimal conversion
- Hexadecimal to decimal conversion
- Octal to decimal conversion

Binary to decimal conversion A binary number can be converted to equivalent decimal number by calculating the sum of the products of each bit multiplied by its corresponding place value.

EXAMPLE 1.7 *Convert the binary number 10101101 into its corresponding decimal number.*

Solution The given binary number is 10101101.

Now, calculate the sum of the products of each bit multiplied by its place value as:

$(1 \times 2^7) + (0 \times 2^6) + (1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$

$= 128 + 0 + 32 + 0 + 8 + 4 + 0 + 1$

$= 173$

Therefore, the binary number 10101101 is equivalent to 173 in the decimal system.

EXAMPLE 1.8 *Convert the binary number 1011.010 into its equivalent in decimal system.*

Solution The given binary number is 1011.010.

Now, calculate the sum of the products of each bit multiplied by its place value as:

$(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3})$

$= 8 + 2 + 1 + ¼$

$= 11 + 0.25$

$= 11.25$

Therefore, the binary number 1011.010 is equivalent to 11.25 in the decimal system.

Hexadecimal to decimal conversion A hexadecimal number can be converted into its equivalent number in decimal system by calculating the sum of the products of each symbol multiplied by its corresponding place value.

EXAMPLE 1.9 *Convert the hexadecimal number A53 into its equivalent in decimal system.*

Solution The given hexadecimal number is A53.

Now, calculate the sum of the products of each symbol multiplied by its place value as:

$(10 \times 16^2) + (5 \times 16^1) + (3 \times 16^0)$

$= 2560 + 80 + 3$

$= 2643$

Therefore, the hexadecimal number A53 is equivalent to 2643 in the decimal system.

EXAMPLE 1.10 *Convert the hexadecimal number AB21.34 into its equivalent in the decimal system.*

Solution The given hexadecimal number is AB21.34.

Now, calculate the sum of the products of each symbol multiplied by its place value as:

$(10 \times 16^3) + (11 \times 16^2) + (2 \times 16^1) + (1 \times 16^0) + (3 \times 16^{-1}) + (4 \times 16^{-2})$

$= 40960 + 2816 + 32 + 1 + 3/16 + 4/256$

$= 43809 + 0.1875 + 0.015625$

$= 43809.203$

Therefore, the hexadecimal number AB21.34 is equivalent to 43809.203 in the decimal system.

Octal to decimal conversion An octal number can be converted into its equivalent number in decimal system by calculating the sum of the products of each digit multiplied by its corresponding place value.

EXAMPLE 1.11 *Convert the octal number 5324 into its equivalent in decimal system.*

Solution The given octal number is 5324.

Now, calculate the sum of the products of each digit multiplied by its place value as:

$(5 \times 8^3) + (3 \times 8^2) + (2 \times 8^1) + (4 \times 8^0)$

$= 2560 + 192 + 16 + 4$

$= 2772$

Therefore, the octal number 5324 is equivalent to 2772 in the decimal system.

EXAMPLE 1.12 *Convert the octal number 325.12 into its equivalent in decimal system.*

Solution The given octal number is 325.12.

Now, calculate the sum of the products of each digit multiplied by its place value as:

$(3 \times 8^2) + (2 \times 8^1) + (5 \times 8^0) + (1 \times 8^{-1}) + (2 \times 8^{-2})$

$= 192 + 16 + 5 + 1/8 + 2/64$

$= 213 + 0.125 + 0.03125$

$= 213.15625$

Therefore, the octal number 325.12 is equivalent to 213.15625 in the decimal system.

1.17.2 Decimal to Non-Decimal

The decimal to non-decimal conversions are carried out by continually dividing the decimal number by the base of the desired number system till the decimal number becomes zero. After the decimal number becomes zero, we may note down the remainders calculated at each successive division from last to first to obtain the decimal number into the desired system. The decimal to non-decimal conversion includes the following number system conversions:

- Decimal to binary conversion
- Decimal to octal conversion

Decimal to binary conversion The decimal to binary conversion is performed by repeatedly dividing the decimal number by 2 till the decimal number becomes zero and then reading the remainders from last to first to obtain the binary equivalent of the given decimal number. The following examples illustrate the method of converting a decimal number to its binary equivalent:

EXAMPLE 1.13 *Convert the decimal number 111 into its equivalent binary number.*

Solution The given decimal number is 111.

The following table lists the steps showing the conversion of the given decimal number to its binary equivalent:

Decimal number	Divisor	Quotient	Remainder
111	2	55	1
55	2	27	1
27	2	13	1
13	2	6	1
6	2	3	0
3	2	1	1
1	2	0	1

Now, read the remainders calculated in the above table in upward direction to obtain the binary equivalent, which is 1101111.

Therefore, the binary equivalent of the decimal number 111 is 1101111.

Decimal to octal conversion The decimal to octal conversion is performed by repeatedly dividing the decimal number by 8 till the decimal number becomes zero and reading the remainders from last to first to obtain the octal equivalent of the given decimal number. The following examples illustrate the method of converting decimal number to its octal equivalent:

EXAMPLE 1.14 *Convert the decimal number 45796 to its equivalent octal number.*

Solution The given decimal number is 45796.

The following table lists the steps showing the conversion of the given decimal number to its octal equivalent:

Decimal number	Divisor	Quotient	Remainder
45796	8	5724	4
5724	8	715	4
715	8	89	3
89	8	11	1
11	8	1	3
1	8	0	1

Now, read the remainders calculated in the above table in upward direction to obtain the octal equivalent, which is 131344.

Therefore, the corresponding octal equivalent of 45796 is 131344.

1.17.3 Octal to Hexadecimal

A given octal number can be converted into its equivalent hexadecimal number in two different steps. Firstly, we need to convert the given octal number into its binary equivalent. After obtaining the binary equivalent, we need to divide the binary number into 4-bit sections starting from the LSB.

The octal to binary conversion is a simple process. In this type of conversion, we need to represent each digit in the octal number to its equivalent 3-bit binary number.

EXAMPLE 1.15 *Convert the octal number 365 into its equivalent hexadecimal number.*

Solution The given octal number is 365.

Firstly, convert the given octal number into its binary equivalent.

The 3-bit binary equivalent of the octal digit 3 is 011.
The 3-bit binary equivalent of the octal digit 6 is 110.
The 3-bit binary equivalent of the octal digit 5 is 101.
Therefore, the binary equivalent of the given octal number is 011110101.
Now, we need to convert this binary number into the equivalent hexadecimal number.
Divide the binary number into 4-bit sections as:

0000 1111 0101

The hexadecimal equivalent of 4-bit binary number 0000 is 0.
The hexadecimal equivalent of 4-bit binary number 1111 is F.
The hexadecimal equivalent of 4-bit binary number 0101 is 5.
Therefore, the hexadecimal equivalent of the given octal number is F5.

SUMMARY

Computer is a machine that accepts data as input, and stores and processes this data, based on the instructions provided by the user. A computer mainly comprises of hardware and software. Along with these components, the data and users are also related to a computer as without them a computer cannot function. Hardware refers to the physical components of the computer, such as keyboard, mouse, CPU and printer. Software refers to the set of instructions provided to the computer by the user for performing a specific task. Data refers to the type of input given by the user to the computer—it can be in the form of numbers, words or images. Users refer to the persons who use the computer for getting the required result corresponding to the given input. Computers are used for performing complex calculations at a very fast speed. Earlier, the manual computing devices, such as sand table, abacus and napier bones were used for performing different calculations. These devices consumed a large amount of time and were unable to handle large numbers in calculations. Due to the drawbacks of manual computing devices, the automated computing devices such as MARK I, ENIAC and EDVAC were developed for performing calculations automatically. These devices were much faster as compared to the manual computing devices.

There are five generations associated with the evolution of computers. During these generations, the computers have seen tremendous shift in technology, size, and speed. Computers can be classified into three categories—analog computers, digital computer and hybrid computers—on the basis of their operating principles. Depending upon their application areas, computers are categorised into two types—general-purpose computers and special-purpose computers. On the basis of their size and capability, computers are categorised into four types—micro computers, mini computers, super computers and mainframe computers. These days, computers are being used in almost every field because of their high processing speed and large storage capacity. Education, science, business and healthcare are some of the areas where computers are widely used.

Computer systems represent and process data in binary form only. However, they can accept data in many forms. Thus, computers support a mechanism of converting data from the accepted form to the binary form; and this is achieved with the help of number systems and computer codes. The various number systems are decimal system, binary system, hexadecimal system and octal system. The various computer codes are BCD, XS-3, EBCDIC, ASCII, etc.

POINTS TO REMEMBER

- **Computer**: It is an electronic machine that takes input from the user, and stores and processes the input to generate the output in the form of useful information to the user.
- **Data**: It refers to the raw details that need to be processed to generate some useful information.
- **Program**: It refers to the set of commands that can be executed by the computer in a sequential or non-sequential manner.
- **CPU**: It is the processor of the computer that is responsible for controlling and executing instructions.
- **Monitor**: It is a screen, which displays the information in the visual form after receiving the video signals from the computer.
- **Transistor**: It is a semiconductor device that is used to increase the power of the incoming signals by preserving the shape of the original signal.
- **ICs**: These are the circuits that combine various electronic components such as transistors, resistors, capacitors, etc. onto a single small silicon chip.
- **Digital computer**: It is a type of computer that stores and processes data in digital form and is also known as the digital information processing system.
- **Hybrid computer**: It is a combination of analog computer and digital computer because it encompasses the best features of both these computers.
- **Micro computer**: It is a small and cheap digital computer that is designed to be used by individuals.
- **Mainframe computer**: It is a very large computer that is employed by large business organisations for handling major applications such as financial transaction processing applications and ERP.
- **Super computer**: It is the fastest type of computer that can perform complex operations at a very high speed.
- **Positional Number System:** It is a number system in which numbers are represented using symbols called digits, and the values of these numbers are determined by taking the position of the digits into consideration.
- **Decimal System:** It is a positional number system that uses 10 as a base to represent different values.
- **Binary System:** It uses base 2 to represent different values.
- **Hexadecimal System:** It is a positional number system that uses base 16 to represent different values.
- **Octal System:** It is the positional number system that uses base 8 to represent different values.
- **4-Bit Binary Coded Decimal (BCD) Systems:** It is employed by computer systems to encode the decimal number into its equivalent binary number.
- **8-Bit BCD Systems:** It can handle numeric as well as nonnumeric data with almost all the special characters such as +, - , *, /, @, $, etc.
- **EBCDIC code:** It is an 8-bit alphanumeric code that was developed by IBM to represent alphabets, decimal digits and special characters, including control characters.

- **ASCII code:** It is pronounced as ASKEE and is used for the same purpose for which the EBCDIC code is used.
- **Gray code:** It is another important code that is also used to convert the decimal number into a 8-bit binary sequence.

REVIEW QUESTIONS

1. The memory unit of a computer is also known as brain of the computer.
2. Mouse is an electronic device, which is used for receiving inputs from the user.
3. The manual computing device, Napier bones was developed by Michael Napier in the year 1614.
4. The idea of using bones to carry out the multiplication of numbers was modified by Edmund Gunter in 1620 to produce a device known as slide rule.
5. Pascaline was a calculator developed by Blaise Pascal in 1642.
6. Gottfried Wilhem is known as the father of the modern computer.
7. Colossus was a special-purpose electronic device that used the integrated circuits in performing different operations.
8. First generation computers used the vacuum tubes technology for calculation as well as for storage and control purposes.
9. UNIVAC is an example of second generation computer.
10. Transistor is a semiconductor device having two terminals—emitter and base.
11. Assembly language is a high-level language that allows the programmer to use simple English words—called mnemonics—to represent different instructions in a program.
12. The main characteristic feature of third generation computers was the use of VLSI technology.
13. The fifth generation computers are based on the Ultra Large Scale Integration (ULSI) technology that allows almost ten million electronic components to be fabricated on one small chip.
14. Fifth generation computers are the fastest and most powerful computers till date.
15. A micro computer is the fastest type of computer that can perform complex operations at a very high speed.
16. CRAY3 and Cyber 205 are the examples of mainframe computers.

1. A _____ is an electronic machine that takes input from the user and stores and processes the given input to generate the output in the form of useful information to the user.

2. The raw details that need to be processed to generate some useful information is known as _____.
3. The set of commands that can be executed by the computer in sequential or non-sequential manner is known as _____.
4. _____ is the processor of the computer that is responsible for controlling and executing the various instructions.
5. _____ is a screen, which displays the information in visual form after receiving the video signals from the computer.
6. _____ was a device that arranged stones in three channels in the sand.
7. In _____, the wooden frame consists of many wires with beads sliding on them and it was also known as a counting frame.
8. _____ computing device consists of a board whose left edge is divided into 9 squares and these 9 squares are used to hold the numbers from 1 to 9.
9. _____ is considered to be the father of modern digital computers.
10. _____was the first device that used all the features of a modern digital computer.
11. Colossus was a special-purpose electronic device that used the _____ technology to perform different operations.
12. _____ computers were also known as vacuum tubes or thermionic valves based machines.
13. _____ is a semiconductor device that is used to increase the power of the incoming signals by preserving the shape of the original signal.
14. _____ is a low-level language that allows the programmer to use simple English words called mnemonics to represent different instructions in a program.
15. The major characteristic features of third generation computers was the use of _____.
16. The invention of _____ and _____ technology led to the development of fourth generation computers.
17. The fifth generation computers are based on the _____ technology that allows almost ten million electronic components to be fabricated on one small chip.
18. _____, also known as digital information processing system, is a type of computer that stores and processes data in digital form.
19. A _____ is the fastest type of computer that can perform complex operations at a very high speed.
20. _____ is a number system in which numbers are represented using some symbols called digits.
21. Decimal system is a positional number system that uses _____ as a base to represent different values.
22. Binary system uses _____ to represent different values.
23. _____ is the positional number system that uses base 16 to represent different values.
24. Octal system is the positional number system that uses _____ to represent different values.

25. _____ is an 8-bit alphanumeric code that was developed by IBM to represent alphabets, decimal digits and special characters, including control characters.
26. Gray code is used to convert _____ into _____ binary sequence.

MULTIPLE CHOICE QUESTIONS

1. Which component of the computer is known as the brain of computer?
   A. Monitor B. CPU
   C. Memory D. None of the above
2. Which of the following is an input device?
   A. Printer B. Monitor
   C. Mouse D. None of the above
3. Which of the following is a characteristic of the modern digital computer?
   A. High speed B. Large storage capacity
   C. Greater accuracy D. All of the above
4. Which one of the following devices used stones for performing calculations?
   A. Sand table B. Abacus
   C. Napier bones D. Pascaline
5. Who is known as the father of modern digital computers?
   A. Gottfried Wilhem Von Leibriz B. Charles Babbage
   C. Alan Mathison D. John Mauchly
6. Which of the following was the first pure electronic programmable computer developed in the year 1944?
   A. MARK I B. ENIAC
   C. Colossus D. EDVAC
7. Which of the following was the first commercial computer developed in the year 1951:
   A. ENIAC B. EDVAC
   C. EDSAC D. UNIVAC
8. Into how many computer generations is the evolution of computers divided?
   A. Four B. Five
   C. Six D. Seven
9. Which technology was used in the first generation computers?
   A. Transistors B. Vacuum tubes
   C. ICs D. None of the above
10. Which technology was used in the second generation computers?
    A. Transistors B. Vacuum tubes
    C. Microprocessors D. ICs

11. Which technology was used in the third generation computers?
    A. Transistors B. Vacuum tubes
    C. ICs D. All of the above
12. Which technology was used in the fourth generation computers?
    A. Microprocessors B. Vacuum tubes
    C. ICs D. Transistors
13. Which semiconductor device is used to increase the power of the incoming signals by preserving the shape of the original signal?
    A. Sand table B. Transistor
    C. Vacuum tubes D. None of the above
14. What of the following is the connection present in a transistor?
    A. Emitter B. Base
    C. Collector D. All of the above
15. In which of the following generation of computers was the assembly language introduced?
    A. First B. Second
    C. Third D. Fourth
16. Which technology is also known as microelectronics technology?
    A. Vacuum tubes B. ICs
    C. Microprocessors D. Transistors
17. Which generation uses the ULSI technology?
    A. Second B. Third
    C. Fourth D. Fifth
18. Which of the following units is a part of the CPU?
    A. ALU B. CU
    C. Memory unit D. All of the above
19. Which of the following is known as a midrange computer?
    A. Microcomputer B. Mini computer
    C. Mainframe computer D. Super computer
20. Which form of data is used in number systems?
    A. 0's and 1's B. 1's and 2's
    C. 3's and 4's D. 0's and 2's
21. Which of the following uses 10 as a base to represent different values?
    A. Decimal system B. Binary system
    C. Hexadecimal system D. Octal system
22. Which of the following uses 2 as a base to represent different values?
    A. Decimal system B. Binary system
    C. Hexadecimal system D. Octal system

23. Which of the following uses 16 as a base to represent different values?
    A. Decimal system
    B. Binary system
    C. Hexadecimal system
    D. Octal system
24. Which of the following uses 8 as a base to represent different values?
    A. Decimal system
    B. Binary system
    C. Hexadecimal system
    D. Octal system
25. What is the 8421 BCD code for decimal number 5327?
    A. 0101 0010 1100 1011
    B. 0011 0001 0101 1100
    C. 0010 0101 1100 0011
    D. 0101 0011 0010 0111
26. What is the decimal code for the character ESC?
    A. 39
    B. 27
    C. 64
    D. 21
27. What is the hexadecimal representation of the character ESC?
    A. 27
    B. 64
    C. 26
    D. 14
28. What is the 8-bit Gray code for decimal number 3?
    A. 00000011
    B. 00000110
    C. 00001100
    D. 00000010
29. Which of the following is the binary equivalent of decimal number 111?
    A. 11000110
    B. 00011110
    C. 00110011
    D. 11011111
30. Which of the following is the hexadecimal equivalent of octal number 365?
    A. C6
    B. D3
    C. B2
    D. F5

ANSWERS

True or False

1. False	2. True	3. False	4. True	5. True	6. False	7. False	8. True
9. False	10. False	11. False	12. False	13. True	14. True	15. False	16. False

Fill in the Blanks

1. Computer
2. Data
3. Program
4. CPU
5. Monitor
6. Sand table
7. Abacus
8. Napeir bones
9. Charles Babbage
10. Analytical engine
11. Vacuum tube
12. First generation computers
13. Transistor
14. Assembly language
15. Integrated Circuits (ICs)
16. LSI technology and VLSI technology

17. Ultra Large Scale Integration (ULSI)
18. Digital computer
19. Super computer
20. Positional number system
21. Base 10
22. Base 2
23. Hexadecimal system
24. Base 8
25. EBCDIC code
26. Decimal number and 8-bit.

Multiple Choice Questions

1. B	2. C	3. D	4. A	5. B	6. C	7. D	8. B
9. B	10. A	11. C	12. A	13. B	14. D	15. B	16. B
17. D	18. D	19. B	20. A	21. A	22. B	23. C	24. D
25. D	26. A	27. A	28. D	29. D	30. D		

EXERCISE QUESTIONS

1. What are the different components of a computer? Explain with the help of an example.
2. Explain the various characteristics of a computer?
3. Differentiate between manual computing devices and automated computing devices.
4. Explain the working of napier bones device for performing calculations.
5. What do you mean by UNIVAC?
6. Explain the first generation computers?
7. What were the advantages and disadvantages of using the first generation computers?
8. Which technology was used in the second generation computer and how was it better than the technology used in the first generation computers?
9. What are the advantages and disadvantages of the second generation computers?
10. Give the advantages and disadvantages of third generation computers.
11. Mention the year in which fourth generation computers were introduced.
12. Explain how the functioning of the fourth generation computer is better than the other generations of computer?
13. Explain the fifth generation computers and also elaborate on the various advantages of using these computers?
14. Differentiate between analog and digital computers.
15. How general-purpose computers are different from special-purpose computers?
16. Describe the various types of computers on the basis of size and capability.
17. Draw the block diagram of a micro computer.
18. Explain number systems and their different types.
19. Explain decimal and binary systems with suitable examples.
20. Explain hexadecimal and octal systems with the help of examples.

ANSWERS TO 2009 QUESTION PAPERS

SHORT ANSWER QUESTIONS

1. State the characteristics of computers. (Anna University, Jan - Feb 2009)

   Ans: Refer Section 1.4

2. How will you classify computer systems? (Anna University, Jan - Feb 2009)

   Ans: Refer Section 1.7

3. Give any two tasks which humans perform better than computers.

   (Anna University, Jan - Feb 2009)

   Ans: The tasks that humans can perform better than computers are:
   - Voice recognition
   - Image recognition

4. What is the use of computers in medicine and healthcare?

   (Anna University, Jan - Feb 2009)

   Ans: The following points explain the use of computers in medicine and healthcare:
   - Computers are used by doctors to diagnose various types of diseases and ailments.
   - Analog and digital devices are connected with computers enabling the doctors to monitor the condition of a patient and view the internal organs of the body.
   - Computers are also used for maintaining patients' records.

5. Convert the binary number 100110 into its octal equivalent.

   (Anna University, Jan - Feb 2009)

   Ans: The octal equivalent of 100 and 110 is shown below.

   100 100

   ↓ ↓

   4 6

   Therefore $(100110)_2 = (46)_8$

6. Mention the basic operations performed by data processors. (GE1102)

   Ans: The basic operations performed by data processors are:

   A. Fetch B. Decode C. Execute D. Store

7. Determine the decimal equivalent of the hexadecimal number AC.C8. (GE1102)

   Ans: Decimal equivalent of $(AC.C8)_{16} = 12\times16^0 + 10\times16^1+12\times16^{-1} + 8\times16^{-2}$

   $\Rightarrow 12 + 160 + 0.75 + 0.3125 = 172.78125$

   Therefore, $(AC.C8)_{16} = (172.78125)_{10}$

8. Differentiate between ASCII-7 and ASCII-8. (GE1102)

Ans:

ASCII-7	ASCII-8
It is a 7-bit code	It is an 8-bit code
Character set range is from 1 to 128	Character set range is from 128 to 255
The character set includes characters like a–z, A–Z, 0–9, backspace, null, etc.	The character set includes characters like Ç, ü, Ĝ, Ω etc.

DESCRIPTIVE QUESTIONS

1. Explain the evolution of computers. (GE2112)

   Ans: Refer Section 1.5
2. With a suitable diagram explain about computer organisation.
   (GE2112, Anna University, Jan - Feb 2009, GE1102)

   Ans: Refer Section 1.8
3. With suitable examples, explain about number systems. (GE2112)

   Ans: Refer Section 1.9
4. Explain in detail about the different generations of computers.
   (Anna University, Jan - Feb 2009, GE1102)

   Ans: Refer Section 1.6
5. Explain briefly about the various characteristics of computers.
   (Anna University, Jan - Feb 2009)

   Ans: Refer Section 1.4
6. Explain briefly about the following classes of computers:
   (Anna University, Jan - Feb 2009)

   A. Mainframe computers B. Super computers

   Ans: Refer Section 1.7.3
7. Explain the various types of computer memory. (Anna University, Jan - Feb 2009)

   Ans: Refer Section 1.8.2
8. Convert (6245.14)8 to its decimal equivalent. (GE1102)

   Ans: Decimal equivalent of $(6245.14)8 = 5\times8^0 + 4\times8^1 + 2\times8^2 + 6\times8^3 + 1\times8^{-1} + 4\times8^{-2}$

   $\Rightarrow 5 + 32 + 128 + 3072 + 0.125 + 0.0625 = 3237.1875$

   Therefore, $(6245.14)_8 = (3237.1875)_{10}$
9. Convert $(111001.101)_2$ to its decimal equivalent. (GE1102)

   Ans: Decimal equivalent of $(111001.101)_2 = 1\times2^0 + 0\times2^1 + 0\times2^2 + 1\times2^3 + 1\times2^4 + 1\times2^5 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}$

   $\Rightarrow = 1 + 0 + 0 + 8 + 16 + 32 + 0.5 + 0 + 0.125 = 57.625$

   Therefore, $(111001.101)_2 = (57.625)_{10}$

10. Convert the following numbers into their binary equivalents:
A. (59.6825)10 B. EBC16 C. 6548 (GE1102)

Ans:

A. The conversion of $(59.6825)_{10}$ into its binary equivalent is shown below:

2	59			
2	29	1	0.6825×2 = 1.365	1
2	14	1	0.365×2 = 0.73	0
2	7	0	0.73×2 = 1.46	1
2	3	1	0.46×2 = 0.88	0
2	1	1		
	0	1		

Therefore, $(59.6825)_{10} = (111011.1010)_2$

B. The binary equivalents of E, B and C are shown below.

E B C

↓ ↓ ↓

1110 1011 1100

Therefore, $(EBC)_{16} = (111010111100)_2$

C. The binary equivalent of 6, 5 and 4 are shown below.

6 5 4

↓ ↓ ↓

110 101 100

Therefore, $(654)_8 = (110101100)_2$

11. Using binary notation, write EBCDIC coding for the word PROGRAM. (GE1102)

Ans: The EBCDIC code for the word PROGRAM is:

P	R	O	G	R	A	M
11010111	11011001	11010110	11000111	11011001	11000001	11010100

12. Using binary notation, write BCD coding for the word COMPUTER. (GE1102)

Ans: The BCD Code for the word COMPUTER is:

C	O	M	P	U	T	E	R
010011	100110	100100	100111	110100	110011	010101	101001

ANSWERS TO 2010 AND 2011 QUESTION PAPERS

SHORT ANSWER QUESTIONS

1. Distinguish between Analog and Digital computer. (AU, Chn, Jan 2010)

   Ans: Analog computers represent the data in the form of continuous electrical signals while digital computers represent the data in digital form i.e. 0s and 1s. Digital computers are more accurate and faster as compared to analog computers.

2. Convert 0.4375 decimal to binary system. (AU, Chn, Jan 2010)

   Ans: The given decimal number is 0.4375.

   The following table lists the steps showing the conversion of the given decimal number to its binary equivalent:

Decimal number	Multiplier	Result	Decimal Part
.4375	2	.875	0
.875	2	1.75	1
.75	2	1.5	1
.5	2	1.0	1

   Now, read the decimal parts calculated in the above table in downward direction to obtain the binary equivalent, which is 0.0111.

   Therefore, $(0.4375)_{10} = (0.0111)_2$

3. List out the characteristics of computer. (AU, Chn, Jan 2011)

   Ans: Refer Section 1.4.

4. What are the classifications of computer? (AU, Chn, Jan 2011)

   Ans: Refer Section 1.7

5. What is a gate? Give a truth table of a NAND gate? (AU, Mdu, Jan 2011)

   Ans:

   Gate: A logic gate is an electronic switch which produces an electrical output signal (representing 0 or 1) based on one or more input signals (representing 0 or 1) using the principle of Boolean logic such as AND, OR, or NOT. Logic gates are used to design various combinational circuits such as half adders and full adders.

   Truth Table of Nand Gate

Input A	Input B	Output Y
0	0	1
0	1	1
1	0	1
1	1	0

6. What do you mean by ASCII? (AU, Mdu, Jan 2011)

Ans: Refer Section 1.15.2.

7. What is a flash drive? (AU, Mdu, Jan 2011)

Ans:

Flash Drive: A Universal Serial Bus (USB) flash drive is a removable storage device that is interfaced on the USB port of a computer system. It is pretty fast and compact in comparison to other storage devices like CD and floppy disk. One of the most important advantages of a USB drive is that it is larger in capacity as compared to other removable storage devices. Off late, it has become very popular amongst computer users.

8. Mention the characteristics of Computer. (AU, Cbe, Dec 10-Jan 11)

Ans: Refer Section 1.4.

9. Convert the Binary to Hexadecimal number:

1100 0110(*) (AU, Cbe, Dec 10-Jan 11)

Ans:

Binary to Hexadecimal Conversion

1. Divide the binary number in pair of four digits.

$$\underbrace{1100}\ \underbrace{0110}$$

2. For each of the pairs, deduce the corresponding decimal number.

$$\underbrace{1100}_{12}\ \underbrace{0110}_{6}$$

3. Represent each of the decimal numbers in hexadecimal form.

$$\underbrace{1100}_{C}\ \underbrace{0110}_{6}$$

Therefore, $(11000110)_2 = (C6)_{16}$

10. Difference between RAM and ROM. (AU, Cbe, Dec 10-Jan 11)

Ans: Refer Section 1.8.2 (Table 1.1)

11. Why computer is called an idiotic genius? (AU, Cbe, Dec 10-Jan 11)

Ans: Computer is referred as an idiotic genius because, irrespective of the fact that it can perform super fast calculations with immaculate accuracy, it still lacks human-like intelligence. It solely relies on users' instructions for performing its tasks; and does not think and make decisions of its own. Though, a special branch of computer science, that is Artificial Intelligence (AI), is specifically working towards imparting intelligence to the computer systems.

12. What is meant by positive and negative logic? (AU, TIR, Dec 10–Jan 11)

Ans:

Positive and Negative Logic

Positive and negative logic signify the type of input voltage that is used to represent the logic states 0 and 1. In case of positive logic, a positive voltage signal represents the logic state 1

while a negative voltage signal represents the logic state 0. Alternatively, in case of negative logic, a negative voltage signal represents the logic state 1 while a positive voltage signal represents the logic state 0.

Thus, positive logic is a situation where positive voltage triggers the digital circuit while negative logic is a situation where negative voltage triggers the digital circuit.

13. Define the terms sampling and quantisation. (AU, TIR, Dec 10–Jan 11)

Ans:

Sampling and Quantisation

Sampling and quantisation helps in digitizing a continuous-time signal or analog signal. Sampling involves taking samples of the input signal at regular time intervals while quantization helps in approximating the continuous range of input values into a finite range of discrete values. The digital signals thus obtained are then fed into a computation system for performing the necessary computations. The discrete set of signals can also be used for reconstructing the original analog signal.

14. What is gray code? (AU, TIR, Dec 10–Jan 11)

Ans: Refer Section 1.15.3

15. Convert decimal number 100 to binary and octal form. (AU, TIR, Dec 10–Jan 11)

Ans:

Decimal to Binary Conversion

The given decimal number is 100.

The following table lists the steps showing the conversion of the given decimal number to its binary equivalent:

Decimal number	Divisor	Quotient	Remainder
100	2	50	0
50	2	25	0
25	2	12	1
12	2	6	0
6	2	3	0
3	2	1	1
1	2	0	1

Now, read the remainders calculated in the above table in upward direction to obtain the binary equivalent, which is 1100100.

Therefore, $(100)_{10} = (1100100)_2$

Decimal to Octal Conversion

The given decimal number is 100.

The following table lists the steps showing the conversion of the given decimal number to its octal equivalent:

Decimal number	Divisor	Quotient	Remainder
100	8	12	4
12	8	1	4
1	8	0	1

Now, read the remainders calculated in the above table in upward direction to obtain the octal equivalent, which is 144.

Therefore, $(100)_{10} = (144)_8$

16. Differentiate between RAM and ROM. (AU, TIR, Dec 10–Jan 11)

Ans: Refer Section 1.8.2 (Table 1.1)

17. What are the advantages of IC version of computer over Vacuum tube version of Computers? (AU, TRI, Jan 2011)

Ans:

Advantages

The various advantages of IC version of computer over vacuum tube version are:

- IC-based computers are faster than the vacuum tube-based computers.
- IC-based computers are smaller in size as compared to vacuum tube-based computers.
- IC-based computers use middle and high level languages and are thus easier to program in comparison to vacuum tube-based computers, which typically receive inputs in machine language.

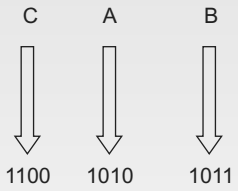
18. How will you convert CAB in hexadecimal to Binary? (AU, TRI, Jan 2011)

Ans:

CAB_{16} to Binary

The given hexadecimal number is CAB.

The binary equivalent of each of the digits is shown below.



Therefore, $(CAB)_{16} = (110010101011)_2$

DESCRIPTIVE QUESTIONS

1. What are the characteristics of a computer? Discuss. (AU, Chn, Jan 2010)

Ans: Refer section 1.4.

2. Briefly explain the various generations of computers. (AU, Chn, Jan 2010)

Ans: Refer section 1.6.

3. Convert the decimal number 59.8125 into binary and octal. (AU, Chn, Jan 2010)

Ans:

Decimal to binary:

The given decimal number is 59.8125.

The following table lists the steps showing the conversion of the decimal part (59) to its binary equivalent:

Decimal number	Divisor	Quotient	Remainder
59	2	29	1
29	2	14	1
14	2	7	0
7	2	3	1
3	2	1	1
1	2	0	1

Now, read the remainders calculated in the above table in upward direction to obtain the binary equivalent of 59, which is 111011.

Now, we need to calculate the binary equivalent of the fractional part.

The following table lists the steps showing the conversion of the fractional part (.8125) to its binary equivalent:

Decimal number	Multiplier	Result	Decimal Part
.8125	2	1.625	1
.625	2	1.25	1
.25	2	.5	0
.5	2	1.0	1

Now, read the decimal parts calculated in the above table in downward direction to obtain the binary equivalent, which is 0.1101.

Therefore, $(59.8125)_{10} = (111011.1101)_2$

Decimal to octal:

The given decimal number is 59.8125.

The following table lists the steps showing the conversion of the decimal part (59) to its octal equivalent:

Decimal number	Divisor	Quotient	Remainder
59	8	7	3
7	8	0	7

Now, read the remainders calculated in the above table in upward direction to obtain the octal equivalent of 59, which is 73.

Now, we need to calculate the octal equivalent of the fractional part.

The following table lists the steps showing the conversion of the fractional part (.8125) to its octal equivalent:

Decimal number	Multiplier	Result	Decimal Part
.8125	8	6.5	6
.5	8	4.0	4

Now, read the decimal parts calculated in the above table in downward direction to obtain the octal equivalent, which is 0.64.

Therefore, $(59.8125)_{10} = (73.64)_8$

4. Explain the different components of a computer system with block diagram. (AU, Chn, Jan 2010)

Ans: Refer section 1.8.

5. Explain the Organization of a computer and describe
(i) Input unit. (ii) Central Processing unit and (iii) Output unit. (AU, Chn, Jan 2011)

Ans:

Organization of a Computer Refer Section 1.8

(i) Input Unit Refer Section 1.8.1

(ii) Central Processing Unit Refer Section 1.8.3

(iii) Output Unit Refer Section 1.8.4

6. Describe various types of memories used in computer. (AU, Chn, Jan 2011)

Ans: Refer Section 1.8.2

7. Explain about binary and BCD number system. Also convert decimal no.356 to binary form. (AU, Mdu, Jan 2011)

Ans:

Binary Number System

The binary number system uses base 2 to represent different values. Therefore, the binary system is also known as base-2 system. As this system uses base 2, only two symbols (0 and 1) are available for representing the different values in the system.

In the binary system, the weight of any bit can be determined by raising 2 to the power equivalent to the position of bit in the number. The decimal value of a given binary number can be determined as the sum of the products of the bits multiplied by the weight of the bit itself.

For example, consider the binary number 1001001.

Now, calculate the sum of the products of each bit multiplied by its place value as shown below:

$(1 \times 2^6) + (0 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$

$= 64 + 0 + 0 + 8 + 0 + 0 + 1 = 73$

Therefore, $(1001001)_2 = (73)_{10}$

BCD System

The BCD system is employed by computer systems to encode the decimal number into its equivalent binary number. This is generally accomplished by encoding each digit of the decimal number into its equivalent binary sequence. In the 4-bit BCD system, each digit of the decimal number is encoded to its corresponding 4-bit binary sequence. The two most popular 4-bit BCD systems are:

- **Weighted 4-bit BCD code:** In this code, each decimal digit is encoded into its 4-bit binary number in which the bits from left to right have the weights 8, 4, 2, and 1 respectively. For example the weighted 4-bit BCD codes for decimal digits 2 and 7 are 0010 and 0111 respectively.
- **Excess – 3 (XS – 3) BCD code:** It transforms the decimal number in to the 4-bit BCD code by first adding 3 to all the digits of the number and then converting the excess digits, so obtained, into their corresponding 8421 BCD code. For example, the XS-3 BCD codes for decimal digits 5 and 9 are 1000 and 1100 respectively.

8-Bit BCD Systems

The 8-bit BCD systems were developed to overcome the limitations of 6-bit BCD systems. The 6-bit BCD systems can handle numeric as well as non-numeric data but with few special characters. The 8-bit BCD systems can handle numeric as well as nonnumeric data with almost all the special characters such as +, - , *, /, @, $, etc. Therefore, the various codes under the category of 8-bit BCD systems are also known as alphanumeric codes. The three most popular 8-bit BCD codes are:

- Extended Binary Coded Decimal Interchange Code (EBCDIC)
- American Standard Code for Information Interchange (ASCII)
- Gray code

Conversion from Decimal to Binary

The given decimal number is 356.

The following table lists the steps showing the conversion of decimal number (356) to its binary equivalent:

Decimal number	Divisor	Quotient	Remainder
356	2	178	0
178	2	89	0
89	2	44	1
44	2	22	0
22	2	11	0
11	2	5	1
5	2	2	1
2	2	1	0
1	2	0	1

Now, read the remainders calculated in the above table in upward direction to obtain the binary equivalent of 356, which is 101100100.

Therefore, $(356)_{10} = (101100100)_2$

8. What is the significance of Unicode? (AU, Mdu, Jan 2011)

Ans:

Significance of Unicode

Many 7-bit and 8-bit character codes are developed to encode only the well- known characters such as letters a –z in lower as well as upper case, digits 0 – 9 and some special symbols. These character sets can represent a maximum of 256 different characters only. These 7-bit and 8-bit character codes are confined to represent the characters used by the popular languages around the globe. Therefore, there was always a need of a character code that could contain all the characters used by almost all the languages in the world.

The 16-bit Unicode is an international 16-bit character set that contains a maximum of 2^{16} = 65,536 different characters. These characters are sufficient to represent almost all the technical and special symbols used by the major languages of the world. The 16-bit Unicode also called 16-bit universal character set encodes the different characters by assigning them a unique value. In computer terminology, this unique value is referred as code point. The code assigned to each character of different languages is universal and can be used on any platform without any modification. Therefore, we can say that the 16-bit Unicode allows the computer systems to deal with almost all the characters belonging to different languages used in the world.

9. Explain about various input and output devices. (AU, Mdu, Jan 2011)

Ans:

Input Devices: Refer Section 1.8.1

Output Devices: Refer Section 1.8.4

10. With the help of a Block diagram, explain the computer organization in detail. (AU, Cbe, Dec 10-Jan 11)

Ans: Refer Section 1.8

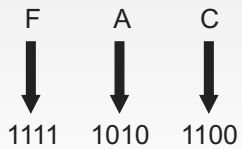
11. Convert the following numerals into their Binary equivalents: FAC_{16}, 561_8. (AU, Cbe, Dec 10-Jan 11)

Ans:

FAC_{16}

The given hexadecimal number is FAC.

The 4-digit binary equivalent of each of the digits of the given number is shown below.



Therefore, $(FAC)_{16} = (111110101100)_2$

561_8

The given octal number is 561.

The 3-digit binary equivalent of each of the digits of the given number is shown below.

Therefore, $(561)_8 = (101110001)_2$

12. With the help of a neat sketch explain the computer organization in detail.

(AU, TRI, Jan 2011)

Ans: Refer Section 1.8.

13. Explain in detail about the classification of computer with examples.

(AU, TRI, Jan 2011)

Ans: Refer Section 1.7.

14. Briefly explain the components of a digital computer. (AU, TIR, Dec 10-Jan 11)

Ans:

Components of a Digital Computer

A computer system comprises of hardware and software components. Hardware refers to the physical parts of the computer system and software is the set of instructions or programs that are necessary for the functioning of a computer to perform certain tasks. Hardware includes the following components:

- **Input devices** They are used for accepting the data on which the operations are to be performed. The examples of input devices are keyboard, mouse and track ball.
- **Processor** Also known as CPU, it is used to perform the calculations and information processing on the data that is entered through the input device.
- **Output devices** They are used for providing the output of a program that is obtained after performing the operations specified in a program. The examples of output devices are monitor and printer.
- **Memory** It is used for storing the input data as well as the output of a program that is obtained after performing the operations specified in a program. Memory can be primary memory as well as secondary memory. Primary memory includes Random Access Memory (RAM) and secondary memory includes hard disks and floppy disks.

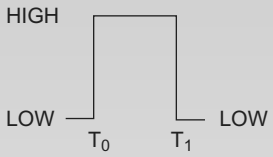
Software supports the functioning of a computer system internally and cannot be seen. It is stored on secondary memory and can be an **application software** as well as **system software**. The application software is used to perform a specific task according to requirements and the system software is mandatory for running application software. The examples of application software include Excel and MS Word and the examples of system software include operating system and networking system.

15. Discuss about digital waveforms in detail. (AU, TIR, Dec 10-Jan 11)

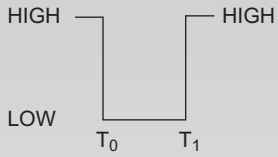
Ans:

Digital Waveforms

Digital waveform comprises of discrete voltage levels (HIGH and LOW) that keep on changing with time. It is different than the analog waveform that comprises of continuous time varying signals. In a digital waveform, a positive pulse is obtained when voltage changes from LOW level to HIGH level and then back to LOW level. Similarly, a negative pulse is obtained when a voltage changes from HIGH level to LOW level and then back to HIGH level. This is shown in figure below.

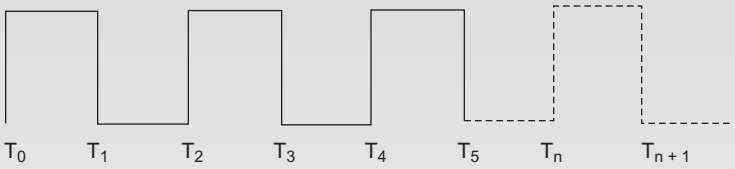


Positive Pulse



Negative Pulse

Here, T_0 and T_1 are the time instants. A complete digital waveform comprises of a series of such positive and negative pulses, as shown below:



Digital Waveform

If the digital pulse waveform repeats itself after certain fixed intervals of time then it is referred as periodic pulse waveform. The frequency of such a waveform is measured by the rate at which it repeats itself. For instance, consider the above waveform:

Here, Time interval = $T_0 = T_1 = T_2 = \ldots = T_n$

Hence, Frequency = $F = 1/T$

16. Describe the classification of integrated circuits in detail. (AU, TIR, Dec 10-Jan 11)

Ans:

Classification of Integrated Circuits

Integrated circuit (IC) is a collection of large number of electronic components, such as transistors, resistors, capacitors, etc. designed and constructed into a single chip to realize a specific functionality. Depending on the density of integration, digital integrated circuits are classified as:

- **Small Scale Integration (SSI) circuit**: Contains less than 12 components
- **Medium Scale Integration (MSI) circuit**: Contains 12 to 99 components

- **Large Scale Integration (LSI) circuit**: Contains 100 to 9999 components
- **Very Large Scale Integration (VLSI) circuit**: Contains 10000 to 99999 components
- **Ultra Large Scale Integration (ULSI) circuit**: Contains 100000 or more components

17. Briefly explain any four logic operations each with an example. (AU, TIR, Dec 10-Jan 11)

Ans:

Logic Operations

The various logic operations are:

- AND
- OR
- NOT
- XOR

AND Operation

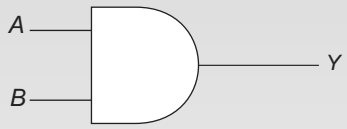
The AND operation gives the output as true if and only if both the inputs are true.

It is logically expressed as:

$Y = A.B$

Here, A and B are the two inputs.

The AND operation is implemented with the help of a logic circuit called AND gate. The following figure shows the circuit symbol of the AND gate.



The circuit symbol of AND gate.

The following table shows the truth table of AND gate.

Input A	Input B	Output Y(A.B)
0	0	0
0	1	0
1	0	0
1	1	1

OR Operation

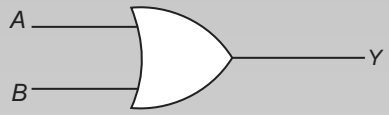
The OR operation gives the output as true if any one of the inputs is true.

It is logically expressed as:

$Y = A + B$

Here, A and B are the two inputs.

The OR operation is implemented with the help of a logic circuit called OR gate. The following figure shows the circuit symbol of the OR gate.



The circuit symbol of OR gate.

The following table shows the truth table of OR gate.

Input A	Input B	Output Y(A+B)
0	0	0
0	1	1
1	0	1
1	1	1

NOT Operation

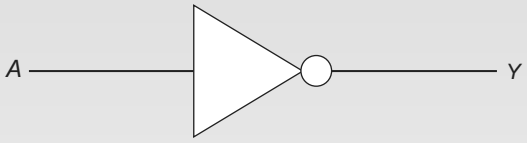
The NOT operation also called invert operation inverts the value of the input for producing the output. For example, if 1 is provided as an input then the output will be 0 and vice-versa.

It is logically expressed as:

$Y = \overline{A}$

Here, A is the input.

The NOT operation is implemented with the help of a logic circuit called NOT gate. The following figure shows the circuit symbol of the NOT gate.



The circuit symbol of NOT gate.

The following table shows the truth table of NOT gate.

Input A	Output Y ($\overline{A}$)
0	1
1	0

XOR Gate

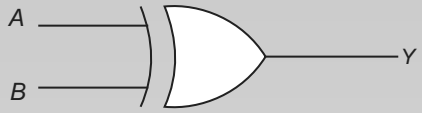
The XOR operation also called Exclusive-OR operation gives the output as true if any one of the inputs is true. However, if both the inputs are same, the output is false.

It is logically expressed as:

$Y = A \oplus B$

Here, A and B are the two inputs.

The XOR operation is implemented with the help of a logic circuit called XOR gate. It is a derived logic gate that is implemented by a combination of two or more basic logic gates. The following figure shows the circuit symbol of the XOR gate.



The circuit symbol of XOR gate.

The following table shows the truth table of XOR gate.

Input A	Input B	Output Y (A ⊕ B)
0	0	0
0	1	1
1	0	1
1	1	0

17. What are the advantages and disadvantages of 1's complement and 2's complement number systems? (AU, TIR, Dec 10-Jan 11)

Ans:

1's Compliment Number System

Advantages

- Symmetricity is one of key advantages of 1's compliment systems.
- It is easier to compute the 1's compliment of a number.

Disadvantages

- It has two different representations for Zero, -0 and +0.
- The design of adder circuits is more complex in one's complement system as compared to 2's compliment system.

2's Compliment Number System

Advantages

- The operations performed in 2's compliment system are non-dependent on the signs of the operands.
- Since the signs of the operands are not required to be explicitly considered the operations in 2's compliment system are relatively simpler.
- Unlike 1's compliment system, it has a single representation for Zero.

Disadvantages

- It is comparatively difficult to convert from a positive number to a negative number and vice versa.

- The range of numbers that can be represented in 2's compliment system is asymmetric. The negative numbers are one more than the range of positive numbers.

18. Compute $1010100_2 + 1000100_2$ using 1's complement and 2's complement. (AU, TIR, Dec 10-Jan 11)

Ans:

10101002 + 10001002 using 1's Compliment

```
  1 0 1 0 1 0 0
+ 1 0 0 0 1 0 0
---------------
 1 0 0 1 1 0 0 0
```

Adding the generated carry, we get:

```
  0 0 1 1 0 0 0
            + 1
---------------
  0 0 1 1 0 0 1
```

Hence, $1010100_2 + 1000100_2 = 0011001_2$

10101002 + 10001002 using 2's Compliment

```
  1 0 1 0 1 0 0
+ 1 0 0 0 1 0 0
---------------
 1 0 0 1 1 0 0 0
```

Discarding the carry to obtain the final answer, which is 0011000.

Hence, $1010100_2 + 1000100_2 = 0011000_2$

19. Explain Hexadecimal and BCD number system in detail. (AU, TIR, Dec 10-Jan 11)

Ans:

Hexadecimal Number System Refer Section 1.12

BCD Number System Refer Section 1.14

20. Explain various digital codes in detail. (AU, TIR, Dec 10-Jan 11)

Ans: Refer Sections 1.14, 1.15, and 1.16

21. Convert $2AC5.D_{16}$ to Decimal, Octal and Binary. (AU, TIR, Dec 10-Jan 11)

$2AC5.D_{16}$ to Decimal

The given hexadecimal number is 2AC5.D.

Now, calculate the sum of the products of each digit multiplied by its place value as shown below:

$(2 \times 16^3) + (10 \times 16^2) + (12 \times 16^1) + (5 \times 16^0) + (13 \times 16^{-1})$

$= 8192 + 2560 + 192 + 5 + 0.8125$

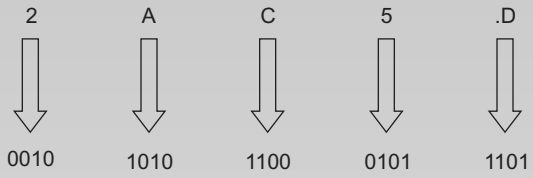
$= 10949.8125$

Therefore, $(2AC5.D)_{16} = (10949.8125)_{10}$

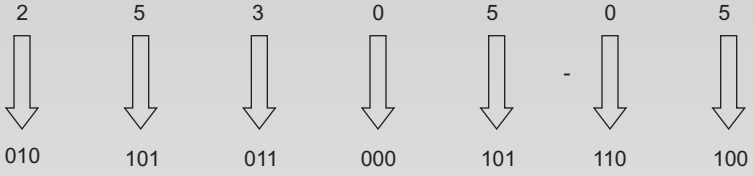
$2AC5.D_{16}$ to Octal

The given hexadecimal number is 2AC5.D.

The binary equivalent of each of the digits is shown below.



Now, let us group the resultant binary values into groups of three; and find their equivalent octal values, as shown below:

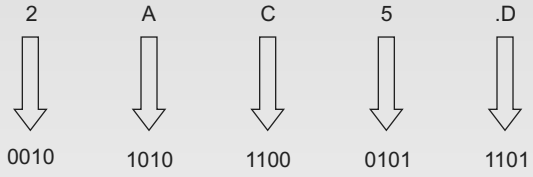


Thus, $(2AC5.D)_{16} = (25305.64)_8$

$2AC5.D_{16}$ to Binary

The given hexadecimal number is 2AC5.D.

The binary equivalent of each of the digits is shown below.



Therefore, $(2AC5.D)_{16} = (10101011000101.1101)_2$

22. Explain different types of ROM in detail. (AU, TIR, Dec 10-Jan 11)

Ans:

Types of ROM

ROM is a type of memory that stores the data permanently. Data can be easily read from this type of memory but it cannot be changed. The various types of ROM are:

- Programmable ROM (PROM)
- Erasable PROM (EPROM)
- Electrically Erasable PROM (EEPROM)
- Flash ROM

Programmable ROM

Programmable ROM (PROM) is a memory chip on which the write operation of data can be performed only once. The data is stored on this chip permanently, i.e., once a program is written on the PROM, it cannot be erased or destroyed. To write the data on the PROM chip, a device known as PROM programmer or PROM burner is used. PROM is reliable and stores the data permanently without allowing any changes to be made in it. It is mostly used in video games and electronic dictionaries.

Erasable PROM

Erasable PROM (EPROM) is a type of ROM in which data can be erased or destroyed using Ultraviolet Light (UL). Erasable ROM allows its contents to be changed, i.e., it can be reprogrammed. It comprises of floating gate transistors, which have the capability to hold an electric charge even when the power of the computer system is switched off. It also facilitates the storage of data for a longer period of time.

Electrically Erasable PROM

Electrically Erasable PROM (EEPROM) is a type of ROM in which data can be erased or destroyed by exposing it to an electric charge. It has the ability to retain the data stored in it even if the power of the computer system is switched off. It stores the data permanently but allows us to make changes in the data by erasing it with the help of electric charge. In this type of memory, the data can be written or erased only one byte at a time because of which it works very slow.

Flash ROM

Flash ROM is a type of EEPROM that stores the information using floating-gate transistors. These transistors store electric charge for a longer period of time as compared to the normal transistors. This type of ROM is mainly used in the memory cards of mobile phones, digital cameras and ipods. The data stored in flash ROM memory can be easily transferred using transmission mediums such as data cable, bluetooth and infrared technology. For example, we can transfer the data stored in flash ROM memory of mobile phone to the memory of a computer using data cable. We can easily erase the data stored in flash ROM memory and reprogram it. Flash ROM has faster speed of reading the data as compared to any other type of ROM. It uses continuous memory cells for data storage.

23. What are the needs for storage devices? Describe various storage devices in detail.

(AU, TIR, Dec 10-Jan 11)

Ans: Refer section 1.8.2.

COMPUTER SOFTWARE

2

CHAPTER OBJECTIVES

In this chapter, we will learn:

1. The fundamentals of computer software.
2. Various types of computer software.
3. Role of system software.
4. Various types of system software and their functions.
5. Functions of an operating system.
6. Different types of application software.
7. Various software development steps like analysis, design, development, testing etc.
8. How the Internet has evolved over the years.
9. The key Internet terminologies.
10. The basic steps of getting connected to Internet applications.

CHAPTER OUTLINE

2.1 INTRODUCTION

All computer systems consist two major components, namely hardware and software. Hardware refers to the physical components that are necessary for performing various operations such as reading and processing data, storing results and providing output to the users in a desired form. The software, on the other hand, refers to a set of computer programs that are required to enable the hardware to work and perform these operations effectively. In this chapter, we will learn the two main types of computer software: system software and application software. These software are further categorised into various other sub-types based on the functionality performed.

Later in this chapter, we will introduce ourselves to the all-popular buzzword that is Internet. Almost everyone working in either government offices or in business organisations is using the Internet to exchange information in one form or the other. The Internet is a massive network of networks that links together thousands of independent networks, thus bringing millions of computers on a single network to provide a global communication system. It acts as a facilitator for exchanging information among the computers that are connected to the Internet. It is like a network of roads in a country that facilitates the movement of vehicles around the country. In this chapter, we will discuss the evolution of the Internet and the basic Internet terminologies. We will also discuss the basic steps for accessing Internet applications.

2.2 OVERVIEW OF COMPUTER SOFTWARE

A computer software is basically a set of logical instructions, written in a computer programming language that tells the computer how to accomplish a specific task. A software is, therefore, an essential interface between the hardware and the user as shown in Fig. 2.1.

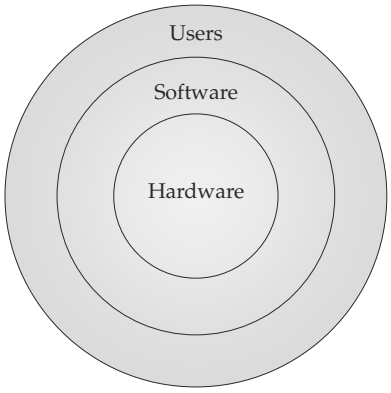


Fig. 2.1 Software interface between hardware and users

We can say that software gives life to the hardware and, therefore, the software is popularly referred as the "soul" of the computer system, while the hardware is referred as the "heart". There is an array of computer software available for operating and controlling the computer.

2.3 TYPES OF COMPUTER SOFTWARE

As stated earlier, computer software performs two distinctive tasks. The first task is to control and coordinate the hardware components and manage their performances, while the second one is to enable the users to accomplish their required tasks. The software that is used to complete the first task is known as the system software and the software that is used to accomplish the second task is known as the application software. While the system software is essential for a computer to work, the application software is the additional software required for the user to perform a specific job. The system software not only controls the hardware functions but also enables the hardware to interact with the application software as well as the users, as illustrated in Fig. 2.2.

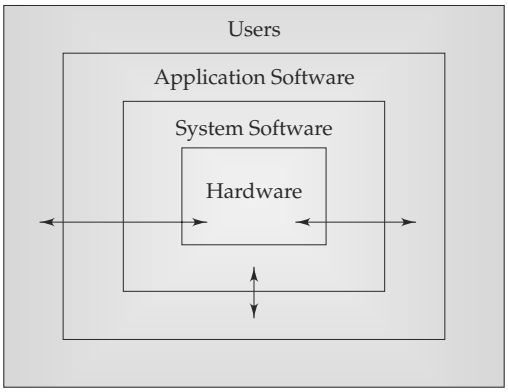


Fig. 2.2 Layers of software and their interactions

2.3.1 System Software

System software has many different programs that manage and support different tasks. Depending upon the task performed, the system software can be classified into two major groups:

- *System management programs* used for managing both the hardware and software systems.
- *System development programs* used for developing and executing application software.

2.3.2 Application Software

Application software includes a variety of programs that are designed to meet the information processing needs of the end users. They can be broadly classified into two groups:

- *Standard application programs* that are designed for performing common application jobs.
- *Unique application programs* that are developed by the users by themselves to support their specific needs.

Figure 2.3 illustrates the major categories of computer software.

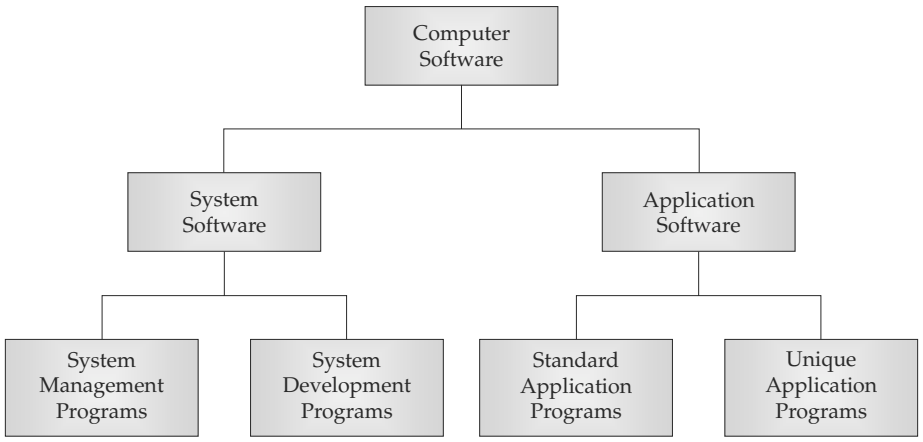


Fig. 2.3 Major categories of computer software

2.4 SYSTEM MANAGEMENT PROGRAMS

System management programs are those programs that are meant for operating the hardware system and managing their resources effectively. They also enable the users to perform certain utility functions, such as creating backup files, recovering damaged files and merging files. They minimise the human intervention during processing and aid in maximising the productivity of a computer system. System management programs include:

- Operating system
- Utility programs
- Device drivers

These programs work in close interaction with each other as shown in Fig. 2.4.

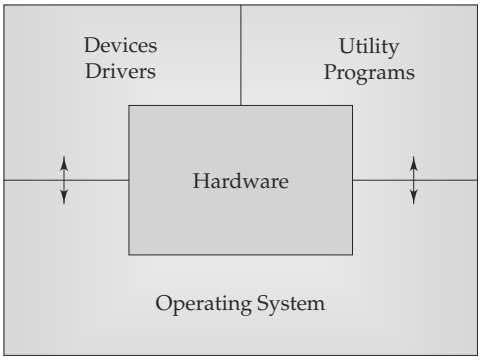


Fig. 2.4 **System management programs**

2.4.1 Operating System

Operating System (OS) is the principal component of system software and is responsible for overall management of computer resources. It also provides an interface between the computer and the user and helps in implementing the application programs, as illustrated in Fig. 2.5.

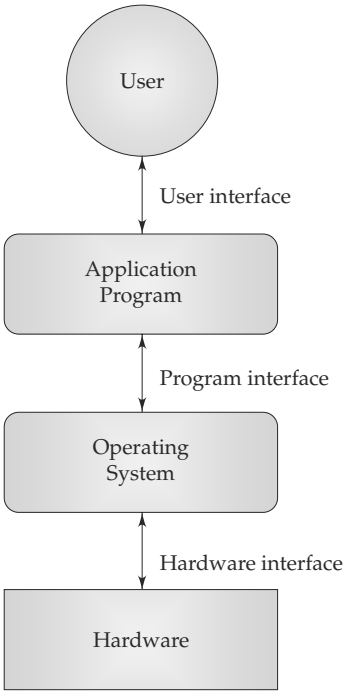


Fig. 2.5 **Hardware-OS-User interface**

Major functions of an operating system are:

- Scheduling and execution of all processes.
- Allocation and management of main memory and other storage areas to the programs.
- Coordination and assignment of different hardware devices to the programs.
- Creating, storing and manipulating files required by the various processes.
- Determining and maintaining the order of execution of programs.
- Interpretation of commands and instructions.
- Coordination and assignment of other development and utility programs.
- Providing a friendly interface between the computer and the user.
- Ensuring security of access to computer resources.

Operating systems are usually supplied by the hardware manufacturers and are rarely developed by the users due to its technical complexity.

2.4.2 Utility Programs

Utility programs refer to small programs, which provide additional capabilities to the computer system in addition to the ones provided by the operating system. They enable an operating system to perform some additional tasks, such as searching and printing the files, scanning the viruses, etc. A utility program is not an essential part of an operating system, because it does not help the operating system in the execution of a command or a program. A utility program only provides the additional features to the computer system. In other words, an operating system can execute most of the programs without having the utility programs. Utility programs are added to an operating system to perform many different tasks, which include:

- **Search** It enables the operating system to search a file on the basis of the specified search criteria.
- **Print** It enables an operating system to initiate the print operation of the printer connected with the computer system.
- **Disk defragmenter** It helps defragmenting the memory space. Defragmentation is the process of storing the data at a single place in the memory instead of disjointed memory locations.
- **System profiler** It provides the information related to the various hardware and software components installed in the computer system. In other words, it provides a list of the hardware components, which are presently connected with the computer system, and the software components, which are currently installed in the computer system.
- **Encryption** It enables the operating system to generate the encrypted format of the messages or the files, which have to be transmitted from one system to another system over a network or the Internet.
- **Virus scanner** It enables the operating system to detect viruses and bugs, which may affect the efficient functioning of the computer system.

- **Backup** It enables the creation of a copy of various files and folders on a secondary medium such as magnetic disk and magnetic tape, in order to keep the original data safe.
- **Data recovery** It enables the retrieval of lost data from a corrupted or damaged primary storage medium.

Like operating systems, utility programs are prewritten by manufacturers and supplied with the hardware. They can also be obtained from standard software vendors. A good range of utility programs can make the life much easier for the user.

2.4.3 Device Drivers

A computer system is connected with multiple input and output (I/O) devices so that it can communicate with the end user. In order to interact with the I/O devices, the computer system requires special software called device driver. The device driver acts as a translator between the I/O devices and the computer.

A device driver of the input device interprets the input provided by the user into the computer understandable form and directs it to the operating system. Similarly, the device drivers of the output devices translate the output generated by the computer into the user understandable format and display it on the screen. In other words, a device driver is a special software that enables a hardware device, such as keyboard, monitor and printer, to perform an operation according to the command given by the end user. Whenever a program needs to use a connected hardware device, it issues a command to the operating system. The operating system calls the respective routine or process of the device driver, which instructs the device to perform the task. For providing an instruction to the devices, the device driver sends various control and data signals through buses and cables to the device. In order to acknowledge the device driver's signal, the device sends the acknowledgement signals to the device driver. After performing the task, the device sends a message to the device driver, which returns the value to the calling program of the operating system. In other words, a device driver instructs a hardware device the way it should accept the input from the operating system and the way in which it should communicate with the other units of the computer system. Figure 2.6 illustrates the working of the device driver.

2.5 SYSTEM DEVELOPMENT PROGRAMS

System development programs known as programming software allow the users to develop programs in different programming languages. The process of developing and executing a program involves the following steps:

- Debugging the program
- Linking various variables and objects with the library files
- Translating the code from one language to another
- Running the machine code to perform the desired task

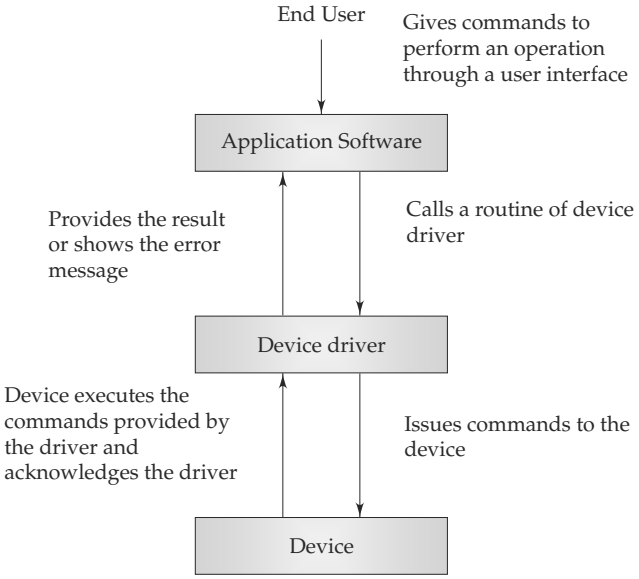


Fig. 2.6 Working of a device driver

In order to carry out these tasks, we need the following system development tools:

- Language translators
- Linkers
- Debuggers
- Editors

2.5.1 Language Translators

Language translator is used to convert the program code written in one language to another language. The program code provided as an input to the language translator is known as source code. The source code is a high-level or an assembly language program. The language translator converts the high-level language program into the low-level language program called object code. Compiler, interpreter and assembler are the most common examples of language translator. A compiler translates a high-level program into a low-level program, and an assembler translates an assembly language program into a low-level program. An interpreter also produces a low-level program from a high-level program, but the working of the interpreter is not similar to that of the compiler. An interpreter processes the high-level program line-by-line and simultaneously produces the low-level program. On the other hand, a compiler compiles the high-level program in one go. Figure 2.7 shows the input and the output of the various language translators.

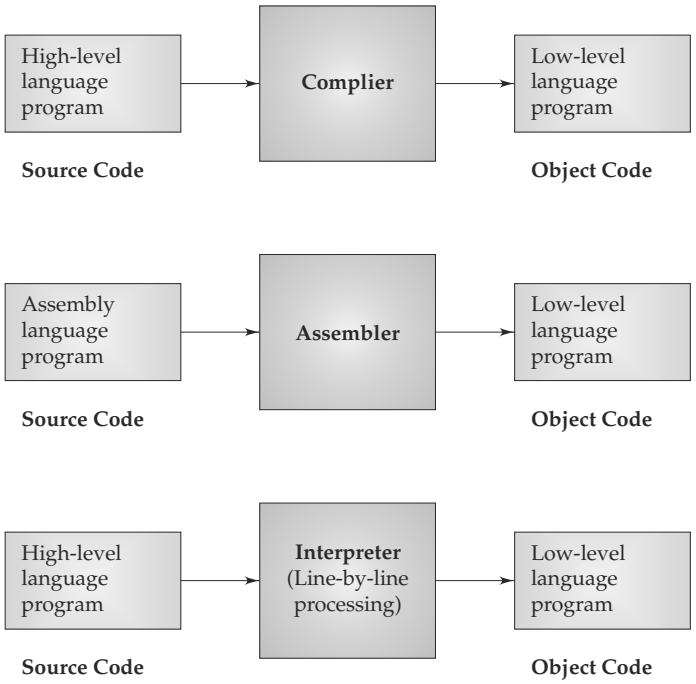


Fig. 2.7 **Language translators**

2.5.2 Linkers

Most of the high-level languages allow the developer to develop a large program containing multiple modules. Linker arranges the object code of all the modules that have been generated by the language translator into a single program. The execution unit of the computer system is incapable of linking all the modules at the execution time and therefore, linker is regarded as one of the important software because of its ability to combine all the modules into a single program. Linker assembles the various objects generated by the compiler in such a manner that all the objects are accepted as a single program during execution. Linker also includes the links of various objects, which are defined in the runtime libraries. In many cases, linker inserts the symbolic address of the objects in place of their real address. Figure 2.8 illustrates the working of a linker.

2.5.3 Debuggers

Debugger is the software that is used to detect the errors and bugs present in the programs. The debugger locates the position of the errors in the program code with the help of what is known as the Instruction Set Simulator (ISS) technique. ISS is capable of stopping the execution of a program at the point where an erroneous statement is encountered.

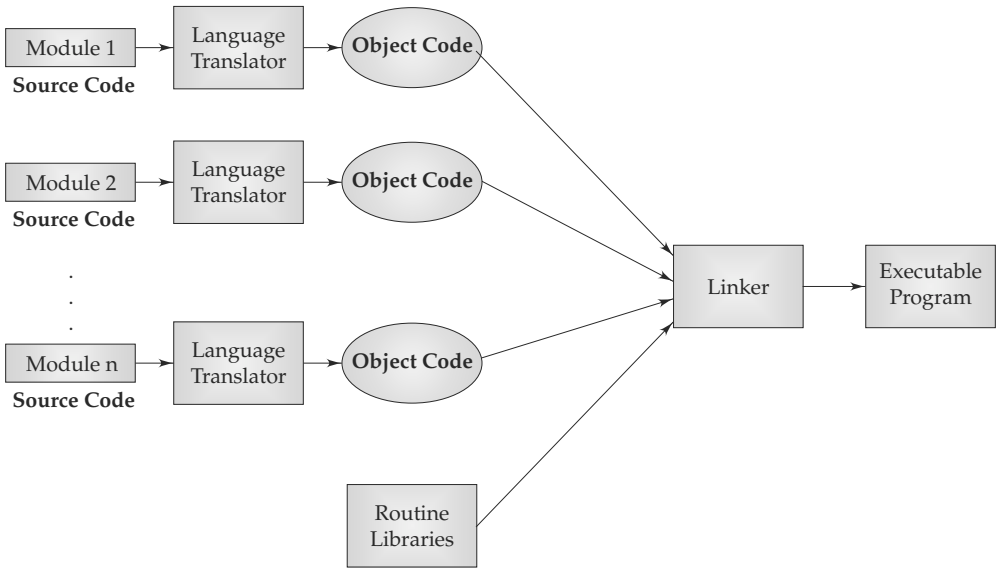


Fig. 2.8 Working of a linker

Debugger is divided into two types, namely machine-level debugger and symbolic debugger. The machine-level debugger debugs the object code of the program and shows all the lines where bugs are detected. On the other hand, the symbolic debugger debugs the original code, i.e., the high-level language code of the program. It shows the position of the bug in the original code of the program developed by the programmer.

While debugging a program, the debugger performs a number of functions other than debugging, such as inserting breakpoints in the original code, tracking the value of specific variables, etc. In order to debug the program, a debugger helps perform the following tasks:

- Step-by-step execution of a program
- Back tracking for checking the previous steps
- Stopping the execution of the program until the errors are corrected

2.5.4 Editors

Editor is a special program that allows the user to work with text in a computer system. It is used for the documentation purposes and enables us to edit the information present in an existing document or a file. The editor enables us to perform various editing operations such as copy, cut and paste while editing the text. On the basis of the content edited by the editors, they are divided into the following categories:

- **Text editor** It is used to edit plain text. An operating system always includes a text editor for updating the configuration files.

- **Digital audio editor** It is used to edit the information related to the audio components of a multimedia application. These editors are used in audio applications where editing the music and the sound signals is necessary.
- **Graphics editor** It is used to edit the information related to the graphical objects. These editors are generally used in the multimedia applications where the user is working with multiple animation objects.
- **Binary file editor** It is used to edit the digital data or the binary data, i.e., data having strings of 0s and 1s.
- **HTML editor** It is used to edit the information included in the Web pages.
- **Source code editor** It is used to edit the source code of a program written in a programming language such as C, C++ and Java.

2.6 STANDARD APPLICATION PROGRAMS

Standard application programs, also known as general-purpose application programs, are programs that perform certain common information processing tasks for the users. For example, preparing documents such as letters and notes are common among almost all organisations as well as individuals. Similarly, creating and managing databases of products and employees are very common activities in large organisations.

Ready-to-use programs for such applications are available from hardware or software vendors. These programs not only significantly increase the productivity of users but also decrease the investment in terms of time and cost. Examples of standard application programs include:

- Word processor
- Spreadsheet
- Database Manager
- Desktop Publisher
- Web Browser

We will briefly discuss some of these programs in this section.

2.6.1 Word Processor

Word processor is an application software that is generally used to create documents such as letters, reports, etc. It is one of the most commonly used computer applications. Word processor enables us to enter text in a document in a particular format. It also allows us to modify the existing text and apply the formatting features, such as boldface and italic, change the colour of the text, change the writing style of the text, etc. The word processor application software also allows us to perform different operations, such as cut-paste for moving the text within the document, copy-paste for copying the text within a document or from one document to another document, etc. MS Word is the most common example of a word processor. Figure 2.9 shows the default window of MS Word.

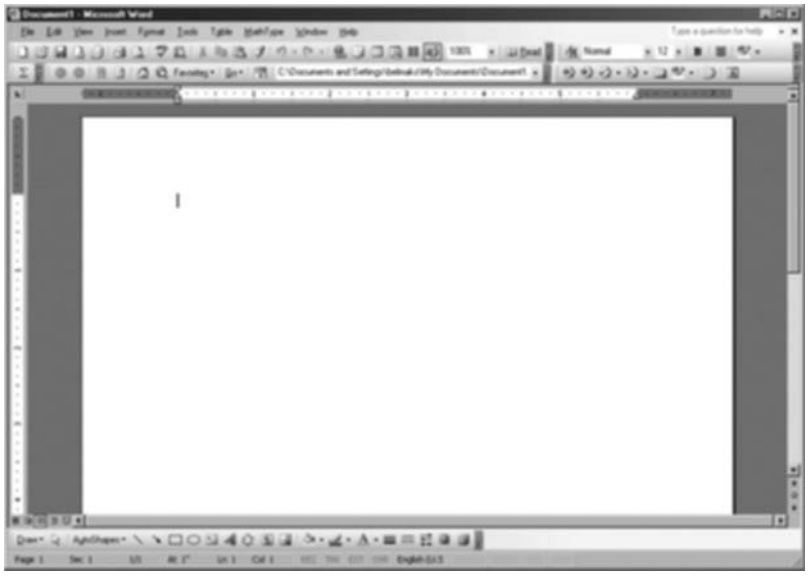


Fig. 2.9 **The default window of MS Word**

2.6.2 Spreadsheet

Spreadsheet is an application software that enables the user to maintain a worksheet in which information is stored in different cells. Each cell in the spreadsheet can contain numeric and alphanumeric data. Spreadsheet is also used as a computational tool by applying formulas on the values of cells. If we change the value of a cell in a spreadsheet, then all the values, which are dependent on that value, get automatically updated according to the new value. As a result, the users, who need to store and maintain financial information, always prefer a spreadsheet for storing the information. MS Excel is the most common example of the spreadsheet software. Figure 2.10 shows the default window of MS Excel.

2.6.3 Database Management System

Database Management System (DBMS) is a computer software that helps us to store and maintain records in a database. Database refers to a set of records stored in a structured manner in the computer system. Record refers to a set of similar or dissimilar values related to an entity such as student and employee. DBMS allows a user to perform various operations such as search, update and delete on the data stored in the database. With the help of DBMS, an end user can store, retrieve and modify the data in an easy manner. The advantages of DBMS are as follows:

- It enables the user to organise all the information in a strategic manner.
- It helps the users in maintaining consistency in the information stored in the database.

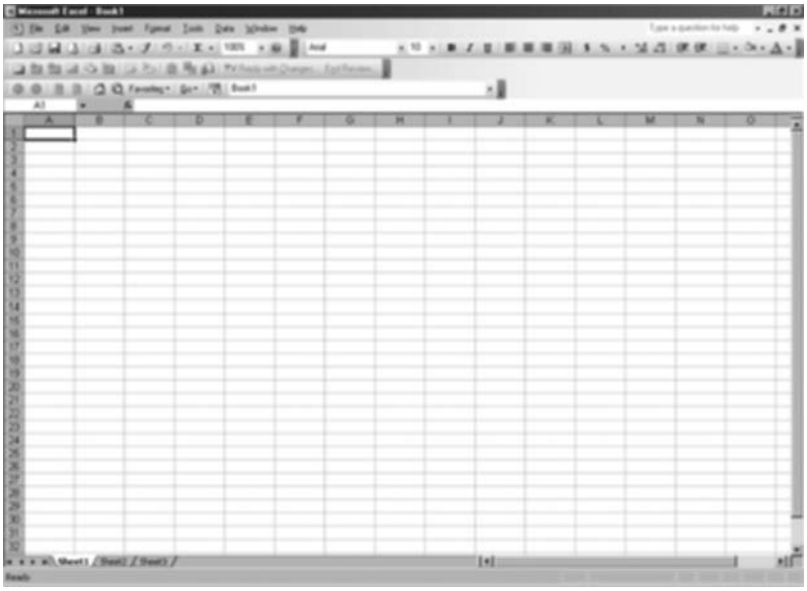


Fig. 2.10 **The default window of MS Excel**

- It helps the user in maintaining data integrity while performing the various operations on the database values.
- It allows the user to maintain the security of the confidential information by protecting the database through password.
- It helps in accessing information from any system connected in a networked system environment.

The DBMS used for maintaining a database always uses a specific data model, such as relational, hierarchical and network, that describes the physical and logical arrangement of data in the database.

In the relational data model, different records are arranged in the form of tables containing rows and columns. In the hierarchical model, records are arranged in the form of a tree where each record is attached with one parent node and one or more child nodes. In the network model, data are arranged in the form of a network where each node is connected with one or more parent nodes as well as child nodes. Figure 2.11 shows the arrangement of records in different data models of the DBMS.

DBMS comprises different components, which are responsible for performing a specific task in the database system. Some of the components of DBMS are as follows:

- **Data structure** Data structure refers to a storage mechanism that is used to store multiple values as a single entity. In DBMS, different types of data structures are used to store the information related to an entity. These data structures include:
  - **File** It refers to a collection of multiple records.

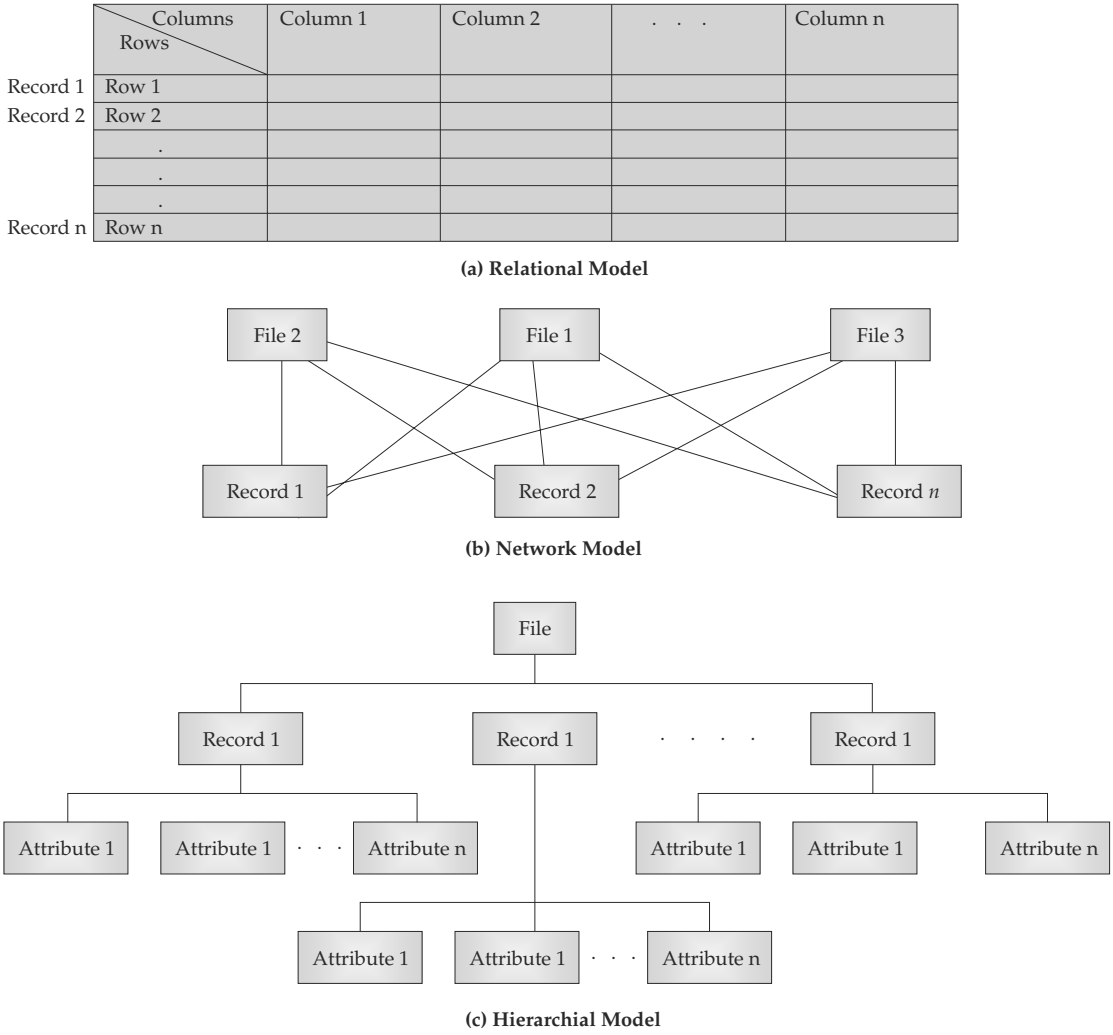


Fig. 2.11 **Different data models used to design DBMS**

- **Record** It refers to a collection of multiple values of different data types related to one entity.
- **Field** It refers to a set of different values having similar data type.

- **Transaction mechanism** DBMS always includes a transaction mechanism, which ensures that the basic properties of the database system are not affected by the processing of the transactions. A transaction mechanism included in a database system consists of the following properties:
  - **Atomicity** The transaction mechanism should complete either all tasks included in a transaction processing or none of them.

  - **Consistency** The transaction mechanism should be capable of maintaining consistency in the information stored in the multiple files of DBMS after the processing of a transaction.
  - **Isolation** The transaction mechanism should not allow any sub-process of the transaction to use the intermediate data of another transaction.
  - **Durability** The transaction mechanism should be capable of providing information related to the transactions, which have been processed to change the information stored in the database.
- **Database query language** DBMS includes a query language system, which enables a user to perform various operations such as searching and updating the information stored in the database.

2.6.4 Desktop Publishing System

Desktop publishing system enables the user to perform various activities required for publishing a page or set of pages. These activities include organising the content layout by setting margins and justification properties of the page. In the desktop publishing system, a page layout software is installed on a computer system that enables the end users to view the changes done at the time of designing the document. The page layout software uses the 'What You See Is What You Get (WYSIWYG)' technique for displaying the preview of the page being designed. Desktop publishing system is also referred as DTP. This system is generally used in the publishing industry, where there is a need to publish information either through print media or display media. It helps the users to design the desired page layout containing different components such as text, graphics and images. Some of the commonly used DTP application software include: Quark XPress, InDesign, Microsoft Publisher, and Apple Pages.

2.6.5 Web Browser

Web browser is a software that is used to access the Internet and the WWW. It is basically used to access and view the web pages of the various websites available on the Internet. A web browser provides many advanced features that help achieve easy access to the Internet and WWW.

There are a number of web browsers provided by different software vendors, such as IE from Microsoft, Firefox from Mozilla and Netscape Navigator from Netscape. IE is the most widely used web browser that was developed in 1995 by Microsoft. The first version of IE, i.e., IE 2.0 could be installed and run on the computers with Macintosh and the 32-bit Windows operating systems. IE 2.0 was specially designed to access secure websites, and hence had the capabilities of tracing any kind of errors.

In 1996, the next version, i.e., IE 3.0 was developed, which had many advanced features such as Internet Mail, Windows Address Book and Window Media Player. This version was basically developed for Windows 95 operating system.

In 1997, the next version, IE 4.0 was developed that included Microsoft Outlook Express 4.0, which is an e-mail software used for sending and receiving e-mail messages. Microsoft Outlook Express was included with IE 4.0 to provide enhanced Internet mail and news features.

The latest version of IE is IE 8.0. It is the most secure web browser as it contains many new privacy and safety features as compared to any of the previous versions of IE. The IE 8.0 version is provided with the latest Windows operating system, i.e., Windows Vista.

To access Internet Explorer on a computer, we need to select Start → Programs → Internet Explorer. The Microsoft Internet Explorer window appears with the home page as shown in Figure 2.12.

Fig. 2.12 **Microsoft Internet Explorer window**

2.7 UNIQUE APPLICATION PROGRAMS

There are situations where organisations need to develop their own programs to accomplish certain tasks that are unique to their areas of operations. Similarly, individuals like scientists, engineers, accountants, teachers and other professionals write their own programs to solve their problems. Such programs are known as unique application programs or application-specific programs. These programs are also referred to as end-user application programs. Examples of unique applications include:

- Managing the inventory of a store
- Preparing pay-bills of employees in an organisation
- Processing examination results of students

- Reserving seats in trains or airlines
- Analysing mathematical models
- Computing personal income tax, and many more

These programs are usually developed in one of the high-level languages, such as C, C++ or Java, and therefore, developing such programs in-house would require skilled programmers with deep knowledge not only in programming languages but also in programming environment. We will discuss in this section two applications that are usually developed in-house to suit the user's unique requirements.

2.7.1 Inventory Management System

Inventory management system helps in keeping a record of the huge number of items stored in warehouses and storerooms. In order to keep a record of inventory or items, application software called inventory management system, which makes use of DBMS, is used. DBMS is responsible for storing the information related to the items in the form of various files and tables. It also helps in storing the information related to the location, where a particular item is placed. Inventory management system enables a user to get a report on the items stored in the warehouse or storeroom on the basis of the different conditions, such as availability of items, shipping date of item, etc.

The database of the inventory management system identifies each item by a unique identification number. The identification number is used as the primary key of an item in the database. Primary key refers to an attribute, which must contain a unique value for each item of the database. The identification number may be assigned on the basis of various properties, such as type, model and manufacturing date of the items.

Nowadays, the identification number of an item is specified on the barcodes printed on each item. Barcode is a group of small and black vertical lines of varying widths. The barcodes are not understandable by a human being and is always read by an input device called barcode reader. A barcode reader is similar to the handheld scanner that scans the barcode of the item in order to enter the barcode in the database. Figure 2.13 shows the sample barcode of an item.

Fig. 2.13 **Barcode of an item**

The use of barcode technology with inventory management system saves a lot of time, as the user is not required to manually enter the item details. These details are automatically read by the barcode scanner and stored in the database of the inventory management system. An inventory management system finds its application in those organisations where a constant maintenance and updation of inventory information is required.

2.7.2 Pay-roll System

Pay-roll system is the application software that enables us to perform various activities related to payroll accounting and pay-roll administration process. It is used by the HR department

of an organisation for the purpose of computation of salary of employees. A typical pay-roll processing system provides an interface to the user through which the user can enter salary related details, such as attendance, tax, deduction, etc. A pay-roll system is an effective and efficient way of computing the salary of employees as it eliminates the manual task of performing computations and automates most of the tasks related to pay-roll processing.

A pay-roll system can be custom built or it can be an off-the-shelf application software. Several ERP products include a dedicated module on pay-roll processing which is particularly used in enterprise-wise implementation of pay-roll processing. A pay-roll system can also be used to print or e-mail the salary slips of employees.

2.8 SOFTWARE DEVELOPMENT STEPS

The entire process of software development and implementation involves a series of steps. Each successive step is dependent on the outcome of the previous step. Thus, the team of software designers, developers and operators are required to interact with each other at each stage of software development so as to ensure that the end product is as per the client's requirements. Figure 2.14 shows the various software development steps:

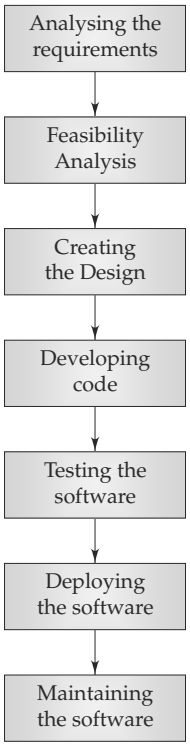


Fig. 2.14 Software development steps

2.8.1 Analysing the Requirements

In this step, the requirements related to the software, which is to be developed, are understood. Analysing the requirements or requirement analysis is an important step in the process of developing a software. If the requirements of the user are not properly understood, then the software is bound to fall short of the end user's expectations. Thus, requirement analysis is always the first step towards development of a software.

Software is abstract in nature; as a result, the users may not be able to provide the complete set of requirements pertaining to the desired software during the requirement analysis stage. Thus, there should be continuous interaction between the software development team and the end users. Moreover, the software development team also needs to take into account the fact that the requirements of the users may keep changing during the development process. Thus, proper analysis of user requirements is quite essential for developing the software within a given timeframe. It will not only help in controlling the software development cost but will also lead to faster and accurate development of a software.

The task of requirement analysis is typically performed by a business analyst. The person is a professional in this field who understands the requirements of the novice end user, and documents and shares it with the development team.

2.8.2 Feasibility Analysis

In this step, the feasibility of developing the software in terms of resources and cost is ascertained. In order to determine the feasibility of software development, the existing system of the user is analysed properly. Apart from studying the existing system, this step involves identifying the need of automation in the existing system. The analysis done in this step is documented in a standard document called feasibility report, which contains the observations and recommendations related to the task of software development. Some of the important activities performed during the feasibility analysis stage are as follows:

- **Determining development alternatives** This activity involves searching for the different alternatives that are available for the development of software. There are mainly four alternatives available for the development of a software. The first alternative is to allow the existing system to continue without developing a new software for automation. The second alternative can be to develop the new software using specific programming languages such as Java, C++, Visual Basic etc. The third alternative is to develop the software using the architectural technologies such as Java 2 Enterprise Edition (J2EE) and mainframe based with thin clients. The fourth development alternative is to buy an already developed software along with its source code from the market and customise it according to the client's requirements.
- **Analysing economic feasibility** This activity involves determining whether the development of a new software will be financially beneficial or not. This type of feasibility analysis is performed to determine the overall profit that can be earned from the development and implementation of the software. This feasibility analysis activity

involves evaluating all the alternatives available for development and selecting the one which is most economical.

- **Assessing technical feasibility** The technical feasibility assessment involves analysing various factors such as performance of the technologies, ease of installation, ease of expansion or reduction in size, interoperability with other technologies, etc. The technical feasibility activity typically involves the study of the nature of technology as to how easily it can be learnt and the level of training required to understand the technology. This type of feasibility assessment greatly helps in selecting the appropriate technologies to be used for developing the software. The selection should be made after evaluating the requirement specification of the software. In addition, the advantages and disadvantages of each identified technology must also be evaluated during technical feasibility assessment.
- **Analysing operational feasibility** Operational feasibility assessment involves studying the software on operational and maintenance fronts. The operational feasibility of any software is done on the basis of several factors, such as:
  - Type of tools needed for operating the software
  - Skill set required for operating the software
  - Documentation and other support required for operating the software

2.8.3 Creating the Design

After the feasibility analysis stage, the next step is creating the architecture and design of the new software. This step involves developing a logical model or basic structure of the new software. For example, if the new software is based on client–server technology then this step would involve determining and specifying the number of tiers to be used in the client–server design. This step also involves documenting the varied specifications pertaining to database and data structure design. The flow of the development process is mainly illustrated in this stage using a special language known as Unified Modelling Language (UML). UML uses pictorial representation methods for depicting the flow of data in the software. Some of the key features, which are considered while designing a software, are:

- **Extensibility** The design of the software should be extensible so that it allows the addition of some new options or modules in future. The architecture of the software should be flexible enough to not get disturbed with the addition of new functionality.
- **Modularity** The software should be modular in nature so that its working and data flow can be understood easily. Modularity also helps in parallel development of the various software modules, which are later integrated into a single software product.
- **Compatibility** Software should run correctly in the existing system with an older version or with other software. Thus, software should be compatible and work well in conjunction with other software.
- **Security** Software must be able to control unauthorised access. While designing a new software, it is ensured that there are proper security mechanisms incorporated in the product.

- **Fault tolerance** The software should be capable of handling exceptions or faults that may occur during its operation. The software must have the capability to recover from failures.
- **Maintainability** The design of the software should be created in a simple manner with appropriate details so that it is easy to maintain.

2.8.4 Developing Code

In this step, the code for the different modules of the new software is developed. The code for the different modules is developed according to the design specifications of each module. The programmers in the software development team use tools like compilers, interpreters and debuggers to perform tasks such as finding errors in the code and converting the code into machine language for its execution. The code can be written using programming languages such as C, C++ or Java. The choice of the programming language to be used for developing the code is made on the basis of the type of software that is to be developed. There are certain key points or conventions, which must be kept in mind while writing code; for instance:

- There should be proper indentation in the code.
- Proper naming conventions should be followed for naming the variables, methods and program files.
- Proper comments should be included to ensure ease of understanding during maintenance.
- The code for different modules of the new software must be simple so that it can be easily understood.
- The code must be logically correct so as to minimise logical errors in the program.

2.8.5 Testing the Software

Testing is basically performed to detect the prevalence of any errors in the new software and rectify those errors. One of the reasons for the occurrence of errors or defects in a new software is that the requirements of the users or client were not properly understood. Another reason for the occurrence of errors is the common mistakes committed by a programmer while developing the code. The two important activities that are performed during testing are verification and validation. Verification is the process of checking the software based on some pre-defined specifications, while validation involves testing the product to ascertain whether it meets the user's requirements. During validation, the tester inputs different values to ascertain whether the software is generating the right output as per the original requirements. The various testing methodologies include:

- Black box testing
- White box testing
- Gray box testing
- Nonfunctional testing

- Unit testing
- Integration testing
- System testing
- Acceptance testing

2.8.6 Deploying the Software

In this step, the newly developed and fully tested software is installed in its target environment. Software documentation is handed over to the users and some initial data are entered in the software to make it operational. The users are also given training on the software's interface and its other functions.

2.8.7 Maintaining the Software

Once the software has been deployed successfully, a continuous support is provided to it for ensuring its full-time availability. A corrupt file, a virus infection and a fatal error are some of the situations where the maintenance personnel are asked to fix the software and bring it back to its normal functioning. Further, a software may also be required to be modified if its environment undergoes a change. In order to successfully maintain the software, it is required that it should have been properly documented at the time of its development. This is because the maintenance person might not be the same who was originally involved in the development of the software. Thus, a good code documentation serves vital for the maintenance person to fix the software.

2.9 EVOLUTION OF INTERNET

The Internet was not a worldwide network initially. It was a small network called ARPAnet, which was developed at the Advanced Research Projects Agency (ARPA) of United States in 1969. ARPAnet was developed to help the researchers at one university to communicate with the researchers at other universities. The computers, which were connected through the ARPAnet, were the computers present at the University of California, Stanford Research Institute (California) and the University of Utah (Nevada).

To transfer data between these computers, the ARPAnet used the concept of packet switching in which the data were divided into small modules known as packets before transmitting. These packets were transmitted individually over the network and were reassembled at the receiver's end.

During the process of transmission of packets on the network, some rules and methods known as protocols were followed. The ARPAnet initially used Network Control Protocol (NCP), which allowed the transmission of files, directories and messages between two computers present on the network. The use of NCP provided end-to-end networking as the

user could only trust the ends, i.e., the sender and the receiver but could not trust the means of transmission, i.e., the network. In 1972, the access to ARPANET was extended and made available to common people and business organisations.

In the year 1974, the scientists developed a new host-to-host protocol, which not only allowed the sharing of files, directories and messages but also helped the users to share software and applications on the network. This protocol was known as Transmission Control Protocol/Internet Protocol (TCP/IP).

TCP/IP was a combination of two protocols, TCP and IP, which worked collectively. The IP was the protocol that guided the packets on the different routes on the network. The IP routed each packet on a different route and the selection of a route for a packet was made in such a manner that the packets should take minimum time to reach their destination. The selection of different routes for different packets also facilitated the transmission of packets even when one of the parts of the network was not working properly. The TCP was the network protocol that was responsible for reassembling the packets at the destination.

The congestion on ARPAnet was increasing continuously because of its extensive use by the military. In 1975, the Defence Communications Agency (DCA) took control of the ARPAnet and changed its name to Defence Advanced Research Projects Agency (DARPA). DARPA allowed all the defence-related organisations to connect to it, but at the same time it denied the connection to all the non-government organisations. This led to the development of other commercial networks such as Telnet.

In 1976, the telephone companies from all over the world broadcasted a new protocol called X.25 with the support of Consultative Committee for International Telegraphy and Telephony (CCITT). The X.25 protocol was similar to the packet-switching technique, but its implementation was different. The X.25 protocol reduced the packet size and provided a more reliable means of transmission of packets. It used the concept of hop-to-hop networking in which the receiving of packet was acknowledged by the hop at every step.

In the X.25 protocol, the information of the path was contained only in the first packet, which helped create a virtual path. The rest of the packets just followed the virtual path created by the first packet. At though this protocol was used for a very short period of time, it was crucial for the development of enterprise networks.

In 1979, a research computer network known as Usenet was developed by a group of computer scientists from all over the world. This network allowed computers to connect through a dial-up connection. In the Usenet network, the UNIX-to-UNIX copy protocol was used to transfer data. Usenet provided two main services, Usenet News and mail servers.

The Usenet News still continues to be used as NetNews, but the mail servers did not prove to be a success. The failure of mail servers was due to the fact that to send a mail through mail servers, a user was required to provide the complete path to the destination computer with the help of the UUCP bang addressing format. The UUCP bang addressing format was the format in which the names of different machines were separated using a bang, i.e., an exclamation mark.

In 1982, a new network known as Eunet was developed in Europe to connect the networks of European countries such as UK, Scandinavia and Netherlands. By the year 1983, a number of networks were added to ARPAnet to connect around 300 computers.

In the same year, TCP/IP was made the standard protocol for ARPAnet. The TCP/IP protocol allowed communication between computers of different networks and the ARPAnet became capable of internetworking. This was the time when ARPAnet was renamed as the Internet. In 1983, the military part of ARPAnet was split and named as MILNET.

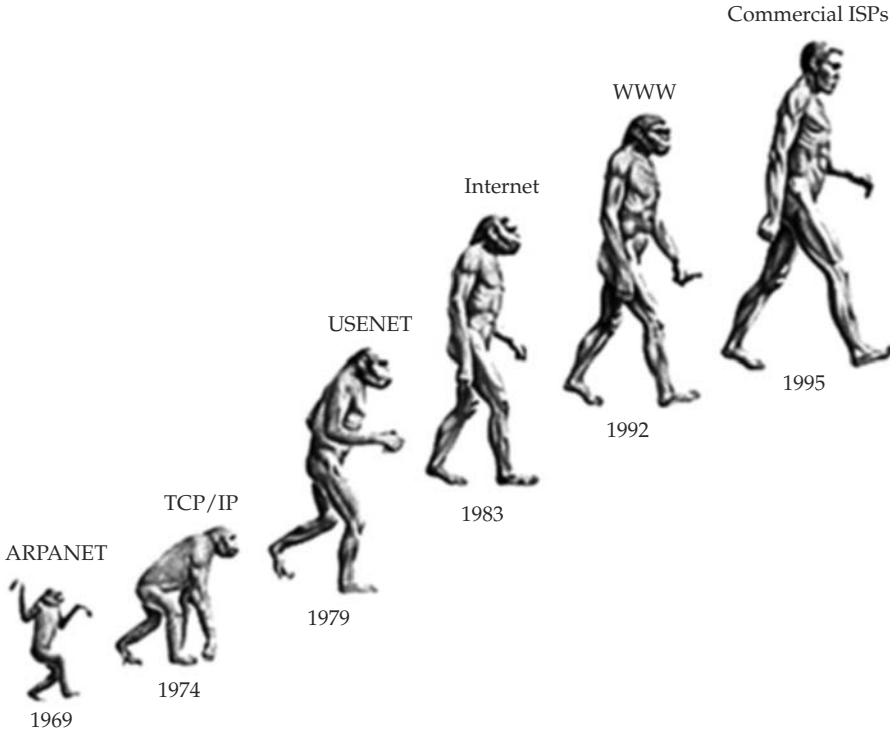


Fig. 2.15 Evolution of Internet

The number of computers connected to ARPAnet increased day-by-day because of which a new problem arose—the mapping of host names to the IP addresses became difficult. The IP addresses were the addresses provided to each host computer on the network. Earlier, the Network Information Center (NIC) maintained a record of the IP addresses and the corresponding host names in a file and every computer downloaded this file whenever required. But with the increase in the number of computers connected to the ARPAnet, it became difficult for the NIC to do so.

In 1984, Domain Name Server (DNS) was developed that helped in the mapping of host names to the IP addresses. The domain names such as .edu for educational, .com for commercial, .gov for governmental and .org for international organisational hosts, were introduced along with a specific code for each country. The DNS converted these domain

names into the corresponding IP addresses. The domain names also made it easier for the users to remember the addresses.

In 1987, the National Science Foundation established a network known as NSFNET that linked the computers with a high speed of 56 Kbps. NSFNET allowed many organisations to connect to the Internet without following the ARPAnet's policies.

By 1990, almost all the organisations, which were connected to ARPAnet, shifted to NFSNET and ARPAnet came to an end. The use of the Internet was limited to email, Telnet and FTP till 1990. In 1990, the McGill University introduced an FTP search tool known as Archie, which helped the users search for the information on the Internet.

In the next year, i.e., in 1991, another application known as Gopher was developed at the University of Minnesota. This service helped the users arrange the documents on the Internet; to locate these documents, a Gopher search tool known as Veronica was used.

In 1992, Tim Berners-Lee, a physicist from Geneva, introduced the World Wide Web (WWW), which was a network of Web sites that could be accessed with the help of a protocol known as Hyper Text Transfer Protocol (HTTP). HTTP searched the required address from where the Web pages had to be accessed and retrieved the Web pages for the user. The Web page on the WWW had links in the form of text known as hypertext. These links were helpful in accessing other Web pages when a user clicked the hypertext with the help of a mouse.

In 1995, the commercial Internet providers started controlling the Internet by providing connections to different people for accessing the Internet. Today the Internet is used in almost every field, such as education, entertainment, business, defence and medicine. In all these fields, the Internet is used to share data, gather information and communicate with the other users. According to the Internet world usage statistics, more than 1.4 billion people use the Internet today.

2.10 BASIC INTERNET TERMINOLOGIES

Internet is a worldwide network of computers comprising various small networks such as Local Area Network (LAN) and Metropolitan Area Network (MAN). It is a powerful communication medium that interconnects people across the world through technologies such as e-mail, Web and newsgroups. With the help of Internet, a large number of people can communicate simultaneously.

Various key terms related to the Internet are as follows:

- **Network** It is a connection established between two or more computers to allow them to share data and resources with each other.
- **LAN** It is a small area network that is generally spread over a small geographical area such as college or office.
- **MAN** It is a large area network as compared to LAN that is usually spread over a city or a campus area ranging between 5 and 50 km. It is used to connect two or more LANs with the help of multiple routers, hubs and switches.

- **Wide Area Network (WAN)** It is a computer network that covers a very large geographical area as compared to LAN and MAN, such as states and countries. It connects various LANs and MANs with each other.
- **Bandwidth** It is defined as the maximum amount of data, which can be transferred over a communication network. It is specified in bits per second (bps). The amount of data that can be transmitted is directly proportional to the bandwidth of the communication network.
- **Browser** It is an application program, which enables the Internet users to access the WWW. It searches the Web page requested by the user on WWW and loads it on the computer system. A browser is typically of the following two types:
  - **Non-graphical** It allows the users to browse only text-based Web pages on the Internet that do not contain any image, sound or video elements. For example, Lynx is a non-graphical browser that is mostly used by the users of Unix operating system.
  - **Graphical** It allows the users to browse multimedia Web pages containing plain text, images, sounds, animations, videos, etc.
- **Web site** It is a collection of related Web pages connected through various hyperlinks and available on the Internet.
- **WWW** World Wide Web is a collection of Web servers, which contain several Web pages pertaining to different Web sites. The Web pages contain hypertext, simple text, images, videos and graphics.
- **HTML** It is a language used for designing the Web pages of a Web site. These Web pages are interconnected through hyperlinks.
- **Domain name** It is a name that helps in identifying a specific Web site on the Internet. It generally consists of two or more parts separated by dots. The leftmost part typically specifies the name of the organisation or entity, while the rightmost part specifies the top-level domain. For example, yahoo.com is a domain name, where yahoo specifies the name of organisation and com specifies the top-level domain, which identifies the purpose of the particular organisation. Table 2.1 provides a list of various top-level domains.

TABLE 2.1 Top-level domains

Name	Purpose
Com	Commercial
Edu	Educational
Gov	Government
Org	International organisation
Mil	Military
Net	Internet Service Provider (ISP)

- **MODEM** MOdulator/DEModulator (MODEM) is a communication device used for transmitting the data between computers and the network over a telephone line. It converts digital data from a computer into analog signals and vice versa for enabling telephone transmission.
- **Search engine** It is software program used for searching information on the Internet on the basis of specified search criteria. Some common examples of search engines are Google, Yahoo and Bing.
- **Uniform Resource Locator (URL)** It specifies the address of a particular resource available on the Internet. It comprises the server name and the resource name. For example, www.yahoo.com/greetings.html is a URL, where www.yahoo.com is the server name and greetings.html is the file name located on the server.
- **HTTP** Hyper Text Transfer Protocol (HTTP) is a protocol used for requesting a Web page from the Web server.
- **HTTPS** Hyper Text Transfer Protocol, Secure (HTTPS) is a secured version of HTTP protocol developed by Netscape. It ensures security through encryption technique.
- **FTP** File Transfer Protocol (FTP) is a protocol used for requesting files from the FTP server.
- **Spam** It is an unwanted e-mail message simultaneously sent to a number of users over the Internet in order to advertise some product.
- **Domain Name System (DNS)** It is an Internet service that maps the domain name such as domain.com to a specific IP address, such as 192.105.112.40.
- **E-commerce** It is a type of commercial business conducted over the Internet. It involves sale and purchase of various goods and services via the Internet.
- **E-mail** It is a document or a message generally exchanged between two or more users over the Internet. It is a very effective communication medium through which two users, physically located apart, can easily send messages to each other.
- **Instant messaging** It allows two or more users at distinct geographical locations to instantly communicate with each other with the help of text messages.
- **Firewall** It is a software program that provides security to a network of computers by restricting the access of unauthorised users.
- **IP Address** It is a 32-bit address divided into four three-digit numbers ranging from 0 to 255 and separated by periods. It uniquely identifies a computing resource over a network.
- **Virus** It is a malicious software program that normally causes damage to a computer system by destroying its files, disabling the software, etc.

2.11 GETTING CONNECTED TO INTERNET APPLICATIONS

To use the Internet, the user is required to set up a connection first. To set up a connection, the most important requirement is that of a modem. Further, the user has to get an Internet connection from one of the Internet Service Providers (ISPs). ISPs provide two types of Internet

connections, namely, dial-up and broadband. Once the Internet connection is set up, the user requires a Web browser software to use the Internet. Let us now understand how Internet applications are actually accessed with the help of a Web browser.

2.11.1 Browsing the Internet

Browsing the Internet is the process of accessing different Web sites available on the Internet using a Web browser. To browse the Internet, we need to first connect our computer to the Internet through a connection provided by an ISP. After connecting to the Internet, we can perform the following steps to browse the Internet using IE Web browser:

1. Select Start -> Programs -> Internet Explorer (as shown in Figure 2.16) to display the Microsoft Internet Explorer window.

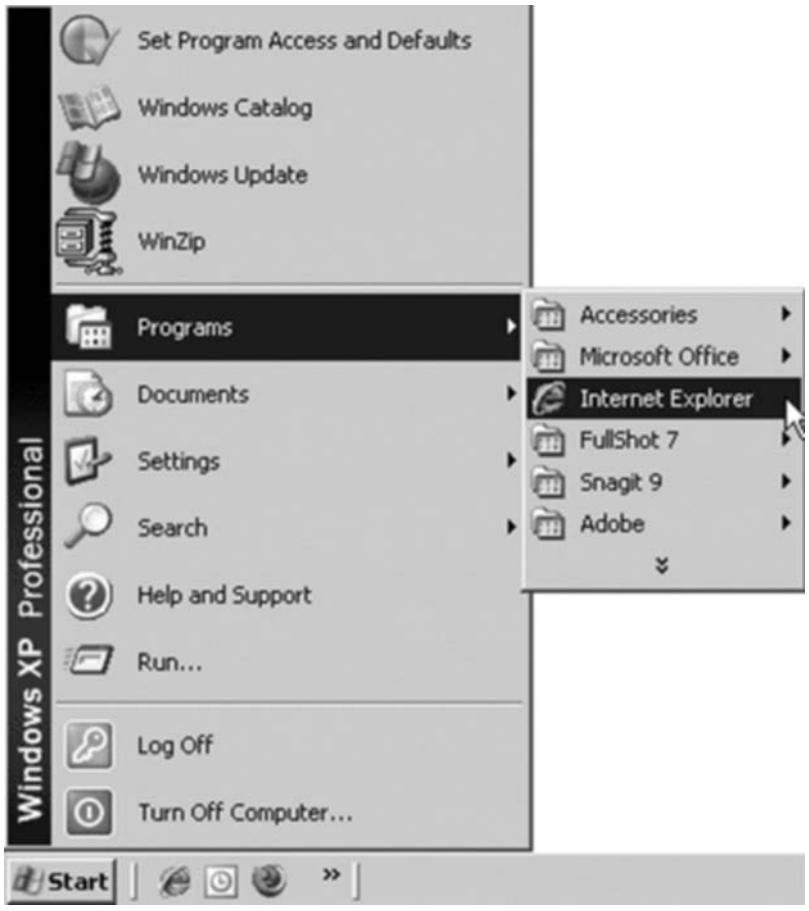


Fig. 2.16 **Launching Internet Explorer**

2. Type the URL of the Web site that you want to access such as java.sun.com in the address bar that appears at the top in the Microsoft Internet Explorer window, as shown in Figure 2.17.

Fig. 2.17 Opening a Web Site

3. Press the Enter key to display the home page of the java.sun.com Web site, as shown in Fig. 2.18.

Fig. 2.18 Accessing a Web site

When browsing the Internet, we can use one of the important features—favourites—of the IE Web browser. This feature allows us to easily and quickly access the frequently visited Web sites. In other words, the favourites feature allows the users who are browsing the Internet to keep a record of all the Web sites that are their favourites and are required to be accessed frequently. This feature also proves to be handy when a user wants to avoid the tedious task of remembering the URL of the Web sites, which are frequently accessed by that user. Users can simply add the Web sites that are frequently accessed by them in the list of favourites so that they can be easily accessed in the future without remembering their URLs. To access a Web site that has been added to the favourites, we can use the Favourites menu or the Favourites button, which appears in the Standard toolbar. When we click the Favourites button in the toolbar, the Favourites pane appears as shown in Figure 2.19.

Fig. 2.19 **The Favourites pane**

The Favourites pane contains a list of names of the Web sites, which have been added to favourites. We can now click a specific name in the Favourites pane to access the corresponding Web site quickly. We can also use another important feature, i.e., 'search' available in IE Web browser when browsing the Internet. The search feature allows us to search for information on a specific topic. If a user is not aware of any Web site, which contains the desired information, then the user can use the search feature of the IE Web browser. When a user searches for information through the search feature of IE, then a list of Web sites containing the related information is displayed. To search for information using the search feature of IE, we can click the Search button, which appears in the Standard toolbar. The Search pane appears as shown in Fig. 2.20.

Fig. 2.20 **The Search pane**

In the Search pane, you can type a word in the 'Find a Web page containing' text box related to the topic on which you want to search the information. After typing the word, click the Search button to search the information related to the topic.

NOTE: Apart from Favourites and Search features, the Standard toolbar of the IE Web browser contains other options like Home, Next, Previous, etc., which are used for performing different functions.

2.11.2 Using a Search Engine

Search engines are the Web sites that provide the users the facility of searching information related to some topics. The search engines maintain an index of Web sites available on the Internet along with a summary of the content in these Web sites. The summary of the content is displayed when a user searches for information on a topic. We can use a number of techniques to make the process of searching the information using search engines efficient and useful. These techniques are as follows:

- **Use of double quotes (" ")** Double quotes are used to search for a Phrase collectively. Generally, a search engine provides a list of Web sites that contain any of the words contained in the text specified for searching of information. However, if a user wants the search engine to consider the entire text specified for searching as a single phrase, then the user should provide the text in double quotes
- **Use of Boolean operators** Boolean operators such as AND, OR and NOT can also be used in the text specified for searching information in order to make the search process more efficient. The Boolean AND operators can be used to search for Web sites that contain the information related to two words such as computer and hard disk. The Boolean OR can be used to search for Web sites that contain either of the two words. The Boolean NOT can be used to search for the Web sites that contain the information other than the information related to the word specified after the NOT operator.
- **Use of plus (+) and minus (–) sign** Plus (+) sign is used to search the Web sites in which all the words included in the text specified for searching are present. Minus (–) sign is used to search all the Web sites in which the words included in the text specified for searching are not present.

Some of the search engines that are commonly used by the users are www.google.com, www.altavista.com, www.askjeeves.com, www.search.yahoo.com and www.ask.com. Among these search engines, Google is the most popular search engine worldwide. To use the Google search engine for searching information, the user has to perform certain steps, which are as follows:

1. Open IE Web browser. The Microsoft Internet Explorer window appears, as shown in Figure 2.21.
2. Type the URL, google.com, in the address bar, which appears at the top in the Microsoft Internet Explorer window, as shown in Figure 2.22.
3. Press Enter to open the home page of the google.com search engine as shown in Figure 2.23.

Figure 2.23 shows the home page of the Google search engine containing a text box in which we can specify the text for searching information. It also contains two buttons, Google

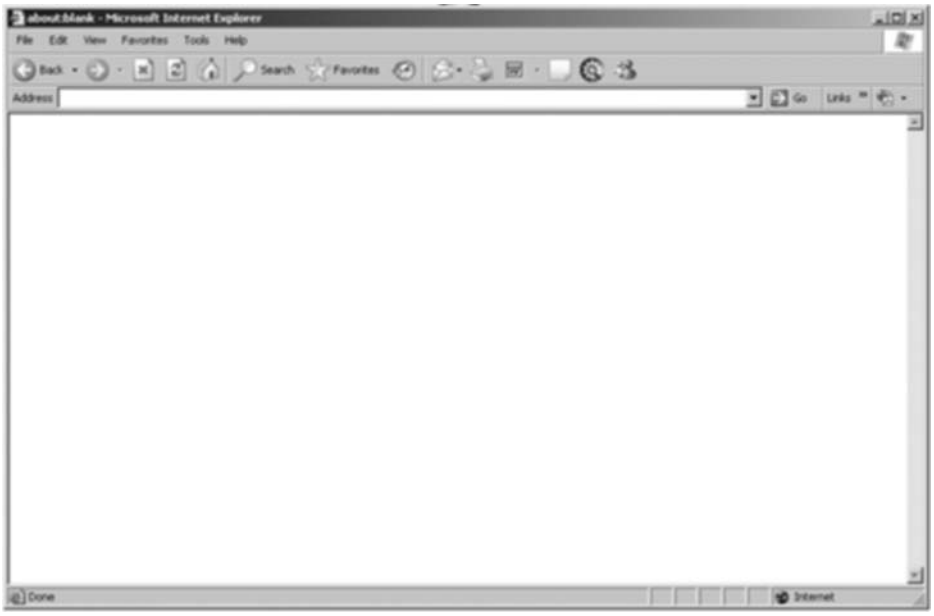


Fig. 2.21 **IE Window**

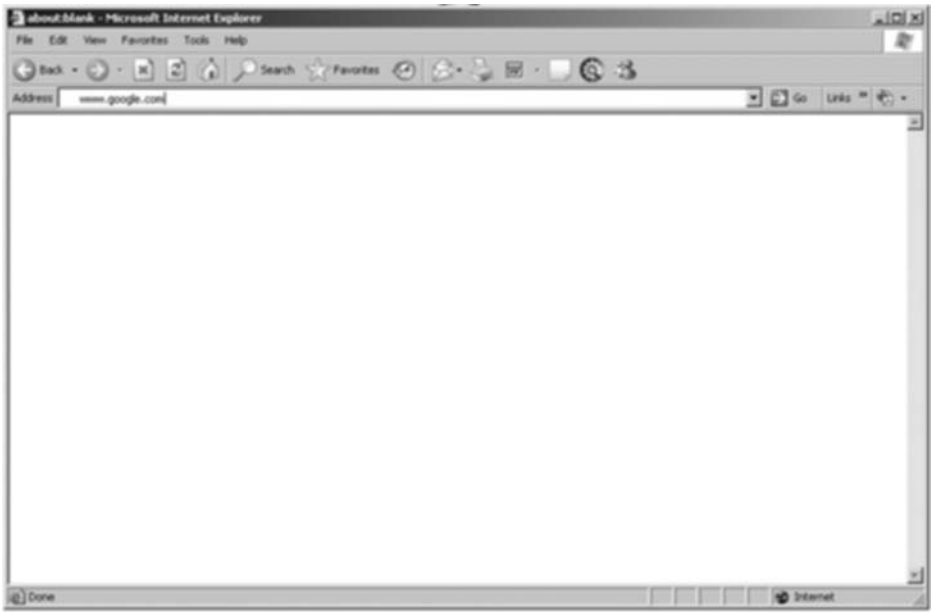


Fig. 2.22 **Opening Google Site**

Fig. 2.23 **The home page of google.com search engine**

Search and I'm feeling Lucky. The Google Search button is used to initiate the searching process using the search engine. The 'I'm Feeling Lucky' button is used to display the first Web site returned as a search result corresponding to the specified text. The home page of the Google search engine also contains the following options:

- **The web** This option allows us to search the WWW for getting the information on a specific topic.
- **Pages from India** This option allows us to search the pages submitted from India for information on specific topic.
- **Advanced search** This option allows us to specify certain advanced settings for making the search more specific and efficient.

Type some text such as 'mouse + computers' in the search text box to specify the text on the basis of which information is to be searched. Click the Google Search button or press the Enter key to initiate the process of searching the information. A list of Web sites containing information related to the specified text gets displayed, as shown in Fig. 2.24.

In Fig. 2.24, the blue text in the search result is the hypertext through which the Web page containing the required information can be accessed and the green text represents the URL of the Web site to which the Web page belongs. A user needs to click the hypertext button to view the Web page containing the required information.

Fig. 2.24 **The search results from google.com search engine**

2.11.3 Uses of Internet

Nowadays, the Internet is used in almost all the fields for varied purposes. People use one or the other services provided by the Internet for performing their day-to-day tasks. The Internet is extensively used in the following fields:

- Business
- Education
- Communication
- Entertainment
- Governance

2.11.4 The Internet in Business

In business, the Internet can be used for many purposes. An organization can provide details about its products on the Internet that can be either used by the other organizations interested in developing business links with it or by the prospective customers. Business transactions such as sale and purchase of products and online payment can also be performed using the Internet. This service of the Internet is called e-business, which can be further classified into the following categories:

- **Business-to-business (B2B)** B2B e-business refers to the business transactions that take place between two business organizations. In B2B, a large website acts as a

market place and helps the buyers and suppliers interact at the organizational level. The website acting as a market place helps the buyers to find new suppliers and the suppliers to search for new buyers. It also saves the time and cost of interaction between the organizations. For example, a supplier business organization can provide certain raw materials to a manufacturing business organization through its website.

- **Business–to–consumer (B2C)** B2C e-business refers to the business transactions that take place between an organization and a consumer directly. In B2C, a consumer can shop online for the products offered by a business organization. It provides all the information regarding the available products through a website and allows the consumers to order and pay for the products online, thus facilitating fast and convenient shopping. For example, the Asian Sky Shop sells the various products offered by different business organizations online and any user who wants to purchase a product can buy it online.
- **Consumer–to–consumer (C2C)** C2C refers to the business transactions that take place between two consumers but with the help of a third party. In C2C, a consumer provides information about a product, which is to be sold, on the website of the third party. Another consumer can buy the same item through bidding on the website of the third party. The consumer, who provides an item for sale on the website, is known as seller; whereas the consumer, who bids for the item, is known as buyer. For example, e-bay is a website on which a consumer can provide information about the products, which s/he needs to sell. The best bidder gets to buy the listed product.
- **Consumer–to–business (C2B)** C2B is a business model that allows individual consumers to offer their products and services to companies in return of which they get payment from the companies. One of the popular examples of C2B model is the online advertising site Google Adsense. It allows individuals to display advertising content or promotional materials on their personal websites. The administration and payment of these ads are done by Google itself. Also, platforms like Google Video and Fotolia are good examples of C2B, where individuals can sell digital content including images, animations, videos, etc. to companies.

2.11.5 The Internet in Education

In the field of education, the Internet is widely used for learning and teaching. The Internet not only helps the students search information on various topics of their interest but also proves useful for the students pursuing distance education. The distance education institutes provide notes, lectures and syllabus to students through their respective websites. The students just have to access the website of the institute to get all the required information from it. If the website of an educational institution supports e-learning, then the students can participate in online lectures through simulations, Web Based Training (WBT), etc.

The Internet also provides the Usenet service, which contains a large number of Newsgroups through which a user can submit as well as obtain the articles on different topics. The members of a newsgroup connect to each other and have discussions through the Usenet network. Usenet contains a number of message boards on which the articles are placed and the software known as newsreader is used to read the articles published on message board.

Most of the newsgroups allow the users to submit their articles on the selected topics such as scientific research, social issues, religion and politics. Moreover, some newsgroups also allow the users to submit their articles on a topic of their own choice. Newsgroup not only helps the users in gaining knowledge but also allows them to make online friends.

The Internet also provides an application similar to Newsgroups known as Discussion forum. The discussion forum also allows a large number of people to hold discussions or place their articles on a particular topic similar to Newsgroups. But the only difference between the Discussion forums and Newsgroups is that the Discussion forums display articles according to the time or the thread of receiving the article. The thread refers to the grouping together of all the messages received on a particular topic. Some discussion forums allow the users to place their articles even without having a membership of the Discussion forum, while the other Discussion forums require the users to have membership along with valid username and password. The members of such Discussion forums have special facilities such as to make alterations in their previous articles, to initiate a new topic and to delete the previous articles submitted by them.

Both Newsgroup and Discussion forums are used by students and other users of the Internet to share their knowledge with each other by participating in a discussion on a specific topic. The extensive use of the Internet in education has led to the creation of what are known as Virtual Universities in many countries.

2.11.6 The Internet in Communication

The Internet is mostly used by the people as a fast and cheap means of communication. Many services provided by the Internet such as e-mail and instant messaging help the users to communicate quickly and cheaply over long distances. E-mail is an application of Internet that allows a user to send and receive text messages electronically.

To use the email services, a user requires an account on a mail server. The account is created by the user by providing a username; a password and other personal information such as address and contact number. Each time the user wants to access the email service, she/he has to log on to the server using the username and the password provided during account creation. If the username or the password provided by a user is invalid, then that user is considered as an unauthorized user and is prohibited from using the service.

The Internet also provides another easy way of communication, i.e., communication through instant messaging. Instant messaging is a service of the Internet through which it is possible for a user to perform real-time communication with one or more users on the Internet. The real-time communication refers to the communication in which there is an immediate response to a message. In case of instant messaging, the communication between two users takes place by instant sending and receiving of message.

To use this service, the users have to log on to the instant messaging server. After a user has successfully logged on to the server, a chat room with a list of online users is made available to the user. An online user is a user who is available for chatting at a specific period of time.

The user can then select an online user from the list and then send a message to that online user. If a response is received from the online user to whom a message was sent, then instant sending and receiving of message takes place. Chat rooms not only provide the sharing of text messages but also allow the users to share images and graphics online.

Apart from e-mail and instant messaging, Internet telephony and web conferencing are the other application areas of the Internet that facilitate, quick, cheap and efficient communication over long distances. Through these mediums, the users can talk to the other uses in real-time through an audio-visual interface.

2.11.7 The Internet in Entertainment

The Internet over the period of time has evolved as a great source of entertainment. It provides many entertainment resources to the users such as games, music and movies. The most popular entertainment resources on the Internet are the games, which are either free of cost or can be bought through the payment of a small price.

Multi User Dungeon (MUD) is a virtual environment in which fantasy characters such as warriors, priests and thieves are adopted by end users for playing games. Each user represents a specific character and interacts with other characters with the help of text messages. The information regarding the game and the virtual environment is also provided to the users through commands displayed on the screen. MUD is also available with graphics that enhance the background of the game by providing it a 3-dimensional look. This feature is known as virtual reality because the background and the characters in the game resemble the real world entities.

Apart from games, the Internet also provides many other entertainment resources. Several websites provide easy access to any type of music and videos, which can be freely downloaded on the computers. The Internet also enables the users to share videos and photos with other users. Many websites on the Internet also provide information regarding the sports events taking place at specific period of time. These websites allow the users to access continuous score updates.

2.11.8 The Internet in Governance

These days, the Internet is playing a crucial role in the functioning of the government organizations. Almost all the government organizations have set up their websites that provide information related to the organization as well as help them in performing their operations. For example, nowadays people can submit the passport application form and file the income tax returns through the use of Internet. Moreover, Internet also enables the government agencies to share the data with each other.

The vast use of IT and Internet has paved the way for e-governance. More and more government agencies are adopting the concept of e-governance to improve their service delivery capabilities.

SUMMARY

Computer software is one of the most important components of a computer system. It helps the computer hardware in carrying out their functions in an effective manner. Computer software can be either system software or application software. System software is responsible for managing and controlling the hardware resources of a computer system. System software can be further divided into two major categories, which are system management programs and system development programs. Operating system, utility programs and device drivers are some of the examples of system management programs. Typical examples of system development programs include linkers, debuggers and editors.

Application software is specially designed to cater the information processing needs of end users. Like system software, application software can be further divided into two major categories, which are standard application programs and unique application programs. Word processor, spreadsheet, database manager, desktop publisher and Web browser are some of the standard application programs. The unique application programs are specially developed by the users to meet their specific requirements. For examples, inventory management system and pay-roll system come under the category of unique application programs.

A series of steps are performed in the development of a software, which are collectively known as software development life cycle. The various stages of software development life cycle are requirement analysis, feasibility analysis, design, development, testing, deployment, and maintenance.

Apart from the standalone computer systems, another computing or rather communication technology that has revolutionised the world is Internet. It is a robust and multi-faceted platform that allows the users all over the world to instantly communicate and collaborate with each other.

POINTS TO REMEMBER

- **Hardware**: It refers to the various electronic and mechanical devices, which are responsible for performing various operations such as storing data into the storage devices, transferring data through buses and cables, etc.
- **Software**: It is a set of programs and procedures containing a number of instructions for processing the data.
- **System software**: It is the program that directly controls the working of hardware components of the computer system, and enables implementation of application software.
- **Application software**: It is the software that enables the end users to perform various tasks such as creating documents, preparing presentations, creating databases, developing graphics, etc.
- **Operating system**: It is a set of various small system software that control the execution of various sub-processes in a computer system.
- **Utility program**: It refers to a small program that helps perform utility tasks in a computer system.
- **Device driver**: It is a special software that enables a hardware device such as keyboard, monitor and printer to perform an operation according to the command given by the end user.

- **Editor**: It is a program that allows the user to write and edit text in the computer system.
- **Spreadsheet**: It is a computer program that enables the user to maintain a worksheet in which information is stored in different cells.
- **Compiler**: It translates a high-level program into a low-level program.
- **Assembler**: It translates an assembly language program into a low-level program.
- **Linker**: It assembles the various objects generated by the compiler in such a manner that all the objects are accepted as a single program during execution.
- **Debugger**: It locates the position of the errors in the program code with the help of ISS technique.
- **LAN**: It is a small area network that is generally spread over a small geographical area such as a college or an office.
- **Wide Area Network (WAN)**: It is a computer network that covers a very large geographical area as compared to LAN and MAN, such as states and countries.
- **Bandwidth**: It is defined as the maximum amount of data, which can be transferred over a communication network.
- **Web site**: It is a collection of related Web pages connected through various hyperlinks and available on the Internet.
- **Search engine**: It is an Internet application used for searching information on the Internet on the basis of specified search criteria.
- **HTTPS**: It is a secured version of HTTP protocol developed by Netscape.
- **Spam**: It is an unwanted e-mail message simultaneously sent to a number of users over the Internet in order to advertise some products.
- **E-mail**: It is a document or a message generally exchanged between two or more users over the Internet.
- **IP Address**: It is a 32-bit address divided into four three-digit numbers ranging from 0 to 255 and separated by periods. It is used to uniquely identify a resource over the network.
- **Virus**: It is a malicious software program that normally causes damage to a computer system by destroying its files, disabling the software, etc.

REVIEW QUESTIONS

1. Software is an interface between the hardware and the user.
2. Computer software is of two types namely system software and application software.
3. System development programs are used for managing both the hardware and software systems.
4. Browser is used to search Web resources on the Internet.
5. System profiler and virus scanner both are application software.

6. A computer system is connected with multiple input and output (I/O) devices so that it can communicate with the end user.
7. A debugger interprets the input provided by the user into the computer understandable form and directs it to the operating system.
8. MS-Excel is an example of system program.
9. Tally is an example of unique application program.
10. Disk defragmenter is an example of application program.
11. Linker assembles the various objects generated by the compiler in such a manner that all the objects are accepted as a single program during execution.
12. Spreadsheet is an application software that enables the users to maintain a worksheet in which information is stored in different cells.
13. Debugger locates the position of the errors in the program code with the help of ISS technique.
14. Desktop publishing system enables the users to perform various activities required for publishing a page or a set of pages.
15. A compiler translates a low-level program into a high-level program and an assembler translates an assembly language program into a low-level program.
16. MODEM only converts digital signals into their analog counterparts.

FILL IN THE BLANKS

1. A computer system consists of two types of components, _____ and _____.
2. Computer software is classified into two categories, namely _____ and _____.
3. System software consists two groups of programs: _____and _____.
4. _____ is responsible for operating the hardware system and managing their resources effectively.
5. Device driver acts as a translator between the _____ and the _____.
6. _____is a computer software that helps us to store and maintain records in a database.
7. Language translator converts the high-level language program into the low-level program called _____.
8. Application software can be broadly classified into two types: _____and _____.
9. _____ is an application software that is generally used to create documents such as letters, reports, etc.
10. Spreadsheet is generally used to calculate the values stored in a cell by applying _____on the values of other cells.
11. _____ refers to a set of records, which are stored in a structured manner in the computer system.
12. In the hierarchical model, records are arranged in the form of a _____in which each record is attached with one parent node and one or more children nodes.

13. Debugger and linker are examples of _____ programs.
14. _____ is a language used for designing the Web pages on the WWW that contain links to other related Web pages.
15. _____ is a collection of various Web sites containing different Web pages.

MULTIPLE CHOICE QUESTIONS

1. Which of the following is not a software?
   A. Device driver
   B. Barcode reader
   C. Interpreter
   D. Linker
2. A device driver acts as an interface between:
   A. End user and I/O device
   B. Application software and I/O device
   C. Application software and operating system
   D. Operating system and I/O device
3. A malicious software program that normally causes damage to a computer system by destroying its files, disabling the software, etc. is known as.
   A. Firewall
   B. Editor
   C. Virus
   D. Device driver
4. Which of the following is not a system software?
   A. Linkers
   B. Device drivers
   C. Operating system
   D. Word processor
5. Which of the following software helps the users to detect the errors in a program?
   A. Language Translator
   B. Debugger
   C. Loader
   D. Linker
6. HTTP is an acronym for:
   A. Hyper Text Transfer Protocol
   B. Hyper Text Transport Protocol
   C. Hyper Target Transfer Protocol
   D. High Text Transport Protocol
7. A program in which multiple instructions of high-level language or assembly language instructions are written is generally known as:
   A. Psuedocode
   B. Source code
   C. Object code
   D. Machine code
8. A small program, which provides additional capabilities to the operating system, is known as:
   A. Application program
   B. System development program
   C. Utility program
   D. Translator program
9. A software, which links different elements of an object code in to a single program, is known as:
   A. Editor
   B. Linker
   C. Loader
   D. Debugger

10. Which of the following is not a utility program?
    A. Virus scanner B. System profiler
    C. Disk defragmenter D. Debugger
11. Which of the following is not a testing method?
    A. Black box testing B. Gray box testing
    C. Final testing D. System testing
12. Which of the following data models enables us to arrange the data files in the form of a tree structure?
    A. Network model B. Relational model
    C. Hierarchical model D. None of these
13. A software system that enables the users to perform various activities required for publishing a page or a set of pages is known as:
    A. Word processor B. Editor
    C. DTP D. Debugger
14. It is a small area network that is generally spread over a small geographical area such as a college and an office.
    A. MAN B. LAN
    C. WAN D. Network
15. It is a communication protocol, which is basically used for transferring the files from one computer to another over the Internet.
    A. TCP/IP B. FTP
    C. HTTP D. URL

ANSWERS

True or False

1. True 2. True 3. False 4. False 5. False 6. True 7. False 8. False
9. True 10. False 11. True 12. True 13. True 14. True 15. False 16. False

Fill in the Blanks

1. Hardware and software
2. System software and application software
3. System management programs and system development programs
4. System management programs
5. I/O devices and the computer
6. DBMS
7. Object code
8. Standard application programs and unique application programs
9. Word processor
10. Formulas
11. Database
12. Tree
13. System development
14. HTML
15. WWW

Multiple Choice Questions

1. B	2. D	3. C	4. D	5. B	6. A.	7. B	8. C
9. B	10. D	11. C	12. C	13. C	14. B	15. B	

1. Describe the basic components of a computer system.
2. Explain the different types of computer software.
3. What do you understand by the term system software?
4. Explain the major functions of an operating system.
5. What is the main purpose of a translator program?
6. What are utility programs?
7. What are the basic functions of DBMS?
8. Explain inventory management system.
9. What are unique application programs?
10. Differentiate between compiler, assembler and interpreter.
11. What is a debugger?
12. Describe the main features of a word processor application.
13. Explain the main features of spreadsheet software.
14. Describe the various software development steps.
15. Explain the following terms:
    A. Network B. LAN C. MAN D. WAN
16. Explain the following terms:
    A. WWW B. HTML C. FTP D. HTTP

ANSWERS TO 2009 QUESTION PAPERS

PART A

1. Enumerate the steps involved in software development. (GE1102)
   Ans: Refer Section 2.8.
2. What is a shareware? What are the advantages and limitations of using a shareware? (GE1102)
   Ans: Shareware is a type of software, which is distributed as a demo version for free but with incomplete functionality. Shareware has limited functionality in comparison to its full version, which is normally available at a certain cost. Sharewares are generally downloaded from the Internet or can be installed from a Compact Disc (CD) supplied with magazines, journals, etc. Some of the advantages and disadvantages of shareware are:

Advantages

- It is free of cost.
- It helps to get a look and feel of the software before actually spending the money to buy it.

Disadvantages

- It does not contain all the features of the software.
- It is available for a limited time period.

3. Distinguish between application software and system software.

(Anna University, Jan - Feb 2009, GE2112)

Ans: Refer Section 2.3.

4. What is a loader? What are its basic tasks? (Anna University, Jan - Feb 2009)

Ans: Loader is a system development program, which loads the executable files on to the main memory. Thus, it helps in executing a program. Some of the basic tasks of loader are:

- Allocating memory space for the executable files
- Creating an image of the executable files in the main memory
- Loading the static data required for a program's execution

5. Distinguish between graphical browsers and text browsers

(Anna University, Jan - Feb 2009)

Ans: It is an application program, which enables the Internet users to access the WWW. It searches the Web page requested by the user on the World Wide Web and loads it on the computer system. A browser is typically of the following two types:

- **Non-graphical**: It allows the users to browse only text-based Web pages on the Internet that do not contain any image, sound or video elements. For example, Lynx is a non-graphical browser that is mostly used by the users of Unix operating system.
- **Graphical**: It allows the users to browse multimedia Web pages containing plain text, images, sounds, animations, videos, etc.

6. Give the major steps of connecting your system with Internet.

(Anna University, Jan - Feb, 2009)

Ans: The major steps of connecting a computer system with Internet are:

A. Install a modem the computer system.
B. Install a graphical Web browser.
C. Set up the Internet connection with an ISP.
D. Launch and use the browser for accessing the Internet.

7. What are the various categories of softwares? (Anna University, Jan - Feb 2009)

Ans: Refer Section 2.3.

8. Name any two Internet applications. (GE2112)

Ans: E-mail and instant messaging.

9. Enumerate the steps involved in software development. (GE1102)
Ans: Refer Section 2.8.

PART B

1. Discuss the importance of system software for a computer system. Describe briefly some of the most commonly known types of system software. (GE1102)
Ans: Refer Section 2.3.
2. Describe briefly any four commonly known application software. (GE1102)
Ans: Refer Sections 2.6 and 2.7.
3. Explain various types of software with suitable examples. (GE2112)
Ans: Refer Section 2.3.
4. Explain in detail about the software development steps. (GE2112)
Ans: Refer Section 2.8.
5. Explain briefly about the evolution of the Internet. (Anna University, Jan - Feb 2009)
Ans: Refer Section 2.9.
6. Define 'URL'. Give its basic structure with an example.
Ans: Uniform Resource Locator or URL specifies the unique address of a particular resource available on the Internet. It comprises the domain name and the resource name. The domain name helps in identifying a specific Web site on the Internet. For example, www.yahoo.com/greetings.html is a URL, where www.yahoo.com is the domain name and greetings.html is the file name located on the server. Each and every URL accessed over the Internet is first resolved into its corresponding IP address. This task is performed by the Domain Name Server (DNS).

ANSWERS TO 2010 AND 2011 QUESTION PAPERS

SHORT ANSWER QUESTIONS

1. Distinguish between compilers and interpreters. (AU, Chn, Jan 2010)
Ans: The following points explain the key differences between compilers and interpreters:
A compiler compiles the entire source code at once while the interpreter translates and executes the code one line at a time.
A compiler analyzes the syntax of the code while the interpreter only checks for the keywords.
2. Give the importance of graphics packages. (AU, Chn, Jan 2010)
Ans: Graphics packages are used for creating and editing graphical images. Some of the important application areas of graphical packages are interface design, art and animations, engineering drawing, etc.
3. What are the differences between software and hardware in computer? (AU, Chn, Jan 2011)

Ans: Difference between Software and Hardware

Software	Hardware
Software is a computer program i.e. a set of logical instructions that tell how a computer/hardware should function.	Hardware comprises of the physical computer components such as monitor, printer, etc. that function as per the instructions provided by the computer software.
Software acts as an interface between computer users and the hardware.	Hardware works at the lower level and is actually responsible for executing the user initiated actions.

4. What is a Network Router? (AU, Mdu, Jan 2011)

Ans:

Network Router

A network router is used for transferring data packets across different computer networks. A router reads the address specified on a data packet to ascertain where it is heading and subsequently forwards it to its target network. There could be a large number of routers used in a dense interconnected networks system.

A router contains a number of interfaces to which different networks connect for internetworking. The interconnecting networks may be based on different technologies (Ethernet or FDDI) or have different topologies (LAN or WAN). The router simply reads the address where the incoming data packets are headed and routes them to their target network.

5. What is a filename in C? (AU, Mdu, Jan 2011)

Ans:

Filename

Filename is a string of characters that make up a valid filename for the operating system. It may contain two parts, a primary name and an optional period with the extension. Examples: Input. data, store, PROG.C, Student.c, Text.out, etc.

6. Define Software and mention its types. (AU, Cbe, Dec 10-Jan 11)

Ans:

Software

Software is a set of logical instructions, written in a computer programming language that tells the computer to perform specific tasks. It acts as an interface between the hardware and the users of the computer system.

Types of Software

Computer software is broadly divided into two types:

- **System software**: Controls and coordinates the hardware components and manages their performance. Example: operating system, device driver, etc.
- **Application software**: Allows the ends users to perform their desired tasks. Example: desktop publishing program, graphics package, etc.

6. List any two constraints on real-time operating systems. (AU, Cbe, Dec 10-Jan 11)

Ans:

Constraints on Real-time Operating Systems

The two key constraints on real-time operating systems are:

- **Time constraint**: It is the most significant constraint on such type of operating systems. It requires them to generate the correct result within a given timeline. A correct result not produced within the stipulated time is as bad as an incorrect result.
- **Resource constraint**: Such operating systems are typically used at places where there is a space constraint, for example, automatic washing machine, automobile fuel injection controller, etc. Hence, most of the times there is a constraint as far as the underlying hardware is concerned.

7. Name the phases of Software Development life cycle. (AU, Cbe, Dec 10-Jan 11)

Ans:

SDLC Phases

1. Analysis
2. Design
3. Development
4. Testing
5. Deployment
6. Maintenance

8. Differentiate a Compiler and Interpreter. (AU, Cbe, Dec 10-Jan 11)

Ans:

Difference between Compiler and Interpreter

The following points explain the key differences between compilers and interpreters:

- A compiler compiles the entire source code at once while the interpreter translates and executes the code one line at a time.
- A compiler analyzes the syntax of the code while the interpreter checks only for the keywords.

5. What is system software? (AU, TIR, Dec 10-Jan 11)

Ans: Refer Section 2.3.1

7. What is intranet and how it is differ from Internet? (AU, TIR, Dec 10-Jan 11)

Ans:

Intranet

Intranet is a private computer network comprising of multiple interconnected local area networks. It is typically used within an organization to cater to its networking requirements. Only the employees of the organization are authorized to access the intranet.

Difference between Intranet and Internet

Intranet	Internet
It is a private network.	It is a public network.
It is typically accessible to users of an organization.	It is accessible to users worldwide through ISPs.

8. Differentiate between LAN and WAN networks. (AU, TIR, Dec 10-Jan 11)

Ans:

Difference between LAN and WAN

LAN	WAN
It is a group of computers interconnected in a small area such as building, home, etc.	It is a group of computers connected in a large area such as continent, country, etc.
It is generally used for connecting two or more personal computers through some medium such as twisted pair, coaxial cable etc.	It is generally used for connecting two or more LANs through some medium such as leased lines, circuit switching methods etc.

9. State the role of Modem. (AU, TRI, Jan 2011)

Ans:

Role of Modem (Chapter 2)

Modem is used for connecting a computer system to the Internet. The primary role of a modem is to convert the computer generated digital messages into analog form for their transmission through an analog communication channel. Likewise, it also receives the analog signals from the network and decodes them back to their digital format.

10. Narrate few Internet applications. (AU, TRI, Jan 2011)

Ans:

Internet Applications

Some typical applications of Internet are:

- **E-mail**: Used for exchanging messages.
- **Instant Messaging**: Used for exchanging instant chat messages in real time.
- **FTP**: Used for file sharing.
- **Search Engine**: Used for searching the desired information on the World Wide Web (WWW).

DESCRIPTIVE QUESTIONS

1. Describe the different types of software with examples. (AU, Chn, Jan 2010)

Ans: Refer sections 2.3-2.7.

2. List the different software development steps and explain. (AU, Chn, Jan 2010)

Ans: Refer section 2.8.

3. Explain the common types of internet access. (AU, Chn, Jan 2010)

Ans:

Types of Internet Access

The various ways of Internet access are:

- **Dial-up connection**: It is one of the earliest and basic types of Internet access medium. It uses a telephone line to establish a connection with an Internet Service Provider (ISP) for accessing the Internet. A modem is connected to the computer for transferring data packets via the telephone line. This is one of the cheapest and relatively slowest means of accessing Internet.
- **Broadband connection**: It is another means of accessing the Internet that is faster and reliable. The two subtypes of broadband connections are Digital Subscriber Line (DSL) and cable network. A DSL connection uses the local telephone network for data transmission. DSL uses higher frequency band thus allowing simultaneous usage of the telephone line for Internet and voice call purposes. In case of cable network, the Internet services are provided along with the normal cable TV network, thus facilitating convenient Internet usage.
- **Wireless connection**: As the name suggests, a wireless connection does not require any cables or telephone lines for accessing the Internet. The connection is established through wireless Internet services such as WiFi network, cellular Internet services, etc. Even though wireless connections at times do not match up to the speed of broadband connections, their single most important advantage is that they allow the users to access the Internet while on the go.

4. Write short notes on web browser. (AU, Chn, Jan 2010)

Ans:

Web Browser

Web browser is the software, which is used to access the Internet and the WWW. It is basically used to access and view the web pages of the various websites available on the Internet. A web browser provides many advanced features that help achieve easy access to the Internet and WWW. When we open a web browser, the first page, which appears in the web browser window, is the home page set for that particular web browser.

The web browsers are categorized into two categories, text based and Graphical User Interface (GUI) based. The text based browsers are the browsers that display unformatted text contained in the HTML files. These types of browsers do not display images, which are inline with the text contained in the HTML files. However, the text based browsers have the ability of displaying the images that are not inline with the text contained in the HTML files. The text based browsers are simple to use and do not require computers with expensive hardware. They allow the downloading of graphic and sound files but only if the computer contains the software and the hardware required for such files. The GUI based browsers, on the other hand, display formatted text along with images, sounds and videos, which are contained in the HTML files. The user has to just click the mouse button to view or download image, sound and video files.

The most commonly used web browsers are Internet Explorer (IE), Netscape Navigator and Mozilla Firefox.

5. Explain a typical structure of URL. (AU, Chn, Jan 2010)

Ans:

URL

Uniform Resource Locator or URL specifies the unique address of a particular resource available on the Internet. It comprises of the domain name and the resource name. The domain name helps in identifying a specific Web site on the Internet. For example, www.yahoo.com/greetings.html is a URL, where www.yahoo.com is the domain name and greetings.html is the file name located on the server. Each and every URL accessed over the Internet is first resolved into its corresponding IP address. This task is performed by the Domain Name Server (DNS).

6. Briefly write about Desktop Publishing Software. (AU, Chn, Jan 2010)

Ans: Refer section 2.6.4.

7. What are the types of software used in computer? Explain. (AU, Chn, Jan 2011)

Ans: Refer Sections 2.3 to 2.7.

8. Describe various software development steps in computer. (AU, Chn, Jan 2011)

Ans: Refer Section 2.8.

9. What are the various hardware components of a typical computer system? Also differentiate system software and application software? (AU, Mdu, Jan 2011)

Ans:

Hardware Components: Refer Section 1.8

Difference between System Software and Application Software

System Software	Application Software
Its primary task is to control and coordinate the hardware components and manage their performances.	Its primary task is to enable the users to perform their desired tasks.
It is essential for the functioning of a computer system.	It is an additional software that allows the users to perform specific jobs.
It gets automatically installed along with the installation of the operating system.	Application software is specifically installed as per the requirements of the user.
It includes system programs such as drivers, compilers, debuggers, etc.	It includes utility programs such as MS Word, medial player, accounting package, etc.
It can run independent of the application software.	It cannot run in the absence of system software.

10. W.hat are the various network topologies? Explain it. (AU, Mdu, Jan 2011)

Ans:

Network Topologies

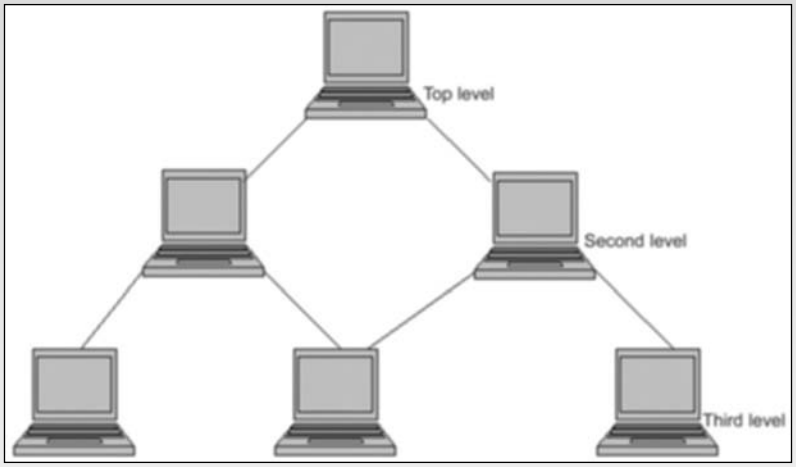
Network topology refers to the arrangement of computers connected in a network through some physical medium such as cable, optical fibre etc. Topology generally determines the shape of the network and the communication path between the various computers (nodes) of the network. The key network topologies are as follows:

- Hierarchical
- Bus
- Star

Hierarchical topology

The hierarchical topology, also known as tree topology, is divided into different levels connected with the help of twisted pair, coaxial cable or fibre optics. This type of topology is arranged in the form of a tree structure in which top level contains parent node (root node), which is connected with the child nodes in the second level of hierarchy with the point-to-point link.

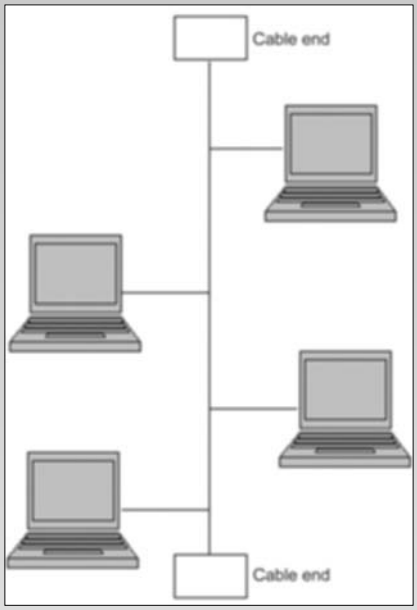
The following figure shows the hierarchical topology:



Bus topology

In the linear bus topology, all the nodes are connected to the single backbone or bus with some medium such as twisted pair, coaxial cable etc. When a node wants to communicate with the other nodes in the network, it simply sends a message to the common bus. All the nodes in the network then receive the message but the node for which it was actually sent only processes it. The other nodes discard the message.

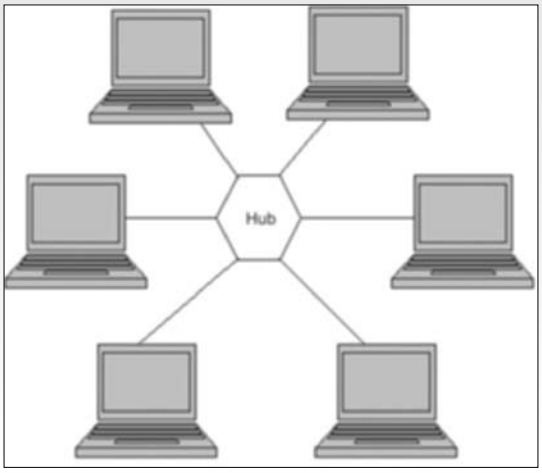
The following figure shows the bus topology:



Star topology

In the star topology, all the nodes are connected to a common device known as hub. Nodes are connected with the help of twisted pair, coaxial cable or optical fibre. When a node wants to send a message to the other nodes, it first sends the message to the hub, which in turn forwards the message to the intended node. Each node in the network is connected with a point-to-point link to the centralized hub. Another task of hub is to detect the faulty node present in the network.

The following figure shows the star topology:



11. What is WWW? What is .the main use of it? (AU, Mdu, Jan 2011)

Ans:

WWW

World Wide Web (WWW) is a collection of interlinked hypertext web pages accessed by the users through the Internet. These web pages comprise of hypertext, simple text, images, videos, animations, graphics, and many more elements. To view the Web pages hosted on a Web server, a software known as Web browser is required on the client computer. To display the web pages, web browser runs the HTML code segment written for a particular web page. Each web page on the Internet is provided its own unique address known as Uniform Resource Identifier (URI) or URL. This URL helps the web browser in locating a web page on the Internet.

One of the key uses of WWW is that it allows remote users to share knowledge and information with each other.

12. Summarize various services offered by Internet. Analyze the architecture of today's Internet and its applications. Identify necessary demands and requirements.

(AU, Cbe, Dec 10-Jan 11)

Ans:

Services offered by Internet

Some of the key services provided by the Internet are:

- World Wide Web
- Search engine
- Email

World Wide Web

World Wide Web is a collection of web servers, which contain several web pages pertaining to different websites. The web pages contain hypertext, simple text, images, videos and graphics. The web pages are designed with the help of HyperText Markup Language (HTML). To view the web pages provided by a web server, a software known as web browser is required. To display the web pages, a web browser runs the HTML code segment written for a particular web page. Each web page on the Internet is provided its own unique address known as Uniform Resource Identifier (URI) or URL. This URL helps the web browser in locating a web page on the Internet. A URL string begins with the name of a protocol such as http or ftp that represents the protocol through which a web page is accessed. The rest of the URL string contains the domain server name of the web page being accessed and the location of the web page on the local web server.

Search Engine

Search engines are the websites that offer the users the facility of searching for information related to some topic. The search engines maintain an index of websites available on the Internet along with a summary of the content contained in these websites. The summary of the content is displayed when a user searches for information on some topic.

Some of the search engines that are commonly used by the users are www.google.com, www.altavista.com, www.askjeeves.com, www.search.yahoo.com and www.ask.com. Among these search engines, www.google.com is the most popular search engine, which is used worldwide.

Email

Email service provided by the Internet is a very effective means of communication as it allows two persons at remote places to communicate with each other. To use the e-mail service, a user must have a valid email account.

The email service can be used by a user to send and receive email messages. An email message has two sections, a header section and a body section. The header section contains the information about the sender, the receiver and the subject of the email message. It consists of the fields such as to, cc and subject. The to field contains the email address of the person to whom the e-mail message is to be sent. The cc field or the carbon copy field contains the email addresses of other persons to whom a copy of the e-mail message has to be sent. The body section contains the actual text message, which has to be sent. This section also contains a signature block at the end where the sender places his/her signature.

Internet Architecture and Applications

Internet has come a long way since Sir Tim Berners-Lee founded the World Wide Web. Today, the power of Internet has given an all together new dimension to how businesses are conducted. It is impossible to imagine even a single day without the services of Internet at our disposal.

The core architecture of Internet has not much changed over the years. It still relies on the standard TCP/IP protocol to interconnect remote entities from discrete networks. But, today's Internet stands out from infrastructure perspective. Significant advancements in communication hardware and technologies have tremendously enhanced the speeds at which data is transferred over the Internet. Further, secured e-commerce, Web 2.0 and mobile Web, etc. have almost entirely remodeled today's Internet services as compared to its earlier days.

Necessary Demands and Requirements

There are certain demands and requirements that are closely associated with today's Internet, such as:

- Lack of cost-effective high bandwidth support
- Lack of availability of quality services
- Limitations on language development
- Recently witnessed IPv4 address exhaustion

13. Briefly discuss about any one of the system software. (AU, TIR, Dec 10-Jan 11)

Ans: Refer sections 2.4 and 2.5.

14. Discuss about various common types of networks with benefits and limitations.
(AU, TIR, Dec 10-Jan 11)

Ans:

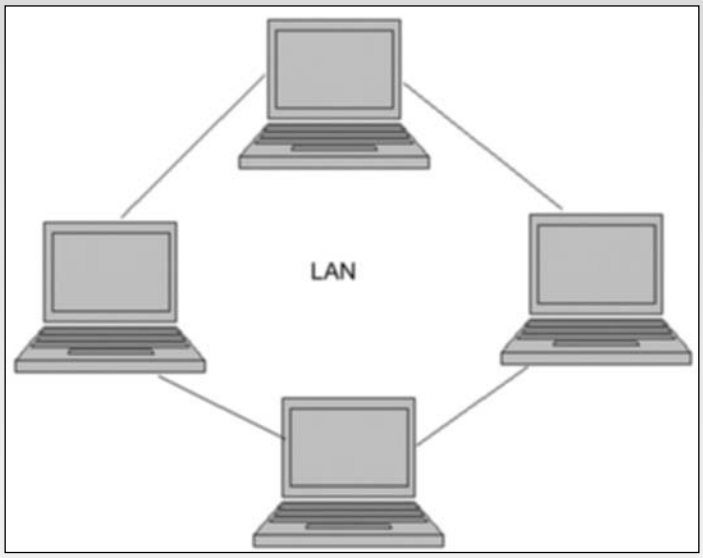
Types of Networks

The various types of networks are:

- Local Area Networks (LANs)
- Wide Area Networks (WANs)
- Metropolitan Area Networks (MANs)
- International Network (Internet)
- Intranet

Local Area Network (LAN)

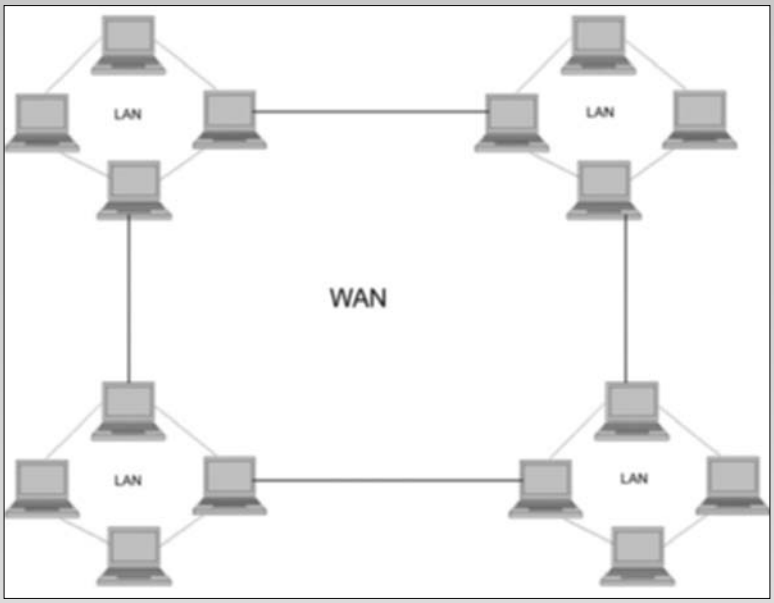
LAN is a group of computers that are connected in a small area such as building, home, etc. Through this type of network, users can easily communicate with each other by sending and receiving messages. LAN is generally used for connecting two or more personal computers through some medium such as twisted pair, coaxial cable etc. Though the number of computers connected in a LAN is limited, the data is transferred at an extremely faster rate.



LAN

Wide Area Network (WAN)

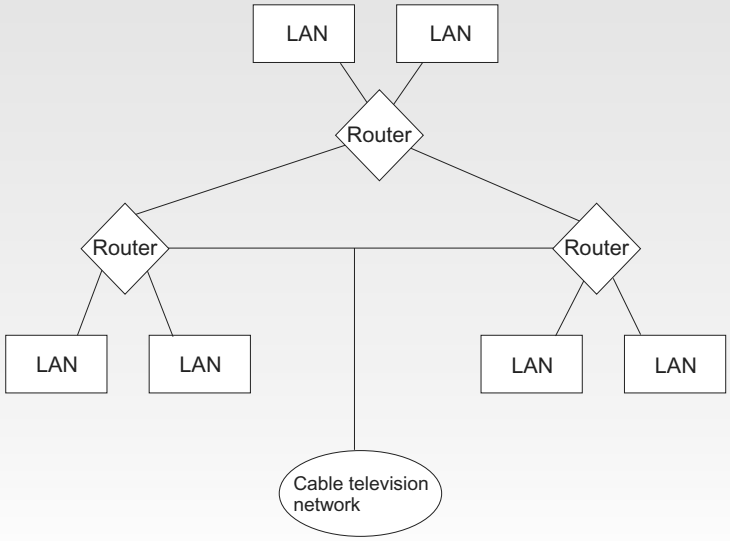
WAN is a group of computers that are connected in a large area like a country or across a continent. WAN is generally used for connecting two or more LANs together. In a WAN, data is transferred at slow rate as compared to a LAN.



A WAN system

Metropolitan Area Network (MAN)

MAN is a network of computers that covers a large area like a city. The size of a MAN lies between that of LAN and WAN, generally covering a distance of 5 to 50 Kms. MAN is generally owned by private organizations. One of the most common examples of MAN is cable television network within a city. A network device known as router is used to connect the LANs together. The router directs the information packets to their desired destinations.



A typical MAN system

Internet

Internet is a global area network that allows computers connected over the network to share resources and information using different protocols. It is basically a network of networks across the globe. Users at different locations can very easily communicate with each other via the Internet. They can access data and information from other computers if they are permitted to do so. The Internet basically uses a set of protocols such as Transmission Control Protocol/ Internet Protocol (TCP/IP) for transferring the data over the Internet. Some of the typical services provided by the Internet to its users are e-mail, instant messaging, video conferencing, ecommerce, etc.

Intranet

Intranet is a private network, which is confined to a single organization only. This type of computer network allows only the internal users of the organization to share the resources. However, the users outside the organization can also access the Intranet but only if they are authorized to do so.

The concept of Intranet is also used for sharing the company's information amongst the employees. Certain protocols such as TCP/IP, HTTP etc are used by Intranet for enabling the communication between the computer systems.

15. Write short notes on structuring the networks. (AU, TIR, Dec 10-Jan 11)

Ans:

Network Topology

Network topology refers to the arrangement of computers connected in a network through some physical medium such as cable, optical fibre etc. Topology generally determines the structure of the network and the communication path between the various computers (nodes) of the network. The various types of network topologies are as follows:

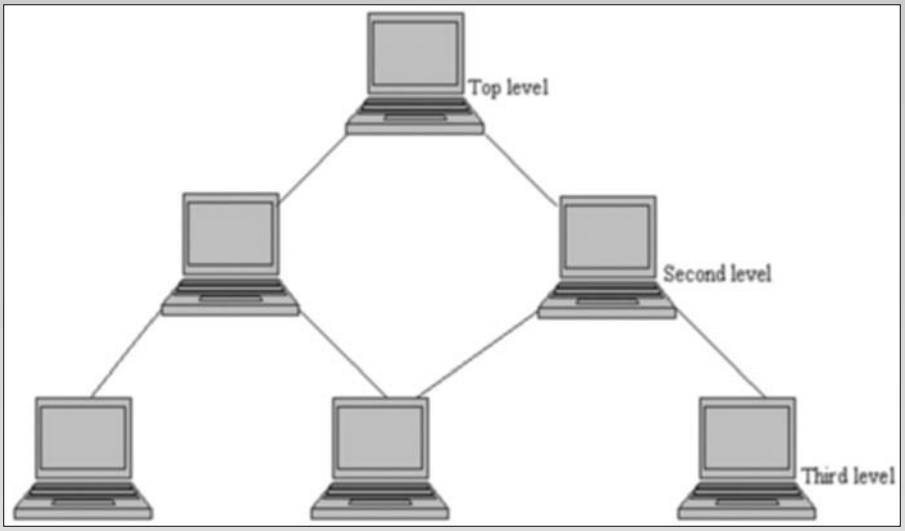
- Hierarchical topology
- Bus topology
- Star topology
- Ring topology
- Mesh topology
- Hybrid topology

Hierarchical Topology

The hierarchical topology also known as tree topology is divided into different levels connected with the help of twisted pair, coaxial cable or fibre optics. This type of topology is arranged in the form of a tree structure in which top level contains parent node (root node), which is connected with the child nodes in the second level of hierarchy with the point-to-point link. The second level nodes are connected to the third level nodes, which in turn are connected to the fourth level nodes and so on. Except the top-level node, each level node has a parent node.

The number of point-to-point links in the hierarchical type of topology is generally one less than the total number of nodes in the structure. The hierarchical topology is symmetrical,

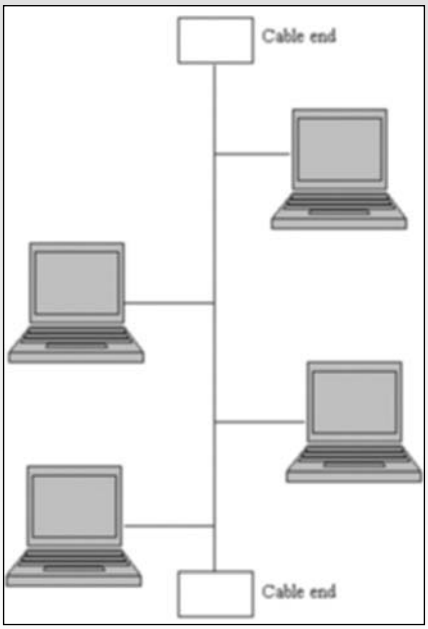
having a fixed branching factor, f, associated with each node. The branching factor is the number of point-to-point links between the levels of hierarchy. The following figure shows the arrangement of computers in hierarchical topology.



The hierarchical topology

Linear Bus Topology

In the linear bus topology, all the nodes are connected to the single backbone or bus with some medium such as twisted pair, coaxial cable etc. When a node wants to communicate with the other nodes in the network, it simply sends a message to the common bus. All the nodes in the network then receive the message but the node for which it was actually sent only processes it. The other nodes discard the message. The following figure shows the arrangement of computers in the linear bus topology.

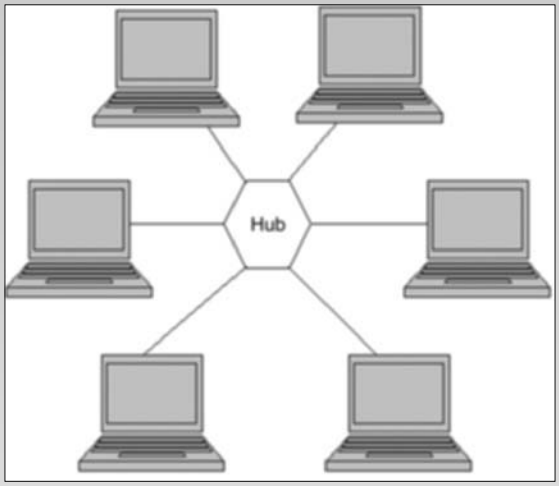


A linear bus topology

Star Topology

In the star topology, all the nodes are connected to a common device known as hub. Nodes are connected with the help of twisted pair, coaxial cable or optical fibre. When a node wants to send a message to the other nodes, it first sends the message to the hub, which in turn forwards the message to the intended node. Each node in the network is connected with a point-to-point link to the centralized hub. The

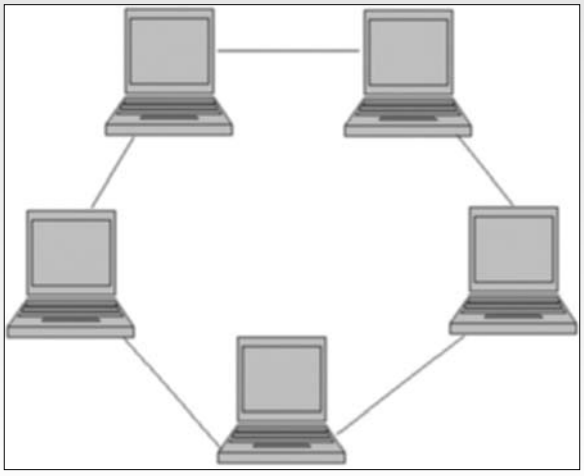
task of hub is to detect the faulty node present in the network. On the other hand, it also manages the overall data transmission in the network. The following figure shows the arrangement of computers in star topology.



A star topology

Ring Topology

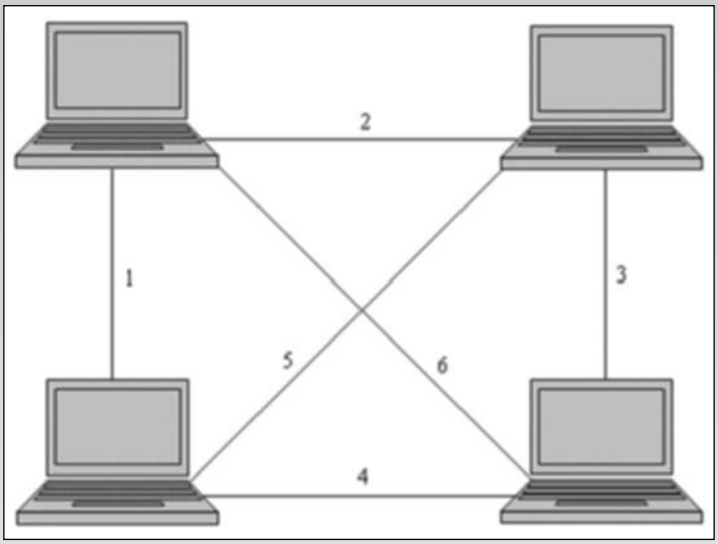
In the ring topology, the nodes are connected in the form of a ring with the help of twisted pair. Each node is connected directly to the other two nodes in the network. The node, which wants to send a message, first passes the message to its consecutive node in the network. Data is transmitted in the clockwise direction from one node to another. The following figure shows the arrangement of computers in the ring topology. Each node incorporates a repeater, which passes the message to next node when the message is intended for another node.



A ring topology

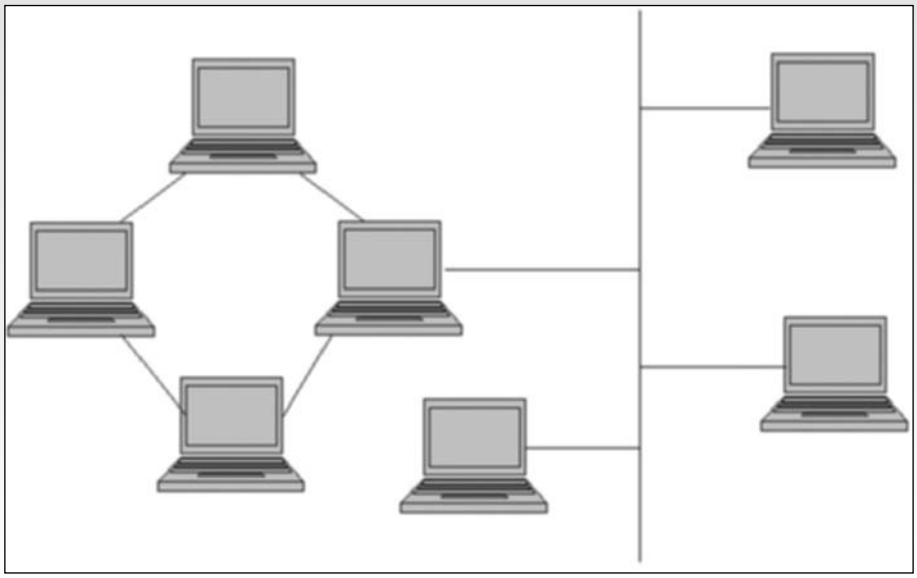
Mesh Topology

In mesh topology, each computer is connected to every other computer in point-to-point mode as shown in figure below. For instance, if we have four computers then we must have six links. If we have n computers, we must have n(n-1)/2 links. A message can take several possible paths to reach a destination.



Mesh topology

Hybrid Topology



A hybrid topology

The hybrid topology is the combination of multiple topologies, used for constructing a single large topology. The hybrid topology is created when two different network topologies are interconnected. If two ring topologies are connected then the resultant topology is not the hybrid topology. On the other hand, if the ring topology is connected to the bus topology then the resulting topology is called the hybrid topology. This topology generally combines the features of the two topologies and is therefore more effective and efficient than the individual topologies. The following figure shows a typical arrangement of computers in hybrid topology.

16. What is Cable modem? Explain how cable modem works. (AU, TIR, Dec 10-Jan 11)

Ans:

Cable Modem

Cable modem is a device that helps a computer to connect to the Internet through cable TV network. Since, cable TV network carries data through coaxial cables; the primary job of a cable modem is to convert the digital signals from the computer into a format suitable for RF transfer and vice versa.

How Cable Modem Works?

The following points describe the functioning of a cable modem:

- One end of the cable modem is connected to the coaxial cable through a one-to-two splitter. The other end is connected to the PC through Ethernet or USB port.
- The one-to-two splitter allows the modem as well as the set top box to be connected in parallel; thus allowing both Internet as well as TV to be used simultaneously.
- At the service provider's end, cable modem termination system (CMTS) is used for demodulating the received signal and passing it on to the fiber optic based infrastructure setup.
- CMTS also receives the data from the Internet and modulates and switches it back to the requesting cable modem.
- Typically, the data transmission happens through a 6 MHz RF channel.
- The upstream data speed through a cable modem is lower than the downstream data speed as a result, it is considered as asymmetrical modem.

17. Discuss about the advantages of Broadband connections in detail. (AU, TIR, Dec 10-Jan 11)

Advantages of Broadband Connections

The various advantages of broadband connection are:

- It allows fast access to the Internet.
- Files, updates, etc. are downloaded very quickly.
- It allows faster streaming of video and music.
- It does not affect the phone line (DSL) or cable TV (cable Internet) while accessing the Internet.
- It enhances the overall online experience. For e.g. VOIP streaming and online gaming is much faster.
- Its billing is not done on time-basis; instead it is entirely usage-based.

- Certain broadband-based services such as videoconferencing, live meeting, etc. are key business enablers.
- It also helps in the delivery of e-learning programs.

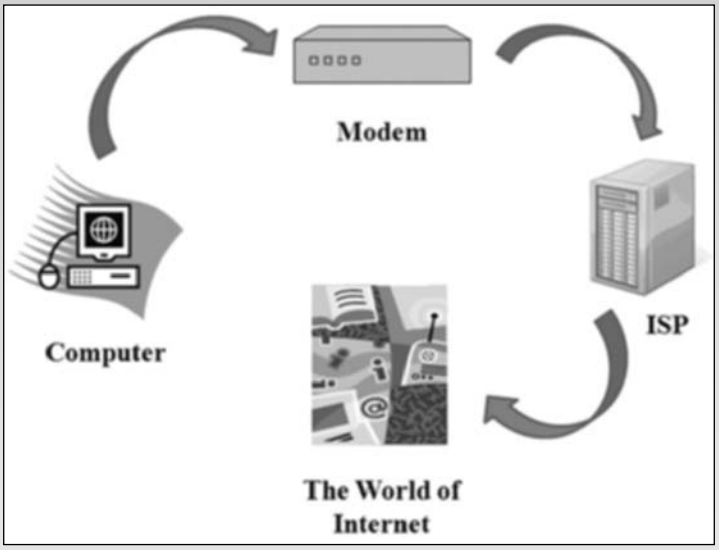
18. Explain the various steps in software development. (AU, IRI, Jan 2011)

Ans: Refer Section 2.8.

19. Explain the steps needed to get connected to internet with its connectivity diagram. (AU, IRI, Jan 2011)

Ans:

Getting Connected to the Internet

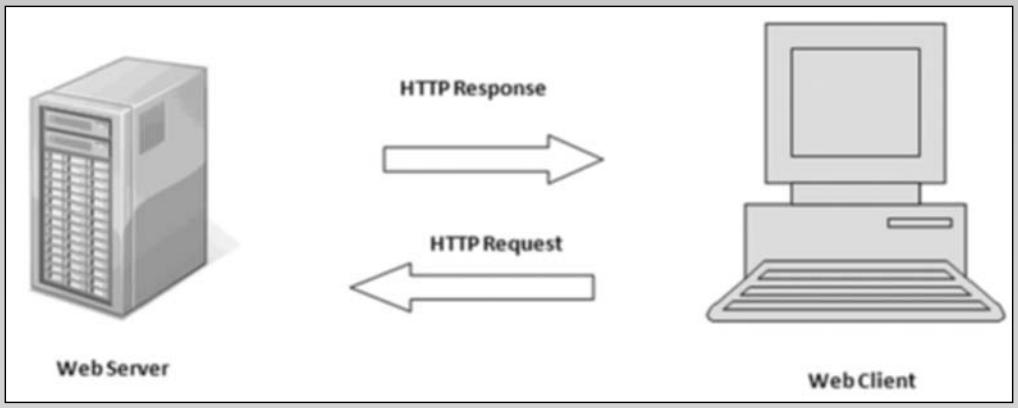


The steps involved in connecting to the Internet are:

1. Computer initiates a connection request.
2. Modem helps in transmitting the request to the Internet Service Provider (ISP).
3. ISP authenticates and processes the request and connects the requesting computer to the Internet.

These are generic steps that are followed while getting connected to the Internet; they may change on case-by-case basis. For instance, if a user is trying to connect to the Internet through cable network then the simple modem is replaced with a cable modem while the ISP is replaced with a cable service provider.

Once connected, a computer sends requests to online Web servers for accessing the World Wide Web (WWW), as shown below:



The steps involved in accessing a Web page through WWW are:

1. User types the URL of the Web page on the Web browser.
2. An HTTP request is sent to the Web address pointed by the URL.
3. The Web server accepts the HTTP request and after validation generates the response object.
4. Based on the Web server's response the Web page is rendered by the Web browser on the client's computer.
5. If the user clicks a hyperlink on the Web page then an HTTP request is sent to the Web address pointed by that hyperlink.

PROBLEM SOLVING AND OFFICE AUTOMATION

3

CHAPTER OBJECTIVES

In this chapter, we will learn:

1. Various steps performed while developing a computer program.
2. The techniques of solving a problem using algorithm, flowcharts and pseudocodes.
3. Various control structures like sequence structure, selection structure, etc.
4. Various application software packages like educational software, product engineering software, etc.
5. The four key MS office packages: MS Word, MS PowerPoint, MS Excel and MS Access.

CHAPTER OUTLINE

3.1 INTRODUCTION

A computer program is basically a set of logical instructions, written in a computer programming language that tells the computer how to accomplish a task. The process of computer program development involves a series of standard steps that realize the solution of a real-world problem into a computer program. The process of program development starts

with identifying the problem first. Once a problem is well understood and documented, a series of problem-solving techniques like algorithms, flowcharts and pseudocodes are carried out in arriving at the most efficient solution.

Although there are a number of software vendors in the market, the main driving force behind the software revolution is the Microsoft Corporation. Microsoft today has a suite of software packages that meet many of the standard application requirements of most organisations.

This software suite, popularly known as Microsoft Office, includes the following application packages:

- **Microsoft Word—**developed in 1983, and it provides powerful tools for creating and manipulating word-processing documents.
- **Microsoft Excel—**developed in 1985, and it enables to create detailed spreadsheets for viewing and collaboration.
- **Microsoft PowerPoint—**developed in 1988, and it provides a complete set of tools for creating presentations.
- **Microsoft Access—**developed in 1992, and it gives powerful tools for creating and managing databases.

In this chapter, we will discuss briefly how to use various features of these packages.

3.2 PLANNING THE COMPUTER PROGRAM

Whenever a user wants to use a computer for solving a problem, he/she has to perform various interrelated tasks in a systematic manner. A user cannot get the solution of a problem by simply providing input to the computer without preparing the base for solving the problem. The working process of a computer is similar to the human mind, which first analyses the complete situation of a problem, its causes and its parameters, and then decides the way to solve the problem on the basis of available parameters. All the activities, which have to be performed by a user in order to solve a problem using computer, are grouped into three phases:

- Identifying the purpose
- Developing a program
- Executing the program

1. Identifying the purpose It is the first stage of problem solving using a computer. It basically focuses on understanding the problem. In this stage, two basic activities are performed by the user. These activities are as follows:

- **Identifying parameters and constraints** A user has to identify the role of different parameters in solving the problem, i.e. the user must have the knowledge about the relation between the various parameters and the problem itself. After identifying the problem and its parameters, the user has to identify the associated constraints that need to be considered in order to generate an accurate solution of the problem. The identification of parameters and constraints help in choosing the most appropriate method to solve the problem.

- **Collecting information** After analysing the problem and choosing the solution method, a user has to collect the information related to the identified parameters of the problem. In order to collect the information, a user can use the documents and reports pertaining to the previous versions of the problem. The collected information helps in designing the layout of the output or solution of the problem.

2. **Developing a program** After analysing the problem, a user has to plan for developing the program, which will provide the solution of the program after execution. A program includes multiple instructions having a specific syntax. For developing a program, a user has to perform the following activities:

- **Identifying the logical structure** It is the most important activity in which a user prepares the logical structure of the program by analysing the various tasks that need to be performed for solving the problem. In order to prepare the logical structure of a program, a user performs the following task:
  (i) Writing algorithm to list the various steps.
  (ii) Drawing flowchart to represent the flow of information.
  (iii) Writing pseudocode to specify the programming specifications.
- **Writing the computer program** After preparing the logic, a user has to write the program code in a particular programming language. The program code should be syntactically and semantically correct in order to generate the desired result.
- **Debugging the program** After writing the complete program, a user has to apply the debugging techniques for removing any possible errors in the program. Several programming environments provide debugging tools that aid the users in effectively and efficiently removing the errors in a program.

3. Executing the program After developing an error-free program, it needs to be executed in order to view the solution of the original problem.

3.3 PROBLEM SOLVING

Problems that can be solved through a computer may range in size and complexity. Since computers do not possess any commonsense and cannot make any unplanned decisions, the problem, whether simple or complex, has to be broken into a well-defined set of solution steps for the computer to implement. Problem solving is the process of solving a problem in a computer system by following a sequence of steps. The major steps that we need to follow for solving a problem are as follows:

1. **Preparing hierarchy chart** A hierarchy chart shows the top–down solution of a problem. In case of large problems, we can break them into parts representing small tasks, prepare several algorithms and later combine them into one large algorithm.
2. **Developing algorithm** An algorithm is a sequence of steps written in the form of English phrases that specify the tasks that are performed while solving a problem. It involves identifying the variable names and types that would be used for solving the problem.

3. **Drawing flowchart** A flowchart is the graphical representation of the flow of control and logic in the solution of a problem. The flowchart is a pictorial representation of an algorithm.
4. **Writing pseudocode** Pseudocode is quite similar to algorithms. It uses generic syntax for describing the steps that are to be performed for solving a problem. Along with the statements written using generic syntax, pseudocode can also use English phrases for describing an action.

3.3.1 Hierarchy Chart

Hierarchy chart is a solution approach that suggests a top–down solution of a problem. We very often come across large problems to be solved using computers. It may be very difficult to comprehend the solution steps of such large problems at one go. In such situations, we can decompose the problem into several parts, each representing a small task, which is easily comprehensible and solvable. We can then prepare solution steps for each task independently and later combine them into one large solution algorithm. Fig. 3.1 illustrates a hierarchy chart for computing pay of an employee in an organisation.

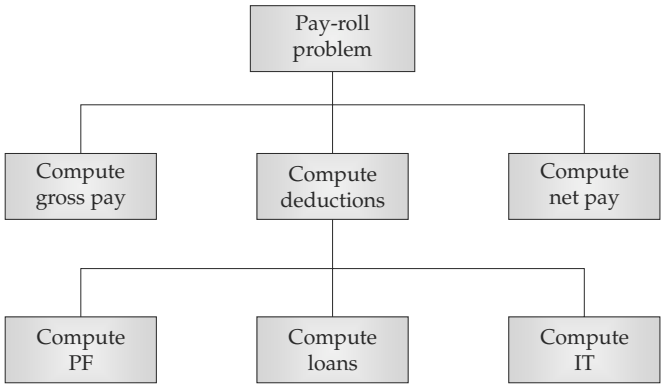


Fig. 3.1 Hierarchy chart for pay-roll problem

Since the chart graphically illustrates the structure of a program, it is also known as a structure chart. While developing a computer program, we may treat each subtask as a module and prepare computer code for testing independently. This approach is popularly known as modular programming. Note that the hierarchy chart does not provide any detail about program logic. We have to use the tools discussed further to prepare logic for each task.

3.3.2 Algorithms

Algorithms help a programmer in breaking down the solution of a problem into a number of sequential steps. Corresponding to each step, a statement is written in a programming language; all these statements are collectively termed as a program. The following is an example of an algorithm to add two integers and display the result:

```
                Algorithm to add two integers and display the result
Step 1 – Accept the first integer as input from the user.
         (integer1)
Step 2 – Accept the second integer as input from the user.
         (integer2)
Step 3 – Calculate the sum of the two integers.
         (integer3 = integer1 + integer2)
Step 4 – Display integer3 as the result.
```

There is a time and space complexity associated with each algorithm. Time complexity specifies the amount of time required by an algorithm for performing the desired task. Space complexity specifies the amount of memory space required by an algorithm for performing the desired task. When solving a complex problem, it is possible to have more than one algorithm to provide the required solution. The algorithm that takes less time and requires less memory space is the best one.

Characteristics of an algorithm The various characteristics that are necessary for a sequence of instructions to qualify as an algorithm are:

- The instructions must be in an ordered form.
- The instructions must be simple and concise. They must not be ambiguous.
- There must be an instruction (condition) for program termination.
- The repetitive programming constructs must possess an exit condition. Otherwise, the program might run infinitely.
- The algorithm must completely and definitely solve the given problem statement.

Qualities of a good algorithm Typically, an algorithm is considered as good, if:

- It uses the most efficient logic to solve the given problem statement. (Time complexity)
- It uses minimal system memory for its execution. (Space complexity)
- It is able to generate the most accurate results for a wide range of input set.
- It is easy to implement in the form of a program.
- It is designed with standard conventions so that others are able to easily modify it while adding additional functionality.

3.3.3 Flowcharts

A flowchart can be defined as the pictorial representation of a process, which describes the sequence and flow of the control and information in a process. The flow of information is represented in a flowchart in a step-by-step form. This technique is mainly used for developing business workflows and solving problems using computers.

Flowchart uses different symbols for depicting different activities, which are performed at different stages of a process. The various symbols used in a flowchart are as follows:

- **Start and end** It is represented by an oval or a rounded rectangle in a flowchart. It is used to represent the starting and the ending of a process. Every process starts and ends at some point, so and therefore a flowchart always contains one start as well as one end point. Figure 3.2 shows the start and the end symbols used in a flowchart.

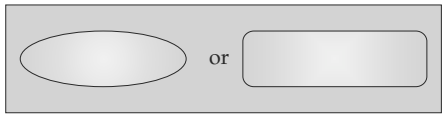


Fig. 3.2 Start and end symbol

- **Input or output** It is represented by a parallelogram in a flowchart. It is used to represent the inputs given by the user to the process and the outputs given by the process to the user. Figure 3.3 shows the input or output symbol.

Fig. 3.3 Input or output symbol

- **Action or process** It is represented by a rectangle. It represents the actions, logics and calculations taking place in a process. Figure 3.4 shows the action or process symbol.

Fig. 3.4 Action or process symbol

- **Decision or condition** It is represented by a rhombus or a diamond shape in a flowchart. It represents the condition or the decision-making step in the flowchart. The result of the decision is a Boolean value, which is either true or false. Each of these values takes the flow of the program to a certain point, which is shown with the help of arrows. Figure 3.5 shows the decision or condition symbol.

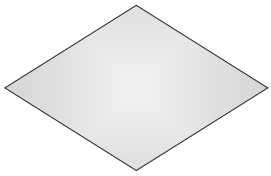


Fig. 3.5 Decision or condition symbol

- **Arrow** It is represented by a directed line in a flowchart. It represents the flow of process and the sequence of steps in the flowchart. It guides the process about the direction and the sequence, which is to be followed while performing the various steps in the process. Figure 3.6 shows the arrow symbol.

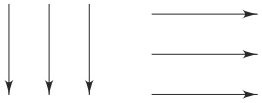


Fig. 3.6 Arrow symbol

- **Connector** It is represented by a circle in a flowchart. It represents the continuation of the flow of steps when a flowchart continues to the next page. A character, such as an alphabet (a to z) or a symbol (a, b or c), etc. can be placed in the circle at the position where the flow is broken and the same character is also placed in the circle at the position from where the flowchart continues. Figure 3.7 shows the connector symbol.

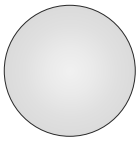


Fig. 3.7 Connector symbol

In order to understand how a flowchart represents the flow of information, consider an example of flowchart in which addition of two numbers is represented (Fig. 3.8).

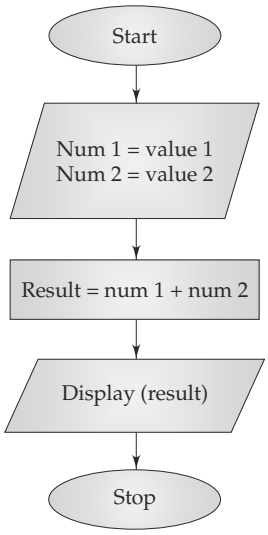


Fig. 3.8 Flowchart of addition of two numbers

Advantages of using a flowchart Some of the key advantages of using a flowchart in program design are:

- It helps to understand the flow of program control in an easy way.
- Developing program code by referring its flow chart is easier in comparison to developing the program code from scratch.
- It helps in avoiding semantic errors.
- Any concept is better understood with the help of visual representation. This fact also holds true for flowcharts. It is easier to understand the pictorial representation of a programming logic.
- A flowchart acts as documentation for the process or program flow.
- The use of flowcharts works well for small program design.

Disadvantages of using a flowchart Flowcharts also have certain limitations, such as:

- For a large program, the flow chart might become very complex and confusing.
- Modification of a flowchart is difficult and requires almost an entire rework.
- Since flowcharts require pictorial representation of programming elements, it becomes a little tedious and time consuming to create a flowchart.
- Excessive use of connectors in a flowchart may at times confuse the programmers.

3.3.4 Pseudocodes

Analysing a detailed algorithm before developing a program is very time consuming. Hence, there arises a need of a specification that only focuses on the logic of the program. Pseudocodes serve this purpose by specifying only the logic, which is used by the programmer for developing a computer program.

Pseudocode is not written using specific syntax of a programming language, rather it is written with a combination of generic syntax and normal English language. It helps the programmer understand the basic logic of the program after which it is the programmer's choice to write the final code in any programming language. An example of a pseudocode to add two numbers and display the result is as follows:

```
A pseudocode to add two numbers and display the result
Define: Integer num1, num2, result.
Input: Integer num1.
Input: Integer num2.
Sum: result = num1 + num2
Output: Display(result).
```

After the pseudocode for a computer program has been written, it is used to develop the source code for the computer program. The source code is developed using a programming language, which can be an assembly language or a high-level programming language. After

the source code has been written, the programmer detects and eliminates any errors in the program so that the program generates the desired output on execution.

Advantages of pseudocodes Some of the key advantages of using a flowchart in program design are:

- Pseudocode is easy to comprehend as it used English phrases for writing program instructions.
- Developing program code using pseudocode is easier in comparison to developing the program code from scratch.
- Developing program code using pseudocode is also easier in comparison to developing the program code from flowchart.
- The pseudocode instructions are easier to modify in comparison to a flowchart.
- The use of pseudocode works well for large program design.

Disadvantages of pseudocodes Pseudocodes also have certain limitations, such as:

- Since, pseudocode does not use any kind of pictorial representations for program elements; it may at times become difficult to understand the program logic.
- There is no standard format for developing a pseudocode. Therefore, it may become a challenge to use the same pseudocode by different programmers.
- Pseudocodes are at a disadvantage in comparison to flowhcarts when it comes to understanding the flow of program control.

3.4 STRUCTURING THE LOGIC

While writing the pseudocode for a problem, it is necessary to define all the logics used in the pseudocode for developing the program. Pseudocode of a problem should be able to describe the sequence of execution of statements and procedures specified in the program. The sequence of the execution of instructions determines the basic structure of a program or the logic used to solve a problem. The basic structure of a program comprises different sets of the statements, whose execution is dependent on some conditions and decisions. These conditions and decision-making statements are specified in a control structure. Depending upon the sequence of the execution of the statements, the control structures are categorised as follows:

- **Sequence structure** The execution of the statements in a sequence structure is done sequentially, i.e. all the statements are executed in the same order as they are written in the program.
- **Selection structure** In the selection structure, two sets of statement blocks are written in a program along with one or more conditions. The execution of a particular block's statements occurs only if the conditional statement specified at the beginning of the block is true. A selection structure is also known as the branching structure.
- **Repetition structure** In the repetition structure, a block of two or more instructions is specified along with a conditional statement. The execution of these instructions is

repeated many times if the conditional statement is true. This structure is also known as the looping structure.

We must incorporate these program constructs into the program design whether as a flowchart or as a pseudocode. We can also combine the constructs, if necessary. For example, a selection structure can be a part of a looping structure.

3.4.1 Sequence Structure

In a sequence structure, multiple statements are written in a simple sequence in the program. The execution of these statements is not affected by any condition. Generally, sequence structure is used for performing simple computations, which do not require any decision-making. Figure 3.9 shows the representation of statements in the sequence structure.

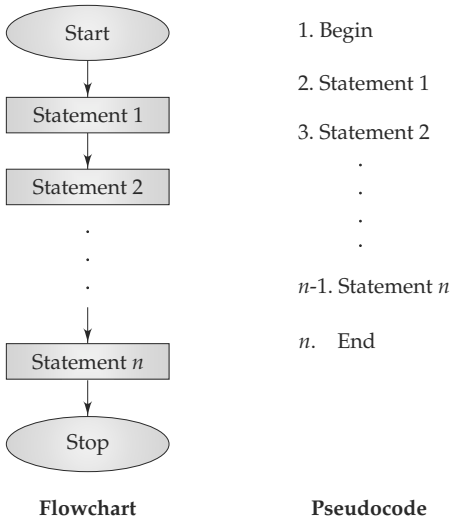


Fig. 3.9 Representation of statements in the sequential structure

3.4.2 Selection Structure

In the selection structure, the execution of a set of statements is done according to a pre-specified condition. The selection structure is also known as decision-making structure because the decision to execute a particular set of statements is made on the basis of the conditional statement. The selection structure is categorised as follows:

- **If–Then** In this selection structure, If and Then clauses are used to represent a condition as well as a set of statements. In the If clause, the conditional statement is written, while in the Then clause the set of statements to be executed is specified. The execution of the statements specified in the Then clause occurs only if the condition is true.

- **If–Then-Else** This selection structure is quite similar to the If–Then selection structure. The only difference between the two is that in If–Then–Else selection structure, two sets of statements are specified. One set of statements is represented in the Then clause and another is in the Else clause. If the condition given in the If clause is true, then all the statements specified in the Then clause are executed; otherwise statements given in the Else clause are executed.
- **Case Type** In this selection structure, multiple sets of statements are specified. Each block of statements is associated with a value. The selection of a particular set of statements is made on the basis of the value of the variable given at the beginning of the selection structure.

Figure 3.10 shows the representation of statements in the If–Then–Else selection structure.

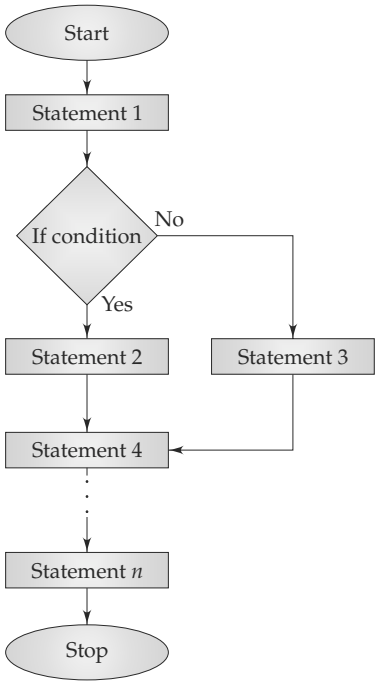


Fig. 3.10 Representation of statements in the If–Then–Else selection structure

3.4.3 Repetition Structure

In the repetition structure, only one set of multiple statements is specified. The same set of statements is executed several times on the basis of the condition specified along with the structure. Various types of repetition structure are as follows:

- **Do–while** In the Do–while structure, a set of statements is given in the Do block and a condition is given in the While block. The statements given in the Do block are executed till the given condition is true. At each instance of execution of block statements, the condition is checked. If the condition is true, then only the block statements are executed; otherwise the repetition structure is terminated.
- **Repeat–until** The Repeat–until structure is opposite to the Do–while repetition structure. In this structure, the repetitive execution of statements given in the Repeat clause occurs only when the condition given in the Until clause is false.

Figure 3.11 shows the representation of statements in the Do–while repetition structure.

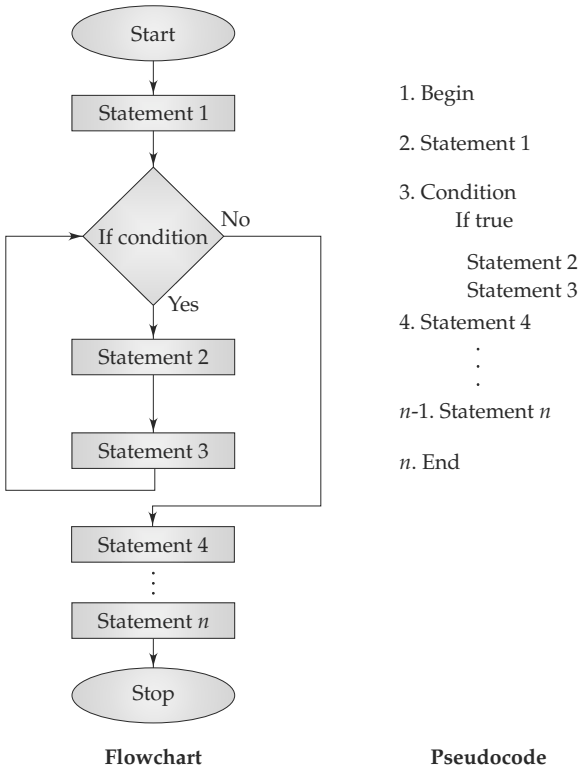


Fig. 3.11 Representation of the statements in the Do–while repetition structure

3.5 APPLICATION SOFTWARE PACKAGES

Application software is a software that helps a user to perform a specific task on the computer. A few examples of application software are MS Word, MS Excel, MS PowerPoint, etc. Application software are broadly classified into two categories—general application software and customised application software.

The generalised application software are designed keeping the general requirements of the users in mind. Thus, these software are capable of fulfilling the requirements of a large number of users simultaneously. However, customised application software are those which are designed keeping the requirements of a specific group of users in mind. These software are also referred as tailor-made application software as they serve the custom requirements of the users.

Many application software are bundled together such that they can be collectively used to accomplish some specific tasks. These software are collectively referred as application packages or application suites. Microsoft Office is one such application package comprising word processor, spreadsheet package and other application software.

The application software are also classified on the basis of their usage. The following are some of the key classifications of application software:

- Enterprise software
- Enterprise infrastructure software
- Educational software
- Product engineering software
- Content access software
- Simulation software
- Information worker software
- Media development software

3.6 INTRODUCTION TO OFFICE PACKAGES

Office packages, as the name suggests, correspond to those software packages that help perform routine office-related tasks. Some of the common tasks that are typically performed in an office environment are:

- Preparing documents, letters, memos, invoices, etc.
- Preparing worksheets, financial statements, etc.
- Preparing company presentations, product presentations, etc.

There common requirements were identified by Microsoft and they launched the MS Office suite comprising the following:

- Word processor
- Spreadsheet software
- Presentation software
- Database package

Even though, Microsoft leads the market of office packages, there are some other similar office packages available for the users to choose from. These include Open Office, Google Docs, Lotus SmartSuite, etc.

3.7 MS WORD

MS Word is an application software that can be used to create, edit, save and print personal as well as professional documents in a very simple and efficient manner. MS Word is an important tool of the MS Office suite that is mainly designed for word processing. Therefore, it is also referred as the word processing program.

MS Word is not the only word processing program available in the market. There are many other word processing applications available, such as Open Office Writer and Google Docs. However, MS Word is the most popular word processing program among all.

3.7.1 Accessing MS Word

For working in MS Word, we need to install MS Office in a computer system. After installing MS Office, we can start MS Word using any of the following two ways:

- Start menu
- Run command

Using Start menu We can start MS Word by performing the following steps using the Start menu:

1. Select Start → Programs → Microsoft Office, as shown in Fig. 3.12.

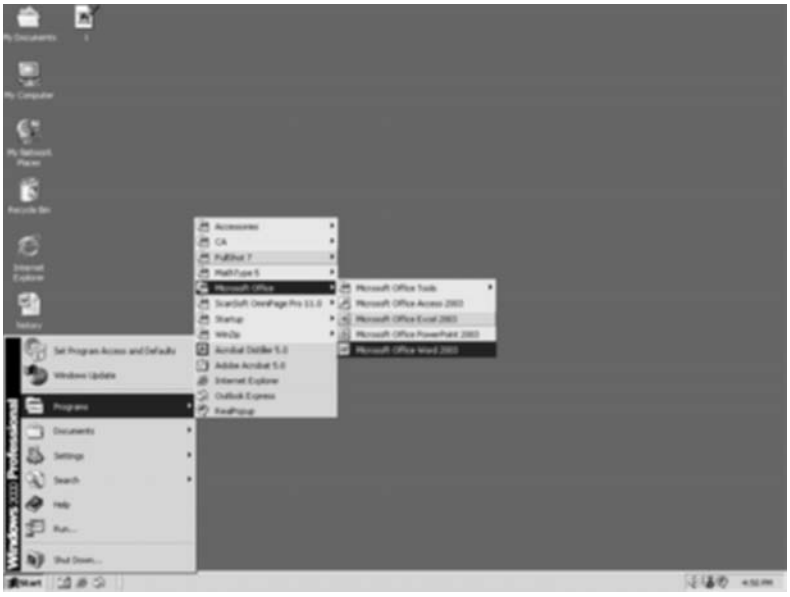


Fig. 3.12 Starting MS Word using the Start menu

2. Select the Microsoft Office Word 2003 option to display the Graphical User Interface (GUI) of MS Word, as shown in Fig. 3.13.

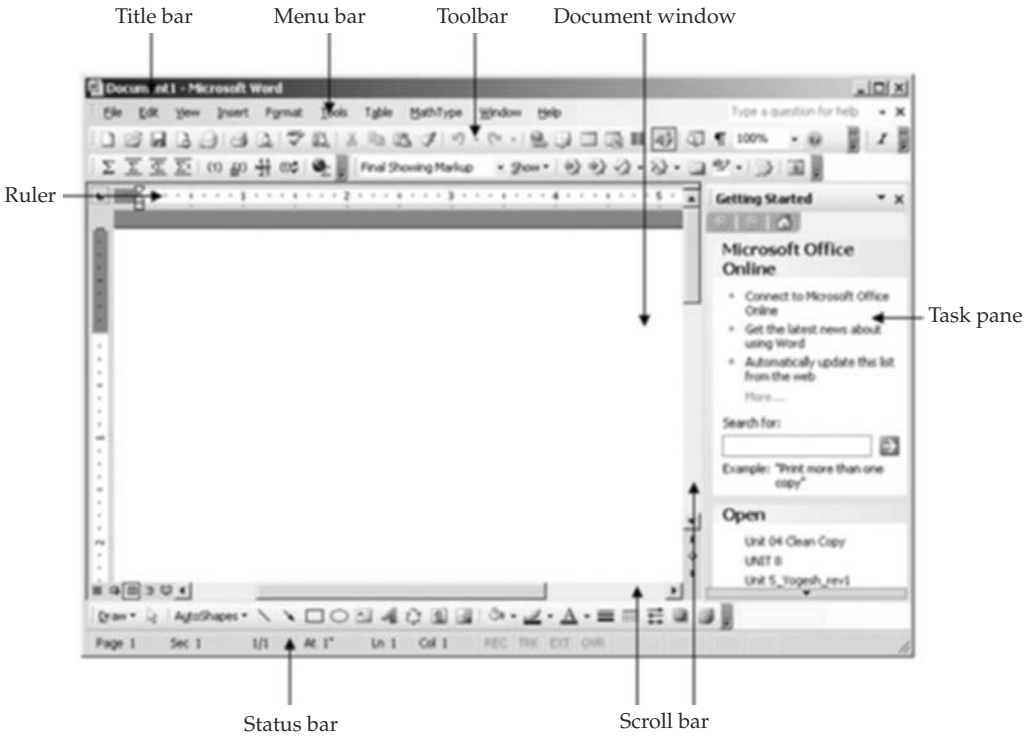


Fig. 3.13 **The Document1- Microsoft Word window**

Using Run command We can also start MS Word by performing the following steps using the Run command:

1. Select Start → Run to display the Run dialog box.
2. Type winword in the Open text box and click OK to display the Document1-Microsoft Word window.

Figure 3.13 shows the GUI of MS Word with the following major components:

- **Title bar** It is a horizontal bar at the top of the window that displays the name of the currently opened MS Word document. By default, MS Word assigns names to new documents as Document1, Document2, Document3 and so on. However, we can also save our document with some other name of our choice.
- **Menu bar** It is located right below the title bar. The Menu bar is used to house a set of commands that can be used to perform various operations such as opening a file, closing a file and creating a table. The Menu bar of MS Word contains the following menus:

- **File menu** It contains a set of commands that can be used to perform various file-handling operations. The file-handling operations in MS Word are usually known as document-handling operations. The various operations that can be performed using the options available in the File menu are opening a new document, opening an existing document, saving the current document, printing the current document, etc.
- **Edit menu** It contains a set of commands that can be used to perform various operations related to content editing and manipulation. The various operations that we can perform using the options available in the Edit menu are copying the currently selected text to clipboard, moving the currently selected text to some other location in the currently opened document, pasting the text from clipboard to the currently opened document, deleting the selected text, etc.
- **View menu** It contains a set of commands that can be used to display the document in different views. The different views available in MS Word are Normal, Web Layout, Print Layout, Document Map, Full Screen and Zoom. Apart from changing the view of a document, we can also perform various other tasks using the View menu like displaying or hiding the toolbar, setting the header and footer for a document and displaying or hiding the rulers.
- **Insert menu** It contains a set of commands that can be used to insert various objects such as clip art, auto shapes, organisation chart, word art and text box in a document to make it more attractive. Apart from these objects, we can also insert date and time, page numbers, symbols, page break and column break in the document.
- **Format menu** It contains a set of commands that can be used to alter the look and layout of the content present in the document. The various tasks that can be performed using the options available in the Format menu are changing the font type, font colour, font size and font style of the selected text, indenting a paragraph, inserting bullets and numbering in the document, etc.
- **Tools menu** It contains a set of commands that can be used to perform advanced operations in the MS Word document. The various tasks that can be performed using the options available in the Tools menu are checking and correcting spelling and grammatical mistakes, counting the number of words and characters, protecting a document and using mail merge, etc.
- **Table menu** It contains a set of commands that can be used to perform various operations related to the creation, modification and deletion of tables in a document.
- **Window menu** It contains a set of commands that can be used to perform various tasks related to the active window in which we are working. Using this menu, we can open a new window containing the same content as the active window, split the active window into different panes and arrange all the opened documents into separate windows in such a manner that all the windows can be viewed at the same time by the user.

  - **Help menu** It assists the user by providing information related to MS Word from various sources such as Office Assistant tool and Microsoft Office Web site, etc. In addition, it can also be used to detect and repair the errors contained in MS Word files.

- **Toolbar** It is located right below the menu bar. A number of toolbars are provided in MS Word for a quick and easy access to the various commands housed in the Menu bar. The three most commonly used toolbars in MS Word are as follows:
  - **Standard toolbar** It provides quick access to the various operations related to the file handling and content editing and manipulation. However, this is not the only purpose of this toolbar. There are certain icons present on this toolbar with the help of which we can perform many other tasks such as spelling and grammar check, applying the formatting of certain section to some other section in the currently opened document, and creating and editing tables.


NOTE: If the Standard toolbar is not visible in the MS Word window, then we need to select View → Toolbars → Standard to make it visible.

  - **Formatting toolbar** As its name implies, it is used to perform various operations related to the look and the layout of the document content. Using the icons available on this toolbar, we can change the font size, style and colour of the selected text, align the selected text to left, centre or right of the screen, create numbered and bulleted list, etc.


NOTE: If the Formatting toolbar is not visible in the MS Word window, then we need to select View → Toolbars → Formatting to make it visible.

  - **Drawing toolbar** It is located at the bottom of the screen just above the status bar. Using the different icons available on the drawing toolbar, we can draw and manipulate different types of graphics in a document. The various shapes that we can draw using the drawing toolbar are line, rectangle, oval, etc. We can also insert clip art, wordart and pictures in a document using this toolbar.


NOTE: If the Drawing toolbar is not visible in the MS Word window, then we need to select View → Toolbars → Drawing to make it visible.

- **Document window** It is the actual area in the MS Word window where we can enter text and draw graphics. The document window of MS Word can also be considered as a text editor because you can edit and apply different types of formatting to the text in this window. The document window is also called the text area of the MS Word window.

- **Status bar** It is located at the bottom of the MS Word window. The status bar provides some helpful information related to the document that we are currently working with. The information provided by the status bar includes:
  - Total number of pages in the document
  - Page number of the active page
  - Line number of the line of text where the cursor is currently positioned
  - Column number of the character in the line of text where the cursor is currently positioned
  - Status of various modes such as track mode, extend selection mode and overtype mode
- **Scroll bar** The term scroll bar usually refers to the horizontal and the vertical bars placed at the right and the bottom of the MS Word window. These bars allow the user to view those portions of the document that cannot fit on the screen at one time.
- **Ruler** It is located below the toolbars. The ruler bar in MS Word is used to set the alignment for the content in the document.


NOTE: If the ruler bar is not displayed in the MS Word window, then we can make it visible by selecting View → Ruler.

- **Task pane** It is a rectangular pane placed at the right side of the MS Word window. The Getting Started task pane appears by default. We can consider the task pane as an open menu that can be used to open a new as well as an existing document. Using the Getting Started task pane, we can open recent documents, create a new document and search for some information using the Microsoft Office Online tool. Apart from the Getting Started task pane, the other task panes that we can view are Assistance, Search Results, Clip Art, Clipboard, etc.


NOTE: If the Task Pane is not visible, then we can open it by selecting View → Task Pane or by pressing the Ctrl and F1 keys simultaneously.

3.7.2 Basic Operations Performed in MS Word

The following are the key operations that we can perform in MS Word:

- Creating a document
- Saving a document
- Editing a document
- Formatting a document
- Printing a document

Creating a document We can create a document by performing the following steps:

1. Open Document1-Microsoft Word window.
2. Type some text; say, "This is my word document", in the document window, as shown in Fig. 3.14.

Fig. 3.14 The Document1-Microsoft Word window with the entered text

Figure 3.14 shows the insertion point, which is actually a vertical blinking bar. In MS Word, the insertion point is more commonly known as a cursor. The insertion point usually defines the place in the document window from where we start typing the text.

Saving the document After we have finished creating a document, we need to save it at some appropriate location in the computer system for future reference.

To save a document, we need to perform the following steps:

1. Select File → Save to display the Save As dialog box, as shown in Fig. 3.15.

Figure 3.15 shows the Save As dialog box in which you can specify the document name and location where you want to save the document.

2. Select a location from the Save in list for saving the document.

NOTE: By default, the My Documents location appears in the Save in list.

Fig. 3.15 **The Save As dialog box**

3. Type a name, say mydoc, in the File name text box and click the Save button to save the document. The specified name, mydoc, appears in the title bar, as shown in Fig. 3.16.

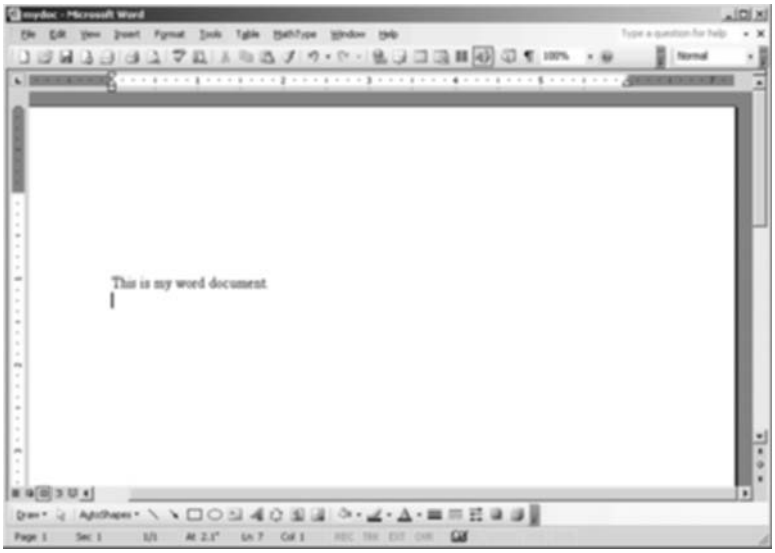


Fig. 3.16 **The mydoc-Microsoft Word document**

Editing the document Editing a document generally involves the operations such as selecting the text, moving and copying the text and deleting either the selected text or the entire text in the document window. For selecting the text, say "This is my word", in the mydoc document, we need to perform the following steps:

1. Open the mydoc-Microsoft Word window.
2. Set the insertion point before the word "This" in the document window, as shown in Fig. 3.17.

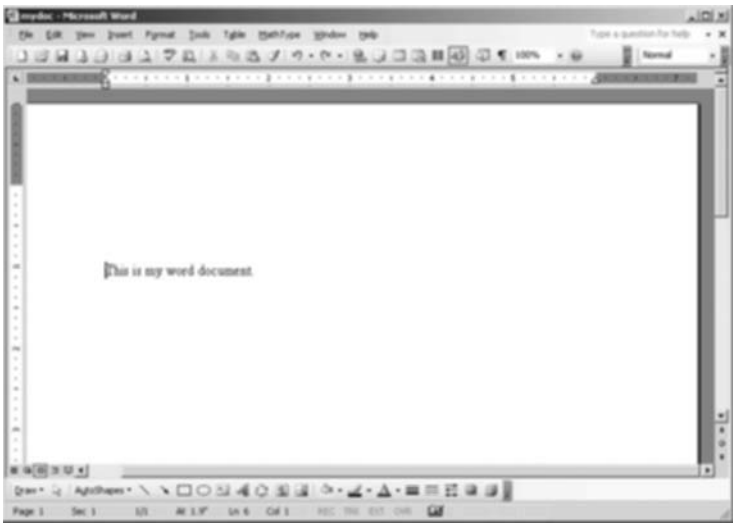


Fig. 3.17 **Placing the insertion point**

3. Press the left mouse button and drag the mouse pointer up to the desired level of the selection.
4. Release the left mouse button to complete the selection, as shown in Fig. 3.18.

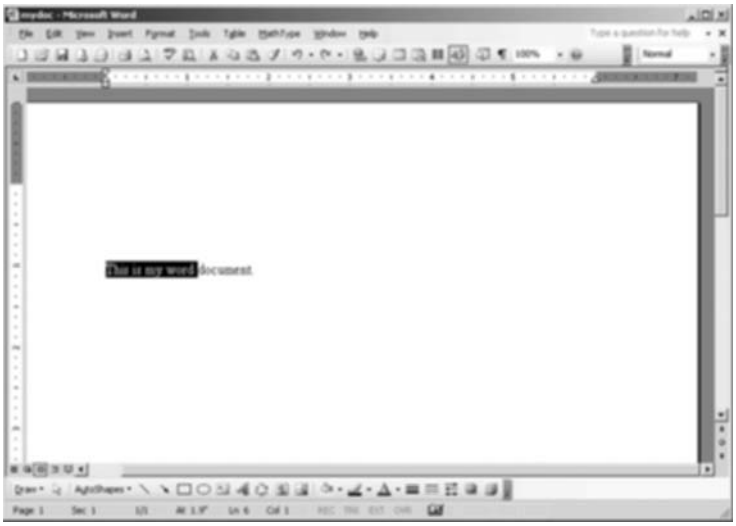


Fig. 3.18 **Selecting the desired text in the document window**

NOTE: We can select the entire content of the document by either selecting Edit → Select All option or by pressing the Ctrl and A keys simultaneously.

After selecting, we can copy the selected text by performing the following steps:

1. Right-click the selected text to display a shortcut menu
2. Select the Copy option to copy the selected text to the clipboard.

NOTE: We can also copy the selected text to the clipboard by either selecting Edit → Copy or by pressing the Ctrl and C keys simultaneously.

After copying, we can paste the selected text either in the same document window or in another document window by performing the following steps:

Set the insertion point where the selected text needs to be pasted, as shown in Fig. 3.19.

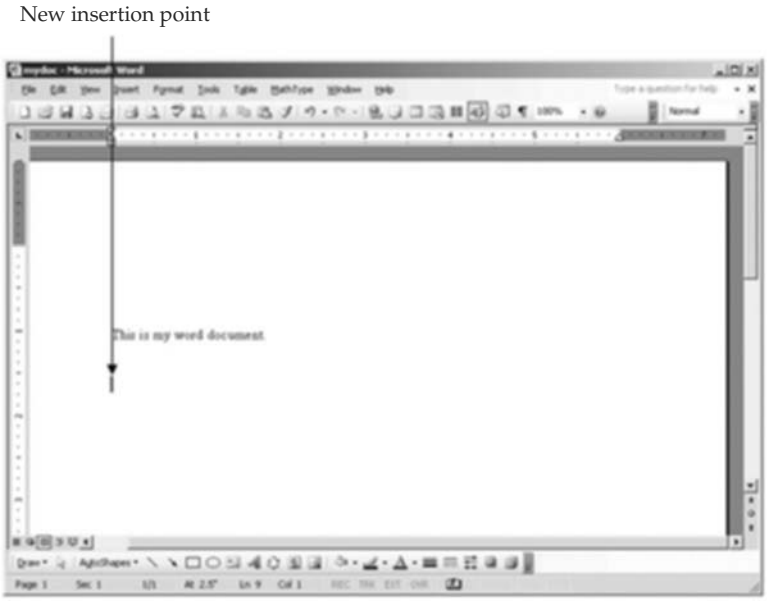


Fig. 3.19 Placing the insertion point for pasting the copied text

2. Right-click to display a shortcut menu and select the Paste option to paste the selected text at the new position in the document window, as shown in Fig. 3.20.

NOTE: We can also paste the selected text by either selecting Edit → Paste or by pressing the Ctrl and V keys simultaneously.

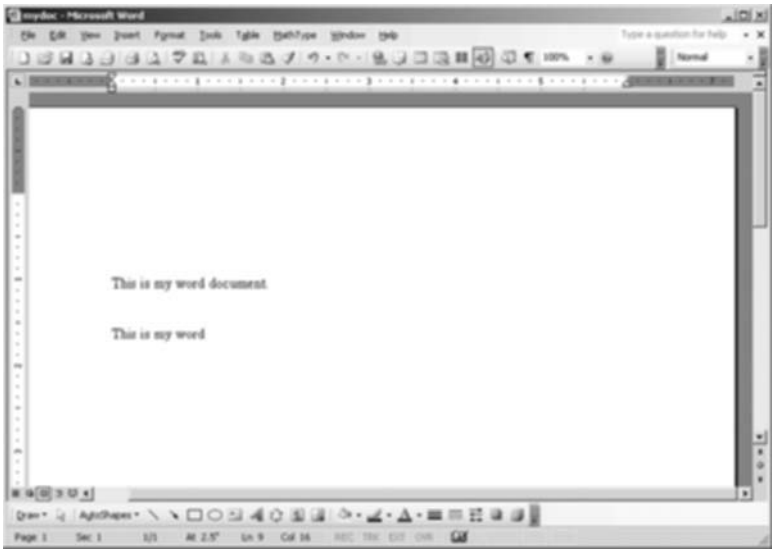


Fig. 3.20 **The mydoc-Microsoft Word containing the text pasted at new position**

Moving the text around the document window generally refers to the process of cutting the selected text from its original location and pasting it at some new place in the document window. To cut the selected text, select the Cut option from the shortcut menu that appears when we right-click on the selected text. Alternatively, we can also cut the selected text by selecting Edit → Cut option or by pressing the Ctrl and X keys simultaneously.

We can also delete either the selected text or the entire text of the document window. To perform the delete operation in the document window, select Edit → Clear → Contents. The content in the document window can also be deleted by pressing the Delete key.

NOTE: We can get back the most recently deleted text by selecting Edit → Undo Clear or by pressing the Ctrl and Z keys simultaneously.

Formatting the document The formatting of a document generally refers to the method of changing the layout and the design of the text according to the requirements. We can use the various options available in the Format menu to change the look and layout of the text in the document. Some of the options that can be used to format a document are as follows:

- Font
- Paragraph
- Bullets and Numbering

Font The Font option in the Format menu can be used to change the appearance of the selected text by assigning it a different font size, font style, font colour and font type. To change the font settings of the selected text, select Format → Font to display the Font dialog box, as shown in Fig. 3.21.

Fig. 3.21 **The Font dialog box**

Figure 3.21 shows the Font dialog box in which we can select various options for applying different formatting styles to the text in a document.

Paragraph The Paragraph option in the Format menu can be used to apply different types of formatting to the paragraphs in the document window. In order to apply paragraph formatting, select Format → Paragraph to display the Paragraph dialog box, as shown in Fig. 3.22.

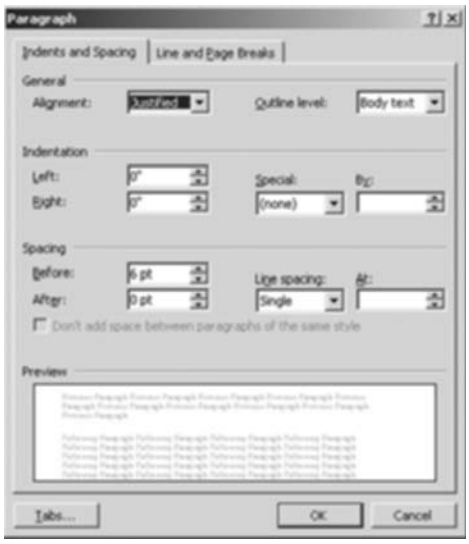


Fig. 3.22 **The Paragraph dialog box**

Figure 3.22 shows the Paragraph dialog box in which we can select various formatting styles either for the whole paragraph or for the selected text in a paragraph. We can change the alignment of the text in a paragraph by clicking Alignment drop down list and selecting an appropriate alignment for the selected text.

The different types of alignment styles available in the Paragraph dialog box are Left, Centered, Right and Justified. We can also indent a paragraph either in the left or right direction by selecting the indentation value in the Left and Right lists under the Indentation section.

Bullets and Numbering The Bullets and Numbering option in the Format menu can be used to create the bulleted and numbered list in a document. To create these lists, select Format → Bullets and Numbering to display the Bullets and Numbering dialog box, as shown in Fig. 3.23.

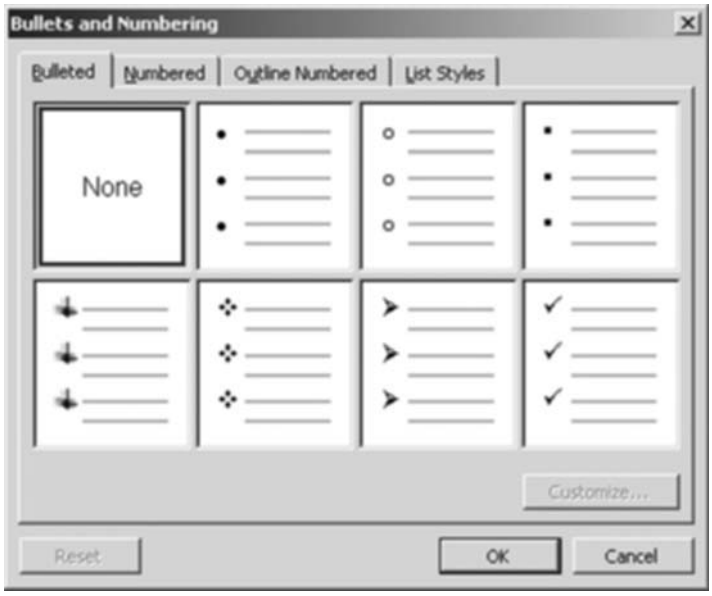


Fig. 3.23 **The Bullets and Numbering dialog box**

NOTE: We can also create the bulleted list by clicking the Bullets icon on the Formatting toolbar.

By default, the bulleted tabbed page appears in the Bullets and Numbering dialog box. As per our requirement, we can select the bullet style from the various bullet styles available in the bulleted tabbed page.

You can create a numbered list by clicking the Numbered tab in the Bullets and Numbering dialog box. The Numbered tabbed page allows us to select a numbering style, as shown in Fig. 3.24

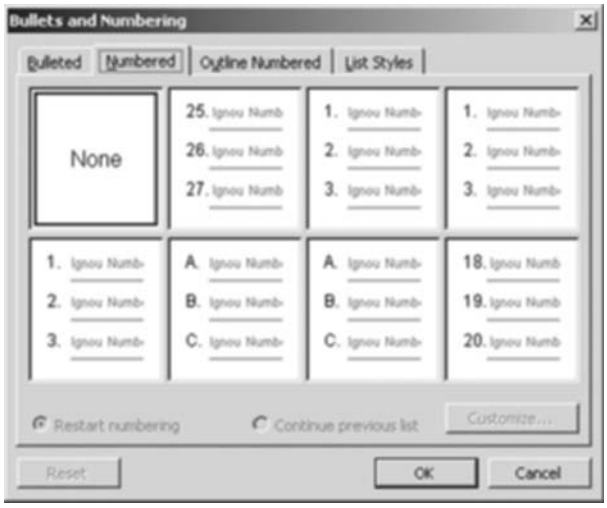


Fig. 3.24 **The Numbered tabbed page**

NOTE: We can also create the Numbered list by clicking the Numbering icon on the Formatting toolbar.

We can also create a numbered list with multiple levels, which is known as outline numbered list. To create an outline numbered list, click the Outline Numbered tab to display the Outline Numbered tabbed page, as shown in Fig. 3.25.

Fig. 3.25 **The Outline Numbered tabbed page**

After this, select the outline numbered style from the various styles available in the Outline Numbered tabbed page.

Printing the document After creating, editing, formatting and saving a document, we can print a copy of the document. However, before printing a document, we need to ensure that the printer is properly connected to the computer system and the appropriate print drivers have been installed. MS Word also provides a feature called Print Preview, which enables the user to get an idea of exactly how the document will appear after printing. This feature also provides the user a chance to make the last-minute changes before actually printing the document.

You can preview the content appearance by selecting File → Print Preview option to display the preview window, as shown in Fig. 3.26.

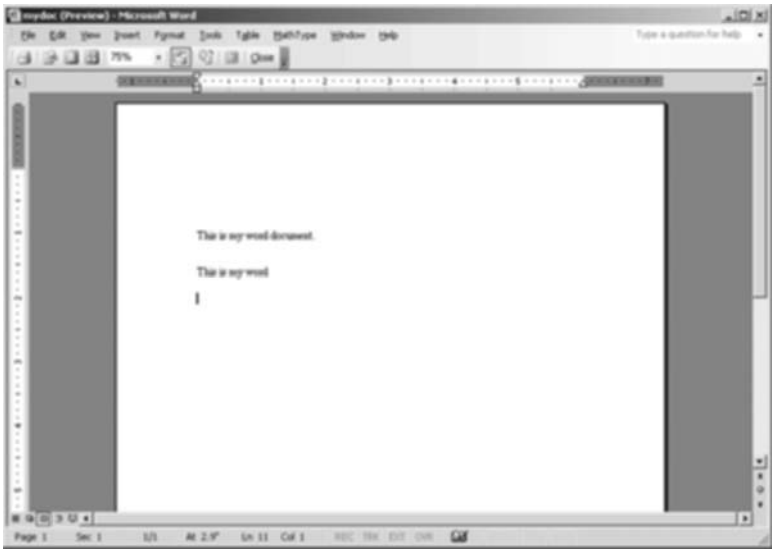


Fig. 3.26 The mydoc (Preview)-Microsoft Word window

NOTE: We can also display the Preview window by clicking the Print Preview icon on the Standard toolbar.

After previewing the content appearance, we can actually print it. To print the document, we need to perform the following steps:

1. Select File → Print to display the Print dialog box, as shown in Fig. 3.27.

Figure 3.27 shows the Print dialog box where we can specify information like printer name, number of pages to be printed, number of copies to be printed, etc.

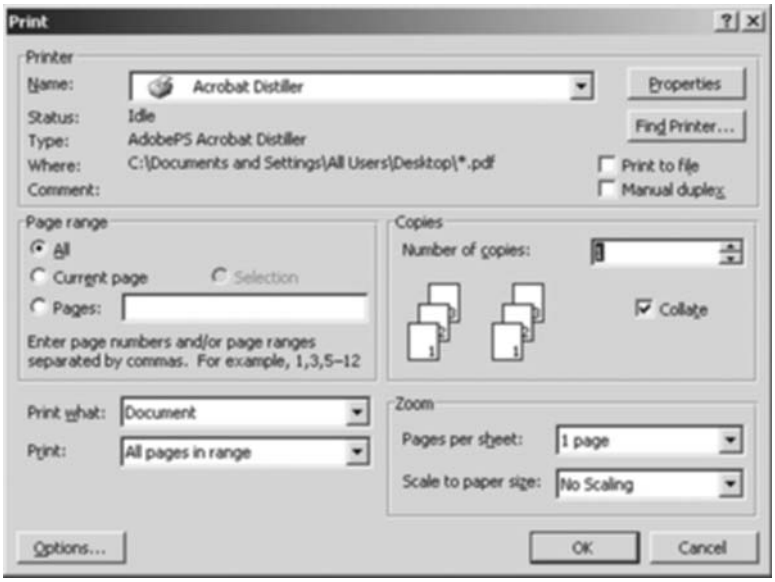


Fig. 3.27 **The Print dialog box**

NOTE: We can also display the Print dialog box either by pressing the Ctrl and P keys simultaneously or by clicking the Print icon on the Standard toolbar.

2. Accept the default settings and click OK to print the document.

Working with Tables You can draw a table in MS Word for presenting information in an organized manner. A table comprises of rows and columns. All the table-related operations in MS Word are performed with the help of Table menu.

Some common table-related tasks are:

- Drawing a table
- Inserting a table, row and column
- Merging the cells of a table

Drawing a table To draw a table, you need to perform the following steps:

1. Select Table → Draw Table, to display the Draw Table pointer and the Tables and Borders toolbar, as shown in Figure 3.28:
2. Drag and drop the Draw Table pointer to draw a rectangle. The boundary of this rectangle acts as the table border.
3. Drag and drop the Draw Table pointer inside the rectangle to add rows and columns in the table, as shown in Figure 3.29:
4. Enter data in the cells of the table.

NOTE: You may draw any number of columns and rows with any width and height sizes as per your requirement.

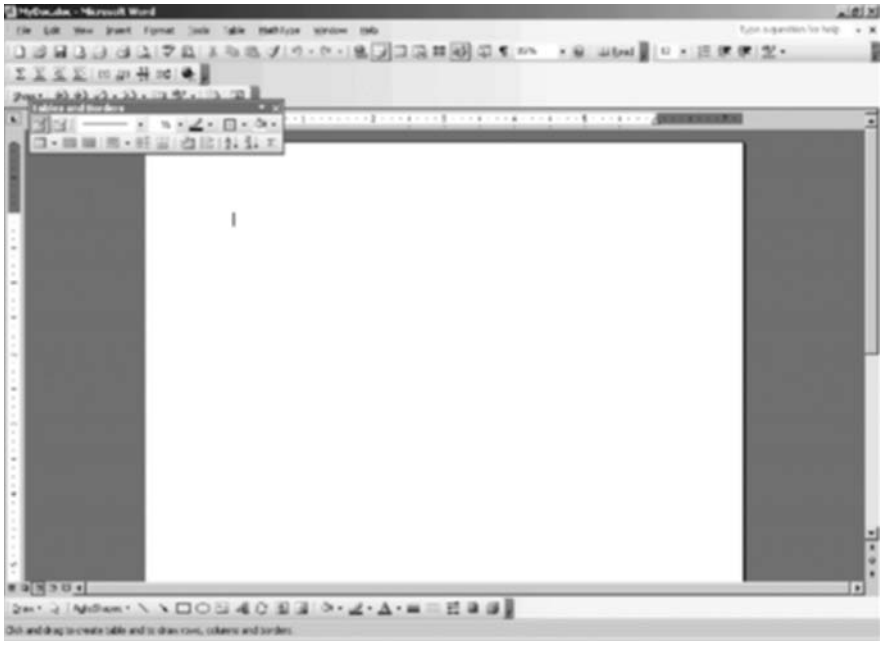


Fig. 3.28 **Drawing a Table**

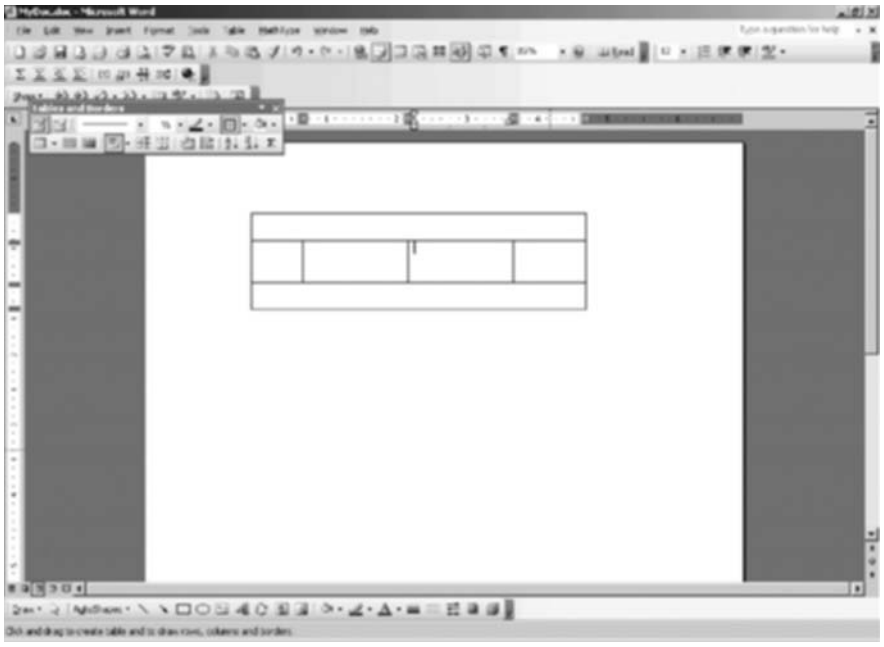


Fig. 3.29 **Adding Rows and Columns in a Table**

Inserting a table, row and column To insert a table, rows and columns, you need to perform the following steps:

1. Select Insert → Table, to display the Insert Table dialog box, as shown in Figure 3.30:

Fig. 3.30 The Insert Table dialog box

2. Enter a number, say 2, in the Number of columns text box and a number, say 5, in the Number of rows text box.
3. Click the OK button to insert the table in the current document, as shown in Figure 3.31:

Fig. 3.31 The Inserted Table

4. Enter some data in the table, as shown in Figure 3.32:

Roll No.	Name
1	Akrit
2	Amit
3	Bhuvnesh
4	Chetan

Fig. 3.32 **Entering data in the table**

5. Place the cursor on 3rd row of the 1st column and select Table → Insert → Rows Below, as shown in Figure 3.33:

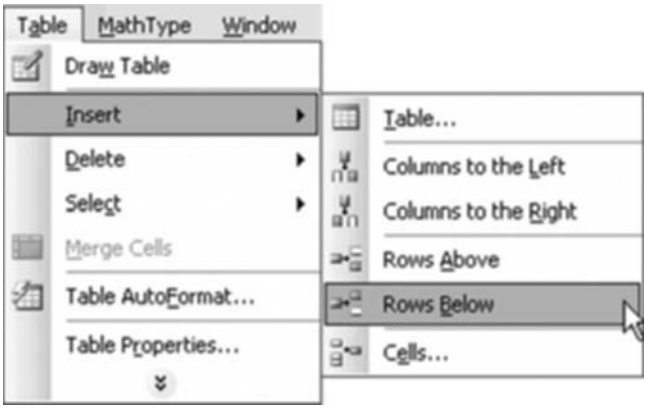


Fig. 3.33 **Inserting Rows**

6. A new row appears between the 3rd and the 4th row, as shown in Figure 3.34:

Roll No.	Name
1	Akrit
2	Amit
3	Bhuvnesh
4	Chetan

Fig. 3.34 **Inserted Row**

7. Similarly you can insert columns to the right or left of an existing column by selecting Table → Insert → Columns to the Left/Right option.

NOTE: You can delete rows, columns or an entire table by choosing an appropriate menu option under Table → Delete.

Merging the cells of a table You can combine two or more cells of a table into a single cell using the Merge cells option. Suppose you are writing a column heading, which is long and not appearing in a single cell. In such a case, you can merge two or more cells to adjust the heading in a single cell. To merge the cells, you need to perform the following steps:

1. Select the cells that you want to merge.
2. Select Table → Merge Cells, to merge the selected cells, as shown in Figure 3.35:

Fig. 3.35 Merging cells

Adding Pictures You can insert a picture, which is stored in the computer system using the Picture option provided by MS Word. To insert a picture from a file, you need to perform the following steps:

1. Place the cursor where you want to insert the picture.
2. Select Insert → Picture → From File option to display the Insert Picture dialog box, as shown in Figure 3.36:
3. Browse to the location where the image file is stored.
4. Select the picture that you want to insert and click the Insert button to insert it in your document, as shown in Figure 3.37:

NOTE: You may also quickly open the Insert Picture dialog box by using the shortcut key Alt + I + P + F.

Fig. 3.36 **Insert Picture dialog box**

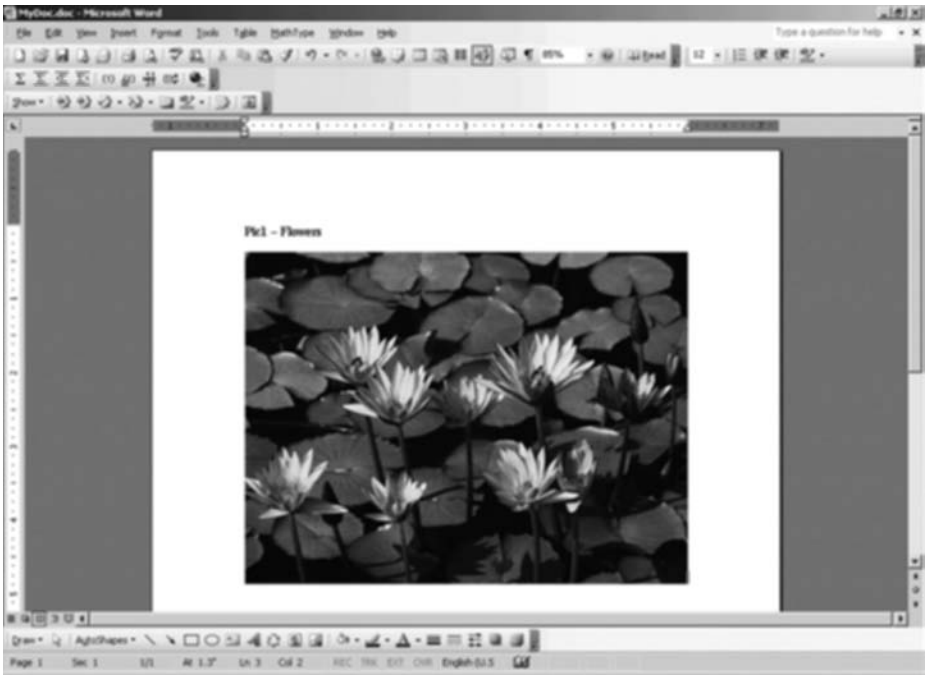


Fig. 3.37 **Inserted Picture**

Using Mail Merge Mail merge is a process in which contact information such as name and address stored in a database, is merged and added to a Word document. After adding the contact information to the letter, you can print the letter and send it to different people. Using mail merge involves the following four stages:

1. Creating a letter or using an existing letter
2. Creating a data source
3. Merging the data source with the letter
4. Printing the merged document

Consider a scenario where you have been asked to inform the members of the Activity Club of your college about a meeting schedule. For this, you need to send a letter containing the meeting schedule to all the members. Let's perform this activity using the mail merge feature. First, draft the letter containing the meeting schedule in a Word document. Now, create a data source by performing the following steps:

1. Open the letter document already drafted.
2. Select Tools → Letters and Mailings → Mail Merge Wizard, to display the Mail Merge pane at the right-hand side of the MS Word window, as shown in Figure 3.38:

Fig. 3.38 The Mail Merge Pane

NOTE: By default the Letters option is selected under the Select document type section of the Mail Merge pane.

3. Click the Next: Starting document link at the bottom of the pane. The Select starting document section appears, as shown in Figure 3.39:

Fig. 3.39 **The Select starting document section**

NOTE: By default, the Use the current document option is selected under the Select starting document section.

4. Click the Next: Select recipients link at the bottom of the pane to display the Select recipients section, as shown in Figure 3.40:

Fig. 3.40 **The Select recipients section**

5. Select the Type a new list option under the Select recipients section. The moment you select the Type a new list option, the Type new list section appears containing the Create link, as shown in Figure 3.41:

Fig. 3.41 **The Type a new list section**

6. Click the Create link to open the New Address List dialog box, as shown in Figure 3.42:

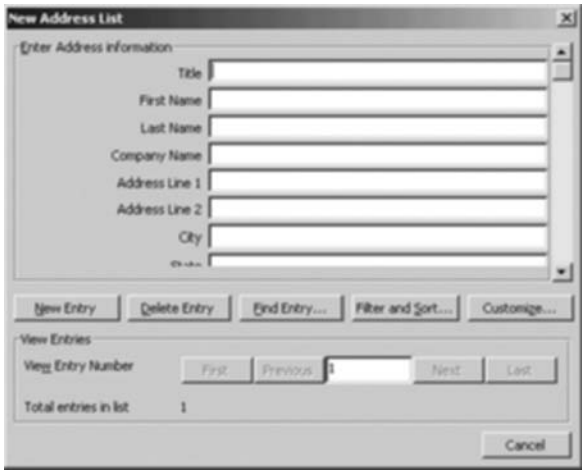


Fig. 3.42 **The New Address List dialog box**

7. Enter information such as title, first name, last name, address and city, in the respective text boxes under the Enter Address Information section to create an entry in the address list.
8. Click the New Entry button to create another entry in the similar manner.
9. Once you have created all the contact entries to be mail merged with the letter, click the Close button to close the New Address List dialog box. The Save Address List dialog box appears, as shown in Figure 3.43:

Fig. 3.43 **The Save Address List dialog box**

10. Specify a name for the address list and the location where you want to save it on the computer.
11. Click the Save button to save the address list. The Mail Merge Recipients window appears with entries, which have been added to create the address list.
12. Click the OK button to display the Select recipients section with the added recipients.

After creating the data source, you need to merge it with the letter (already written in Word document) by performing the following steps:

1. Place the cursor after the word, To, for inserting the title.
2. Click the Insert Merge Fields icon on the Mail Merge toolbar to display the Insert Merge Field dialog box, as shown in Figure 3.44:

Fig. 3.44 The Insert Merge Field dialog box

3. Select the Title field from the Fields list and click the Insert button to insert the title field.
4. Insert other fields such as First Name, Last Name, Company Name and City in the same way according to your requirements.
5. Similarly, place the cursor after the word, Dear and click the Insert Merge Fields icon on the Mail Merge toolbar to display the Insert Merge Field dialog box.
6. Select the First Name field to insert the First Name field and click the Insert button to insert the first name field. Similarly, insert all the other required fields.
7. Click the Close button to close the Insert Merge Field dialog box.

8. Click the Next: Write your letter link in the Mail Merge pane. Since, you have already drafted your letter, so click the Next: Preview your letters link at the bottom. A preview of your letter with the inserted fields appears in the current page.
9. Click the Next: Complete the merge link at the bottom of the pane.
10. Click the Edit Individual letters link under the Complete the merge section to display the Merge to New Document dialog box. By default, the All option is selected in the dialog box.
11. Click the OK button. The Letter1 document appears containing the merged documents.
12. Save the Letter1 document with a name, say Merged Letter.

Now, you can take out the printed copies of the merged document using the Print option of the File menu. Now, you have the meeting schedule for each member of the Activity Club ready.

3.8 MS EXCEL

MS Excel is an application program that allows us to create spreadsheets, which are represented in the form of a table containing rows and columns. The horizontal sequence in which the data is stored is referred to as a row. The vertical sequence in which the data is stored is referred to as a column. In a spreadsheet, a row is identified by a row header and a column is identified by a column header. Each value in a spreadsheet is stored in a cell, which is the intersection of rows and columns. A cell can contain either numeric value or a character string. We can also specify the contents of a cell using formulas. In a spreadsheet, we can perform various mathematical operations using formulas, such as addition, subtraction, multiplication, division, average, percentage, etc.

MS Excel also allows us to represent the complex data graphically. These are generally used to represent the information with the help of images, colours, etc. so that their presentation is simple and more meaningful. Some of the graphs available in spreadsheet are bar graphs, line graphs, 3-D graphs, area graphs, etc.

3.8.1 Accessing MS Excel

For working with MS Excel, we first need to install MS Office in our computer system. After installing MS Office, we can start MS Excel using any of the following two ways:

- Start menu
- Run command

Using Start menu We can start MS Excel by performing the following steps using the Start menu:

1. Select Start → Programs → Microsoft Office, as shown in Fig. 3.45.

Select the Microsoft Office Excel 2003 option to display the GUI of MS Excel, as shown in Fig. 3.46.

Fig. 3.45 Starting MS Excel using the Start menu

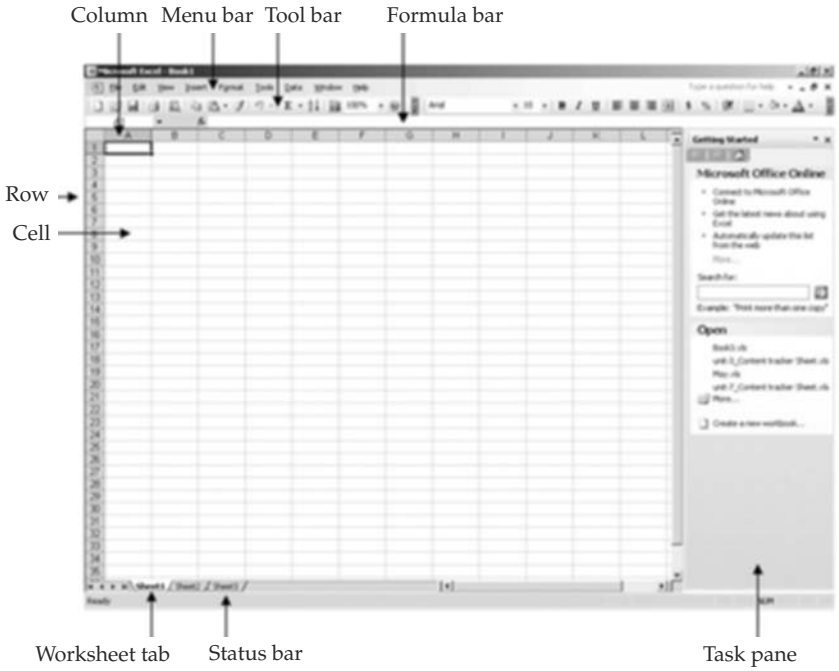


Fig. 3.46 The Microsoft Excel-Book1 window

Figure 3.46 shows the initial workbook of MS Excel, which in turn contains worksheets. Each worksheet contains rows and columns where we can enter data.

Using Run command We can also start MS Excel by performing the following steps using the Run command:

1. Select Start → Run to display the Run dialog box.
2. Type excel in the Open text box and click OK to display the Microsoft Excel- Book1 window.

3.8.2 Basic Operations Performed in MS Excel

Worksheet is the actual working area consisting of rows and columns. The worksheets are also known as the spreadsheets. A workbook in MS Excel is a combination of several worksheets. Each workbook of MS Excel contains three worksheets by default. The key operations that are performed in MS Excel include:

- Creating a worksheet
- Saving a worksheet
- Modifying a worksheet
- Renaming a worksheet
- Deleting a worksheet
- Moving a worksheet
- Editing a worksheet

Creating the worksheet We can create a worksheet in MS Excel by simply inserting the data in the cells of the worksheet. To create a worksheet, perform the following steps:

1. Open the Microsoft Excel - Book1 window.
2. Insert the data into the cells according to the requirement, as shown in Fig. 3.47.

Saving the worksheet After entering the data in the worksheet, we need to save the worksheet at the desired location in the computer system. To save a worksheet, perform the following steps:

1. Select File → Save As to display the Save As dialog box.
2. Select a location from the Save in list where the worksheet is to be saved.
3. Enter the name of the file in the File name text box.
4. Click the Save button to save the file.

Modifying the worksheet A worksheet in MS Excel can be modified in the following two ways:

- By inserting rows and columns in the existing worksheet.
- By changing the width or height of rows and columns.

Inserting rows and columns In order to insert a row in the worksheet, select Insert → Rows, as shown in Fig. 3.48.

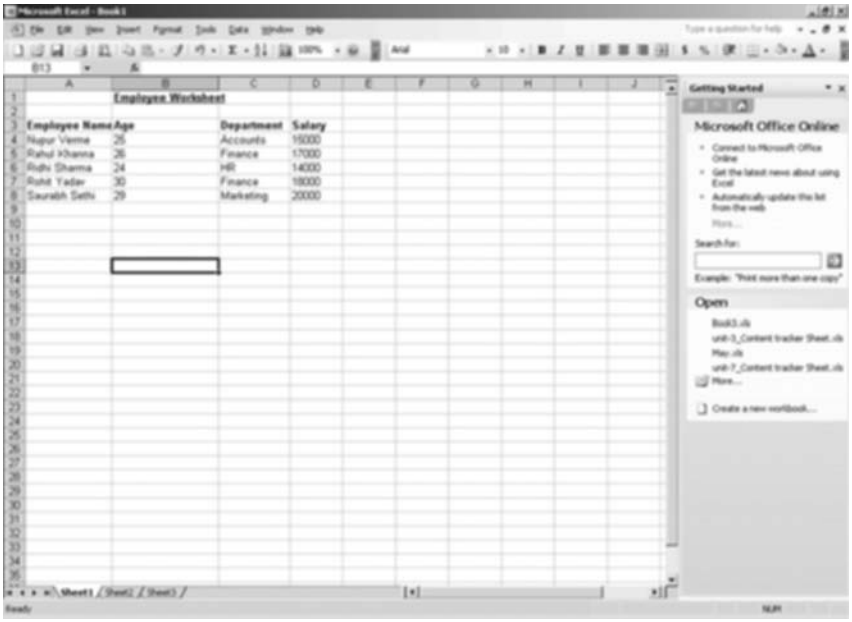


Fig. 3.47 Entering data in the Microsoft Excel-Book1 window

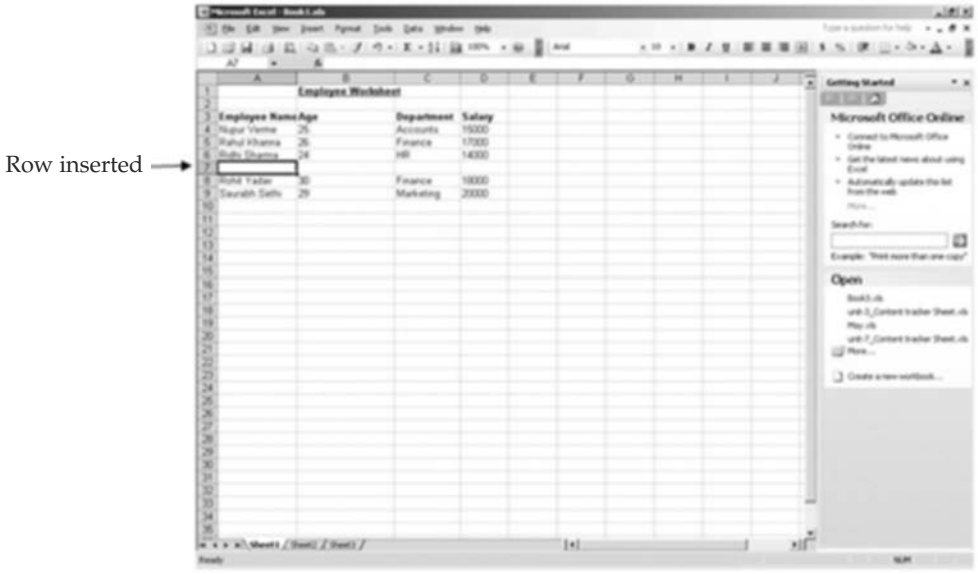


Fig. 3.48 Inserting a row in Microsoft Excel-Book1

NOTE: Similarly, we can insert a column in the worksheet by selecting Insert → Columns.

Changing the width or height of the rows and columns In order to change the height of the rows in the worksheet, perform the following steps:

1. Select Format → Row → Height to display the Row Height dialog box, as shown in Fig. 3.49.
2. Enter the required height in the Row Height text box and click OK to apply height specifications to the rows. Figure 3.50 shows the Microsoft Excel Book1 with the height of the row modified to 20.

Fig. 3.49 The Row Height dialog box

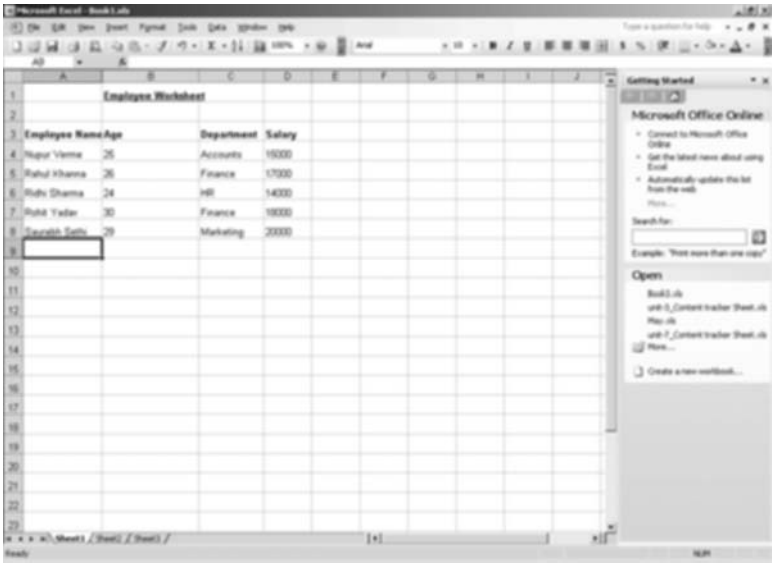


Fig. 3.50 Formatting rows in the Microsoft Excel-Book1 window

Renaming the worksheet Usually the default names of the worksheet in MS Excel are Sheet1, Sheet2, Sheet3, etc. In order to rename the worksheet, we need to perform the following steps:

1. Open the Microsoft Excel - Book1 window.
2. Right-click the Sheet1 tab to display a shortcut menu.
3. Select the Rename option and change the name of the worksheet.

Figure 3.51 shows the Microsoft Excel-Book1 window in which Sheet1 has been renamed to Employee Worksheet.

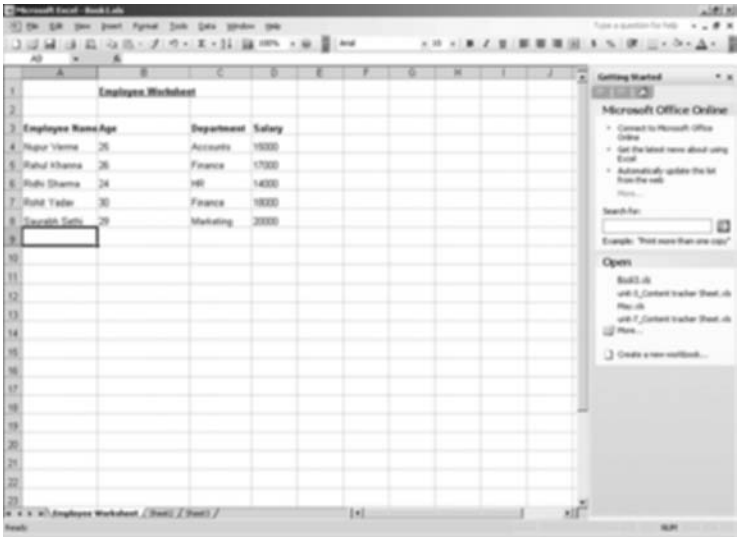


Fig. 3.51 **Renaming the Microsoft Excel-Book1**

Deleting the worksheet In order to delete a worksheet from the workbook, we need to perform the following steps:

1. Open the Microsoft Excel - Book1 window.
2. Right-click the Sheet2 tab to display the shortcut menu.
3. Select the Delete option to delete the Sheet2 worksheet.

Figure 3.52 shows the Microsoft Excel - Book1 window with Sheet2 deleted.

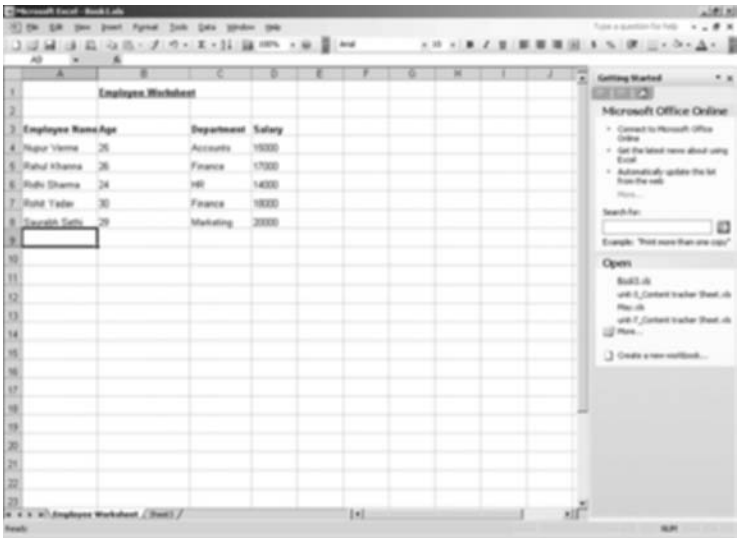


Fig. 3.52 **The Microsoft-Excel Book1 window with Sheet2 deleted.**

Moving the worksheet In order to move a worksheet from one location to another, we need to perform the following steps:

1. Open the Microsoft Excel-Book1 window.
2. Right-click the Employee Worksheet tab to display a shortcut menu.
3. Select the Move or Copy option to display the Move or Copy dialog box, as shown in Fig. 3.53.
4. Select the location, say (move to end), from the Before sheet list, to move the current sheet before the selected one and press OK.

Figure 3.54 shows that the Employee Worksheet moved to the end.

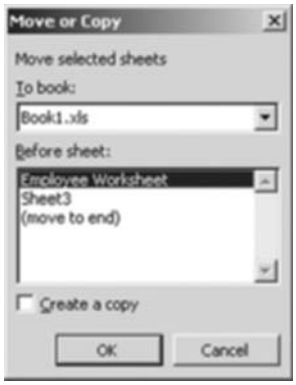


Fig. 3.53 **The Move or Copy dialog box**

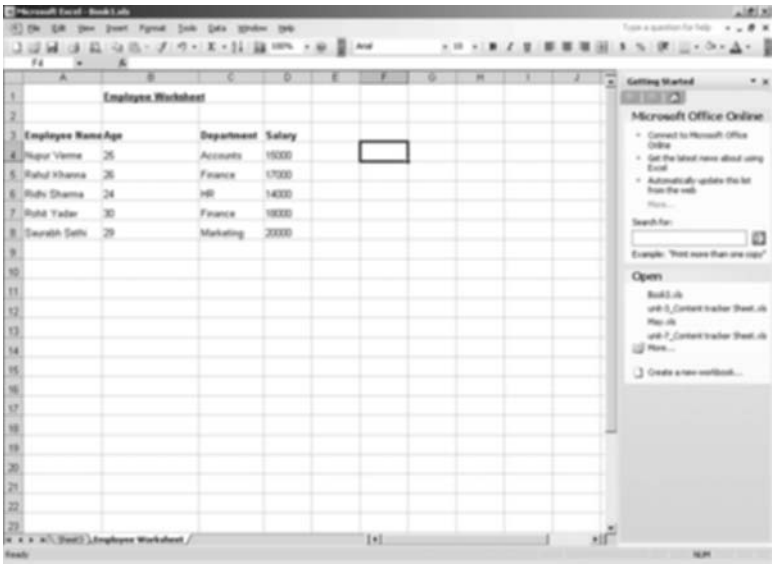


Fig. 3.54 **Displaying the changed position of the Employee Worksheet**

Editing the worksheet In order to edit a worksheet in the workbook, we need to perform the following steps:

1. Open the Microsoft Excel-Book1 window.
2. Double-click the cell in which we want to make the changes.
3. Enter the new data in the cell.

Figure 3.55 shows the Microsoft Excel-Book1 with modified data.

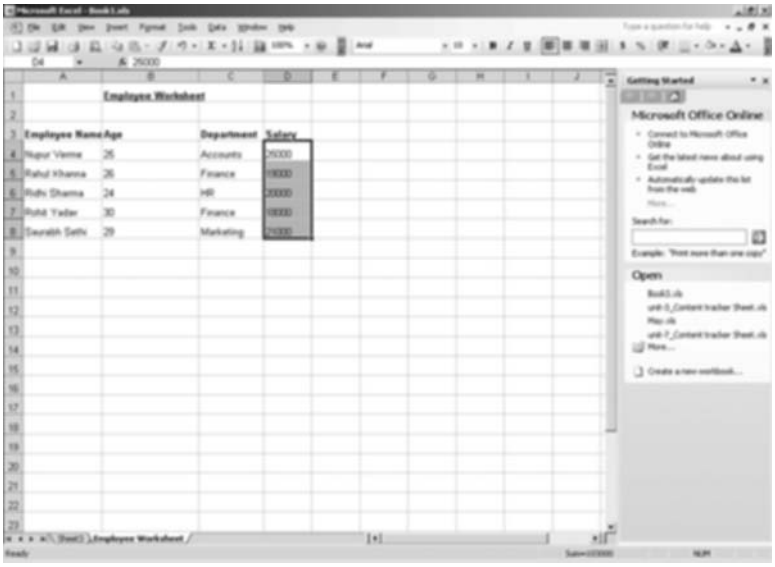


Fig. 3.55 **Editing the Microsoft Excel-Book1**

Using Formulas and Functions In MS Excel, formulas are equations that are used for manipulating the data of the worksheet. To write a formula in a cell, first insert the equal to sign (=) and then enter the value, otherwise the content of the cell is considered as simple text. A formula may simply contain numbers and operators or a predefined expression to perform a particular computation. In general, a formula consists of the following parts:

- **Constants** — It includes simple numbers and texts that are not subjected to change, i.e. their values do not depend upon any formula. For example, the numbers 24, 892 and 1000, and texts 'sum of numbers' and 'product of quotient' are constants.
- **Operators** — It includes symbols that represent the type of operations, such as addition, subtraction, multiplication, and division to be performed on the data. Thus, the symbols that constitutes operators are +, –, *, and /.
- **Functions** — It includes predefined expressions that accepts a set of data and returns a set of values after performing a particular operation. Examples of functions are SUM, COUNT, MAX, and IF.
- **Reference** — It indicates the address of a cell or a range of cells in a worksheet. It helps to refer the values in different parts of a worksheet in a formula. Using references, you can also refer cells on different worksheet on same or different workbook.

To perform any calculation in MS Excel, you need to write the corresponding formula in a cell. Suppose you want to calculate the number of working hours in a week, if you worked daily eight hours for six days. To write the formula for computing number of working hours, you need to perform the following steps:

1. Select the cell that should contain the result of the formula.
2. Type the = (equal to) symbol in the selected cell.

3. Enter the values and operators to build an expression for the formula, say 8*6.
4. Press Enter to display the result.

Alternatively, you can also write the formula in the formula bar after selecting the required cell.

The operators used in the formulas may be either unary or binary in nature. A unary operator requires only one operand, whereas a binary operator requires two operands. In MS Excel, the operators can be categorized into different types, which are:

- **Arithmetic operators** — They are used to perform simple arithmetic calculations. Examples of arithmetic operators are +, –, *, /, %, and ^.
- **Text operator** — It is used to concatenate any two strings. Ampersand, &, is the text operator.
- **Comparison operators** — They are used to compare any two values. Examples of comparison operators are <, >, <=, >=, and <>.
- **Reference operators** — They are used to group a range of cell in a formula. The reference operators are:
- **Range** — It is used to specify a range of cells, which is indicated by colon (:). For example, the following formula calculates the sum of all the values in the cells from A5 to A15:

=SUM(A5:A15)

- **Union** — It is used to specify values from different parts of the worksheet in the formula. It is indicated by comma (,). For example, the following formula calculates the sum of the values at location A8, B10, and D16:

=SUM(A8, B10, D16)

- **Intersect** — It is used to perform a specific operation on cells that are common in different groups of cells. It is specified by space. For example, the following formula calculated the sum of all the values in the cells from B8 to B12:

=SUM(B4:bB14 B8:B16)

If a formula contains more than one operator, then the formula is executed according to the precedence rule of the operators. Table 3.1 lists the precedence rule of the operators used in MS Excel.

A formula may also contain references other than constant values and operators. The references are used when a range of cells is to be specified or when the cell to be included in the formula contains another formula. A cell that contains a formula is known as dependent cell because its value depends on the values of other cells. For example, the cell B4 contains the following formula:

=A1+ B7

This means that the value in B2 depends on the values in A1 and B7.

You can also use predefined functions to perform the required operations. For example, to add two numbers, you can use either the + operator or the SUM() function. The operands that

TABLE 3.1 Precedence rule of operators

Operator	Name of operator
:	Range
Space	Intersect
,	Union
–	Negation
%	Percentage
^	Exponent
*, /	Multiplication and division
+, –	Addition and subtraction
&	Ampersand
<, >, <=, >=, =, <>	Comparison operators

are used with the + operator are used as the arguments of the SUM function. In MS Excel, the functions can be categorized into the following types:

- **Financial** — It includes functions, such as FV, PMT, and RATE that are used in the computation of financial data.
- **Date & Time** — It includes functions, such as DATE, DAY, MONTH, TIME, TODAY, and YEAR that are used to retrieve day and time related information for a particular date.
- **Math & Trig** — It includes functions, such as SIN, COS, EXP, FACT, CEILING, and FLOOR that are used to evaluate mathematical as well as trigonometric equations.
- **Statistical** — It includes functions, such as COUNT, MAX, MEDIAN, and MODE that are used to evaluate statistical data.
- **Lookup & Reference** — It includes functions, such as COLUMN, HLOOKUP, ROW, and VLOOKUP that are used to locate the values and references satisfying the given condition.
- **Database** — It includes functions, such as DCOUNT, DMAX, and DMIN that are used to retrieve specific information from a particular database.
- **Text** — It includes functions, such as CHAR, CONCATENATE, FIND, LEN, and PROPER that are used to perform manipulation on strings.
- **Logical** — It includes functions, such as AND, IF, NOT, and OR that are used evaluate a logical expression.
- **Information** — It includes functions, such as CELL, ERROR.TYPE, ISBALNK, and ISERR that provides information about the values of the cells used in the formula.

To insert a function as a formula, you need to perform the following steps:

1. Open the worksheet in which you want to perform the computation.
2. Select the cell that should contain the result of the formula.

3. Select Insert -> Function to display the Insert Function dialog box, as shown in Figure 3.56.

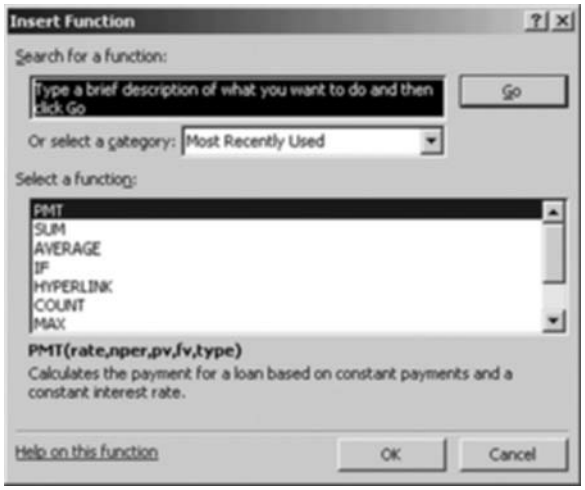


Fig. 3.56 The Insert Function dialog box

The above figure shows the Insert Function dialog box in which to select a function; you can either write the function name in the Search for a function text box or select the name of the function from the Select a function list box.

4. Select the name of the function, say IF from the Select a function list.

NOTE: The description of the function, which is selected in the Select a function list, appears at the bottom of the Insert Function dialog box.

5. Click OK to display the Function Arguments dialog box for the IF function, as shown in Figure 3.57.

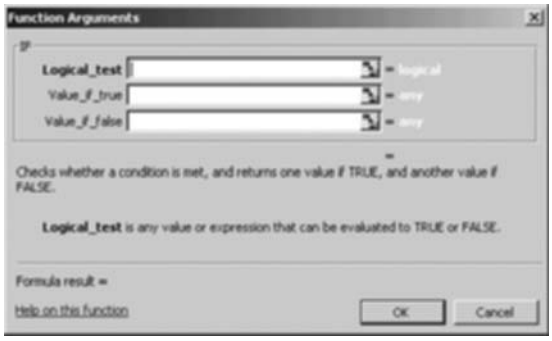


Fig. 3.57 The Function Arguments dialog box

The above figure shows the Function Arguments dialog box, in which you can specify the condition to be tested in the Logical_test test box and the resultant values if the condition is true or false in the Value_if_true and Value_if_false text boxes respectively.

6. Enter the logical condition, say A4 > 100 in the Logical_test text box.
7. Enter the values in the Value_if_true and Value_if_false text boxes.
8. Click OK to close the Function Arguments dialog box. The result of the function appears in the selected cell.

Creating Charts in Excel Charts are a brilliant way of analyzing tabular or numerical data in graphical form. To create a chart in MS Excel, you need to perform the following steps:

1. Select the data fields from the worksheet, as shown in Figure 3.58:

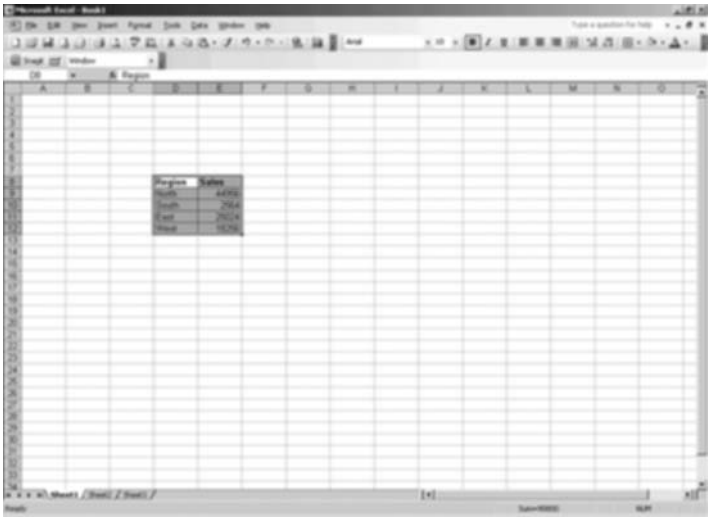


Fig. 3.58 Selecting the data fields

2. Click on the chart button on the Toolbar to display the Chart Wizard dialog box, as shown in Figure 3.59.
3. Under the Chart type section select the Pie option to create the pie chart type and click Next. The Chart Source Data screen appears, as shown in Figure 3.60.
4. Under the Data range field, you can modify the data range corresponding to which you want to create the chart. Click Next to display the Chart Options screen, as shown in Figure 3.61.
5. Select the Value check box under the Label Contains section to display the individual values in the chart and click the Next button. The Chart Location screen appears.
6. Click Finish to display the newly created chart in the worksheet, as shown in Figure 3.62.

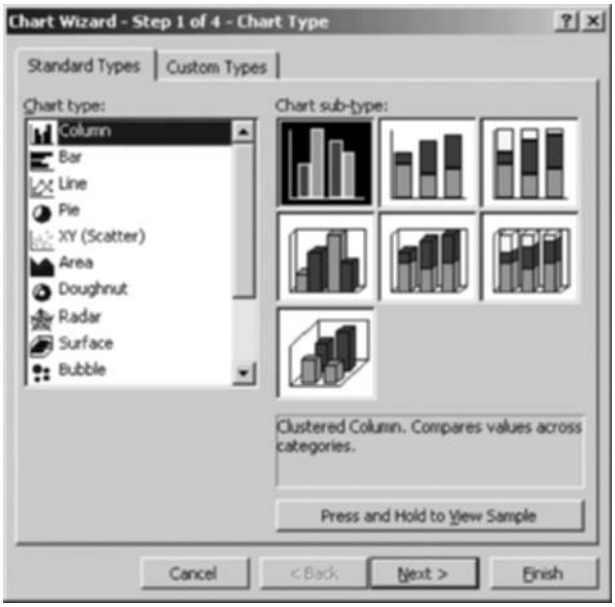


Fig. 3.59 **The Chart Wizard dialog box**

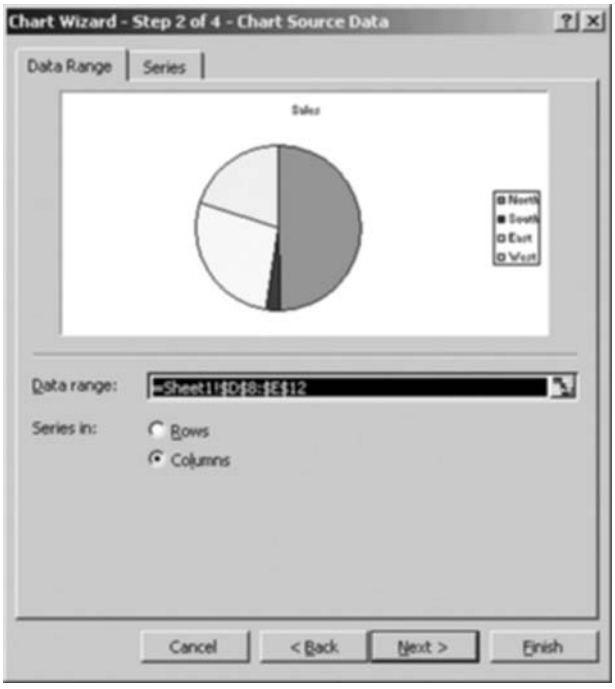


Fig. 3.60 **The Chart Source Data screen**

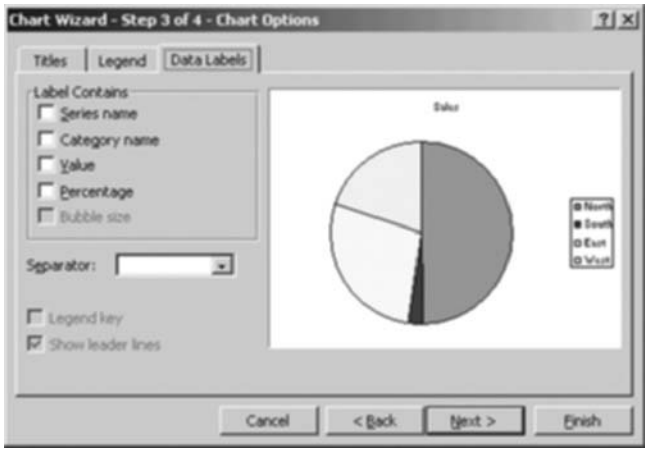


Fig. 3.61 **The Chart Options screen**

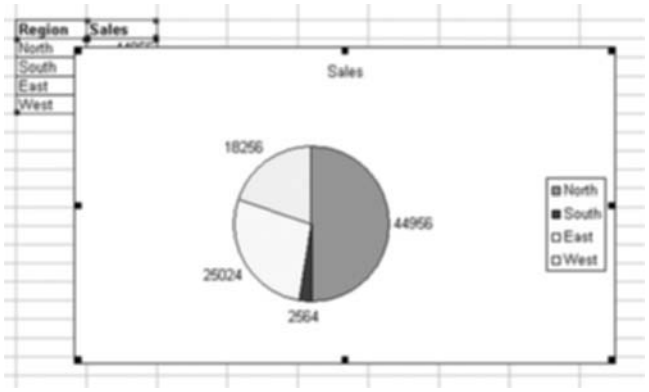


Fig. 3.62 **The newly create chart**

3.9 MS POWERPOINT

MS PowerPoint is a software application included in the MS Office package that allows the user to create presentations. PowerPoint provides a GUI with the help of which we can create attractive presentations quickly and easily. The presentation may include slides, handouts, notes, outlines, graphics and animations. A slide in PowerPoint is a combination of images, text, graphics, charts, etc. that is used to convey some meaning information. The presentations in MS PowerPoint are usually saved with the extension .ppt. The interface of MS PowerPoint is similar to the other interfaces of MS Office applications. PowerPoint presentations are commonly used in business, schools, colleges, training programmes, etc.

3.9.1 Accessing MS PowerPoint

For working in MS PowerPoint, we need to first install the MS Office package in our computer system. After installing MS Office, we can start MS PowerPoint using any of the following two ways:

- Start menu
- Run command

Using Start menu We can start MS PowerPoint by performing the following steps using the Start menu:

1. Select Start → Programs → Microsoft Office, as shown in Fig. 3.63.

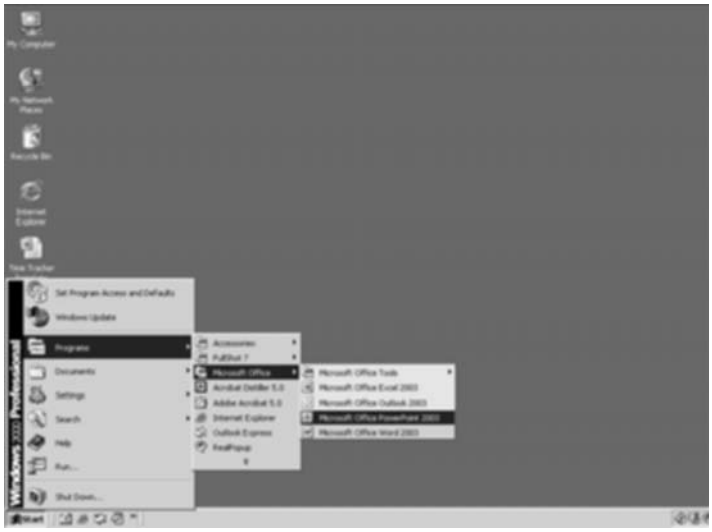


Fig. 3.63 Starting MS PowerPoint using the Start menu

2. Select the Microsoft Office PowerPoint 2003 option to display the GUI of MS PowerPoint, as shown in Fig. 3.64.

Using Run command We can also start MS PowerPoint by performing the following steps using the Run command:

1. Select Start → Run to display the Run dialog box.
2. Type powerpnt in the Open text box and click OK to display the Microsoft PowerPoint-[Presentation1] window.

3.9.2 Basic Operations Performed on a Presentation

The following are the key operations that can be performed in MS PowerPoint:

- Creating a new presentation

Fig. 3.64 The Microsoft PowerPoint-[Presentation1] Window

- Designing the presentation
- Saving a new presentation
- Adding slides to the presentation
- Printing the presentation

Creating a new presentation We can create a new presentation in MS PowerPoint by performing the following steps:

1. Open the Microsoft PowerPoint-[Presentation1] window.
2. Enter some text in the slide, as shown in Fig. 3.65.

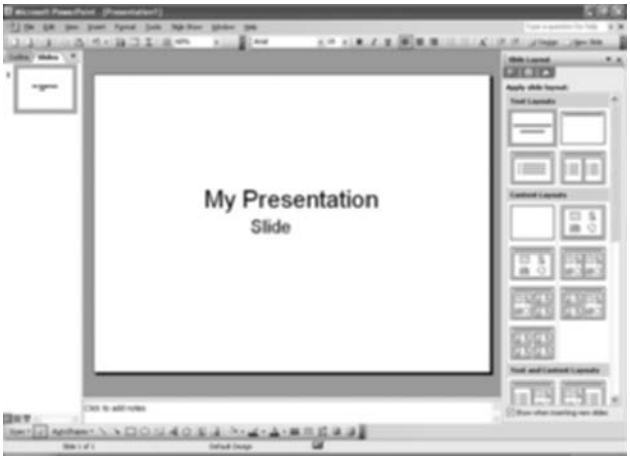


Fig. 3.65 Entering text in the Microsoft PowerPoint-[Presentation1] window

Designing the presentation After creating a presentation, we can design it by performing the following steps:

1. Select Format → Slide Design to display the Slide Design task pane at the right-hand side of the Microsoft PowerPoint-[Presentation1] window, as shown in Fig. 3.66.

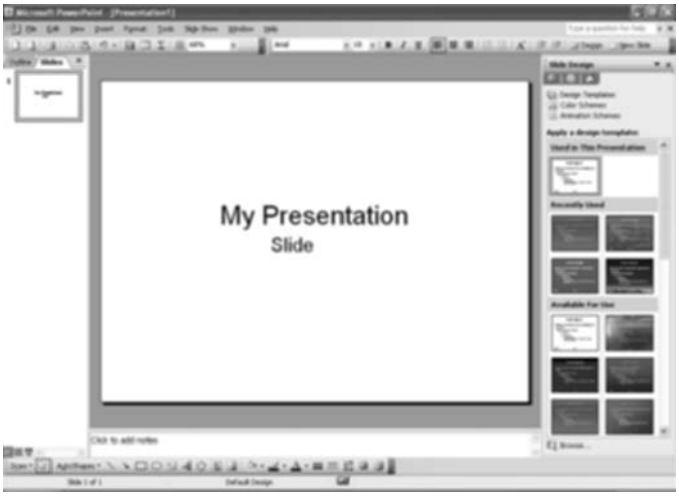


Fig. 3.66 **The Slide Design task pane**

2. Select a slide design, say Maple, from the Apply a design template section, as shown in Fig. 3.67.

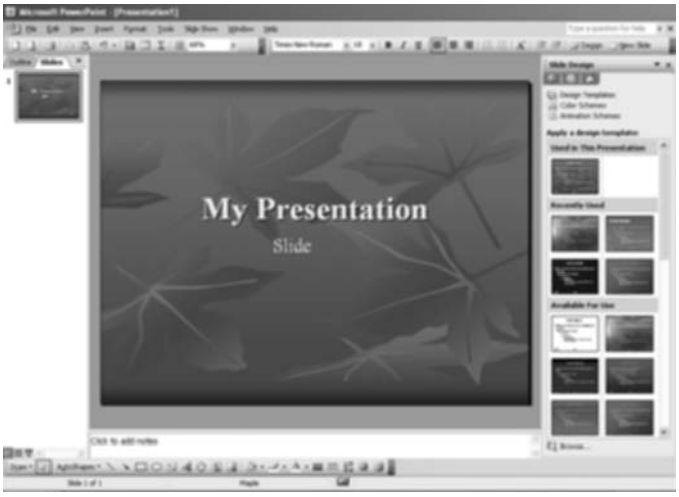


Fig. 3.67 **Designing the presentation**

Saving the presentation After we have finished creating and designing a presentation, we need to save it at some appropriate location in the computer system for future reference. To save the presentation, we need to perform the following steps:

1. Select File → Save to display the Save As dialog box in which the File name text box contains the default name suggested by MS PowerPoint. However, we can change the name as per our requirement.
2. Select a location from the Save in list for saving the presentation.
3. Type a name, say My Presentation1, in the File name text box and click the Save button to save the presentation.

Adding slides to the presentation You can add a new slide to the presentation by performing the following steps:

1. Open the Microsoft PowerPoint-[My Presentation1] window.
2. Select Insert → New Slide option to add a new slide, as shown in Fig. 3.68.

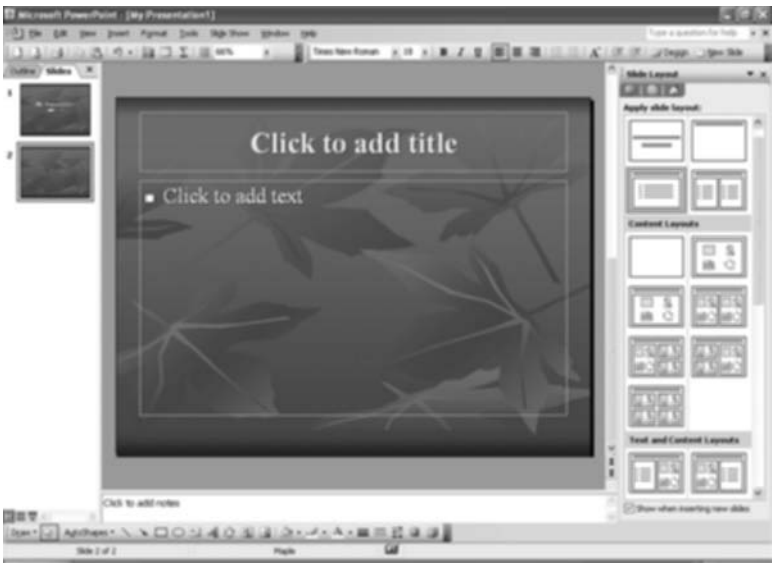


Fig. 3.68 The Microsoft PowerPoint- [My Presentation1] window with a new slide

NOTE: You can also insert a new slide by right-clicking the slide thumbnail and then selecting the New Slide option.

Printing the presentation After creating, designing and saving a presentation, we can print a copy of it. Just like MS Word, MS PowerPoint also provides the feature of print preview. We can preview the content appearance before issuing the print command by selecting File → Print Preview to display the preview window, as shown in Fig. 3.69.

Fig. 3.69 The Microsoft PowerPoint-[My Presentation1 [Preview]] window

After previewing, we need to perform the following steps to print the MS PowerPoint presentation:

1. Select File → Print to display the Print dialog box.
2. Specify the number of pages and numbers of copies of each page to be printed and click OK to print the presentation.

3.10 MS ACCESS

Microsoft Access (MS Access) is a Relational Database Management System (RDBMS) that allows the users to create a database and store the data in the form of rows and columns, i.e. in the form of tables. MS Access comprises a database engine known as Microsoft Jet Database Engine and a GUI. It also supports various tools that help the users in creating and managing databases. MS Access is used by business organisations and programmers for creating an organised database system.

3.10.1 Accessing MS Access

For working in MS Access, we need to install first MS Office in our computer system. After installing MS Office, we can start MS Access using any of the following two ways:

- Start menu
- Run command

Using Start Menu We can start MS Access by performing the following steps using the Start menu:

1. Select Start → Programs → Microsoft Office, as shown in Fig. 3.70.

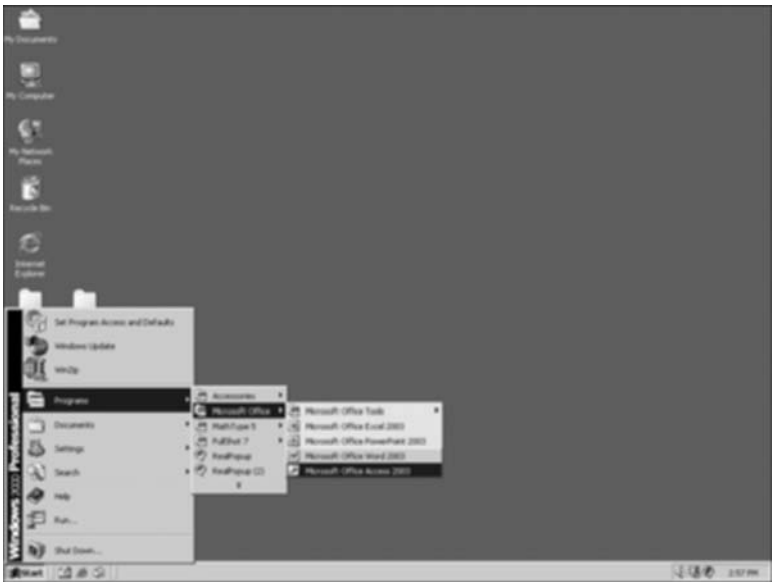


Fig. 3.70 Starting MS Access using the Start menu

2. Select the Microsoft Office Access 2003 option to display the GUI of MS Access, as shown in Fig. 3.71.

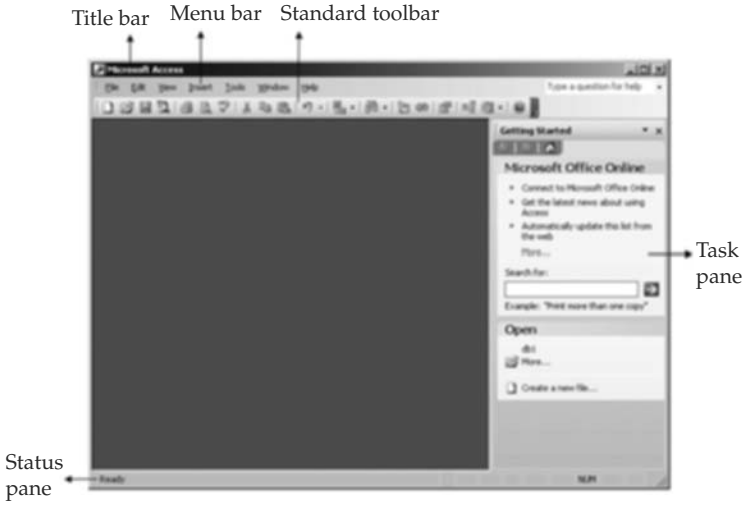


Fig. 3.71 The Microsoft Access window

Using Run command We can also start MS Access by performing the following steps using the Run command:

1. Select Start → Run to display the Run dialog box.
2. Type msaccess in the Open text box and click OK to display the Microsoft Access window.

NOTE: The basic functions of the GUI components of MS Access are same as that of MS Word.

3.10.2 Basic Operations Performed in MS Access

MS Access is a database management system that can be used for creating databases. We can perform various operations in MS Access for storing the data in an efficient manner. The following are some of the key operations that can be performed in MS Access:

- Creating a database
- Creating a database table
- Defining relationships
- Creating a database query

Creating a database We can create a database in MS Access by performing the following steps:

1. Open the Microsoft Access window.
2. Select File → New to display the New File task pane on the right side of the Microsoft Access window, as shown in Fig. 3.72.

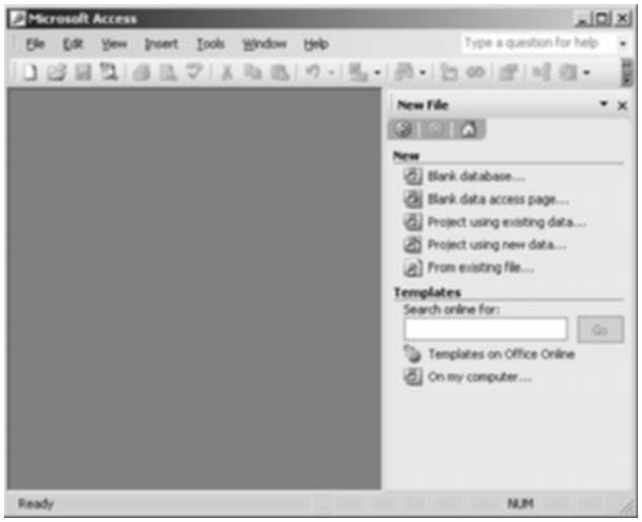


Fig. 3.72 The Microsoft Access window containing the New File task pane

3. Click the Blank database icon under the New section to display File New Database dialog box, as shown in Fig. 3.73.

Fig. 3.73 The File New Database dialog box

4. Select the location from the Save in list for saving the database.
5. Enter the name of the database such as Student Database in the File name text box and click the Create button to create the Student Database. Figure 3.74 shows the Student Database: Database (Access 2000 file format) window.

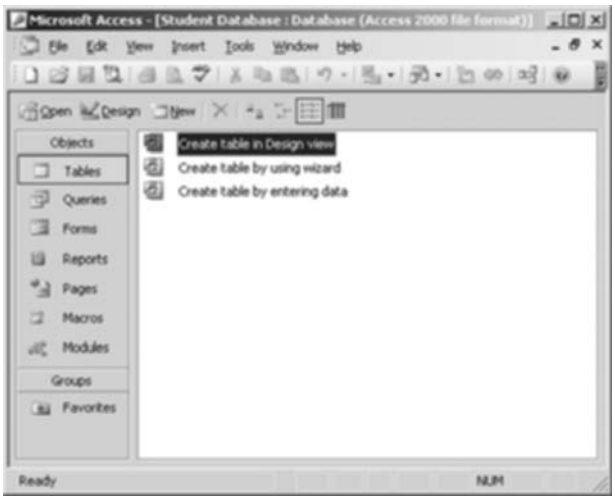


Fig. 3.74 The Student Database window

Creating a database table We can create a table in MS Access using any of the following three options available in the Student Database: Database (Access 2000 file format) window:

- **Create table in Design view** This option allows us to first create the design of the table, i.e. the fields of the table along with its data type and then enter data into it.
- **Create table by using wizard** This option provides some sample tables to the user. These sample tables are divided into two categories—business and personal. After selecting a sample table and its fields, the users can enter data into the newly created table.
- **Create table by entering data** This option allows the user to design the table and enter data in the table simultaneously. There is no need to specify the data type for the fields. The most commonly used method for creating a table in MS Access is the Create table in Design view option. For creating a table in MS Access using the Create table in Design view option, we need to perform the following steps:
  1. Double-click the Create table in Design view option in Student Database: Database (Access 2000 file format) window to display the Table1: Table page, as shown in Fig. 3.75.

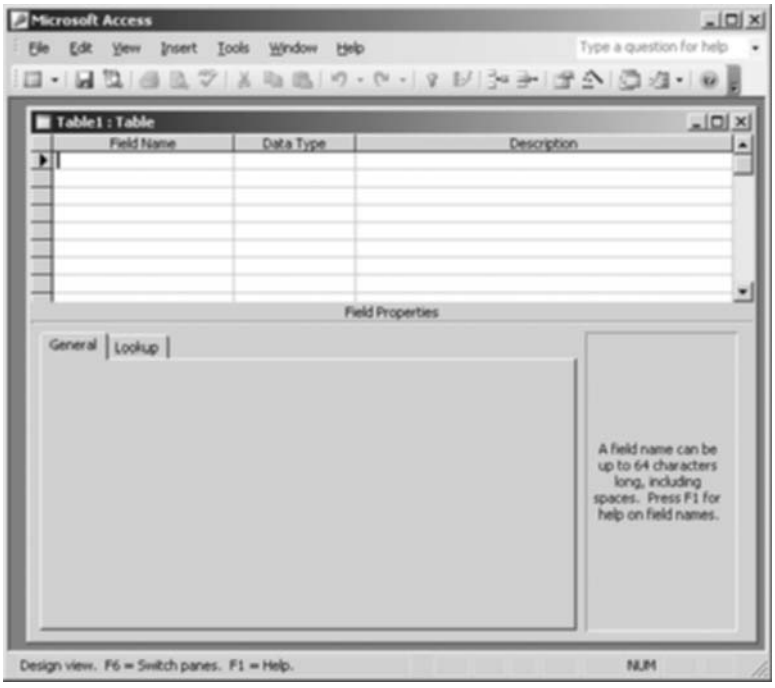


Fig. 3.75 **The design view of Table1 : Table page in the Microsoft Access window**

  2. Enter the name of the fields in the Field Name column and the data types in the Data Type column, as shown in Fig. 3.76.

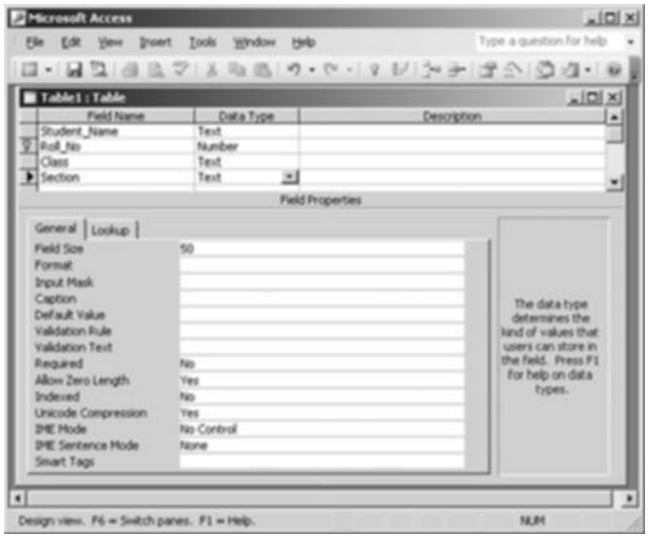


Fig. 3.76 **The Table1 : Table page with field names and their data types**

3. Close the Table1 : Table page. Before closing the Table1 : Table page, a message prompt appears, as shown in Fig. 3.77.

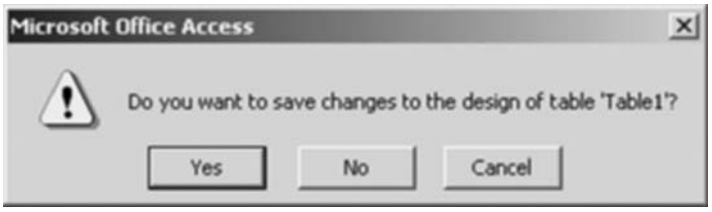


Fig. 3.77 **Message Prompt for saving the design of table**

4. Click the Yes button to display the Save As dialog box, as shown in Fig. 3.78.

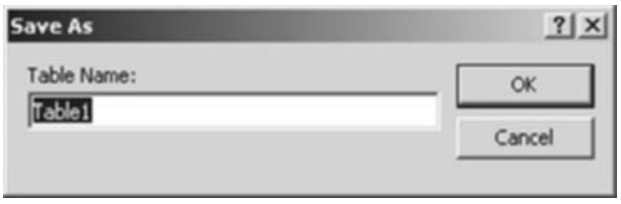


Fig. 3.78 **Saving the design of the table**

5. Enter the name of the table, say Student_Details, in the Table Name text box and click OK. A message prompt again appears, as shown in Fig. 3.79.

Fig. 3.79 **Prompting whether to define a primary key or not**

In Fig. 3.71, we can either select the Yes or No option depending upon our requirement of defining a primary key. A primary key is a single field or a combination of multiple fields that are used to uniquely identify a record in the database table.

6. Click the NO button to display the Microsoft Access-[Student Database: Database (Access 2000 file format)] window, as shown in Fig. 3.80.

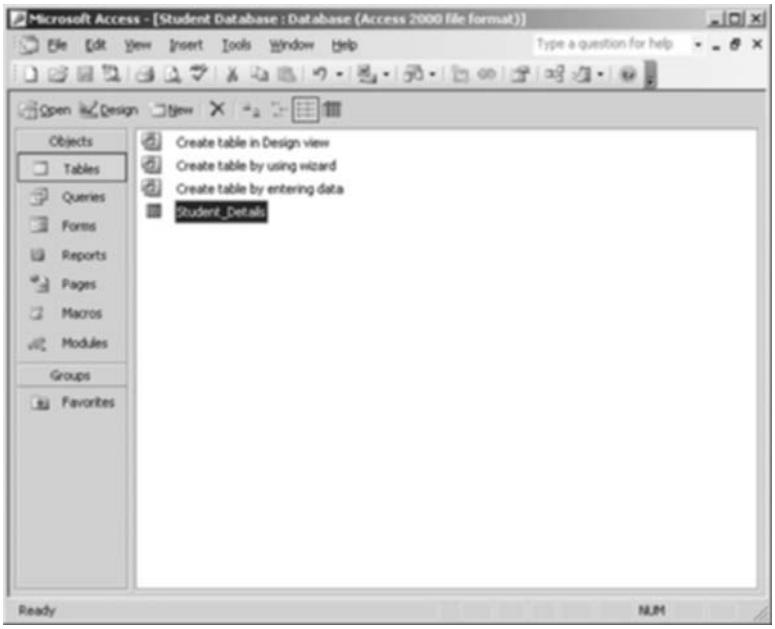


Fig. 3.80 **The Microsoft Access- [Student Database: Database (Access 2000 file format)] with the Student_Details table**

Entering data in a table In order to enter data in the Student_Details table, we need to perform the following steps:

1. Double-click the Student_Details icon to display the Microsoft Access-[Student_ Details:Table] window, as shown in Fig. 3.81.

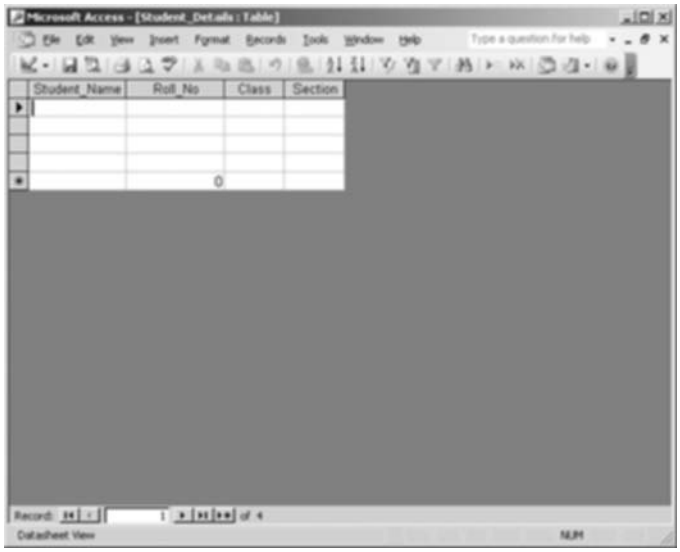


Fig. 3.81 The Microsoft Access – [Student_Details:Table] window

2. Enter data in the table, as shown in Fig. 3.82.

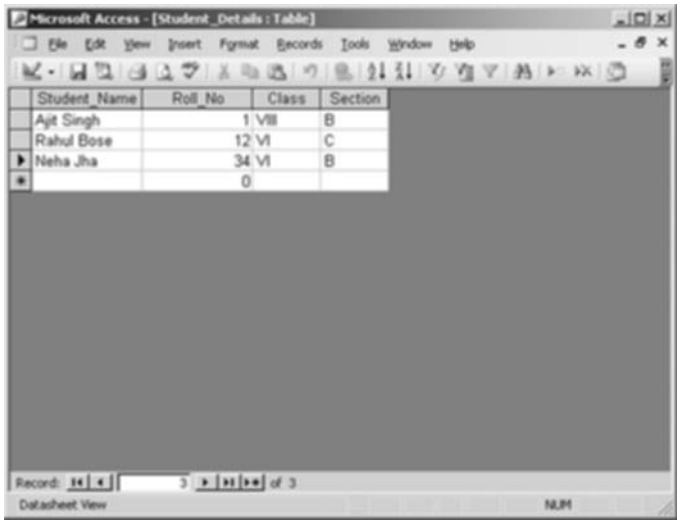


Fig. 3.82 The Microsoft Access – [Student_Details:Table] window with the entered data

3. Close the Microsoft Access-[Student_Details:Table] window to display the Microsoft Access-[Student Database: Database (Access 2000 file format)] window.

Manipulating a table Manipulating a table involves the addition of new rows and columns to the table. A row represents a record in the table, whereas a column represents a field. For adding a record, we need to perform the following steps:

1. Open the Student_Details table to add new records.
2. Enter the data in the empty record marked with an asterisk (*), as shown in Fig. 3.83.

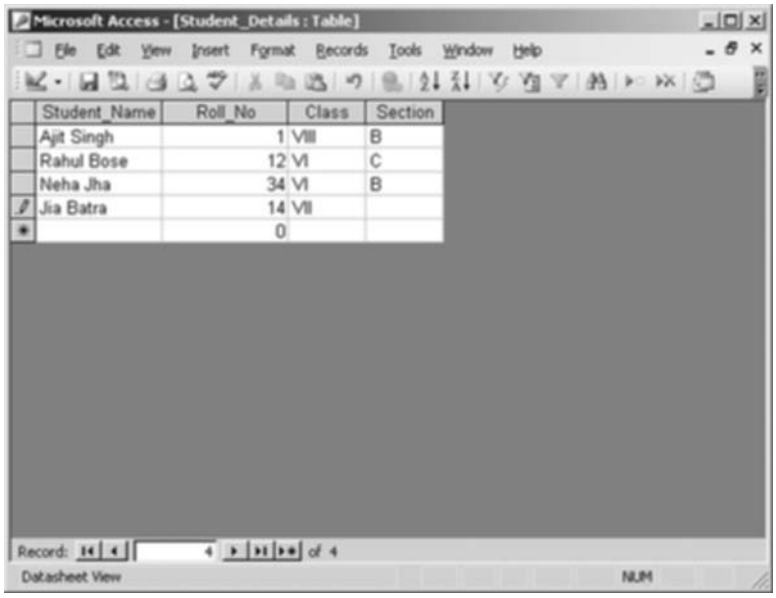


Fig. 3.83 The Microsoft Access-[Student_Details:Table] window a New Record Added

NOTE: The asterisk (*) shifts to the next record as soon as data is entered in the new record.

3. Close the Microsoft Access-[Student_Details:Table] window to display the Microsoft Access-[Student Database: Database (Access 2000 file format)] window.

For adding a new column, we need to perform the following steps:

1. Open the Student_Details table.
2. Select a field, say Class, before which the new field is to be added.
3. Select Insert → Column to insert a new blank field before the Class field, as shown in Fig. 3.84.

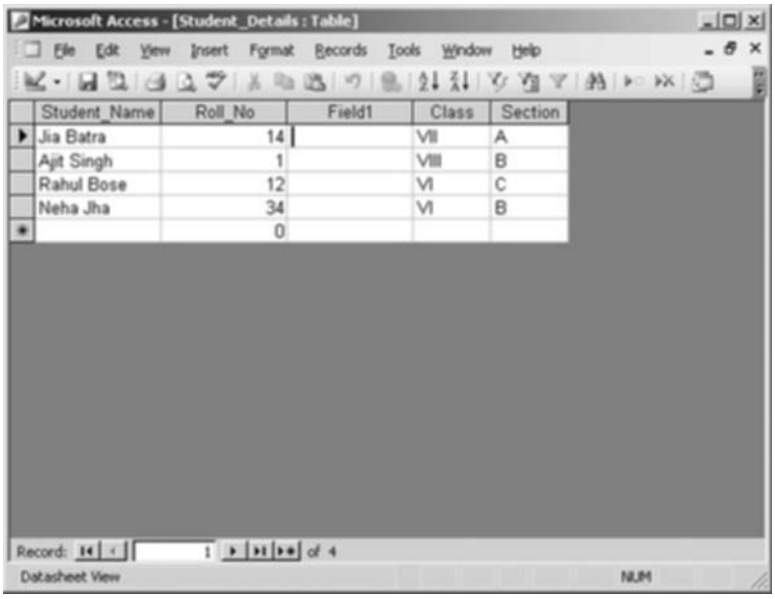


Fig. 3.84 **A new blank field inserted before the Class field**

4. Enter D_o_Admission as the name of the field by double-clicking Field1 column header.
5. Enter data in the D_o_Admission field, as shown in Fig. 3.85.

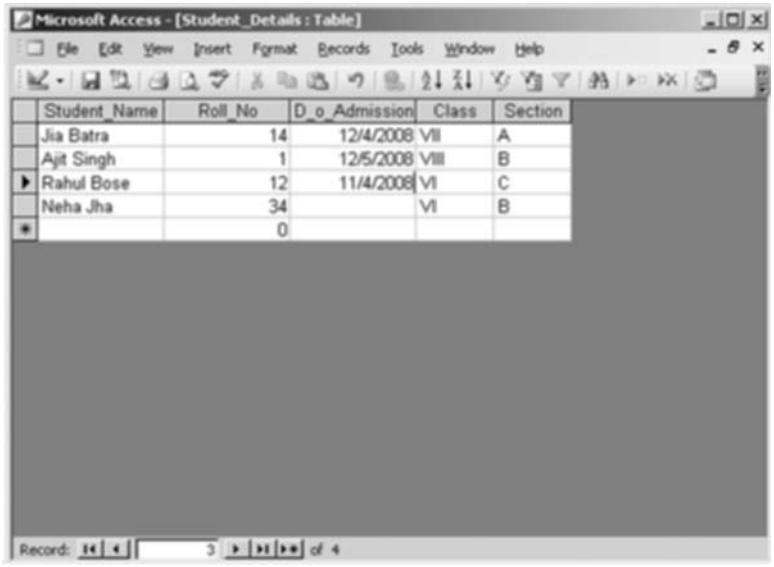


Fig. 3.85 **The newly inserted field**

6. Close the Microsoft Access-[Student_Details:Table] window to display the Microsoft Access-[Student Database: Database (Access 2000 file format)] window.

Editing a table Editing a table involves changing the existing data in the table. For editing a table, we need to perform the following steps:

1. Open the Student_Details table for editing the records.
2. Select the cell for which the value has to be changed, e.g., we can change the Roll_No of Ajit Singh from 1 to 39, as shown in Fig. 3.86.

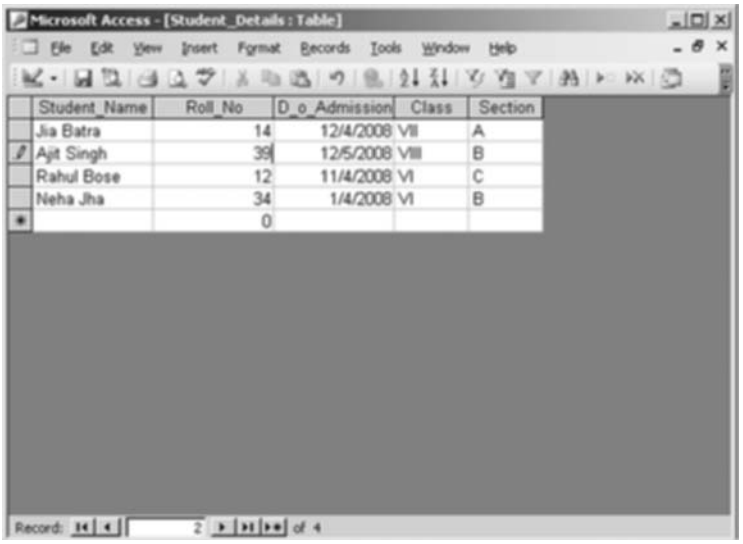


Fig. 3.86 The Microsoft Access – [Student_Details:Table] window with the changed value

3. Close the Microsoft Access-[Student_Details:Table] window to display the Microsoft Access-[Student Database: Database (Access 2000 file format)] window.

Deleting a record In order to delete a record, we need to perform the following steps:

1. Select the record that is to be deleted.
2. Select Edit → Delete Record to display a message prompt confirming the deletion of record, as shown in Fig. 3.87.

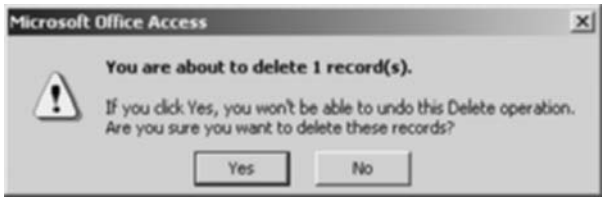


Fig. 3.87 Message prompt confirming the deletion of record

3. Click the Yes button to permanently delete the record. Figure 3.88 shows the Microsoft Access-[Student_Details:Table] window after the deletion of the third record.

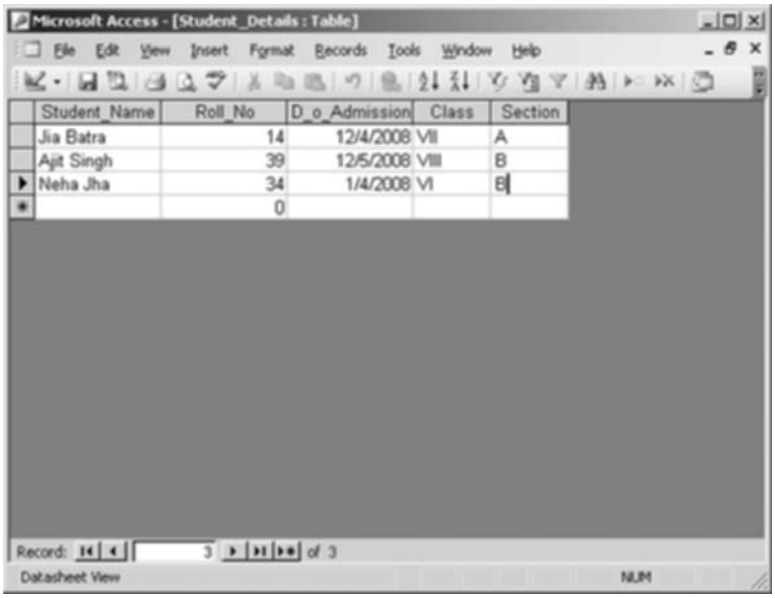


Fig. 3.88 The Microsoft Access- [Student_Details:Table] window after the deletion of a record.

4. Close the Microsoft Access-[Student_Details:Table] window to display the Microsoft Access-[Student Database: Database (Access 2000 file format)] window.

Deleting a table In order to delete a table from the database, we need to perform the following steps:

1. Open the Students database.
2. Right-click on the table that is to be deleted to display a shortcut menu.
3. Select the Delete option to delete the Student_Details table. A message prompt appears confirming the deletion of table, as shown in Fig. 3.89.

Fig. 3.89 Message prompt confirming the deletion of table

4. Click the Yes button to delete the table permanently. The Microsoft Access- [Student Database: Database (Access 2000 file format)] window appears.

Defining relationships In MS Access, relationships between two or more tables of a database help to create a link between the two tables. It links the two tables with the help of the columns of same types present in both the tables. The linking of same columns helps in reducing the duplicacy and inconsistency in data. Three types of relationships can be defined for databases:

- **One-to-one** One-to-one relationship maps one record of a table to a single record of another table in the same database.
- **One-to-many** One-to-many relationship maps one record of a table in a database to multiple records of another table in the same database.
- **Many-to-many** Many-to-many relationship maps multiple records of a table to multiple records of another table in the same database. To create a relationship between two tables of a database, we need to perform the following steps:

Select Tools → Relationships to display Show Table dialog box containing the names of all the tables contained in the database, as shown in Fig. 3.90.

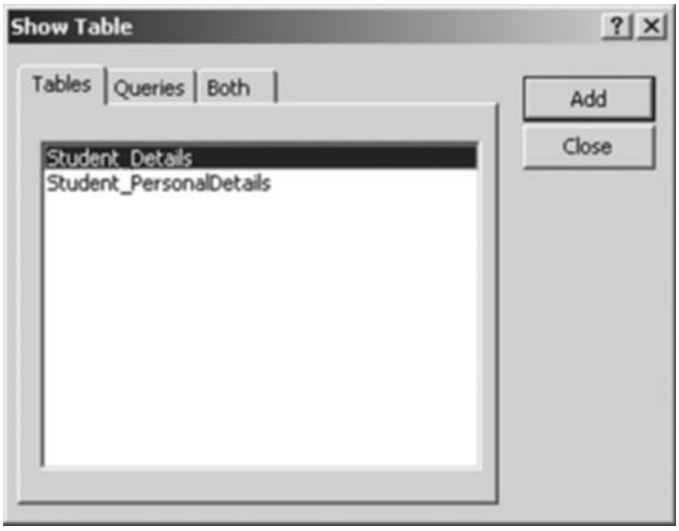


Fig. 3.90 The Show Table dialog box

2. Select a table name and click the Add button to add a relationship to it.
3. Click the Close button to close the Show Table dialog box and display the Relationships window, as shown in Fig. 3.91.
4. Drag the Roll_No field, which is present in the Student_Details table to the Roll_No field in the Student_PersonelData table to display the Edit Relationships dialog box, as shown in Fig. 3.92.

NOTE: The Enforce Referential Integrity option present in the Edit Relationships dialog box ensures that the relationships are valid and prevents the accidental deletion of data from the database.

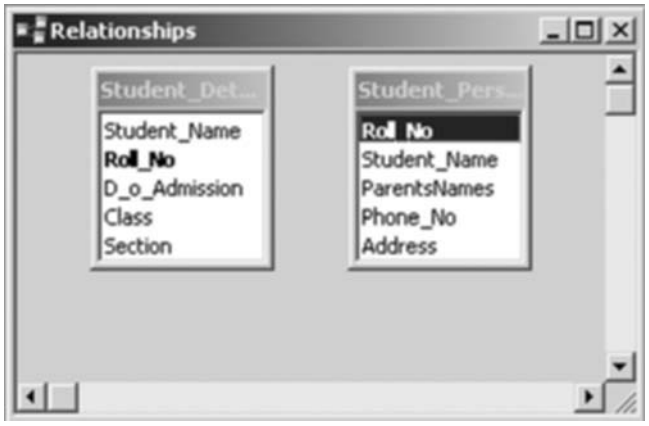


Fig. 3.91 **The Relationships window**

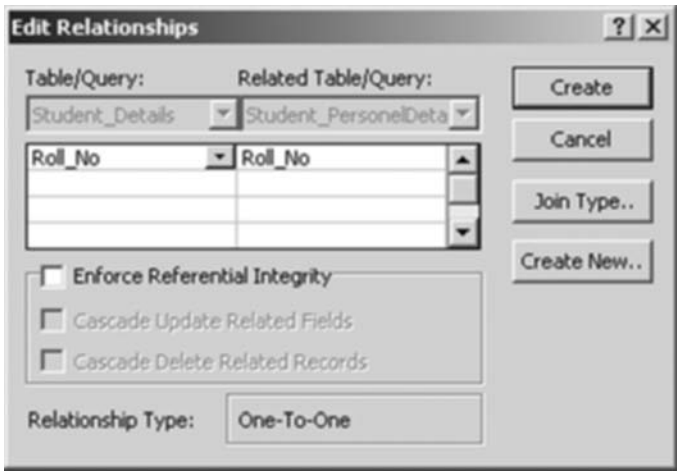


Fig. 3.92 **The Edit Relationships dialog box**

5. Click the Create button to establish a relationship between the two tables. Figure 3.93 shows the Relationships window displaying a relation created between the two tables.
6. Close the Relationships window. A message prompt appears asking whether to save the changes or not, as shown in Fig. 3.94.
7. Click the Yes button to save the relationship.

Creating a database query Query is a request through which a record or a set of records can be accessed conditionally from a database. On the basis of the query, only the records fulfilling the specified condition are displayed in the result. There exists two ways of creating a query, which are as follows:

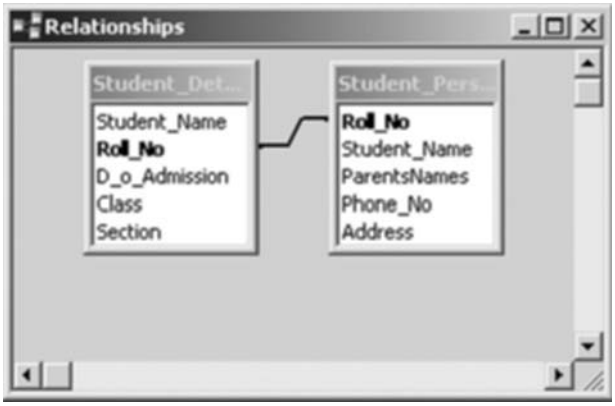


Fig. 3.93 **The Relationships window showing a none-to-one relationship**

Fig. 3.94 **Message prompt asking whether to save the changes or not**

- Create a query in Design view
- Create a query using wizard

The most commonly used method for creating a query in MS Access is by using the Create query in Design view option. For creating a query in MS Access using the Create query in Design view option, we need to perform the following steps:

1. Click the Queries button in left pane of the Microsoft Access - [Student Database: Database (Access 2000 file format)] window to display the various query options, as shown in Fig. 3.95.
2. Double-click the Create query in Design view option to display the Show Table window.
3. Click the Add button to add the tables from which data are to be extracted.
4. Click the Close button to close the Show Table window. The Microsoft Access - [Query1: Select Query] window appears, as shown in Fig. 3.96.
5. Select the field name to be displayed from the Field list and the table name from the Table list.
6. Specify the condition on the basis of which data are to be extracted in the Criteria field, as shown in Fig. 3.97.

Fig. 3.95 Displaying the available query options

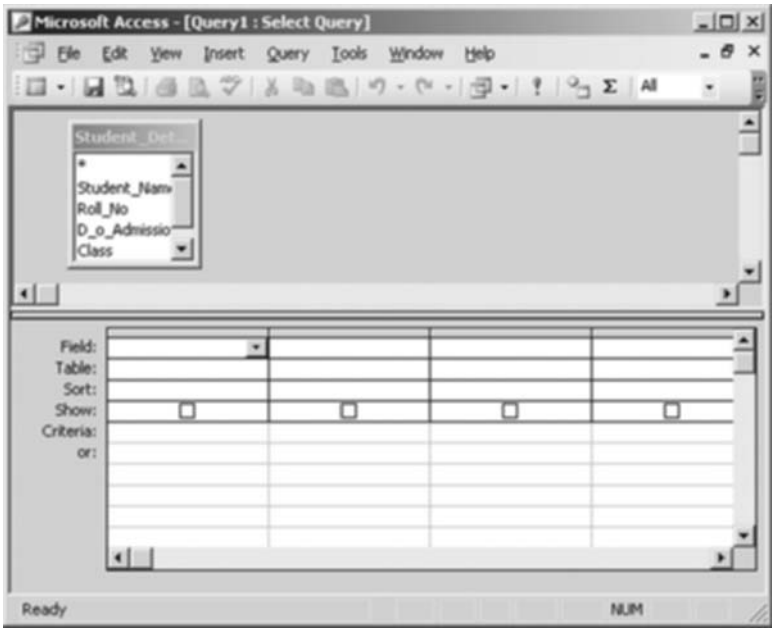


Fig. 3.96 The Microsoft Access-[Query1: Select Query] window

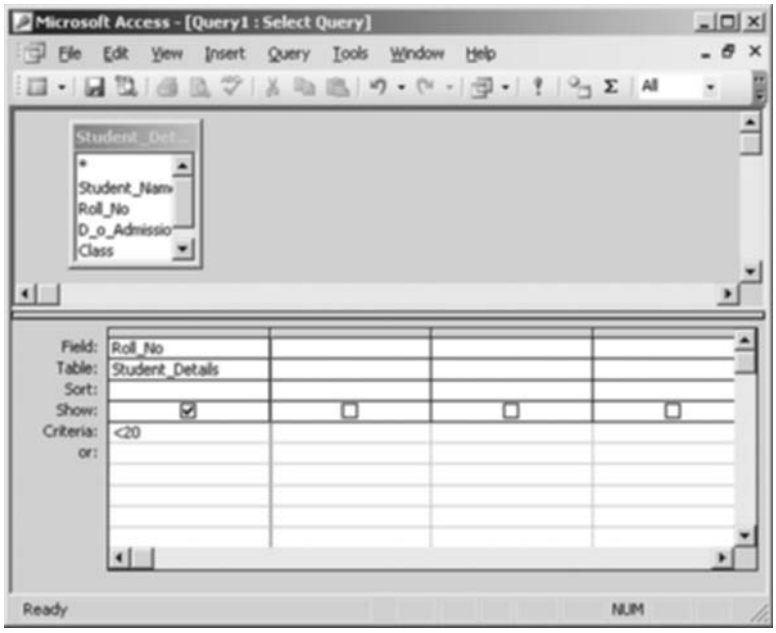


Fig. 3.97 **The Microsoft Access - [Query1:Select Query] window with field names and specified criteria**

7. Close the Microsoft Access - [Query1: Select Query] window. A message prompt appears asking whether to save the changes pertaining to the query or not, as shown in Fig. 3.98.

Fig. 3.74 **Prompting whether to save changes pertaining to the query**

8. Click the Yes button to save the query. Figure 3.99 shows the Save As dialog box.

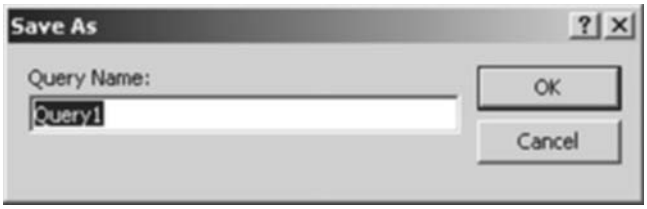


Fig. 3.99 **Saving the query name**

9. Enter the name of the query, say Query_Roll in the Query Name text box.
10. Click OK to save the query. The Microsoft Access - [Student Database: Database (Access 2000 file format)] window appears containing the name of the query.

To view the result of the query, perform the following steps:

1. Double-click the query name, Query_Roll in the Microsoft Access - [Student Database: Database (Access 2000 file format)] window to display the results of the query, as shown in Fig. 3.100.

Fig. 3.100 **The results of the Query_Roll query**

SUMMARY

In this chapter, we have learnt the standard programming techniques that are carried out during program development. Program development begins with identifying the problem and the associated parameters and constraints. The overall structure of the program is then represented with the help of a hierarchy chart. Algorithms and flowcharts are used to design the flow of program control and ascertain the various programming constructs that are to be used during program development. Finally, the pseudocode of the program is developed using statements written in generic syntax.

In the past two decades, Microsoft has developed robust, efficient and user-friendly operating systems and application software. Some of the commonly used Microsoft-based application software for performing utility tasks include MS Word, MS Excel, MS PowerPoint and MS Access. MS Word is generally used for creating professional as well as personal documents in an efficient manner. The basic operations performed in MS Word include saving, editing, formatting and printing a

document. MS Excel is a spreadsheet application program that allows us to create spreadsheets. A spreadsheet comprises rows and columns with the data stored in the intersection of rows and columns, i.e. cells. MS PowerPoint is an application software that enables us to create presentations. A PowerPoint presentation comprises slides that contain text, images, charts, tree-diagrams, etc. for presenting the information in an attractive and concise manner. The basic operations that can be performed in MS PowerPoint include creating a presentation, saving a presentation, adding slides to a presentation, printing a presentation, etc.

In addition to Word, Excel, and PowerPoint, MS Office Suite also comes bundled with a very handy database management system, MS Access. It is a relational database management system that allows us to create databases and also define relationships between the various tables of the databases. In order to access records from a database, queries are created. A query retrieves the data from the database on the basis of some pre-defined criteria.

POINTS TO REMEMBER

- **Computer program**: It is a set of instructions, which are provided to a computer for performing a specific task.
- **Hierarchy chart**: It is a solution approach that suggests a top–down solution of a problem.
- **Algorithm**: It is a sequence of steps written in the form of English phrases that specify the tasks that are to be performed while solving a problem.
- **Flowchart**: A flowchart is the graphical representation of the flow of control and logic in the solution of a problem.
- **Pseudocode**: It is pretty much similar to algorithms. It uses generic syntax for describing the steps that are to be performed for solving a problem.
- **MS Word**: It is an application software bundled in MS Office package that allows us to create, edit, save and print personal as well as professional documents in a very simple and efficient manner.
- **MS Excel**: It is a spreadsheet application program that enables the users to create the spreadsheets.
- **Row**: It is a horizontal sequence of data stored in a spreadsheet.
- **Column**: It is a vertical sequence of data stored in a spreadsheet.
- **Worksheet**: It is the actual working area of a spreadsheet program that comprises rows and columns.
- **Workbook**: It is a combination of multiple worksheets in MS Excel.
- **Graphs**: Graphs are the pictorial representation of complex data.
- **MS Access**: It is a database management system that allows the users to create databases.
- **Database**: It is a collection of data organized in the form of tables.
- **Table**: It is a collection of rows and columns where data are actually stored.
- **MS PowerPoint**: It is an application software included in the MS Office package that allows us to create presentations.
- **Slide**: It is the fundamental unit of a PowerPoint presentation containing a combination of images, text, graphics, charts, etc.

REVIEW QUESTIONS

1. A computer program is a set of instructions, which are provided to a computer for performing a specific task.
2. A flowchart is a pictorial representation of a process, which describes the sequence and flow of the control and information in a process.
3. Start and end in a flow chart are represented by a rectangle.
4. Algorithms help a programmer in breaking down the solution of a problem into a number of sequential steps.
5. Time complexity specifies the amount of time required by an algorithm for performing the desired task.
6. Space complexity specifies the amount of memory space required by an algorithm for performing the desired task.
7. If the Standard toolbar is not visible in the MS Word window, then we need to select Window → Toolbars → Standard to make it visible.
8. In a flow chart, input and output are represented by an oval.
9. In a flow chart, decision or conditional statements are represented by a parallelogram.
10. Pseudocode is written using specific syntax of a programming language.
11. We can launch MS PowerPoint by typing powerpnt in the Run dialog box.
12. We can launch MS Access by typing access in the Run dialog box.
13. Title bar displays the name of the document while working in MS-word.
14. The horizontal sequence in which the data are stored is referred to as a column in MS excel.
15. We can launch MS Excel by typing msexcel in the Run dialog box.
16. We can launch MS Word by typing winword in the Run dialog box.
17. MS Power point does not support graphics.
18. MS Access is a RDBMS.
19. Data in MS access are stored in the form of tables.

1. _____ helps a programmer in breaking down the solution of a problem into a number of sequential steps.
2. _____ can be defined as the pictorial representation of a process, which describes the sequence and flow of the control and information in a process.
3. _____ focuses on the logic of the program.

4. _____ is a set of instructions, which are provided to a computer for performing a specific task.
5. _____ is an application software included in MS Office for working with documents.
6. MS Word can be accessed either using _____ or _____.
7. MS Word uses a _____ interface to interact with the users.
8. The horizontal bar at the top of the MS Word window is called _____.
9. The blinking bar in MS Word that indicates the position of the next key stroke or the character to be inserted is called _____.
10. The feature in the MS Word that helps the user to view the appearance of the document before printing is called _____.
11. _____ is a spreadsheet application program that is widely used in business applications.
12. The horizontal sequence of data stored in a spreadsheet is known as _____.
13. The vertical sequence of data stored in a spreadsheet is known as _____.
14. Complex data in MS-Excel is stored pictorially in the form of _____.
15. _____ is an application software included in MS Office package for creating presentations.
16. The presentations in the MS PowerPoint are usually saved with the _____ extension.
17. _____ is the Microsoft-based database management system that allows the users to create databases.
18. You can type _____ in the Run dialog box to launch MS Excel.

MULTIPLE CHOICE QUESTIONS

1. Which one of the following is typed in the Run dialog box to access MS Word?
   A. winword B. word
   C. msword D. msdoc
2. Which of the following is a word-processing program?
   A. MS Excel B. MS DOS
   C. MS Word D. MS PowerPoint
3. Which of the following is a spreadsheet application program?
   A. MS Access B. MS Word
   C. MS Excel D. MS-DOS
4. Which of the following is the extension used for MS PowerPoint files?
   A. doc B. ppt
   C. pnt D. pwpt
5. Which of the following is a database management system used for creating tables?
   A. MS Access B. MS Excel
   C. MS Dos D. All of the above

6. Pseudocode uses the following format for depicting the logic of a program:
   A. C language syntax B. Assembly language syntax
   C. Generic syntax D. Machine language syntax
7. Decision-making statements are represented with which of the following symbols in a flowchart?
   A. Circle B. Rectangle
   C. Oval D. Rhombus
8. Start and end program statements are represented with which of the following symbols in a flowchart?
   A. Circle B. Rectangle
   C. Oval D. Rhombus
9. What is the name of the task pane used for designing slides in MS PowerPoint?
   A. Slide Design B. Slide Layout
   C. Design Slide D. None of the above
10. What is the intersection of row and column called in MS Excel?
   A. Cell B. Worksheet
   C. Row D. Column
11. What is the combination of worksheets in MS Excel called?
   A. Workbook B. Spreadsheet
   C. Excel sheet D. None of the above
12. Which of the following is not an Office package?
   A. MS Office B. Google Docs
   C. Open Office D. MS DOS
13. 'Ctrl + X' command is used for which of the following tasks in an MS Office package?
   A. Copy B. Move
   C. Save D. Delete
14. Which of the following is not present in an MS Office package?
   A. Toolbar B. Title bar
   C. Menu bar D. Task bar
15. Which of the following will not save a document in MS word?
   A. Ctrl + S B. Alt + F + S
   C. Shift + S D. File → Save

ANSWERS

True or False

1. True	2. True	3. False	4. True	5. True	6. True	7. False	8. False
9. False	10. False	11. True	12. False	13. True	14. False	15. False	16. True
17. False	18. True	19. True					

Fill in the Blanks

1. Algorithm
2. Flowchart
3. Pseudocode
4. Computer program
5. MS Word
6. Run command or Start menu
7. GUI
8. Title bar
9. Cursor
10. Print Preview
11. MS Excel
12. Row
13. Column
14. Chart
15. MS PowerPoint
16. ppt
17. MS Access18. excel

Multiple Choice Questions

1. A	2. C	3. C	4. B	5. A	6. C	7. D	8. C
9. B	10. A	11. A	12. D	13. B	14. D	15. C	

EXERCISE QUESTIONS

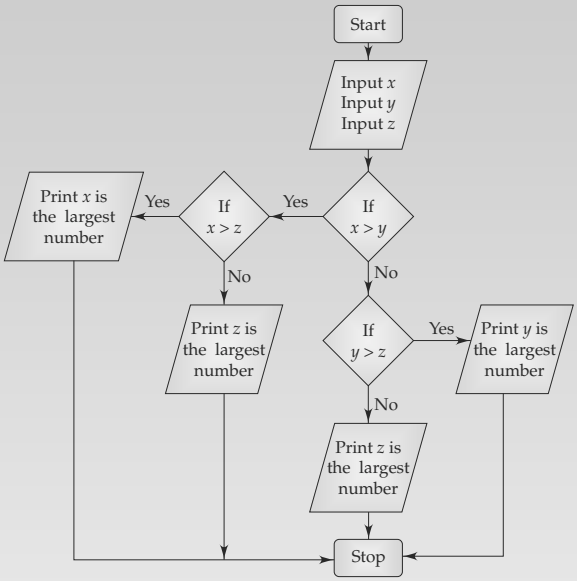
1. What is the basic use of MS word? Explain with the help of an example.
2. What are the basic operations performed on a Word document? Briefly explain five of these operations.
3. What do you mean by MS-Excel? Explain the different ways of starting MS-Excel in a computer system.
4. What are the different operations possible on a worksheet in MS-Excel?
5. What are the different methods of accessing MS PowerPoint?
6. What is the difference between creating and designing a new presentation in MS PowerPoint?
7. How can a new slide be added to a presentation in MS PowerPoint?
8. What is MS Access? How can we create database in MS Access?
9. Explain the method of creating tables in MS Access with design view.
10. What do you mean by queries? How can we create queries in MS Access?
11. What is the function of Slide Design option in MS PowerPoint?
12. Explain the basic operations performed on a presentation.
13. Define the following terms pertaining to MS-PowerPoint:
    (a) Slide pane
    (b) Task pane
    (c) Notes pane
    (d) Status bar

ANSWERS TO 2009 QUESTION PAPERS

PART A

1. Draw a flowchart to find the maximum among the three numbers. (GE2112)

   Ans:



2. Name any four application software packages. (GE2112)

   Ans: MS Excel, MS power point, MS word and MS Access

3. Compare and contrast flowchart and algorithm. (Anna University, Jan - Feb 2009)

   Ans: Refer Sections 3.3.2 and 3.3.3

4. What is meant by pseudocode? (Anna University, Jan - Feb 2009)

   Ans: Refer Section 3.3.4

5. Mention the characteristics necessary for a sequence of instructions to qualify as an algorithm. (GE1102)

   Ans: Refer Section 3.3.2

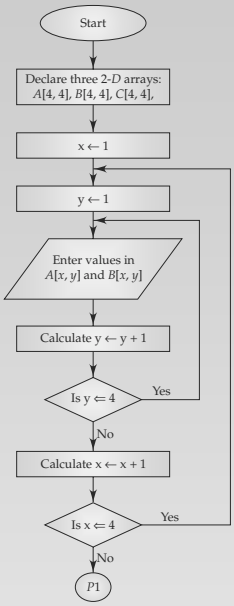
6. What are the basic logic structures used in writing structured program? (GE1102)

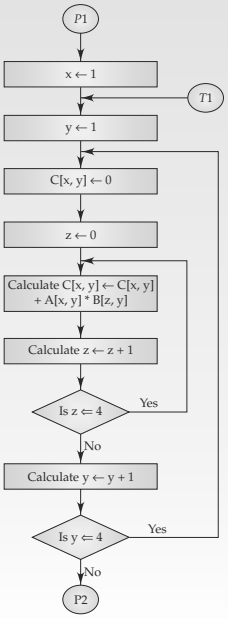
   Ans: Refer Section 3.4

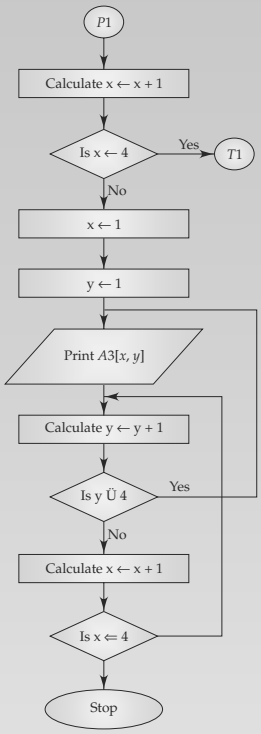
PART B

1. Draw a flowchart to multiply two matrices. (GE2112)

Ans:







2. Write the pseudocode to multiply two matrices. (GE2112)

Ans:

Assumptions

- The two given matrices are M1 and M2
- M1 is a pq matrix
- M2 is a qr matrix
- M3 is the matrix used for storing the product of M1 and M2.

Pseudocode

```
if columns_in_[M1] NOTEQUALTO rows_in_[M2]
    error message "Size of matrices are not compatible for multiplication "
else for x ← 1 to rows_in_[M1]
do for y ← 1 to columns_in_[M2]
    do M3[x,y] ← 0
    for z ← 1 to columns_in_[M1]
        do M3[x,y] ← M3[ x, y ] + M1[ x , z ]* M2[z,y]
            return M3
```

3. What is an algorithm? Write an algorithm to print even numbers from 2 to 100. (GE2112)

Ans:

What is an algorithm Refer Section 3.3.2

The algorithm to print even numbers from 2 to 100 is as follows.

```
Step 1: Start
Step 2: Initialize NUM ← 2
Step 3: Execute Steps 4 and 5 Till NUM EQUAL100
Step 4: Calculate NUM ← NUM + 2
Step 5: Print NUM
Step 6: Stop
```

4. Write the various steps involved in the program development cycle. (Anna University, Jan - Feb 2009)

Ans: Refer Section 3.2

5. Give the advantages and limitations of: (Anna University, Jan - Feb 2009)
   1. Flowcharts
   2. Pseudocode

Ans:

Advantages and limitations of flowchart Refer Section 3.3.3.

Advantages of pseudocode

- Pseudocode is easy to comprehend as it uses English phrases for writing program instructions.
- Developing program code using pseudocode is easier in comparison to developing the program code from scratch.
- The pseudocode instructions are easier to modify in comparison to a flowchart.

Limitations of pseudocode

- Pseudocode does not provide any kind of pictorial representations for specifying the flow of program control. Hence, it may at times become difficult to understand the program.
- Developing a pseudocode is tedious in comparison to drawing a flowchart.
- There is no standard format for developing a pseudocode. Therefore, it may become a challenge to use the same pseudocode by different programmers.

6. Describe briefly the key features supported by modern word-processing packages. (GE1102)

Ans: Refer Section 3.7

7. Mention the various guidelines to be followed while drawing a flowchart. Discuss the advantages and limitations of a flowchart. (GE1102)

Ans: Some of the standard guidelines that must be followed while designing a flowchart are:

- It must begin with 'Start' and 'Stop' keyword.
- The instructions specified in the flowchart must be crisp and concise.
- The arrows must be aligned properly so as to clearly depict the flow of program control.
- The use of connectors should be generally avoided as they make the program look more complex.

Advantages and disadvantages of flow chart

Refer Section 3.3.3

8. Explain the various stages involved in program design. (GE1102)

Ans: Refer Section 3.2.

ANSWERS TO 2010 AND 2011 QUESTION PAPERS

SHORT ANSWER QUESTIONS

1. What is a web server? (AU, Chn, Jan 2010)

Ans: A Web server is a computer that receives client requests over the internet and sends back appropriate response objects. The browsing of a Web site over the Internet is a typical example where the host Web server receives HTTP request from the remote user and sends back requested Web pages in response.

2. What is an algorithm? (AU, Chn, Jan 2010)

Ans:

Algorithm An algorithm is a method of representing the step-by-step procedure for solving a problem. An algorithm is very useful for finding the right answer to a simple or a difficult problem by breaking it into simple cases.

Example Following is an algorithm for in-order traversal (recursive):

1. Step 1: check if tree is empty by verifying root pointer R. Print tree empty if R=NULL
2. Step 2: if leftptr(R)!=NULL then call in-order (leftptr(R))
3. Step 3: print data(R).
4. Step 4: if rightptr(R)!= NULL then call in-order(rightptr(R))
5. Step 5: return.

3. What are the applications of software? (AU, Chn, Jan 2011)

Ans:

Applications of Software

The various applications of software are:

- It enables the users to perform their desired tasks.
- It controls and coordinates the hardware components and manages their performances.

4. What are the characteristics of algorithms in computer? (AU, Chn, Jan 2010)

   Ans: Refer Section 3.3.2.

5. What is a Pseudocode? (AU, Mdu, Jan 2011)

   Ans: Refer Section 3.3.4

6. What is a space complexity? Why it is used? (AU, Mdu, Jan 2011)

   Ans:

   Space complexity and its Use

   Space complexity specifies the amount of memory space required by an algorithm for performing the desired task. It is quite possible that there are multiple algorithms available for solving a particular problem. The assessment of space complexity helps to choose the most efficient algorithm for solving a problem.

7. Define a Pseudocode. (AU, Cbe, Dec 10-Jan 11)

   Ans:

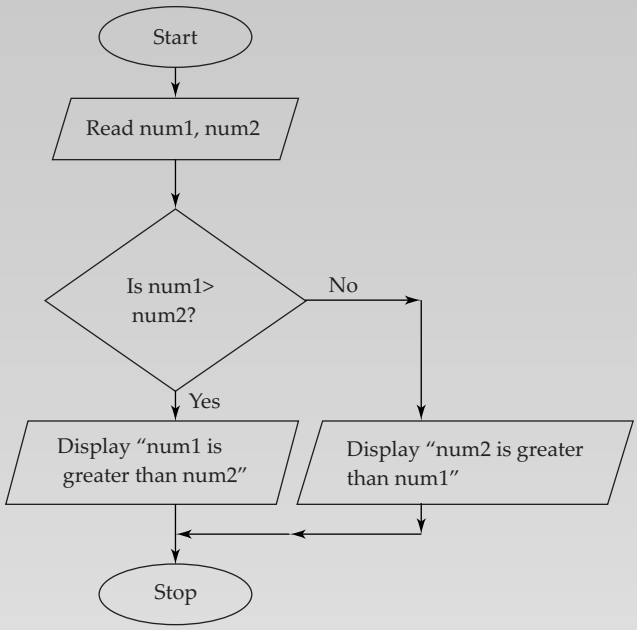
   Pseudocode

   Pseudocode uses generic syntax for describing the steps that are to be performed for solving a problem. Along with the statements written using generic syntax, pseudocode can also use English phrases for describing an action.

   Sample Pseudocode

```
BEGIN
DEFINE: Integer i, sum
SET: sum=0
SET:  i=0
WHILE: i >= 0 AND <= 100
COMPUTE: sum = sum + i
i = i + 2
END WHILE
DISPLAY: sum
END
```

8. Draw a flow chart to trace the larger of two integers. (AU, Cbe, Dec 10-Jan 11)



9. Write an algorithm to compute the factorial of a number. (AU, Cbe, Dec 10-Jan 11)

Ans:

Algorithm

```
Step 1 - Start
Step 2 - Initialize fact = 1
Step 3 – Read num
Step 4 – Initialize the looping counter i = num
Step 5 – Repeat Steps 6-7 while i>=1
Step 6 – fact = fact * i;
Step 7 - i=i-1;
Step 8 - Display fact as the factorial value of num
Step 9 - Stop
```

10. Mention the features of Word Processor.

Ans:

Features of Word Processor: Some of the key features of a word processor are:

- Formatting support to manage the look and feel of the text
- Table feature for creating and managing tabular data

- Pagination support for paginating the document in desired format
- Chart feature for adding charts
- Graphic support for adding graphics in a document

11. What is an algorithm? Give the characteristics of algorithm. (AU, TIR, Dec 10-Jan 11)

Ans: Refer Section 3.3.2.

12. What is Pseudo code? (AU, TRI, Jan 2011)

Ans: Refer Section 3.3.4.

13. State the difference between Programming language and a package. (AU, TRI, Jan 2011)

Ans:

Difference between Programming Language and Package

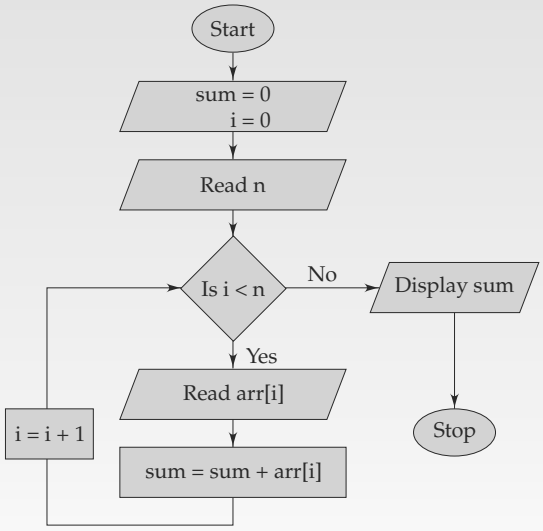
Programming Language	Package
It helps in writing the program code.	It helps to perform a specific task.
Example: C, C++, Java	**Example**: MS Word, Adobe Photoshop, Windows Media Player

DESCRIPTIVE QUESTIONS

1. Draw and explain the various symbols of flowchart and also draw the flowchart to add an array of N elements. (AU, Chn, Jan 2010)

Ans: Flowchart Symbols: Refer Section 3.3.3.

Flowchart to add an array of N elements.



2. Explain the features of PowerPoint package. (AU, Chn, Jan 2010)

Ans:

Features of PowerPoint Package

A PowerPoint package is a presentaion software that is used to create engaging presentations in the form of slide shows. They are typically used by a presenter while giving presentations to an audience. PowerPoint presentations find a great amount of usage in business, teaching and training fields. Some of the key features of PowerPoint package are:

- It allows text, graphics, animations and videos to be used while creating presentation slides
- It supports a number of built-in graphical and smart objects that can be used to present information effectively
- It allows the users to add special transitional effects into the presentation
- It supports rich formatting features for formatting text and graphical elements
- It allows the users to embed Word and Spreadsheet objects inside the PowerPoint presentation
- It supports a number of built-in templates for quick presentation development
- The concept of Master slide helps the users to ensure consistent look and feel throughout the presentation

3. List and explain the features supported by spreadsheet package. (AU, Chn, Jan 2010)

Ans:

Features of Spreadsheet Package

A Spreadsheet package helps create and manage tabular data. It is quite a handy tool for accountants and users from financial background as it supports a number of built-in formulas for performing common computational tasks. It is also used for creating and managing large databases of tabular records. Some of the key features of Spreadsheet package are:

- It allows users to store data in the form of rows and columns
- It supports a number of built-in tools such as Autofilter and data validation that help the users to quickly search and update relevant records
- The Graphs and Charts feature of the spreadsheet package allows the users to present the numerical data in the form of informational graphical figures
- It allows users to create custom formulas as per their requirements
- It supports a number of formatting features for formatting the tabular data
- The conditional formatting feature helps format specific rows and columns only when certain user-specified conditions are met
- It allows the users to maintain multiple worksheets inside the main spreadsheet

4. Explain with the diagram symbols used in the flowchart and the basic design structures in flowchart. (AU, Chn, Jan 2011)

Ans: Refer Section 3.3.3

5. Explain sequences logic, selection logic and iteration logic design structure in the pseudo code. (AU, Chn, Jan 2011)

   Ans: Refer Section 3.4

6. Write an algorithm to find the factors of a given number. (AU, Mdu, Jan 2011)

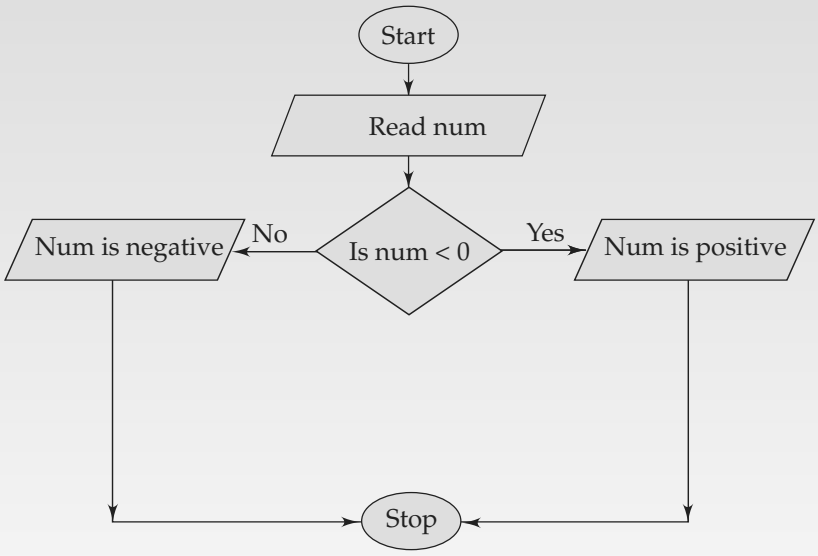
   Ans:

   Algorithm

```
Step 1 - Start
Step 2 – Accept a number from the user (num)
Step 3 – Initialize looping counter i = 1
Step 4 – Repeat Step 5-7 while i < num
Step 5 – If remainder of num divided by i (num % i) is Zero then goto Step 6
         else goto Step 7
Step 6 - Display i
Step 7 – Set i = i + 1
Step 9 - Stop
```

7. Give a flow chart for checking whether a number is positive or negative? (AU, Mdu, Jan 2011)

   Ans:



8. Mention the guidelines in detail while drawing a flowchart and list out the merits and demerits of flowcharting. (AU, Cbe, Dec 10-Jan 11)

Ans:

Guidelines for Drawing a Flowchart: Some of the important guidelines to be followed while drawing a flowchart are:

- It must begin with "Start" keyword and end with "Stop" keyword.
- The instructions specified in the flowchart must be crisp and concise.
- Correct symbols must be used for depicting various actions.
- The arrows must be aligned properly so as to clearly depict the flow of program control.
- The use of connectors should be generally avoided as they make the program look more complex.

Advantages and Disadvantages of Flowcharts: Refer Section 3.3.3.

9. Write a Pseudocode to find the given year is a leap year or not. (AU, Cbe, Dec 10-Jan 11)

Ans:

Pseudocode

```
BEGIN
DEFINE: Integer year
DISPLAY: "Enter the year value: "
READ: year
IF: year%4=0
DISPLAY: "'year' is a leap year"
ELSE
DISPLAY: "'year' is not a leap year"
END IF
END
```

10. Write the characteristics and qualities of a good algorithm. (AU, Cbe, Dec 10-Jan 11)

Ans: Refer Section 3.3.2.

11. Describe the application software packages. (AU, Cbe, Dec 10-Jan 11)

or

12. Write short note on application software . (AU, TRI, Jan 2011)

Ans: Refer Section 3.5.

13. Explain the need for an algorithm and highlight its advantages. (AU, TRI, Jan 2011)

Ans:

Need for an Algorithm: An algorithm is a sequence of steps written in the form of English phrases that specify the tasks that are performed while solving a problem. The need of an algorithm in program development is highlighted with the help of following points:

- It allows a programmer to break down the solution of a problem into a number of simplified sequential steps.

- It helps a programmer in specifying the variable names and types that would be used in solving a problem.
- It allows a programmer to choose the most efficient solution for a given problem by assessing the time and space complexities of different algorithms.

Advantages of Algorithm: The various advantages of algorithm are:

- It represents the program instructions in simple and concise form.
- It completely and definitely solves the given problem statement.
- It allows the programmer to use the most efficient logic to solve the given problem. (Time complexity)
- It allows the programmer to use a solution that requires minimum memory for its execution. (Space complexity)
- It eases the process of actual development of program code.

14. Write short note on office packages. (AU, TRI, Jan 2011)

Ans: Refer Section 3.6.

15. Explain the basic symbols used for constructing a flow chart and state the difference between a program flow chart and system flow chart. (AU, TRI, Jan 2011)

Ans: Refer Section 3.3.3.

Difference between Program and System Flowchart

Program Flowchart	System Flowchart
It depicts the flow of program control.	It depicts the flow of data through a data processing or information system.
Each box in a program flowchart represents an instruction.	Each box in a system flowchart represents a process or an operation.
The arrows in a program flowchart represent the order in which program instructions will be executed.	The arrows in a system flowchart represent the direction in which data will flow in the system.

INTRODUCTION TO C

4

CHAPTER OBJECTIVES

In this chapter, we will learn:

1. The overview of C programming language.
2. The character set, keywords and data types used in C.
3. The use of constants and variables in C.
4. The different operators and statements used in C.
5. The control statements used in C programs.
6. The various jumping statements.

CHAPTER OUTLINE

4.1 INTRODUCTION

C is a powerful, portable and elegantly structured programming language. It combines the features of a high-level language with the elements of an assembler and, therefore, it is suitable for writing both system software and application packages. It is the most widely used general-purpose language today. In fact, C has been used for implementing systems such as operating systems, compilers, linkers, word processors and utility packages. Figure 4.1 shows the various features of C that make it so much popular among programmers.

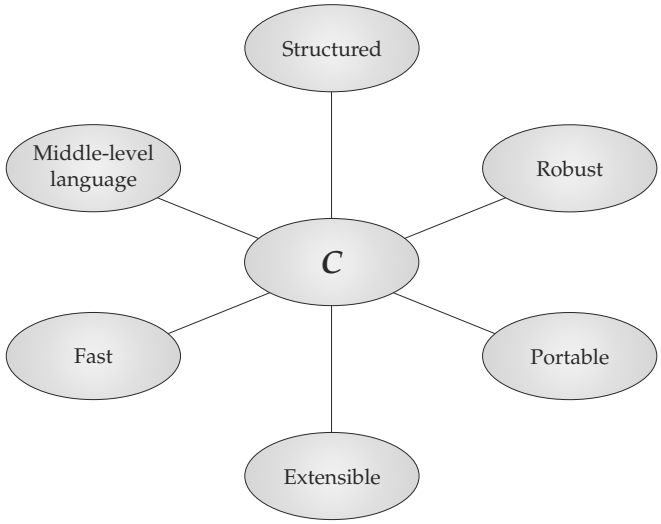


Fig. 4.1 Features of C

To write programs in C, we need to understand various terms such as characters, identifiers, keywords, constants, variables, data types, programming constructs, etc. All this shall form the basis of this chapter along with actual programs written in C language.

4.2 OVERVIEW OF C

The increasing popularity of C is probably due to its many desirable qualities. It is a robust language whose rich set of built-in functions and operators can be used to write any complex program. The C compiler combines the capabilities of an assembly language with the features of a high-level language and, therefore, it is well suited for writing both system software and business packages. In fact, many C compilers available in the market are written in C.

Programs written in C are efficient and fast. This is due to its variety of data types and powerful operators. It is many times faster than BASIC. For example, a program to increment a variable from 0 to 15000 takes about one second in C, while it takes more than 50 seconds in an interpreter of BASIC.

There are only 32 keywords in ANSI C, and its strength lies in its built-in functions. Several standard functions are available which can be used for developing programs. C is highly portable. This means that C programs written for one computer can be run on another with little or no modification. Portability is important if we plan to use a new computer with a different operating system.

C language is well suited for structured programming, thus requiring the user to think of a problem in terms of function modules or blocks. A proper collection of these modules would make a complete program. This modular structure makes program debugging, testing and maintenance easier.

Another important feature of C is its ability to extend itself. A C program is basically a collection of functions that are supported by the C library. We can continuously add our own functions to C library. With the availability of a large number of functions, the programming task becomes simple.

4.2.1 History of C

'C' seems a strange name for a programming language. But this strange-sounding language is one of the most popular computer languages today because it is a structured, high-level, machine-independent language. It allows software developers to develop programs without worrying about the hardware platforms where these will be implemented.

The root of all modern languages is ALGOL, introduced in the early 1960s. ALGOL was the first computer language to use a block structure. Although it never became popular in USA, it was widely used in Europe. ALGOL introduced the concept of structured programming to the computer science community. Computer scientists like Corrado Bohm, Guiseppe Jacopini and Edsger Dijkstra popularised this concept during 1960s. Subsequently, several languages were announced.

In 1967, Martin Richards developed a language called BCPL (Basic Combined Programming Language) primarily for writing system software. In 1970, Ken Thompson created a language using many features of BCPL and called it simply B. B was used to create early versions of UNIX operating system at Bell Laboratories. Both BCPL and B were 'typeless' system programming languages.

C was evolved from ALGOL, BCPL and B by Dennis Ritchie at the Bell Laboratories in 1972. C uses many concepts from these languages and added the concept of data types and other powerful features. Since it was developed along with the UNIX operating system, it is strongly associated with UNIX. This operating system, which was also developed at Bell Laboratories, was coded almost entirely in C. UNIX is one of the most popular network operating systems in use today and the heart of the Internet data superhighway.

For many years, C was used mainly in academic environments, but eventually with the release of many C compilers for commercial use and the increasing popularity of UNIX, it started receiving widespread support among computer professionals. Today, C is running under a variety of operating system and hardware platforms.

During 1970s, C had evolved into what is now known as 'traditional C'. The language became more popular after publication of the book *The C Programming Language* by Brian Kerningham and Dennis Ritchie in 1978. The book was so popular that the language came to be known as 'K&R C' among the programming community. The rapid growth of C led to the development of different versions of the language that were similar but often incompatible. This posed a serious problem for system developers.

To assure that the C language remains standard, in 1983, American National Standards Institute (ANSI) appointed a technical committee to define a standard for C. The committee approved a version of C in December 1989, which is now known as ANSI C. It was then approved by the International Standards Organization (ISO) in 1990. This version of C is also referred as C99.

During 1990s, C++, a language entirely based on C, underwent a number of improvements and changes, and became an ANSI/ISO approved language in November 1977. C++ added several new features to C to make it not only a true object-oriented language but also a more versatile language. During the same period, Sun Microsystems of USA created a new language Java modelled on C and C++.

All popular computer languages are dynamic in nature. They continue to improve their power and scope by incorporating new features, and C is no exception. Although C++ and Java were evolved out of C, the standardization committee of C felt that a few features of C++/Java, if added to C, would enhance the usefulness of the language. The result was the 1999 standard for C. This version is usually referred to as C99. The history and development of C is illustrated in Fig. 4.2.

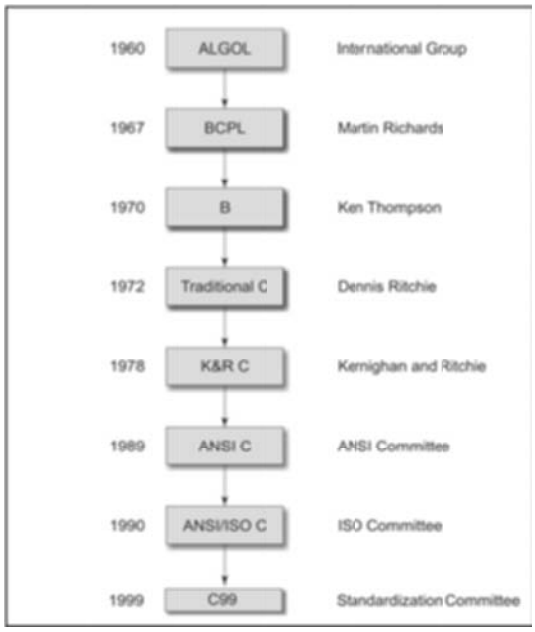


Fig. 4.2 History of ANSI C

Although C99 is an improved version, still many commonly available compilers do not support all the new features incorporated in C99.

4.2.2 Characteristics of C

The increasing popularity of C is probably due to its many desirable qualities. Some of the important characteristics are:

- It is a highly structured language.
- It uses features of high-level languages.
- It can handle bit-level operations.
- C is a machine-independent language and, therefore, highly portable.
- It supports a variety of data types and a powerful set of operators.
- It supports dynamic memory management by using the concept of pointers.
- It enables the implementation of hierarchical and modular programming with the help of functions.
- C can extend itself by addition of functions to its library continuously.

Before discussing specific features of C, we shall look at some sample C programs, and analyze and understand how they work

4.2.3 Sample Program 1: Printing a Message

Consider a very simple program given in Fig. 4.3.

```
main( )
{
/*.....printing begins.....*/
     printf("I see, I remember");
/*.....printing ends.....*/
}
```

Fig. 4.3 A program to print one line of text

This program when executed will produce the following output:

```
I see, I remember
```

Let us have a close look at the program. The first line informs the system that the name of the program is main and the execution begins at this line. The **main( )** is a special function used by the C system to tell the computer where the program starts. Every program must have exactly one main function. If we use more than one main function, the compiler cannot understand which one marks the beginning of the program.

The empty pair of parentheses immediately following main indicates that the function main has no arguments (or parameters). The concept of arguments will be discussed in chapter 5.

The opening brace "{ " in the second line marks the beginning of the function main and the closing brace "}" in the last line indicates the end of the function. In this case, the closing brace also marks the end of the program. All the statements between these two braces form the function body. The function body contains a set of instructions to perform the given task.

In this case, the function body contains three statements out of which only the printf line is an executable statement. The lines beginning with **/*** and ending with ***/** are known as comment lines. These are used in a program to enhance its readability and understanding. Comment lines are not executable statements and therefore anything between /* and */ is ignored by the compiler. In general, a comment can be inserted wherever blank spaces can occur—at the beginning, middle or end of a line—"but never in the middle of a word ".

Although comments can appear anywhere, they cannot be nested in C. That means, we cannot have comments inside comments. Once the compiler finds an opening token, it ignores everything until it finds a closing token. Thus the following comment line is not valid and will result in an error.

```
/* = = = =/* = = = = */ = = = = */
```

Since comments do not affect the execution speed and the size of a compiled program, we should use them liberally in our programs. They help the programmers and other users in understanding the various functions and operations of a program and serve as an aid to debugging and testing. We shall see the use of comment lines more in the examples that follow.

Let us now look at the printf( ) function, the only executable statement of the program, as shown under:

```
printf("I see, I remember");
```

printf is a predefined standard C function for printing output. Predefined means that it is a function that has already been written and compiled, and linked together with our program at the time of linking. The printf function causes everything between the starting and the ending quotation marks to be printed out. In this case, the output will be:

```
I see, I remember
```

Note that the print line ends with a semicolon. Every statement in C should end with a semicolon (;) mark. Suppose we want to print the above quotation in two lines as

```
I see,
I remember!
```

This can be achieved by adding another printf function as shown below:

```
printf("I see, \n");
printf("I remember !");
```

The information contained between the parentheses is called the argument of the function. This argument of the first printf function is " I see, \n" and the second is "I remember !". These arguments are simply strings of characters to be printed out.

Notice that the argument of the first printf contains a combination of two characters **** and **n** at the end of the string. This combination is collectively called the newline character. A newline character instructs the computer to go to the next (new) line. It is similar in concept to the carriage return key on a typewriter. After printing the character comma (,) the presence of the newline character **\n** causes the string "I remember !" to be printed on the next line. No space is allowed between \ and n.

If we omit the newline character from the first printf statement, then the output will again be a single line as shown below:

```
I see, I remember !
```

This is similar to the output of the program in Fig. 4.3.

It is also possible to produce two or more lines of output by one printf statement with the use of newline character at appropriate places. For example, the statement printf("I see,\n I remember !"); will generate the following output:

```
I see,
I remember !
```

However, the statement printf("I\n.. see,\n… … … I\n… … … remember !"); will print out the following:

```
I
.. see,
… … … I
… … … remember !
```

NOTE: Some authors recommend the inclusion of the statement #include <stdio.h> at the beginning of all programs that use any input/output library functions. However, this is not necessary for the functions printf and scanf which have been defined as a part of the C language.

Before we proceed to discuss further examples, we must note one important point. C does make a distinction between uppercase and lowercase letters. For example, printf and PRINTF are not the same. In C, everything is written in lowercase letters. However, uppercase letters are used for symbolic names representing constants. We may also use uppercase letters in output strings like "I SEE" and "I REMEMBER"

The above example that printed I see, I remember is one of the simplest programs. Figure 4.4 highlights the general format of such simple programs. All C programs need a main function.

Fig. 4.4 Format of simple C programs

The main Function

The main is a part of every C program. C permits different forms of main statement. Following forms are allowed.

- main()
- int main()
- void main()
- main(void)
- void main(void)
- int main(void)

The empty pair of parentheses indicates that the function has no arguments. This may be explicitly indicated by using the keyword void inside the parentheses. We may also specify the keyword int or void before the word main. The keyword void means that the function does not return any information to the operating system and int means that the function returns an integer value to the operating system. When int is specified, the last statement in the program must be "return 0". For the sake of simplicity, we use the first form in our programs.

4.2.4 Sample Program 2: Adding Two Numbers

Consider another program, which performs addition on two numbers and displays the result. The complete program is shown in Fig. 4.5:

```
/* Program ADDITION                line-1 */
/* Written by EBG                  line-2 */
main()                           /* line-3 */
{                               /*  line-4 */
    int number;                 /*  line-5 */
    float amount;               /*  line-6 */
                                /*  line-7 */
    number = 100;               /*  line-8 */
                                /*  line-9 */
    amount = 30.75 + 75.35;     /*  line-10 */
    printf("%d\n",number);      /*  line-11 */
    printf("%5.2f",amount);     /*  line-12 */
}                               /*  line-13 */
```

Fig. 4.5 Program to add two numbers

This program when executed will produce the following output:

```
100
106.10
```

The first two lines of the program are comment lines. It is a good practice to use comment lines in the beginning to give information such as name of the program, author, date, etc. Comment characters are also used in other lines to indicate line numbers.

The words **number** and **amount** are variable names that are used to store numeric data. The numeric data may be either in integer form or in real form. In C, all variables should be declared to tell the compiler what the variable names are and what type of data they hold.

The variables must be declared before they are used. In lines 5 and 6, the declarations **int number**; and **float amount**; tell the compiler that **number** is an integer (int) and **amount** is a floating (float) point number. Declaration statements must appear at the beginning of the functions, as shown in Fig. 4.5. All declaration statements end with a semicolon.

The words such as **int** and **float** are called the keywords and cannot be used as variable names. Data is stored in a variable by assigning a data value to it. This is done in lines 8 and 10. In line 8, an integer value 100 is assigned to the integer variable **number** and in line 10, the result of addition of two real numbers 30.75 and 75.35 is assigned to the floating point variable **amount**. Thus, the following statements are called the assignment statements:

```
number = 100;
amount = 30.75 + 75.35;
```

Every assignment statement must have a semicolon at the end.

The next statement is an output statement that prints the value of number, as shown below:

```
printf("%d\n", number);
```

The above print statement contains two arguments. The first argument "%d" tells the compiler that the value of the second argument number should be printed as a decimal integer. Note that these arguments are separated by a comma. The newline character \n causes the next output to appear on a new line.

The last statement of the program i.e. printf("%5.2f", amount); prints out the value of amount in floating point format. The format specification **%5.2f** tells the compiler that the output must be in floating point, with five places in all and two places to the right of the decimal point.

4.2.5 Sample Program 3: Interest Calculation

The program in Fig. 4.6 calculates the value of money at the end of each year of investment, assuming an interest rate of 11 percent and prints the year, and the corresponding amount, in two columns. The output is shown in Fig. 4.7 for a period of 10 years with an initial investment of 5000.00. The program uses the following formula:

```
Value at the end of year = Value at start of year (1 + interest rate)
```

In the program, the variable value represents the value of money at the end of the year while amount represents the value of money at the start of the year. The statement amount = value; makes the value at the end of the current year as the value at start of the next year.

```
/*———————— INVESTMENT PROBLEM ————————*/
#define PERIOD 10
#define PRINCIPAL   5000.00
/*———————— MAIN PROGRAM BEGINS ————————*/
main()
{ /*————————— DECLARATION STATEMENTS —————————*/
     int year;
     float amount, value, inrate;
/*————————— ASSIGNMENT STATEMENTS —————————*/
     amount = PRINCIPAL;
     inrate = 0.11;
     year = 0;
/*————————— COMPUTATION STATEMENTS —————————*/
/*————————— COMPUTATION USING While LOOP —————————*/
     while(year <= PERIOD)
```

```
        {
           printf("%2d   %8.2f\n",year, amount);
           value= amount + inrate * amount;
           year = year + 1;
           amount  = value;
        }
  /*———————— while LOOP ENDS ————————*/
  }
  /*———————————— PROGRAM ENDS ————————————*/
```

Fig. 4.6 **Program for investment problem**

```
        0     5000.00
        1     5550.00
        2     6160.50
        3     6838.15
        4     7590.35
        5     8425.29
        6     9352.07
        7     10380.00
        8     11522.69
        9     12790.00
        10    14197.11
```

Fig. 4.7 **Output of the investment program**

Let us consider the new features introduced in this program. The second and third lines begin with **#define** instructions. A #define instruction defines value to a symbolic constant for use in the program. Whenever a symbolic name is encountered, the compiler substitutes the value associated with the name automatically. To change the value, we have to simply change the definition. In this example, we have defined two symbolic constants PERIOD and PRINCIPAL and assigned values 10 and 5000.00 respectively. These values remain constant throughout the execution of the program.

The #define Directive

A #define is a preprocessor compiler directive and not a statement. Therefore, #define lines should not end with a semicolon. Symbolic constants are generally written in uppercase so that they are easily distinguished from lowercase variable names.

#define instructions are usually placed at the beginning before the main() function. Symbolic constants are not declared in declaration section.

We must note that the defined constants are not variables. We may not change their values within the program by using an assignment statement. For example, the following statement is illegal:

```
PRINCIPAL = 10000.00;
```

The declaration section declares **year** as integer and **amount**, **value** and **inrate** as floating point variables. Note all the floating point variables are declared in one statement. They can also be declared as below:

```
float amount;
float value;
float inrate;
```

When two or more variables are declared in one statement, they are separated by a comma.

All computations and printing are accomplished in a **while** loop. **while** is a mechanism for evaluating repeatedly a statement or a group of statements. In this case as long as the value of year is less than or equal to the value of PERIOD, the four statements that follow **while** are executed. Note that these four statements are grouped by braces. We exit the loop when year becomes greater than PERIOD.

C supports the basic four arithmetic operators (–, +, *, /) along with several others.

4.2.6 Sample Program 4: Use of Subroutines

So far, we have used only printf function that has been provided for us by the C system. The program shown in Fig. 4.F6 uses a user-defined function. A function defined by the user is equivalent to a subroutine in FORTRAN or subprogram in BASIC. Figure 4.8 presents a very simple program that uses a **mul ( )** function.

```
Program
/*————————— PROGRAM USING FUNCTION ————————*/
int mul (int a, int b); /*—— DECLARATION ———*/
/*———————— MAIN PROGRAM BEGINS ———————*/
        main ()
        {
             int a, b, c;
             a = 5;
             b = 10;
             c = mul (a,b);
```

```
        printf ("multiplication of %d and %d is %d",a,b,c);
        }
/* ————————MAIN PROGRAM ENDS
           MUL() FUNCTION STARTS ————————————*/
     int mul (int x, int y)
     int p;
     {
          p = x*y;
          return(p);
     }
/* ———————————— MUL () FUNCTION ENDS ————————————*/
```

Fig. 4.8 A program using a user-defined function

In the above program, the **mul ( )** function multiplies the values of **x** and **y** and the result is returned to the **main ( )** function when it is called in the following statement:

```
c = mul (a, b);
```

The mul ( ) has two arguments x and y that are declared as integers. The values of **a** and **b** are passed on to x and y respectively when the function mul ( ) is called. User-defined functions are considered in detail in Chapter 5

4.2.7 Sample Program 5: Use of Math functions

We often use standard mathematical functions such as **cos**, **sin**, **exp**, etc. We shall see now the use of a mathematical function in a program. The standard mathematical functions are defined and kept as a part of C math library. If we want to use any of these mathematical functions, we must add an **#include** instruction in the program. Like **#define**, it is also a compiler directive that instructs the compiler to link the specified mathematical functions from the library. The instruction takes the following form:

```
#include <math.h>
```

Here, **math.h** is the filename containing the required function. Figure 4.9 illustrates the use of cosine function. The program calculates cosine values for angles 0, 10, 20…….180 and prints out the results with headings.

```
Program
/*———————— PROGRAM USING COSINE FUNCTION ———————— */
#include <math.h>
#define  PI 3.1416
```

```
#define  MAX   180
main ( )
{
        int angle;
        float x,y;
        angle = 0;
        printf("   Angle   Cos(angle)\n\n");
        while(angle <= MAX)
        {
             x = (PI/MAX)*angle;
             y = cos(x);
             printf("%15d %13.4f\n", angle, y);
             angle = angle + 10;
        }
}
Output
Angle Cos(angle)
      0     1.0000
      10    0.9848
      20    0.9397
      30    0.8660
      40    0.7660
      50    0.6428
      60    0.5000
      70    0.3420
      80    0.1736
      90    –0.0000
      100     –0.1737
      110     –0.3420
      120     –0.5000
      130     –0.6428
      140     –0.7660
      150     –0.8660
      160     –0.9397
      170     –0.9848
      180     –1.0000
```

Fig. 4.9 Program using a math function

The #include Directive

As mentioned earlier, C programs are divided into modules or functions. Some functions are written by users, like us, and many others are stored in the C library. Library functions are grouped category-wise and stored in different files known as header files. If we want to access the functions stored in the library, it is necessary to tell the compiler about the files to be accessed.

This is achieved by using the preprocessor directive #include as follows:

```
#include <filename>
```

Here, filename is the name of the library file that contains the required function definition. Preprocessor directives are placed at the beginning of a program.

4.3 BASIC STRUCTURE OF C PROGRAMS

The examples discussed so far illustrate that a C program can be viewed as a group of building blocks called functions. A function is a subroutine that may include one or more statements designed to perform a specific task. To write a C program, we first create functions and then put them together. A C program may contain one or more sections as shown in Fig. 4.10.

The documentation section consists of a set of comment lines giving the name of the program, the author and other details, which the programmer would like to use later. The link section provides instructions to the compiler to link functions from the system library. The definition section defines all symbolic constants.

There are some variables that are used in more than one function. Such variables are called global variables and are declared in the global declaration section that is outside of all the functions. This section also declares all the user-defined functions.

Every C program must have one main() function section. This section contains two parts, declaration part and executable part. The declaration part declares all the variables used in the executable part. There is at least one statement in the executable part. These two parts must appear between the opening and the closing braces. The program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function section is the logical end of the program. All statements in the declaration and executable parts end with a semicolon(;).

The subprogram section contains all the user-defined functions that are called in the main

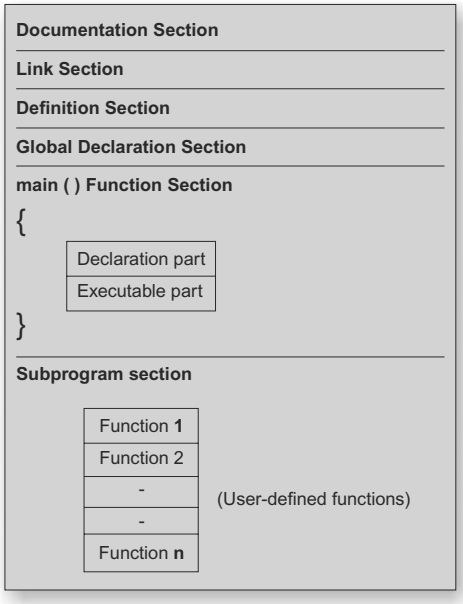


Fig. 4.10 **Typical structure of a C program**

function. User-defined functions are generally placed immediately after the main function, although they may appear in any order.

All sections, except the main function section may be absent when they are not required.

4.4 PROGRAMMING STYLE

Unlike some other programming languages (COBOL, FORTRAN, etc.,) C is a free-form language. That is, the C compiler does not care, where on the line we begin typing. While this may be a licence for bad programming, we should try to use this fact to our advantage in developing readable programs. Although several alternative styles are possible, we should select one style and use it with total consistency.

First of all, we must develop the habit of writing programs in lowercase letters. C program statements are written in lowercase letters. Uppercase letters are used only for symbolic constants.

Braces, group program statements together and mark the beginning and the end of functions. A proper indentation of braces and statements would make a program easier to read and debug.

Since C is a free-form language, we can group statements together on one line. For example, consider the following statements

```
a = b;
x = y + 1;
z = a + x;
```

The above statements can be written in one line as under:

```
a= b; x = y+1; z = a+x;
```

Similarly, consider the following program:

```
main( )
{
     printf("hello C");
}
```

It can also be written in one line as under:

```
main( ) {printf("Hello C")};
```

However, this style makes the program more difficult to understand and should not be used.

4.5 EXECUTING A 'C' PROGRAM

Executing a program written in C involves a series of steps. These are:

1. Creating the program;
2. Compiling the program;
3. Linking the program with functions that are needed from the C library; and
4. Executing the program.

Figure 4.11 illustrates the process of creating, compiling and executing a C program. Although these steps remain the same irrespective of the operating system, system commands for implementing the steps and conventions for naming files may differ on different systems.

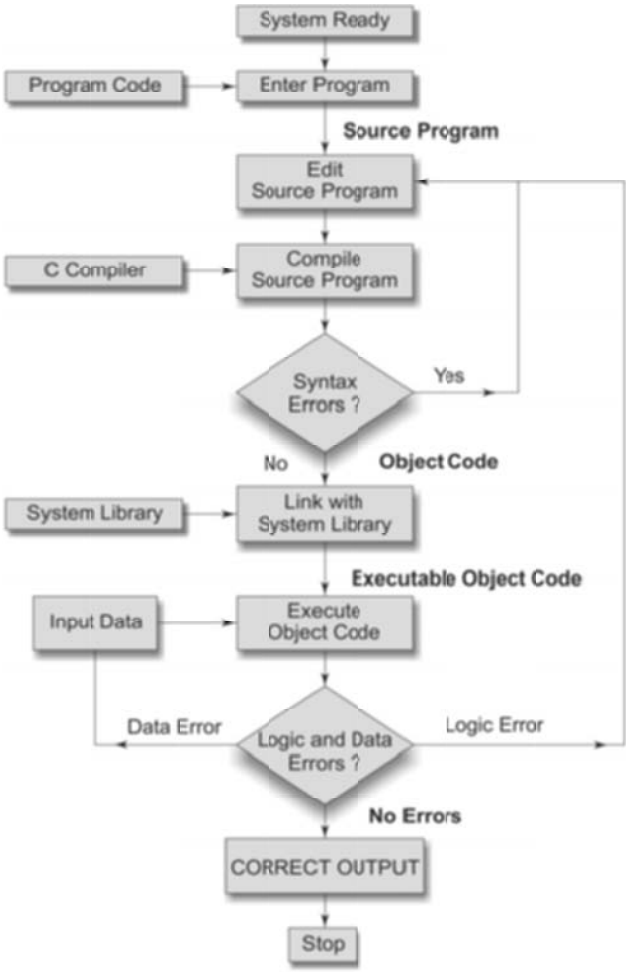


Fig. 4.11 Process of compiling and running a C program

An operating system is a program that controls the entire operation of a computer system. All input/output operations are channeled through the operating system. The operating system, which is an interface between the hardware and the user, handles the execution of user programs.

The two most popular operating systems today are UNIX (for minicomputers) and MS-DOS (for microcomputers). We shall discuss briefly the procedure to be followed in executing C programs under both these operating systems in the following sections.

4.6 UNIX SYSTEM

Creating the program Once we load the UNIX operating system into the memory, the computer is ready to receive program. The program must be entered into a file. The file name can consist of letters, digits and special characters, followed by a dot and a letter c. Examples of valid file names are:

```
hello.c
program.c
ebg1.c
```

The file is created with the help of a text editor, either ed or vi. The command for calling the editor and creating the file is

```
ed filename
```

If the file existed before, it is loaded. If it does not yet exist, the file has to be created so that it is ready to receive the new program. Any corrections in the program are done under the editor.

When the editing is over, the file is saved on disk. It can then be referenced any time later by its file name. The program that is entered into the file is known as the source program, since it represents the original form of the program.

Compiling and Linking Let us assume that the source program has been created in a file named ebg1.c. Now the program is ready for compilation. The compilation command to achieve this task under UNIX is

```
cc ebg1.c
```

The source program instructions are now translated into a form that is suitable for execution by the computer. The translation is done after examining each instruction for its correctness. If everything is alright, the compilation proceeds silently and the translated program is stored on another file with the name ebg1.o. This program is known as object code.

Linking is the process of putting together other program files and functions that are required by the program. For example, if the program is using exp() function, then the object

code of this function should be brought from the math library of the system and linked to the main program. Under UNIX, the linking is automatically done (if no errors are detected) when the cc command is used.

If any mistakes in the syntax and semantics of the language are discovered, they are listed out and the compilation process ends right there. The errors should be corrected in the source program with the help of the editor and the compilation is done again.

The compiled and linked program is called the executable object code and is stored automatically in another file named a.out.

Executing the Program Execution is a simple task. The command would load the executable object code into the computer memory and execute the instructions:

```
a.out
```

During execution, the program may request for some data to be entered through the keyboard. Sometimes the program does not produce the desired results. Perhaps, something is wrong with the program logic or data. Then it would be necessary to correct the source program or the data. In case the source program is modified, the entire process of compiling, linking and executing the program should be repeated.

Creating Our Own Executable File Note that the linker always assigns the same name a.out. When we compile another program, this file will be overwritten by the executable object code of the new program. If we want to prevent this from happening, we should rename the file immediately by using the following command:

```
mv a.out name
```

We may also achieve this by specifying an option in the cc command as follows:

```
cc –o name source-file
```

This will store the executable object code in the file name and prevent the old file a.out from being destroyed.

Multiple Source Files To compile and link multiple source program files, we must append all the files names to the cc command, as shown below:

```
cc filename-1.c …. filename-n.c
```

These files will be separately compiled into object files called filename-i.o and then linked to produce an executable program file a.out as shown in Fig. 4.F10.

It is also possible to compile each file separately and link them later. For example, the following commands will compile the source files mod1.c and mod2.c into objects files mod1.o and mod2.o:

```
cc –c mod1.c
cc –c mod2.c
```

They can be linked together by the following command:

```
cc mod1.o mod2.o
```

We may also combine the source files and object files as follows:

```
cc mod1.c mod2.o
```

Only mod1.c is compiled and then linked with the object file mod2.o. This approach is useful when one of the multiple source files need to be changed and recompiled or an already existing object files is to be used along with the program to be compiled.

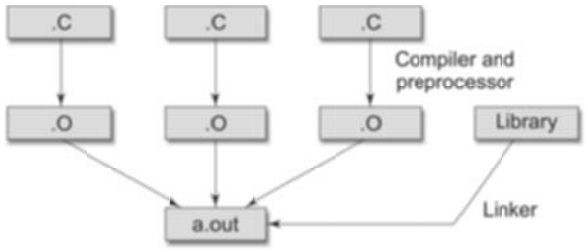


Fig. 4.12 Compilation of multiple files

4.7 C CHARACTER SET

A programming language is designed to help process certain kinds of data consisting of numbers, characters and strings and to provide useful output known as information. The task of processing of data is accomplished by executing a sequence of precise instructions called a program. These instructions are formed using certain symbols and words according to some rigid rules known as syntax rules (or grammar). Every program instruction must confirm precisely to the syntax rules of the language.

In C, the characters that can be used to form words, numbers and expressions depend upon the computer on which the program is run. However, a subset of characters is available that can be used on most personal, micro, mini and mainframe computers. The characters in C are grouped into the following categories:

1. Letters
2. Digits
3. Special characters
4. White spaces

The entire character set is given in Table 4.1.

TABLE 4.1 C Character Set

Characters	Notations
Letters	• Uppercase A.....Z • Lowercase a.....z
Digits	• All decimal digits 0 9
Special characters	• , comma • & ampersand • . period • ^ caret • ; semicolon • * asterisk • : colon • – minus sign • ? question mark • + plus sign • ‘ apostrophe • < opening angle bracket (or less than sign) • “ quotation mark • # number sign • ! exclamation mark • > closing angle bracket (or greater than sign) • | vertical bar • / slash • (left parenthesis • \ backslash •) right parenthesis • ~ tilde • [left bracket • _ under score •] right bracket • $ dollar sign

White Spaces	• { left brace • % percent sign • } right brace • Blank space • Horizontal tab • Carriage return • New line • Form feed

The compiler ignores white spaces unless they are a part of a string constant. White spaces may be used to separate words, but are prohibited between the characters of keywords and identifiers.

4.7.1 Trigraph Characters

Many non-English keyboards do not support all the characters mentioned in Table 4.1. ANSI C introduces the concept of 'trigraph' sequences to provide a way to enter certain characters that are not available on some keyboards. Each trigraph sequence consists of three characters (two question marks followed by another character) as shown in Table 4.2. For example, if a keyboard does not support square brackets, we can still use them in a program using the trigraph sequence ??(or ??).

TABLE 4.2 ANSI C Trigraph Sequences

Trigraph sequence	Translation
??=	# number sign
??(	[left bracket
??)	] right bracket
??<	{ left brace
??>	} right brace
??!	| vertical bar
??/	\ back slash
??-	~ tilde

4.8 C TOKENS

In a passage of text, individual words and punctuation marks are called tokens. Similarly, in a C program the smallest individual units are known as C tokens. C has six types of tokens as shown in Fig. 4.13. C programs are written using these tokens and the syntax of the language.

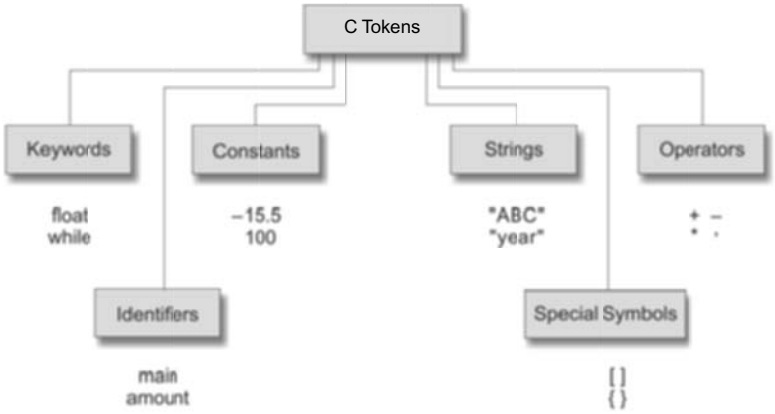


Fig. 4.13 **C tokens and examples**

4.9 KEYWORDS AND IDENTIFIERS

Every C word is classified as either a keyword or an identifier. All keywords have fixed meanings and these meanings cannot be changed. Keywords serve as the basic building blocks for program statements. The list of all keywords of ANSI C is shown in Table 4.3. All keywords must be written in lowercase. Some compilers may use additional keywords that must be identified from the C manual.

TABLE 4.3 ANSI C Keywords

Auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Identifiers refer to the names of variables, functions and arrays. These are user-defined names and consist a sequence of letters and digits, with a letter as the first character. Both uppercase and lowercase letters are permitted, although lowercase letters are commonly used. The underscore character is also permitted in identifiers. It is usually used as a link between two words in long identifiers. Rules for identifiers are as follows:

- The first character must be an alphabet (or underscore).
- It must consist only letters, digits or underscore.

- Only first 31 characters are significant in an identifier.
- It cannot be same as a keyword.
- It must not contain white space.

4.10 CONSTANTS

Constants in C refer to fixed values that do not change during the execution of a program. C supports several constants as illustrated in Fig. 4.14.

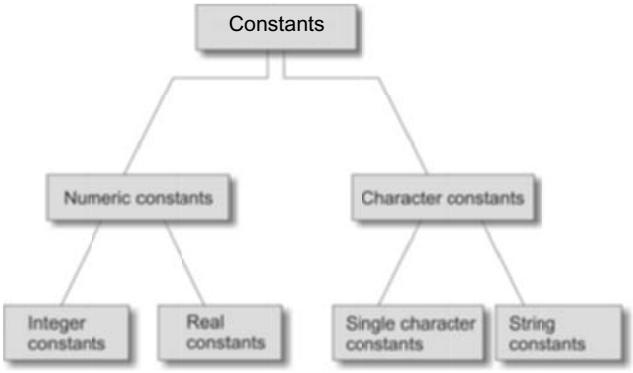


Fig. 4.14 **Basic types of C constants**

4.10.1 Integer Constants

An integer constant refers to a sequence of digits. There are three types of integers, namely, decimal integer, octal integer and hexadecimal integer. Decimal integers consist a set of digits, 0 through 9, preceded by an optional – or + sign. Examples of some valid decimal integer constants are:

123 – 321 0 654321 +78

Embedded spaces, commas and nondigit characters are not permitted between digits. Examples of some invalid decimal integer constants are:

15 750 20,000 $1000

An octal integer constant consists any combination of digits from the set 0 through 7, with a leading 0. Some examples of octal integer are:

037 0 0435 0551

A sequence of digits preceded by 0x or 0X is considered as a hexadecimal integer. They may also include alphabets A through F or a through f. The letters A through F represent the numbers 10 through 15. Following are the examples of valid hexa decimal integers:

0X2 0x9F 0Xbcd 0x

The largest integer value that can be stored is machine-dependent. It is 32767 on 16-bit machines and 2,147,483,647 on 32-bit machines. It is also possible to store larger integer constants on these machines by appending qualifiers such as U, L and UL to the constants. For example:

```
56789U          or 56789u       (unsigned integer)
987612347UL     or 98761234ul   (unsigned long integer)
9876543L        or 9876543l     (long integer)
```

EXAMPLE 4.1 *Representation of integer constants on a 16-bit computer.*

The program in Fig. 4.15 illustrates the use of integer constants on a 16-bit machine. The output shows that the integer values larger than 32767 are not properly stored on a 16-bit machine. However, when they are qualified as long integer (by appending L), the values are correctly stored.

Program

```
main()
{
   printf("Integer values\n\n");
   printf("%d %d %d\n", 32767,32767+1,32767+10);
   printf("\n");
   printf("Long integer values\n\n");
   printf("%ld %ld %ld\n",32767L,32767L+1L,32767L+10L);
}
```

Output

```
Integer values
32767 -32768 -32759
Long integer values
32767 32768 32777
```

Fig. 4.15 Representation of integer constants on a 16-bit machine

4.10.2 Real Constants

Integer numbers are inadequate to represent quantities that vary continuously, such as distances, heights, temperatures, prices and so on. These quantities are represented by numbers containing fractional parts like 17.548. Such numbers are called real (or floating point) constants. Some examples of real constants are:

0.0083 –0.75 435.36 +247.0

These numbers are shown in decimal notation, having a whole number followed by a decimal point and the fractional part. It is possible to omit digits before the decimal point, or digits after the decimal point. For example, the following are all valid real numbers.

215. .95 –.71 +.5

A real number may also be expressed in exponential (or scientific) notation. For example, the value 215.65 may be written as 2.1565e2 in exponential notation. e2 means multiply by 10^2. The general form is:

mantissa e exponent

The mantissa is either a real number expressed in decimal notation or an integer. The exponent is an integer number with an optional plus or minus sign. The letter e separating the mantissa and the exponent can be written in either lowercase or uppercase. Since the exponent causes the decimal point to 'float', this notation is said to represent a real number in floating point form. Examples of legal floating-point constants are:

0.65e4 12e-2 1.5e+5 3.18E3 –1.2E-1

Exponential notation is useful for representing numbers that are either very large or very small in magnitude. For example, 7500000000 may be written as 7.5E9 or 75E8. Similarly, -0.000000368 is equivalent to –3.68E-7.

Floating-point constants are normally represented as double-precision quantities. However, the suffix f or F may be used to force single-precision and l or L to extend double precision further. Some examples of valid and invalid numeric constants are given in Table 4.4.

TABLE 4.4 Examples of numeric constants

Constant	Valid ?	Remarks
698354L	Yes	Represents long integer
25,000	No	Comma is not allowed
+5.0E3	Yes	(ANSI C supports unary plus)
3.5e-5	Yes	
7.1e 4	No	No white space is permitted
-4.5e-2	Yes	
1.5E+2.5	No	Exponent must be an integer
$255	No	$ symbol is not permitted
0X7B	Yes	Hexadecimal integer

4.10.3 Single Character Constants

A single character constant (or simply character constant) contains a single character enclosed within a pair of single quote marks. Example of character constants are:

'5' 'X' ';' ' '

Note that the character constant '5' is not the same as the number 5. The last constant is a blank space. Character constants have integer values known as ASCII values. For example, the following statement would print the number 97, the ASCII value of the letter a.

```
printf("%d", 'a');
```

Similarly, the following statement would output the letter 'a'.

```
printf("%c", '97');
```

NOTE: Since each character constant represents an integer value, it is also possible to perform arithmetic operations on character constants.

Backslash Character Constants C supports some special backslash character constants that are used in output functions. For example, the symbol '\n' stands for newline character. A list of such backslash character constants is given in Table 4.5. Note that each one of them represents one character, although they consist two characters. These character combinations are known as escape sequences.

TABLE 4.5 Backslash character constants

Constant	Meaning
'\a'	audible alert (bell)
'\b'	back space
'\f'	form feed
'\n'	new line
'\r'	carriage return
'\t'	horizontal tab
'\v'	vertical tab
'\''	single quote
'\"'	double quote
'\?'	question mark
'\\'	backslash
'\0'	null

4.10.4 String Constants

A string constant is a sequence of characters enclosed in double quotes. The characters may be letters, numbers, special characters and blank space. Some of the examples of string constants are:

"Hello!" "1987" "WELL DONE" "?...!" "5+3" "X"

It is important to note that a character constant (e.g., 'X') is not equivalent to the single character string constant (e.g., "X"). Further, a single character string constant does not have an equivalent integer value, while a character constant has an integer value. Character strings are often used in programs to make it meaningful.

4.11 VARIABLES

A variable is a data name that may be used to store a data value. Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different times during execution. A variable name can be chosen by the programmer in a meaningful way so as to reflect its function or nature in the program. Some examples of such names are:

```
Average
height
Total
Counter_1
class_strength
```

As mentioned earlier, variable names may consist letters, digits and the underscore (_) character, subject to the following conditions:

1. They must begin with a letter. Some systems permit underscore as the first character.
2. ANSI standard recognises a length of 31 characters. However, length should not be normally more than eight characters, since only the first eight characters are treated as significant by many compilers.
3. Uppercase and lowercase are significant. That is, the variable Total is not the same as total or TOTAL.
4. It should not be a keyword.
5. White space is not allowed.

Some examples of valid variable names are:

```
John      Value     T_raise
Delhi     x1        ph_value
mark      sum1      distance
```

Invalid examples include:

123 (area)
% 25^{th}

Further examples of variable names and their correctness are given in Table 4.6.

TABLE 4.6 Examples of variable names

Variable name	Valid ?	Remark
First_tag	Valid	
char	Not valid	char is a keyword
Price$	Not valid	Dollar sign is illegal
group one	Not valid	Blank space is not permitted
average_number	Valid	First eight characters are significant
int_type	Valid	Keyword may be part of a name

4.12 DATA TYPES

C language is rich in its data types. Storage representations and machine instructions to handle constants differ from machine to machine. The variety of data types available allow the programmer to select the type appropriate to the needs of the application as well as the machine.

ANSI C supports three classes of data types:

1. Primary (or fundamental) data types
2. Derived data types
3. User-defined data types

All C compilers support five fundamental data types, namely integer (int), character (char), floating point (float), double-precision floating point (double) and void. Many of them also offer extended data types such as long int and long double. Various data types and the terminology used to describe them are given in Fig. 4.16. The range of the basic four types are given in Table 4.7.

Fig. 4.16 **Primary data types in C**

TABLE 4.7 Size and range of basic data types on 16-bit machines

Data type	Range of values
char	–128 to 127
int	–32,768 to 32,767
float	3.4e–38 to 3.4e+38
double	1.7e–308 to 1.7e+308

4.12.1 Integer Types

Integers are whole numbers with a range of values supported by a particular machine. Generally, integers occupy one word of storage, and since the word sizes of machines vary (typically, 16 or 32 bits) the size of an integer that can be stored depends on the computer. If we use a 16-bit word length, the size of the integer value is limited to the range –32768 to +32767 (that is, –215 to +215–1). A signed integer uses one bit for sign and 15 bits for the magnitude of the number. Similarly, a 32-bit word length can store an integer ranging from–2,147,483,648 to 2,147,483,647.

In order to provide some control over the range of numbers and storage space, C has three classes of integer storage, namely short int, int and long int, in both signed and unsigned forms. ANSI C defines these types so that they can be organised from the smallest to the largest, as shown in Fig. 4.17. For example, short int represents fairly small integer values and requires half the amount of storage as a regular int number uses. Unlike signed integers, unsigned integers use all the bits for the magnitude of the number and are always positive. Therefore, for a 16-bit machine, the range of unsigned integer numbers will be from 0 to 65,535.

Fig. 4.17 Integer types

We declare long and unsigned integers to increase the range of values. The use of qualifier signed on integers is optional because the default declaration assumes a signed number. Table 4.8 shows all the allowed combinations of basic types and qualifiers and their size and range on a 16-bit machine.

4.12.2 Floating Point Types

Floating point (or real) numbers are stored in 32 bits (on all 16 bit and 32 bit machines), with 6 digits of precision. Floating point numbers are defined in C by the keyword float. When the accuracy provided by a float number is not sufficient, the type double can be used to define the number.

TABLE 4.8 Size and range of data types on a 16-bit machine

Type	Size (bits)	Range
char or signed char	8	–128 to 127
unsigned char	8	0 to 255
int or signed int	16	–32,768 to 32,767
unsigned int	16	0 to 65535
short int or signed short int	8	–128 to 127
unsigned short int	8	0 to 255
long int or signed long int	32	–2,147,483,648 to 2,147,483,647
unsigned long int	32	0 to 4,294,967,295
Float	32	3.4E – 38 to 3.4E + 38
double	64	1.7E – 308 to 1.7E + 308
long double	80	3.4E – 4932 to 1.1E + 4932

A double data type number uses 64 bits giving a precision of 14 digits. These are known as double precision numbers. Remember that double type represents the same data type that float represents, but with a greater precision. To extend the precision further, we may use long double which uses 80 bits. The relationship among floating types is illustrated in Fig. 4.18.

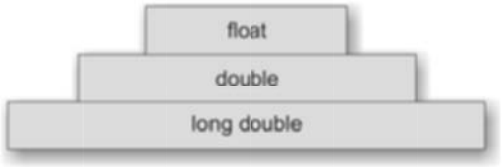


Fig. 4.18 **Floating-point types**

4.12.3 Void Types

The void type has no values. This is usually used to specify the type of functions. The type of a function is said to be void when it does not return any value to the calling function. It can also play the role of a generic type, meaning that it can represent any of the other standard types.

4.12.4 Character Types

A single character can be defined as a character (char) type data. Characters are usually stored in 8 bits (one byte) of internal storage. The qualifier signed or unsigned may be explicitly applied to char. While unsigned chars have values between 0 and 255, signed chars have values from –128 to 127.

4.13 DECLARATION OF VARIABLES

After designing suitable variable names, we must declare them to the compiler. Variable declaration basically does two things:

1. It tells the compiler what the variable name is.
2. It specifies what type of data the variable will hold.

The declaration of variables must be done before they are used in the program.

4.13.1 Primary Type Declaration

A variable can be used to store a value of any data type. That is, the name has nothing to do with its type. The syntax for declaring a variable is as follows:

```
data-type v1,v2,....vn ;
```

`v1, v2, ....vn` are the names of variables. Variables are separated by commas. A declaration statement must end with a semicolon. For example, the following represent valid variable declarations:

```
int count;
int number, total;
double ratio;
```

`int` and **`double`** are the keywords to represent integer type and real type data values, respectively. Table 4.9 shows various data types and their keyword equivalents.

TABLE 4.9 Data types and their keywords

Data type	Keyword equivalent
Character	char
Unsigned character	unsigned char
Signed character	signed char
Signed integer	signed int (or int)
Signed short integer	signed short int (or short int or short)
Signed long integer	signed long int (or long int or long)
Unsigned integer	unsigned int (or unsigned)
Unsigned short integer	unsigned short int (or unsigned short)
Unsigned long integer	unsigned long int (or unsigned long)
Floating point	float
Double-precision floating point	double
Extended double-precision floating point	long double

The program given in Fig. 4.19 illustrates declaration of variables.

main() is the beginning of the program. The opening brace { signals the execution of the program. Declaration of variables is usually done immediately after the opening brace of the program. The variables can also be declared outside (either before or after) the main function.

```
main() /*.........Program Name....................... */
{
     /*................Declaration.....................*/
        float    x, y;
        int        code;
        short int    count;
        long int     amount;
        double     deviation;
        unsigned     n;
        char       c;
     /*................Computation..................... */
        . . . .
        . . . .
        . . . .
}    /*.............Program ends.....................*/
```

Fig. 4.19 **Declaration of variables**

When an adjective (qualifier), short, long, or unsigned, is used without a basic data type specifier, C compilers treat the data type as an int. If we want to declare a character variable as unsigned, then we must do so using both the terms like unsigned char.

4.13.2 User-defined Type Declaration

C supports a feature known as 'type definition' that allows users to define an identifier that would represent an existing data type. The user-defined data type identifier can later be used to declare variables. It takes the general form as follows:

```
typedef type identifier;
```

where type refers to an existing data type and 'identifier' refers to the 'new' name given to the data type. The existing data type may belong to any class of type, including the user-defined ones. Remember that the new type is 'new' only in name, but not the data type. typedef cannot create a new type. Some examples of type definition are as follows:

```
typedef int units;
typedef float marks;
```

Here, units symbolises int and marks symbolises float. They can be later used to declare variables as follows:

```
units batch1, batch2;
marks name1[50], name2[50];
```

Here `batch1` and `batch2` are declared as **int** variable and `name1[50]` and `name2[50]` are declared as 50 element floating point array variables. The main advantage of typedef is that we can create meaningful data type names for increasing the readability of the program.

Another user-defined data type is enumerated data type provided by ANSI standard. It is defined as follows:

```
enum identifier {value1, value2, ... value n};
```

where 'identifier' is a user-defined enumerated data type, which can be used to declare variables that can have one of the values enclosed within the braces (known as enumeration constants). After this definition, we can declare variables to be of this 'new' type as follows:

```
enum identifier v1, v2, ... vn;
```

The enumerated variables v1, v2, ... vn can only have one of the values value1, value2, ... valuen. The assignments of the following types are valid:

```
v1 = value3;
v5 = value1;
```

Following is an example of using enum data type:

enum day {Monday,Tuesday, ... Sunday};

enum day week_st, week_end;

```
week_st = Monday;
week_end = Friday;
if(week_st == Tuesday)
week_end = Saturday;
```

4.14 DECLARATION OF STORAGE CLASS

Variables in C can have not only data type but also storage class that provides information about their location and visibility. The storage class decides the portion of the program within which the variables are recognised. Consider the following example:

```
  /* Example of storage classes */
int  m;
main()
{
  int i;
  float balance;
  ....
  ....
  function1();
}
function1()
{
  int i;
  float sum;
  ....
  ....
}
```

The variable *m* which has been declared before the main is called global variable. It can be used in all the functions in the program. It need not be declared in other functions. A global variable is also known as an external variable.

The variables *i, balance* and *sum* are called local variables because they are declared inside a function. Local variables are visible and meaningful only inside the functions in which they are declared. They are not known to other functions. Note that the variable *i* has been declared in both the functions. Any change in the value of *i* in one function does not affect its value in the other.

C provides a variety of storage class specifiers that can be used to declare explicitly the scope and lifetime of variables. Table 4.10 provide a list of the various storage class specifiers along with their meanings:

TABLE 4.10 Storage classes and their meaning

Storage class	Meaning
auto	Local variable known only to the function in which it is declared. Default is auto.
static	Local variable which exists and retains its value even after the control is transferred to the calling function.
extern	Global variable known to all functions in the file.
register	Local variable which is stored in the register.

4.15 ASSIGNING VALUES TO VARIABLES

Variables are created for use in program statements such as,

```
value = amount + inrate * amount;
while (year <= PERIOD)
{
   ....
   ....
   year = year + 1;
}
```

In the first statement, the numeric value stored in the variable *inrate* is multiplied by the value stored in amount, and the product is added to amount. The result is stored in the variable value. This process is possible only if the variables amount and *inrate* have already been given values. The variable value is called the target variable.

While all the variables are declared for their type, the variables that are used in expressions (on the right side of equal (=) sign of a computational statement) must be assigned values before they are encountered in the program. Similarly, the variable year and the symbolic constant PERIOD in the while statement must be assigned values before this statement is encountered.

4.15.1 Assignment Statement

Values can be assigned to variables using the assignment operator = as follows:

variable_name = constant;

Examples are:

```
initial_value = 0;
final_value = 100;
balance = 75.84;
yes  = 'x';
```

C permits multiple assignments in one line. For example:

```
initial_value = 0; final_value = 100;
```

An assignment statement implies that the value of the variable on the left of the 'equal sign' is set equal to the value of the quantity (or the expression) on the right. The following statement means that the 'new value' of year is equal to the 'old value' of year plus 1.

```
year = year + 1;
```

During assignment operation, C converts the type of value on the right-hand side to the type on the left. This may involve truncation when real value is converted to an integer.

It is also possible to assign a value to a variable at the time the variable is declared. This takes the following form:

```
int final_value = 100;
char yes  ='x';
double balance = 75.84;
```

The process of giving initial values to variables is called initialisation. C permits the initialisation of more than one variables in one statement using multiple assignment operators. For example:

```
p =  q = s = 0;
x =  y = z = MAX;
```

Here, the first statement initialises the variables *p*, *q*, and *s* to zero while the second initialises *x*, *y*, and *z* with MAX. Note that MAX is a symbolic constant defined at the beginning.

NOTE: Remember that external and static variables are initialised to zero by default. Automatic variables that are not initialised explicitly contain garbage values.

EXAMPLE 4.2 *The program in Fig. 4.20 shows typical declarations, assignments and values stored in various types of variables.*

Program

```
main()
{
/*.........DECLARATIONS............................*/
     float    x, p ;
     double    y, q ;
     unsigned   k ;
/*.........DECLARATIONS AND ASSIGNMENTS............*/
     int        m = 54321 ;
     long int   n = 1234567890 ;
/*.........ASSIGNMENTS............................*/
     x = 1.234567890000 ;
```

```
            y = 9.87654321 ;
            k = 54321 ;
            p = q = 1.0 ;
      /*.........PRINTING..............................*/
            printf("m = %d\n", m) ;
            printf("n = %ld\n", n) ;
            printf("x = %.12lf\n", x) ;
            printf("x = %f\n", x) ;
            printf("y = %.12lf\n",y) ;
            printf("y = %lf\n", y) ;
            printf("k = %u p = %f q = %.12lf\n", k, p, q) ;
      }
Output
            m = -11215
            n = 1234567890
            x = 1.234567880630
            x = 1.234568
            y = 9.876543210000
            y = 9.876543
            k = 54321 p = 1.000000 q = 1.000000000000
```

Fig. 4.20 Examples of assignments

In the program given in Fig. 4.20, the variables *x* and *p* have been declared as floating-point variables. Note that the way the value of 1.234567890000 that we assigned to *x* is displayed under different output formats. The value of *x* is displayed as 1.234567880630 under %.12lf format, while the actual value assigned is 1.234567890000. This is because the variable *x* has been declared as a float that can store values only up to six decimal places.

The variable *m* that has been declared as int is not able to store the value 54321 correctly. Instead, it contains some garbage. Since this program was run on a 16-bit machine, the maximum value that an *int* variable can store is only 32767. However, the variable *k* (declared as unsigned) has stored the value 54321 correctly. Similarly, the long *int* variable *n* has stored the value 1234567890 correctly.

The value 9.87654321 assigned to *y* declared as double has been stored correctly but the value is printed as 9.876543 under %lf format. Note that unless specified otherwise, the printf function will always display a float or double value to six decimal places.

4.15.2 Reading Data from Keyboard

Another way of giving values to variables is to input data through keyboard using the scanf function. It is a general input function available in C and is very similar in concept to the printf function. It works much like an INPUT statement in BASIC. The general format of scanf is as follows:

```
scanf("control string", &variable1,&variable2,....);
```

The control string contains the format of data being received. The ampersand symbol & before each variable name is an operator that specifies the variable name's address. We must always use this operator, otherwise unexpected results may occur. Let us look at an example:

```
scanf("%d", &number);
```

When this statement is encountered by the computer, the execution stops and waits for the value of the variable number to be typed in. Since the control string "%d" specifies that an integer value is to be read from the terminal, we have to type in the value in integer form. Once the number is typed in and the 'Return' Key is pressed, the computer then proceeds to the next statement. Thus, the use of scanf provides an interactive feature and makes the program 'user friendly'. The value is assigned to the variable number.

EXAMPLE 4.3 *The program in Fig. 4.21 illustrates the use of scanf function.*

The first executable statement in the program is a printf, requesting the user to enter an integer number. This is known as 'prompt message' and appears on the screen like as:

```
Enter an integer number
```

As soon as the user types in an integer number, the computer proceeds to compare the value with 100. If the value typed in is less than 100, then the following message is printed on the screen.

```
Your number is smaller than 100
```

Otherwise, the following message is displayed

```
Your number contains more than two digits
```

Outputs of the program run for two different inputs are also shown in Fig. 4.21.

```
Program
     main()
     {
          int number;

          printf("Enter an integer number\n");
          scanf ("%d", &number);
          if ( number < 100 )
             printf("Your number is smaller than 100\n\n");
          else
             printf("Your number contains more than two digits\n");
     }
```

```
Output
      Enter an integer number
      54
      Your number is smaller than 100
      Enter an integer number
      108
      Your number contains more than two digits
```

Fig. 4.21 Use of scanf function for interactive computing.

NOTE: The above program uses a decision statement if...else to decide whether the number is less than 100. We will learn about decision-making statements later in this chapter.

EXAMPLE 4.4 *Figure 4.22 shows a program that uses the scanf function to receive values from the end user.*

```
Program
      main()
      {
            int year, period ;
            float amount, inrate, value ;

            printf("Input amount, interest rate, and period\n\n") ;
            scanf ("%f %f %d", &amount, &inrate, &period) ;
            printf("\n") ;
            year = 1 ;

            while( year <= period )
            {
                    value = amount + inrate * amount ;
                    printf("%2d Rs %8.2f\n", year, value) ;
                    amount = value ;
                    year = year + 1 ;
            }
      }
Output
Input amount, interest rate, and period

      10000 0.14 5

        1 Rs 11400.00
        2 Rs 12996.00
```

```
        3 Rs 14815.44
        4 Rs 16889.60
        5 Rs 19254.15
Input amount, interest rate, and period
     20000 0.12 7
        1 Rs 22400.00
        2 Rs 25088.00
        3 Rs 28098.56
        4 Rs 31470.39
        5 Rs 35246.84
        6 Rs 39476.46
        7 Rs 44213.63
```

Fig. 4.22 **Interactive investment program.**

In the above program, the computer requests the user to input the values of the amount to be invested, interest rate and period of investment by printing the following prompt message and then waits for input values.

```
Input amount, interest rate, and period
```

As soon as we finish entering the three values corresponding to the three variables amount, inrate and period, the computer begins to calculate the amount at the end of each year, up to 'period' and produces output as shown above.

4.15.3 Declaring a Variable as a Constant

We may like the value of certain variables to remain constant during the execution of a program. We can achieve this by declaring the variable with the qualifier const at the time of initialisation. Here is an example:

```
const int class_size = 40;
```

Here, const is a new data type qualifier defined by ANSI standard. This tells the compiler that the value of the *int* variable class_size must not be modified by the program. However, it can be used on the right_hand side of an assignment statement like any other variable.

4.15.4 Declaring a Variable as Volatile

ANSI standard defines another qualifier volatile that could be used to tell explicitly the compiler that a variable's value may be changed at any time by some external sources (from outside the program). Here is the example:

```
volatile int date;
```

The value of date may be altered by some external factors even if it does not appear on the left-hand side of an assignment statement. When we declare a variable as volatile, the compiler will examine the value of the variable each time it is encountered to see whether any external alteration has changed the value.

Remember that the value of a variable declared as volatile can be modified by its own program as well. If we wish that the value must not be modified by the program while it may be altered by some other process, then we may declare the variable as both const and volatile as shown further:

```
volatile const int location = 100;
```

4.16 CASE STUDIES

1. Calculation of Average of Numbers

A program to calculate the average of a set of N numbers is given in Fig. 4.23.

```
Program
#define N 10             /* SYMBOLIC CONSTANT */
main()
{
int count;            /* DECLARATION OF */
float  sum, average, number;   /* VARIABLES */
     sum = 0;              /* INITIALIZATION */
     count= 0;          /* OF VARIABLES */
printf("\nEnter 10 numbers");
     while( count < N )
        {
          scanf("%f", &number);
          sum = sum + number;
          count = count + 1;
        }
     average = sum/N;
     printf("N = %d Sum = %f", N, sum);
     printf(" Average = %f", average);
}
Output
     1
     2.3
     4.67
```

```
1.42
7
3.67
4.08
2.2
4.25
8.21
N = 10     Sum = 38.799999 Average = 3.880
```

Fig. 4.23 Average of N numbers

The variable number is declared as float and therefore it can take both integer and real numbers. Since the symbolic constant N is assigned the value of 10 using the #define statement, the program accepts ten values and calculates their sum using the while loop. The variable count counts the number of values and as soon as it becomes 11, the while loop is exited and then the average is calculated.

Notice that the actual value of sum is 38.8 but the value displayed is 38.799999. In fact, the actual value that is displayed is quite dependent on the computer system. Such an inaccuracy is due to the way the floating point numbers are internally represented inside the computer.

2. Temperature Conversion Problem

The program presented in Fig. 4.24 converts the given temperature in fahrenheit to celsius using the following conversion formula:

$$C = \frac{F - 32}{1.8}$$

```
Program
#define F_LOW  0  /* - - - - - - - - - - - - - - - - */
#define F_MAX  250   /* SYMBOLIC CONSTANTS */
#define STEP   25 /* - - - - - - - - - - - - - - -  */
main()
{
typedef float REAL;    /* TYPE DEFINITION */
      REAL fahrenheit, celsius;  /* DECLARATION */
      fahrenheit = F_LOW;      /* INITIALIZATION */
      printf("Fahrenheit Celsius\n\n");
      while( fahrenheit <= F_MAX )
        {
          celsius = ( fahrenheit - 32.0 ) / 1.8;
          printf("%5.1f %7.2f\n", fahrenheit, celsius);
          fahrenheit = fahrenheit + STEP;
```

```
        }
}
Output
     Fahrenheit      Celsius

     0.0             -17.78
     25.0            -3.89
     50.0            10.00
     75.0            23.89
     100.0           37.78
     125.0           51.67
     150.0           65.56
     175.0           79.44
     200.0           93.33
     225.0           107.22
     250.0           121.11
```

Fig. 4.24 Temperature conversion—fahrenheit-celsius

The program prints a conversion table for reading temperature in celsius, given the fahrenheit values. The minimum and maximum values and step size are defined as symbolic constants. These values can be changed by redefining the #define statements. A user-defined data type name REAL is used to declare the variables fahrenheit and celsius.

The formation specifications %5.1f and %7.2 in the second printf statement produces two-column output as shown.

4.17 MANAGING INPUT AND OUTPUT OPERATIONS

Reading, processing and writing of data are the three essential functions of a computer program. Most programs take some data as input and display the processed data, often known as information or results, on a suitable medium. So far we have seen two methods of providing data to the program variables. One method is to assign values to variables through the assignment statements such as $x = 5$; $a = 0$; and so on. Another method is to use the input function `scanf`, which can read data from a keyboard. We have used both the methods in most of our earlier example programs. For outputting results, we have used extensively the function `printf`, which sends results out to a terminal.

Unlike other high-level languages, C does not have any built-in input/output statements as part of its syntax. All input/output operations are carried out through function calls such as `printf` and `scanf`. There exist several functions that have more or less become standard for input and output operations in C. These functions are collectively known as the standard I/O library.

4.17.1 Reading a Character

The simplest of all input/output operations is reading a character from the 'standard input' unit (usually the keyboard) and writing it to the 'standard output' unit (usually the screen). Reading a single character can be done by using the function `getchar`. getchar takes the following form:

```
variable_name = getchar( );
```

Here, `variable_name` is a valid C name that has been declared as char type. When this statement is encountered, the computer waits until a key is pressed and then assigns this character as a value to getchar function. Since getchar is used on the right-hand side of an assignment statement, the character value of getchar is in turn assigned to the variable name on the left. For example, the following statements will assign the character 'H' to the variable name when we press the key H on the keyboard.

```
char name;
name = getchar();
```

EXAMPLE 4.5 *The program in Fig. 4.25 shows the use of getchar function in an interactive environment.*

```
Program
   #include <stdio.h>
   main()
   {
      char answer;
      printf("Would you like to know my name?\n");
      printf("Type Y for YES and N for NO: ");
      answer = getchar(); /*....Reading a character...*/
      if(answer == 'Y' || answer == 'y')
         printf("\n\nMy name is BUSY BEE\n");
      else
         printf("\n\nYou are good for nothing\n");
   }
Output
   Would you like to know my name?
   Type Y for YES and N for NO: Y
   My name is BUSY BEE

   Would you like to know my name?
   Type Y for YES and N for NO: n
   You are good for nothing
```

Fig. 4.25 **Use of getchar function to read a character from keyboard.**

The above program in Fig. 4.25 displays a question of YES/NO type to the user and reads the user's response in a single character (Y or N). If the response is Y or y, it outputs the following message:

```
My name is BUSY BEE
```

However, if the response is N or n, it outputs the following message:

```
You are good for nothing
```


NOTE: There is one line space between the input text and output message.

The getchar function may be called successively to read the characters contained in a line of text. For example, the following program segment reads characters from keyboard one after another until the 'Return' key is pressed.

```
————————
————————
char character;
character = ' ';
while(character != '\n')
{
     character = getchar();
}
————————
————————
```

EXAMPLE 4.6 *The program shown in Fig. 4.26 requests the user to enter a character and displays a message on the screen telling the user whether the character is an alphabet or digit, or any other special character.*

```
Program
     #include <stdio.h>
     #include <ctype.h>
     main()
     {
       char character;
       printf("Press any key\n");
       character = getchar();
```

```
        if (isalpha(character) > 0)/* Test for letter */
          printf("The character is a letter.");
        else
          if (isdigit (character) > 0)/* Test for digit */
            printf("The character is a digit.");
        else
          printf("The character is not alphanumeric.");
     }
Output
     Press any key
     h
     The character is a letter.
     Press any key
     5
     The character is a digit.
     Press any key
     *
     The character is not alphanumeric.
```

Fig. 4.26 **Program to test the character type.**

The above program receives in Fig. 4.26 a character from the keyboard and tests whether it is a letter or digit and prints out a message accordingly. These tests are done with the help of the following functions:

```
     isalpha(character)
     isdigit(character)
```

For example, isalpha assumes a value non-zero (TRUE) if the argument character contains an alphabet; otherwise it assumes 0 (FALSE). Similar is the case with the function isdigit.

C supports many other similar functions, which are given in Table 4.11. These character functions are contained in the file ctype.h and, therefore, the following pre-processor directive statement must be included in a program before using these functions:

```
#include <ctype.h>
```

4.17.2 Writing a Character

Like getchar, there is an analogous function putchar for writing characters one at a time to the terminal. It takes the form as shown further:

TABLE 4.11 Character test functions

Function	Test
isalnum(c)	Is c an alphanumeric character?
isalpha(c)	Is c an alphabetic character?
isdigit(c)	Is c a digit?
islower(c)	Is c lower case letter?
isprint(c)	Is c a printable character?
ispunct(c)	Is c a punctuation mark?
isspace(c)	Is c a white space character?
isupper(c)	Is c an upper case letter?

```
putchar (variable_name);
```

Here, variable_name is a type char variable containing a character. This statement displays the character contained in the variable_name at the terminal. For example, the following statements will display the character Y on the screen:

```
answer = 'Y';
putchar (answer);
```

EXAMPLE 4.7 *A program that reads a character from the keyboard and then prints it in reverse case is given in Fig. 4.27. That is, if the input is upper case, the output will be lower case and vice versa.*

```
Program
     #include <stdio.h>
     #include <ctype.h>
     main()
     {
        char alphabet;
        printf("Enter an alphabet");
        putchar('\n'); /* move to next line */
        alphabet = getchar();
        if (islower(alphabet))
        putchar(toupper(alphabet));/* Reverse and display */
     else
        putchar(tolower(alphabet)); /* Reverse and display */
     }
```

```
Output
      Enter an alphabet
      a
      A
      Enter an alphabet
      Q
      q
      Enter an alphabet
      z
      Z
```

Fig. 4.27 **Reading and writing of alphabets in reverse case.**

The above program in Fig. 4.27 uses three new functions: `islower`, `toupper`, and `tolower`. The function `islower` is a conditional function and takes the value TRUE if the argument is a lowercase alphabet; otherwise it takes the value FALSE. The function toupper converts the lowercase argument into an uppercase alphabet, while the function tolower does the reverse.

4.17.3 Formatted Input

Formatted input refers to the input data that have been arranged in a particular format. For example, consider the following data:

15.75 123 John

This line contains three pieces of data, arranged in a particular form. Such data have to be read conforming to the format of its appearance. For example, the first part of the data should be read into a variable float, the second into int, and the third part into char. This is possible in C using the `scanf` function.

We have already used this input function in a number of examples. Here, we shall explore all the options that are available for reading the formatted data with `scanf` function. The general form of `scanf` is

```
scanf ("control string", arg1, arg2, ..... argn);
```

The control string specifies the field format in which the data are to be entered and the arguments arg1, arg2,…, argn specify the address of locations where the data are stored. Control string and arguments are separated by commas.

Control string (also known as format string) contains field specifications, which direct the interpretation of input data. It may include:

- Field (or format) specifications, consisting the conversion character %, a data type character (or type specifier), and an optional number, specifying the field width.
- Blanks, tabs or newlines.

- Blanks, tabs and newlines are ignored. The data type character indicates the type of data that is to be assigned to the variable associated with the corresponding argument. The field width specifier is optional.

Inputting Integer Numbers The field specification for reading an integer number is:

% w sd

The percentage sign (%) indicates that a conversion specification follows. w is an integer number that specifies the field width of the number to be read and d, known as data type character, indicates that the number to be read is in integer mode. Consider the following example:

```
scanf ("%2d %5d", &num1, &num2);
```

Suppose the following data are entered at the console:

50 31426

Here, the value 50 is assigned to num1 and 31426 to num2.

Now, suppose the input data are as follows:

31426 50

Now, the variable *num*1 will be assigned 31 (because of %2*d*) and *num*2 will be assigned 426 (unread part of 31426). The value 50 that is unread will be assigned to the first variable in the next scanf call. This kind of errors may be eliminated if we use the field specifications without the field-width specifications.

Input data items must be separated by spaces, tabs or newlines. Punctuation marks do not count as separators. When the scanf function searches the input data line for a value to be read, it will always bypass any white space characters.

What happens if we enter a floating point number instead of an integer? The fractional part may be stripped away! Also, scanf may skip reading further input.

When the scanf reads a particular value, reading of the value will be terminated as soon as the number of characters specified by the field width is reached (if specified) or until a character that is not valid for the value being read is encountered. In the case of integers, valid characters are an optionally signed sequence of digits.

An input field may be skipped by specifying * in the place of field width. For example, consider the following statement:

```
scanf("%d %*d %d", &a, &b)
```

Suppose, you enter the following data as input:

123 456 789

In this case, the value 123 will be assigned to *a* and 789 to *b*. The value 456 will be skipped (because of *).

Further, the data type character *d* may be preceded by 'l' to read long integers and h to read short integers.

EXAMPLE 4.8 *Various input formatting options for reading integers are experimented in the program shown in Fig. 4.28.*

```
Program
     main()
     {
        int a,b,c,x,y,z;
        int p,q,r;
        printf("Enter three integer numbers\n");
        scanf("%d %*d %d",&a,&b,&c);
        printf("%d %d %d \n\n",a,b,c);
        printf("Enter two 4-digit numbers\n");
        scanf("%2d %4d",&x,&y);
        printf("%d %d\n\n", x,y);
        printf("Enter two integers\n");
        scanf("%d %d", &a,&x);
        printf("%d %d \n\n",a,x);
        printf("Enter a nine digit number\n");
        scanf("%3d %4d %3d",&p,&q,&r);
        printf("%d %d %d \n\n",p,q,r);
        printf("Enter two three digit numbers\n");
        scanf("%d %d",&x,&y);
        printf("%d %d",x,y);
     }
Output
        Enter three integer numbers
        1 2 3
        1 3 -3577
        Enter two 4-digit numbers
        6789 4321
        67 89
        Enter two integers
        44 66
        4321 44
        Enter a nine-digit number
        123456789
        66 1234 567
        Enter two three-digit numbers
        123 456
        89 123
```

Fig. 4.28 **Reading integers using scanf**

In the above program in Fig. 4.28, the first scanf requests input data for three integer values *a*, *b* and *c*, and accordingly three values 1, 2 and 3 are keyed in. Because of the specification %*d, the value 2 has been skipped and 3 is assigned to the variable *b*. Notice that since no data are available for *c*, it contains garbage.

The second scanf specifies the format %2d and %4d for the variables *x* and *y* respectively. Whenever we specify field width for reading integer numbers, the input numbers should not contain more digits than the specified size. Otherwise, the extra digits on the right-hand side will be truncated and assigned to the next variable in the list. Thus, the second scanf has truncated the four digit number 6789 and assigned 67 to *x* and 89 to *y*. The value 4321 has been assigned to the first variable in the immediately following scanf statement.

NOTE: It is legal to use a non-whitespace character between field specifications. However, the scanf expects a matching character in the given location. For example, scanf("%d-%d", &a, &b); accepts input like123-456 to assign 123 to a and 456 to b.

Inputting Real Numbers Unlike integer numbers, the field width of real numbers is not to be specified and, therefore. scanf reads real numbers using the simple specification %f for both the notations, namely, decimal point notation and exponential notation. For example, consider the following statement

```
scanf("%f %f %f", &x, &y, &z);
```

Suppose, the following data are entered as input:

475.89 43.21E-1 678

Here, the value 475.89 will be assigned to *x*, 4.321 to *y* and 678.0 to *z*. The input field specifications may be separated by any arbitrary blank spaces.

If the number to be read is of double type, then the specification should be %lf instead of simple %f. A number may be skipped using %*f specification.

EXAMPLE 4.9 *Reading of real numbers (in both decimal point and exponential notation) is illustrated in Fig. 4.29.*

```
Program
     main()
     {
        float x,y;
        double p,q;
        printf("Values of x and y:");
        scanf("%f %e", &x, &y);
        printf("\n");
        printf("x = %f\ny = %f\n\n", x, y);
```

```
        printf("Values of p and q:");
        scanf("%lf %lf", &p, &q);
        printf("\n\np = %.12lf\np = %.12e", p,q);
    }
Output
        Values of x and y:12.3456 17.5e-2
        x = 12.345600
        y = 0.175000
        Values of p and q:4.142857142857 18.5678901234567890
        p = 4.142857142857
        q = 1.856789012346e+001
```

Fig. 4.29 **Reading of real numbers.**

Inputting Character Strings We have already seen how a single character can be read from the terminal using the getchar function. The same can be achieved using the scanf function also. In addition, a scanf function can input strings containing more than one character. Following are the specifications for reading character strings:

%ws or %wc

The corresponding argument should be a pointer to a character array. However, %c may be used to read a single character when the argument is a pointer to a char variable.

EXAMPLE 4.10 *Reading of strings using %wc and %ws is illustrated in Fig. 4.30.*

```
Program
    main()
    {
        int no;
        char name1[15], name2[15], name3[15];
        printf("Enter serial number and name one\n");
        scanf("%d %15c", &no, name1);
        printf("%d %15s\n\n", no, name1);
        printf("Enter serial number and name two\n");
        scanf("%d %s", &no, name2);
        printf("%d %15s\n\n", no, name2);
        printf("Enter serial number and name three\n");
        scanf("%d %15s", &no, name3);
        printf("%d %15s\n\n", no, name3);
    }
```

```
Output
        Enter serial number and name one
        1 123456789012345
        1 123456789012345r
        Enter serial number and name two
        2 New York
        2       New
        Enter serial number and name three
        2       York
        Enter serial number and name one
        1 123456789012
        1 123456789012r
        Enter serial number and name two
        2 New-York
        2    New-York
        Enter serial number and name three
        3 London
        3    London
```

Fig. 4.30 Reading of strings

The program in Fig. 4.30 illustrates the use of various field specifications for reading strings. When we use %wc for reading a string, the system will wait until the wth character is keyed in. Note that the specification %s terminates reading at the encounter of a blank space. Therefore, name2 has read only the first part of 'New York' and the second part is automatically assigned to name3. However, during the second run, the string 'New-York' is correctly assigned to name2.

Inputting Mixed Data Types It is possible to use one scanf statement to input a data line containing mixed mode data. In such cases, care should be exercised to ensure that the input data items match the control specifications in order and type. When an attempt is made to read an item that does not match the type expected, the scanf function does not read any further and immediately returns the values read. For example, consider the following statement:

```
scanf ("%d %c %f %s", &count, &code, &ratio, name);
```

Suppose, you enter the following data as input:

15 p 1.575 coffee

Here, the values will be correctly read and assigned to the variables in the order in which they appear. Some systems accept integers in the place of real numbers and vice versa, and the input data are converted to the type specified in the control string.


NOTE: A space before the %c specification in the format string is necessary to skip the white space before p.

Detecting Errors in Input When a scanf function completes reading its list, it returns the value of number of items that are successfully read. This value can be used to test whether any errors occurred in reading the input. For example, consider the following statement:

```
scanf ("%d %f %s, &a, &b, name);
```

Suppose, the following data is entered at the console:

```
20 150.25 motor
```

This will return the value 3 as three input values have been entered in correct order.

However, the value 1 will be returned if the following line is entered:

```
20 motor 150.25
```

This is because the function would encounter a string when it was expecting a floating-point value and would, therefore, terminate its scan after reading the first value.

EXAMPLE 4.11 *The program shown in Fig. 4.31 illustrates the testing for correctness of reading of data by scanf function.*

```
Program
     main()
     {
        int a;
        float b;
        char c;
        printf("Enter values of a, b and c\n");
        if (scanf("%d %f %c", &a, &b, &c) == 3)
           printf("a = %d b = %f c = %c\n" , a, b, c);
        else
           printf("Error in input.\n");
     }
Output
           Enter values of a, b and c
           12 3.45 A
           a = 12     b = 3.450000    c = A
           Enter values of a, b and c
```

```
23 78 9
a = 23    b = 78.000000   c = 9
Enter values of a, b and c
8 A 5.25
Error in input.
Enter values of a, b and c
Y 12 67
Error in input.
Enter values of a, b and c
15.75 23 X
a = 15    b = 0.750000    c = 2
```

Fig. 4.31 **Detection of errors in scanf input.**

The function `scanf` is expected to read three items of data and, therefore, when the values for all the three variables are read correctly, the program prints out their values. During the third run, the second item does not match with the type of variable and, therefore, the reading is terminated and the error message is printed. Same is the case with the fourth run.

In the last run, although data items do not match the variables, no error message has been printed. When we attempt to read a real number for an *int* variable, the integer part is assigned to the variable, and the truncated decimal part is assigned to the next variable.

Table 4.12 shows the commonly used `scanf` format codes:

TABLE 4.12 Commonly used scanf format codes

Code	Meaning
%c	read a single character
%d	read a decimal integer
%e	read a floating-point value
%f	read a floating-point value
%g	read a floating-point value
%h	read a short integer
%i	read a decimal, hexadecimal or octal integer
%o	read an octal integer
%s	read a string
%u	read an unsigned decimal integer
%x	read a hexadecimal integer
%[..]	read a string of word(s)

The following letters may be used as prefix for certain conversion characters.

- h: For short integers
- l: For long integers or double
- L: For long double

4.17.4 Points to Remember while Using scanf

If we do not plan carefully, some 'crazy' things can happen with scanf. Following are some of the general points to keep in mind while writing a scanf statement.

- All function arguments, except the control string, must be pointers to variables.
- Format specifications contained in the control string should match the arguments in order.
- Input data items must be separated by spaces and must match the variables receiving the input in the same order.
- The reading will be terminated, when scanf encounters a 'mismatch' of data or a character that is not valid for the value being read.
- When searching for a value, scanf ignores line boundaries and simply looks for the next appropriate character.
- Any unread data items in a line will be considered as part of the data input line to the next scanf call.
- When the field width specifier w is used, it should be large enough to contain the input data size.
- Never end the format string with whitespace. It is a fatal error!
- The scanf reads until:
  - A whitespace character is found in a numeric specification, or
  - The maximum number of characters have been read or
  - An error is detected, or
  - The end of file is reached

4.17.5 Formatted Output

We have seen the use of **`printf`** function for printing captions and numerical results. It is highly desirable that the outputs are produced in such a way that they are understandable and are in an easy-to-use form. It is, therefore, necessary for the programmer to give careful consideration to the appearance and clarity of the output produced by his program.

The **`printf`** statement provides certain features that can be effectively exploited to control the alignment and spacing of outputs on the terminals. The general form of printf statement is:

```
printf("control string", arg1, arg2, ....., argn);
```

In this code, the control string may comprise the following three types of items:

- Characters that will be printed on the screen as they appear.
- Format specifications that define the output format for display of each item.
- Escape sequence characters such as \n, \t, and \b.

The control string indicates how many arguments follow and what their types are. The arguments *arg*1, *arg*2, ..., *argn* are the variables whose values are formatted and printed according to the specifications of the control string. The arguments should match in number, order and type with the format specifications.

A simple format specification has the following form:

```
% w.p type-specifier
```

Here, *w* is an integer number that specifies the total number of columns for the output value and *p* is another integer number that specifies the number of digits to the right of the decimal point (of a real number) or the number of characters to be printed from a string. Both *w* and *p* are optional. Some examples of formatted printf statement are as under:

```
printf("Programming in C");
printf(" ");
printf("\n");
printf("%d", x);
printf("a = %f\n b = %f", a, b);
printf("sum = %d", 1234);
printf("\n\n");
```

printf never supplies a newline automatically and, therefore, multiple printf statements may be used to build one line of output. A newline can be introduced by the help of a newline character '\n' as shown in some of the examples above.

Output of Integer Numbers The format specification for printing an integer number is:

```
% w d
```

Here *w* specifies the minimum field width for the output. However, if a number is greater than the specified field width, it will be printed in full, overriding the minimum specification. *d* specifies that the value to be printed is an integer. The number is written right-justified in the given field width. Leading blanks will appear as necessary. Table 4.13 illustrates the output of the number 9876 under different formats.

TABLE 4.13 Commonly used scanf format codes

Format	Output
printf("%d", 9876)	9 8 7 6
printf("%6d", 9876)	␣ ␣ 9 8 7 6
printf("%2d", 9876)	9 8 7 6
printf("%-6d", 9876)	9 8 7 6 ␣ ␣
printf("%06d", 9876)	0 0 9 8 7 6

It is possible to force the printing to be left-justified by placing a minus sign directly after the % character, as shown in the fourth example here. It is also possible to pad with zeros the leading blanks by placing a 0 (zero) before the field width specifier, as shown in the last item here. The minus (–) and zero (0) are known as flags.

Long integers may be printed by specifying ld in the place of d in the format specification. Similarly, we may use hd for printing short integers.

EXAMPLE 4.12 *The program in Fig. 4.32 illustrates the output of integer numbers under various formats.*

```
Program
     main()
     {
       int m = 12345;
       long n = 987654;
       printf("%d\n",m);
       printf("%10d\n",m);
       printf("%010d\n",m);
       printf("%-10d\n",m);
       printf("%10ld\n",n);
       printf("%10ld\n",-n);
     }
Output
       12345
            12345
       0000012345
       12345
            987654
          – 987654
```

Fig. 4.32 Formatted output of integers.

Output of Real Numbers The output of a real number may be displayed in decimal notation using the following format specification:

```
% w.p f
```

The integer *w* indicates the minimum number of positions that are to be used for the display of the value and the integer *p* indicates the number of digits to be displayed after the decimal point (precision). The value, when displayed, is rounded to *p* decimal places and printed right-justified in the field of *w* columns. Leading blanks and trailing zeros will appear

as necessary. The default precision is 6 decimal places. The negative numbers will be printed with the minus sign. The number will be displayed in the form [–] mmm-nnn.

We can also display a real number in exponential notation by using the specification:

```
% w.p e
```

The display takes the following form:

```
[ - ] m.nnnne[ ± ]xx
```

Here, the length of the string of *n*'s is specified by the precision *p*. The default precision is 6. The field width *w* should satisfy the condition.

```
w ³ p+7
```

The value will be rounded off and printed right justified in the field of *w* columns.

Padding the leading blanks with zeros and printing with left-justification are also possible by using flags 0 or – before the field width specifier *w*. Table 4.14 illustrates the output of the number *y* = 98.7654 under different format specifications:

TABLE 4.14 Commonly used scanf format codes

Format	Output
printf("%7.4f",y)	9 8 . 7 6 5 4
printf("%7.2f",y)	␣ ␣ 9 8 . 7 6
printf("%-7.2f",y)	9 8 . 7 7 ␣ ␣
printf("%f",y)	9 8 . 7 6 5 4
printf("%10.2e",y)	␣ ␣ 9 . 8 8 e + 0 1
printf("%11.4e",-y)	– 9 . 8 7 6 5 e + 0 1
printf("%-10.2e",y)	9 . 8 8 e + 0 1 ␣ ␣
printf("%e",y)	9 . 8 7 6 5 4 0 e + 0 1

Some systems also support a special field specification character that lets the user define the field size at run time. This takes the following form:

```
printf("%*.*f", width, precision, number);
```

In this case, both the field width and the precision are given as arguments which will supply the values for *w* and *p*. For example, printf("%*.*f",7,2,number); is equivalent to printf("%7.2f",number);. The advantage of this format is that the values for width and precision may be supplied at run time, thus making the format a dynamic one.

EXAMPLE 4.13 *All the options of printing a real number are illustrated in Fig. 4.33.*

```
Program
     main()
     {
        float y = 98.7654;
        printf("%7.4f\n", y);
        printf("%f\n", y);
        printf("%7.2f\n", y);
        printf("%-7.2f\n", y);
        printf("%07.2f\n", y);
        printf("%*.*f", 7, 2, y);
        printf("\n");
        printf("%10.2e\n", y);
        printf("%12.4e\n", -y);
        printf("%-10.2e\n", y);
        printf("%e\n", y);
     }
Output
        98.7654
        98.765404
        98.77
        98.77
        0098.77
        98.77
        9.88e+001
        -9.8765e+001
        9.88e+001
        9.876540e+001
```

Fig. 4.33 Formatted output of real numbers

Printing of a Single Character A single character can be displayed in a desired position using the format:

```
%wc
```

The character will be displayed right-justified in the field of *w* columns. We can make the display left-justified by placing a minus sign before the integer *w*. The default value for *w* is 1.

Printing of Strings The format specification for outputting strings is similar to that of real numbers. It is of the following form:

```
%w.ps
```

Here, *w* specifies the field width for display and *p* instructs that only the first *p* characters of the string are to be displayed. The display is right-justified. Figure 4.34 shows the effect of variety of specifications in printing a string "NEW DELHI 110001", containing 16 characters (including blanks).

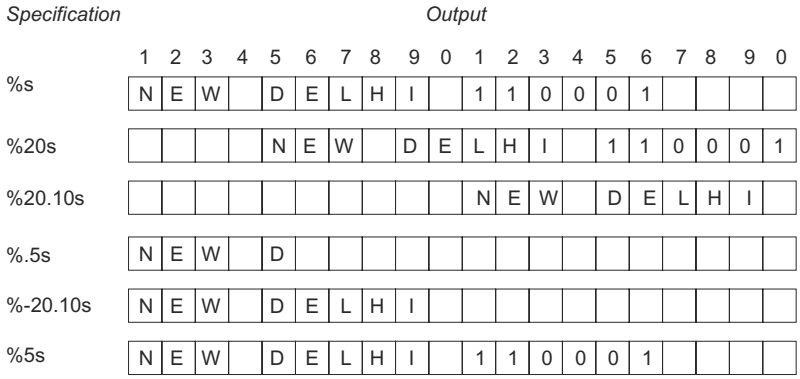


Fig. 4.34 Printing a string through different format specifications.

Mixed Data Output It is permitted to mix data types in one `printf` statement. For example, the following statement is valid in C.

```
printf("%d %f %s %c", a, b, c, d);
```

As pointed out earlier, printf uses its control string to decide how many variables to be printed and what their types are. Therefore, the format specifications should match the variables in number, order and type. If there are not enough variables or if they are of the wrong type, the output results will be incorrect.

Table 4.15 shows some commonly used printf format codes:

TABLE 4.15 Commonly used printf Format Codes

Code	Meaning
%c	print a single character
%d	print a decimal integer
%e	print a floating point value in exponent form
%f	print a floating point value without exponent
%g	print a floating point value either e-type or f-type depending on
%i	print a signed decimal integer

%o	print an octal integer, without leading zero
%s	print a string
%u	print an unsigned decimal integer
%x	print a hexadecimal integer, without leading Ox

The following letters may be used as prefix for certain conversion characters.

- h: for short integers
- l: for long integers or double
- L: for long double.

Enhancing the Readability of Output Computer outputs are used as information for analysing certain relationships between variables and for making decisions. Therefore, the correctness and clarity of outputs are of utmost importance. While the correctness depends on the solution procedure, the clarity depends on the way the output is presented. Following are some of the steps we can take to improve the clarity and hence the readability and understandability of outputs.

- Provide enough blank space between two numbers.
- Introduce appropriate headings and variable names in the output.
- Print special messages whenever a peculiar condition occurs in the output.
- Introduce blank lines between the important sections of the output.

4.18 CASE STUDIES

1. Inventory Report

Problem:

The ABC Electric Company manufactures four consumer products. Their inventory position on a particular day is given below:

Code	Quantity	Rate (Rs)
F105	275	575.00
H220	107	99.95
I019	321	215.50
M315	89	725.00

It is required to prepare the inventory report table in the following format:

INVENTORY REPORT

Code	Quantity	Rate	Value
____	____	____	____
____	____	____	____

———	———	———	———
———	———	———	———
		Total Value:	———

The value of each item is given by the product of quantity and rate.

Program:

The program given in Fig. 4.35 reads the data from the terminal and generates the required output.

```
Program
#define ITEMS 4
main()
{          /* BEGIN */
      int i, quantity[5];
      float rate[5], value, total_value;
      char code[5][5];        /* READING VALUES */
      i = 1;
      while ( i <= ITEMS)
      {
         printf("Enter code, quantity, and rate:");
         scanf("%s %d %f", code[i], &quantity[i],&rate[i]);
            i++;
      }
      /*.......Printing of Table and Column Headings.......*/
      printf("\n\n");
      printf(" INVENTORY REPORT \n");
      printf("- - - - - - - - - - - - - - - - - - - - - - \n");
      printf(" Code Quantity Rate Value \n");
      printf("- - - - - - - - - - - - - - - - - - - - - - \n");
      /*.......Preparation of Inventory Position.........*/
         total_value = 0;
         i = 1;
         while ( i <= ITEMS)
         {
            value = quantity[i] * rate[i];
            printf("%5s %10d %10.2f %e\n",code[i],quantity[i],rate[i],value);
            total_value += value;
            i++;
         }
      /*.......Printing of End of Table..................*/
         printf("- - - - - - - - - - - - - - - - - - \n");
```

```
printf("Total Value = %e\n",total_value);
        printf("- - - - - - - - - - - - - - - - - \n");
     } /* END */
Output
     Enter code, quantity, and rate:F105 275  575.00
     Enter code, quantity, and rate:H220 107 99.95
     Enter code, quantity, and rate:I019 321 215.50
     Enter code, quantity, and rate:M315 89 725.00
```

INVENTORY REPORT

Code	Quantity	Rate	Value
F105	275	575.00	1.581250e+005
H220	107	99.95	1.069465e+004
I019	321	215.50	6.917550e+004
M315	89	725.00	6.452500e+004
	Total Value =	3.025202e+005	

Fig. 4.35 **Program for inventory report**

2. Reliability Graph

Problem: The reliability of an electronic component is given by the following equation:

reliability (r) = $e^{-\lambda t}$

where:

- λ is the component failure rate per hour
- t is the time of operation in hours

A graph is required to determine the reliability at various operating times, from 0 to 3000 hours. The failure rate (lambda) is 0.001.

Program: The program given in Fig. 4.36 produces a shaded graph. The values of t are self-generated by the for statement

for (t=0; t <= 3000; t = t+150) in steps of 150.

The integer 50 in the statement R = (int)(50*r+0.5) is a scale factor which converts r to a large value where an integer is used for plotting the curve. Remember r is always less than 1.

```
Problem
#include <math.h>
#define LAMBDA 0.001
main()
{
     double t;
     float r;
```

```
      int i, R;
      for (i=1; i<=27; ++i)
      {
         printf("– –");
      }
      printf("\n");
      for (t=0; t<=3000; t+=150)
      {
         r = exp(–LAMBDA*t);
         R = (int)(50*r+0.5);
         printf(" |");
         for (i=1; i<=R; ++i)
         {
            printf("*");
         }
         printf("#\n");
      }
      for (i=1; i<3; ++i)
      {
         printf(" |\n");
      }
 }
Output
      ___________________________________________
      |************************************************#
      |*******************************************#
      |*********************************#
      |*****************************#
      |*************************#
      |***********************#
      |*******************#
      |*****************#
      |***************#
      |*************#
      |***********#
      |**********#
      |********#
      |*******#
      |******#
      |*****#
      |*****#
      |****#
      |***#
      |***#
      |**#
```

Fig. 4.36 Program to draw reliability graph

4.19 OPERATORS AND EXPRESSIONS

C supports a rich set of built-in operators. We have already used several of them, such as =, +, –, *, & and <. An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables. They usually form a part of the mathematical or logical expressions.

C operators can be classified into a number of categories. They include:

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Increment and decrement operators
- Conditional operators
- Bitwise operators
- Special operators

An expression is a sequence of operands and operators that reduces to a single value. For example, the following expression will result in the value 25.

15 + 10

4.19.1 Arithmetic Operators

C provides all the basic arithmetic operators. They are listed in Table 4.16. The operators +, –, * and / all work the same way as they do in other languages. These can operate on any built-in data type allowed in C. The unary minus operator, in effect, multiplies its single operand by –1. Therefore, a number preceded by a minus sign changes its sign.

TABLE 4.16 Arithmetic operators

Operator	Meaning
+	Addition or unary plus
–	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division

Integer division truncates any fractional part. The modulo division operation produces the remainder of an integer division. Examples of use of arithmetic operators are:

```
a – b
a + b
```

```
a * b
a / b
a % b
–a * b
```

Here, *a* and *b* are variables and are known as operands. The modulo division operator % cannot be used on floating point data. Note that C does not have an operator for exponentiation. Older version of C does not support unary plus but ANSI C supports it.

Integer Arithmetic When both the operands in a single arithmetic expression such as *a+b* are integers, the expression is called an integer expression, and the operation is called integer arithmetic. Integer arithmetic always yields an integer value. The largest integer value depends on the machine, as pointed out earlier. In the above examples, if *a* and *b* are integers, then for *a* = 14 and *b* = 4 we have the following results:

a – b = 10
a + b = 18
a * b = 56
a / b = 3 (decimal part truncated)
a % b = 2 (remainder of division)

During integer division, if both the operands are of the same sign, the result is truncated towards zero. If one of them is negative, the direction of truncation is implementation dependent. That is,

6/7 = 0 and –6/–7 = 0

but –6/7 may be zero or –1 (machine dependent).

Similarly, during modulo division, the sign of the result is always the sign of the first operand (the dividend), as shown further:

–14 % 3 = –2
–14 % –3 = –2
14 % –3 = 2

EXAMPLE 4.14 *The program in Fig. 4.37 shows the use of integer arithmetic to convert a given number of days into months and days.*

```
Program
     main ()
     {
        int months, days ;

        printf("Enter days\n") ;
        scanf("%d", &days) ;
```

```
        months = days / 30 ;
        days = days % 30 ;
        printf("Months = %d Days = %d", months, days) ;
    }
Output
    Enter days
    265
    Months = 8 Days = 25
    Enter days
    364
    Months = 12 Days = 4
    Enter days
    45
    Months = 1 Days = 15
```

Fig. 4.37 **Illustration of integer arithmetic.**

The variables months and days are declared as integers. Therefore, the statement `months = days/30;` truncates the decimal part and assigns the integer part to months. Similarly, the statement `days = days%30;` assigns the remainder part of the division to days. Thus the given number of days is converted into an equivalent number of months and days and the result is printed as shown in the output.

Real Arithmetic An arithmetic operation involving only real operands is called real arithmetic. A real operand may assume values either in decimal or exponential notation. Since, floating-point values are rounded to the number of significant digits permissible, the final value is an approximation of the correct result. If *x*, *y* and *z* are floats, then we will have:

x = 6.0/7.0 = 0.857143
y = 1.0/3.0 = 0.333333
z = –2.0/3.0 = –0.666667

The operator % cannot be used with real operands.

Mixed-mode Arithmetic When one of the operands is real and the other is integer, the expression is called a mixed-mode arithmetic expression. If either operand is of real type, then only the real operation is performed and the result is always a real number. Thus

15/10.0 = 1.5

whereas

15/10 = 1

More about mixed operations will be discussed later when we deal with the evaluation of expressions.

4.19.2 Relational Operators

We often compare two quantities and depending on their relation, take certain decisions. For example, we may compare the age of two persons, or the price of two items, and so on. These comparisons can be done with the help of relational operators. We have already used the symbol '<', meaning 'less than'.

An expression such as $a < b$ or $1 < 20$ containing a relational operator is termed as a relational expression. The value of a relational expression is either one or zero. It is one if the specified relation is true and zero if the relation is false. For example

10 < 20 is true

but

20 < 10 is false

C supports six relational operators in all. These operators and their meanings are illustrated in Table 4.17.

TABLE 4.17 Relational operators

Operator	Meaning
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

A simple relational expression contains only one relational operator and takes the following form:

```
ae-1 relational operator ae-2
```

Here, ae-1 and ae-2 are arithmetic expressions, which may be simple constants, variables or combination of them. Following are some examples of simple relational expressions and their values:

4.5 <= 10	TRUE
5 < –10	FALSE
–35 >= 0	FALSE
10 < 7+5	TRUE
$a+b = c+d$	TRUE only if the sum of values of a and b is equal to the sum of values of c and d.

When arithmetic expressions are used on either side of a relational operator, the arithmetic expressions will be evaluated first and then the results compared. That is, arithmetic operators have a higher priority over relational operators.

Relational expressions are used in decision-making statements such as **if** and **while** to decide the course of action of a running program.

4.19.3 Logical Operators

In addition to the relational operators, C supports the following three logical operators.

&&	meaning logical	AND
||	meaning logical	OR
!	meaning logical	NOT

The logical operators && and || are used when we want to test more than one condition and make decisions. An example is:

```
a > b && x == 10
```

An expression of this kind, which combines two or more relational expressions, is termed as a logical expression or a compound relational expression. Like the simple relational expressions, a logical expression also yields a value of one or zero, according to the truth table, shown in Table 4.18. The logical expression given here is true only if $a > b$ is true and $x == 10$ is true. If either (or both) of them are false, the expression is false.

TABLE 4.18 Truth table

op-1	op-2	Value of the expression	
		op-1 && op-2	op-1 || op-2
Non-zero	Non-zero	1	1
Non-zero	0	0	1
0	Non-zero	0	1
0	0	0	0

Some examples of the usage of logical expressions are:

1. if (age > 55 && salary < 1000)
2. if (number < 0 || number > 100)

We will see more of them when we discuss decision statements.

NOTE: Relative precedence of the relational and logical operators is as follows:

Highest	!
	> >= < <=
	== !=
	&&
Lowest	||

It is important to remember this when we use these operators in compound expressions.

4.19.4 Assignment Operators

Assignment operators are used to assign the result of an expression to a variable. We have seen the usual assignment operator, '='. In addition, C has a set of 'shorthand' assignment operators of the form

```
v op= exp;
```

Here, v is a variable, exp is an expression and op is a C binary arithmetic operator. The operator op= is known as the shorthand assignment operator.

The assignment statement v op= exp; is equivalent to

```
v = v op (exp);
```

Here, v evaluated only once.

Now, consider the following example

```
x += y+1;
```

This expression is same as the following:

```
x = x + (y+1);
```

The shorthand operator += means 'add y+1 to x' or 'increment x by y+1'. For y = 2, the above statement becomes

```
x += 3;
```

When the above statement is executed, 3 is added to x. If the old value of x is, say 5, then the new value of x is 8. Some of the commonly used shorthand assignment operators are illustrated in Table 4.19:

TABLE 4.19 Shorthand Assignment Operators

Statement with simple assignment operator	Statement with shorthand operator
a = a + 1	a += 1
a = a – 1	a –= 1
a = a * (n+1)	a *= n+1
a = a / (n+1)	a /= n+1
a = a % b	a %= b

The use of shorthand assignment operators has the following advantages:

- What appears on the left-hand side need not be repeated and therefore it becomes easier to write.
- The statement is more concise and easier to read.
- The statement is more efficient.

These advantages may be appreciated if we consider a slightly more involved statement like

```
value(5*j–2) = value(5*j–2) + delta;
```

With the help of the += operator, this can also be written as follows:

```
value(5*j–2) += delta;
```

It is easier to read and understand and is more efficient because the expression 5*j–2 is evaluated only once.

EXAMPLE 4.15 *The program in Fig. 4.38 prints a sequence of squares of numbers. Note the use of the shorthand operator *= .*

```
Program
     #define   N 100
     #define   A 2
     main()
     {
             int a;
             a = A;
             while( a < N )
             {
                  printf("%d\n", a);
                  a *= a;
```

```
            }
      }
Output
      2
      4
      16
```

Fig. 4.38 **Use of shorthand operator *=**

The program attempts to print a sequence of squares of numbers starting from 2. The statement a *= a; is identical to a = a*a; and it replaces the current value of a by its square. When the value of a becomes equal or greater than N (=100) the while is terminated. Note that the output contains only three values 2, 4 and 16.

4.19.5 Increment and Decrement Operators

C allows two very useful operators not generally found in other languages. These are the increment and decrement operators:

++ and — –

The operator ++ adds 1 to the operand, while – – subtracts 1. Both are unary operators and take the following form:

++m; or m++;
– –m; or m– –;
++m; is equivalent to m = m+1; (or m += 1;)
– –m; is equivalent to m = m–1; (or m –= 1;)

We use the increment and decrement statements in **for** and **while** loops extensively. While ++m and m++ mean the same thing when they form statements independently, they behave differently when they are used in expressions on the right-hand side of an assignment statement. Consider the following expressions:

```
m = 5;
y = ++m;
```

In this case, the value of *y* and *m* would be 6. Suppose, if we rewrite the above statements as

```
m = 5;
y = m++;
```

Here, the value of *y* would be 5 and *m* would be 6. A prefix operator first adds 1 to the operand and then the result is assigned to the variable on left. On the other hand, a postfix operator first assigns the value to the variable on left and then increments the operand.

Similar is the case when we use ++ (or – –) in subscripted variables. That is, the statement `a[i++] = 10;` is equivalent to the following:

```
a[i] = 10;
i = i+1;
```

The increment and decrement operators can be used in complex statements. For example:

```
m = n++ –j+10;
```

Here, the old value of *n* is used in evaluating the expression. *n* is incremented after the evaluation. Some compilers require a space on either side of *n*++ or ++*n*.

Some of the rules for using ++ and – – operators are:

- Increment and decrement operators are unary operators and thus require variable as their operands.
- When postfix ++ (or – –) is used with a variable in an expression, the expression is evaluated first using the original value of the variable and then the variable is incremented (or decremented) by one.
- When prefix ++ (or – –) is used in an expression, the variable is incremented (or decremented) first and then the expression is evaluated using the new value of the variable.
- The precedence and associativity of ++ and – – operators are the same as those of unary + and unary –.

4.19.6 Conditional Operator

A ternary operator pair '`? :`' is available in C to construct conditional expressions of the following form:

```
exp1 ? exp2 : exp3
```

where exp1, exp2 and exp3 are expressions.

The operator ? : works as follows: exp1 is evaluated first. If it is nonzero (true), then the expression exp2 is evaluated and becomes the value of the expression. If exp1 is false, exp3 is evaluated and its value becomes the value of the expression. Note that only one of the expressions (either exp2 or exp3) is evaluated. For example, consider the following statements.

```
a = 10;
b = 15;
x = (a > b) ? a : b;
```

In this example, x will be assigned the value of b. This can also be achieved using the if… else statements as follows:

```
if (a > b)
  x = a;
else
  x = b;
```

Now, consider the evaluation of the following function:

$$y = 1.5x + 3 \text{ for } x <= 2$$
$$y = 2x + 5 \text{ for } x > 2$$

This can be evaluated using the conditional operator as follows:

```
y = ( x > 2 ) ? (2 * x + 5) : (1.5 * x + 3);
```

The conditional operator may be nested for evaluating more complex assignment decisions. For example, consider the weekly salary of a salesgirl who is selling some domestic products. If x is the number of products sold in a week, her weekly salary is given by

$$\text{salary} = \begin{cases} 4x + 100 & \text{for } x < 40 \\ 300 & \text{for } x = 40 \\ 4.5x + 150 & \text{for } x > 40 \end{cases}$$

This complex equation can be written using conditional operators as follows:

```
salary = (x != 40) ? ((x < 40) ? (4*x+100) : (4.5*x+150)) : 300;
```

The same can be evaluated using if...else statements as follows:

```
if (x <= 40)
       if (x < 40)
         salary = 4 * x+100;
       else
         salary = 300;
else
         salary = 4.5 * x+150;
```

When the conditional operator is used, the code becomes more concise, and perhaps more efficient. However, the readability is poor. It is better to use if statements when more than a single nesting of conditional operator is required.

4.19.7 Bitwise Operators

C has a distinction of supporting special operators known as bitwise operators for manipulation of data at bit level. These operators are used for testing the bits, or shifting them right or left. Bitwise operators may not be applied to float or double. Table 4.20 provides a list the bitwise operators and their meanings.

TABLE 4.20 Bitwise operators

Operator	Meaning
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
<<	shift left
>>	shift right

4.19.8 Special Operators

C supports some special operators of interest such as comma operator, sizeof operator, pointer operators (& and *) and member selection operators (. and –>).

Comma Operator The comma operator can be used to link the related expressions together. A comma-linked list of expressions are evaluated left to right and the value of the right-most expression is the value of the combined expression. For example, consider the following statement:

```
value = (x = 10, y = 5, x+y);
```

This statement first assigns the value 10 to *x*, then assigns 5 to *y*, and finally assigns 15 (i.e. 10 + 5) to value. Since comma operator has the lowest precedence of all operators, the parentheses are necessary. Some applications of comma operator are:

- In for loops:

```
for ( n = 1, m = 10, n <=m; n++, m++)
```

- In while loops:

```
while (c = getchar( ), c != '10')
```

- Exchanging values:

```
t = x, x = y, y = t;
```

sizeof Operator The sizeof is a compile time operator and, when used with an operand, it returns the number of bytes the operand occupies. The operand may be a variable, a constant or a data type qualifier. Some examples arte shown further:

```
m = sizeof (sum);
n = sizeof (long int);
k = sizeof (235L);
```

The sizeof operator is normally used to determine the lengths of arrays and structures when their sizes are not known to the programmer. It is also used to allocate memory space dynamically to variables during execution of a program.

EXAMPLE 4.16 *Figure 4.39 shows a program that employs different kinds of operators. The results of their evaluation are also shown for comparison.*

```
Program

     main()
     {
          int a, b, c, d;
          a = 15;
          b = 10;
          c = ++a - b;
          printf("a = %d b = %d c = %d\n",a, b, c);
          d = b++ +a;
          printf("a = %d b = %d d = %d\n",a, b, d);
          printf("a/b = %d\n", a/b);
          printf("a%%b = %d\n", a%b);
          printf("a *= b = %d\n", a*=b);
          printf("%d\n", (c>d) ? 1 : 0);
          printf("%d\n", (c<d) ? 1 : 0);
     }
Output
        a = 16  b = 10 c = 6
        a = 16  b = 11 d = 26
        a/b = 1
        a%b = 5
        a *= b = 176
        0
        1
```

Fig. 4.39 Further illustration of arithmetic operators.

4.19.9 Operator Precedence

Precedence of operators refers to the order in which they are operated in a program. Table 4.21 provides a list of the precedence of operators.

TABLE 4.21 Precedence of operators

Type of operator	Operators	Associativity
Unary operators	+, –, !, ~, ++, --, type, size of	Right to left
Arithmetic operators	*, /, %, +, -	Left to right
Bit-manipulation operators	<<, >>	Left to right
Relational operators	>, <, >=, <=, = =, !=	Left to right
Logical operators	&&, ||	Left to right
Conditional operators	?, :	Left to right
Assignment operators	=, +=, -=, *=, /=, %=	Right to left

It is important to note the following:

- Precedence rules decide the order in which different operators are applied
- Associativity rule decides the order in which multiple occurrences of the same level operator are applied

Evaluation of Expressions Expressions are evaluated using an assignment statement of the form:

```
variable = expression;
```

Variable is any valid C variable name. When the statement is encountered, the expression is evaluated first and the result then replaces the previous value of the variable on the left-hand side. All variables used in the expression must be assigned values before evaluation is attempted. Examples of evaluation statements are as follows:

```
x = a * b - c;
y = b / c * a;
z = a - b / c + d;
```

The blank space around an operator is optional and adds only to improve readability. When these statements are used in a program, the variables *a*, *b*, *c* and *d* must be defined before they are used in the expressions.

EXAMPLE 4.17 *The program in Fig. 4.40 illustrates the use of variables in expressions and their evaluation. The output of the program also illustrates the effect of presence of parentheses in expressions.*

```
Program

  main()
  {
       float a, b, c, x, y, z;
       a = 9;
       b = 12;
       c = 3;

       x = a – b / 3 + c * 2 - 1;
       y = a – b / (3 + c) * (2 - 1);
       z = a – (b / (3 + c) * 2) - 1;

       printf("x = %f\n", x);
       printf("y = %f\n", y);
       printf("z = %f\n", z);
  }
Output

       x = 10.000000
       y = 7.000000
       z = 4.000000
```

Fig. 4.40 Illustrations of evaluation of expressions

4.19.10 Precedence of Arithmetic Operators

An arithmetic expression without parentheses will be evaluated from left to right using the rules of precedence of operators. There are two distinct priority levels of arithmetic operators in C:

High priority * / %

Low priority + –

The basic evaluation procedure includes 'two' left-to-right passes through the expression. During the first pass, the high priority operators (if any) are applied as they are encountered. During the second pass, the low priority operators (if any) are applied as they are encountered.

Now, consider the following evaluation statement:

$$x = a–b/3 + c*2–1$$

When a = 9, b = 12, and c = 3, the statement becomes

$$x = 9–12/3 + 3*2–1$$

And, it is evaluated as follows

First pass

Step1: x = 9–4+3*2–1

Step2: x = 9–4+6–1

Second pass

Step3: x = 5+6–1

Step4: x = 11–1

Step5: x = 10

These steps are illustrated in Fig. 4.41. The numbers inside parentheses refer to step numbers.

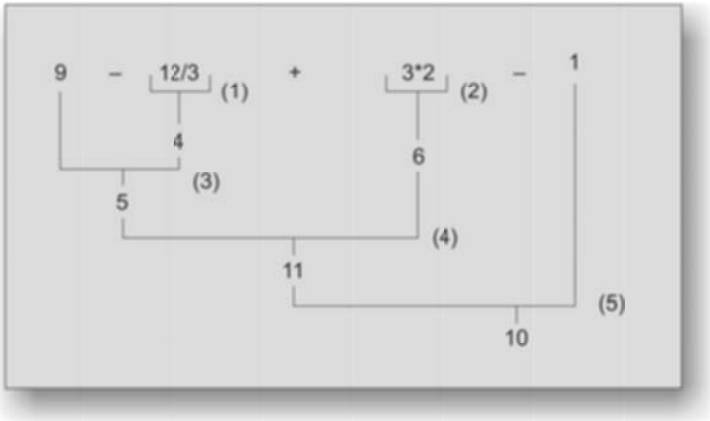


Fig. 4.41 **Illustration of hierarchy of operations**

However, the order of evaluation can be changed by introducing parentheses into an expression.

Consider the same expression with parentheses as shown below:

$$9–12/(3+3)*(2–1)$$

Whenever parentheses are used, the expressions within parentheses assume highest priority. If two or more sets of parentheses appear one after another as shown above, the expression contained in the left-most set is evaluated first and the right-most in the last. Given below are the new steps.

First pass

Step1: 9-12/6 * (2-1)

Step2: 9-12/6 * 1

Second pass

Step3: 9-2 * 1

Step4: 9-2

Third pass

Step5: 7

This time, the procedure consists of three left-to-right passes. However, the number of evaluation steps remains the same as 5 (i.e. equal to the number of arithmetic operators).

Parentheses may be nested, and in such cases, evaluation of the expression will proceed outward from the innermost set of parentheses. Just make sure that every opening parenthesis has a matching closing parenthesis.

While parentheses allow us to change the order of priority, we may also use them to improve understandability of the program. When in doubt, we can always add an extra pair just to make sure that the priority assumed is the one we require.

Rules for Evaluation of Expression

1. First, parenthesized sub-expression from left to right are evaluated.
2. If parentheses are nested, the evaluation begins with the innermost sub-expression.
3. The precedence rule is applied in determining the order of application of operators in evaluating sub-expressions.
4. The associativity rule is applied when two or more operators of the same precedence level appear in a sub-expression.
5. Arithmetic expressions are evaluated from left to right using the rules of precedence.
6. When parentheses are used, the expressions within parentheses assume highest priority.

4.19.11 Some Computational Problems

When expressions include real values, then it is important to take necessary precautions to guard against certain computational errors. We know that the computer gives approximate values for real numbers and the errors due to such approximations may lead to serious problems. For example, consider the following statements:

```
a = 1.0/3.0;
b = a * 3.0;
```

We know that (1.0/3.0) 3.0 is equal to 1. But there is no guarantee that the value of b computed in a program will equal 1.

Another problem is division by zero. On most computers, any attempt to divide a number by zero will result in abnormal termination of the program. In some cases such a division may produce meaningless results. Care should be taken to test the denominator that is likely to assume zero value and avoid any division by zero.

The third problem is to avoid overflow or underflow errors. It is our responsibility to guarantee that operands are of the correct type and range, and the result may not produce any overflow or underflow.

EXAMPLE 4.18 *The program in Fig. 4.42 shows round-off errors that can occur in computation of floating point numbers.*

```
Program
     /*————————— Sum of n terms of 1/n ————————————*/
     main()
     {
        float sum, n, term ;
        int count = 1 ;
        sum = 0 ;
        printf("Enter value of n\n") ;
           scanf("%f", &n) ;
        term = 1.0/n ;
        while( count <= n )
        {
             sum = sum + term ;
             count++ ;
        }
        printf("Sum = %f\n", sum) ;
     }
Output
        Enter value of n
        99
        Sum = 1.000001
        Enter value of n
        143
        Sum = 0.999999
```

Fig. 4.42 Round-off errors in floating point computations

We know that the sum of n terms of 1/n is 1. However, due to errors in floating point representation, the result is not always 1.

4.19.12 Type Conversions in Expressions

Implicit Type Conversion C permits mixing of constants and variables of different types in an expression. C automatically converts any intermediate values to the proper type so that the expression can be evaluated without losing any significance. This automatic conversion is known as implicit-type conversion.

During evaluation, it adheres to very strict rules of type conversion. If the operands are of different types, the 'lower' type is automatically converted to the 'higher' type before the operation proceeds. The result is of the higher type. A typical type conversion process is illustrated in Fig. 4.43.

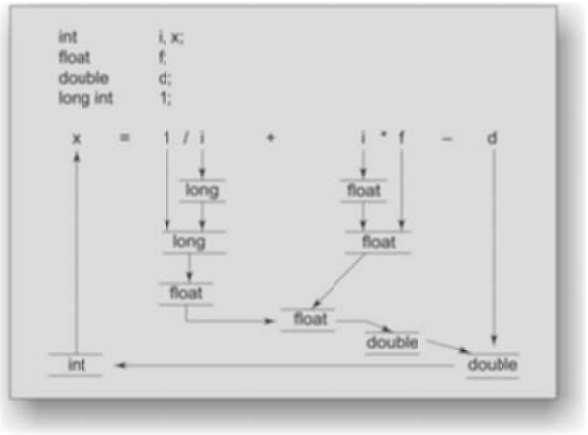


Fig. 4.43 **Process of implicit type conversion**

Figure 4.44 shows the cconversion hierarchy for implicit-type conversion in an expression.

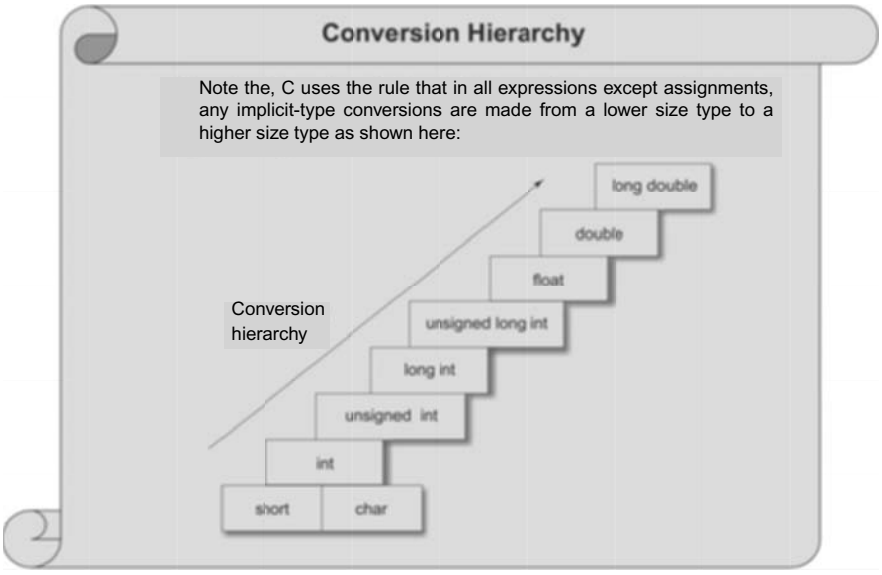


Fig. 4.44 **Conversion hierarchy**

The sequence of rules that are applied while implicit type conversion is as follows: all `short` and `char` are automatically converted to int; then

1. if one of the operands is `long double`, the other will be converted to `long double` and the result will be `long double`;
2. else, if one of the operands is `double`, the other will be converted to `double` and the result will be `double`;
3. else, if one of the operands is `float`, the other will be converted to `float` and the result will be `float`;
4. else, if one of the operands is `unsigned long int`, the other will be converted to `unsigned long int` and the result will be `unsigned long int`;
5. else, if one of the operands is `long int` and the other is `unsigned int`, then
   (a) if `unsigned int` can be converted to `long int`, the `unsigned int` operand will be converted as such and the result will be `long int`;
   (b) else, both operands will be converted to `unsigned long int` and the result will be `unsigned long int`;
6. else, if one of the operands is `long int`, the other will be converted to `long int` and the result will be `long int`;
7. else, if one of the operands is `unsigned int`, the other will be converted to `unsigned int` and the result will be `unsigned int`.

The final result of an expression is converted to the type of the variable on the left of the assignment sign before assigning the value to it. However, the following changes are introduced during the final assignment.

- `float` to `int` causes truncation of the fractional part.
- `double` to `float` causes rounding of digits.
- `long` int to `int` causes dropping of the excess higher order bits.

Explicit Conversion We have just learnt how C performs type conversion automatically. However, there are instances when we want to force a type conversion in a way that is different from the automatic conversion. Consider, for example, the calculation of ratio of females to males in a town.

ratio = female_number/male_number

Since female_number and male_number are declared as integers in the program, the decimal part of the result of the division would be lost and ratio would represent a wrong figure. This problem can be solved by converting locally one of the variables to the floating point as shown below:

ratio = (float) female_number/male_number

The operator (`float`) converts the female_number to floating point for the purpose of evaluation of the expression. Then using the rule of automatic conversion, the division is performed in floating point mode, thus retaining the fractional part of result.

Note that in no way does the operator (float) affect the value of the variable female number. And also, the type of female number remains as int in the other parts of the program.

The process of such a local conversion is known as explicit conversion or casting a value. The general form of a cast is:

(type-name) expression

Here, type-name is one of the standard C data types. The expression may be a constant, variable or an expression. Some examples of casts and their actions are shown in Table 4.22.

TABLE 4.22 Use of casts

Example	Action
x = (int) 7.5	7.5 is converted to integer by truncation.
a = (int) 21.3/(int)4.5	Evaluated as 21/4 and the result would be 5.
b = (double)sum/n	Division is done in floating point mode.
y = (int) (a+b)	The result of *a*+*b* is converted to integer.
z = (int)a+b	*a* is converted to integer and then added to *b*.
p = cos((double)x)	Converts *x* to double before using it.

Casting can be used to round-off a given value. For example, consider the following statement:

```
x =  (int) (y+0.5);
```

If *y* is 27.6, *y*+0.5 is 28.1 and on casting, the result becomes 28, the value that is assigned to *x*. Of course, the expression, being cast is not changed.

4.19.13 Operator Precedence and Associativity

As mentioned earlier each operator, in C has a precedence associated with it. This precedence is used to determine how an expression involving more than one operator is evaluated. There are distinct levels of precedence and an operator may belong to one of these levels. The operators at the higher level of precedence are evaluated first. The operators of the same precedence are evaluated either from 'left to right' or from 'right to left', depending on the level. This is known as the associativity property of an operator. Table 4.23 provides a complete list of operators, their precedence levels, and their rules of association. The groups are listed in the order of decreasing precedence. Rank 1 indicates the highest precedence level and 15 the lowest. The list also includes those operators, which we have not yet been discussed.

TABLE 4.23 Summary of C Operators

Operator	Description	Associativity	Rank
( ) [ ]	Function call Array element reference	Left to right	1
+ – ++ – – ! ~ * & sizeof (type)	Unary plus Unary minus Increment Decrement Logical negation Ones complement Pointer reference (indirection) Address Size of an object Type cast (conversion)	Right to left	2
* / %	Multiplication Division Modulus	Left to right	3
+ –	Addition Subtraction	Left to right	4
<< >>	Left shift Right shift	Left to right	5
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left to right	6
== |=	Equality Inequality	Left to right	7
&	Bitwise AND	Left to right	8
^	Bitwise XOR	Left to right	9
|	Bitwise OR	Left to right	10
&&	Logical AND	Left to right	11
||	Logical OR	Left to right	12
?:	Conditional expression	Right to left	13
= * = /= %= += –= &= ^= |= <<= >>=	Assignment operators	Right to left	14
,	Comma operator	Left to right	15

It is very important to note carefully, the order of precedence and associativity of operators. Consider the following conditional statement:

```
if (x == 10 + 15 && y < 10)
```

The precedence rules say that the addition operator has a higher priority than the logical operator (&&) and the relational operators (== and <). Therefore, the addition of 10 and 15 is executed first. This is equivalent to:

```
if (x == 25 && y < 10)
```

The next step is to determine whether x is equal to 25 and y is less than 10. If we assume a value of 20 for x and 5 for y, then

x == 25 is FALSE (0)

y < 10 is TRUE (1)

Note that since the operator < enjoys a higher priority as compared to ==, y < 10 is tested first and then x == 25 is tested.

Finally we get:

if (FALSE && TRUE)

Because one of the conditions is FALSE, the complex condition is FALSE.

In the case of &&, it is guaranteed that the second operand will not be evaluated if the first is zero and in the case of ||, the second operand will not be evaluated if the first is non-zero.

Applying De Morgan's Rule

While designing decision statements, we often come across a situation where the logical NOT operator is applied to a compound logical expression, like !(x&&y||!z). However, a positive logic is always easy to read and comprehend than a negative logic. In such cases, we may apply what is known as De Morgan's rule to make the total expression positive. This rule is as follows:

"Remove the parentheses by applying the NOT operator to every logical expression component, while complementing the relational operators."

That is,

- x becomes !x
- !x becomes x
- && becomes ||
- || becomes &&

Examples:

- !(x && y || !z) becomes !x || !y && z
- !(x <=0 || !condition) becomes x >0&& condition

Dangling Else Problem

One of the classic problems encountered when we start using nested if….else statements is the dangling else. This occurs when a matching else is not available for an if. The answer to this problem is very simple. Always match an else to the most recent unmatched if in the current block. In some cases, it is possible that the false condition is not required. In such situations, else statement may be omitted

"else is always paired with the most recent unpaired if"

4.20 CASE STUDIES

1. Salesman's Salary

Problem:

A computer manufacturing company has the following monthly compensation policy to their sales-persons:

- Minimum base salary: 1500.00
- Bonus for every computer sold: 200.00
- Commission on the total monthly sales: 2 per cent

Since the prices of computers are changing, the sales price of each computer is fixed at the beginning of every month.

Program:

Given the base salary, bonus, and commission rate, the inputs necessary to calculate the gross salary are the price of each computer and the number sold during the month

The gross salary is given by the equation:

Gross salary = base salary + (quantity * bonus rate) + (quantity * Price) * commission rate

A program to compute a sales-person's gross salary is given in Fig. 4.45.

```
Program
#define BASE_SALARY 1500.00
#define BONUS_RATE  200.00
#define COMMISSION  0.02
main()
{
      int quantity;
      float gross_salary, price;
      float bonus, commission;
      printf("Input number sold and price\n");
      scanf("%d %f", &quantity, &price);
      bonus      = BONUS_RATE * quantity;
      commission   = COMMISSION * quantity * price;
      gross_salary  = BASE_SALARY + bonus + commission ;
```

```
      printf("\n");
      printf("Bonus   = %6.2f\n", bonus);
      printf("Commission = %6.2f\n", commission);
      printf("Gross salary  = %6.2f\n", gross_salary);
}
Output
        Input number sold and price
        5 20450.00
        Bonus        = 1000.00
        Commission   = 2045.00
        Gross salary = 4545.00
```

Fig. 4.45 Program of salesman's salary

2. Solution of the Quadratic Equation

An equation of the form is known as the quadratic equation:

$$ax^2 + bx + c = 0$$

The values of x that satisfy the equation are known as the roots of the equation. A quadratic equation has two roots, which are given by the following two formulae:

$$\text{root 1} = \frac{-b + \text{sqrt}(b^2 - 4ac)}{2a}$$

$$\text{root 2} = \frac{-b - \text{sqrt}(b^2 - 4ac)}{2a}$$

A program to evaluate these roots is given in Fig. 4.46. The program requests the user to input the values of a, b and c and outputs root 1 and root 2.

```
Program
#include <math.h>
main()
{
      float a, b, c, discriminant, root1, root2;
      printf("Input values of a, b, and c\n");
      scanf("%f %f %f", &a, &b, &c);
      discriminant = b*b - 4*a*c ;
      if(discriminant < 0)
          printf("\n\nROOTS ARE IMAGINARY\n");
      else
      {
```

```
            root1 = (-b + sqrt(discriminant))/(2.0*a);
            root2 = (-b - sqrt(discriminant))/(2.0*a);
            printf("\n\nRoot1 = %5.2f\n\nRoot2 = %5.2f\n",
            root1,root2 );
          }
}
Output
      Input values of a, b, and c
      2 4 -16
      Root1 = 2.00
      Root2 = -4.00
      Input values of a, b, and c
      1 2 3
      ROOTS ARE IMAGINARY
```

Fig. 4.46 **Solution of a quadratic equation**

The term (b^2–4ac) is called the discriminant. If the discriminant is less than zero, its square roots cannot be evaluated. In such cases, the roots are said to be imaginary numbers and the program outputs an appropriate message.

4.21 DECISION MAKING AND BRANCHING

We have seen that a C program is a set of statements that are normally executed sequentially in the order in which they appear. This happens when no options or no repetitions of certain calculations are necessary. However, in practice, we have a number of situations where we may have to change the order of execution of statements based on certain conditions, or repeat a group of statements until certain specified conditions are met. This involves a kind of decision making to see whether a particular condition has occurred or not and then direct the computer to execute certain statements accordingly.

C language possesses such decision-making capabilities by supporting the following statements:

- if statement
- switch statement
- goto statement

These statements are popularly known as decision-making statements. Since these statements 'control' the flow of execution, they are also known as control statements. We have already used some of these statements in the earlier examples. Here, we will discuss their features, capabilities and applications in detail.

4.21.1 Decision Making with if Statement

The if statement is a powerful decision-making statement and is used to control the flow of execution of statements. It is basically a two-way decision statement and is used in conjunction with an expression. It takes the following form:

```
if (test expression)
```

It allows the computer to evaluate the expression first and then, depending on whether the value of the expression (relation or condition) is 'true' (or non-zero) or 'false' (zero), it transfers the control to a particular statement. This point of program has two paths to follow—one for the true condition and the other for the false condition,as shown in Fig. 4.47.

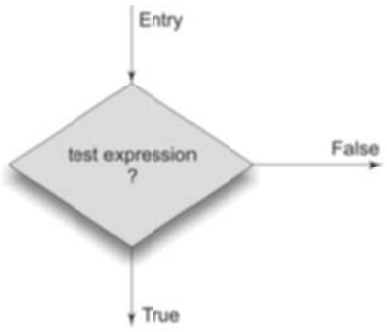


Fig. 4.47 **Two-way branching**

Some examples of decision making, using if statements, are:

- if (bank balance is zero)
  borrow money
- if (room is dark)
  put on lights
- if (code is 1)
  person is male
- if (age is more than 55)
  person is retired

The if statement may be implemented in different forms depending on the complexity of conditions to be tested. The different forms are:

- Simple if statement
- If…else statement
- Nested if…else statement
- else if ladder.

We will discuss each one of them in the following sections.

Simple if Statement The general form of a simple if statement is

```
if (test expression)
{
 statement-block;
}
statement-x;
```

The `statement-block` may be a single statement or a group of statements. If the `test expression` is true, the `statement-block` will be executed; otherwise the `statement-block` will be skipped and the execution will jump to the `statement-x`. Remember, when the condition is true, both the `statement-block` and the `statement-x` are executed in sequence. This is illustrated in Fig. 4.48.

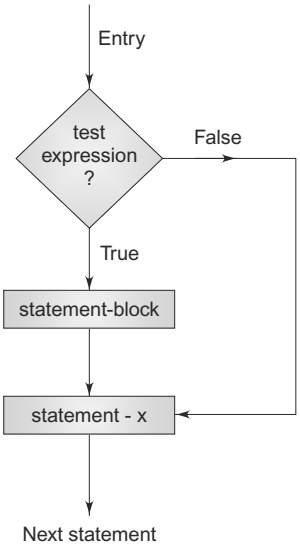


Fig. 4.48 **Flow chart of simple if control**

Consider the following segment of a program that is written for processing of the marks obtained in an entrance examination.

```
.........
.........
if (category == SPORTS)
{
marks = marks + bonus_marks;
}
printf("%f", marks);
.........
.........
```

The program tests the type of category of the student. If the student belongs to the `SPORTS` category, then additional `bonus_marks` are added to his marks before they are printed. For others, `bonus_marks` are not added.

EXAMPLE 4.19 *The program given in Fig. 4.49 reads four values a, b, c and d from the terminal and evaluates the ratio of (a+b) to (c–d) and prints the result, if c–d is not equal to zero.*

```
Program
     main()
     {
          int a, b, c, d;
          float ratio;

          printf("Enter four integer values\n");
          scanf("%d %d %d %d", &a, &b, &c, &d);

          if (c-d != 0) /* Execute statement block */
          {
               ratio = (float)(a+b)/(float)(c-d);
               printf("Ratio = %f\n", ratio);
          }
     }

Output

     Enter four integer values
     12 23 34 45
     Ratio = -3.181818

     Enter four integer values
     12 23 34 34
```

Fig. 4.49 Illustration of simple if statement

The above program has been run for two sets of data to see that the paths function properly. The result of the first run is printed as,

```
Ratio = –3.181818
```

The second run has neither produced any results nor any message. During the second run, the value of (*c–d*) is equal to zero and, therefore, the statements contained in the `statement-block` are skipped. Since no other statement follows the `statement-block`, program stops without producing any output.

Note the use of float conversion in the statement evaluating the ratio. This is necessary to avoid truncation due to integer division. Remember, the output of the first run –3.181818 is printed correct to six decimal places. The answer contains a round off error. If we wish to have higher accuracy, we must use double or long double data type.

The simple if is often used for counting purposes, as illustrated in the following example.

EXAMPLE 4.20 *The program given in Fig. 4.50 counts the number of boys whose weight is less than 50 kg and height is greater than 170 cm.*

```
Program
     main()
     {
          int count, i;
          float weight, height;

          count = 0;
          printf("Enter weight and height for 10 boys\n");

          for (i =1; i <= 10; i++)
          {
               scanf("%f %f", &weight, &height);
               if (weight < 50 && height > 170)
                    count = count + 1;
          }
          printf("Number of boys with weight < 50 kg\n");
          printf("and height > 170 cm = %d\n", count);
     }

Output

     Enter weight and height for 10 boys
     45   176.5
     55   174.2
     47   168.0
     49   170.7
     54   169.0
     53   170.5
     49   167.0
     48   175.0
     47   167
     51   170
     Number of boys with weight < 50 kg
     and height > 170 cm = 3
```

Fig. 4.50 **Use of if for counting**

The above program tests two conditions, one for weight and another for height. This is done using the compound relation

```
if (weight < 50 && height > 170)
```

This would have been equivalently done using two if statements as follows:

```
if (weight < 50)
if (height > 170)
count = count +1;
```

If the value of weight is less than 50, then the following statement is executed, which in turn is another `if` statement. This `if` statement tests height and if the height is greater than 170, then the count is incremented by 1.

The if…else Statement The if...else statement is an extension of the simple if statement. It takes the following general form:

```
If (test expression)
{
  True-block statement(s)
}
else
{
  False-block statement(s)
}
statement-x
```

If the test expression is true, then the true-block statement(s), immediately following the if statements, are executed; otherwise, the false-block statement(s) are executed. In either case, either true-block or false-block will be executed, not both. This is illustrated in Fig. 4.51. In both the cases, the control is transferred subsequently to the statement-x.

Let us consider an example of counting the number of boys and girls in a class. We use code 1 for a boy and 2 for a girl. The program statement to do this may be written as follows:

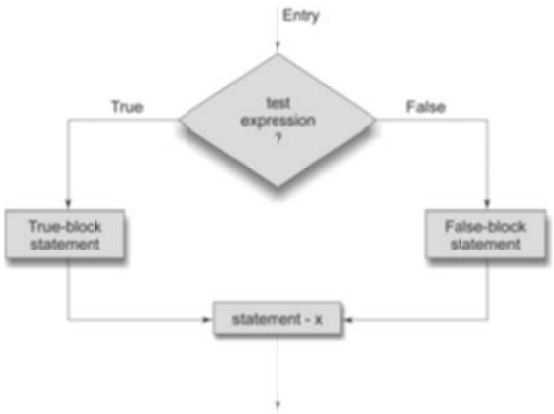


Fig. 4.51 Flow chart of if…else control

```
.........
.........
if (code == 1)
     boy = boy + 1;
if (code == 2)
     girl = girl+1;
.........
.........
```

The first test determines whether or not the student is a boy. If yes, the number of boys is increased by 1 and the program continues to the second test. The second test again determines whether the student is a girl. This is unnecessary. Once a student is identified as a boy, there is no need to test again for a girl. A student can be either a boy or a girl, not both. The above program segment can be modified using the `else` clause as follows:

```
..........
..........
if (code == 1)
     boy = boy + 1;
else
     girl = girl + 1;
xxxxxxxxxx
..........
```

Here, if the code is equal to 1, the statement `boy = boy + 1;` is executed and the control is transferred to the statement `xxxxxx`, after skipping the else part. If the code is not equal to 1, the statement `boy = boy + 1;` is skipped and the statement in the `else` part `girl = girl + 1;` is executed before the control reaches the statement `xxxxxxxx`.

Nesting of if…else Statements When a series of decisions are involved, we may have to use more than one if...else statement in nested form as shown in Fig. 4.52:

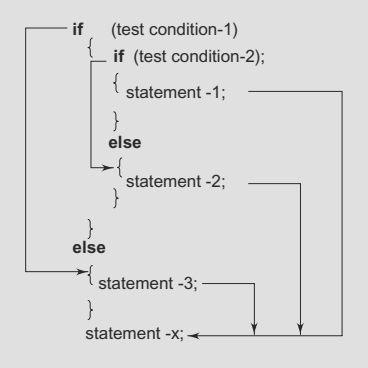


Fig. 4.52 **Nested if…else statements**

The logic of execution is illustrated in Fig. 4.53. If the condition-1 is false, the statement-3 will be executed; otherwise it continues to perform the second test. If the condition-2 is true, the statement-1 will be evaluated; otherwise the statement-2 will be evaluated and then the control is transferred to the statement-x.

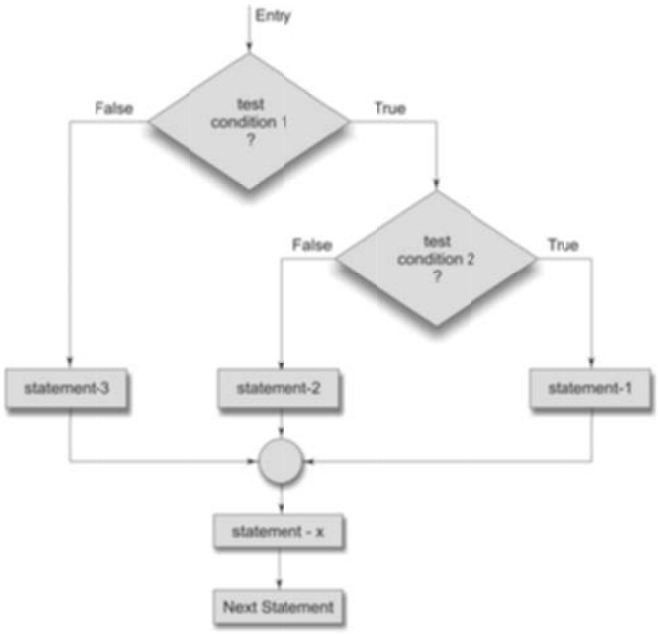


Fig. 4.53 Flow chart of nested if…else statements

Let us now try to understand the concept of nested `if…else` with the help of a real-life example. A commercial bank has introduced an incentive policy of giving bonus to all its deposit holders. The policy is as follows: A bonus of 2 per cent of the balance held on 31st December is given to every one, irrespective of their balance, and 5 per cent is given to female account holders if their balance is more than Rs. 5000. This logic can be coded with the help of nested `if…else` as follows:

```
.........
if (sex is female)
{
     if (balance > 5000)
       bonus = 0.05 * balance;
     else
       bonus = 0.02 * balance;
}
```

```
else
{
     bonus = 0.02 * balance;
}
balance = balance + bonus;
.........
.........
```

When nesting, care should be exercised to match every `if` with an `else`. Consider the following alternative to the earlier program (which looks right at the first sight):

```
if (sex is female)
     if (balance > 5000)
       bonus = 0.05 * balance;
     else
       bonus = 0.02 * balance;
     balance = balance + bonus;
```

There is an ambiguity as to over which if the else belongs to. In C, an else is linked to the closest non-terminated if. Therefore, the else is associated with the inner if and there is no else option for the outer if. This means that the computer is trying to execute the statement balance = balance + bonus; without really calculating the bonus for the male account holders.

EXAMPLE 4.21 *The program given in Fig. 4.54 selects and prints the largest of the three numbers using nested if…else statements.*

```
Program
     main()
     {
     float A, B, C;
     printf("Enter three values\n");
     scanf("%f %f %f", &A, &B, &C);
     printf("\nLargest value is ");
     if (A>B)
     {
       if (A>C)
         printf("%f\n", A);
       else
         printf("%f\n", C);
     }
```

```
      else
      {
         if (C>B)
            printf("%f\n", C);
         else
            printf("%f\n", B);
      }
 }
Output
      Enter three values
      23445 67379 88843
      Largest value is 88843.000000
```

Fig. 4.54 **Selecting the largest of three numbers.**

The else if Ladder There is another way of putting ifs together when multipath decisions are involved. A multipath decision is a chain of ifs in which the statement associated with each else is an if. It takes the general form, as shown in Fig. 4.55:

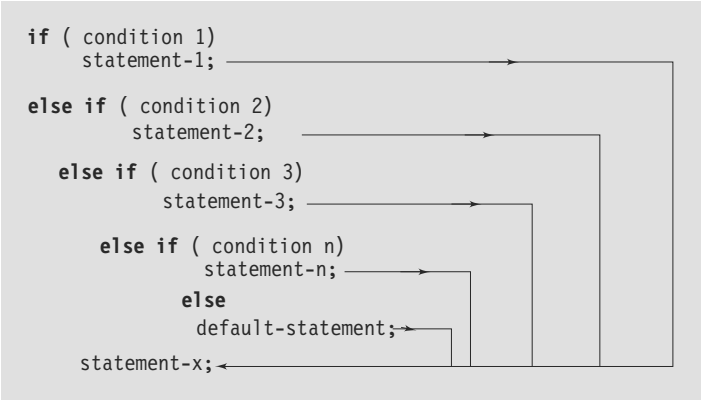


Fig. 4.55 **The else if ladder**

This construct is known as the `else if` ladder. The conditions are evaluated from the top (of the ladder) downwards. As soon as a true condition is found, the statement associated with it is executed and the control is transferred to the `statement-x` (skipping the rest of the ladder). When all the *n* conditions become false, then the final `else` containing the default-statement will be executed. Figure 4.56 shows the logic of execution of `else if` ladder statements.

Fig. 4.56 **Flow chart of else…if ladder.**

Let us consider an example given below:

```
____
____
if (code == 1)
     colour = "RED";
else if (code == 2)
       colour = "GREEN";
     else if (code == 3)
            colour = "WHITE";
          else
            colour = "YELLOW";
___
___
```

Here, code numbers other than 1, 2 or 3 are considered to represent YELLOW colour. The same results can be obtained by using nested if...else statements, as follows:

```
if (code != 1)
     if (code != 2)
       if (code != 3)
          colour = "YELLOW";
       else
          colour = "WHITE";
     else
          colour = "GREEN";
else
          colour = "RED";
```

In such situations, the choice is left to the programmer. However, in order to choose an `if` structure that is both effective and efficient, it is important that the programmer is fully aware of the various forms of an `if` statement and the rules governing their nesting.

Rules for Indentation When using control structures, a statement often controls many other following statements. In such situations, it is a good practice to use indentation to show that the indented statements are dependent on the preceding controlling statement. Some guidelines that could be followed while using indentation are listed as follows:

- Indent statements that are dependent on the previous statements; provide at least three spaces of indentation.
- Align vertically else clause with their matching if clause.
- Use braces on separate lines to identify a block of statements.
- Indent the statements in the block by at least three spaces to the right of the braces.
- Align the opening and closing braces.
- Use appropriate comments to signify the beginning and end of blocks.
- Indent the nested statements as per the above rules.
- Code only one clause or statement on each line.

4.21.2 The switch Statement

We have seen that when one of the many alternatives is to be selected, we can use an `if` statement to control the selection. However, the complexity of such a program increases dramatically when the number of alternatives increases. The program becomes difficult to read and follow. At times, it may confuse even the person who designed it. Fortunately, C has a built-in multiway decision statement known as a `switch`. The `switch` statement tests the value of a given variable (or expression) against a list of case values and when a match is found, a block of statements associated with that case is executed. The general form of the switch statement is shown further:

```
switch (expression)
{
  case value-1:
    block-1
    break;
  case value-2:
    block-2
    break;
  ......
  ......
  default:
    default-block
    break;
}
statement-x;
```

The expression is an integer expression or characters. value-1, value-2,…are constants or constant expressions (evaluable to an integral constant) and are known as case labels. Each of these values should be unique within a `switch` statement. block-1, block-2,…are statement lists and may contain zero or more statements. There is no need to put braces around these blocks. Note that case labels end with a colon (:).

When the `switch` is executed, the value of the expression is successfully compared against the values value-1, value-2,.... If a case is found whose value matches with the value of the expression, then the block of statements that follows the case are executed.

The break statement at the end of each block signals the end of a particular case and causes an exit from the `switch` statement, transferring the control to the `statement-x` following the `switch`.

The default is an optional case. When present, it will be executed if the value of the expression does not match with any of the case values. If not present, no action takes place if all matches fail and the control goes to the `statement-x`. (ANSI C permits the use of as many as 257 case labels)

The selection process of `switch` statement is illustrated in the flow chart shown in Fig. 4.57.

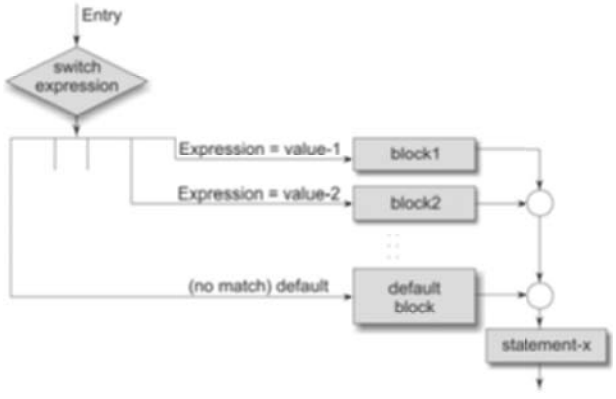


Fig. 4.57 **Selection process of the switch statement.**

The switch statement is often used for menu selection. For example:

```
____
____
printf(" TRAVEL GUIDE\n\n");
printf(" A Air Timings\n" );
printf(" T Train Timings\n");
printf(" B Bus Service\n" );
printf(" X To skip\n" );
printf("\n Enter your choice\n");
character = getchar();
switch (character)
{
     case 'A' :
               air-display();
               break;
     case 'B' :
               bus-display();
               break;
     case 'T' :
               train-display();
               break;
default :
               printf(" No choice\n");
}
     ____
     ____
```

It is possible to nest the `switch` statements. That is, a `switch` may be part of a case statement. ANSI C permits 15 levels of nesting.

Here are some key rules for using `switch` statement:

- The `switch` expression must be an integral type.
- Case labels must be constants or constant expressions.
- Case labels must be unique. No two labels can have the same value.
- Case labels must end with semicolon.
- The `break` statement transfers the control out of the `switch` statement.
- The `break` statement is optional. That is, two or more case labels may belong to the same statements.

- The default label is optional. If present, it will be executed when the expression does not find a matching case label.
- There can be at most one default label.
- The default may be placed anywhere but usually placed at the end.
- It is permitted to nest switch statements.

4.21.3 The goto Statement

So far we have discussed ways of controlling the flow of execution based on certain specified conditions. Like many other languages, C supports the `goto` statement to branch unconditionally from one point to another in the program. Although it may not be essential to use the `goto` statement in a highly structured language like C, there may be occasions when the use of `goto` might be desirable.

The `goto` requires a label in order to identify the place where the branch is to be made. A label is any valid variable name, and must be followed by a colon. The label is placed immediately before the statement where the control is to be transferred. The general forms of `goto` and label statements are shown in Fig. 4.58:

Fig. 4.58 **Use of goto statement**

The `label:` can be anywhere in the program either before or after the `goto label;` statement. During running of a program when a statement like `goto begin;` is met, the flow of control jumps to the statement immediately following the label begin. This happens unconditionally.

Note that a `goto` breaks the normal sequential execution of the program. If the `label:` is before the statement `goto label;` a loop will be formed and some statements will be executed repeatedly. Such a jump is known as a backward jump. On the other hand, if the label: is placed after the `goto label;`, some statements will be skipped and the jump is known as a forward jump.

A `goto` is often used at the end of a program to direct the control to go to the input statement, to read further data. Consider the following example:

```
main()
{
       double x, y;
```

```
        read:
        scanf("%f", &x);
        if (x < 0) goto read;
        y = sqrt(x);
        printf("%f %f\n", x, y);
        goto read;
}
```

This program is written to evaluate the square root of a series of numbers read from the terminal. The program uses two `goto` statements, one at the end, after printing the results to transfer the control back to the input statement, and the other to skip any further computation when the number is negative.

Due to the unconditional `goto` statement at the end, the control is always transferred back to the input statement. In fact, this program puts the computer in a permanent loop known as an infinite loop. The computer goes round and round until we take some special steps to terminate the loop. To avoid such situations, it is always advisable not to use `goto` statement.

EXAMPLE 4.22 *The program shown in Fig. 4.59 illustrates the use of `goto` statement.*

```
Program
     #include <math.h>
     main()
     {
          double x, y;
          int count;
          count = 1;
          printf("Enter FIVE real values in a LINE \n");
     read:
          scanf("%lf", &x);
          printf("\n");
          if (x < 0)
             printf("Value - %d is negative\n",count);
          else
          {
             y = sqrt(x);
             printf("%lf\t %lf\n", x, y);
          }
          count = count + 1;
```

```
            if (count <= 5)
      goto read;
            printf("\nEnd of computation");
      }
Output
      Enter FIVE real values in a LINE
      50.70 40 -36 75 11.25
      50.750000       7.123903
      40.000000       6.324555
      Value -3 is negative
      75.000000       8.660254
      11.250000       3.354102
      End of computation
```

Fig. 4.59 **Use of the goto statement**

The program in Fig. 4.59 evaluates the square root for five numbers. The variable count keeps the count of numbers read. When count is less than or equal to 5, goto read; directs the control to the label read; otherwise, the program prints a message and stops.

4.22 CASE STUDIES

1. Range of Numbers

Problem:

A survey of the computer market shows that personal computers are sold at varying costs by the vendors. The following is the list of costs (in hundreds) quoted by some vendors:

35.00, 40.50, 25.00, 31.25, 68.15,
47.00, 26.65, 29.00 53.45, 62.50

Determine the average cost and the range of values.

Problem analysis:

Range is one of the measures of dispersion used in statistical analysis of a series of values. The range of any series is the difference between the highest and the lowest values in the series. That is

Range = highest value – lowest value

It is therefore necessary to find the highest and the lowest values in the series.

Program: A program to determine the range of values and the average cost of a personal computer in the market is given in Fig. 4.60.

```
Program
main()
{
      int count;
      float value, high, low, sum, average, range;
      sum = 0;
      count = 0;
      printf("Enter numbers in a line : input a NEGATIVE number to end\n");
input:
         scanf("%f", &value);
         if (value < 0) goto output;
            count = count + 1;
         if (count == 1)
            high = low = value;
         else if (value > high)
               high = value;
            else if (value < low)
                  low = value;
         sum = sum + value;
         goto input;
output:
         average = sum/count;
         range = high - low;
         printf("\n\n");
         printf("Total values : %d\n", count);
         printf("Highest-value: %f\nLowest-value : %f\n",
                  high, low);
         printf("Range        : %f\nAverage : %f\n",
                  range, average);
      }
Output
      Enter numbers in a line : input a NEGATIVE number to end
      35 40.50 25 31.25 68.15 47 26.65 29 53.45 62.50 -1
      Total values : 10
      Highest-value: 68.150002
      Lowest-value : 25.000000
      Range: 43.150002
      Average : 41.849998
```

Fig. 4.60 Calculation of range of values

When the value is read the first time, it is assigned to two buckets, high and low, through the statement

```
high = low = value;
```

For subsequent values, the value read is compared with high; if it is larger, the value is assigned to high. Otherwise, the value is compared with low; if it is smaller, the value is assigned to low. Note that at a given point, the buckets high and low hold the highest and the lowest values read so far.

The values are read in an input loop created by the goto input; statement. The control is transferred out of the loop by inputting a negative number. This is caused by the following statement:

```
if (value < 0) goto output;
```

2. Pay-Bill Calculations

Problem:

A manufacturing company has classified its executives into four levels for the benefit of certain perks. The levels and corresponding perks are shown below:

Level	Perks	
	Conveyance allowance	Entertainment allowance
1	1000	500
2	750	200
3	500	100
4	250	–

An executive's gross salary includes basic pay, house rent allowance at 25% of basic pay and other perks. Income tax is withheld from the salary on a percentage basis as follows:

Gross salary	Tax rate
Gross <= 2000	No tax deduction
2000 < Gross <= 4000	3%
4000 < Gross <= 5000	5%
Gross > 5000	8%

Write a program that will read an executive's job number, level number, and basic pay and then compute the net salary after withholding income tax.

Problem analysis:

Gross salary = basic pay + house rent allowance + perks

Net salary = Gross salary – income tax.

The computation of perks depends on the level, while the income tax depends on the gross salary. The major steps are:

1. Read data.
2. Decide level number and calculate perks.
3. Calculate gross salary.
4. Calculate income tax.
5. Compute net salary.
6. Print the results.

Program: A program and the results of the test data are given in Fig. 4.61. Note that the last statement should be an executable statement. That is, the label stop: cannot be the last line.

```
Program
#define CA1 1000
#define CA2 750
#define CA3 500
#define CA4 250
#define EA1 500
#define EA2 200
#define EA3 100
#define EA4 0
main()
{
     int level, jobnumber;
     float gross,
        basic,
        house_rent,
        perks,
        net,
        incometax;
input:
        printf("\nEnter level, job number, and basic pay\n");
        printf("Enter 0 (zero) for level to END\n\n");
        scanf("%d", &level);
        if (level == 0) goto stop;
        scanf("%d %f", &jobnumber, &basic);
        switch (level)
        {
          case 1:
               perks = CA1 + EA1;
               break;
          case 2:
               perks = CA2 + EA2;
               break;
```

```
            case 3:
                 perks = CA3 + EA3;
                 break;
            case 4:
                 perks = CA4 + EA4;
                 break;
            default:
            printf("Error in level code\n");
            goto stop;
         }
         house_rent = 0.25 * basic;
         gross = basic + house_rent + perks;
         if (gross <= 2000)
            incometax = 0;
         else if (gross <= 4000)
              incometax = 0.03 * gross;
            else if (gross <= 5000)
               incometax = 0.05 * gross;
            else
               incometax = 0.08 * gross;
         net = gross - incometax;
         printf("%d %d %.2f\n", level, jobnumber, net);
         goto input;
         stop: printf("\n\nEND OF THE PROGRAM");
}
Output
      Enter level, job number, and basic pay
      Enter 0 (zero) for level to END
      1 1111 4000
      1 1111 5980.00
      Enter level, job number, and basic pay
      Enter 0 (zero) for level to END
      2 2222 3000
      2 2222 4465.00
      Enter level, job number, and basic pay
      Enter 0 (zero) for level to END
      3 3333 2000
      3 3333 3007.00
      Enter level, job number, and basic pay
      Enter 0 (zero) for level to END
      4 4444 1000
      4 4444 1500.00
      Enter level, job number, and basic pay
      Enter 0 (zero) for level to END
      0
      END OF THE PROGRAM
```

Fig. 4.61 **Pay-bill calculations**

4.23 DECISION MAKING AND LOOPING

Looping or iterative statements are used for running a particular set of code for any number of times. Iterative statements consist two parts—head and body. The head of the statement decides the number of times for which the instructions present in the body of the statement are to be executed.

In looping, a sequence of statements are executed until some conditions for termination of the loop are satisfied. Depending on the position of the control statement in the loop, a control structure may be classified either as the entry-controlled loop or as the exit-controlled loop.

The flow charts in Fig. 4.62 illustrate these structures. In the entry-controlled loop, the control conditions are tested before the start of the loop execution. If the conditions are not satisfied, then the body of the loop will not be executed. In the case of an exit-controlled loop, the test is performed at the end of the body of the loop and, therefore, the body is executed unconditionally for the first time. The entry-controlled and exit-controlled loops are also known as pre-test and post-test loops, respectively.

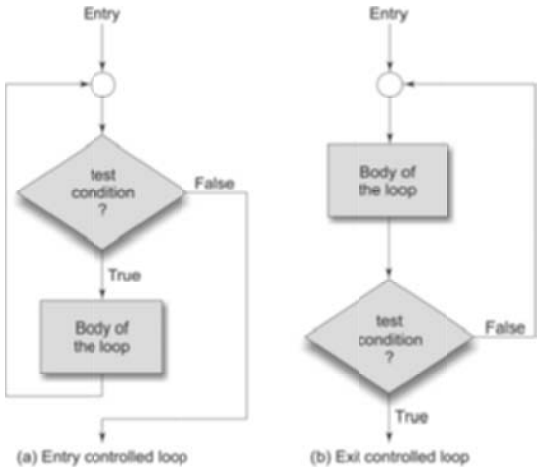


Fig. 4.62 Loop control structures

The test conditions should be carefully stated in order to perform the desired number of loop executions. It is assumed that the test condition will eventually transfer the control out of the loop. In case, due to some reason it does not do so, the control sets up an infinite loop and the body is executed over and over again.

A looping process, in general, would include the following four steps:

1. Setting and initialisation of a condition variable.
2. Execution of the statements in the loop.
3. Test for a specified value of the condition variable for execution of the loop.
4. Incrementing or updating the condition variable.

The test may be either to determine whether the loop has been repeated the specified number of times or to determine whether a particular condition has been met.

The C language provides for three constructs for performing loop operations. They are:

1. The `while` statement.
2. The `do` statement.
3. The `for` statement.

4.23.1 The while Statement

The simplest of all the looping structures in C is the `while` statement. We have used `while` in many of our earlier programs. The basic format of the `while` statement is

```
while (test condition)
{
body of the loop
}
```

The `while` is an entry-controlled loop statement. The test-condition is evaluated and if the condition is true, then the body of the loop is executed. After execution of the body, the test-condition is once again evaluated and if it is true, the body is executed once again. This process of repeated execution of the body continues until the test-condition finally becomes false and the control is transferred out of the loop. On exit, the program continues with the statement immediately after the body of the loop.

The body of the loop may have one or more statements. The braces are needed only if the body contains two or more statements. However, it is a good practice to use braces even if the body has only one statement.

An example of `while` statement, which uses the keyboard input, is shown as here:

```
=========
character = ' ' ;
while (character != 'Y')
     character = getchar();
xxxxxxx;
=========
```

First the character is initialized to ' '. The `while` statement then begins by testing whether character is not equal to *Y*. Since the character was initialized to ' ', the test is true and the loop statement character = `getchar();` is executed. Each time a letter is keyed in, the test is carried out and the loop statement is executed until the letter *Y* is pressed.

When Y is pressed, the condition becomes false because character equals Y, and the loop terminates, thus transferring the control to the statement xxxxxxx;.

EXAMPLE 4.23 *A program to evaluate the equation $y = x^n$, where n is a non-negative integer, is given in Fig. 4.63.*

```
Program
     main()
     {
        int count, n;
        float x, y;
        printf("Enter the values of x and n : ");
        scanf("%f %d", &x, &n);
        y = 1.0;
        count = 1;         /* Initialisation */
        /* LOOP BEGINs */
        while ( count <= n)  /* Testing */
        {
           y = y*x;
           count++;              /* Incrementing */
        }
        /* END OF LOOP */
        printf("\nx = %f; n = %d; x to power n = %f\n",x,n,y);
     }
Output
     Enter the values of x and n : 2.5 4
     x = 2.500000; n = 4; x to power n = 39.062500
     Enter the values of x and n : 0.5 4
     x = 0.500000; n = 4; x to power n = 0.062500
```

Fig. 4.63 Program to compute x to the power n using while loop

In this program, the variable *y* is initialised to 1 and then multiplied by *x*, *n* times using the `while` loop. The loop control variable `count` is initialised outside the loop and incremented inside the loop. When the value of `count` becomes greater than *n*, the control exists the loop.

4.23.2 The do Statement

The while loop construct, which we have discussed in the previous section, makes a test of condition before the loop is executed. Therefore, the body of the loop may not be executed at all if the condition is not satisfied at the very first attempt. On some occasions, it might be necessary to execute the body of the loop before the test is performed. Such situations can be handled with the help of the do statement. This takes the following form:

```
do
{
     body of the loop
}
while (test-condition);
```

On reaching the `do` statement, the program proceeds to evaluate the body of the loop first. At the end of the loop, the `test-condition` in the `while` statement is evaluated. If the condition is true, the program continues to evaluate the body of the loop once again. This process continues as long as the condition is true. When the condition becomes false, the loop will be terminated and the control goes to the statement that appears immediately after the `while` statement.

Since the `test-condition` is evaluated at the bottom of the loop, the `do...while` construct provides an exit-controlled loop and, therefore, the body of the loop is always executed at least once.

Consider the following example:

```
_______
I = 1;
/*Initializing */
sum = 0;
do
{
     sum = sum + I;
     I = I+2;                    /* Incrementing */
}
while(sum < 40 || I < 10);       /* Testing */
printf("%d %d\n", I, sum);
_______
```

This loop will be executed as long as one of the two relations is true.

EXAMPLE 4.24 *A program to print the multiplication table from 1 ´* 1 to 12 ´* 10 is given in Fig. 4.64.*

```
     Program:
        #define COLMAX 10
        #define ROWMAX 12
        main()
        {
```

```
            int row,column, y;
            row = 1;
            printf(" MULTIPLICATION TABLE \n");
            do /*......OUTER LOOP BEGINS........*/
            {
                  column = 1;
                  do /*.......INNER LOOP BEGINS.......*/
                  {
                     y = row * column;
                     printf("%4d", y);
                     column = column + 1;
            }

            while (column <= COLMAX); /*... INNER LOOP ENDS ...*/
            printf("\n");
            row = row + 1;
            }
            while (row <= ROWMAX);/*..... OUTER LOOP ENDS .....*/
      }
   Output
                MULTIPLICATION TABLE
1    2     3     4     5     6     7     8     9     10
2    4     6     8     10    12    14    16    18    20
3    6     9     12    15    18    21    24    27    30
4    8     12    16    20    24    28    32    36    40
5    10    15    20    25    30    35    40    45    50
6    12    18    24    30    36    42    48    54    60
7    14    21    28    35    42    49    56    63    70
8    16    24    32    40    48    56    64    72    80
9    18    27    36    45    54    63    72    81    90
10   20    30    40    50    60    70    80    90    100
11   22    33    44    55    66    77    88    99    110
12   24    36    48    60    72    84    96    108   120
```

Fig. 4.64 Printing of a multiplication table using do...while loop.

Here, the program contains two `do…while` loops in nested form. The outer loop is controlled by the variable row and executed 12 times. The inner loop is controlled by the variable column and is executed 10 times, each time the outer loop is executed. That is, the inner loop is executed a total of 120 times, each time printing a value in the table.

Notice that the printf of the inner loop does not contain any new line character (\n). This allows the printing of all row values in one line. The empty printf in the outer loop initiates a new line to print the next row.

4.23.3 The for Statement

Simple 'for' Loops The for loop is another entry-controlled loop that provides a more concise loop control structure. The general form of the for loop is

```
for ( initialization ; test-condition ; increment)
{
    body of the loop
}
```

The execution of the for statement is as follows:

1. Initialisation of the control variables is done first, using assignment statements such as $i = 1$ and *count* = 0. The variables *i* and *count* are known as loop-control variables.
2. The value of the control variable is tested using the test-condition. The test-condition is a relational expression, such as $i < 10$ that determines when the loop will exit. If the condition is true, the body of the loop is executed; otherwise the loop is terminated and the execution continues with the statement that immediately follows the loop.
3. When the body of the loop is executed, the control is transferred back to the for statement after evaluating the last statement in the loop. Now, the control variable is incremented using an assignment statement such as $i = i+1$ and the new value of the control variable is again tested to see whether it satisfies the loop condition. If the condition is satisfied, the body of the loop is again executed. This process continues till the value of the control variable fails to satisfy the test-condition.

Consider the following segment of a program:

```
for ( x = 0 ; x <= 9 ; x = x+1)
{
     printf("%d", x);
}
printf("\n");
```

This for loop is executed 10 times and prints the digits 0 to 9 in one line. The three sections enclosed within parentheses must be separated by semicolons. Note that there is no semicolon at the end of the increment section, $x = x+1$.

The for statement allows for negative increments as well. For example, the loop discussed earlier can be written as follows:

```
for ( x = 9 ; x >= 0 ; x = x–1 )
     printf("%d", x);
printf("\n");
```

This loop is also executed 10 times, but the output would be from 9 to 0 instead of 0 to 9. Note that braces are optional when the body of the loop contains only one statement.

Since the conditional test is always performed at the beginning of the loop, the body of the loop may not be executed at all, if the condition fails at the start. For example, the following loop will never be executed because the test condition fails at the very beginning itself.

```
for (x = 9; x < 9; x = x-1)
     printf("%d", x);
```

One of the important points about the `for` loop is that all the three actions, namely initialisation, testing and incrementing, are placed in the for statement itself, thus making them visible to the programmers and users, in one place. The for statement and its equivalent of while and do statements are shown in Table 4.24.

TABLE 4.24 Comparison of the three loops

for	while	do
for (*n*=1; *n*<=10; ++*n*)	*n* = 1;	*n* = 1;
{	while (*n*<=10)	do
————	{	{
————	————	————
}	————	————
	n = *n*+1;	*n* = *n*+1;
	}	}
		while(*n*<=10);

EXAMPLE 4.25 *The program in Fig. 4.65 uses a* `for` *loop to print the "Powers of 2" table for the power 0 to 20, both positive and negative.*

```
Program
     main()
     {
        long int p;
        int n;
        double q;
        printf("--------------------------------\n");
```

```
        printf(" 2 to power n n 2 to power -n\n");
        printf("---------------------------------\n");
        p = 1;
        for (n = 0; n < 21 ; ++n) /* LOOP BEGINS */
        {
           if (n == 0)
              p = 1;
           else
              p = p * 2;
           q = 1.0/(double)p ;
           printf("%10ld %10d %20.12lf\n", p, n, q);
        }                                /* LOOP ENDS */
        printf("---------------------------------\n");
     }
Output
     ----------------------------------------
        2 to power n    n       2 to power -n
     ----------------------------------------
              1         0      1.000000000000
              2         1      0.500000000000
              4         2      0.250000000000
              8         3      0.125000000000
             16         4      0.062500000000
             32         5      0.031250000000
             64         6      0.015625000000
            128         7      0.007812500000
            256         8      0.003906250000
            512         9      0.001953125000
           1024        10      0.000976562500
           2048        11      0.000488281250
           4096        12      0.000244140625
           8192        13      0.000122070313
          16384        14      0.000061035156
          32768        15      0.000030517578
          65536        16      0.000015258789
         131072        17      0.000007629395
         262144        18      0.000003814697
         524288        19      0.000001907349
        1048576        20      0.000000953674
     ----------------------------------------
```

Fig. 4.65 **Program to print 'Power of 2' table using for loop**

Additional Features of for Loop The for loop in C has several capabilities that are not found in other loop constructs. For example, more than one variable can be initialised at a time in the for statement, as shown further:

```
for (p=1, n=0; n<17; ++n)
```

Note that the initialisation section has two parts $p = 1$ and $n = 1$ separated by a comma.

Like the initialisation section, the increment section may also have more than one part. The multiple arguments in the increment section are separated by commas. For example:

```
for (n=1, m=50; n<=m; n=n+1, m=m-1)
{
     p = m/n;
     printf("%d %d %d\n", n, m, p);
}
```

Further, the `test-condition` may have any compound relation and the testing need not be limited only to the loop control variable. For example,

```
sum = 0;
for (i = 1; i < 20 && sum < 100; ++i)
{
     sum = sum+i;
     printf("%d %d\n", i, sum);
}
```

Here, the loop is executed as long as both the conditions $i < 20$ and `sum` < 100 are true. The `sum` is evaluated inside the loop.

It is also permissible to use expressions in the assignment statements of initialisation and increment sections. For example,

```
for (x = (m+n)/2; x > 0; x = x/2)
```

Another unique aspect of `for` loop is that one or more sections can be omitted, if necessary. For instance, the following `for` declaration is perfectly valid:

```
———————
m = 5;
for ( ; m != 100 ; )
{
     printf("%d\n", m);
     m = m+5;
}
———————
```

Here, both the initialisation and increment sections are omitted in the `for` statement. The initialisation has been done before the `for` statement and the control variable is incremented inside the loop. In such cases, the sections are left 'blank'. However, the semicolons separating the sections must remain. If the `test-condition` is not present, the `for` statement sets up an 'infinite' loop. Such loops can be broken using `break` or `goto` statements in the loop.

We can also set up time delay loops using the null statement as follows:

```
For  ( j = 1000; j > 0; j = j-1)
     ;
```

This loop is executed 1000 times without producing any output; it simply causes a time delay. Notice that the body of the loop contains only a semicolon, known as a null statement.

Nesting of for Loops Nesting of loops, that is, one for statement within another for statement, is allowed in C. For example, two loops can be nested as shown in Fig. 4.66:

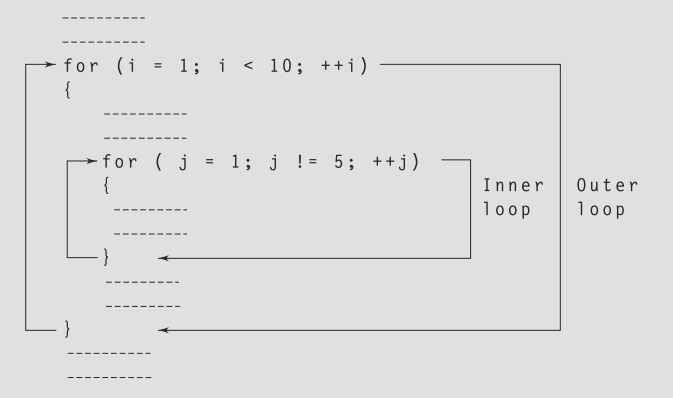


Fig. 4.66 **Nesting of for loops**

The nesting may continue up to any desired level. The loops should be properly indented so to enable the reader to easily determine which statements are contained within each for statement. (ANSI C allows up to 15 levels of nesting. However, some compilers permit more.)

A typical example of nesting of `for` loops is the program to print the multiplication table, as shown further:

```
for (row = 1; row <= ROWMAX ; ++row)
{
     for (column = 1; column <= COLMAX ; ++column)
     {
```

```
            y = row * column;
            printf("%4d", y);
        }
        printf("\n");
}
```

Here, the outer loop controls the rows while the inner loop controls the columns.

4.23.4 Jumps in Loops

Loops perform a set of operations repeatedly until the control variable fails to satisfy the test-condition. The number of times a loop is repeated is decided in advance and the test condition is written to achieve this. Sometimes, when executing a loop, it becomes desirable to skip a part of the loop or to leave the loop as soon as a certain condition occurs. For example, consider the case of searching for a particular name in a list containing, say, 100 names. A program loop written for reading and testing the names 100 times must be terminated as soon as the desired name is found. C permits a jump from one statement to another within a loop as well as a jump out of a loop.

Jumping Out of a Loop An early exit from a loop can be accomplished by using the break statement or the goto statement. We have already seen the use of the break in the switch statement and the goto in the if...else construct. These statements can also be used within while, do or for loops. The use of break in loops is illustrated in Fig. 4.67.

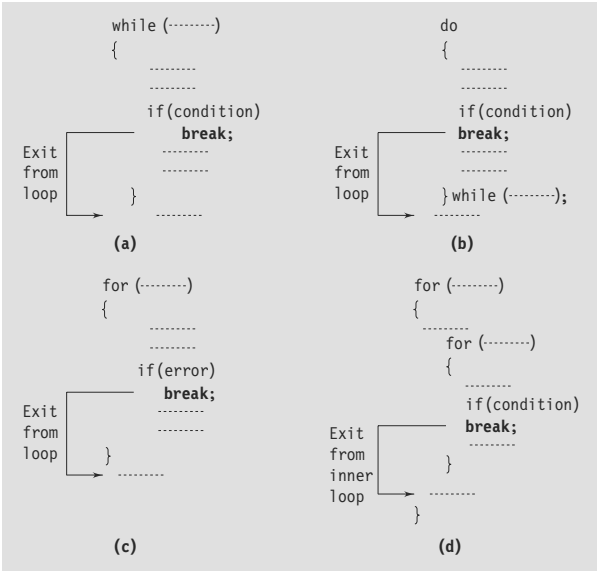


Fig. 4.67 Exiting a loop with break statement

When a `break` statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop. When the loops are nested, the `break` would only exit from the loop containing it. That is, the `break` will exit only a single loop.

Figure 4.68 shows the flowchart illustrating the use of break statement in a loop.

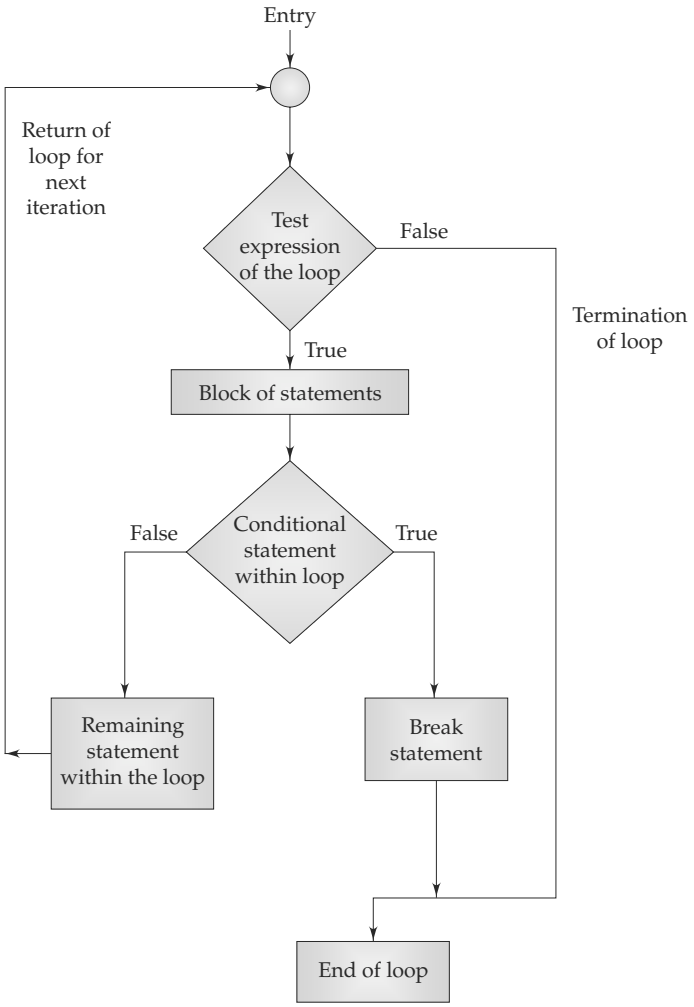


Fig. 4.68 **Flowchart for exiting a loop with break statement**

Since a `goto` statement can transfer the control to any place in a program, it is useful to provide branching within a loop. Another important use of `goto` is to exit from deeply nested loops when an error occurs. A simple `break` statement would not work here. Figure 4.69 shows the use of `goto` statement for jumping within or outside of a loop.

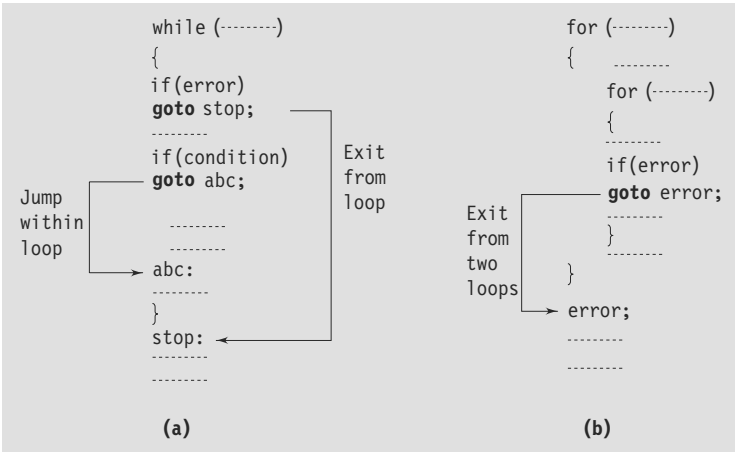


Fig. 4.69 Jumping within and exiting from the loops with goto statement

Figure 4.70 shows the flowchart illustrating the use of goto statement in a loop.

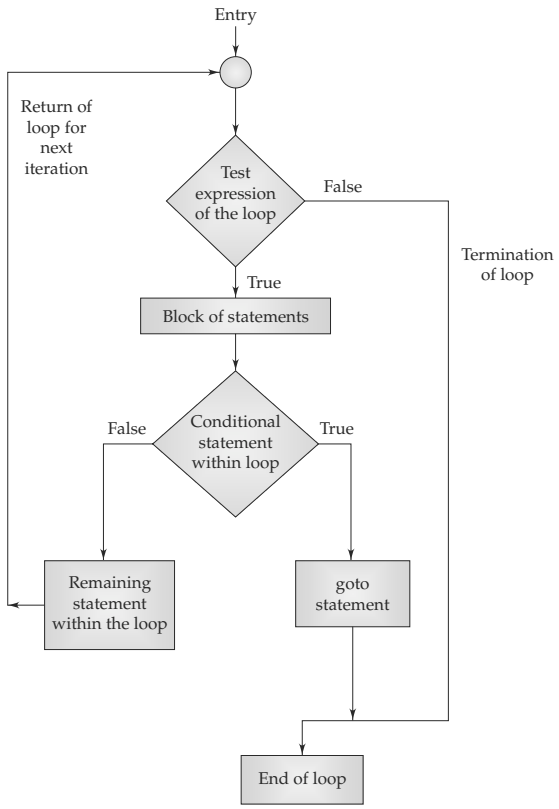


Fig. 4.70 Flowchart for using goto statement in a loop

EXAMPLE 4.26 *The program in Fig. 4.71 illustrates the use of the break statement in a C program.*

```
Program
     main()
     {
        int m;
        float x, sum, average;
        printf("This  program computes the average of a
           set of numbers\n");
        printf("Enter values one after another\n");
        printf("Enter a NEGATIVE number at the end.\n\n");
        sum = 0;
        for (m = 1 ; m < = 1000 ; ++m)
        {
             scanf("%f", &x);
             if (x < 0)
                break;
             sum += x ;
        }
        average = sum/(float)(m-1);
        printf("\n");

        printf("Number of values   = %d\n", m-1);
        printf("Sum                   = %f\n", sum);
        printf("Average            = %f\n", average);
     }
Output
     This program computes the average of a set of numbers
     Enter values one after another
     Enter a NEGATIVE number at the end.

     21 23 24 22 26 22 -1

     Number of values   = 6
     Sum                = 138.000000
     Average            = 23.000000
```

Fig. 4.71 **Use of break in a program.**

The program reads in Fig. 4.71 a list of positive values and calculates their average. The `for` loop is written to read 1000 values. However, if we want the program to calculate the average of any set of values less than 1000, then we must enter a 'negative' number after the last value in the list, to mark the end of input.

Each value, when it is read, is tested to see whether it is a positive number or not. If it is positive, the value is added to the sum; otherwise, the loop terminates. On exit, the average of the values read is calculated and the results are printed out.

Skipping a Part of a Loop During the loop operations, it may be necessary to skip a part of the body of the loop under certain conditions. For example, in processing of applications for some job, we might like to exclude the processing of data of applicants belonging to a certain category. On reading the category code of an applicant, a test is made to see whether his application should be considered or not. If it is not to be considered, the part of the program loop that processes the application details is skipped and the execution continues with the next loop operation.

Like the `break` statement, C supports another similar statement called the `continue` statement. However, unlike the `break` that causes the loop to be terminated, the `continue`, as the name implies, causes the loop to be continued with the next iteration after skipping any statements in between. The `continue` statement tells the compiler, "SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT ITERATION". The format of the `continue` statement is simple.

```
continue;
```

The use of the `continue` statement in loops is illustrated in Fig. 4.72.

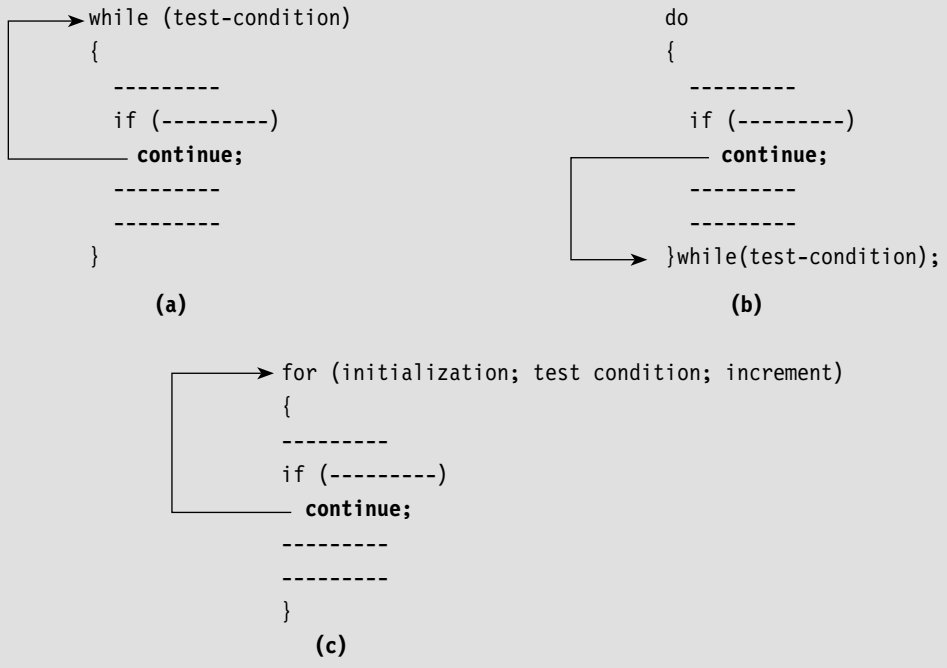


Fig. 4.72 Bypassing and continuing in loops.

EXAMPLE 4.27 *The program shown in Fig. 4.73 illustrates the use of continue statement.*

```
Program:
#include<stdio.h>
#include<conio.h>
void main()
{
int x,y;
clrscr();
for(x=1;x<=3;x++)
{
        for(y=1;y<=3;y++)
        {
        if(x==y)
           continue;
        printf("\n%d\t%d\t",x,y);
        }
}
getch();
}

Output:
1    2
1    3
2    1
2    3
3    1
3    2
```

Fig. 4.73 Use of continue statement

In this above code, two variables, x and y, are taken as an integer data type. Here, when the value of x is equal to y, the control of the program is transferred to the inner for loop using the continue statement.

4.24 JUMPING OUT OF THE PROGRAM

We have just seen that we can jump out of a loop using either the `break` statement or `goto` statement. In a similar way, we can jump out of a program by using the library function `exit( )`. In case, due to some reason, we wish to break out of a program and return to the operating system, we can use the `exit( )` function, as shown follows:

```
........
........
if (test-condition)
exit(0);
........
........
```

The `exit( )` function takes an integer value as its argument. Normally zero is used to indicate normal termination and a nonzero value to indicate termination due to some error or abnormal condition. The use of `exit( )` function requires the inclusion of the header file <stdlib.h>.

4.25 STRUCTURED PROGRAMMING

Structured programming is an approach to the design and development of programs. It is a discipline of making a program's logic easy to understand by using only the basic three control structures:

- Sequence (straight line) structure
- Selection (branching) structure
- Repetition (looping) structure

While sequence and loop structures are sufficient to meet all the requirements of programming, the selection structure proves to be more convenient in some situations. The use of structured programming techniques helps ensure well-designed programs that are easier to write, read, debug and maintain compared to those that are unstructured.

Structured programming discourages the implementation of unconditional branching using jump statements such as goto, break and continue. In its purest form, structured programming is synonymous with "goto less programming". Do not go to goto statement!

4.26 CASE STUDIES

1. Table of Binomial Coefficients

Problem:

Binomial coefficients are used in the study of binomial distributions and reliability of multicomponent redundant systems. It is given by

$$B(m,x) = \binom{m}{x} = \frac{m!}{x!\,(m-x)!},\ m >= x$$

A table of binomial coefficients is required to determine the binomial coefficient for any set of m and x.

Problem Analysis:

The binomial coefficient can be recursively calculated as follows:

$B(m,o) = 1$

$$B(m,x) = B(m,x-1)\left[\frac{m-x+1}{x}\right],\ x = 1,2,3,\ldots,m$$

Further,

$B(o,o) = 1$

That is, the binomial coefficient is one when either x is zero or m is zero. The program in Fig. 4.74 prints the table of binomial coefficients for m = 10. The program employs one do loop and one while loop.

```
Program
#define MAX 10
main()
{
      int m, x, binom;
      printf(" m x");
      for (m = 0; m <= 10 ; ++m)
          printf("%4d", m);
      printf("\n———————————————————\n");
      m = 0;
      do
      {
         printf("%2d ", m);
         x = 0; binom = 1;
         while (x <= m)
         {
            if(m == 0 || x == 0)
            printf("%4d", binom);
         else
         {
            binom = binom * (m - x + 1)/x;
            printf("%4d", binom);
         }
         x = x + 1;
         }
         printf("\n");
         m = m + 1;
         }
         while (m <= MAX);
```

```
        printf("————————————————————\n");
}
Output
     mx   0    1    2    3    4    5    6    7    8    9    10
  ------------------------------------------------------------------
     0    1
     1    1    1
     2    1    2    1
     3    1    3    3    1
     4    1    4    6    4    1
     5    1    5    10   10   5    1
     6    1    6    15   20   15   6    1
     7    1    7    21   35   35   21   7    1
     8    1    8    28   56   70   56   28   8    1
     9    1    9    36   84   126  126  84   36   9    1
     10   1    10   45   120  210  252  210  120  45   10   1
--------------------------------------------------------------------
```

Fig. 4.74 **Program to print binomial coefficient table**

2. Minimum Cost

Problem:

The cost of operation of a unit consists of two components C1 and C2 which can be expressed as functions of a parameter p as follows:

C1 = 30 – 8p
C2 = 10 + p2

The parameter p ranges from 0 to 10. Determine the value of p with an accuracy of + 0.1 where the cost of operation would be minimum.

Problem Analysis:

$$\text{Total cost} = C1 + C2 = 40 - 8p + p2$$

The cost is 40 when p = 0, and 33 when p = 1 and 60 when p = 10. The cost, therefore, decreases first and then increases. The program in Fig. 4.75 evaluates the cost at successive intervals of p (in steps of 0.1) and stops when the cost begins to increase. The program employs break and continue statements to exit the loop.

```
Program
main()
{
     float p, cost, p1, cost1;
```

```
      for (p = 0; p <= 10; p = p + 0.1)
      {
         cost = 40 - 8 * p + p * p;
         if(p == 0)
         {
         cost1 = cost;
            continue;
         }
         if (cost >= cost1)
            break;
         cost1 = cost;
            p1 = p;
      }
      p = (p + p1)/2.0;
      cost = 40 - 8 * p + p * p;
      printf("\nMINIMUM COST = %.2f AT p = %.1f\n", cost, p);
}
Output
      MINIMUM COST = 24.00 AT p = 4.0
```

Fig. 4.75 **Program of minimum cost problem**

SUMMARY

C is the most popularly used procedure-oriented programming language. To write programs in C, we need to understand the various terms such as characters, identifiers, keywords, constants, variables, data types, etc. Characters include alphabets, numbers and some special characters such as @, % and &. The combination of the characters forms identifiers. Variables are the elements whose values can be changed during the execution of a program. On the other hand, constants are the values that remain unchanged during the execution of a program.

The keywords are the reserved words that cannot be used as a variable name or as a function name. Each keyword has a specific meaning and functionality of its own. For instance, the data types specify the type of a data entity, such as integer, float or character.

To perform specific operations, we can make use of different operators available in C. Following are the various categories of operators:

- **Arithmetic operators:** These operators are used to perform arithmetic operations on data and variables. Arithmetic operators are of two types, unary and binary. Examples: +, -, *, ++, --.
- **Relational operators:** These operators are used to carry out the comparison between different operands. Relational operators help in decision-making process in C. Examples: <, >, ==, !=.

- **Logical operators:** These operators are used to perform logical AND, OR and NOT operations on data. Logical operators are generally used to compare Boolean expressions. Examples: &&, ||, !.
- **Assignment operators:** These operators are used to evaluate an expression and assign the value present at the right-hand side of the expression to the left-hand side of the expression. Examples: =, +=, -=, *=.
- **Conditional operators:** These operators are used to perform certain conditional tests on the expressions. These operators operate on three expressions and contain two symbols, ? and :.
- **Bitwise operators:** These operators are used for testing, shifting and complementing data bits.

C language also supports several control structures that are responsible for controlling the execution of various instructions included in a program. The various control structures are:

- **Branching control structures:** These structures are used as decision-making statements whether to execute or skip a particular block of statements. Examples: `if`, `if-else`, `switch`.
- **Looping control structures:** These structures are used for repeatedly executing a block of statements for a particular number of iterations governed by a control variable. Examples: `while`, `do`, `for`.

POINTS TO REMEMBER

- **Function**: It is a subroutine that may include one or more statements designed to perform a specific task.
- **Global variable**: It is a variable that can be used in more than one function.
- **Function body**: It is a part of a program that contains all the statements between the two braces, i.e. { and }.
- **Newline character**: It instructs the computer to go to the next (new) line.
- **Arguments**: Arguments are the values that are passed to a function as input.
- **Program**: A program contains a sequence of instructions written to perform a specific task.
- **Identifiers**: These are the names of variables, functions and arrays.
- **Constant**: It is a fixed value that does not change during the execution of a program.
- **String constant**: It is a sequence of characters enclosed in double quotes that represents a text string.
- **Variable**: It is a data name that may be used to store a data value.
- **Information**: The processed data generated by a program is called information.
- **Formatted input**: It refers to the input data that have been arranged in a particular format.
- **Control string**: It contains field specifications, which direct the interpretation of input data. It is also known as format string.
- **Formatted output**: It refers to the generated output that has been arranged in a particular format.
- **printf**: It is a function used to print and display output of a program.
- **scanf**: It is a function used to read values entered by the user upon execution of a program.

- **Operator**: It is a symbol that tells the computer to perform certain mathematical or logical computations.
- **Expression**: It is a sequence of operands and operators that reduces to a single value.
- **Integer expression**: When both the operands in a single arithmetic expression are integers, then that expression is termed as integer expression.
- **Real arithmetic**: An arithmetic operation involving only real operands is known as real arithmetic.
- **Relational operators**: These operators are used for making comparisons between two expressions.
- **Logical operators**: These operators are used for testing more than one condition and making decisions.
- **Assignment operators**: These operators are used for assigning the result of an expression to a variable.
- **Bitwise operators**: These operators are used for testing the bits, or shifting them right or left.
- **Comma operator**: It is used to link the related expressions together.
- **sizeof operator**: It is a compile time operator and when used with an operand, it returns the number of bytes the operand occupies.
- **Arithmetic expressions**: It is a combination of variables, constants and operators arranged as per the syntax of the language.
- **Decision-making statements**: These statements control the flow of execution and make decisions to see whether a particular condition has occurred or not.
- **switch statement**: It is built-in multiway decision statement used for testing the value of a given variable against a list of case values.
- **Conditional operator**: It is an operator comprising three operands that is used for making two-way decisions.
- **goto statement**: It is a statement used to transfer the flow of execution unconditionally from one point to another in a program.
- **Program loop**: It consists of two segments, one known as the body of the loop and the other known as the control statement. On the basis of the control statement, the body of the loop is executed repeatedly.
- **Control statement**: It tests certain conditions and then directs the repeated execution of the statements contained in the body of the loop.
- **Infinite loop**: It is a permanent loop in which the body is executed over and over again.
- **while statement**: It is an entry-controlled loop statement in which the test-condition is evaluated first and if the condition is true, then the body of the loop is executed.
- **do statement**: It executes the body of the loop before the test is performed.
- **continue statement**: It causes the loop to be continued with the next iteration after skipping further statements in the current iteration.
- **break statement**: It causes the loop to be terminated in which it is enclosed.

REVIEW QUESTIONS

1. Every line in a C program should end with a semicolon.
2. Every C program ends with an END word.
3. main( ) is where the program begins its execution.
4. A line in a program may have more than one statement.
5. The closing brace of the main( ) in a program is the logical end of the program.
6. The purpose of the header file such as stdio.h is to store the source code of a program.
7. Comments cause the computer to print the text enclosed between /* and */ when executed.
8. Syntax errors will be detected by the compiler.
9. Any valid printable ASCII character can be used in an identifier.
10. All variables must be given a type when they are declared.
11. Variable declarations can appear anywhere in a program.
12. ANSI C treats the variables name and Name to be same.
13. Floating point constants, by default, denote float type values.
14. Like variables, constants have a type.
15. Character constants are coded using double quotes.
16. All arithmetic operators have the same level of precedence.
17. A unary expression consists only one operand with no operators.
18. Associativity is used to decide which of several different expressions is evaluated first.
19. An expression statement is terminated with a period.
20. `If` is an iterative control structure.
21. The C standard function that receives a single character from the keyboard is `getchar`.
22. The `scanf` function cannot be used to read a single character from the keyboard.
23. A program stops its execution when a `break` statement is encountered.
24. The `do…while` statement first executes the loop body and then evaluate the loop control expression.
25. An exit-controlled loop is executed a minimum of one time.
26. The three loop expressions used in a for loop header must be separated by commas.

1. Every program statement in a C program must end with a _____
2. The _____ function is used to display the output on the screen.

3. The _____ header file contains mathematical functions.
4. The escape sequence character _____ causes the cursor to move to the next line on the screen.
5. _____ is the largest value that an unsigned `short int` type variable can store.
6. A global variable is also known as _____ variable.
7. A variable can be made constant by declaring it with the qualifier _____ at the time of initialisation.
8. The _____ operator is true only when both the operands are true.
9. _____ operators are used for testing the bits, or shifting them right or left.
10. The _____ statement when executed in a `switch` statement causes immediate exit from the structure.
11. The expression ! (*x* ! = *y*) can be replaced by the expression _____
12. The _____ operator returns the number of bytes the operand occupies.
13. The order of evaluation can be changed by using _____ in an expression.
14. _____ is used to determine the order in which different operators in an expression are evaluated.
15. In `do-while` loop, the body is executed at least _____ number of time.
16. The _____ statement is used to skip the remaining part of the statements in a loop.
17. A `for` loop with the no test condition is known as _____ loop.
18. _____ should be avoided as part of structured programming approach.
19. n++ is equivalent to the expression _____.

1. In which year C language was developed?
   A. 1951 B. 1962
   C. 1972 D. 1947
2. C is a _____.
   A. Structured language B. Object-oriented language
   C. Machine language D. Assembly language
3. The execution of a program written in a C language begins at:
   A. main( ) B. scanf( )
   C. printf( ) D. #include<iostream.h>
4. Which one of the following is an example of a valid file name in C language?
   A. system B. system.c
   C. system.cpp D. system.java

5. Which of the following is the correct syntax for the printf statement?
   A. printf('Hello world'); B. printf("Hello world")
   C. printf("Hello world"); D. printf{'Hello world'};
6. Which of the following is an intermediary file generated during the execution of a C program?
   A. .c B. .obj
   C. .exe D. .bak
7. Which of the following will run successfully in C?
   A.
```
main()
{
printf(" Hello world");
}
```
   B.
```
main()
{ printf(" Hello world"); }
```
   C.
```
main() {printf(" Hello world");}
```
   D. All of the above
8. Which of the following is the correct form of writing comments?
   A. /* comment */ B. /* comment /*
   C. */ comment /* D. */comment */
9. Which one of the following is not a real constant?
   A. 15.25 B. 0.962
   C. 10 D. +24.85
10. Which one of the following is a string constant?
   A. '5' B. "hello"
   C. 25 D. None of the above
11. Which one of the following does not represent a variable?
   A. % B. height
   C. xy1 D. m_width
12. The range of values for a char data type is
   A. -128 to 127 B. 3.4e-38 to 3.4e+38
   C. 1.7e-308 to 1.7e+308 D. -32,768 to 32,768
13. Which one of the following is a floating-point data type?
   A. int B. long int
   C. double D. None of the above
14. Which command is used for reading data from keyboard in C language?
   A. `cout` B. `cin`
   C. `printf ( )` D. `scanf ( )`

15. Which one of the following commands is used for the purpose of displaying output in C language?
    A. `scanf ( )` B. `printf ( )`
    C. `system.out.println ( )` D. `None of the above`
16. Which of the following is the trigraph sequence for representing #?
    A. ??= B. ??-
    C. ??(D. ??>
17. Which of the following is a legible variable name?
    A. income_tax B. income-tax
    C. income.tax D. income,tax
18. Which of the following is not a string?
    A. "S" B. 'S'
    C. "\"S\"" D. All of the above are strings
19. Which of the following cannot be used for declaring integer type variables?
    A. `int` B. `short int`
    C. `long int` D. `double int`
20. Which of the following will not read an integer number?
    A. %d B. %f
    C. %ld D. %c
21. Which of the following is not an arithmetic operator?
    A. + B. -
    C. * D. &
22. Which of the following operators are used for obtaining the remainder in a division operation?
    A. / B. %
    C. ! D. None of the above
23. Which of the following is the correct statement for computing logical AND?
    A. a<b & x>y B. a<b && x>y
    C. a<b AND x>y D. None of the above
24. *a* +=1 will result in:
    A. *a* = *a* + *a* B. *a* = *a*+1
    C. *a* = 1+1 D. *a* = *a* + (*a*+1)
25. Which of the following is the bitwise left shift operator?
    A. < B. <<
    C. <<< D. ^

ANSWERS

True or False

1. T	2. False	3. False	4. True	5. True	6. True	7. False	8. False
9. True	10. False	11. True	12. False	13. False	14. True	14. True	15. False
16. False	17. False	18. True	19. False	20. False	21. True	22. False	23. False
24. True	25. True	26. True					

Fill in the Blanks

1. semicolon	2. printf()	3. math.h	4. newline or '\n'
5. 255	6. external	7. const	8. AND
9. bitwise	10. break	11. x ==y	12. sizeof
13. parentheses	14. precedence	15. one	16. continue
17. infinite	18. goto	19. n=n+1	

Multiple Choice Questions

1. C	2. A	3. A	4. B	5. C	6. B	7. D	8. A
9. C	10. B	11. A	12. A	13. C	14. D	15. B	16. A
17. A	18. B	19. D	20. B	21. D	22. B	23. B	24. B
25. B							

EXERCISE QUESTIONS

1. Give a brief description of the history of C language.
2. Discuss briefly the characteristics of C.
3. What are the advantages and disadvantages of C?
4. What are the various keywords used in C?
5. Explain different data types in C.
6. What is a constant? Explain the constants in C.
7. What is a variable? How are the variables declared in C?
8. List the different types of control statements in C.
10. Give the output of the following code:

```
float c = 34.78650;
printf ("%6.2f" , c);
```

11. Write a program in C to find the value of y using the relation $y = x_2 + 2x - 1$.

12. Write a program in C to find the sum and the average of three numbers.
13. Write a program in C to calculate simple interest.
14. Write a program in C to convert the temperature from °C to °F.
15. Write a program in C to evaluate the series $S = 1 + 2 * 1 + 3 * 2 + \ldots N * N - 1$.
16. Write a program to determine and print the sum of the following harmonic series for a given value of *n*:

    $$1+ 1/2 +1/3 + \ldots + 1/n$$

    The value of *n* should be given interactively through the terminal.
17. Write a program to read the price of an item in decimal form (like 15.95) and print the output in paise (like 1595 paise).
18. Write a program that prints the even numbers from 1 to 100.
19. Write a program that requests two float-type numbers from the user and then divides the first number by the second and displays the result along with the numbers.
20. The price of 1 kg rice is Rs. 16.75 and 1 kg of sugar is Rs. 15. Write a program to get these values from the user and display the prices as follows:

```
*** LIST OF ITEMS ***
     Item            Price
     Rice            Rs 16.75
     Sugar           Rs 15.00
```

21. Write program to count and print the number of negative and positive numbers in a given set of numbers. Test your program with a suitable set of numbers. Use scanf to read the numbers. Reading should be terminated when the value 0 is encountered.
22. Write a program to do the following:
    (a) Declare *x* and *y* as integer variables and *z* as a short integer variable.
    (b) Assign two 6 digit numbers to *x* and *y*
    (c) Assign the sum of *x* and *y* to *z*
    (d) Output the values of *x*, *y* and *z*

    Comment on the output.
23. Write a program to read two floating-point numbers using a `scanf` statement, assign their sum to an integer variable and then output the values of all the three variables.
24. Write a program to illustrate the use of typedef declaration in a program.
25. Write a program to illustrate the use of symbolic constants in a real-life application.

ANSWERS TO 2009 QUESTION PAPERS

PART A

1. What is meant by entry-controlled loop? (Anna University, Jan - Feb 2009)

   Ans: Refer Section 4.23

2. Define conditional operators. (Anna University, Jan - Feb 2009)

   Ans: Refer Section 4.19.6

3. What are trigraph characters? (Anna University, Jan - Feb 2009)

   Ans: Refer Section 4.7.1

4. Mention the use of break and continue statement. (Anna University, Jan - Feb 2009)

   Ans: Refer Section 4.23.4

5. What would be the value of *x* after execution of the following statement int *x*, *y* = 10; char *z* = '*a*' ; *x* = *y*+*z*? (Anna University, Jan - Feb 2009)

   Ans: The answer is 107

   Because, ASCII value of *a* = 97

6. State the associativity property of an operator. (Anna University, Jan - Feb 2009)

   Ans: Associativity rule decides the order in which multiple occurrences of the same level operator are applied. For example, in case of arithmetic operators, *, /, %, +, and -, the associativity is from left to right.

7. What are the various character test functions used in 'C' language? (Anna University, Jan - Feb 2009)

   Ans: Refer Section 4.17.2

8. Write the following conditions using " ? : " operators. (GE2112)

$$\text{Salary} = \begin{cases} 4x + 100 & \text{for } x, 40 \\ 300 & \text{for } x = 40 \\ 4.5x = 150 & \text{for } x > 40 \end{cases}$$

   Ans: The given complex equation can be realised using conditional operator as:

   salary = (*x* != 40) ? ((*x* < 40) ? (4**x*+100) : (4.5**x*+150)) : 300;

9. Give the meaning of the following keywords in C: auto, double, int, long. (GE1102)

   Ans: Refer Section 4.12, 4.13 and 4.14

10. What is the main advantage of conditional operator? Give the syntax of conditional operators in C. (GE1102)

    Ans: Refer Section 4.19.6

11. What is a ternary operator? Give an example. (Anna University, Jan - Feb 2009)

    Ans: Refer Section 4.19.6

12. What is the use of "break" statement in C? (Anna University, Jan - Feb 2009)
    Ans: Refer Section 4.23.4
13. List the rules for defining variables in C. (Anna University, Jan - Feb 2009)
    Ans: Refer Section 4.11
14. What is an identifier? Give any two examples for an identifier.
    (Anna University, Jan - Feb 2009)
    Ans: Refer Section 4.9
15. Write a C program to print the following pattern: (Anna University, Jan - Feb 2009)

```
    0
  1 0 1
2 1 0 1 2
```

Ans:

```
Program
#include<conio.h>
#include<stdio.h>
main()
{
int a,b,c,d;
clrscr();
printf("%5d\n",0);
printf("%6d\n",101);
printf("%7d",21012);
getch();
}
Output
     0
    101
  21012
```

PART B

1. Explain the program control structures with suitable examples.
   (Anna University, Jan - Feb 2009)
   Ans: Refer Sections 4.21 and 4.23
2. What is meant by the operator precedence? Discuss its impact in the programming with suitable example. (Anna University, Jan - Feb 2009)
   Ans: Refer Section 4.19.9 and 4.19.10
3. How does switch statement differ from `if` statement? Explain with example.
   (Anna University, Jan - Feb 2009)

Ans: The switch statement is particularly useful in situations where decision-making is to be performed on multiple alternative options. Such a situation can also be realised with the help of nested if-else blocks, but such an approach is complex and error prone. A single if statement allows us to select only between two alternatives, whereas switch statement allows the selection among multiple alternatives.

For more information on If and Switch statements, refer Sections 4.21.1 and 4.21.2.

4. Explain the various storage classes used in 'C' language. (Anna University, Jan - Feb 2009)

Ans: Refer Section 4.14.

5. Differentiate between operator and operand. Give an example. Describe the various types of operators supported by C. (GE1102)

Ans: An operator specifies the operation that is to be performed, while operands refer to the data values on which the operators perform the specific operations. For example, consider the following arithmetic operations:

$$z=x+y$$

Here,

+ is the arithmetic addition operator, = is the assignment operator, x and y are the operands on which addition operation is performed and z is used to store the result of the addition operation.

For information on various types of operators, refer Section 4.19.

6. Write a program in C to find Fibonacci series up to 100. (GE1102)

Ans:

Program

```
#include<conio.h>
#include<stdio.h>
void main()
{
int i,x,m=0,n=1;

clrscr();
printf("The fibonacci series up to 100 terms is as follows:\n\n");
printf("%d\n%d\n",m,n);
for(i=0;i<10;i++)
{
x=m;
m=n;
n=x+n;
printf("%d\n",n);
}
```

```
getch();
}
```

Output

```
The fibonacci series up to 100 terms is as follows:

0
1
1
2
3
5
8
13
21
34
55
89
```

7. Write C assignment statement to evaluate the following equations: (GE1102)
   (a) Area = $\pi r^2 + 2\pi rh$
   (b) Side = $\sqrt{a^2 + b^2 - 2ab\cos(x)}$

   Ans: (a) Area = 3.14**r***r*+ 2*3.14**r***h*
   (b) Side= sqrt(*a***a*+ *b***b*-(2**a***b**cos(x)))
8. How can the precision be specified within a printf function? (GE1102)
   Ans: Refer Section 4.17.5
9. Describe different types of operators included in C. (GE1102)
   Ans: Refer Section 4.19
10. Explain any eight formatting features with an example for each. (GE2112)
    Ans: Refer Section 4.17
11. Explain the various looping constructs. Give an example for each and explain the working of the construct. (GE2112)
    Ans: Refer Section 4.23

12. Write a C program to print the given number in reverse order. (GE2112)

Ans:

```
Program
#include<conio.h>
#include<stdio.h>
main()
{
long m,n,d,r=0;
clrscr();
printf("Enter the number to reverse:\t");
scanf("%ld",&m);
while(m!=0)
{
d=m%10;
r=r*10+d;
m=m/10;
n=r;
}
printf("\nThe reverse of entered number is %ld\n\t",n);
getch();
}
Output
Enter the number to reverse:    123456789
The reverse of entered number is 987654321
```

13. Explain any 2 constructs used for decision making. Give an example for each and explain the working of the construct. (GE2112)

Ans: Refer Section 4.21

14. Write note on storage classes in C. (GE2112)

Ans: Refer Section 4.14

15. Explain the various loop structures available in C. (Anna University, Jan - Feb 2009)

Ans: Refer Section 4.23

16. Write a C program to check if a given number is a prime number.

(Anna University, Jan - Feb 2009)

Ans:

```
Program
#include<conio.h>
#include<stdio.h>
```

```
void main()
{
int a,c=0,i,n;
clrscr();
printf("Enter the number: ");
scanf("%d",&n);
for(i=1;i<=n;i++)
   {
     a=n%i;
     if(a==0)
  {
     c=c+1;
  }
    }
if (c==2)
    printf("The entered number is prime");
else
    printf("The entered number is not prime");
getch();
}
Output
Enter the number: 8
The entered number is not prime
```

17. Write a C program to convert the number of years represented by an integer into the following units of time:
    (a) Minutes
    (b) Hours
    (c) Days
    (d) Months
    (e) Seconds

    Use `switch ( )` statement.

    State the assumption made in your problem. (Anna University, Jan - Feb 2009)

    Ans:

```
Program
/* Assumptions made in this program are:
1 year =365 days
1 year= 12 months
1 day = 24 hours
```

```
1 hour = 60 minutes
1 minute = 60 seconds
*/

#include<conio.h>
#include<stdio.h>
void main()
{
long double years;
long double temp;
int option;
clrscr();
printf("Enter the number of years\n\n");
scanf("%Lf",&years);
printf("\nSelect the option for converting the years into corresponding unit
of time:-\n\n 1. Hours\n 2. Minutes\n 3. Seconds\n 4. Days\n 5. Months\n");
scanf("%d",&option);
switch(option)
{
case 1:
      temp=years*365*24;
      break;
case 2:
      temp=years*365*24*60;
      break;
case 3:
      temp=years*365*24*60*60;
      break;
case 4:
      temp=years*365;
      break;
case 5:
    temp=years*12;
    break;
}
printf("The output is: %10.2Lf ",temp);
getch();
}
```

```
Output
Enter the number of years
2
Select the options for converting the years into corresponding unit:-

 1. Hours
 2. Minutes
 3. Seconds
 4. Days
 5. Months
3
The output is: 63072000.00
Enter the number of years
5
Select the options for converting the years into corresponding unit:-

 1. Hours
 2. Minutes
 3. Seconds
 4. Days
 5. Months
1
The output is:   43800.00
```

ANSWERS TO 2010 AND 2011 QUESTION PAPERS

SHORT ANSWER QUESTIONS

1. Write a code segment using while statement to print numbers from 10 down to 1.
(AU, Chn, Jan 2010)

Ans:

```
i = 10;
while(i > 0)
{
printf("%d\n",i);
i = i – 1;
}
```

2. What are the features of word processor software packages? (AU, Chn, Jan 2011)

Ans:

Features of Word Processor Packages

Some of the key features supported by modern word processing packages are:

- Formatting support to manage the look and feel of the text
- Table feature for creating and managing tabular data
- Pagination support for paginating the document in desired format
- Chart feature for adding charts
- Graphic support for adding graphics in a document

3. What is the main feature and applications of C language? (AU, Chn, Jan 2011)

Ans: Refer Section 4.1

4. Define with example integer and floating type of data in C language. (AU, Chn, Jan 2011)

Ans:

Integer Types: Refer Section 4.12.1.

Floating Point Types: Refer Section 4.12.2.

5. What is the significance of WHILE statement in C? Give a typical format of it? (AU, Mdu, Jan 2011)

Ans:

Significance of while Statement

The while statement is a looping construct of C language that allows the execution of a particular set of code multiple number of times. It is divided into two parts: head and body. The head of the statement decides the number of times instructions present inside the body are going to be executed. It is an important C programming construct that is frequently used in different programming situations.

Format of while Statement

```
while(test condition)
{
 body of the loop
}
```

6. What is Bit wise operation in C? (AU, Mdu, Jan 2011)

Ans:

Bit Wise Operation

A bit wise operation is performed for manipulating the data at bit level. C supports various bit wise operators for performing bit wise operations. For example, & performs the bitwise AND operation while | performs the bitwise OR operator.

The following program demonstrates the use of bitwise operators:

Program

```
/*Program for demonstrating the use of bitwise operators*/
#include <stdio.h>
#include <conio.h>

void main()
{
 char a='A',b='B';
 clrscr();

 printf("a & b = %c",a&b);/*Using bitwise & operator*/
 printf("\na | b = %c",a|b);/*Using bitwise | operator*/

 getch();
}
```

Output

```
a & b = @
a | b = C
```

7. Define a structured programming language. (AU, Cbe, Dec 10-Jan 11)

   Ans: Refer Section 4.25.

8. Categorize the operators used in C. (AU, Cbe, Dec 10-Jan 11)

   Ans:

   Categories of Operators used in C: The various categories of operators used in C are:

   - Arithmetic operators
   - Relational operators
   - Logical operators
   - Assignment operators
   - Increment and decrement operators
   - Conditional operators
   - Bitwise operators
   - Special operators

9. Write Bitwise Operator with example. (AU, Cbe, Dec 10-Jan 11)

Ans:

Bitwise Operators

Bitwise operators are special C operators that help manipulate data at bit level. Some of the typical bitwise operations are: bitwise AND, bitwise OR, shift left, shift right, etc.

Example

```
/*Program for demonstrating the use of bitwise operators*/

void main()
{
 char a='A',b='B';

 printf("a & b = %c",a&b);/*Using bitwise & operator*/
 printf("\na | b = %c",a|b);/*Using bitwise | operator*/

}
```

Output

```
a & b = @
a | b = C
```

10. Write Ruler for defining Real Constants. (AU, Cbe, Dec 10-Jan 11)

Ans:

Rules for defining Real Constant

- It must have at least one digit.
- It must have a decimal point.
- It should not have any blank space or comma inside.
- The mantissa and the exponential parts must be separated by 'e'.
- Negative values should be written by an explicit minus (-) sign. If no sign is mentioned then it is assumed to be positive by default.

11. What are the bitwise operators in C language? (AU, TIR, Dec 10-Jan 11)

Ans: Refer Section 4.19.7.

12. Give two examples for Logical and Relational expressions. (AU, TRI, Jan 2011)

Ans: Refer Sections 4.19.2 and 4.19.3.

13. Differentiate do-while and while-do loop. (AU, TRI, Jan 2011)

Ans:

Difference between do-while and while-do Loops

While-Do	Do-While
It is an entry-controlled loop.	It is an exit-controlled loop.
In while-do or simply while loop, the test condition is checked first and the body of the loop is executed only if the test condition is true.	In do-while loop, the body of the loop is executed first and then the test condition is checked to determine whether the next iteration of the loop will be executed or not.
The body of the loop may never get executed even once if the test condition fails in its very first evaluation.	The body of the loop will be executed at least once even if the test condition fails in its very first evaluation.

Example
`.`
`.`
`int i;`
`i=1;`
`while(i<=10)`
`{`
`printf("%d",i);`
`printf("\n");`
`i=i+1;`
`}`
`.`
`.`
	Example
	`.`
	`.`
	`int i;`
	`i=1;`
	`do`
	`{`
	`printf("%d",i);`
	`printf("\n");`
	`i=i+1;`
	`}`
	`while(i<=10);`
	`.`
	`.`

DESCRIPTIVE QUESTIONS

1. What are the different operators available in C? Explain with examples. (AU, Chn, Jan 2010)
   Ans: Refer Section 4.19.
2. Differentiate between signed and unsigned integer. (AU, Chn, Jan 2010)
   Ans:
   Difference between Signed and Unsigned integer
   The key differences between signed and unsigned integer are:
   - A signed integer also stores the sign of integer value i.e. whether the value is positive or negative. An unsigned integer on the other hand only stores the absolute value
   - In signed integer, the Most Significant Bit (MSB) stores the sign value of the integer (0 for positive and 1 for negative); the remaining seven bits store the absolute value of

the integer. However, in case of unsigned integer, all the eight bits are used to store the absolute value

3. Explain the following conditional statements. (AU, Chn, Jan 2010)

   1. nested if-else statement, 2. switch-case statement

   Ans:

   1. Nested if-else statement: Refer Section 4.21.1.
   2. Switch-case statement: Refer Section 4.21.2.

4. Write a C program that reads a number and display whether the number is prime or not. (AU, Chn, Jan 2010)

   Ans:

   Program to determine whether a number is prime or not

```
#include <stdio.h>
#include <conio.h>
void main()
{
int num,i;
printf("\nEnter a number - ");
scanf("%d",&num);
for(i=2;i<num;i++)
{
if(num%i==0)
{
printf("\n%d is not a prime number",num);
break;
}
else
;
}
if(i==num)
printf("\n%d is a prime number",num);
getch();
}
```

5. What is the basic structure of a C program? Explain with example unformatted Input and Output statements in C language. (AU, Chn, Jan 2011)

   Ans:

   Structure of a C Program: Refer Section 4.3.

Unformatted Input Statements: Refer Section 4.17.1.

Unformatted Output Statements: Refer Section 4.17.2.

6. With example describe the structure of
   (i) if-else statement.
   (ii) Nested if..else statement
   (iii) switch statement in C language. (AU, Chn, Jan 2011)

   Ans:
   (i) if-else Statement: Refer Section 4.21.1.
   (ii) Nested if..else Statement: Refer Section 4.21.1.
   (iii) Switch Statement: Refer Section 4.21.2.
7. Explain a structure of a C program. What are the advantages and applications of C Language ? (AU, Mdu, Jan 2011)

   Ans:

   Structure of a C Program: Refer Section 4.3.

   Advantages and Applications of C Language: Refer Sections 4.1 and 4.2.
8. Define arithmetic operator and its types with example. (AU, Cbe, Dec 10-Jan 11)

   Ans: Refer Section 4.19.1.
9. Explain in detail about managing input and output operators. (AU, Cbe, Dec 10-Jan 11)

   Ans: Refer Section 4.17.
10. Write a C Program to determine whether a given numbers in odd or not using if else statement. (AU, Cbe, Dec 10-Jan 11)

    Ans:

    Program

```
#include <stdio.h>
#include <conio.h>

void main()
{
int num;

printf("Enter a number: ");
scanf("%d",&num);

if(num%2==0)
 printf("%d is an even number",num);
else
 printf("%d is an odd number",num);
```

```
    getch();
    }
```

Output

```
Enter a number: 45
45 is an odd number
```

11. Explain switch case statement with suitable example. (AU, TIR, Dec 10-Jan 11)

    Ans: Refer Section 4.21.2.

12. Explain in detail about the various syntaxes that supports decision making process. (AU, TRI, Jan 2011)

    Ans: Refer Sections 4.21.1 and 4.21.2.

13. Explain the various syntaxes used for performing the branching and looping process. (AU, TRI, Jan 2011)

    Ans:

    Looping Syntaxes: Refer Sections 4.23.1, 4.23.2, 4.23.3.

    Branching Syntaxes: Refer Sections 4.21.3, 4.23.4.

ARRAYS, FUNCTIONS AND POINTERS

5

CHAPTER OBJECTIVE

In this chapter, we will learn:

1. Various types of arrays like one-dimensional, two-dimensional etc.
2. Various string manipulating functions.
3. How to use user-defined functions in C.
4. The concept of structures and unions.
5. The underlying concepts of pointers like pointer to pointer, pointer to functions, pointers as function arguments etc.
6. Various forms of preprocessor directives.
7. The guidelines for developing C programs.

CHAPTER OUTLINE

5.1 INTRODUCTION

In the previous chapter we learned how to get started with C. In this chapter, we'll shift our focus to some of the advanced features of C language that help us to create real-life applications. Some of these features include:

- **Array** It is a collection of multiple sequentially-indexed elements having similar data types. We may use arrays to store symmetric entities together like marks of a student.
- **Pointer** It is a variable that always stores the address location of another variable. We may use pointers for dynamic allocation of memory resources.
- **Structure** It is a user-defined data type, which is used to represent a group of data items of different types under a single name. You may use structures to realize real-life entities like customer details, his name, address, telephone number, etc.

We will also discuss in detail functions and different methods of calling a function such as call by value and call by reference. Finally, we'll go through some important guidelines that should be followed while developing a program in C.

5.2 ARRAYS

So far we have used only the fundamental data types, namely char, int, float, double and variations of int and double. Although these types are very useful, they are constrained by the fact that a variable of these types can store only one value at any given time. Therefore, they can be used only to handle limited amounts of data.

In many applications, however, we need to handle a large volume of data in terms of reading, processing and printing. To process such large amounts of data, we need a powerful data type that would facilitate efficient storing, accessing and manipulation of data items. C supports a derived data type known as array that can be used for such applications.

An array is a fixed-size sequenced collection of elements of the same data type. It is simply a grouping of like-type data. In its simplest form, an array can be used to represent a list of numbers, or a list of names. Some examples where the concept of an array can be used:

- List of temperatures recorded every hour in a day, or a month, or a year.
- List of employees in an organization.
- List of products and their cost sold by a store.
- Test scores of a class of students.
- List of customers and their telephone numbers.
- Table of daily rainfall data.

Since an array provides a convenient structure for representing data, it is classified as one of the data structures in C. Other data structures include structures, lists, queues and trees.

As we mentioned earlier, an array is a sequenced collection of related data items that share a common name. For instance, we can use an array name salary to represent a set of salaries of a group of employees in an organization. We can refer to the individual salaries by writing a number called index or subscript in brackets after the array name. For example, salary [9] represents the salary of 10th employee. While the complete set of values is referred to as an array, individual values are called elements.

The ability to use a single name to represent a collection of items and to refer to an item by specifying the item number enables us to develop concise and efficient programs. For example, we can use a loop construct, discussed in the previous chapter, with the subscript as the control variable to read the entire array, perform calculations, and print out the results.

We can use arrays to represent not only simple lists of values but also tables of data in two, three or more dimensions. In this chapter, we introduce the concept of an array and discuss how to use it to create and apply the following types of arrays.

- One-dimensional arrays
- Two-dimensional arrays
- Multidimensional arrays

5.2.1 One-Dimensional Arrays

A list of items can be given one variable name using only one subscript and such a variable is called a single-subscripted variable or a one-dimensional array. In mathematics, we often deal with variables that are single-subscripted. For instance, we use the following equation to calculate the average of n values of x:

$$A = \frac{\sum_{i=1}^{n} x_i}{n}$$

Here, the subscripted variable Xi refers to the ith element of X. In C, single-subscripted variable Xi can be expressed as

x[1], x[2], x[3],.....x[n]

The subscript can begin with number 0. That is x[0] is allowed.

For example, if we want to represent a set of five numbers, say (35,40,20,57,19), by an array variable number, then we may declare the variable number as follows:

```
int number[5];
```

For the above declaration, the computer will reserve five storage locations as shown in Figure 5.1:

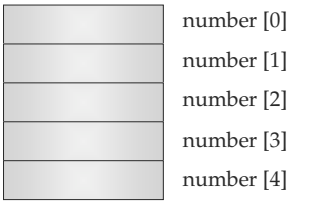


Fig. 5.1 **Representation of array in memory**

The values to the array elements can be assigned as follows:

```
number[0] = 35;
number[1] = 40;
number[2] = 20;
number[3] = 57;
number[4] = 19;
```

This would cause the array number to store the values as shown in Figure 5.2:

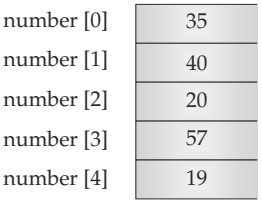


Fig. 5.2 **Storing values in an array**

These elements may be used in programs just like any other C variable. For example, the following are valid statements:

```
a = number[0] + 10;
number[4] = number[0] + number [2];
number[2] = x[5] + y[10];
value[6] = number[i] * 3;
```

The subscripts of an array can be integer constants, integer variables like i, or expressions that yield integers. C performs no bounds checking and, therefore, care should be exercised to ensure that the array indices are within the declared limits.

Declaration of One-Dimensional Arrays Like any other variable, arrays must be declared before they are used so that the compiler can allocate space for them in memory. The general form of array declaration is

```
type variable-name[ size ];
```

The type specifies the type of element that will be contained in the array, such as int, float, or char and the size indicates the maximum number of elements that can be stored inside the array. For example:

```
float height[50];
```

This will declare the height to be an array containing 50 real elements. Any subscripts 0 to 49 are valid. Similarly, int group[10]; will declare the group as an array to contain a maximum of 10 integer constants. However, it is important to remember the following:

- Any reference to the arrays outside the declared limits would not necessarily cause an error. Rather, it might result in unpredictable program results.
- The size should be either a numeric constant or a symbolic constant.

C language treats character strings simply as arrays of characters. The size in a character string represents the maximum number of characters that the string can hold. For instance, consider the following declaration:

```
char name[10];
```

This will declare the name as a character array (string) variable that can hold a maximum of 10 characters. Now, suppose we read the following string constant into the string variable name.

"WELL DONE"

Each character of the string is treated as an element of the array name and is stored in the memory as shown in Figure 5.3:

When the compiler sees a character string, it terminates it with an additional null character. Thus, the element name[10] holds the null character '\0'. When declaring character arrays, we must allow one extra element space for the null terminator.

Fig. 5.3 **Storing string in an array**

EXAMPLE 5.1 *Write a program using a single-subscripted variable to evaluate the following expression:*

$$\text{Total} = \sum_{i=1}^{10} x_2^2$$

The values of x1,x2,....are read from the terminal.

The program in Figure 5.4 uses a one-dimensional array x to read the values and compute the sum of their squares.

```
Program
      main()
      {
         int i ;
         float x[10], value, total ;
/* . . . . . .READING VALUES INTO ARRAY . . . . . . */

      printf("ENTER 10 REAL NUMBERS\n") ;
      for( i = 0 ; i < 10 ; i++ )
      {
            scanf("%f", &value) ;
            x[i] = value ;
      }
/* . . . . . .COMPUTATION OF TOTAL . . . . . .*/
         total = 0.0 ;
         for( i = 0 ; i < 10 ; i++ )
         total = total + x[i] * x[i] ;
/*. . . . PRINTING OF x[i] VALUES AND TOTAL . . . */
            printf("\n");
            for( i = 0 ; i < 10 ; i++ )
               printf("x[%2d] = %5.2f\n", i+1, x[i]) ;
            printf("\ntotal = %.2f\n", total) ;
       }
Output
      ENTER 10 REAL NUMBERS
      1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 9.9 10.10
            x[ 1] = 1.10
            x[ 2] = 2.20
            x[ 3] = 3.30
            x[ 4] = 4.40
            x[ 5] = 5.50
            x[ 6] = 6.60
            x[ 7] = 7.70
            x[ 8] = 8.80
            x[ 9] = 9.90
            x[10] = 10.10
            Total = 446.86
```

Fig. 5.4 **Program to illustrate one-dimensional array**

After an array is declared, its elements must be initialized. Otherwise, they will contain "garbage". An array can be initialized at either of the following stages:

- At compile time
- At run time

Compile Time Initialization We can initialize the elements of arrays in the same way as the ordinary variables when they are declared. The general form of initialization of arrays is:

```
type array-name[size] = { list of values };
```

The values in the list are separated by commas. For example, consider the following statement:

```
int number[3] = { 0,0,0 };
```

This statement will declare the variable number as an array of size 3 and will assign zero to each element. If the number of values in the list is less than the number of elements, then only that many elements will be initialized. The remaining elements will be set to zero automatically. For instance, consider the following statement:

```
float total[5] = {0.0,15.75,–10};
```

This will initialize the first three elements to 0.0, 15.75, and –10.0 and the remaining two elements to zero.

The size may be omitted. In such cases, the compiler allocates enough space for all initialized elements. For example, consider the following statement:

```
int counter[ ] = {1,1,1,1};
```

This will declare the counter array to contain four elements with initial values 1. This approach works fine as long as we initialize every element in the array.

Character arrays may be initialized in a similar manner. Thus, the statement char name[ ] = {'J','o', 'h', 'n', '\0'}; declares the name to be an array of five characters, initialized with the string "John" ending with the null character.

Run Time Initialization An array can be explicitly initialized at run time. This approach is usually applied for initializing large arrays. For example, consider the following segment of a C program.

```
_______
_______
for (i = 0; i < 100; i = i+1)
{
      if    i < 50
            sum[i] = 0.0;            /* assignment statement */
      else
            sum[i] = 1.0;
}
_______
_______
```

The first 50 elements of the array sum are initialized to zero while the remaining 50 elements are initialized to 1.0 at run time.

EXAMPLE 5.2 *Figure 5.5 shows a simple program that prompts the user to enter values in an array at run time and then prints the same array values on the screen.*

```
#include<stdio.h>
#include<conio.h>
void main()
{
int array[10]; /*Declaration of an array*/
```

```
int i;
clrscr();
printf("Enter the elements of an array:\n");
for(i=0;i<10;i++)
{
      scanf("%d",&array[i]);
}
printf("The array elements are:\n");
for(i=0;i<10;i++)
      printf("array[%d]=%d\n",i,array[i]);
getch();
}

Output
Enter the elements of an array:
10
20
30
40
50
60
70
80
91
100
The array elements are:
array[0]=10
array[1]=20
array[2]=30
array[3]=40
array[4]=50
array[5]=60
array[6]=70
array[8]=90
array[9]=100
```

Fig. 5.5 A simple array program

EXAMPLE 5.3 *Given below is the list of marks obtained by a class of 50 students in an annual examination.*

43 65 51 27 79 11 56 61 82 09 25 36 07 49 55 63 74 81 49 37

40 49 16 75 87 91 33 24 58 78 65 56 76 67 45 54 36 63 12 21

73 49 51 19 39 49 68 93 85 59

Write a program to count the number of students belonging to each of following groups of marks: 0–9, 10–19, 20–29,.....,100.

The program shown in Figure 5.6 uses the array group containing 11 elements, one for each range of marks. Each element counts those values falling within the range of values it represents.

For any value, we can determine the correct group element by dividing the value by 10. For example, consider the value 59. The integer division of 59 by 10 yields 5. This is the element into which 59 is counted.

```
Program
      #define   MAXVAL     50
      #define   COUNTER    11
      main()
      {
            float       value[MAXVAL];
            int            i, low, high;
            int   group[COUNTER] = {0,0,0,0,0,0,0,0,0,0,0};
            /* . . . . . . . .READING AND COUNTING . . . . . .*/
            for( i = 0 ; i < MAXVAL ; i++ )
            {
            /*. . . . . . . .READING OF VALUES . . . . . . . . */
               scanf("%f", &value[i]) ;
            /*. . . . . .COUNTING FREQUENCY OF GROUPS. . . . . */
               ++ group[ (int) ( value[i]) / 10] ;
            }
            /* . . . .PRINTING OF FREQUENCY TABLE . . . . . .*/
            printf("\n");
            printf(" GROUP   RANGE    FREQUENCY\n\n") ;
            for( i = 0 ; i < COUNTER ; i++ )
            {
                  low = i * 10 ;
                  if(i == 10)
                     high = 100 ;
                  else
                     high = low + 9 ;
                  printf(" %2d %3d to %3d %d\n",
                           i+1, low, high, group[i] ) ;
            }
      }
Output
      43 65 51 27 79 11 56 61 82 09 25 36 07 49 55 63 74
      81 49 37 40 49 16 75 87 91 33 24 58 78 65 56 76 67 (Input data)
      45 54 36 63 12 21 73 49 51 19 39 49 68 93 85 59
    GROUP       RANGE       FREQUENCY
      1     0     to    9        2
      2     10    to    19       4
      3     20    to    29       4
      4     30    to    39       5
      5     40    to    49       8
      6     50    to    59       8
      7     60    to    69       7
      8     70    to    79       6
      9     80    to    89       4
      10    90    to    99       2
      11    100   to    100      0
```

Fig. 5.6 Program for frequency counting

Note that here we have used the following initialization statement:

```
int group [COUNTER] = {0,0,0,0,0,0,0,0,0,0};
```

This statement can also be replaced by:

```
int group [COUNTER] = {0};
```

This will initialize all the elements to zero.

Searching and Sorting

Searching and sorting are the two most frequent operations performed on arrays. Computer Scientists have devised several data structures and searching and sorting techniques that facilitate rapid access to data stored in lists.

Sorting is the process of arranging elements in the list according to their values, in ascending or descending order. A sorted list is called an ordered list. Sorted lists are especially important in list searching because they facilitate rapid search operations. Many sorting techniques are available. The three simple and most important among them are:

- Bubble sort
- Selection sort
- Insertion sort

Other sorting techniques include Shell sort, Merge sort and Quick sort.

Searching is the process of finding the location of the specified element in a list. The specified element is often called the search key. If the process of searching finds a match of the search key with a list element value, the search said to be successful; otherwise, it is unsuccessful. The two most commonly used search techniques are:

- Sequential search
- Binary search

A detailed discussion on these techniques is beyond the scope of this text. Consult any good book on data structures and algorithms.

EXAMPLE 5.7 *Write a program for sorting the elements of an array in descending order.*

1. Set the size n of an array.
2. Store n elements in the array.
3. Read and store elements of the array in descending order.
4. Print the elements of array in descending order.

Figure 5.7 gives a program to implement this algorithm.

```
Program
#include <stdio.h>
#include <conio.h>
void main()
{
```

```
      int *arr,temp,i,j,n;
      clrscr();
      printf("Enter the number of elements in the array:");
      scanf("%d",&n);
      arr=(int*)malloc(sizeof(int)*n);
      for(i=0;i<n;i++)
      {
         printf("Enter a number:");
         scanf("%d",&arr[i]);
      }
      for(i=0;i<n;i++)
      {
            for(j=i+1;j<n;j++)
            {
               if(arr[i]<arr[j])
               {
                  temp=arr[i];
                  arr[i]=arr[j];
                  arr[j]=temp;
               }
            }
      }
      printf("Elements of array in descending order are:\n");
      for(i=0;i<n;i++)
         printf("%d\n",arr[i]);
      getch();
}
Output
Enter the number of elements in the array:5
Enter a number:32
Enter a number:43
Enter a number:23
Enter a number:57
Enter a number:47
Elements of array in descending order are:
57
47
43
32
23
```

Fig. 5.7 Program to sort the elements of an array in descending order

EXAMPLE 5.8 *Write a program for finding the largest number in an array.*

1. Set the size of the array as n.
2. Store n elements in the array.
3. Assign the first element of the array to LARGE.
4. Compare each element in the array with LARGE.
5. If an element of the array is greater then LARGE, then set LARGE=element.

6. Else go to step 4.
7. Repeat this process until it reaches the last element of the array.
8. Print the largest element found in the array.

Figure 5.8 gives a program to implement this algorithm.

```
Program
#include <stdio.h>
#include <conio.h>
void main()
{
      int *arr,i,j,n,LARGE;
      clrscr();
      printf("Enter the number of elements in the array:");
      scanf("%d",&n);
      arr=(int*)malloc(sizeof(int)*n);
      for(i=0;i<n;i++)
      {
         printf("Enter a number:");
         scanf("%d",&arr[i]);
      }
      LARGE=arr[0];
      for(i=1;i<n;i++)
      {
         if(arr[i]>LARGE)
            LARGE=arr[i];
      }
         printf("The largest number in the array is: %d",LARGE);
      getch();
}
Output
Enter the number of elements in the array:5
Enter a number:32
Enter a number:43
Enter a number:23
Enter a number:57
Enter a number:47
The largest number in the array is:57
```

Fig. 5.8 Program to find the largest element in an array

5.2.2 Two-Dimensional Arrays

So far we have discussed the array variables that can store a list of values. There could be situations where a table of values will have to be stored. Consider Table 5.1, which shows the value of sales of three items by four sales girls.

The table contains a total of 12 values, three in each line. We can think of this table as a matrix consisting of four rows and three columns. Each row represents the values of sales by a particular salesgirl and each column represents the values of sales of a particular item.

TABLE 5.1 Sale figures

Name	Item1	Item2	Item3
Salesgirl #1	310	275	365
Salesgirl #2	210	190	325
Salesgirl #3	405	235	240
Salesgirl #4	260	300	380

In mathematics, we represent a particular value in a matrix by using two subscripts such as vij. Here v denotes the entire matrix and vij refers to the value in the ith row and jth column. For example, in the above table v23 refers to the value 325.

C allows us to define such tables of items by using two-dimensional arrays. The table discussed above can be defined in C as

v[4][3]

Two-dimensional arrays are declared as follows:

```
type array_name [row_size][column_size];
```

Note that unlike most other languages, which use one pair of parentheses with commas to separate array sizes, C places each size in its own set of brackets.

Two-dimensional arrays are stored in memory, as shown in Figure 5.9. As with the single-dimensional arrays, each dimension of the array is indexed from zero to its maximum size minus one; the first index selects the row and the second index selects the column within that row.

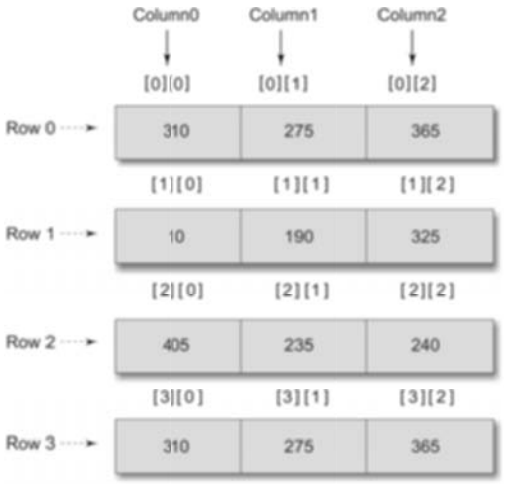


Fig. 5.9 **Representation of a two-dimensional array in memory**

Initializing Two-Dimensional Arrays Like the one-dimensional arrays, two-dimensional arrays may be initialized by following their declaration with a list of initial values enclosed in braces. For example:

```
int table[2][3] = { 0,0,0,1,1,1};
```

The above statement initializes the elements of the first row to zero and the second row to one. The initialization is done row by row. The above statement can be equivalently written as:

```
int table[2][3] = {{0,0,0}, {1,1,1}};
```

Here, we have surrounded the elements of the each row by braces.

We can also initialize a two-dimensional array in the form of a matrix as shown below:

```
int table[2][3] = {
{0,0,0},
{1,1,1}
            };
```

Note the syntax of the above statements. Commas are required after each brace that closes off a row, except in the case of the last row.

When the array is completely initialized with all values, explicitly, we need not specify the size of the first dimension. For example:

```
int table [ ] [3] = {
{ 0, 0, 0},
{ 1, 1, 1}
            };
```

If the values are missing in an initializer, they are automatically set to zero. For instance, consider the following statement:

```
int table[2][3] = {
{1,1},
{2}
            };
```

This will initialize the first two elements of the first row to one, the first element of the second row to two, and all other elements to zero.

When all the elements are to be initialized to zero, the following short-cut method may be used.

```
int m[3][5] = { {0}, {0}, {0}};
```

EXAMPLE 5.9 *Let us create a program using a two-dimensional array to compute and print the following information from the Table 5.1 discussed above:*

(a) Total value of sales by each girl.
(b) Total value of each item sold.
(c) Grand total of sales of all items by all girls.

Figure 5.10 shows the program and its output.

```
Program
   #define  MAXGIRLS 4
   #define  MAXITEMS 3
   main()
   {
      int   value[MAXGIRLS][MAXITEMS];
      int   girl_total[MAXGIRLS] , item_total[MAXITEMS];
      int   i, j, grand_total;
```

```
        /*.......READING OF VALUES AND COMPUTING girl_total ...*/
           printf("Input data\n");
           printf("Enter values, one at a time, row-wise\n\n");

           for( i = 0 ; i < MAXGIRLS ; i++ )
           {
                girl_total[i] = 0;
                for( j = 0 ; j < MAXITEMS ; j++ )
                {
                     scanf("%d", &value[i][j]);
                     girl_total[i] = girl_total[i] + value[i][j];
                }
           }
        /*.......COMPUTING item_total.........................*/
           for( j = 0 ; j < MAXITEMS ; j++ )
           {
        item_total[j] = 0;
        for( i =0 ; i < MAXGIRLS ; i++ )
           item_total[j] = item_total[j] + value[i][j];
           }
/*.......COMPUTING grand_total.........................*/
           grand_total = 0;
           for( i =0 ; i < MAXGIRLS ; i++ )
               grand_total = grand_total + girl_total[i];
/*.......PRINTING OF RESULTS.........................*/
           printf("\n GIRLS TOTALS\n\n");
           for( i = 0 ; i < MAXGIRLS ; i++ )
           printf("Salesgirl[%d] = %d\n", i+1, girl_total[i] );
           printf("\n ITEM TOTALS\n\n");

           for( j = 0 ; j < MAXITEMS ; j++ )
           printf("Item[%d] = %d\n", j+1 , item_total[j] );
           printf("\nGrand Total = %d\n", grand_total);
        }
      Output
        Input data
        Enter values, one at a time, row_wise
        310 257 365
        210 190 325
        405 235 240
        260 300 380
        GIRLS TOTALS
        Salesgirl[1] = 950
        Salesgirl[2] = 725
        Salesgirl[3] = 880
        Salesgirl[4] = 940
        ITEM TOTALS
        Item[1] = 1185
        Item[2] = 1000
        Item[3] = 1310
        Grand Total = 3495
```

Fig. 5.10 Illustration of two-dimensional arrays

EXAMPLE 5.10 *Write a program to compute and print a multiplication table for numbers 1 to 5 as shown below:*

	1	2	3	4	5
1	1	2	3	4	5
2	2	4	6	8	10
3	3	6	.	.	.
4	4	8	.	.	.
5	5	10	.	.	25

Figure 5.11 shows the program to print the multiplication table:

```
Program
        #define  ROWS       5
        #define  COLUMNS 5
        main()
        {
           int row, column, product[ROWS][COLUMNS] ;
           int i, j ;
           printf(" MULTIPLICATION TABLE\n\n") ;
           printf(" ") ;
           for( j = 1 ; j <= COLUMNS ; j++ )
              printf("%4d" , j ) ;
           printf("\n") ;
           printf("————————————————————————————\n");
           for( i = 0 ; i < ROWS ; i++ )
           {
                 row = i + 1 ;
                 printf("%2d |", row) ;
                 for( j = 1 ; j <= COLUMNS ; j++ )
                 {
                    column = j ;
                    product[i][j] = row * column ;
                    printf("%4d", product[i][j] ) ;
                 }
                 printf("\n") ;
           }
        }
     Output
                      MULTIPLICATION TABLE
              1       2       3       4       5
     ————————————————————————————————————————————
      1   |   1       2       3       4       5
      2   |   2       4       6       8       10
      3   |   3       6       9       12      15
      4   |   4       8       12      16      20
      5   |   5       10      15      20      25
```

Fig. 5.11 Program to print multiplication table using two-dimensional array

In the above program, a two-dimensional array is used to store the table values. Each value is calculated using the control variables of the nested for loops as follows:

```
product[i] [j] = row * column
```

Here i denotes rows and j denotes columns of the product table. Since the indices i and j range from 0 to 4, we have introduced the following transformation:

```
row = i+1
column = j+1
```

Memory Layout The subscripts in the definition of a two-dimensional array represent rows and columns. This format maps the way that data elements are laid out in the memory. The elements of all arrays are stored contiguously in increasing memory locations, essentially in a single list. If we consider the memory as a row of bytes, with the lowest address on the left and the highest address on the right, a simple array will be stored in memory with the first element at the left end and the last element at the right end. Similarly, a two-dimensional array is stored "row-wise, starting from the first row and ending with the last row, treating each row like a simple array. This is illustrated below in Figure 5.12:

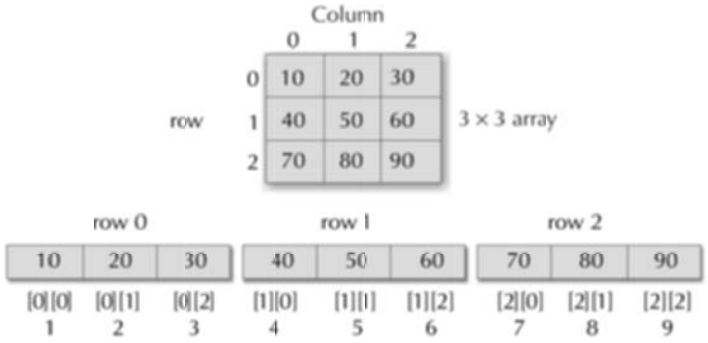


Fig. 5.12 Memory Layout

For a multi-dimensional array, the order of storage is that the first element stored has 0 in all its subscripts; the second has all of its subscripts 0 except the far right which has a value of 1 and so on.

The elements of a 2 x 3 x 3 array will be stored as shown in Figure 5.13:

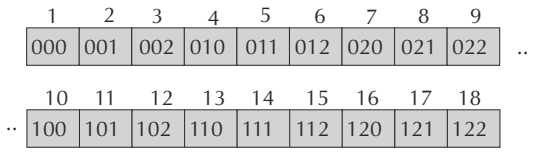


Fig. 5.13 Memory Layout of a 2 x 3 x 3 array

The far right subscript increments first and the other subscripts increment in order from right to left. The sequence numbers 1, 2,……, 18 represent the location of that element in the list.

5.2.3 Multi-Dimensional Arrays

C also allows arrays of three or more dimensions. The exact limit is determined by the compiler. The general form of a multi-dimensional array is

```
type array_name[s1][s2][s3]....[sm];
```

Here si is the size of the i^{th} dimension. Some examples are:

```
int survey[3][5][12];
float table[5][4][5][3];
```

Here, survey is a three-dimensional array declared to contain 180 integer type elements. Similarly, table is a four-dimensional array containing 300 elements of floating-point type. The array survey may represent a survey data of rainfall during the last three years from January to December in five cities.

If the first index denotes year, the second city and the third month, then the element survey[2][3][10] denotes the rainfall in the month of October during the second year in city-3.

Remember that a three-dimensional array can be represented as a series of two-dimensional arrays as shown below:

	Month city	**1**	**2**	……	**12**
Year 1	1 5				

	Month city	**1**	**2**	……	**12**
Year 2	1 5				

NOTE: ANSI C does not specify any limit for array dimension. However, most compilers permit seven to ten dimensions. Some allow even more.

5.3 CASE STUDY

Evaluating a Test

A test consisting of 25 multiple-choice items is administered to a batch of 3 students. Correct answers and student responses are tabulated as shown Figure 5.14:

Fig. 5.14 Student Responses to MCQs

The algorithm for evaluating the answers of students is as follows:

1. Read correct answers into an array.
2. Read the responses of a student and count the correct ones.
3. Repeat step-2 for each student.
4. Print the results.

A program to implement this algorithm is given in Figure 5.15. The program uses the following arrays:

- key[i] - To store correct answers of items
- response[i] - To store responses of students
- correct[i] - To identify items that are answered correctly

```
Program
#define STUDENTS 3
#define ITEMS 25
main( )
{
      char key[ITEMS+1],response[ITEMS+1];
      int count, i, student,n, correct[ITEMS+1];
      /* Reading of Correct answers */
      printf("Input key to the items\n");
      for(i=0; i < ITEMS; i++)
         scanf("%c",&key[i]);
      scanf("%c",&key[i]);
      key[i] = '\0';
      /* Evaluation begins */
      for(student = 1; student <= STUDENTS ; student++)
      {
      /*Reading student responses and counting correct ones*/
         count = 0;
```

```
            printf("\n");
         printf("Input responses of student-%d\n",student);
         for(i=0; i < ITEMS ; i++)
            scanf("%c",&response[i]);
         scanf("%c",&response[i]);
         response[i] = '\0';
         for(i=0; i < ITEMS; i++)
            correct[i] = 0;
         for(i=0; i < ITEMS ; i++)
            if(response[i] == key[i])
            {
               count = count +1 ;
               correct[i] = 1 ;
            }
         /* printing of results */
         printf("\n");
         printf("Student-%d\n", student);
         printf("Score is %d out of %d\n",count, ITEMS);
         printf("Response to the items below are wrong\n");
         n = 0;
         for(i=0; i < ITEMS ; i++)
            if(correct[i] == 0)
            {
               printf("%d ",i+1);
               n = n+1;
            }
         if(n == 0)
            printf("NIL\n");
         printf("\n");
      } /* Go to next student */
      /* Evaluation and printing ends */
}
Output
      Input key to the items
      abcdabcdabcdabcdabcdabcda
      Input responses of student-1
      abcdabcdabcdabcdabcdabcda
      Student-1
      Score is 25 out of 25
      Response to the following items are wrong
      NIL
      Input responses of student-2
      abcddcbaabcdabcddddddddddd
      Student-2
      Score is 14 out of 25
      Response to the following items are wrong
      5 6 7 8 17 18 19 21 22 23 25
      Input responses of student-3
      aaaaaaaaaaaaaaaaaaaaaaaaa
      Student-3
      Score is 7 out of 25
      Response to the following items are wrong
      2 3 4 6 7 8 10 11 12 14 15 16 18 19 20 22 23 24
```

Fig. 5.15 **Program to evaluate responses to a multiple-choice test**

5.4 HANDLING OF CHARACTER STRINGS

A string is a sequence of characters that is treated as a single data item. We have used strings in a number of examples in the past. Any group of characters (except double quote sign) defined between double quotation marks is a string constant. Example:

"Man is obviously made to think."

If we want to include a double quote in the string to be printed, then we may use it with a back slash as shown below.

"\" Man is obviously made to think."\" said Pascal."

For example:

printf ("\" Well Done !"\"); will result in " Well Done !"

printf(" Well Done !"); will result in Well Done !

Character strings are often used to build meaningful and readable programs. The common operations performed on character strings include:

- Declaring and initializing string variables
- Reading strings from terminal
- Writing strings to terminal
- Combining strings together.
- Comparing strings for equality.
- Copying one string to another.

In the subsequent sections, we shall discuss these operations in detail and examine library functions that implement them.

5.4.1 Declaring and Initializing String Variables

C does not support strings as a data type. However, it allows us to represent strings as character arrays. In C, therefore, a string variable is any valid C variable name and is always declared as an array of characters. The general form of declaration of a string variable is:

```
char string_name[ size ];
```

The size determines the number of characters in the string_name. Some examples are:

```
char city[10];
char name[30];
```

When the compiler assigns a character string to a character array, it automatically supplies a null character ('\0 ') at the end of the string. Therefore, the size should be equal to the maximum number of characters in the string plus one.

Like numeric arrays, character arrays may be initialized when they are declared. C permits a character array to be initialized in either of the following two forms:

```
char city [9] = " NEW YORK ";
char city [9]={'N','E','W',' ','Y','O','R','K','\O'};
```

The reason that city had to be 9 elements long is that the string NEW YORK contains 8 characters and one element space is provided for the null terminator. Note that when we initialize a character array by listing its elements, we must supply explicitly the null terminator.

C also permits us to initialize a character array without specifying the number of elements. In such cases, the size of the array will be determined automatically, based on the number of elements initialized. For example, the following statement defines the array string as a five element array:

```
char string [ ] = {'G','O','O','D','\0'};
```

We can also declare the size much larger than the string size in the initializer. For instance, consider the following statement:

```
char str[10] = "GOOD";
```

In this case, the computer creates a character array of size 10, places the value "GOOD" in it, terminates with the null character, and initializes all other elements to NULL. Figure 5.16 shows how the storage of this string will look like:

Fig. 5.16 Storage representation of a string

However, the following declaration is illegal:

```
char str2[3] = "GOOD";
```

This will result in a compile time error. Also note that we cannot separate the initialization from declaration. For example, the following is not allowed:

```
char str3[5];
str3 = "GOOD";
```

Terminating Null Character

You must be wondering, "why do we need a terminating null character?" As we know, a string is not a data type in C, but it is considered a data structure stored in an array. The string is a variable-length structure and is stored in a fixed-length array. The array size is not always the size of the string and most often it is much larger than the string stored in it. Therefore, the last element of the array need not represent the end of the string. We need some way to determine the end of the string data and the null character serves as the "end-of-string" marker.

5.4.2 Reading Strings from Terminal

Using scanf Function The familiar input function scanf can be used with %s format specification to read in a string of characters. Example:

```
char address[10]
scanf("%s", address);
```

The problem with the scanf function is that it terminates its input on the first white space it finds. A white space includes blanks, tabs, carriage returns, form feeds, and new lines. Therefore, if the string NEW YORK is typed in at the terminal then only the string "NEW" will be read into the array address, since the blank space after the word 'NEW' will terminate the reading of string.

The scanf function automatically terminates the string that is read with a null character and therefore the character array should be large enough to hold the input string plus the null character. Note that unlike previous scanf calls, in the case of character arrays, the ampersand (&) is not required before the variable name.

EXAMPLE 5.10 *Figure 5.17 shows a program to read a series of words from a terminal using scanf function.*

```
Program
      main( )
      {
            char word1[40], word2[40], word3[40], word4[40];
            printf("Enter text : \n");
            scanf("%s %s", word1, word2);
            scanf("%s", word3);
            scanf("%s", word4);
            printf("\n");
            printf("word1 = %s\nword2 = %s\n", word1, word2);
            printf("word3 = %s\nword4 = %s\n", word3, word4);
      }

Output
      Enter text :
      Oxford Road, London M17ED
      word1 = Oxford
      word2 = Road,
      word3 = London
      word4 = M17ED
      Enter text :
      Oxford-Road, London-M17ED United Kingdom
      word1 = Oxford-Road
      word2 = London-M17ED
      word3 = United
      word4 = Kingdom
```

Fig. 5.17 Reading a series of words using scanf function

Note in the above output that the string 'Oxford Road' is treated as two words while the string 'Oxford-Road' as one word.

We can also specify the field width using the form %ws in the scanf statement for reading a specified number of characters from the input string. Example:

```
scanf("%ws", name);
```

Here, two things may happen.

- The width w is equal to or greater than the number of characters typed in. In this case, the entire string will be stored in the string variable.
- The width w is less than the number of characters in the string. In this case, the excess characters will be truncated and left unread.

Now, consider the following statements:

```
char name[10];
scanf("%5s", name);
```

Figure 5.18 shows how the above specified field width will store the input string RAM in the memory:

Fig. 5.18 Storing the string RAM in the memory with field width 5

Figure 5.19 shows how the above specified field width will store the input string KRISHNA in the memory:

Fig. 5.19 Storing the string KRISHNA in the memory with field width 5

Reading a Line of Text We have seen just now that scanf with %s or %ws can read only strings without whitespaces. That is, they cannot be used for reading a text containing more than one word. However, C supports a format specification known as the edit set conversion code %[. .] that can be used to read a line containing a variety of characters, including whitespaces. For example, consider the following statements:

```
char line [80];
scanf("%[^\n]", line);
printf("%s", line);
```

This will read a line of input from the keyboard and display the same on the screen. We would very rarely use this method, as C supports an intrinsic string function to do this job. This is discussed in the next section.

Using getchar Function We can read a single character from the terminal, using the function getchar. We can use this function repeatedly to read successive single characters from the input and place them into a character array. Thus, an entire line of text can be read and stored in an array. The reading is terminated when the newline character ('\n') is entered and the null character is then inserted at the end of the string. The getchar function call takes the following form:

```
char ch;
ch = getchar( );
```

Note that the getchar function is called with no parameters.

EXAMPLE 5.11 *Figure 5.20 shows a program to read a line of text containing a series of words from the terminal.*

```
Program
      #include <stdio.h>
      main( )
      {
         char line[81], character;
         int c;
         c = 0;
         printf("Enter text. Press <Return> at end\n");
         do
         {
            character = getchar();
            line[c] = character;
            c++;
         }
         while(character != '\n');
         c = c - 1;
         line[c] = '\0';
         printf("\n%s\n", line);
      }
Output
      Enter text. Press <Return> at end
      Programming in C is interesting.
      Programming in C is interesting.
      Enter text. Press <Return> at end
      National Centre for Expert Systems, Hyderabad.
      National Centre for Expert Systems, Hyderabad.
```

Fig. 5.20 Program to read a line of text from terminal

The above program can read a line of text (up to a maximum of 80 characters) into the string line using getchar function. Every time a character is read, it is assigned to its location in the string line and then tested for newline character. When the newline character is read (signalling the end of line), the reading loop is terminated and the newline character is replaced by the null character to indicate the end of character string.

Using gets Function Another more convenient method of reading a string of text containing whitespaces is to use the library function gets available in the <stdio.h> header file. This is a simple function with one string parameter and called as under:

```
gets (str);
```

Here, gets reads characters into str from the keyboard until a new-line character is encountered and then appends a null character to the string. Unlike scanf, it does not skip whitespaces. For example, consider the following statements:

```
char line [80];
gets (line);
printf ("%s", line);
```

This will read a line of text from the keyboard and displays it on the screen. The last two statements may be combined as follows:

printf("%s", gets(line));

5.4.3 Writing Strings to Terminal

Using printf Function We have used extensively the printf function with %s format to print strings to the screen. The format %s can be used to display an array of characters that is terminated by the null character. For example, consider the statement:

```
printf("%s", name);
```

This will display the entire contents of the array name.

We can also specify the precision with which the array is displayed. For instance, the specification %10.4 indicates that the first four characters are to be printed in a field width of 10 columns. However, if we include the minus sign in the specification (e.g., %-10.4s), the string will be printed left-justified.

The following illustrates the effect of various %s specifications.

EXAMPLE 5.12 *Figure 5.21 shows the program to store the string "United Kingdom" in the array country and display the string under various format specifications.*

```
Program
      main()
      {
         char country[15] = "United Kingdom";
         printf("\n\n");
         printf("*123456789012345*\n");
         printf(" ------ \n");
         printf("%15s\n", country);
         printf("%5s\n", country);
         printf("%15.6s\n", country);
         printf("%-15.6s\n", country);
         printf("%15.0s\n", country);
         printf("%.3s\n", country);
         printf("%s\n", country);
         printf("------ \n");
      }
Output
      *123456789012345*
```

```
-----
United Kingdom
United Kingdom
       United
United
Uni
United Kingdom
-----
```

Fig. 5.21 Writing strings using %s format

Using putchar Function Like getchar, C supports another character handling function putchar to output the values of character variables. It takes the following form:

```
char ch = 'A';
putchar (ch);
```

The function putchar requires one parameter. The above putchar statement can be realized using printf as under:

```
printf("%c", ch);
```

We can use putchar function to write characters to the screen. We can use this function repeatedly to output a string of characters stored in an array using a loop; just like we used the getchar function for inputting a string of characters in to an array. Example:

```
char name[6] = "PARIS"
for (i=0, i<5; i++)
     putchar(name[i];
putchar('\n');
```

Using puts Function Another more convenient way of printing string values is to use the function puts declared in the header file <stdio.h>. This is a one parameter function and is invoked as under:

```
puts ( str );
```

Here, str is a string variable containing a string value. This prints the value of the string variable str and then moves the cursor to the beginning of the next line on the screen. For example, consider the following program segment:

```
char line [80];
gets (line);
puts (line);
```

This will read a line of text from the keyboard and display it on the screen. Note that the syntax is very simple compared to using the scanf and printf statements.

5.4.4 String-Handling Functions

Just as we cannot assign one string to another directly, we cannot join two strings together by the simple arithmetic addition. That is, the following statements are invalid in C:

```
string3 = string1 + string2;
string2 = string1 + "hello";
```

Similarly, we can not use the logical operators to compare two strings with each other. The C library supports a large number of string-handling functions that can be used to carry out many of these string manipulations. Table 5.2 lists the most commonly used string-handling functions:

TABLE 5.2 String Manipulation Functions

Function	Action
strcat()	Concatenates two strings
strcmp()	Compares two strings
strcpy()	Copies one string over another
strlen()	Finds the length of a string

We shall discuss briefly how each of these functions can be used in the processing of strings.

strcat() Function The strcat function joins two strings together. It takes the following form:

```
strcat(string1, string2);
```

string1 and string2 are character arrays. When the function strcat is executed, string2 is appended to string1. It does so by removing the null character at the end of string1 and placing string2 from there. The string at string2 remains unchanged. For example, consider the three strings shown in Figure 5.22:

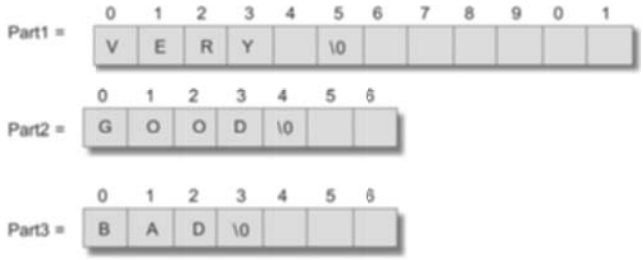


Fig. 5.22 Strings Part, Part2, Part3

Figure 5.23 shows the result of the string function call strcat(part1, part2):

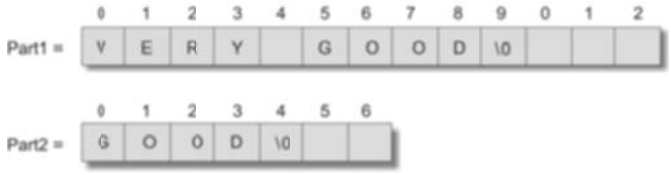


Fig. 5.23 strcat(part1, part2);

Figure 5.24 shows the result of the string function call strcat(part1, part3):

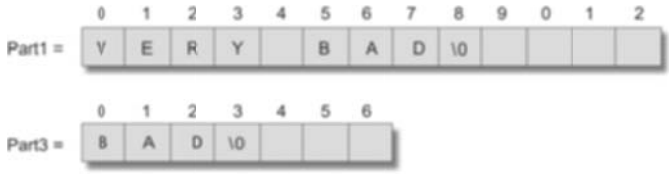


Fig. 5.24 strcat(part1, part3);

We must make sure that the size of string1 (to which string2 is appended) is large enough to accommodate the final string.

strcat function can also be used to append a string constant to a string variable. Thus, the following statement is valid:

```
strcat(part1,"GOOD");
```

C also permits the nesting of strcat functions. Thus, the following statement is valid in C:

```
strcat(strcat(string1,string2), string3);
```

Here, the resultant string is stored in string1.

strcmp() Function The strcmp function compares two strings identified by the arguments and has a value 0 if they are equal. If they are not, it has the numeric difference between the first nonmatching characters in the strings. It takes the following form:

```
strcmp(string1, string2);
```

string1 and string2 may be string variables or string constants. For example, all the following strcmp function calls are valid:

```
strcmp(name1, name2);
strcmp(name1, "John");
strcmp("Rom", "Ram");
```

Our major concern is to determine whether the strings are equal; if not, which is alphabetically above. The value of the mismatch is rarely important. For example, consider the following statement:

```
strcmp("their", "there");
```

This will return a value of –9 which is the numeric difference between ASCII "i" and ASCII "r". That is, "i" minus "r" in ASCII code is –9. If the value is negative, string1 is alphabetically above string2 and vice versa.

strcpy() Function The strcpy function works almost like a string-assignment operator. It takes the following form:

```
strcpy(string1, string2);
```

This will assign the contents of string2 to string1. string2 may be a character array variable or a string constant. For example, consider the following statement:

```
strcpy(city, "DELHI");
```

This will assign the string "DELHI" to the string variable city. Similarly, the statement strcpy(city1, city2) will assign the contents of the string variable city2 to the string variable city1. However, the size of the array city1 should be large enough to receive the contents of city2.

```
strlen() Function
```

This function counts and returns the number of characters in a string. It takes the following form:

```
n = strlen(string);
```

Here n is an integer variable, which receives the value of the length of the string. The argument may be a string constant. The counting ends at the first null character.

EXAMPLE 5.13 *Figure 5.25 shows a program that performs common string manipulation operations. It uses three string variables s1, s2, and s3. Two string constants are first read into s1 and s2 and compared whether they are equal or not. If they are not equal then both are joined together. Then, contents of s1 are copied in to the variable s3. At the end, the program prints the contents of all the three string variables and their lengths.*

```
Program
      #include <string.h>
      main()
      {  char s1[20], s2[20], s3[20];
         int x, l1, l2, l3;
         printf("\n\nEnter two string constants \n");
         printf("?");
         scanf("%s %s", s1, s2);
      /* comparing s1 and s2 */
         x = strcmp(s1, s2);
```

```
         if(x != 0)
         {  printf("\n\nStrings are not equal \n");
            strcat(s1, s2); /* joining s1 and s2 */
         }
         else
            printf("\n\nStrings are equal \n");
      /* copying s1 to s3
         strcpy(s3, s1);
      /* Finding length of strings */
         l1 = strlen(s1);
         l2 = strlen(s2);
         l3 = strlen(s3);
      /* output */
         printf("\ns1 = %s\t length = %d characters\n", s1, l1);
         printf("s2 = %s\t length = %d characters\n", s2, l2);
         printf("s3 = %s\t length = %d characters\n", s3, l3);
      }
Output
      Enter two string constants
      ? New York
      Strings are not equal
      s1 = NewYork  length = 7 characters
      s2 = York        length = 4 characters
      s3 = NewYork  length = 7 characters
      Enter two string constants
      ? London London
      Strings are equal
      s1 = London length = 6 characters
      s2 = London length = 6 characters
      s3 = London length = 6 characters
```

Fig. 5.25 Illustration of string handling functions

In the above program, the input strings are "New" and "York" during the first run. These strings are compared by the statement x = strcmp(s1, s2). Since they are not equal, they are joined together and copied into s3 using the statement strcpy(s3, s1). The program outputs all the three strings with their lengths.

During the second run, the two strings s1 and s2 are equal, and therefore, they are not joined together. In this case all the three strings contain the same string constant "London".

Other String Functions The header file <string.h> contains many more string manipulation functions. They might be useful in certain situations.

strncpy In addition to the function strcpy that copies one string to another, we have another function strncpy that copies only the left-most n characters of the source string to the target string variable. This is a three-parameter function and is invoked as follows:

```
strncpy(s1, s2, 5);
```

This statement copies the first 5 characters of the source string s2 into the target string s1. Since the first 5 characters may not include the terminating null character, we have to place it explicitly in the 6th position of s2 as shown below:

```
s1[6] ='\0';
```

Now, the string s1 contains a proper string.

strncmp A variation of the function strcmp is the function strncmp. This function has three parameters as illustrated in the function call below:

```
strncmp (s1, s2, n);
```

It compares the left-most n characters of s1 to s2 and returns.

- 0 if they are equal;
- negative number, if s1 sub-string is less than s2; and
- positive number, otherwise.

strncat This is another concatenation function that takes three parameters as shown below:

```
strncat (s1, s2, n);
```

This call will concatenate the left-most n characters of s2 to the end of s1. An example is shown in Figure 5.26:

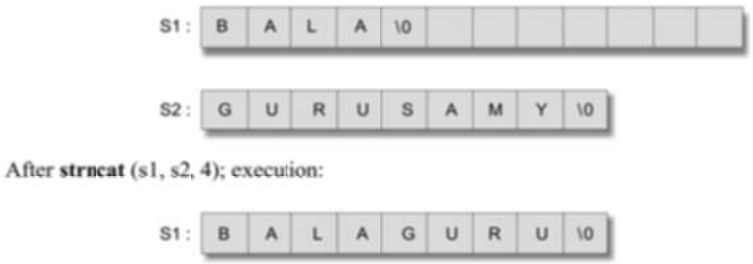


Fig. 5.26 Usage of strncat function

strstr It is a two-parameter function that can be used to locate a sub-string in a string. This takes the forms:

```
strstr (s1, s2);
strstr (s1, "ABC");
```

The function strstr searches the string s1 to see whether the string s2 is contained in s1. If yes, the function returns the position of the first occurrence of the sub-string. Otherwise, it returns a NULL pointer. Example:

```
if (strstr (s1, s2) == NULL)
    printf("substring is not found");
else
    printf("s2 is a substring of s1");
```

strchr & strrchr We also have functions to determine the existence of a character in a string. Consider the following function call:

```
strchr(s1, 'm');
```

It will locate the first occurrence of the character 'm' in string s1.

Similarly, the following function call will locate the last occurrence of the character 'm' in the string s1.

```
strrchr(s1, 'm');
```

Warnings

- When allocating space for a string during declaration, remember to count the terminating null character.
- When creating an array to hold a copy of a string variable of unknown size, we can compute the size required using the expression

  `strlen (stringname) +1.`
- When copying or concatenating one string to another, we must ensure that the target (destination) string has enough space to hold the incoming characters. Remember that no error message will be available even if this condition is not satisfied. The copying may overwrite the memory and the program may fail in an unpredictable way.
- When we use strncpy to copy a specific number of characters from a source string, we must ensure to append the null character to the target string, in case the number of characters is less than or equal to the source string.

5.4.5 Other Library Functions

In addition to string functions, C provides several other utility inbuilt functions. These functions are grouped in different sets across different header files. In total there are 24 header files in ANSI C's library. Table 5.3 gives a snapshot of the various library functions.

TABLE 5.3 Library Functions

Trigonometric Functions	
Function	**Meaning**
acos(x)	Arc cosine of x
asin(x)	Arc sine of x
atan(x)	Arc tangent of x
atan 2(x,y)	Arc tangent of x/y
cos(x)	Cosine of x
sin(x)	Sine of x
tan(x)	Tangent of x

Hyperbolic Functions	
Function	**Meaning**
cosh(x)	Hyperbolic cosine of x
sinh(x)	Hyperbolic sine of x
tanh(x)	Hyperbolic tangent of x

Character Functions	
isalnum():	Determines whether a character is an alphabet or a number.
isgraph():	Determines whether a character is any of the printable characters other than white space.
islower():	Determines whether a character is in lowercase.
isupper():	Determines whether a character is in uppercase.

Time and Date Functions
ctime(): Converts the time value into string.
time(): Returns the current time
difftime: Returns the time difference.

Miscellaneous Mathematical functions	
Function	**Meaning**
ceil(x)	x rounded up to the nearest integer
exp(x)	e to the x power (ex)
fabs(x)	Absolute value of x.
floor(x)	x rounded down to the nearest integer
fmod(x,y)	Remainder of x/y
log(x)	Natural log of x, x > 0
log10(x)	Base 10 log of x, x > 0
pow(x,y)	x to the power y (xy)
sqrt(x)	Square root of x, x > = 0

5.5 CASE STUDY

Counting Words in a Text

One of the practical applications of string manipulations is counting the words in a text. We assume that a word is a sequence of any characters, except escape characters and blanks, and that two words are separated by one blank character. The algorithm for counting words is as follows:

1. Read a line of text.
2. Beginning from the first character in the line, look for a blank. If a blank is found, increment words by 1.
3. Continue steps 1 and 2 until the last line is completed.

The implementation of this algorithm is shown in Figure 5.CS3. The first while loop will be executed once for each line of text. The end of text is indicated by pressing the 'Return' key an extra time after the entire text has been entered. The extra 'Return' key causes a newline character as input to the last line and as a result, the last line contains only the null character.

The program checks for this special line using the following test:

```
if ( line[0] == '\0')
```

If the first (and only the first) character in the line is a null character, then counting is terminated. Note the difference between a null character and a blank character.

```
Program
#include <stdio.h>
main()
{
      char line[81], ctr;
      int i,c,
            end = 0,
            characters = 0,
            words = 0,
            lines = 0;
      printf("KEY IN THE TEXT.\n");
      printf("GIVE ONE SPACE AFTER EACH WORD.\n");
      printf("WHEN COMPLETED, PRESS 'RETURN'.\n\n");
      while( end == 0)
      {
         /* Reading a line of text */
         c = 0;
         while((ctr=getchar()) != '\n')
            line[c++] = ctr;
         line[c] = '\0';
         /* counting the words in a line */
         if(line[0] == '\0')
            break ;
         else
         {
            words++;
            for(i=0; line[i] != '\0';i++)
               if(line[i] == ' ' || line[i] == '\t')
               words++;
         }
         /* counting lines and characters */
         lines = lines +1;
         characters = characters + strlen(line);
```

```
        }
        printf ("\n");
        printf("Number of lines = %d\n", lines);
        printf("Number of words = %d\n", words);
        printf("Number of characters = %d\n", characters);
}
Output
        KEY IN THE TEXT.
        GIVE ONE SPACE AFTER EACH WORD.
        WHEN COMPLETED, PRESS 'RETURN'.
        Admiration is a very short-lived passion.
        Admiration involves a glorious obliquity of vision.
        Always we like those who admire us but we do not
        like those whom we admire.
        Fools admire, but men of sense approve.
        Number of lines = 5
        Number of words = 36
        Number of characters = 205
```

Fig. 5.27 **Counting of characters, words and lines in a text**

The above program also counts the number of lines read and the total number of characters in the text. Remember, the last line containing the null string is not counted.

After the first while loop is exited, the program prints the results of counting.

5.6 USER-DEFINED FUNCTIONS

We have mentioned earlier that one of the strengths of C language is C functions. They are easy to define and use. We have used functions in every program that we have discussed so far. However, they have been primarily limited to the three functions, namely, main, printf, and scanf. In this section, we shall consider in detail the following:

- How a function is designed?
- How a function is integrated into a program?
- How two or more functions are put together? and
- How they communicate with one another?

C functions can be classified into two categories, namely, library functions and user-defined functions. The main distinction between these two categories is that library functions are not required to be written by us whereas a user-defined function has to be developed by the user at the time of writing a program. However, a user-defined function can later become a part of the C program library. In fact, this is one of the strengths of C language.

5.6.1 Need for User-defined Functions

As pointed out earlier, main is a specially recognized function in C. Every program must have a main function to indicate where the program has to begin its execution. While it is possible to code any program utilizing only main function, it leads to a number of problems.

The program may become too large and complex and as a result the task of debugging, testing, and maintaining becomes difficult.

If a program is divided into functional parts, then each part may be independently coded and later combined into a single unit. These independently coded programs are called subprograms that are much easier to understand, debug, and test. In C, such subprograms are referred to as 'functions'.

There are times when certain type of operations or calculations are repeated at many points throughout a program. For instance, we might use the factorial of a number at several points in the program. In such situations, we may repeat the program statements wherever they are needed. Another approach is to design a function that can be called and used whenever required. This saves both time and space.

This "division" approach clearly results in a number of advantages, as explained below:

- It facilitates top-down modular programming as shown in Figure 5.28. In this programming style, the high level logic of the overall problem is solved first while the details of each lower-level function are addressed later.
- The length of a source program can be reduced by using functions at appropriate places. This factor is particularly critical with microcomputers where memory space is limited.
- It is easy to locate and isolate a faulty function for further investigations.
- A function may be used by many other programs. This means that a C programmer can build on what others have already done, instead of starting all over again from scratch.

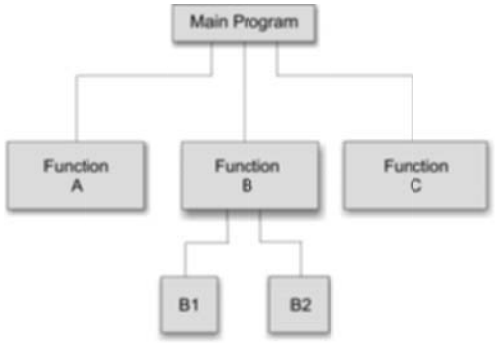


Fig. 5.28 Top-down modular programming using functions

5.6.2 A Multi-function Program

A function is a self-contained block of code that performs a particular task. Once a function has been designed and packed, it can be treated as a 'black box' that takes some data from the main program and returns a value. The inner details of operation are invisible to the rest of the

program. All that the program knows about a function is: What goes in and what comes out. Every C program can be designed using a collection of these black boxes known as functions.

Consider a set of statements as shown below:

```
void printline(void)
{
      int i;
      for (i=1; i<40; i++)
      printf("–");
      printf("\n");
}
```

The above set of statements define a function called printline, which could print a line of 39-character length. This function can be used in a program as follows:

```
void printline(void);  /* declaration*/
 main( )
 {
      printline( );
      printf("This illustrates the use of C functions\n");
      printline();
 }
 void printline(void)
 {
       int i;
       for(i=1; i<40; i++)
       printf("–");
       printf("\n");
 }
```

This program will print the following output:

This illustrates the use of C functions

In the above program, during the execution of the main, the first statement encountered is

```
printline( );
```

This indicates that the function printline is to be executed. At this point, the program control is transferred to the function printline. After executing the printline function, which outputs a line of 39 character length, the control is transferred back to the main. Now, the execution continues at the point where the function call was executed. After executing the printf statement, the control is again transferred to the printline function for printing the line once more.

The main function calls the user-defined printline function two times and the library function printf once. We may notice that the printline function itself calls the library function printf 39 times repeatedly.

Any function can call any other function. In fact, it can call itself. A 'called function' can also call another function. A function can be called more than once. In fact, this is one of the main features of using functions. Figure 5.29 illustrates the flow of control in a multi-function program.

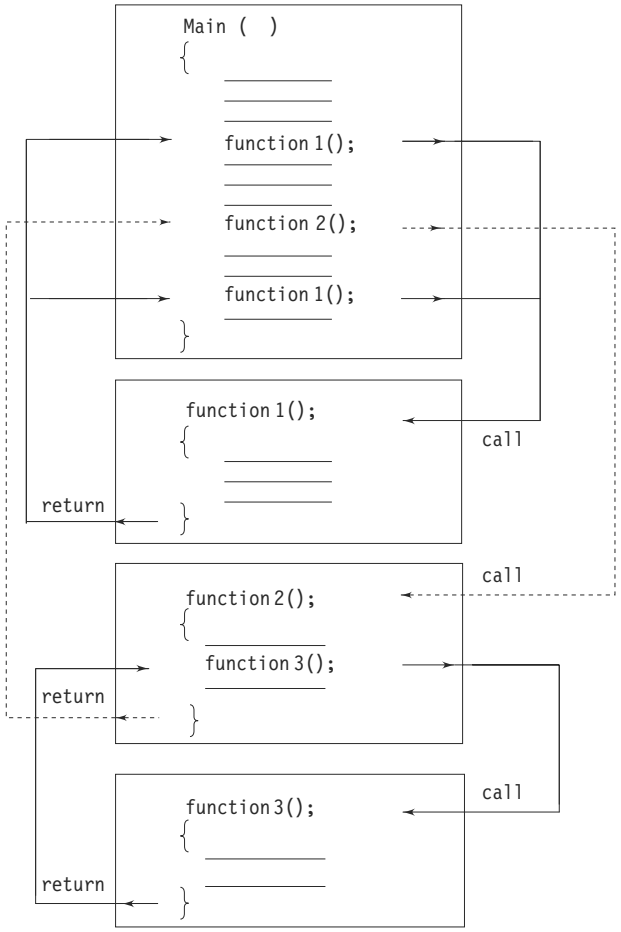


Fig. 5.29 Flow of control in a multi-function program

Except the starting point, there are no other predetermined relationships, rules of precedence, or hierarchies among the functions that make up a complete program. The functions can be placed in any order. A called function can be placed either before or after the calling function. However, it is the usual practice to put all the called functions at the end.

Modular Programming

Modular programming is a strategy applied to the design and development of software systems. It is defined as organizing a large program into small, independent program segments called modules that are separately named and individually callable program units. These modules are carefully integrated to become a software system that satisfies the system requirements. It is basically a "divide-and-conquer" approach to problem solving.

Modules are identified and designed such that they can be organized into a top-down hierarchical structure (similar to an organization chart). In C, each module refers to a function that is responsible for a single task.

Some characteristics of modular programming are:

- Each module should do only one thing.
- Communication between modules is allowed only by a calling module.
- A module can be called by one and only one higher module.
- No communication can take place directly between modules that do not have calling-called relationship.
- All modules are designed as single-entry, single-exit systems using control structures.

5.6.3 Elements of User-Defined Functions

We have discussed and used a variety of data types and variables in our programs so far. However, declaration and use of these variables were primarily done inside the main function. Functions are mainly classified as one of the derived data types in C. We can therefore define functions and use them like any other variables in C programs. It is therefore not a surprise to note that there exist some similarities between functions and variables in C.

- Both function names and variable names are considered identifiers and therefore they must adhere to the rules for identifiers.
- Like variables, functions have types (such as int) associated with them.
- Like variables, function names and their types must be declared and defined before they are used in a program.

In order to make use of a user-defined function, we need to establish three elements that are related to functions.

1. Function definition.
2. Function call.
3. Function declaration.

The function definition is an independent program module that is specially written to implement the requirements of the function. In order to use this function we need to invoke it at a required place in the program. This is known as the function call. The program (or a function) that calls the function is referred to as the calling program or calling function. The calling program should declare any function (like declaration of a variable) that is to be used later in the program. This is known as the function declaration or function prototype.

Definition of Functions A function definition, also known as function implementation shall include the following elements;

1. function name;
2. function type;
3. list of parameters;
4. local variable declarations;
5. function statements; and
6. a return statement.

All the six elements are grouped into two parts, namely,

- function header (First three elements); and
- function body (Second three elements).

A general format of a function definition to implement these two parts is given below:

```
function_type function_name(parameter list)
{
      local variable declaration;
      executable statement1;
      executable statement2;
      . . . . .
      . . . . .
      return statement;
}
```

The first line function_type function_name(parameter list) is known as the function header and the statements within the opening and closing braces constitute the function body, which is a compound statement.

Function Header The function header consists of three parts: the function type (also known as return type), the function name and the formal parameter list. Note that a semicolon is not used at the end of the function header.

Name and Type The function type specifies the type of value (like float or double) that the function is expected to return to the program calling the function. If the return type is not explicitly specified, C will assume that it is an integer type. If the function is not returning anything, then we need to specify the return type as void. Remember, void is one of the fundamental data types in C. It is a good programming practice to code explicitly the return type, even when it is an integer. The value returned is the output produced by the function.

The function name is any valid C identifier and therefore must follow the same rules of formation as other variable names in C. The name should be appropriate to the task performed by the function. However, care must be exercised to avoid duplicating library routine names or operating system commands.

Formal Parameter List The parameter list declares the variables that will receive the data sent by the calling program. They serve as input data to the function to carry out the specified

task. Since they represent actual input values, they are often referred to as formal parameters. The parameters are also known as arguments.

The parameter list contains declaration of variables separated by commas and surrounded by parentheses. Examples:

```
float quadratic (int a, int b, int c) {. . . . }
double power (double x, int n) {. . . ..}
float mul (float x, float y) {. . . . }
int sum (int a, int b) {. . . . }
```

Remember, there is no semicolon after the closing parentheses. Note that the declaration of parameter variables cannot be combined. That is, int sum (int a,b) is illegal.

A function need not always receive values from the calling program. In such cases, functions have no formal parameters. To indicate that the parameter list is empty, we use the keyword void between the parentheses as shows under:

```
void printline (void)
{
     . . . .
}
```

This function neither receives any input values nor returns back any value.

Function Body The function body contains the declarations and statements necessary for performing the required tasks. The body enclosed in braces, contains three parts, in the order given below:

1. Local declarations that specify the variables needed by the function.
2. Function statements that perform the task of the function.
3. A return statement that returns the value evaluated by the function.

If a function does not return any value (like the printline function), we can omit the return statement. However, note that its return type should be specified as void. Again, it is nice to have a return statement even for void functions.

Some examples of typical function definitions are shown below:

```
(a)  float mul (float x, float y)
     {
          float result;          /* local variable */
          result = x * y;          /* computes the product */
          return (result);           /* returns the result */
     }
(b)  void sum (int a, int b)
     {
          printf ("sum = %s", a + b);    /* no local variables */
          return;             /* optional */
     }
```

```
(c)   void display (void)
      {               /* no local variables */
              printf ("No type, no parameters");
                      /* no return statement */
      }
```

NOTE:

1. When a function reaches its return statement, the control is transferred back to the calling program. In the absence of a return statement, the closing brace acts as a void return.
2. A local variable is a variable that is defined inside a function and used without having any role in the communication between functions.

Return Values and their Types As pointed out earlier, a function may or may not send back any value to the calling function. If it does, it is done through the return statement. While it is possible to pass to the called function any number of values, the called function can only return one value per call, at the most.

The return statement can take one of the following forms:

```
return;
return(expression);
```

The first, the 'plain' return does not return any value; it acts much as the closing brace of the function. When a return is encountered, the control is immediately passed back to the calling function. An example of the use of a simple return is as follows:

```
if(error)
return;
```

The second form of return with an expression returns the value of the expression. For example, consider the following function:

```
int mul (int x, int y)
{
int p;
      p = x*y;
      return(p);
}
```

It returns the value of p which is the product of the values of x and y. The last two statements can be combined into one statement as follows:

```
return (x*y);
```

A function may have more than one return statements. This situation arises when the value returned is based on certain conditions. For example:

```
if( x <= 0 )
return(0);
else
return(1);
```

What type of data does a function return? All functions by default return int type data. But what happens if a function must return some other type? We can force a function to return a particular type of data by using a type specifier in the function header as discussed earlier.

When a value is returned, it is automatically cast to the function's type. In functions that do computations using doubles, yet return ints, the returned value will be truncated to an integer. For instance, consider the following function:

```
int product (void)
{
return (2.5 * 3.0);
}
```

It will return the value 7, only the integer part of the result.

Function Calls A function can be called by simply using the function name followed by a list of actual parameters (or arguments), if any, enclosed in parentheses. Example:

```
main( )
{
      int y;
      y = mul(10,5);      /* Function call */
      printf("%d\n", y);
}
```

When the compiler encounters a function call, the control is transferred to the function mul(). This function is then executed line by line as described and a value is returned when a return statement is encountered. This value is assigned to y. This is illustrated Figure 5.30:

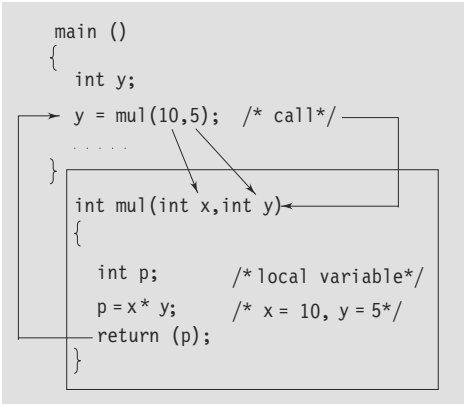


Fig. 5.30 Illustration of a function call

The function call sends two integer values 10 and 5 to the function int mul(int x, int y) which are assigned to x and y respectively. The function computes the product x and y, assigns the result to the local variable p, and then returns the value 25 to the main where it is assigned to y again.

There are many different ways to call a function. Listed below are some of the ways the function mul can be invoked.

```
mul (10, 5)
mul (m, 5)
mul (10, n)
mul (m, n)
mul (m + 5, 10)
mul (10, mul(m,n))
mul (expression1, expression2)
```

Note that the sixth call uses its own call as its one of the parameters. When we use expressions, they should be evaluated to single values that can be passed as actual parameters.

A function which returns a value can be used in expressions like any other variable. Thus, each of the following statements is valid:

```
printf("%d\n", mul(p,q));
y = mul(p,q) / (p+q);
if (mul(m,n)>total) printf("large");
```

However, a function cannot be used on the right side of an assignment statement. For instance, the following statement is invalid:

```
mul(a,b) = 15;
```

A function that does not return any value may not be used in expressions; but can be called in to perform certain tasks specified in the function. The function printline( ) discussed earlier belongs to this category. Such functions may be called in by simply stating their names as independent statements. Example:

```
main( )
{
      printline( );
}
```

Note the presence of a semicolon at the end.

Function Call

A function call is a postfix expression. The operator (. .) is at a very high level of precedence. Therefore, when a function call is used as a part of an expression, it will be evaluated first, unless parentheses are used to change the order of precedence.

In a function call, the function name is the operand and the parentheses set (. .) which contains the actual parameters is the operator. The actual parameters must match the function's formal parameters in type, order and number. Multiple actual parameters must be separated by commas.

Note:

- If the actual parameters are more than the formal parameters, the extra actual arguments will be discarded.
- On the other hand, if the actuals are less than the formals, the unmatched formal arguments will be initialized to some garbage.
- Any mismatch in data types may also result in some garbage values.

Function Declaration Like variables, all functions in a C program must be declared, before they are invoked. A function declaration (also known as function prototype) consists of four parts.

- Function type (return type).
- Function name.
- Parameter list.
- Terminating semicolon.

They are coded in the following format:

```
Function-type function-name (parameter list);
```

This is very similar to the function header line except the terminating semicolon. For example, mul function defined in the previous section will be declared as under:

```
int mul (int m, int n); /* Function prototype */
```

It is important to note the following with regards to function declaration:

- The parameter list must be separated by commas.
- The parameter names do not need to be the same in the prototype declaration and the function definition.
- The types must match the types of parameters in the function definition, in number and order.
- Use of parameter names in the declaration is optional.
- If the function has no formal parameters, the list is written as (void).
- When the declared types do not match with the types in the function definition, compiler will produce an error.

Equally acceptable forms of declaration of mul function are:

```
int  mul  (int, int);
mul  (int a, int b);
mul  (int, int);
```

When a function does not take any parameters and does not return any value, its prototype is written as:

```
void display (void);
```

A prototype declaration may be placed in two places in a program.

- **Above all the functions (including main)** When we place the declaration above all the functions (in the global declaration section), the prototype is referred to as a global prototype. Such declarations are available for all the functions in the program.
- **Inside a function definition** When we place it in a function definition (in the local declaration section), the prototype is called a local prototype. Such declarations are primarily used by the functions containing them.

Thus, the place of declaration of a function defines a region in a program in which the function may be used by other functions. This region is known as the scope of the function. It is a good programming style to declare prototypes in the global declaration section before main. It adds flexibility, provides an excellent quick reference to the functions used in the program, and enhances documentation.

Parameters Everywhere!

Parameters (also known as arguments) are used in three places:

1. in declaration (prototypes),
2. in function call, and
3. in function definition.

The parameters used in prototypes and function definitions are called formal parameters and those used in function calls are called actual parameters. Actual parameters used in a calling statement may be simple constants, variables or expressions.

The formal and actual parameters must match exactly in type, order and number. Their names, however, do not need to match.

Prototypes: Yes or No

Prototype declarations are not essential. If a function has not been declared before it is used, C will assume that its details available at the time of linking. Since the prototype is not available, C will assume that the return type is an integer and that the types of parameters match the formal definitions. If these assumptions are wrong, the linker will fail and we will have to change the program. The moral is that we must always include prototype declarations, preferably in global declaration section.

5.6.4 Nesting of Functions

C permits nesting of functions freely. main can call function1, which calls function2, which calls function3, ……. and so on. There is in principle no limit as to how deeply functions can be nested. Consider the following program:

```
float ratio (int x, int y, int z);
int difference (int x, int y);
```

```
main( )
{
      int a, b, c;
      scanf("%d %d %d", &a, &b, &c);
      printf("%f \n", ratio(a,b,c));
}
float ratio(int x, int y, int z)
{
      if(difference(y, z))
         return(x/(y-z));
      else
         return(0.0);
}
int difference(int p, int q)
{
      if(p != q)
       return (1);
      else
       return(0);
}
```

The above program calculates the ratio $\frac{a}{b-c}$ and prints the result. Here, we have the following three functions in the program:

main( )

ratio( )

difference( )

main reads the values of a, b and c and calls the function ratio to calculate the value a/(b–c). This ratio cannot be evaluated if (b–c) = 0. Therefore, ratio calls another function difference to test whether the difference (b–c) is zero or not; difference returns 1, if b is not equal to c; otherwise returns zero to the function ratio.

In turn, ratio calculates the value a/(b–c) if it receives 1 and returns the result in float. In case, ratio receives zero from difference, it sends back 0.0 to main indicating that (b–c) = 0.

Nesting of function calls is also possible. For example, the following statement is valid in C:

```
P = mul(mul(5,2),6);
```

This represents two sequential function calls. The inner function call is evaluated first and the returned value is again used as an actual argument in the outer function call. If mul returns the product of its arguments, then the value of p would be 60 (= 5X2X6).

 **NOTE:** Nesting does not mean defining one function within another. Doing this is illegal!

5.6.5 Recursion

When a called function in turn calls another function a process of 'chaining' occurs. Recursion is a special case of this process, where a function calls itself. A very simple example of recursion is presented below:

```
main( )
{
printf("This is an example of recursion\n")
main( );
}
```

When executed, this program will produce an output something like this:

```
This is an example of recursion
This is an example of recursion
This is an example of recursion
This is an ex
```

Here, the execution is terminated abruptly; otherwise the execution will continue indefinitely.

Another useful example of recursion is the evaluation of factorials of a given number. The factorial of a number n is expressed as a series of repetitive multiplications as shown below:

factorial of n = n(n–1)(n–2).....1

For example:

factorial of 4 = 4 X 3 X 2 X 1 = 24

EXAMPLE 5.12 *Figure 5.31 shows an example of recursion for calculating factorial of an integer number:*

```
Program
#include<stdio.h>
#include<conio.h>
long double factorial(long double );
void main()
{
 long double x,y;
 clrscr();
 printf("\nEnter a number: ");
 scanf(" %Lf", &x);
 y = factorial(x);
 printf("\nThe factorial of %2.1Lf, is %2.1Lf \n", x,y);
 getch();
}
```

```
long double factorial(long double n)
{
 long double fact;
 if( n == 0 )
 return(1);
 else
 fact = n*factorial(n - 1);
 return(fact);
}
Output
Enter a number: 15
The factorial of 15 is 1307674368000
Enter a number: 100
The factorial of 100 is equal to
9.33262154439441526000000000000000000000000e+157
```

Fig. 5.31 Program to calculate factorial of a number

Recursive functions can be effectively used to solve problems where solution is expressed in terms of successively applying the same solution to subsets of the problem. When we write recursive functions, we must have an if statement somewhere to force the function to return without the recursive call being executed. Otherwise, the function will never return.

5.6.6 Tower of Hanoi

The tower of Hanoi problem is a game, which comprises of three towers with discs stacked around the first tower initially. The game involves shifting the disks across the adjoining towers. There are some conditions that need to be adhered during the shifting:

1. A larger disk can not be placed on a smaller disk
2. Only one disc can be shifted at a time.

The end objective is to replicate the initial stack of discs into another tower.

Figure 5.32 shows a C program to realize the tower of Hanoi problem.

```
Program
#include<conio.h>
#include<stdio.h>

void tower_hanoi( int num_disk_f,char tower1, char tower2, char tower3)
{
 if ( num_disk_f == 1 )
 {
      printf( "\nShift uppermost disk from tower %c to tower %c.", tower1, tower2 );
      return;
 }
```

```
 tower_hanoi( num_disk_f - 1,tower1, tower3, tower2 );
 printf( "\nShift uppermost disk from tower %c to tower %c.", tower1, tower2 );
 tower_hanoi( num_disk_f - 1,tower3, tower2, tower1 );
}
void main()
{
 int num_of_disk;
 clrscr();
 printf("\nEnter the number of disks: ");
 scanf("%d",&num_of_disk);
 if(num_of_disk<1)
 printf("\n There are no disks to shift");
 else
 printf("\nThere are %d disks in tower 1\n ",num_of_disk);
 tower_hanoi(num_of_disk,'1','2','3');
 printf("\n\nThe %d disks in tower 1 are shifted in tower 2",num_of_disk);
 getch();
}
Output
Enter the number of disks: 4
There are 4 disks in tower 1
Shift uppermost disk from tower 1 to tower 3.
Shift uppermost disk from tower 1 to tower 2.
Shift uppermost disk from tower 3 to tower 2.
Shift uppermost disk from tower 1 to tower 3.
Shift uppermost disk from tower 2 to tower 1.
Shift uppermost disk from tower 2 to tower 3.
Shift uppermost disk from tower 1 to tower 3.
Shift uppermost disk from tower 1 to tower 2.
Shift uppermost disk from tower 3 to tower 2.
Shift uppermost disk from tower 3 to tower 1.
Shift uppermost disk from tower 2 to tower 1.
Shift uppermost disk from tower 3 to tower 2.
Shift uppermost disk from tower 1 to tower 3.
Shift uppermost disk from tower 1 to tower 2.
Shift uppermost disk from tower 3 to tower 2.
The 4 disks in tower 1 are shifted in tower 2
```

Fig. 5.32 **Tower of Hanoi**

5.6.7 Passing Arrays to Functions

One-Dimensional Arrays Like the values of simple variables, it is also possible to pass the values of an array to a function. To pass a one-dimensional an array to a called function,

it is sufficient to list the name of the array, without any subscripts, and the size of the array as arguments. For example, consider the following function call:

```
largest(a,n)
```

This will pass the whole array a to the called function. The called function expecting this call must be appropriately defined. The largest function header might look like:

```
float largest(float array[ ], int size)
```

The function largest is defined to take two arguments, the array name and the size of the array to specify the number of elements in the array. The declaration of the formal argument array is made as follows:

```
float array[ ];
```

The pair of brackets informs the compiler that the argument array is an array of numbers. It is not necessary to specify the size of the array here.

EXAMPLE 5.13 *Figure 5.33 shows a program to find the largest value in an array of elements.*

```
main( )
{
        float largest(float a[ ], int n);
        float value[4] = {2.5,-4.75,1.2,3.67};
        printf("%f\n", largest(value,4));
}
float largest(float a[], int n)
{
        int i;
        float max;
        max = a[0];
        for(i = 1; i < n; i++)
              if(max < a[i])
              max = a[i];
        return(max);
}
```

Fig. 5.33 Program to calculate the largest array element

When the function call largest(value,4) is made, the values of all elements of array value become the corresponding elements of array a in the called function. The largest function finds the largest value in the array and returns the result to the main.

It is important to note the following while passing an array to a function:

- The function must be called by passing only the name of the array.
- In the function definition, the formal parameter must be an array type; the size of the array does not need to be specified.
- The function prototype must show that the argument is an array.

When dealing with array arguments, we should remember one major distinction. If a function changes the values of the elements of an array, then these changes will be made to the original array that was passed to the function. When an entire array is passed as an argument, the contents of the array are not copied into the formal parameter array; instead, information about the addresses of array elements are passed on to the function.

Therefore, any changes introduced to the array elements are truly reflected in the original array in the calling function. However, this does not apply when an individual element is passed on as argument. The next example illustrates this point.

EXAMPLE 5.14 *Figure 5.34 shows a program to sort an array of integers.*

```
Program
      void sort(int m, int x[ ]);
      main()
      {
         int i;
         int marks[5] = {40, 90, 73, 81, 35};

         printf("Marks before sorting\n");
         for(i = 0; i < 5; i++)
            printf("%d ", marks[i]);
         printf("\n\n");

         sort (5, marks);
         printf("Marks after sorting\n");
         for(i = 0; i < 5; i++)
            printf("%4d", marks[i]);
         printf("\n");
      }
      void sort(int m, int x[ ])
      {
         int i, j, t;

         for(i = 1; i <= m-1; i++)
         for(j = 1; j <= m-i; j++)
         if(x[j-1] >= x[j])
            {
               t = x[j-1];
               x[j-1] = x[j];
               x[j] = t;
            }
      }
Output
         Marks before sorting
         40 90 73 81 35

         Marks after sorting
         35 40 73 81 90
```

Fig. 5.34 Sorting of array elements using a function

The above output clearly shows that a function can change the values in an array passed as an argument.

Two-Dimensional Arrays Like one-dimensional arrays, we can also pass multi-dimensional arrays to functions. The approach is similar to the one we did with one-dimensional arrays. The rules are simple, as listed below:

- The function must be called by passing only the array name.
- In the function definition, we must indicate that the array has two-dimensions by including two sets of brackets.
- The size of the second dimension must be specified.
- The prototype declaration should be similar to the function header.

The function given below calculates the average of the values in a two-dimensional matrix.

```
double average(int x[][N], int M, int N)
{
        int i, j;
        double sum = 0.0;
        for (i=0; i<M; i++)
              for(j=1; j<N; j++)
             sum += x[i][j];
        return(sum/(M*N));
}
```

This function can be used in a main function as illustrated below:

```
main( )
{
 int M=3, N=2;
 double average(int [ ] [N], int, int);
 double mean;
 int matrix [M][N]=
    {
        {1,2},
        {3,4},
        {5,6}
    };
 mean = average(matrix, M, N);
 . . . . . .
 . . . . . .
}
```

5.6.8 Passing Strings to Functions

The strings are treated as character arrays in C and therefore the rules for passing strings to functions are very much similar to those for passing arrays to functions. The basic rules are:

The string to be passed must be declared as a formal argument of the function when it is defined. Example:

```
void display(char item_name[ ])
     {
        . . . . . .
        . . . . . .
     }
```

The function prototype must show that the argument is a string. For the above function definition, the prototype can be written as:

```
void display(char str[ ]);
```

A call to the function must have a string array name without subscripts as its actual argument. Example:

```
display (names);
```

Here, names is a properly declared string array in the calling function.

 **NOTE:** Like arrays, strings in C cannot be passed by value to functions.

5.6.9 The Scope, Visibility and Lifetime of Variables

Variables in C differ in behaviour from those in most other languages. For example, in a BASIC program, a variable retains its value throughout the program. It is not always the case in C. It all depends on the 'storage' class a variable may assume.

In C not only do all variables have a data type, they also have a storage class. The following variable storage classes are most relevant to functions:

Automatic variables.

External variables.

Static variables.

Register variables.

We shall briefly discuss the scope, visibility and longevity of each of the above class of variables. The scope of variable determines over what region of the program a variable is actually available for use ('active'). Longevity refers to the period during which a variable retains a given value during execution of a program ('alive'). So longevity has a direct effect on the utility of a given variable. The visibility refers to the accessibility of a variable from the memory.

The variables may also be broadly categorized, depending on the place of their declaration, as internal (local) or external (global). Internal variables are those which are declared within a particular function, while external variables are declared outside of any function.

It is very important to understand the concept of storage classes and their utility in order to develop efficient multifunction programs.

Automatic Variables Automatic variables are declared inside a function in which they are to be utilized. They are created when the function is called and destroyed automatically when

the function is exited, hence the name automatic. Automatic variables are therefore private (or local) to the function in which they are declared. Because of this property, automatic variables are also referred to as local or internal variables.

A variable declared inside a function without storage class specification is, by default, an automatic variable. For instance, the storage class of the variable number in the example below is automatic.

```
main( )
{
        int number;
        _____
        _____
}
```

We may also use the keyword auto to declare automatic variables explicitly, as shown below:

```
main( )
{
        auto int number;
        _____
        _____
}
```

One important feature of automatic variables is that their value cannot be changed accidentally by what happens in some other function in the program. This assures that we may declare and use the same variable name in different functions in the same program without causing any confusion to the compiler.

EXAMPLE 5.15 *Write a multifunction to illustrate how automatic variables work.*

A program with two subprograms function1 and function2 is shown in Figure 5.35 where m is an automatic variable and it is declared at the beginning of each function. m is initialized to 10, 100, and 1000 in function1, function2, and main respectively.

```
Program
      void function1(void);
      void function2(void);
      main( )
      {
            int m = 1000;
            function2();

            printf("%d\n",m);   /* Third output */
      }
       void function1(void)
      {
            int m = 10;
```

```
            printf("%d\n",m);   /* First output */
      }
      void function2(void)
      {
            int m = 100;
            function1();
            printf("%d\n",m);   /* Second output */
      }
 Output
      10
      100
      1000
```

Fig. 5.35 Working of automatic variables

When executed, main calls function2 which in turn calls function1. When main is active, m = 1000; but when function2 is called, the main's m is temporarily put on the shelf and the new local m = 100 becomes active. Similarly, when function1 is called, both the previous values of m are put on the shelf and the latest value of m (=10) becomes active. As soon as function1 (m=10) is finished, function2 (m=100) takes over again. As soon it is done, main (m=1000) takes over. The output clearly shows that the value assigned to m in one function does not affect its value in the other functions; and the local value of m is destroyed when it leaves a function.

There are two consequences of the scope and longevity of auto variables worth remembering. First, any variable local to main will be normally alive throughout the whole program, although it is active only in main. Secondly, during recursion, the nested variables are unique auto variables, a situation similar to function-nested auto variables with identical names.

External Variables Variables that are both alive and active throughout the entire program are known as external variables. They are also known as global variables. Unlike local variables, global variables can be accessed by any function in the program. External variables are declared outside a function. For example, the external declaration of integer number and float length might appear as:

```
int number;
float length = 7.5;
main( )
{
 _______
 _______
}
function1( )
{
 _______

 _______
```

```
}
function2( )
{
 _______
 _______
}
```

The variables number and length are available for use in all the three functions. In case a local variable and a global variable have the same name, the local variable will have precedence over the global one in the function where it is declared. Consider the following example:

```
int count;
main( )
{
 count = 10;
 _____
 _____
}
function( )
{
 int count = 0;
 ______
 ______
 count = count+1;
}
```

Here, when the function references the variable count, it will be referencing only its local variable, not the global one. The value of count in main will not be affected.

EXAMPLE 5.16 *Write a multifunction program to illustrate the properties of global variables.*

A program to illustrate the properties of global variables is presented in Figure 5.36.

```
Program
      int fun1(void);
      int fun2(void);
      int fun3(void);
      int x ;       /* global */
      main( )
      {
         x = 10 ;      /* global x */
         printf("x = %d\n", x);
         printf("x = %d\n", fun1());
         printf("x = %d\n", fun2());
         printf("x = %d\n", fun3());
      }
      fun1(void)
      {
           x = x + 10 ;
      }
      int fun2(void)
      {
           int x ;          /* local */
```

```
            x = 1 ;
            return (x);
      }
      fun3(void)
      {
            x = x + 10 ;      /* global x */
      }
 Output
      x = 10
      x = 20
      x = 1
      x = 30
```

Fig. 5.36 Illustration of properties of global variables

Note that variable x is used in all functions but none except fun2, has a definition for x. Because x has been declared 'above' all the functions, it is available to each function without having to pass x as a function argument. Further, since the value of x is directly available, we need not use return(x) statements in fun1 and fun3. However, since fun2 has a definition of x, it returns its local value of x and therefore uses a return statement. In fun2, the global x is not visible. The local x hides its visibility here.

Once a variable has been declared as global, any function can use it and change its value. Then, subsequent functions can reference only that new value.

Global Variables as Parameters

Since all functions in a program source file can access global variables, they can be used for passing values between the functions. However, using global variables as parameters for passing values poses certain problems.

The values of global variables which are sent to the called function may be changed inadvertently by the called function.

Functions are supposed to be independent and isolated modules. This character is lost, if they use global variables.

It is not immediately apparent to the reader which values are being sent to the called function.

A function that uses global variables suffers from reusability.

One other aspect of a global variable is that it is available only from the point of declaration to the end of the program. Consider a program segment as shown below:

```
main( )
{
        y = 5;
        . . . .
        . . . .
}
int y;        /* global declaration */
func1( )
{
          y = y+1;
}
```

We have a problem here. As far as main is concerned, y is not defined. So, the compiler will issue an error message. Unlike local variables, global variables are initialized to zero by default. Thus the following statement in fun1 will, therefore, assign 1 to y.

```
y = y+1;
```

External Declaration In the program segment above, the main cannot access the variable y as it has been declared after the main function. This problem can be solved by declaring the variable with the storage class extern.

Example:

```
main( )
{
        extern int y;    /* external declaration */
        . . . . .
        . . . . .
}
func1( )
{
        extern int y;    /* external declaration */
        . . . . .
        . . . . .
}
int y;                   /* definition */
```

Although the variable y has been defined after both the functions, the external declaration of y inside the functions informs the compiler that y is an integer type defined somewhere else in the program. Note that extern declaration does not allocate storage space for variables. In case of arrays, the definition should include their size as well.

Example:

```
main( )
{       int i;
        void print_out(void);
        extern float height [ ];
        . . . . .
        . . . . .
        print_out( );
}
void print_out(void)
{
        extern float height [ ];
        int i;
        . . . . .
        . . . . .
}
float height[SIZE];
```

An extern within a function provides the type information to just that one function. We can provide type information to all functions within a file by placing external declarations before any of them.

Example:

```
extern float height[ ];
main( )
{
        int i;
        void print_out(void);
        . . . . .
        . . . . .
        print_out( );
}
void print_out(void)
{
        int i;
        . . . . .
        . . . . .
}
float height[SIZE];
```

The distinction between definition and declaration also applies to functions. A function is defined when its parameters and function body are specified. This tells the compiler to allocate space for the function code and provides type information for the parameters. Since functions are external by default, we declare them (in the calling functions) without the qualifier extern. Therefore, the following two declarations are equivalent:

```
void print_out(void);
extern void print_out(void);
```

Function declarations outside of any function behave the same way as variable declarations.

Static Variables As the name suggests, the value of static variables persists until the end of the program. A variable can be declared static using the keyword static like

```
static int x;
static float y;
```

A static variable may be either an internal type or an external type depending on the place of declaration.

Internal static variables are those which are declared inside a function. The scope of internal static variables extend up to the end of the function in which they are defined. Therefore, internal static variables are similar to auto variables, except that they remain in existence (alive) throughout the remainder of the program. Therefore, internal static variables can be used to retain values between function calls. For example, it can be used to count the number of calls made to a function.

EXAMPLE 5.17 *Write a program to illustrate the properties of a static variable.*

The program in Figure 5.37 explains the behavior of a static variable.

```
Program
      void stat(void);
      main ( )
      {
         int i;
         for(i=1; i<=3; i++)
         stat( );
      }
      void stat(void)
      {
         static int x = 0;

         x = x+1;
         printf("x = %d\n", x);
      }
Output
      x = 1
      x = 2
      x = 3
```

Fig. 5.37 Illustration of static variable

A static variable is initialized only once, when the program is compiled. It is never initialized again. During the first call to stat, x is incremented to 1. Because x is static, this value persists and therefore, the next call adds another 1 to x giving it a value of 2. The value of x becomes three when the third call is made.

Had we declared x as an auto variable, the output would have been:

```
x = 1
x = 1
x = 1
```

This is because each time stat is called; the auto variable x is initialized to zero. When the function terminates, its value of 1 is lost.

An external static variable is declared outside of all functions and is available to all the functions in that program. The difference between a static external variable and a simple external variable is that the static external variable is available only within the file where it is defined while the simple external variable can be accessed by other files.

It is also possible to control the scope of a function. For example, we would like a particular function accessible only to the functions in the file in which it is defined, and not to any function in other files. This can be accomplished by defining 'that' function with the storage class static.

Register Variables We can tell the compiler that a variable should be kept in one of the machine's registers, instead of keeping in the memory (where normal variables are stored).

Since a register access is much faster than a memory access, keeping the frequently accessed variables (e.g. loop control variables) in the register will lead to faster execution of programs. This is done as follows:

```
register int count;
```

NOTE: Although, ANSI standard does not restrict its application to any particular data type, most compilers allow only int or char variables to be placed in the register.

Since only a few variables can be placed in the register, it is important to carefully select the variables for this purpose. However, C will automatically convert register variables into non-register variables once the limit is reached.

Table 5.4 summarizes the information on the visibility and lifetime of variables in functions and files.

TABLE 5.4 Scope and Lifetime of Variables

Storage Class	Where declared	Visibility (Active)	Lifetime (Alive)
None	Before all functions in a file (may be initialized)	Entire file plus other files where variable is declared with extern	Entire program (Global)
extern	Before all functions in a file (cannot be initialized) extern and the file where originally declared as global.	Entire file plus other files where variable is declared	Global
static	Before all functions in a file	Only in that file	Global
None or auto	Inside a function (or a block)	Only in that function or block	Until end of function or block
register	Inside a function or block	Only in that function or block	Until end of function or block
static	Inside a function	Only in that function	Global

Nested Blocks A set of statements enclosed in a set of braces is known a block or a compound statement. Note that all functions including the main use compound statement. A block can have its own declarations and other statements. It is also possible to have a block of such statements inside the body of a function or another block, thus creating what is known as nested blocks as shown in Figure 5.38:

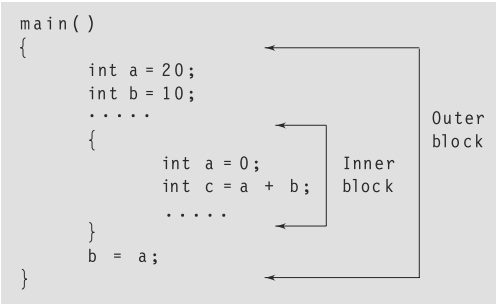


Fig. 5.38 **Nested Blocks**

When this program is executed, the value c will be 10, not 30. The statement b = a; assigns a value of 20 to b and not zero. Although the scope of a extends up to the end of main it is not "visible" inside the inner block where the variable a has been declared again. The inner a hides the visibility of the outer a in the inner block. However, when we leave the inner block, the inner a is no longer in scope and the outer a becomes visible again.

Remember, the variable b is not re-declared in the inner block and therefore it is visible in both the blocks. That is why when the statement int c = a + b; is evaluated, a assumes a values of 0 and b assumes a value of 10.

Although main's variables are visible inside the nested block, the reverse is not true.

5.6.10 Pass by Value versus Pass by Pointers

The technique used to pass data from one function to another is known as parameter passing. Parameter passing can be done in two ways:

- Pass by value (also known as call by value).
- Pass by pointers (also known as call by pointers).

In pass by value, values of actual parameters are copied to the variables in the parameter list of the called function. The called function works on the copy and not on the original values of the actual parameters. This ensures that the original data in the calling function cannot be changed accidentally.

In pass by pointers (also known as pass by address), the memory addresses of the variables rather than the copies of values are sent to the called function. In this case, the called function directly works on the data in the calling function and the changed values are available in the calling function for its use.

Pass by pointers method is often used when manipulating arrays and strings. This method is also used when we require multiple values to be returned by the called function. This is covered in more detail later in the chapter.

5.7 CASE STUDY

Calculation of Area under a Curve

One of the applications of computers in numerical analysis is computing the area under a curve. One simple method of calculating the area under a curve is to divide the area into a number of trapezoids of same width and summing up the area of individual trapezoids. The area of a trapezoid is given by

$$\text{Area} = 0.5 * (h1 + h2) * b$$

Here, h1 and h2 are the heights of two sides and b is the width as shown in Figure 5.39.

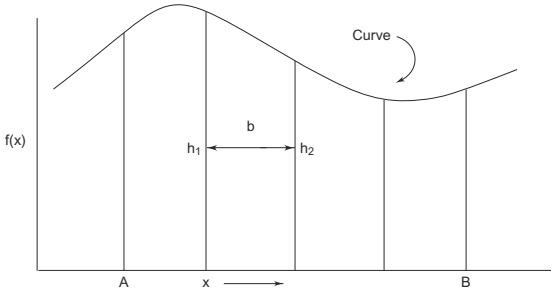


Fig.5.39 Area under a curve

The program in Figure 5.41 calculates the area for a curve of the function

$f(x) = x^2 + 1$, between any two given limits, say, A and B.

Input

Lower limit (A)

Upper limit (B)

Number of trapezoids

Output

Total area under the curve between the given limits.

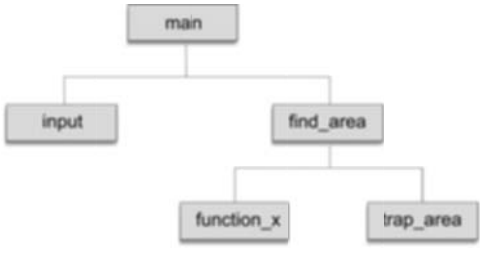


Fig. 5.40 Modular chart

Algorithm

1. Input the lower and upper limits and the number of trapezoids.
2. Calculate the width of trapezoids.
3. Initialize the total area.
4. Calculate the area of trapezoid and add to the total area.
5. Repeat step-4 until all the trapezoids are completed.
6. Print total area.

The algorithm is implemented in top-down modular form as in Figure 5.40

The evaluation of f(x) has been done using a separate function so that it can be easily modified to allow other functions to be evaluated.

The output for two runs shows that better accuracy is achieved with larger number of trapezoids. The actual area for the limits 0 and 3 is 12 units (by analytical method).

```
Program
#include <stdio.h>
float  start_point,        /* GLOBAL VARIABLES */
       end_point,
       total_area;
       int   numtraps;
       main( )
{
       void  input(void);
       float  find_area(float a,float b,int n); /* prototype */
       print("AREA UNDER A CURVE");
       input( );
       total_area = find_area(start_point, end_point, numtraps);
       printf("TOTAL AREA = %f", total_area);
}
void input(void)
{
       printf("\n Enter lower limit:");
       scanf("%f", &start_point);
       printf("Enter upper limit:");
       scanf("%f", &end_point);
       printf("Enter number of trapezoids:");
       scanf("%d", &numtraps);
}
float find_area(float a, float b, int n)
{
       float base, lower, h1, h2; /* LOCAL VARIABLES */
       float function_x(float x);  /* prototype */
       float trap_area(float h1,float h2,float base);/*prototype*/
       base = (b-1)/n;
          lower = a;
 for(lower =a; lower <= b-base; lower = lower + base)
          {
             h1 =  function_x(lower);
             h1 =  function_x(lower + base);
             total_area += trap_area(h1, h2, base);
          }
             return(total_area);
float trap_area(float height_1,float height_2,float base)
{
       float area;       /* LOCAL VARIABLE */
       area = 0.5 * (height_1 + height_2) * base;
       return(area);
}
float function_x(float x)
{
```

```
        /* F(X) = X * X + 1 */
        return(x*x + 1);
}
Output
        AREA UNDER A CURVE
        Enter lower limit: 0
        Enter upper limit: 3
        Enter number of trapezoids: 30
        TOTAL AREA = 12.005000
        AREA UNDER A CURVE
        Enter lower limit: 0
        Enter upper limit: 3
        Enter number of trapezoids: 100
        TOTAL AREA = 12.000438
```

Fig. 5.41 **Computing area under a curve**

5.8 STRUCTURES AND UNIONS

We have seen that arrays can be used to represent a group of data items that belong to the same type, such as int or float. However, we cannot use an array if we want to represent a collection of data items of different types using a single name. Fortunately, C supports a constructed data type known as structures, a mechanism for packing data of different types. A structure is a convenient tool for handling a group of logically related data items. For example, it can be used to represent a set of attributes, such as student_name, roll_number and marks. The concept of a structure is analogous to that of a 'record' in many other languages. More examples of such structures are:

time	:	seconds, minutes, hours
date	:	day, month, year
book	:	author, title, price, year
city	:	name, country, population
address	:	name, door-number, street, city
inventory	:	item, stock, value
customer	:	name, telephone, city, category

Structures help to organize complex data in a more meaningful way. It is a powerful concept that we may often need to use in our program design.

5.8.1 Defining a Structure

Unlike arrays, structures must be defined first for their format that may be used later to declare structure variables. Let us use an example to illustrate the process of structure definition and the creation of structure variables. Consider a book database consisting of book name, author, number of pages, and price. We can define a structure to hold this information as follows:

```
struct book_bank
{
      char  title[20];
      char  author[15];
      int   pages;
      float price;
};
```

The keyword **struct** declares a structure to hold the details of four data fields, namely title, author, pages, and price. These fields are called structure elements or members. Each member may belong to a different type of data. **book_bank** is the name of the structure and is called the structure tag. The tag name may be used subsequently to declare variables that have the tag's structure.

Note that the above definition has not declared any variables. It simply describes a format called template to represent information as shown in Figure 5.42:

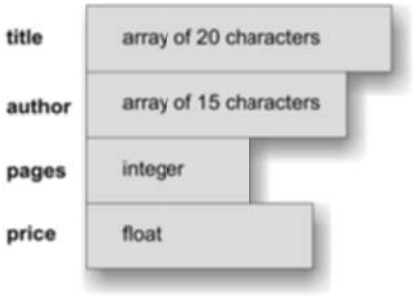


Fig. 5.42 Structure Format

The general format of a structure definition is as follows:

```
struct     tag_name
{
      data_type        member1;
      data_type        member2;
      ________         ____
      ________         ____
};
```

While defining a structure it is important to note the following about its syntax:

- The template is terminated with a semicolon.
- While the entire definition is considered as a statement, each member is declared independently for its name and type in a separate statement inside the template.
- The tag name such as book_bank can be used to declare structure variables of its type, later in the program.

Arrays vs Structures

Both the arrays and structures are classified as structured data types as they provide a mechanism that enable us to access and manipulate data in a relatively easy manner. But they differ in a number of ways.

1. An array is a collection of related data elements of same type. Structure can have elements of different types.
2. An array is derived data type whereas a structure is a programmer-defined one.
3. Any array behaves like a built-in data type. All we have to do is to declare an array variable and use it. But in the case of a structure, first we have to design and declare a data structure before the variables of that type are declared and used.

5.8.2 Declaring Structure Variables

After defining a structure format we can declare variables of that type. A structure variable declaration is similar to the declaration of variables of any other data types. It includes the following elements:

- The keyword struct.
- The structure tag name.
- List of variable names separated by commas.
- A terminating semicolon.

For example, consider the following statement:

```
struct book_bank, book1, book2, book3;
```

It declares book1, book2, and book3 as variables of type struct book_bank. Each one of these variables has four members as specified by the template.

The complete declaration might look like this:

```
struct book_bank
{
      char  title[20];
      char  author[15];
      int   pages;
      float  price;
};

struct book_bank book1, book2, book3;
```

Remember that the members of a structure themselves are not variables. They do not occupy any memory until they are associated with the structure variables such as book1. When the compiler comes across a declaration statement, it reserves memory space for the structure variables. It is also allowed to combine both the structure definition and variables declaration in one statement, as under:

```
struct book_bank
{
      char title[20];
      char author[15];
      int pages;
      flat price;
} book1, book2, book3;
```

5.8.3 Accessing Structure Members

We can access and assign values to the members of a structure in a number of ways. As mentioned earlier, the members themselves are not variables. They should be linked to the structure variables in order to make them meaningful members. For example, the word

title, has no meaning whereas the phrase 'title of book3' has a meaning. The link between a member and a variable is established using the member operator '.' which is also known as 'dot operator' or 'period operator'. For example:

book1.price

This is the variable representing the price of book1 and can be treated like any other ordinary variable. Here is how we would assign values to the members of book1:

```
strcpy(book1.title, "BASIC");
strcpy(book1.author, "Balagurusamy");
book1.pages = 250;
book1.price = 120.50;
```

We can also use scanf to give the values through the keyboard. For example:

```
scanf("%s\n", book1.title);
scanf("%d\n", &book1.pages);
```

EXAMPLE 5.17 *Figure 5.43 shows a program that defines a structure type, struct personal that would contain person name, date of joining and salary.*

The program reads this information for one person from the keyboard and prints the same on the screen.

```
Program
      struct personal
      {
         char name[20];
         int day;
         char month[10];
         int year;
         float salary;
      };
      main()
      {
         struct personal person;
         printf("Input Values\n");
         scanf("%s %d %s %d %f", person.name,&person.day, person.month, &person.year, &person.
salary);
         printf("%s %d %s %d %f\n",person.name, person.day, person.month, person.year, person.
salary);
      }

Output
      Input Values
      M.L.Goel 10 January 1945 4500
      M.L.Goel 10 January 1945 4500.00
```

Fig. 5.43 Illustration of using structures in C

In the above program, scanf and printf functions illustrate how the member operator '.' is used to link the structure members to the structure variables.

5.8.4 Structure Initialization

Like any other data type, a structure variable can be initialized at compile time. Example:

```
main()
 {
      struct
      {
         int weight;
         float height;
      }
      student = {60, 180.75};
      .....
      .....
 }
```

This assigns the value 60 to student.weight and 180.75 to student.height. There is a one-to-one correspondence between the members and their initializing values.

A lot of variation is possible in initializing a structure. The following statements initialize two structure variables. Here, it is essential to use a tag name.

```
main()
{
      struct st_record
      {
         int weight;
         float height;
      };
      struct st_record student1 = { 60, 180.75 };
      struct st_record student2 = { 53, 170.60 };
      .....
      .....
}
```

Another method is to initialize a structure variable outside the function as shown below:

```
struct st_record
{
      int weight;
      float height;
}     student1 = {60, 180.75};
main()
{
      struct st_record student2 = {53, 170.60};
      .....
      .....
}
```

C language does not permit the initialization of individual structure members within the template. The initialization must be done only in the declaration of the actual variables.

Note that the compile-time initialization of a structure variable must have the following elements:

- The keyword struct.
- The structure tag name.
- The name of the variable to be declared.
- The assignment operator =.
- A set of values for the members of the structure variable, separated by commas and enclosed in braces.
- A terminating semicolon.

Rules for Initializing Structures

There are a few rules to keep in mind while initializing structure variables at compile-time.

1. We cannot initialize individual members inside the structure template.
2. The order of values enclosed in braces must match the order of members in the structure definition.
3. It is permitted to have a partial initialization. We can initialize only the first few members and leave the remaining blank. The uninitialized members should be only at the end of the list.
4. The uninitialized members will be assigned default values as follows:
   - Zero for integer and floating point numbers.
   - '\0' for characters and strings.

5.8.5 Copying and Comparing Structure Variables

Two variables of the same structure type can be copied the same way as ordinary variables. If person1 and person2 belong to the same structure, then the following statements are valid:

```
person1 = person2;
person2 = person1;
```

However, the following statements are invalid as C does not permit any logical operations on structure variables.

```
person1 == person2
person1 != person2
```

In case, we need to compare them, we may do so by comparing the members individually.

5.8.6 Operations on Individual Members

As pointed out earlier, the individual members are identified using the member operator, the dot. A member with the dot operator along with its structure variable can be treated like any other variable name and therefore can be manipulated using expressions and operators. For example, the following operations are valid:

```
if (student1.number == 111)
student1.marks += 10.00;
float sum = student1.marks + student2.marks;
student2.marks * = 0.5;
```

We can also apply increment and decrement operators to numeric type members. For example, the following statements are valid:

```
student1.number ++;
++ student1.number;
```

The precedence of the member operator is higher than all arithmetic and relational operators and therefore no parentheses are required.

Three Ways to Access Members

We have used the dot operator to access the members of structure variables. In fact, there are two other ways. Consider the following structure:

typedef struct

```
{
            int x;
            int y;
} VECTOR;
VECTOR n, *ptr;
ptr = & n;
```

The identifier ptr is known as pointer that has been assigned the address of the structure variable n. Now, the members can be accessed in three ways:

1. using dot notation : n.x
2. using indirection notation : (*ptr).x
3. using selection notation : ptr –> x

5.8.7 Arrays of Structures

We use structures to describe the format of a number of related variables. For example, in analyzing the marks obtained by a class of students, we may use a template to describe student name and marks obtained in various subjects and then declare all the students as structure variables. In such cases, we may declare an array of structures, each element of the array representing a structure variable. For example, the following statement defines an array called student that consists of 100 elements.

```
struct class student[100];
```

Each element is defined to be of the type struct class. Now, consider the following declaration:

```
struct marks
{
      int   subject1;
      int   subject2;
      int   subject3;
};
main()
{
struct marks student[3] = {{45,68,81}, {75,53,69}, {57,36,71}};
```

This declares the student as an array of three elements student[0], student[1], and student[2] and initializes their members as follows:

```
student[0].subject1 = 45;
student[0].subject2 = 65;
....
....
student[2].subject3 = 71;
```

Note that the array is declared just as it would have been with any other array. Since student is an array, we use the usual array-accessing methods to access individual elements and then the member operator to access members. Remember, each element of student array is a structure variable with three members.

An array of structures is stored inside the memory in the same way as a multi-dimensional array. Figure 5.44 shows how the array student actually looks like inside the memory.

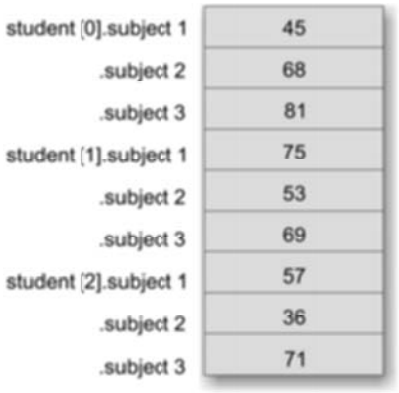


Fig. 5.44 **The array student inside memory**

EXAMPLE 5.18 *For the student array discussed above, Figure 5.45 shows a program to calculate the subject-wise and student-wise totals and store them as a part of the structure.*

```
Program
      struct marks
      {
         int sub1;
         int sub2;
         int sub3;
         int total;
      };
      main()
      {
         int i;
         struct marks student[3] = {{45,67,81,0},
                                    {75,53,69,0},
                                    {57,36,71,0}};
         struct marks total;
         for(i = 0; i <= 2; i++)
         {
```

```
            student[i].total =  student[i].sub1 +
                                student[i].sub2 +
                                student[i].sub3;
            total.sub1 = total.sub1 + student[i].sub1;
            total.sub2 = total.sub2 + student[i].sub2;
            total.sub3 = total.sub3 + student[i].sub3;
            total.total = total.total + student[i].total;
      }
      printf(" STUDENT TOTAL\n\n");
      for(i = 0; i <= 2; i++)
      printf("Student[%d] %d\n", i+1,student[i].total);
      printf("\n SUBJECT TOTAL\n\n");
      printf("%s %d\n%s %d\n%s %d\n",
             "Subject 1 ", total.sub1,
             "Subject 2 ", total.sub2,
             "Subject 3 ", total.sub3);
      printf("\nGrand Total = %d\n", total.total);
 }
Output
 STUDENT      TOTAL
 Student[1]   193
 Student[2]   197
 Student[3]   164
 SUBJECT      TOTAL
 Subject 1    177
 Subject 2    156
 Subject 3    221
 Grand Total = 554
```

Fig. 5.45 **Arrays of structures: Illustration of subscripted structure variables**

In the above program, we have declared a four-member structure, the fourth one for keeping the student-totals. We have also declared an array total to keep the subject-totals and the grand-total. The grand-total is given by total.total. Note that a member name can be any valid C name and can be the same as an existing structure variable name. The linked name total.total represents the total member of the structure variable total.

5.8.8 Arrays within Structures

C permits the use of arrays as structure members. We have already used arrays of characters inside a structure. Similarly, we can use single-dimensional or multi-dimensional arrays of type int or float. For example, the following structure declaration is valid:

```
struct marks
{
      int number;
      float subject[3];
} student[2];
```

Here, the member subject contains three elements, subject[0], subject[1] and subject[2]. These elements can be accessed using appropriate subscripts. For example, the following reference would refer to the marks obtained in the third subject by the second student.

```
student[1].subject[2];
```

EXAMPLE 5.19 *Rewrite the program of Example 12.3 using an array member to represent the three subjects.*

The modified program is shown in Fig. 5.46. You may notice that the use of array name for subjects has simplified in code.

```
Program
 main()
 {
 struct marks
 {
 int sub[3];
 int total;
 };
 struct marks student[3] =
 {45,67,81,0,75,53,69,0,57,36,71,0};
 struct marks total;
 int i,j;
 for(i = 0; i <= 2; i++)
 {
 for(j = 0; j <= 2; j++)
 {
       student[i].total += student[i].sub[j];
       total.sub[j] += student[i].sub[j];
 }
 total.total += student[i].total;
 }
 printf("STUDENT TOTAL\n\n");
 for(i = 0; i <= 2; i++)
       printf("Student[%d] %d\n", i+1, student[i].total);
 printf("\nSUBJECT TOTAL\n\n");
 for(j = 0; j <= 2; j++)
       printf("Subject-%d %d\n", j+1, total.sub[j]);
 printf("\nGrand Total = %d\n", total.total);
 }
Output
 STUDENT TOTAL
 Student[1] 193
 Student[2] 197
 Student[3] 164
 SUBJECT TOTAL
 Subject-1 177
 Subject-2 156
 Subject-3 221
 Grand Total = 554
```

Fig. 5.46 **Use of subscripted members arrays in structures**

5.8.9 Structures within Structures

Structures within a structure means nesting of structures. Nesting of structures is permitted in C. Let us consider the following structure defined to store information about the salary of employees.

```
struct salary
{
      char  name;
      char  department;
      int   basic_pay;
      int   dearness_allowance;
      int   house_rent_allowance;
      int   city_allowance;
}
employee;
```

This structure defines name, department, basic pay and three kinds of allowances. We can group all the items related to allowance together and declare them under a substructure as shown below:

```
struct salary
{
      char name;
      char department;
      struct
      {
         int dearness;
         int house_rent;
         int city;
      }
      allowance;
}
employee;
```

The salary structure contains a member named allowance, which itself is a structure with three members. The members contained in the inner structure namely dearness, house_rent, and city can be referred to as:

```
employee.allowance.dearness
employee.allowance.house_rent
employee.allowance.city
```

An inner-most member in a nested structure can be accessed by chaining all the concerned structure variables (from outer-most to inner-most) with the member using dot operator. An inner structure can also have more than one variable. Thus, the following form of declaration is legal:

```
struct salary
{
      .....
```

```
      struct
      {
         int dearness;
         .....
      }
      allowance,
      arrears;
}
employee[100];
```

Here, the inner structure has two variables, allowance and arrears. This implies that both of them have the same structure template. Note the comma after the name allowance. A base member can be accessed as follows:

```
employee[1].allowance.dearness
employee[1].arrears.dearness
```

We can also use tag names to define inner structures. For example:

```
          struct pay
             {
                int dearness;
                int house_rent;
                int city;
             };
             struct salary
             {
                char name;
                char department;
                struct pay allowance;
                struct pay arrears;
             };
             struct salary employee[100];
```

Here, pay template is defined outside the salary template and is used to define the structure of allowance and arrears inside the salary structure.

It is also permissible to nest more than one type of structures.

```
struct personal_record
{
      struct name_part name;
      struct addr_part address;
      struct date date_of_birth;
      .....
      .....
};
struct personal_record person1;
```

The first member of this structure is name, which is of the type struct name_part. Similarly, other members have their structure types.

5.8.10 Structures and Functions

We know that the main philosophy of C language is the use of functions. And therefore, it is natural that C supports the passing of structure values as arguments to functions. There are three methods by which the values of a structure can be transferred from one function to another.

- The first method is to pass each member of the structure as an actual argument of the function call. The actual arguments are then treated independently like ordinary variables. This is the most elementary method and becomes unmanageable and inefficient when the structure size is large.
- The second method involves passing of a copy of the entire structure to the called function. Since the function is working on a copy of the structure, any changes to structure members within the function are not reflected in the original structure (in the calling function). It is, therefore, necessary for the function to return the entire structure back to the calling function. All compilers may not support this method of passing the entire structure as a parameter.
- The third approach employs a concept called pointers to pass the structure as an argument. In this case, the address location of the structure is passed to the called function. The function can access indirectly the entire structure and work on it. This is similar to the way arrays are passed to function. This method is more efficient as compared to the second one.

The general format of sending a copy of a structure to the called function is:

```
function_name (structure_variable_name);
```

The called function takes the following form:

```
data_type function_name(struct_type st_name)
{
      ......
      ......
      return(expression);
}
```

Here, the following points are important to note:

- The called function must be declared for its type, appropriate to the data type it is expected to return. For example, if it is returning a copy of the entire structure, then it must be declared as struct with an appropriate tag name.
- The structure variable used as the actual argument and the corresponding formal argument in the called function must be of the same struct type.
- The return statement is necessary only when the function is returning some data back to the calling function. The expression may be any simple variable or structure variable or an expression using simple variables.
- When a function returns a structure, it must be assigned to a structure of identical type in the calling function.
- The called functions must be declared in the calling function appropriately.

EXAMPLE 5.20 *Write a simple program to illustrate the method of sending an entire structure as a parameter to a function.*

A program to update an item is shown in Figure 5.47. The function update receives a copy of the structure variable item as one of its parameters. Note that both the function update and the formal parameter product are declared as type struct stores. It is done so because the function uses the parameter product to receive the structure variable item and also to return the updated values of item.

```
Program
 /* Passing a copy of the entire structure */
 struct stores
 {
      char name[20];
      float price;
      int quantity;
 };
 struct stores update (struct stores product, float p, int q);
 float mul (struct stores stock);
 main()
 {
 float p_increment, value;
 int q_increment;

 struct stores item = {"XYZ", 25.75, 12};

 printf("\nInput increment values:");
 printf(" price increment and quantity increment\n");
 scanf("%f %d", &p_increment, &q_increment);

 /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
 item = update(item, p_increment, q_increment);
 /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
 printf("Updated values of item\n\n");
 printf("Name : %s\n",item.name);
 printf("Price : %f\n",item.price);
 printf("Quantity : %d\n",item.quantity);

 /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
 value = mul(item);
 /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

 printf("\nValue of the item = %f\n", value);
 }
 struct stores update(struct stores product, float p, int q)
 {
      product.price += p;
      product.quantity += q;
      return(product);
 }
 float mul(struct stores stock)
 {
```

```
 return(stock.price * stock.quantity);
 }

Output
Input increment values: price increment and quantity increment
10 12
Updated values of item
Name : XYZ
Price : 35.750000
Quantity : 24
Value of the item = 858.000000
```

Fig. 5.47 Using structure as a function parameter

In the above program, the function mul is of type float because it returns the product of price and quantity. However, the parameter stock, which receives the structure variable item is declared as type struct stores.

The entire structure returned by update can be copied into a structure of identical type. The following statement replaces the old values of item by the new ones.

```
item = update(item,p_increment,q_increment);
```

You may also notice that the template of stores is defined before main(). This has made the data type struct stores as global and has enabled the functions update and mul to make use of this definition.

5.8.11 Unions

Unions are a concept borrowed from structures and therefore follow the same syntax as structures. However, there is major distinction between them in terms of storage. In structures, each member has its own storage location, whereas all the members of a union use the same location. This implies that, although a union may contain many members of different types, it can handle only one member at a time. Like structures, a union can be declared using the keyword union as follows:

```
union item
{
      int m;
      float x;
      char c;
}code;
```

This declares a variable code of type union item. The union contains three members, each with a different data type. However, we can use only one of them at a time. This is due to the fact that only one location is allocated for a union variable, irrespective of its size.

The compiler allocates a piece of storage that is large enough to hold the largest variable type in the union. In the declaration above, the member x requires 4 bytes which is the largest among the members. Figure 5.48 shows how all the three variables share the same address. This assumes that a float variable requires 4 bytes of storage.

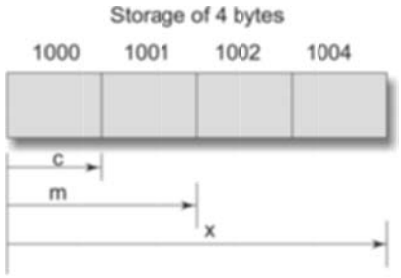


Fig. 5.48 **Sharing of a storage location by union members**

To access a union member, we can use the same syntax that we use for structure members. That is:

```
code.m
code.x
code.c
```

While accessing, we should make sure that we are accessing the member whose value is currently stored. For example, the following statements would produce erroneous output (which is machine dependent).

```
code.m = 379;
code.x = 7859.36;
printf("%d", code.m);
```

In effect, a union creates a storage location that can be used by any one of its members at a time. When a different member is assigned a new value, the new value supersedes the previous member's value.

Unions may be used in all places where a structure is allowed. The notation for accessing a union member which is nested inside a structure remains the same as for the nested structures.

Unions may be initialized when the variable is declared. But, unlike structures, it can be initialized only with a value of the same type as the first union member. For example, with the preceding example, the following declaration is valid:

```
union item abc = {100};
```

But this is invalid:

```
union item abc = {10.75};
```

This is because; the type of the first member is int. Other members can be initialized by either assigning values or reading from the keyboard.

5.8.12 Size of Structures

We normally use structures, unions, and arrays to create variables of large sizes. The actual size of these variables in terms of bytes may change from machine to machine. We may use the unary operator sizeof to determine the size of a structure (or any variable). For example, consider the following expression:

```
sizeof(struct x)
```

This will evaluate the number of bytes required to hold all the members of the structure x.

5.9 CASE STUDY

Book Shop Inventory

A book shop uses a personal computer to maintain the inventory of books that are being sold at the shop. The list includes details such as author, title, price, publisher, stock position, etc. Whenever a customer wants a book, the shopkeeper inputs the title and author of the book and the system replies whether it is in the list or not. If it is not, an appropriate message is displayed. If book is in the list, then the system displays the book details and asks for number of copies. If the requested copies are available, the total cost of the books is displayed; otherwise the message "Required copies not in stock" is displayed.

A program to accomplish this is shown in Figure 5.49. The program uses a template to define the structure of the book. Note that the date of publication, a member of record structure, is also defined as a structure.

```
Programs
#include <stdio.h>
#include <string.h>
struct record
{
char author[20];
 char title[30];
 float price;
 struct
 {
 char month[10];
 int year;
 }
 date;
 char publisher[10];
 int quantity;
};
 int look_up(struct record table[],char s1[],char s2[],int m);
 void get (char string [ ] );
 main()
 {
 char title[30], author[20];
 int index, no_of_records;
 char response[10], quantity[10];
 struct record book[] = {
 {"Ritche","C Language",45.00,"May",1977,"PHI",10},
 {"Kochan","Programming in C",75.50,"July",1983,"Hayden",5},
 {"Balagurusamy","BASIC",30.00,"January",1984,"TMH",0},
 {"Balagurusamy","COBOL",60.00,"December",1988,"Macmillan",25}};
```

```
      no_of_records = sizeof(book)/ sizeof(struct record);
      do
      {
         printf("Enter title and author name as per the list\n");
         printf("\nTitle: ");
         get(title);
         printf("Author: ");
         get(author);
         index = look_up(book, title, author, no_of_records);
         if(index != -1) /* Book found */
         {
            printf("\n%s %s %.2f %s %d %s\n\n",
            book[index].author,
         book[index].title,
         book[index].price,
         book[index].date.month,
         book[index].date.year,
         book[index].publisher);
 printf("Enter number of copies:");
      get(quantity);
      if(atoi(quantity) < book[index].quantity)
 printf("Cost of %d copies = %.2f\n",atoi(quantity), book[index].price * atoi(quantity));
      else
         printf("\nRequired copies not in stock\n\n");
      }
 else
 printf("\nBook not in list\n\n");
 printf("\nDo you want any other book? (YES / NO):");
 get(response);
 }
 while(response[0] == 'Y' || response[0] == 'y');
 printf("\n\nThank you. Good bye!\n");
}
void get(char string [] )
{
char c;
 int i = 0;
 do
 {
        c = getchar();
 string[i++] = c;
 }
 while(c != '\n');
 string[i-1] = '\0';
}
int look_up(struct record table[],char s1[],char s2[],int m)
{
 int i;
 for(i = 0; i < m; i++)
 if(strcmp(s1, table[i].title) == 0 &&
 strcmp(s2, table[i].author) == 0)
 return(i); /* book found */
 return(-1); /* book not found */
```

```
}
Output
 Enter title and author name as per the list
 Title: BASIC
 Author: Balagurusamy
 Balagurusamy BASIC 30.00 January 1984 TMH
 Enter number of copies:5
 Required copies not in stock
 Do you want any other book? (YES / NO):y
 Enter title and author name as per the list
 Title: COBOL
 Author: Balagurusamy
 Balagurusamy COBOL 60.00 December 1988 Macmillan
 Enter number of copies:7
 Cost of 7 copies = 420.00
 Do you want any other book? (YES / NO):y
 Enter title and author name as per the list
 Title: C Programming
 Author: Ritche
 Book not in list
 Do you want any other book? (YES / NO):n
 Thank you. Good bye!
```

Fig. 5.49 Program of bookshop inventory

When the title and author of a book are specified, the program searches for the book in the list using the function

look_up(table, s1, s2, m)

The parameter table which receives the structure variable book is declared as type struct record. The parameters s1 and s2 receive the string values of title and author while m receives the total number of books in the list. Total number of books is given by the expression

sizeof(book)/sizeof(struct record)

The search ends when the book is found in the list and the function returns the serial number of the book. The function returns –1 when the book is not found. Remember that the serial number of the first book in the list is zero. The program terminates when we respond "NO" to the question

Do you want any other book?

Note that we use the following function to get title, author, etc. from the terminal.

get(string)

This enables us to input strings with spaces such as "C Language". We cannot use scanf to read this string since it contains two words.

Since we are reading the quantity as a string using the get(string) function, we have to convert it to an integer before using it in any expressions. This is done using the atoi() function.

5.10 POINTERS

A pointer is a derived data type in C. It is built from one of the fundamental data types available in C. Pointers contain memory addresses as their values. Since these memory addresses are the locations in the computer memory where program instructions and data are stored, pointers can be used to access and manipulate data stored in the memory.

Pointers are undoubtedly one of the most distinct and exciting features of C language. It has added power and flexibility to the language. Although they appear little confusing and difficult to understand for a beginner, they are a powerful tool and handy to use once they are mastered.

Pointers are used frequently in C, as they offer a number of benefits to the programmers. These include:

- Pointers are more efficient in handling arrays and data tables.
- Pointers can be used to return multiple values from a function via function arguments.
- Pointers permit references to functions and thereby facilitating passing of functions as arguments to other functions.
- The use of pointer arrays to character strings results in saving of data storage space in memory.
- Pointers allow C to support dynamic memory management.
- Pointers provide an efficient tool for manipulating dynamic data structures such as structures, linked lists, queues, stacks and trees.
- Pointers reduce length and complexity of programs.
- They increase the execution speed and thus reduce the program execution time.

5.10.1 Understanding Pointers

The computer's memory is a sequential collection of storage cells as shown in Figure 5.50. Each cell, commonly known as a byte, has a number called address associated with it. Typically, the addresses are numbered consecutively, starting from zero. The last address depends on the memory size. A computer system having 64 K memory will have its last address as 65,535.

Whenever we declare a variable, the system allocates, somewhere in the memory, an appropriate location to hold the value of the variable. Since, every byte has a unique address number; this location will have its own address number. Consider the following statement

```
int quantity = 179;
```

Fig. 5.50 Memory organisation

This statement instructs the system to find a location for the integer variable quantity and puts the value 179 in that location. Let us assume that the system has chosen the address location 5000 for quantity. We may represent this as shown in Figure 5.51.

Fig. 5.51 **Representation of a variable**

NOTE: The address of a variable is the address of the first byte occupied by that variable.

During execution of the program, the system always associates the name quantity with the address 5000. (This is something similar to having a house number as well as a house name.) We may have access to the value 179 by using either the name quantity or the address 5000. Since memory addresses are simply numbers, they can be assigned to some variables that can be stored in memory, like any other variable. Such variables that hold memory addresses are called pointer variables. A pointer variable is, therefore, nothing but a variable that contains an address, which is a location of another variable in memory.

Remember, since a pointer is a variable, its value is also stored in the memory in another location. Suppose, we assign the address of quantity to a variable p. The link between the variables p and quantity can be visualized as shown in Figure 5.52. The address of p is 5048.

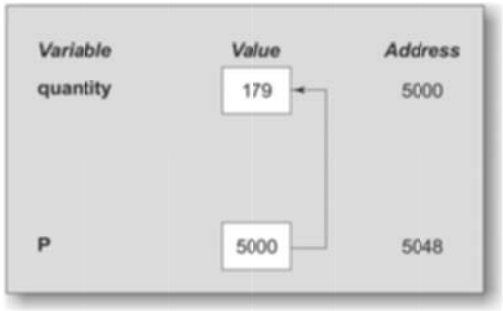
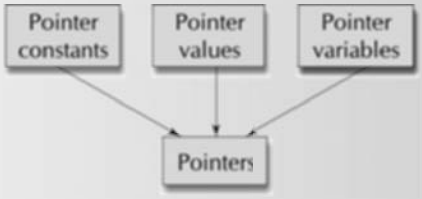


Fig. 5.52 **Pointer variable**

Since the value of the variable p is the address of the variable quantity, we may access the value of quantity by using the value of p and therefore, we say that the variable p 'points' to the variable quantity. Thus, p gets the name 'pointer'. (We are not really concerned about the actual values of pointer variables. They may be different every time we run the program. What we are concerned about is the relationship between the variables p and quantity.)

Underlying Concepts of Pointers

Pointers are built on the three underlying concepts as illustrated below:



Memory addresses within a computer are referred to as pointer constants. We cannot change them; we can only use them to store data values. They are like house numbers.

We cannot save the value of a memory address directly. We can only obtain the value through the variable stored there using the address operator (&). The value thus obtained is known as pointer value. The pointer value (i.e. the address of a variable) may change from one run of the program to another.

Once we have a pointer value, it can be stored into another variable. The variable that contains a pointer value is called a pointer variable.

5.10.2 Accessing the Address of a Variable

The actual location of a variable in the memory is system dependent and therefore, the address of a variable is not known to us immediately. How can we then determine the address of a variable? This can be done with the help of the operator & available in C. We have already seen the use of this address operator in the scanf function. The operator & immediately preceding a variable returns the address of the variable associated with it. For example, consider the following statement:

```
p = &quantity;
```

This would assign the address 5000 (the location of quantity) to the variable p. The & operator can be remembered as 'address of'.

The & operator can be used only with a simple variable or an array element. The following are illegal uses of address operator:

- &125 (pointing at constants)
- int x[10];
  &x (pointing at array names)
- &(x+y) (pointing at expressions)

If x is an array, then expressions such as &x[0] and &x[i+3] are valid and represent the addresses of 0th and (i+3)th elements of x.

EXAMPLE 5.21 *Figure 5.33 shows a program that prints the address of a variable along with its value.*

```
Program
main()
{
     char a;
     int x;
     float p, q;

     a = 'A';
     x = 125;
     p = 10.25, q = 18.76;
     printf("%c is stored at addr %u.\n", a, &a);
     printf("%d is stored at addr %u.\n", x, &x);
     printf("%f is stored at addr %u.\n", p, &p);
     printf("%f is stored at addr %u.\n", q, &q);
}
```

```
Output
      A is stored at addr 4436.
      125 is stored at addr 4434.
      10.250000 is stored at addr 4442.
      18.760000 is stored at addr 4438.
```

Fig. 5.53 Accessing the address of a variable

The above program declares and initializes four variables and then prints out these values with their respective storage locations. Note that we have used %u format for printing address values. Memory addresses are unsigned integers.

5.10.3 Declaring Pointer Variables

In C, every variable must be declared for its type. Since pointer variables contain addresses that belong to a separate data type, they must be declared as pointers before we use them. The declaration of a pointer variable takes the following form:

```
data_type *pt_name;
```

This tells the compiler three things about the variable pt_name.

- The asterisk (*) tells that the variable pt_name is a pointer variable.
- pt_name needs a memory location.
- pt_name points to a variable of type data_type.

For example, consider the following declaration:

```
int *p; /* integer pointer */
```

This declares the variable p as a pointer variable that points to an integer data type. Remember that the type int refers to the data type of the variable being pointed to by p and not the type of the value of the pointer. Similarly, the consider the following statement:

```
float *x; / * float pointer */
```

This declares x as a pointer to a floating-point variable.

The declarations cause the compiler to allocate memory locations for the pointer variables p and x. Since the memory locations have not been assigned any values, these locations may contain some unknown values in them and therefore they point to unknown locations as depicted in Figure 5.54:

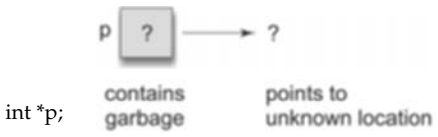


Fig. 5.54 Uninitialized pointer variable

5.10.4 Initialization of Pointer Variables

The process of assigning the address of a variable to a pointer variable is known as initialization. As pointed out earlier, all uninitialized pointers will have some unknown values that will be interpreted as memory addresses. They may not be valid addresses or they may point to some values that are wrong. Since the compilers do not detect these errors, the programs with uninitialized pointers will produce erroneous results. It is therefore important to initialize pointer variables carefully before they are used in the program.

Once a pointer variable has been declared we can use the assignment operator to initialize the variable. Example:

```
int quantity;
int *p; /* declaration*/
p = &quantity;/* initialisation */
```

We can also combine the initialization with the declaration. Example:

```
int *p = &quantity;
```

The only requirement here is that the variable quantity must be declared before the initialization takes place. Remember, this is an initialization of p and not *p.

We must ensure that the pointer variables always point to the corresponding type of data. For instance, the following will result in erroneous output because we are trying to assign the address of a float variable to an integer pointer.

```
float a, b;
int x, *p;
p = &a; /* wrong */
b = *p;
```

When we declare a pointer to be of int type, the system assumes that any address that the pointer will hold will point to an integer variable. Since the compiler will not detect such errors, care should be taken to avoid wrong pointer assignments.

It is also possible to combine the declaration of data variable, the declaration of pointer variable and the initialization of the pointer variable in one step. For example:

```
int x, *p = &x;  /* three in one */
```

It declares x as an integer variable and p as a pointer variable and then initializes p to the address of x. And also remember that the target variable x is declared first. Thus, the following statement would be invalid:

```
int *p = &x, x;
```

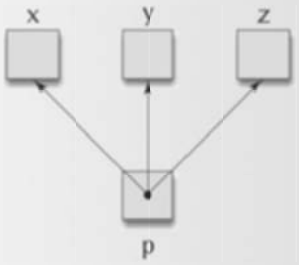
We could also define a pointer variable with an initial value of NULL or 0 (zero). Example:

```
int *p = NULL;
int *p = 0;
```

Pointer Flexibility

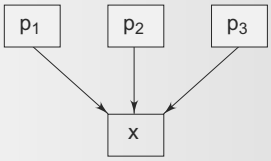
Pointers are flexible. We can make the same pointer to point to different data variables in different statements. Example;

```
int x, y, z, *p;
.....
p = &x;
.....
p = &y;
.....
p = &z;
.....
```



We can also use different pointers to point to the same data variable. Example.

```
int x;
int *p1 = &x;
int *p2 = &x;
int *p3 = &x;
.....
.....
```



5.10.5 Accessing a Variable through its Pointer

Once a pointer has been assigned the address of a variable, the question remains as to how to access the value of the variable using the pointer? This is done by using another unary operator * (asterisk), usually known as the indirection operator. Another name for the indirection operator is the dereferencing operator. Consider the following statements:

```
int quantity, *p, n;
quantity = 179;
p = &quantity;
n = *p;
```

The first line declares quantity and n as integer variables and p as a pointer variable pointing to an integer. The second line assigns the value 179 to quantity and the third line assigns the address of quantity to the pointer variable p. The fourth line contains the indirection operator *.

When the operator * is placed before a pointer variable in an expression (on the right-hand side of the equal sign), the pointer returns the value of the variable of which the pointer value is the address. In this case, *p returns the value of the variable quantity, because p is the

address of quantity. The * can be remembered as 'value at address'. Thus, the value of n would be 179.

In C, the assignment of pointers and addresses is always done symbolically, by means of symbolic names. You cannot access the value stored at the address 5368 by writing *5368. It will not work. The following example illustrates the distinction between pointer value and the value it points to.

EXAMPLE 5.22 *Figure 5.55 shows a program to illustrate the use of indirection operator '*' to access the value pointed to by a printer.*

```
Program
      main()
      {
         int x, y;
         int *ptr;
         x = 10;
         ptr = &x;
         y = *ptr;
         printf("Value of x is %d\n\n",x);
         printf("%d is stored at addr %u\n", x, &x);
         printf("%d is stored at addr %u\n", *&x, &x);
         printf("%d is stored at addr %u\n", *ptr, ptr);
         printf("%d is stored at addr %u\n", ptr, &ptr);
         printf("%d is stored at addr %u\n", y, &y);
         *ptr = 25;
         printf("\nNow x = %d\n",x);
      }
Output
      Value of x is 10
      10    is stored at addr 4104
      10    is stored at addr 4104
      10    is stored at addr 4104
      4104  is stored at addr 4106
      10    is stored at addr 4108
      Now x = 25
```

Fig. 5.55 Accessing a variable through its pointer

The above program clearly shows how we can access the value of a variable using a pointer. You may notice that the value of the pointer ptr is 4104 and the value it points to is 10. Further, you may also note the following equivalences:

x = *(&x) = *ptr = y

&x = &*ptr

The actions performed by the above program are illustrated in Figure 5.56. The statement ptr = &x assigns the address of x to ptr and y = *ptr assigns the value pointed to by the pointer ptr to y.

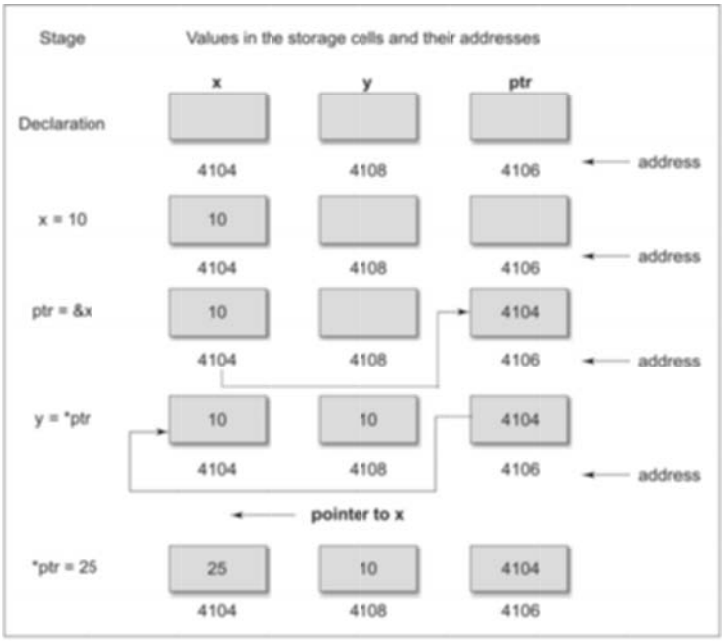


Fig. 5.56 **Illustration of pointer assignments**

5.10.6 Pointer to Pointer

It is possible to make a pointer to point to another pointer, thus creating a chain of pointers as shown in Figure 5.57:

Fig. 5.57 **Illustration of pointer to pointer**

Here, the pointer variable p2 contains the address of the pointer variable p1, which points to the location that contains the desired value. This is also known as multiple indirections.

A variable that is a pointer to a pointer must be declared using additional indirection operator symbols in front of the name. Example:

```
int **p2;
```

This declaration tells the compiler that p2 is a pointer to a pointer of int type. Remember, the pointer p2 is not a pointer to an integer, but rather a pointer to an integer pointer.

We can access the target value indirectly pointed to by pointer to a pointer by applying the indirection operator twice. For example, consider the following code:

```
main ( )
{
      int x, *p1,**p2;
      x = 100;
      p1 = &x; /* address of x */
      p2 = &p1 /* address of p1 */
      printf ("%d", **p2);
}
```

This code will display the value 100. Here, p1 is declared as a pointer to an integer and p2 as a pointer to a pointer to an integer.

5.10.7 Pointer Expressions

Like other variables, pointer variables can be used in expressions. For example, if p1 and p2 are properly declared and initialized pointers, then the following statements are valid.

```
y = *p1 * *p2;same as (*p1) * (*p2)
sum = sum + *p1;
z = 5* – *p2/ *p1;  same as (5 * (– (*p2)))/(*p1)
*p2 = *p2 + 10;
```

Note that there is a blank space between / and * in the 3rd expression above. The following expression is wrong:

```
z = 5* – *p2 /*p1;
```

The symbol /* is considered as the beginning of a comment and therefore the statement fails.

C allows us to add integers to or subtract integers from pointers, as well as to subtract one pointer from another. p1 + 4, p2–2 and p1 – p2 are all allowed. If p1 and p2 are both pointers to the same array, then p2 – p1 gives the number of elements between p1 and p2.

We may also use short-hand operators with the pointers. Example:

```
p1++;
--p2;
sum += *p2;
```

In addition to arithmetic operations discussed above, pointers can also be compared using the relational operators. The expressions such as p1 > p2, p1 = = p2, and p1 != p2 are allowed. However, any comparison of pointers that refer to separate and unrelated variables makes no sense. Comparisons can be used meaningfully in handling arrays and strings.

We may not use pointers in division or multiplication. For example, the following expressions are invalid:

```
p1 / p2 or p1 * p2 or p1 / 3
```

Similarly, two pointers cannot be added. That is, p1 + p2 is illegal.

EXAMPLE 5.23 *Write a program to illustrate the use of pointers in arithmetic operations.*

The program in Figure 5.58 shows how the pointer variables can be directly used in expressions.

```
Program
      main()
      {
         int a, b, *p1, *p2, x, y, z;
         a = 12;
         b = 4;
         p1 = &a;
         p2 = &b;
         x = *p1 * *p2 – 6;
         y   = 4* – *p2 / *p1 + 10;
         printf("Address of a = %u\n", p1);
         printf("Address of b = %u\n", p2);
         printf("\n");
         printf("a = %d, b = %d\n", a, b);
         printf("x = %d, y = %d\n", x, y);
         *p2 = *p2 + 3;
         *p1 = *p2 – 5;
         z = *p1 * *p2 – 6;
         printf("\na = %d, b = %d,", a, b);
         printf(" z = %d\n", z);
      }
Output
         Address of a = 4020
         Address of b = 4016
         a = 12, b = 4
         x = 42, y = 9
         a = 2, b = 7, z = 8
```

Fig. 5.58 Evaluation of pointer expressions

The above program also illustrates the order of evaluation of expressions. For example, consider the following expression:

$$4* – *p2 / *p1 + 10$$

It is evaluated as follows:

$$((4 * (–(*p2))) / (*p1)) + 10$$

When *p1 = 12 and *p2 = 4, this expression evaluates to 9. Remember, since all the variables are of type int, the entire evaluation is carried out using the integer arithmetic.

5.10.8 Pointer Increments and Scale Factor

We have seen that the pointers can be incremented like:

```
p1 = p2 + 2;
p1 = p1 + 1;
```

Remember, however, an expression as shown below will cause the pointer p1 to point to the next value of its type.

```
p1++;
```

For example, if p1 is an integer pointer with an initial value, say 2800, then after the operation p1 = p1 + 1, the value of p1 will be 2802, and not 2801. That is, when we increment a pointer, its value is increased by the 'length' of the data type that it points to. This length called the scale factor.

For an IBM PC, the length of various data types are as follows:

- characters 1 byte
- integers 2 bytes
- floats 4 bytes
- long integers 4 bytes
- doubles 8 bytes

The number of bytes used to store various data types depends on the system and can be found by making use of the sizeof operator. For example, if x is a variable, then sizeof(x) returns the number of bytes needed for the variable. (Systems like Pentium use 4 bytes for storing integers and 2 bytes for short integers.)

Rules of Pointer Operations

The following rules apply when performing operations on pointer variables.

1. A pointer variable can be assigned the address of another variable.
2. A pointer variable can be assigned the values of another pointer variable.
3. A pointer variable can be initialized with NULL or zero value.
4. A pointer variable can be pre-fixed or post-fixed with increment or decrement operators.
5. An integer value may be added or subtracted from a pointer variable.
6. When two pointers point to the same array, one pointer variable can be subtracted from another.
7. When two pointers point to the objects of the same data types, they can be compared using relational operators.
8. A pointer variable cannot be multiplied by a constant.
9. Two pointer variables cannot be added.
10. A value cannot be assigned to an arbitrary address (i.e &x = 10; is illegal).

5.10.9 Pointers and Arrays

When an array is declared, the compiler allocates a base address and sufficient amount of storage to contain all the elements of the array in contiguous memory locations. The base address is the location of the first element (index 0) of the array. The compiler also defines

the array name as a constant pointer to the first element. Suppose we declare an array x as follows:

```
int x[5] = {1, 2, 3, 4, 5};
```

Suppose the base address of x is 1000 and assuming that each integer requires two bytes, the five elements will be stored as depicted in Figure 5.59:

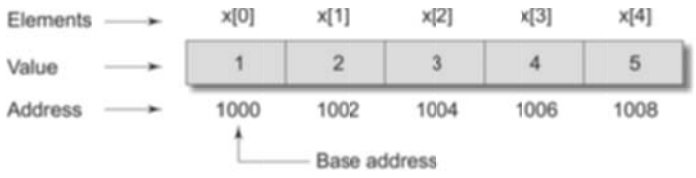


Fig. 5.59 Illustration of array storage

The name x is defined as a constant pointer pointing to the first element, x[0] and therefore the value of x is 1000, the location where x[0] is stored. That is,

$$x = \&x[0] = 1000$$

If we declare p as an integer pointer, then we can make the pointer p to point to the array x by the following assignment:

```
p = x;
```

This will be equivalent to p = &x[0];

Now, we can access every value of x using p++ to move from one element to another. The relationship between p and x is shown as under:

p = &x[0] (= 1000)

p+1 = &x[1] (= 1002)

p+2 = &x[2] (= 1004)

p+3 = &x[3] (= 1006)

p+4 = &x[4] (= 1008)

You may notice that the address of an element is calculated using its index and the scale factor of the data type. For instance,

address of x[3] = base address + (3 x scale factor of int)

= 1000 + (3 x 2) = 1006

When handling arrays, instead of using array indexing, we can use pointers to access array elements. Note that *(p+3) gives the value of x[3]. The pointer accessing method is much faster than array indexing.

EXAMPLE 5.24 *Figure 5.60 shows a program that uses pointers to compute the sum of all elements stored in an array.*

```
Program
      main()
      {
         int *p, sum, i;
         int x[5] = {5,9,6,3,7};
         i = 0;
         p = x;      /* initializing with base address of x */
         printf("Element Value Address\n\n");
         while(i < 5)
         {
            printf(" x[%d] %d %u\n", i, *p, p);
            sum = sum + *p; /* accessing array element */
            i++, p++; /* incrementing pointer */
         }
         printf("\n Sum = %d\n", sum);
         printf("\n &x[0] = %u\n", &x[0]);
         printf("\n p = %u\n", p);
      }

Output
        Element    Value     Address
          x[0]        5         166
          x[1]        9         168
          x[2]        6         170
          x[3]        3         172
          x[4]        7         174
         Sum = 55
         &x[0] = 166
         p = 176
```

Fig. 5.60 Accessing one-dimensional array elements using the pointer

The above program illustrates how a pointer can be used to traverse an array element. Since incrementing an array pointer causes it to point to the next element, we need only to add one to p each time we go through the loop.

It is possible to avoid the loop control variable i as shown below:

```
.....
p = x;
while(p <= &x[4])
{
      sum += *p;
      p++;
}
.....
```

Here, we compare the pointer p with the address of the last element to determine when the array has been traversed.

Pointers can be used to manipulate two-dimensional arrays as well. Suppose we declare an array a as follows:

```
int a[3][4] = {  {15,27,11,35},
                 {22,19,31,17},
                 {31,23,14,36}
              };
```

The elements of a will be stored as shown in Figure 5.61:

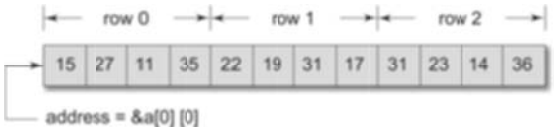


Fig. 5.61 **Two-dimensional array storage**

If we declare p as an int pointer with the initial address of &a[0][0], then,

a[i][j] is equivalent to *(p+4 ´ i+j)

You may notice that, if we increment i by 1, the p is incremented by 4, the size of each row. Then the element a[2][3] is given by *(p+2 ´ 4+3) = *(p+11).

This is the reason why, when a two-dimensional array is declared, we must specify the size of each row so that the compiler can determine the correct storage mapping.

5.10.10 Pointers and Character Strings

Strings are treated like character arrays and therefore, they are declared and initialized as follows:

```
char str [5] = "good";
```

The compiler automatically inserts the null character '\0' at the end of the string. C supports an alternative method to create strings using pointer variables of type char. Example:

```
char *str = "good";
```

This creates a string for the literal and then stores its address in the pointer variable str.

The pointer str now points to the first character of the string "good" as shown in Figure 5.62:

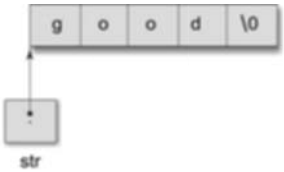


Fig. 5.62 **String referencing using pointers**

We can also use the run-time assignment for giving values to a string pointer. Example

```
char * string1;
string1 = "good";
```

We can print the contents of the string string1 using either printf or puts functions as follows:

```
printf("%s", string1);
puts (string1);
```

Remember, although string1 is a pointer to the string, it is also the name of the string. Therefore, we do not need to use indirection operator * here. Like in one-dimensional arrays, we can use a pointer to access the individual characters in a string.

5.10.11 Array of Pointers

One important use of pointers is in handling of a table of strings. Consider the following array of strings:

```
char name [3][25];
```

This says that the name is a table containing three names, each with a maximum length of 25 characters (including null character). The total storage requirements for the name table are 75 bytes.

We know that rarely the individual strings will be of equal lengths. Therefore, instead of making each row a fixed number of characters, we can make it a pointer to a string of varying length. For example:

```
char *name[3] = {
        "New Zealand",
        "Australia",
        "India"
};
```

This declares name to be an array of three pointers to characters, each pointer pointing to a particular name as shown in Figure 5.63.

name [0] → .New Zealand
name [1] → .Australia
name [2] → .India

Fig. 5.63 Array of Pointers

This declaration allocates only 28 bytes, sufficient to hold all the characters as shown in Figure 5.64.

The following statement would print out all the three names:

```
for(i = 0; i <= 2; i++)
printf("%s\n", name[i]);
```

Fig. 5.64 Array of strings of varied lengths

5.10.12 Pointers as Function Arguments

We have seen earlier that when an array is passed to a function as an argument, only the address of the first element of the array is passed, but not the actual values of the array elements. If x is an array, when we call sort(x), the address of x[0] is passed to the function sort. The function uses this address for manipulating the array elements. Similarly, we can pass the address of a variable as an argument to a function in the normal fashion.

When we pass addresses to a function, the parameters receiving the addresses should be pointers. The process of calling a function using pointers to pass the addresses of variables is known as 'call by reference'. (You know, the process of passing the actual value of variables is known as "call by value".) The function which is called by 'reference' can change the value of the variable used in the call.

Consider the following code:

```
main()
{
      int x;
      x = 20;
      change(&x); /* call by reference or address */
      printf("%d\n",x);
}
change(int *p)
{
      *p = *p + 10;
}
```

When the function change( ) is called, the address of the variable x, not its value, is passed into the function change(). Inside change( ), the variable p is declared as a pointer and therefore p is the address of the variable x. The statement, *p = *p + 10; means 'add 10 to the value stored at the address p'. Since p represents the address of x, the value of x is changed from 20 to 30. Therefore, the output of the program will be 30, not 20.

Thus, call by reference provides a mechanism by which the function can change the stored values in the calling function. Note that this mechanism is also known as "call by address" or "pass by pointers"

EXAMPLE 5.25 *Figure 5.65 shows a program to swap the values stored in two locations in the memory.*

```
Program
 void exchange (int *, int *);  /* prototype */
 main()
 {
      int x, y;
      x = 100;
      y = 200;
      printf("Before exchange : x = %d y = %d\n\n", x, y);
      exchange(&x,&y);    /* call */
      printf("After exchange : x = %d y = %d\n\n", x, y);
```

```
}
exchange (int *a, int *b)
{
     int t;
     t = *a; /* Assign the value at address a to t */
     *a = *b; /* put b into a */
     *b = t; /* put t into b */
}
Output
 Before exchange : x = 100 y = 200
 After exchange : x = 200 y = 100
```

Fig. 5.65 Passing of pointers as function parameters

In the above program, the contents of two locations can be exchanged using their address locations. The function exchange() receives the addresses of the variables x and y and exchanges their contents. Here, you may note the following points:

- The function parameters are declared as pointers.
- The dereferenced pointers are used in the function body.
- When the function is called, the addresses are passed as actual arguments.

5.10.13 Functions Returning Pointers

We have seen so far that a function can return a single value by its name or return multiple values through pointer parameters. Since pointers are a data type in C, we can also force a function to return a pointer to the calling function. Consider the following code:

```
int *larger (int *, int *);    /* prototype */
main ( )
{
     int a = 10;
     int b = 20;
     int *p;
     p = larger(&a, &b); /Function call */
     printf ("%d", *p);
}
int *larger (int *x, int *y)
{
     if (*x>*y)
        return (x); / *address of a */
     else
        return (y); /* address of b */
}
```

Here, the function larger receives the addresses of the variables a and b, decides which one is larger using the pointers x and y and then returns the address of its location. The returned value is then assigned to the pointer variable p in the calling function. In this case, the address of b is returned and assigned to p and therefore the output will be the value of b, namely, 20.

NOTE: The address returned must be the address of a variable in the calling function. It is an error to return a pointer to a local variable in the called function.

5.10.14 Pointers to Functions

A function, like a variable, has a type and an address location in the memory. It is therefore, possible to declare a pointer to a function, which can then be used as an argument in another function. A pointer to a function is declared as follows:

```
type (*fptr) ();
```

This tells the compiler that fptr is a pointer to a function, which returns type value. The parentheses around *fptr are necessary. Remember that a statement like type *gptr(); would declare gptr as a function returning a pointer to type.

We can make a function pointer to point to a specific function by simply assigning the name of the function to the pointer. For example, consider the following statements:

```
double mul(int, int);
double (*p1)();
p1 = mul;
```

Here, p1 as a pointer to a function and mul is a function to which p1 points. To call the function mul, we may now use the pointer p1 with the list of parameters. That is,

```
(*p1)(x,y) /* Function call */
```

The above function call is equivalent to

```
mul(x,y)
```

5.10.15 Pointers and Structures

We know that the name of an array stands for the address of its zeroth element. The same thing is true of the names of arrays of structure variables. Suppose product is an array variable of struct type. The name product represents the address of its zeroth element. Now, consider the following declaration:

```
struct inventory
{
      char  name[30];
      int   number;
      float price;
} product[2], *ptr;
```

This statement declares product as an array of two elements, each of the type struct inventory and ptr as a pointer to data objects of the type struct inventory. Now, consider the following assignment statement:

```
ptr = product;
```

This would assign the address of the zeroth element of product to ptr. That is, the pointer ptr will now point to product[0]. Its members can be accessed using the following notation.

ptr –> name

ptr –> number

ptr –> price

The symbol –> is called the arrow operator (also known as member selection operator) and is made up of a minus sign and a greater than sign. Note that ptr–> is simply another way of writing product[0].

When the pointer ptr is incremented by one, it is made to point to the next record, i.e., product[1]. The following for statement will print the values of members of all the elements of product array.

```
for(ptr = product; ptr < product+2; ptr++)
printf ("%s %d %f\n", ptr–>name, ptr–>number, ptr–>price);
```

We could also use the following notation (*ptr).number to access the member number. The parentheses around *ptr are necessary because the member operator '.' has a higher precedence than the operator *.

EXAMPLE 5.26 *Figure 5.66 shows a program to illustrate the use of structure pointers.*

```
Program
 struct invent
 {
      char *name[20];
      int number;
      float price;
 };
 main()
 {
      struct invent product[3], *ptr;
      printf("INPUT\n\n");
      for(ptr = product; ptr < product+3; ptr++)
      scanf("%s %d %f", ptr–>name, &ptr–>number, &ptr–>price);
      printf("\nOUTPUT\n\n");
      ptr = product;
      while(ptr < product + 3)
      {
         printf("%–20s %5d %10.2f\n",
         ptr–>name,
         ptr–>number,
         ptr–>price);
         ptr++;
      }
 }
```

```
Output
 INPUT
 Washing_machine 5 7500
 Electric_iron 12 350
 Two_in_one 7 1250
 OUTPUT
 Washing machine    5    7500.00
 Electric_iron      12    350.00
 Two_in_one         7    1250.00
```

Fig. 5.66 **Pointer to structure variables**

5.10.16 Dynamic Memory Allocation

One of the key advantages of pointers is their use in dynamic memory allocation. So far, we have created derived data types like arrays at compile time. They are created at compile time by specifying the size in the source code, which is fixed and cannot be modified at run time. This process of allocating memory at compile time is known as static memory allocation. This approach works fine as long as we know exactly what our data requirements are.

Consider a situation where we want to use an array that can vary greatly in size. We must guess what will be the largest size ever needed and create the array accordingly. A difficult task in fact! Modern languages like C do not have this limitation. In C it is possible to allocate memory to derived data types like arrays at run time. This feature is known as dynamic memory allocation.

Dynamic memory allocation is done with the help of pointer variables and memory management functions such as malloc, calloc and realloc. These functions are included in the header file <stdlib.h>. The concept of dynamic arrays is used in creating and manipulating data structures such as linked lists, stacks and queues.

malloc The malloc function is used to allocate a block of memory whose size and type is specified during run time. It takes the following syntax:

```
variable_name = (Data_Type*) malloc(size)
```

Example:

```
int *p;
p = (int*) malloc (25);
```

The above expression when executed reserves 25 bytes in the memory space, and returns the address of the first byte of the memory block to the pointer variable p of type int.

calloc The calloc function is used to allocate multiple blocks of memory whose size and type is specified during run time. It takes the following syntax:

```
variable_name = (Data_Type*) calloc(size1. size2)
```

Example:

```
int *p;
p = (int*) calloc (5,2);
```

The above expression when executed reserves 5 blocks in the memory space with each block containing two bytes each.

realloc The realloc function is used to reallocate the already allocated memory space. It is basically used to either shrink or enlarge an already reserved memory space as per requirements. It sends the address of the revised memory block to the pointer variable. It takes the following syntax:

```
variable_name = (Data_Type*) realloc(size1)
```

Example:

```
int *p;
p = (int*) realloc (10);
```

free The free function is used to release the memory space already allocated with malloc or calloc functions. It takes the following syntax:

```
free(variable_name);
```

Example:

```
free(p);
```

The above expression when executed will release the reserved memory space pointed by pointer variable p.

EXAMPLE 5.27 *Figure 5.67 shows a program to implement the queue data structure using dynamic memory allocation.*

```
#include<stdio.h>
#include<alloc.h>
#include<conio.h>
#define size 10
#define true 1
#define false 0

struct que
{
 int front,rear;
 int num;
 int arr[size];
};

void init(struct que* queue);
int empty(struct que* queue);
int full(struct que* queue);
```

```
int add(struct que* queue,int);
int rem(struct que* queue);
void display(struct que* queue);

void main()
{
 int ele,k;
 int ch;

 struct que *queue = (struct que*)malloc(sizeof(struct que));
 init(queue);

 while(1)
 {
 clrscr();
 printf("=========================================");
 printf("\n\t\tMENU\n");
 printf("=========================================");
 printf("\n\t[1] To insert an element into the queue");
 printf("\n\t[2] To remove an element from the queue");
 printf("\n\t[3] To display all the elements in the queue");
 printf("\n\t[4] Exit");
 printf("\n\n\t Enter your choice: ");
 scanf("%d",&ch);

 switch(ch)
 {
 case 1:
 {
      printf("\nEnter the element to be inserted:");
      scanf("%d",&ele);
      add(queue,ele);
      break;
 }

 case 2:
 {
      if(!empty(queue))
      {
       k=rem(queue);
       printf("\n%d element is removed\n",k);
       getch();
      }
      else
      {
       printf("\tQueue is Empty. No element can be removed.");
       getch();
      }
      break;
 }

 case 3:
 {
      display(queue);
      getch();
      break;
 }
```

```
 case 4:
      exit(0);

 default:
      printf("\tIncorrect Choice.");
      getch();
      break;
 }
 }
}

void init(struct que* queue)
{
 queue->front = 0;
 queue->rear = -1;
 queue->num = 0;
}

int empty(struct que* queue)
{
 if(queue->num==0)
 return true;
 return false;
}

int full(struct que* queue)
{
 if(queue->num == size)
 return true;
 return false;
}

int add(struct que* queue,int j)
{
 if(full(queue))
 return false;

 if(queue->rear == size - 1)
 queue->rear = -1;
 queue->arr[++queue->rear] = j;
 queue->num++;
 return true;
}

int rem(struct que* queue)
{
 int j;
 if(empty(queue))
 return -9999;
 j = queue->arr[queue->front++];
 if(queue->front == size)
 queue->front = 0;
 queue->num--;
 return j;
}
```

```
void display(struct que* queue)
{
 int j;
 if(empty(queue))
 {
 printf("Queue is Empty. No records to display.");
 return;
 }
 printf("\nElements present in the Queue are: ");
 for(j=queue->front;j<=queue->rear;j++)
 printf("%d\t",queue->arr[j]);
 printf("\n");
}
Output
==========================================
 MENU
==========================================
 [1] To insert an element into the queue
 [2] To remove an element from the queue
 [3] To display all the elements in the queue
 [4] Exit

Enter your choice: 1
Enter the element to be inserted: 25
==========================================
 MENU
==========================================
 [1] To insert an element into the queue
 [2] To remove an element from the queue
 [3] To display all the elements in the queue
 [4] Exit

Enter your choice: 1
Enter the element to be inserted: 50
==========================================
 MENU
==========================================
 [1] To insert an element into the queue
 [2] To remove an element from the queue
 [3] To display all the elements in the queue
 [4] Exit

Enter your choice: 3
Elements present in the Queue are: 25 50
==========================================
 MENU
==========================================
 [1] To insert an element into the queue
 [2] To remove an element from the queue
 [3] To display all the elements in the queue
 [4] Exit

Enter your choice: 2
25 element is removed
==========================================
```

```
MENU
=============================================
 [1] To insert an element into the queue
 [2] To remove an element from the queue
 [3] To display all the elements in the queue
 [4] Exit

Enter your choice: 3
Elements present in the Queue are: 50
```

Fig. 5.67

The above program uses the malloc function to reserve memory space for the queue during run time. When the end user chooses option '4' to quit the program, the memory space allocated to the queue data structure is released with the help of free function.

5.11 CASE STUDY

Processing of Examination Marks

Marks obtained by a batch of students in the Annual Examination are tabulated as follows:

Student name	Marks obtained
S. Laxmi	45 67 38 55
V.S. Rao	77 89 56 69
-	- - - -

It is required to compute the total marks obtained by each student and print the rank list based on the total marks.

The program in Figure 5.68 stores the student names in the array name and the marks in the array marks. After computing the total marks obtained by all the students, the program prepares and prints the rank list.

```
Program
#define STUDENTS 5
#define SUBJECTS 4
#include <string.h>
main()
{
char name[STUDENTS][20];
int marks[STUDENTS][SUBJECTS+1];
printf("Input students names & their marks in four subjects\n");
 get_list(name, marks, STUDENTS, SUBJECTS);
 get_sum(marks, STUDENTS, SUBJECTS+1);
 printf("\n");
 print_list(name,marks,STUDENTS,SUBJECTS+1);
 get_rank_list(name, marks, STUDENTS, SUBJECTS+1);
 printf("\nRanked List\n\n");
 print_list(name,marks,STUDENTS,SUBJECTS+1);
}
```

```
     /* Input student name and marks */
get_list(char *string[ ],
 int array [ ] [SUBJECTS +1], int m, int n)
{
int i, j, (*rowptr)[SUBJECTS+1] = array;
for(i = 0; i < m; i++)
{
        scanf("%s", string[i]);
 for(j = 0; j < SUBJECTS; j++)
 scanf("%d", &(*(rowptr + i))[j]);
     }
}
/* Compute total marks obtained by each student */
get_sum(int array [ ] [SUBJECTS +1], int m, int n)
{
int i, j, (*rowptr)[SUBJECTS+1] = array;
 for(i = 0; i < m; i++)
 {
        (*(rowptr + i))[n-1] = 0;
     for(j =0; j < n-1; j++)
 (*(rowptr + i))[n-1] += (*(rowptr + i))[j];
 }
}
 /* Prepare rank list based on total marks */

get_rank_list(char *string [ ],
 int array [ ] [SUBJECTS + 1]
        int m,
        int n)
{
 int i, j, k, (*rowptr)[SUBJECTS+1] = array;
 char *temp;

 for(i = 1; i <= m–1; i++)
 for(j = 1; j <= m–i; j++)
 if( (*(rowptr + j–1))[n–1] < (*(rowptr + j))[n–1])
 {
     swap_string(string[j-1], string[j]);
  for(k = 0; k < n; k++)
 swap_int(&(*(rowptr + j–1))[k],&(*(rowptr+j))[k]);
 }
}
 /* Print out the ranked list */
print_list(char *string[ ],
int array [] [SUBJECTS + 1],
 int m,
 int n)
{
        int i, j, (*rowptr)[SUBJECTS+1] = array;
     for(i = 0; i < m; i++)
     {
        printf("%–20s", string[i]);
        for(j = 0; j < n; j++)
     printf("%5d", (*(rowptr + i))[j]);
     printf("\n");
     }
 }
```

```
 /* Exchange of integer values */
swap_int(int *p, int *q)
{
int temp;
 temp = *p;
 *p = *q;
 *q = temp;
}

/* Exchange of strings */
swap_string(char s1[ ], char s2[ ])
{
 char swaparea[256];
 int i;
 for(i = 0; i < 256; i++)
 swaparea[i] = '\0';
 i = 0;
 while(s1[i] != '\0' && i < 256)
 {
 swaparea[i] = s1[i];
 i++;
 }
 i = 0;
 while(s2[i] != '\0' && i < 256)
 {
        s1[i] = s2[i];
     s1[++i] = '\0';
 }
 i = 0;
 while(swaparea[i] != '\0')
 {
        s2[i] = swaparea[i];
     s2[++i] = '\0';
 }
}

Output
 Input students names & their marks in four subjects
 S.Laxmi    45    67    38    55
 V.S.Rao    77    89    56    69
 A.Gupta    66    78    98    45
 S.Mani     86    72    0     25
 R.Daniel   44    55    66    77

 S.Laxmi    45    67    38    55    205
 V.S.Rao    77    89    56    69    291
 A.Gupta    66    78    98    45    287
 S.Mani     86    72    0     25    183
 R.Daniel   44    55    66    77    242

 Ranked List
 V.S.Rao    77    89    56    69    291
 A.Gupta    66    78    98    45    287
 R.Daniel   44    55    66    77    242
 S.Laxmi    45    67    38    55    205
 S.Mani     86    72    0     25    183
```

Fig. 5.68 **Preparation of the rank list of a class of students**

In the above program, the following declaration defines marks as a pointer to the array's first row.

```
int marks[STUDENTS][SUBJECTS+1];
```

We use rowptr as the pointer to the row of marks. The rowptr is initialized as follows:

```
int (*rowptr)[SUBJECTS+1] = array;
```

Note that array is the formal argument whose values are replaced by the values of the actual argument marks. The parentheses around *rowptr makes the rowptr as a pointer to an array of SUBJECTS+1 integers.

Remember, the following statement would declare rowptr as an array of SUBJECTS+1 elements.

```
int *rowptr[SUBJECTS+1];
```

When we increment the rowptr (by rowptr+1), the incrementing is done in units of the size of each row of array, making rowptr point to the next row. Since rowptr points to a particular row, (*rowptr)[x] points to the xth element in the row.

2. Inventory Updating

The price and quantity of items stocked in a store changes every day. They may either increase or decrease. The program in Figure 5.69 reads the incremental values of price and quantity and computes the total value of the items in stock.

```
Program
struct stores
{
char name[20];
 float price;
 int quantity;
};
main()
{
void update(struct stores *, float, int);
 float p_increment, value;
 int q_increment;
 struct stores item = {"XYZ", 25.75, 12};
 struct stores *ptr = &item;
 printf("\nInput increment values:");
 printf(" price increment and quantity increment\n");
 scanf("%f %d", &p_increment, &q_increment);
 /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
 update(&item, p_increment, q_increment);
 /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
 printf("Updated values of item\n\n");
 printf("Name : %s\n",ptr->name);
 printf("Price : %f\n",ptr->price);
 printf("Quantity : %d\n",ptr->quantity);
  /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

```
 value = mul(&item);
 /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
 printf("\nValue of the item = %f\n", value);
}
void update(struct stores *product, float p, int q)
{
product–>price += p;
 product–>quantity += q;
}
float mul(struct stores *stock)
{
return(stock–>price * stock–>quantity);
}
Output
 Input increment values: price increment and quantity increment
 10 12
 Updated values of item
 Name : XYZ
 Price : 35.750000
 Quantity : 24
 Value of the item = 858.000000
```

Fig. 5.69 **Use of structure pointers as function parameters**

The above program illustrates the use of structure pointers as function parameters. &item, the address of the structure item, is passed to the functions update() and mul(). The formal arguments product and stock, which receive the value of &item, are declared as pointers of type struct stores.

5.12 PREPROCESSOR DIRECTIVES

Preprocessor directives are used to specify special instructions in a program which are processed before the actual program is executed. Preprocessor directives are recognized by the hash sign, #, placed before them. Semicolon is not placed at the end of these statements as is done in the case of other statements in C. These statements usually get completed in a single line and are not continued to the next line. The common preprocessor directives are given below.

5.12.1 Macro Directives

Macros are used for the purpose of defining symbolic constants. In C programs, macros are specified using the following syntax:

```
#define (identifier name) (value/content of macro)
```

At the instance of preprocessing of the program, whenever the identifier name is encountered, the preprocessor replaces it with the value specified in the macro definition. The value specified may be a string or any integer.

#define This directive is used to define constants or macros using the following syntax:

```
#define <identifier> <replacement name>
```

The execution of this directive will replace the identifier coming anywhere in the code with the replacement name. For example, suppose max_marks is the identifier, which is to be replaced by 100 everywhere in the program. It can be implemented as follows using the #define directive:

```
#define max_marks 100
int marks1[max_marks];
int marks2[max_marks];
```

The code will now become equivalent to the following statements:

```
int marks1[100];
int marks2[100];
```

In a C program, the macros are defined before the main function. Even multiple macro definitions can be included in a C program.

EXAMPLE 5.28 *Figure 5.70 shows a very basic program depicting the usage of macros in C:*

```
Program
#include <stdio.h>
#include <conio.h>
#define PROFIT 5
void main( )
{
int x, y;
clrscr();
printf("Enter the value of COST PRICE ");
scanf("%d", &x);
y = x+PROFIT;
printf("\nValue of SELLING PRICE is %d", y);
}
Output
Enter the value of COST PRICE 5
Value of SELLING PRICE is 10
```

Fig. 5.70 Use of macros in C

In the above code, the occurrence of 'PROFIT' is replaced with its integer value 5 by the C preprocessor.

#undef This directive is used to undefine a macro, which has been defined before so that it can be defined for a different value. To undefine a defined macro, the following syntax is used:

#undef Name

For example, the identifier max_marks is undefined as follows:

#undef max_marks

5.12.2 File Inclusion

#include This directive is used to include another file in the program. It can include file using the following two ways:

```
#include <filename>
#include "filename"
```

If the name of the file to be included is specified within <> symbol, then the compiler will search the file in the directory specified as the include directory. So, the standard header files are generally specified within angular quotes. If the name of the file to be included is specified within " " symbol, then the compiler searches the file in all the directories in the current drive.

All the programs that we have seen so far have used at least one #include directive.

5.12.3 Conditional Inclusion

Some directives are used to compile a part of the program based on certain conditions. It means that if a specific condition is fulfilled, then the program will be executed otherwise not. These conditional directives are # if, # ifdef, # ifndef, # elif, # else and # endif, as explained below:

#ifdef This directive is used to compile a part of the program only if the macro is defined as a parameter in the program irrespective of its value. The use of #ifdef can be explained with the help of the following example:

```
#ifdef max_marks
int marks[max_marks];
#endif
```

In the above example, the second statement will be executed only if the macro max_marks has been defined before in the program. If it has not been defined previously, then the int statement will not be included in the program execution.

#ifndef This directive is the opposite of #ifdef directive. It means that the statements written between #ifndef and #endif will be executed only if the identifier has not been defined previously. The use of #ifnfdef can be explained with the help of the following example:

```
#ifndef max_marks
#define max_marks 100
#endif
```

According to this directive, the second statement will be executed only if the macro max_marks has not been defined earlier.

#if, #else, #elif These directives are used in between a program to allow or prevent the further execution of the program. #if is used to check whether the result of a given expression

is zero or not. If the result is not zero, then the statements after #if are compiled else not. The use of #if, #else and #elif can be explained with the help of the following example:

```
main()
{
#if AGE<=18

printf("Juvenile");
#elif AGE>=18 AGE<=50
printf("Adult");
#else
printf("Aged");
#endif
}
```

5.13 DEVELOPING A C PROGRAM – SOME GUIDELINES

While writing a C program the programmer must follow certain standard guidelines so as to develop the code efficiently and correctly. The following are some key guidelines that are always advisable to be kept in mind while developing a C program:

- **Coding character set** A programmer must use the ASCII character set for coding. Any non-ASCII characters lead to confusion and may even cause errors on some compilers.
- **Identifiers** A programmer must be careful while naming the identifiers. He should follow standard conventions while naming them.
- **Declaration** All the variables and identifiers must be declared before they are used in the code.
- **Naming conventions** While naming variables and identifiers a programmer can use an underscore (_)and an upper case for initial character (A – Z). In addition, natural language must be used for naming the variables and identifiers.
- **Naming restrictions** A programmer must take care of the restrictions related the naming of files, variables and functions in C programming language. The number of characters of an external identifier that are supported by the linker are restricted. No identifier must begin with an underscore (_). Besides this the keywords are also restricted from being used as the name of an identifier.
- **Complexity** A programmer must take special care of the complexity of the code. The excessive use of nested expressions may cause unreadability and complexity. The code segments must also not be repeated as it may lead to larger number of errors.
- **Modularity** A programmer must use the concept of modularity to manage the complexity of the code.
- **Buffer** A programmer must not define functions to which buffer has to be passed as an argument because the overflow of buffer can cause corruption of the adjacent data.
- **Data type** A programmer must take special care while declaring the data type of the variables. The data type of the variable containing the result may vary on the basis of the operations performed.

- **Goto** The use of goto statement should be as much avoided as possible.
- **Array** While referring an array, it must not run out of bounds.
- **Arithmetic expressions** The program expressions should always be checked to avoid any 'divide by zero' condition.

SUMMARY

In this chapter we learned about some of the advanced concepts of C like arrays, functions, pointers, structures and unions. An array is a collection of similar data type variables. It is a very crisp and concise method of handling large group of similar data values, like name of customers, marks of students, etc. Structures and unions are very much similar to arrays in the sense that they are also used to group multiple data values together. But they are fundamentally different due to the fact that structures and unions can club together variables of unlike types.

Sometimes in complex programs, the repetition of a particular block of statements increases the size of the program. In order to avoid this situation, functions are used in C programs. Functions are the blocks of code defined to perform a specific task. In a C program, one or more functions can be used. Functions are called in the main ( ) function and are defined outside the main function. We can call a function using the function name with the required number of arguments in the parentheses. Arguments are specified in the functions to pass the values between the main program and the function.

Later in this chapter, we learned about one of the most important concepts of C language, that is pointers. A variable pointing to another variable is known as pointer. Pointers are used in C to preserve memory space as they help in dynamic memory allocation. Finally, we learned about some important standards and conventions that should be kept in mind while developing C programs.

POINTS TO REMEMBER

- **Array**: It is a fixed-size sequenced collection of elements of the same data type.
- **Structured data types**: These data types provide an organizational scheme that shows the relationships among the individual elements and facilitate efficient data manipulations.
- **One-dimensional array**: In this array, a list of items is given one variable name using only one subscript.
- **Two-dimensional array**: It is used to store a table of values with two dimensions.
- **Multi-dimensional array**: It is used to store data with three or more dimensions.
- **Static memory allocation**: It is the process of allocating memory at compile time.
- **Static arrays**: These are the arrays, which receive static memory allocation.
- **Dynamic memory allocation**: It is the process of allocating memory at run time.
- **Dynamic arrays**: These are the arrays created at run time.
- **String**: It is a sequence of characters that is treated as a single data item.
- **strcat**: It is a function used to join two strings together.

- **strcmp**: It is a function used to compare two strings passed as arguments. It generates a value of 0 if they are equal.
- **strcpy**: It is a function used to copy one string into another.
- **Modular programming**: It is a programming approach of organizing a large program into small, independent program segments known as modules.
- **Program definition**: It is an independent program module that is specially written to implement the requirements of the function.
- **Calling program**: It is the program that calls the function.
- **Function type**: It specifies the type of value that the function is supposed to return.
- **Parameter list**: It declares the variables that will receive the data sent by the calling program.
- **Function body**: It contains the declarations and statements necessary for performing the required task.
- **Local variables**: The variables declared inside a function are known as local variables.
- **Recursion**: It is the process in which a called function in turn calls another function.
- **Internal variables**: These are the variables, which are declared within a particular function
- **External variables**: These variables are declared outside of any function. These are alive and active throughout the entire program.
- **Block statement**: It is a set of statements enclosed in a set of braces.
- **Structure**: It is a collection of related data elements of different types.
- **Dot operator**: It is a member operator used to identify the individual members in a structure.
- **Pointer**: It is a derived data type that contains memory address as its value.
- **Memory**: It is a sequential collection of storage cells.
- **Pointer variables**: These variables hold memory addresses and are stored in the memory.
- **Call by reference**: It is the process of calling a function using pointers to pass the address of the variables.
- **Call by value**: It is the process of passing the actual value of variables.

REVIEW QUESTIONS

TRUE OR FALSE

1. The type of all elements in an array must be the same.
2. Accessing an array outside its range is a compile time error.
3. A char type variable cannot be used as a subscript in an array.
4. When we use expressions as a subscript, its result should be always greater than zero.
5. While declaring an array, the array size can be a constant or variable or an expression.
6. The declaration int x[2] = {1,2,3}; is illegal.

7. When initializing a string variable during its declaration, we must include the null character as part of the string constant, like "GOOD\0".
8. When reading a string with scanf, it automatically inserts the terminating null character.
9. We cannot perform arithmetic operations on character variables.
10. The function scanf cannot be used in any way to read a line of text with the white-spaces.
11. The function call strcpy(s2, s1); copies string s2 into string s1.
12. C functions can return only one value under their function name.
13. A function in C should have at least one argument.
14. A user-defined function must be called at least once; otherwise a warning message will be issued.
15. Any name can be used as a function name.
16. Program execution always begins in the main function irrespective of its location in the program.
17. Global variables are visible in all blocks and functions in the program.
18. A function can call itself.
19. A function prototype must always be placed outside the calling function.
20. In parameter passing by pointers, the formal parameters must be prefixed with the symbol * in their declarations.
21. In passing arrays to functions, the function call must have the name of the array to be passed without brackets.
22. Pointer variables are declared using the address operator.
23. Pointers to pointers is a term used to describe pointers whose contents are the address of another pointer.
24. When an array is passed as an argument to a function, a pointer is passed.
25. Value of a local variable in a function can be changed by another function.

FILL IN THE BLANKS

1. The variable used as a subscript in an array is popularly known as _____ variable.
2. An array that uses more than two subscript is referred to as _____ array.
3. _____ is the process of arranging the elements of an array in order.
4. We can initialize a string using the string manipulation function _____.
5. The function strcat has _____ parameters.
6. The function _____ is used to determine the length of a string.
7. The function call strcat (s2, s1); appends _____ to _____.
8. The printf may be replaced by _____ function for printing strings.
9. The parameters used in a function call are called _____.
10. A variable declared inside a function is called _____

11. By default, _____ is the return type of a C function.
12. _____ refers to the region where a variable is actually available for use.
13. A function that calls itself is known as a _____ function.
14. If a local variable has to retain its value between calls to the function, it must be declared as _____.
15. A _____ is a collection of data items under one name in which the items share the same storage.
16. The variables declared in a structure definition are called its _____.
17. A pointer variable contains as its value the _____ of another variable.
18. The _____ operator returns the value of the variable to which its operand points.

MULTIPLE CHOICE QUESTIONS

1. Which of the following is not a data structure?
   A. Linked list B. Stack
   C. Tree D. Pointer
2. int a[n] will reserve how many locations in the memory?
   A. n B. n-1
   C. n+1 D. None of the above
3. Which of the following is the correct syntax for initialization of one-dimensional arrays?
   A. num[3]= {0 0 0 }; B. num[3]= {0, 0, 0 };
   C. num[3]= {0; 0 ;0 }; D. num[3]= 0
4. Which of the following is the correct syntax for initialization of two-dimensional arrays?
   A. table[2][3]={0,0,0,1,1,1} B. table[2][3]={
   {0,0,0}
   {1,1,1}
   }
   C. table[2][3]={0,1}, {0,1},{0,1} D. None of the above
5. Which of the following multi-dimensional array declaration is correct for realizing a 2 ×3 matrix?
   A. int m[2][3]; B. int m[3][2];
   C. int m[3], m[2]; D. None of the above
6. Which of the following is not the name of a sorting technique?
   A. Bubble B. Selection
   C. Binary D. Insertion

7. In a character string, the last element stores which of the following values?
   A. Last element of the string
   B. Blank space
   C. Null character
   D. Newline character
8. Which of the following is used to display a string on the I/O console?
   A. %s
   B. %c
   C. %d
   D. %f
9. What will be the result of the following character arithmetic expression:
   X= 'A'-2;
   A. 63
   B. 64
   C. 65
   D. 66
10. Which of the following is used to determine the length of a string?
    A. strlen
    B. strcmp
    C. strcpy
    D. strcat
11. Which of the following header files are required to be included for performing string operations?
    A. string.h
    B. conio.h
    C. stdio.h
    D. ctype.h
12. Recursion is a situation where:
    A. A function calls the main function
    B. A function calls any of the system functions
    C. A function calls itself
    D. None of the above
13. The formal arguments in the function header must be prefixed by which of the following indirection operator?
    A. *
    B. +
    C. -
    D. /
14. Which of the following is an incorrect way of declaring structure type variables?
    A.
    ```
    struct book
    {
    Datatypes
    }book1, book2;
    ```
    B.
    ```
    struct book
    {
    Datatypes
    };
    Struct book book1, book2;
    ```
    C.
    ```
    struct book1, book2
    {
    Datatype
    };
    ```
    D. All of the above are correct
15. Which of the following is the correct way of assigning a value to the 'name' field of a structure 'book'?
    A. book.name
    B. book->name
    C. book(name)
    D. None of the above

16. Which of the following expressions are correct for accessing the 'num' variable value of the i^{th} element of a structure array 'student'?
    A. student[i].num
    B. student.num[i]
    C. student[i]->num
    D. None of the above
17. Pointer is an example of which of the following type?
    A. Derived type
    B. Fundamental type
    C. User-defined type
    D. None of the above
18. In the expression *ptr=&a, what does & signify
    A. Address of a
    B. Address of ptr
    C. Value of a
    D. None of the above
19. Which of the following expressions in C is used for accessing the address of a variable var?
    A. &var
    B. *var
    C. &&var
    D. ***var
20. Which of the following is a syntactically correct pointer declaration?
    A. float *x;
    B. float* x;
    C. float * x;
    D. All of the above are correct
21. Which of the following pointer expressions will give the value stored in variable x?
    A. &&x
    B. *x
    C. *&x
    D. &x
22. If a1=&x and a2 =&a1, what will be the output generated by the expression **a2?
    A. Address of a2
    B. Address of a1
    C. Value of x
    D. Address of x
23. What will be the expression for obtaining the address of the i^{th} element of an array A?
    A. A[i]
    B. &A[i]
    C. *A[i]
    D. A[&i]

ANSWERS

True or False

1. True	2. False	3. True	4. True	5. False	6. False	7. False	8. True
9. False	10. False	11. False	12. True	13. False	14. True	15. False	16. True
17. True	18. True	19. False	20. True	21. True	22. False	23. True	24. True
25. False							

Fill in the Blanks

1. Subscripted variable	2. Multi dimensional array	3. Sorting
4. strcpy()	5. Two	6. strlen()
7. s1,s2	8. puts	9. actual parameters

10. local variable	11. integer	12. scope
13. recursive	14. Global variable	15. union
16. members	17. address	18. indirection

Multiple Choice Questions

1. D	2. C	3. B	4. B	5. A	6. C	7. C	8. A
9. A	10. A	11. A	12. C	13. A	14. C	15. A	16. B
17. A	18. A	19. A	20. A	21. B	22. C	23. B	

EXERCISE QUESTIONS

1. Write a program to extract a portion of a character string and print the extracted string. Assume that m characters are extracted, starting with the n^{th} character.
2. Write a program to replace a particular word by another word in a given string. For example, the word "PASCAL" should be replaced by "C" in the text "It is good to program in PASCAL language."
3. Describe the two ways of passing parameters to functions. When do you prefer to use each of them?
4. What is prototyping? Why is it necessary?
5. Write a program that reads a string from the keyboard and determines whether the string is a palindrome or not. (A string is a palindrome if it can be read from left and right with the same meaning. For example, Madam and Anna are palindrome strings. Ignore capitalization).
6. Write program that reads the cost of an item in the form RRRR.PP (Where RRRR denotes Rupees and PP denotes Paisa) and converts the value to a string of words that expresses the numeric value in words. For example, if we input 125.75, the output should be "ONE HUNDRED TWENTY FIVE AND PAISE SEVENTY FIVE".
7. Develop a program that will read and store the details of a list of students in the format

Roll No.	Name	Marks obtained
_____	_____	_____
_____	_____	_____
_____	_____	_____

   and produce the following output list:
   (a) Alphabetical list of names, roll numbers and marks obtained.
   (b) List sorted on roll numbers.
   (c) List sorted on marks (rank-wise list)
8. Write a function to calculate the roots. The function must use two pointer parameters, one to receive the coefficients a, b, and c, and the other to send the roots to the calling function.

9. Distinguish between the following:
   (a) Actual and formal arguments
   (b) Global and local variables
   (c) Automatic and static variables
   (d) Scope and visibility of variables
   (e) & operator and * operator

9. Define a structure data type called time_struct containing three members integer hour, integer minute and integer second. Develop a program that would assign values to the individual members and display the time in the following form:

   16:40:51

11. Modify the above program such that a function is used to input values to the members and another function to display the time.
12. Define a structure data type named date containing three integer members day, month and year. Develop an interactive modular program to perform the following tasks;
    - To read data into structure members by a function
    - To validate the date entered by another function
    - To print the date in the following format by a third function.

      April 29, 2009

    The input data should be three integers like 29, 4, and 2009 corresponding to day, month and year. Examples of invalid data:

    31, 4, 2009 – April has only 30 days

    29, 2, 2009 – 2009 is not a leap year

13. Create two structures named metric and British which store the values of distances. The metric structure stores the values in meters and centimeters and the British structure stores the values in feet and inches. Write a program that reads values for the structure variables and adds values contained in one variable of metric to the contents of another variable of British. The program should display the result in the format of feet and inches or metres and centimetres as required.
14. Define a structure named census with the following three members:
    - A character array city [ ] to store names
    - A long integer to store population of the city
    - A float member to store the literacy level
    - Write a program to do the following:
    - To read details for 5 cities randomly using an array variable
    - To sort the list alphabetically
    - To sort the list based on literacy level
    - To sort the list based on population
    - To display sorted lists

15. Define a structure that can describe a hotel. It should have members that include the name, address, grade, average room charge, and number of rooms.
    - Write functions to perform the following operations:
    - To print out hotels of a given grade in order of charges
    - To print out hotels with room charges less than a given value
16. Define a structure called cricket that will describe the following information:

    player name

    team name

    batting average

    Using cricket, declare an array player with 50 elements and write a program to read the information about all the 50 players and print a team-wise list containing names of players with their batting average.
17. Write a function exchange to interchange the values of two variables, say x and y. Illustrate the use of this function, in a calling function. Assume that x and y are defined as global variables.
18. The Fibonacci numbers are defined recursively as follows:

    F1 = 1

    F2 = 1

    Fn = F n–1+F n–2, n > 2
19. Write a function that will generate and print the first n Fibonacci numbers. Test the function for n = 5, 10, and 15.
20. Write a function that will round a floating-point number to an indicated decimal place. For example the number 17.457 would yield the value 17.46 when it is rounded off to two decimal places.
21. Write a function prime that returns 1 if its argument is a prime number and returns zero otherwise.
22. Write a function that will scan a character string passed as an argument and convert all lowercase characters into their uppercase equivalents.
23. Write a function that can be called to find the largest element of an m by n matrix.
24. Design and code an interactive modular program that will use functions to a matrix of m by n size, compute column averages and row averages, and then print the entire matrix with averages shown in respective rows and columns.

ANSWERS TO 2009 QUESTION PAPERS

PART A

1. State the differences between union and structure? (Anna University, Jan - Feb 2009)

   Ans: Refer Section 5.8.10.
2. What is meant by nested structure? (Anna University, Jan - Feb 2009, GE1102)

   Ans: Refer Section 5.8.9.

3. What are various string handling functions? (Anna University, Jan - Feb 2009)
   Ans: Refer Section 5.4.4.
5. Is it possible to refer to the elements of an array by using pointer notation? If so, give an example. (Anna University, Jan - Feb 2009)
   Ans: Refer Section 5.10.9.
6. What is recursion? Give its application. (Anna University, Jan - Feb 2009)
   Ans: Refer Section 5.6.5.
7. What are the advantages of pointers? (Anna University, Jan - Feb 2009)
   Ans: Refer Section 5.10.
8. Write the limitations of using getchar( ) and scanf( ) functions for reading strings. (GE2112)
   Ans: Refer Section 5.4.2.
9. Define a C function to exchange the content of two variables. (GE2112)
   Ans: Refer Section 5.10.12.
10. What are the operators exclusively used with pointers? (GE1102)
    Ans: Refer Section 5.10.7.

PART B

1. Explain about structures and unions with suitable examples. (GE2112)
   Ans: Refer Section 5.8.
2. Write note on pointers. (GE2112)
   Ans: Refer Section 5.10.1.
3. Explain the need for array variables. With respect to arrays, describe the following: Declaration of arrays, Two-dimensional array and Accessing an array element. (GE1102)
   Ans: Refer Section 5.2.
4. What are the advantages of using pointers? How are pointers declared and initialized? How the value of a variable is accessed using pointers? Give examples. (GE1102)
   Ans: Refer Section 5.10.
5. Differentiate between array of pointers and pointers to array, with example. (GE1102)
   Ans: Refer Sections 5.10.9 and 5.10.12.
6. How can a function return a pointer to its calling function? (GE1102)
   Ans: Refer Section 5.10.13.
7. What is a structure? How does a structure differ from an array? (GE1102)
   Ans: Refer Section 5.8.1.
8. 10 How an array of structure is initialized? (GE1102)
   Ans: Refer Section 5.8.7.
9. How a structure member be accessed? (GE1102)
   Ans: Refer Section 5.8.3.

10. How does structure differ from union? (GE1102)

    Ans: Refer Section 5.8.10.

11. Write a C program using a user-defined function to sort numbers in descending order.
    (Anna University, Jan - Feb 2009)

    Ans:

```
Program
#include<conio.h>
#include<stdio.h>
void main()
{
 int limit;
 clrscr();
 printf("Enter no. of elements in the list:- ");
 scanf ("%d",&limit);
 sort(limit);
 getch();
}
sort(int upper)
{
 int i,j,temp=0,min,k,array[20];
 for (i=1;i<=upper;i++)
 {
 printf("\nEnter the list item %d : ",i);
 scanf("%d",&array[i]);
 }
 for (i=1;i<=(upper-1);i++)
 {
 min=array[i];
 k=i;
 for (j=(i+1);j<=upper;j++)
 {
 if (array[j]<min)
 {
 min=array[j];
 k=j;
 }
 }
 temp=array[k];
 array[k]=array[i];
 array[i]=temp;
 }
printf("\nThe descending order of list is:\n");
for (i=upper;i>=1;i--)
    printf("\n%d",array[i]);
}
Output
Enter no. of elements in the list:- 3
Enter the list item 1 : 45
Enter the list item 2 : 85
```

```
Enter the list item 3 : 26
The descending order of list is:
85
45
26
```

12. Explain in detail how an array can be passed as a parameter in a user-defined function. Illustrate your answer with an example program. (Anna University, Jan - Feb 2009)

Ans: Refer Section 5.6.7.

13. Write a C program to copy one string into another and count the number of characters copied? (Anna University, Jan - Feb 2009)

```
Program
#include<conio.h>
#include<stdio.h>
#include<string.h>
main()
{
char string1[20],string2[20];
int x,l1,l2;
clrscr();
printf("\nEnter String1:\t");
scanf("%s",string1);
printf("\nEnter String2:\t");
scanf("%s",string2);
printf("\nBefore copying: \nString1:\t %s \nstring2:\t %s",string1,string2);
strcpy(string2,string1);
l1=strlen(string1);
l2=strlen(string2);
printf("\n\nAfter copying: \nString1:\t %s \nstring2:\t %s",string1,string2);
printf("\n\nCharacters copied from string1 to string2:\t %d",l2);
getch();
}
Output
Enter String1: Abhishek
Enter String2: Sharma
Before copying:
String1: Abhishek
string2: Sharma
After copying:
String1: Abhishek
string2: Abhishek
Characters copied from string1 to string2: 8
```

14. State the various issues when structure has been used as an argument for function. (Anna University, Jan - Feb 2009)

Ans: Refer heading 5.8.10.

ANSWERS TO 2010 AND 2011 QUESTION PAPERS

SHORT ANSWER QUESTIONS

1. Write a C program for the following expressions.
   (i) a=5<=8 && 6!=5
   (ii) a=b++ + ++b where b=50 (AU, Chn, Jan 2010)

Ans:

(i) a=5<=8 && 6!=5

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a;
a = 5<=8 && 6!=5;
printf("%d",a);
getch();
}
Output:
1
```

(ii) a=b++ + ++b where b=50

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a,b=50;
clrscr();
a = b++ + ++b;
printf("%d",a);
getch();
}
Output:
102
```

2. How are strings represented in C language? (AU, Chn, Jan 2010)

Ans: Strings are represented in C with the help of character arrays. Following are the examples of string declarations in C:

```
char name[50];
char city [15] = "New York";
```

3. What are the advantages of unions over structures? (AU, Chn, Jan 2010)

   Ans: The advantages of unions over structures are:

   - Unions save memory space as the size of a union variable is equal to its largest sized member. In contrast, the size of a structure variable is equal to the sum of the sizes of all its individual member elements.
   - Unions are particularly useful in situations where there is a need to use only one of its member elements at any given point of time.

4. What is the need for user defined functions? (AU, Chn, Jan 2011)

   Ans: Refer Section 5.6.1.

5. What is call by value? (AU, Chn, Jan 2011)

   Ans:

   Call by value

   In this technique, a function is called by passing the value of a variable as argument. The passed value is copied into another variable in the parameter list of the function. Thus, any change made to the new variable's value inside the called function is not reflected back in the calling function. The following code snippet depicts this scenario:

Code

```
void change(int n)
{
   n=100;
}
void main()
{
 int num=50;
 printf("Before function call, num = ",num);
 change(num);
 printf("\nAfter function call, num = ",num);
getch();
}
```

Output

```
Before function call, num = 50
After function call, num = 50
```

6. What is a pointer? List out any two features of it? (AU, Mdu, Jan 2011)

   Ans: Refer Section 5.10.

7. Difference between Call by Reference and Call by Value. (AU, Cbe, Dec 10-Jan 11)

   Ans: In 'call by value' function call, only the values of the variables are passed as parameters. Thus, any change made to these values by the function code is not reflected at the original variable storage location. Alternatively, in 'call by reference' function call, a pointer or reference

to the variable is passed as parameter. Thus, any change made to the variables gets reflected at the original location of the variable.

8. Write the syntax for pointer to Structures. (AU, Cbe, Dec 10-Jan 11)

Ans: Syntax of Pointer to Structure

```
struct S
{
 char datatype1;
 int datatype2;
 float datatype3;
};

struct S *SPTR; //SPTR is pointer to structure S
```

9. Differentiate Structure and Union. (AU, Cbe, Dec 10-Jan 11)

Ans:

Difference between Structure and Union

Structure	Union
A structure is defined with 'struct' keyword.	A union is defined with 'union' keyword.
All members of a structure can be manipulated simultaneously.	The members of a union can be manipulated only one at a time.
The size of a structure object is equal to the sum of the individual sizes of the member objects.	The size of a union object is equal to the size of largest member object.
Structure members are allocated distinct memory locations.	Union members share common memory space for their exclusive usage.
Structures are not considered as memory efficient in comparison to unions.	Unions are considered as memory efficient particularly in situations when the members are not required to be accessed simultaneously.

10. Explain two dimensional array. (AU, Cbe, Dec 10-Jan 11)

Ans: Refer Section 5.2.2.

11. What are pointers? (AU, IRI, Jan 2011)

Ans: Refer Section 5.10.

12. State the advantage of user defined functions over pre-defined functions. (AU, IRI, Jan 2011)

Ans: The various advantages of user-defined functions over pre-defined functions are:

- A user-defined function allows the programmer to define the exact functionality of the module as per requirement. However, this may not be the case with pre-defined function; it may or may not serve the desired purpose completely.

- A user-defined function gives flexibility to the programmer to use optimal (most efficient) programming instructions. However, this may not be the case with pre-defined functions.

DESCRIPTIVE QUESTIONS

1. Write a C program to reverse a given string. (AU, Chn, Jan 2010)

Ans:

```
Program

#include <stdio.h>
#include <conio.h>
#include <string.h>

void main()
{
char str[30],revstr[30];
int i,len;
printf("\nEnter a string: ");
scanf("%s",&str);
len=strlen(str);
for(i=0;i<len;i++)
revstr[len-i-1]=str[i];
revstr[len]='\0';
printf("\n\nThe reverse of string %s is %s",str,revstr);
getch();
}
```

2. Differentiate pass by value and pass by address in C. (AU, Chn, Jan 2010)

Ans: Refer Section 5.6.10.

3. Write a C program that gets and displays the report of n students with their personal and academic details using structures. (AU, Chn, Jan 2010)

Ans:

```
Program
#include <stdio.h>
#include <conio.h>
void main ()
{
int num, i=0;
struct student
{
char name[30];
char fname[30];
char course[30];
```

```
int rollno;
int t_marks;
};
struct student std[10];
clrscr();
printf("Enter the number of students: ");
scanf("%d",&num);

for(i=0;i<num;i++)
{
printf("\nEnter the details for %d student",i+1);
printf("\n\n Name ");
scanf("%s",std[i].name);
printf("\n Father's Name ");
scanf("%s",std[i].fname);
printf("\n Course ");
scanf("%s",std[i].course);
printf("\n Roll No. ");
scanf("%d",&std[i].rollno);
printf("\n Total Marks ");
scanf("%d",&std[i].t_marks);
}
printf("\n Press any key to display the student details!");
getch();
for(i=0;i<num;i++)
printf("\nstudent %d \n Name %s \n Father's Name %s \n Course %s \n Roll No. %d
\n Total Marks %d\n",i+1,std[i].name, std[i].fname, std[i].course, std[i].rollno,
std[i].t_marks);
getch();
}
```

4. Explain the function
   (i) strlen()
   (ii) strcpy()
   (iii) strcat()
   (iv) strcmp() with example. (AU, Chn, Jan 2011)

   Ans: Refer Section 5.4.4
5. Describe with example:
   (i) Declaring a structure
   (ii) Pointer to multidimensional array
   (iii) Union. (AU, Chn, Jan 2011)

Ans:

(i) Declaring a Structure: Refer Section 5.8.2.

(ii) Pointer to Multidimensional Array: Refer Section 5.10.9.

(iii) Union: Refer Section 5.8.11.

6. What is an array ? How to declare a two dimensional array? (AU, Mdu, Jan 2011)

Ans:

Array: Refer Section 5.2.

Declaration of two dimensional array: Refer Section 5.2.2.

7. Write a C program to display numbers in ascending orders. (AU, Mdu, Jan 2011)

Ans:

Program to Display Numbers in Ascending Order

```
#include <stdio.h>
#include <conio.h>

void main()
{
  void sort(int []);
  int num[10],i;
     printf("Enter the ten elements to sort:\n");

  for (i=0;i<10;i++)
    scanf("%d",&num[i]);
    sort(num);

  printf("\nThe sorted elements are:\n");
  for(i=0;i<10;i++)
    printf("%d\n",num[i]);
    getch();
}

void sort(int num[])
{
 int i,j,temp;
 for(j=1;j<10;j++)
  {
    temp=num[j];
     for(i=j-1;i>=0 && temp<num[i];i–)
       num[i+1]=num[i];
     num[i+1]=temp;
   }
}
```

Output

```
1 Output
 Enter the ten elements to sort:
 22
 33
 1
 2
 65
 18
 7
 54
 78
 5

 The sorted elements are:
 1
 2
 5
 7
 18
 22
 33
 54
 65
 78
```

8. What are the various types of functions in C ? Explain with examples. (AU, Mdu, Jan 2011)
   Ans: Refer Section 5.6.4.
9. Write short notes on following.
   (a) Features of structure, (b) Union. (AU, Mdu, Jan 2011)
   Ans:
   (a) Features of Structure: The various features of structures are:
   - It allows grouping together of dissimilar type elements.
   - It helps in realizing real-life data objects.
   - It gives flexibility to the programmers to define their own data types as per requirement.
   - It helps in building complex data types.
   - With the help of pointers, structures help in realizing various data structures.

   (b) Union: Refer Section 5.8.11.

10. Write a C Program to Count numbers of 0's , 1's and blank spaces and other character?

(AU, Cbe, Dec 10-Jan 11)

Ans: Program

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

void main()
{
 char str[100];
 int len,i,flag;
 int count0,count1,countw,counto;
 count0=count1=countw=counto=0;
 flag=0;

 gets(str);

 len=strlen(str);
 for(i=0;i<len;i++)
 {
  if(str[i]=='0')
   count0++;
  else if(str[i]=='1')
   count1++;
       else if(str[i]==' ')
             countw++;
       else
        counto++;
 }

 printf("Number of 0's = %d\n",count0);
 printf("Number of 1's = %d\n",count1);
 printf("Number of white spaces = %d\n",countw);
 printf("Number of Other characters = %d\n",counto);

 getch();
}
```

Output

```
This is a test string for counting 0000s and 111s
Number of 0's = 4
Number of 1's = 3
Number of white spaces = 9
Number of Other characters = 33
```

11. Write a C Program to read and write employee and their data of joining using nested structure. (AU, Cbe, Dec 10-Jan 11)

Ans: Program

```
#include <stdio.h>
#include <conio.h>

void main()
{
 struct employee
 {
  char name[20];
  struct doj
  {
   int day;
   char month[9];
   int year;
  }date;
 }emp;

 printf("Enter the name of the employee: ");
 gets(emp.name);

 printf("Enter the date of joining:\n");
 printf("Day: ");
 scanf("%d",&emp.date.day);

 fflush(stdin);
 printf("Month: ");
 gets(emp.date.month);

printf("Year: ");
 scanf("%d",&emp.date.year);

 printf("\n\nEmployee Name: %s",emp.name);
 printf("\nEmployee date of joining: %d %s %d",emp.date.day,emp.date.month,emp.date.year);

 getch();
}
```

Output

```
Enter the name of the employee: James
Enter the date of joining:
```

```
Day: 22
Month: Nov
Year: 2010

Employee Name: James
Employee date of joining: 22 Nov 2010
```

12. Write a C program that performs matrix addition. (AU, Cbe, Dec 10-Jan 11)

Ans: Program

```
#include <stdio.h>
#include <conio.h>

void main()
{
int i,j,a[3][3],b[3][3],c[3][3];


printf("Enter the first 3 X 3 matrix:\n");
for(i=0;i<3;i++)
{
  for(j=0;j<3;j++)
  {
  printf("a[%d][%d] = ",i,j);
  scanf("%d",&a[i][j]);
  }
}
printf("Enter the second 3 X 3 matrix:\n");
for(i=0;i<3;i++)
{
  for(j=0;j<3;j++)
  {
  printf("b[%d][%d] = ",i,j);
  scanf("%d",&b[i][j]);
  }
}
printf("\nThe entered matrices are: \n");

for(i=0;i<3;i++)
{
  printf("\n");
  for(j=0;j<3;j++)
{
```

```
    printf("%d\t",a[i][j]);
      }
      printf("\t\t");
      for(j=0;j<3;j++)
      {
      printf("%d\t",b[i][j]);
      }
    }
    for(i=0;i<3;i++)
    {
      for(j=0;j<3;j++)
     c[i][j] =a[i][j]+b[i][j];
    }
    printf("\n\nThe sum of the two matrices is shown below: \n");

    for(i=0;i<3;i++)
    {
      printf("\n\t\t    ");
      for(j=0;j<3;j++)
      {
      printf("%d\t",c[i][j]);
      }
    }
    getch();
    }
```

Output

```
Enter the first 3 X 3 matrix:
a[0][0] = 1
a[0][1] = 2
a[0][2] = 3
a[1][0] = 1
a[1][1] = 1
a[1][2] = 1
a[2][0] = 3
a[2][1] = 2
a[2][2] = 1
Enter the second 3 X 3 matrix:
b[0][0] = 1
b[0][1] = 1
b[0][2] = 1
b[1][0] = 1
```

```
b[1][1] = 1
b[1][2] = 1
b[2][0] = 1
b[2][1] = 1
b[2][2] = 1

The entered matrices are:

1       2       3                       1       1       1
1       1       1                       1       1       1
3       2       1                       1       1       1

The sum of the two matrices is shown below:

                    2   3       4
                    2   2       2
                    4   3       2
```

13. Explain the role of C processor and describe file inclusive directive. (AU, Cbe, Dec 10-Jan 11)

Ans:

C Preprocessor

The function of a preprocessor is to process the source code before it passes through the compiler. It operates under the control of preprocessor command lines or directives. Preprocessor directives are placed in the source program before the main block. When the source code passes through the compiler, it is examined by the preprocessor for any preprocessor directives. If they are present, appropriate actions (as per the directives) are taken and then the source program is handed over to the compiler.

File Inclusion Directive

The file inclusion directive is used to include another file in a program. It can include the file using any of the following two ways:

```
#include <filename>
#include "filename"
```

If the name of the file to be included is specified within <> symbol, then the compiler will search the file in the directory specified as the include directory. So, the standard header files are generally specified within angular quotes. If the name of the file to be included is specified within " " symbol, then the compiler searches the file in all the directories in the current drive.

The following are some of the examples of file inclusion directives:

```
#include <string.h>
#include "codefile_a22"
```

14. How are pointer declared and initialized? How is the Value of variable accessed using pointer? (AU, Cbe, Dec 10-Jan 11)

Ans:

Pointer Declaration and Initialization: Refer Section 5.10.3 and 5.10.4.

Accessing Variable value using Pointer: Refer Section 5.10.5.

15. What is an array? How do declare and initialize a two-dimensional array in C? (AU, TIR, Dec 10-Jan 11)

Ans:

Array: Refer Section 5.2.

Declaring and Initializing Two-dimensional Array: Refer Section 5.2.2.

16. Explain the parameter passing mechanisms in C in detail. (AU, TIR, Dec 10-Jan 11)

Ans: Refer Sections 5.6.10 and 5.10.12.

17. What is user-defined function? Write a function to find the factorial of a number. (AU, TIR, Dec 10-Jan 11)

Ans:

User-defined Function: Refer Section 5.6.

Function to Find Factorial of a Number: Refer Example 5.12.

18. Explain the following concepts with an example. (AU, TRI, Jan 2011)

(i) Call by reference; (ii) Call by value.

Ans:

(i) Call by reference: In this method, the memory addresses of the variables rather than the copies of values are passed to the function. Thus, any changes made to the values stored at the memory addresses stay permanent and get reflected back to the main function. The following code snippet depicts this scenario:

Code

```
void change(int *n)
{
   *n=100;
}
void main()
{
 int num=50;
 cout<<"Before function call, num = "<<num;
 change(&num);
 cout<<"\nAfter function call, num = "<<num;
getch();
}
```

Output

```
Before function call, num = 50
After function call, num = 100
```

(ii) Call by value: In this method, the value of a variable is passed to a function. The passed value is copied into another variable in the parameter list of the function. Thus, any change made to the new variable's value inside the function is not reflected back in the main function. The following code snippet depicts this scenario:

Code

```
void change(int n)
{
   n=100;
}
void main()
{
 int num=50;
 cout<<"Before function call, num = "<<num;
 change(num);
 cout<<"\nAfter function call, num = "<<num;
getch();
}
```

Output

```
Before function call, num = 50
After function call, num = 50
```

Appendix A
SAMPLE PROGRAMS

APPENDIX OUTLINE

A.1 Ask the user to enter 10 integer numbers on the I/O terminal and then compute and display their average.

A.2 Write a program to accept a square matrix from the user and then display its transpose.

A.3 Write a program to enter a date in the dd\ mm\yyyy format and determine whether the entered date is correct or not.

A.4 Write a program to calculate the sum of the series $1^4 + 3^4 + 5^4$……up to 100 terms.

A.5 Write a program to calculate the sum of two numbers which are passed as arguments using the call by value method.

A.6 Write a program to calculate the sum of two numbers which are passed as arguments using the call by reference method.

A.7 Write a program to check whether a given string is a palindrome or not.

A.8 Write a program to multiply the elements of two ***N*** X ***N*** matrices.

A.9 Write a program to create a student structure and then list all the students who scored more than 75 marks.

A.10 Write a program that searches the integer value entered by the user in an array of 20 elements.

A.11 Write a sample program to illustrate the use of break statement in C.

A.12 Write a program to calculate the sum of all the numbers between 3 and 20 excluding the multiples of 3.

A.13 Write a program to find out all the prime numbers between 1 and n (where n is entered by the user).

A.14 Write a program to find out the sum of two complex numbers.

A.15 Write a program to find the final velocity of a moving object using the following equation:

v=u+ a*t

A.16 Write a program to find out whether the given number is even or odd.

A.17 Write a program to determine whether the year entered by the user is a leap year or not.

A.1 Ask the user to enter 10 integer numbers on the I/O terminal and then compute and display their average.

```
Program
#include<stdio.h>
#include<conio.h>
void main()
{
int arr[10],sum=0,i;
float avg;
clrscr();
for(i=0;i<10;i++)
{
printf("\nEnter the value of %d element = ",i);
scanf("%d",&arr[i]);
}
for(i=0;i<10;i++)
{
sum=sum+arr[i];
}
avg=sum/10;
printf("\n\nAverage of 10 numbers is equal to %2.2f",avg);
getch();
}
Output
Enter the value of 0 element = 2
Enter the value of 1 element = 3
Enter the value of 2 element = 4
Enter the value of 3 element = 5
Enter the value of 4 element = 6
Enter the value of 5 element = 7
Enter the value of 6 element = 8
Enter the value of 7 element = 33
Enter the value of 8 element = 44
Enter the value of 9 element = 45
Average of 10 numbers is equal to 15.00
```

Fig A.1 Program to calculate average of ten numbers

A.2 Write a program to accept a square matrix from the user and then display its transpose.

```
Program
#include<stdio.h>
#include<conio.h>
```

```
void main()
{
int matrix1[10][10],i,j,a,b;
clrscr();
printf("Enter the size of the square matrix\n");
scanf ("%d", &a);
b=a;
printf("You have to enter the matrix elements in row-wise fashion\n");
for(i=0;i<a;i++)
{
for(j=0;j<b;j++)
{
printf("\nEnter the next element in the matrix=");
scanf("%d",&matrix1[i][j]);
}
}
printf("\n\nEntered matrix is\n");
for(i=0;i<a;i++)
{           printf("\n");
for(j=0;j<b;j++)
printf(" %d ",matrix1[i][j]);
}
printf("\n\nTranspose of the entered matrix is\n");
for(i=0;i<a;i++)
{           printf("\n");
for(j=0;j<b;j++)
printf(" %d ",matrix1[j][i]);
}
getch();
}
Output
Enter the size of the square matrix
3
You have to enter the matrix elements in row-wise fashion
Enter the next element in the matrix=11
Enter the next element in the matrix=22
Enter the next element in the matrix=33
Enter the next element in the matrix=44
Enter the next element in the matrix=55
Enter the next element in the matrix=66
Enter the next element in the matrix=77
Enter the next element in the matrix=88
Enter the next element in the matrix=99
```

```
Entered matrix is
 11  22  33
 44  55  66
 77  88  99
Transpose of the entered matrix is
 11  44  77
 22  55  88
 33  66  99
```

Fig A.2 **Program for displaying the transpose of an N × N matrix**

A.3 Write a program to enter a date in the dd\mm\yyyy format and determine whether the entered date is correct or not.

```
Program
#include <stdio.h>
#include <conio.h>
void main()
{
 int dd,mm,yyyy; /*Initialization of variables*/
 clrscr(); /*Clearing the I/O console of any previous data*/
printf("Enter the date in dd\mm\yyyy format "); /*Prompting the user to enter a date*/
 scanf("%d %d %d",&dd,&mm,&yyyy); /*Reading the date values entered by the user and storing them
in the declared variables*/
 if(dd<1 || dd>31 || mm<1 || mm>12 || yyyy<0 || (mm==2 && dd>29))
/*Checking whether day, month, and year is correct */
 printf("NO"); /*Displaying the "NO" message if the entered date is not correct*/
 else
 printf("YES"); /*Displaying the "YES" message if the entered date is not correct*/
 getch();
} /*End of Program*/
Output
Enter the date in dd\mm\yyyy format 26
05
1982
YES
Enter the date in dd\mm\yyyy format 26
26
1982
NO
```

Fig A.3 **Program to check the validity of the entered date**

A.4 Write a program to calculate the sum of the series 14 + 34 +54……up to 100 terms.

```
Program
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
int i;
long sum=0;
clrscr();
for(i=1;i<=100;i=i+2)
{
sum=sum+pow(i,4);
}
printf("\n Sum of the series up to first 100 terms is equal to %ld",sum);
getch();
}
Output
Sum of the series up to first 100 terms is equal to 999666690
```

Fig A.4 **Program to calculate the sum of the series 14 + 34 +54……up to 100 terms**

A.5 Write a program to calculate the sum of two numbers which are passed as arguments using the call by value method.

```
Program
#include<stdio.h>
#include<conio.h>
int add(int a, int b);
main()
{
int z,x = 10;
int y = 20;
clrscr();
z=add(x,y);
printf("\n Sum of numbers %d and %d is equal to %d",x,y,z);
getch();
}
//Values of x and y are transferred using call by value
int add(int a, int b)
{
int c;
c= a+b;
```

```
return(c);
}
Output
Sum of numbers 10 and 20 is equal to 30
```

Fig. A.5 **Program to calculate sum of two numbers using the call by value method**

A.6 Write a program to calculate the sum of two numbers which are passed as arguments using the call by reference method.

```
Program
#include<stdio.h>
#include<conio.h>
void swap(int *p, int *q);
main()
{
int x=10;
int y=20;
clrscr();
printf("\nValue of X and Y before swapping are X=%d and Y=%d",x,y);
swap(&x,&y);
printf("\n\nValue of X and Y after swapping are X=%d and Y=%d",x,y);
getch();
}
void swap(int *p, int *q)//Values of x and y are transferred using call by reference
{
int r;
r=*p;
*p=*q;
*q=r;
}
Output
Value of X and Y before swapping are X=10 and Y=20
Value of X and Y after swapping are X=20 and Y=10
```

Fig. A.6 **Program to pass the arguments using call by reference method**

A.7 Write a program to check whether a given string is a palindrome or not.

```
Program
#include<stdio.h>
#include<conio.h>
#include <string.h>
void main()
{
```

```
char a[50];
int len,i,flag=0;
clrscr();
printf("Enter a string ");
scanf ("%s", &a);
len=strlen(a);
for(i=0;i<len/2;i++)
{
if(a[i]==a[len-i-1])
continue;
else
{
flag=1;
break;
}
}
if(flag==0)
printf("\n%s is a palindrome string",a);
else
printf("\n%s is not a palindrome string",a);
getch();
}
Output
Enter a string gaurav
gaurav is not a palindrome string
Enter a string rotor
rotor is a palindrome string
```

Fig. A.7 Program to check whether a given string is a palindrome or not

A.8 Write a program to multiply the elements of two N×N matrices.

```
Program
#include<stdio.h>
#include<conio.h>
void main()
{
int a1[10][10],a2[10][10],c[10][10],i,j,k,a,b;
clrscr();
printf("Enter the size of the square matrix\n");
scanf ("%d", &a);
b=a;
printf("You have to enter the matrix elements in row-wise fashion\n");
for(i=0;i<a;i++)
```

```
{
for(j=0;j<b;j++)
{
printf("\nEnter the next element in the 1st matrix=");
scanf("%d",&a1[i][j]);
}
}
for(i=0;i<a;i++)
{
for(j=0;j<b;j++)
{
printf("\n\nEnter the next element in the 2nd matrix=");
scanf("%d",&a2[i][j]);
}
}
printf("\n\nEntered matrices are\n");
for(i=0;i<a;i++)
{           printf("\n");
for(j=0;j<b;j++)
printf(" %d ",a1[i][j]);
}
printf("\n");
for(i=0;i<a;i++)
{           printf("\n");
for(j=0;j<b;j++)
printf(" %d ",a2[i][j]);
}
printf("\n\nProduct of the two matrices is\n");
for(i=0;i<a;i++)
 for(j=0;j<b;j++)
  {
  c[i][j]=0;
  for(k=0;k<a;k++)
  c[i][j]=c[i][j]+a1[i][k]*a2[k][j];
  }
for(i=0;i<a;i++)
{           printf("\n");
for(j=0;j<b;j++)
printf(" %d ",c[i][j]);
}
getch();
}
```

```
Output
Enter the size of the square matrix
2
You have to enter the matrix elements in row-wise fashion
Enter the next element in the 1st matrix=1
Enter the next element in the 1st matrix=0
Enter the next element in the 1st matrix=2
Enter the next element in the 1st matrix=3
Enter the next element in the 2nd matrix=4
Enter the next element in the 2nd matrix=5
Enter the next element in the 2nd matrix=0
Enter the next element in the 2nd matrix=2
Entered matrices are
 1  0
 2  3
 4  5
 0  2
Product of the two matrices is
 4  5
 8  16
```

Fig. A.8 Program for N × N matrix multiplication

A.9 Write a program to create a student structure and then list all the students who scored more than 75 marks.

```
Program
#include <stdio.h>
#include <conio.h>
void main()
{
 struct student
 {
  int rollno;
  char name[50];
  char fname[50];
  int age;
  char city[50];
  int marks;
  };
  struct student students[20];
  int nos,i,flag=0;
   clrscr();
  printf("Enter the number of students whose records you want to enter ");
```

```
  scanf("%d",&nos);
  for(i=0;i<nos;i++)
  {
   printf("\nEnter the details of the %d student", i+1);
   printf("\nRoll no ");
   scanf("%d",&students[i].rollno);
   printf("\nName ");
   scanf("%s",&students[i].name);
   printf("\nFather's Name ");
   scanf("%s",&students[i].fname);
   printf("\nAge ");
   scanf("%d",&students[i].age);
   printf("\nCity ");
   scanf("%s",&students[i].city);
   printf("\nMarks ");
   scanf("%d",&students[i].marks);
  }
  printf("\nThe name(s) of the student(s) with more than 75 marks is/are: ");
  for(i=0;i<nos;i++)
  {
   if(students[i].marks>75)
   {
    printf("\n%s",students[i].name);
    flag=1;
   }
  }
  if(flag==0)
  printf("\nNone!");
   getch();
}
Output
Enter the number of students whose records you want to enter 3
Enter the details of the 1 student
Roll no 99011
Name Ram
Father's Name Ghansham
Age 23
City Delhi
Marks 74
Enter the details of the 2 student
Roll no 99012
Name Nitin
Father's Name Mukesh
Age 24
```

```
City Noida
Marks 75
Enter the details of the 3 student
Roll no 99013
Name Ramesh
Father's Name Suresh
Age 23
City Gurgaon
Marks 76
The name(s) of the student(s) with more than 75 marks is/are:
Ramesh
```

Fig A.9 **Program to list the students who scored more than 75 marks**

A.10 Write a program that searches the integer value entered by the user in an array of 20 elements.

```
Program
#include <stdio.h>
# include <conio.h>

void main()
{
 int a[20],i,search_item;
clrscr();
for(i=0;i<20;i++)
 {
 printf("\nEnter the %d element of the array ",i);
 scanf("%d",&a[i]);
 }
 printf("\nEnter the element to be searched ");
 scanf("%d",&search_item);
 for(i=0;i<20;i++)
 {
  if(a[i]==search_item)
  {
   printf("\n%d is present in the list",search_item);
   break;
  }
 }
 if(i==20)
 printf("\n%d is not in the list",search_item);
 getch();
 }
```

```
Output
Enter the 0 element of the array 11
Enter the 1 element of the array 22
Enter the 2 element of the array 33
Enter the 3 element of the array 44
Enter the 4 element of the array 55
Enter the 5 element of the array 66
Enter the 6 element of the array 77
Enter the 7 element of the array 88
Enter the 8 element of the array 99
Enter the 9 element of the array 111
Enter the 10 element of the array 222
Enter the 11 element of the array 333
Enter the 12 element of the array 444
Enter the 13 element of the array 555
Enter the 14 element of the array 666
Enter the 15 element of the array 777
Enter the 16 element of the array 888
Enter the 17 element of the array 999
Enter the 18 element of the array 1
Enter the 19 element of the array 2
Enter the element to be searched 999
999 is present in the list
```

Fig. A.10 **Program to search a given integer in an array of 20 elements**

A.11 Write a sample program to illustrate the use of break statement in C.

```
Program
#include<stdio.h>
#include<conio.h>
void main()
{
  int i, sum=0;
  clrscr();
  for(i=0;;i++)
  {
  if(i>5)
     break;
  sum=sum+i;
  }
  printf("\n Sum of first 5 numbers is equal to %d", sum);
  getch();
}
Output
Sum of first 5 numbers is equal to 15
```

Fig A.11 **Using break statement in C**

A.12 Write a program to calculate the sum of all the numbers between 3 and 20 excluding the multiples of 3.

```
Program
#include<stdio.h>
#include<conio.h>
void main()
{
  int i,sum=0;
  clrscr();
  for (i=3;i<20;i++)
  {
 if(i%3==0)
continue;
sum=sum+i;
}
printf("\n Sum of the numbers between 3 to 20 excluding multiples of 3 is equal to %d", sum);
getch();
}
Output
Sum of the numbers between 3 to 20 excluding multiples of 3 is equal to 124
```

Fig A.12 Program to calculate sum of the numbers between 3 to 20 excluding multiples of 3

A.13 Write a program to find out all the prime numbers between 1 and n (where n is entered by the user).

```
Program
#include <stdio.h>
#include <conio.h>
#include <math.h>
void main()
{
      int n,i,j;
      clrscr();
      printf("Enter a number upto which you want prime numbers:");
      scanf("%d",&n);
      if(n<=1)
      {
         printf("Enter a number greater than 1.");
         getch();
         exit(0);
      }
      printf("Prime numbers between 1 and %d are:",n);
      printf("\n2");
```

```
      for(i=3;i<=n;i++)
      {
         for(j=2;j<=sqrt(i);j++)
         {
            if(i%j==0)
               break;
         }
         if(j>sqrt(i))
            printf("\n%d",i);
      }
      getch();
}
Output
Enter a number upto which you want prime numbers:
5
Prime numbers between 1 and 5 are:
2
3
5
```

Fig. A.13 **Program to find prime numbers upto a given number**

A.14 Write a program to find out the sum of two complex numbers.

```
Program
#include<stdio.h>
#include<math.h>
struct comp
{
     double real;
     double imagi;
};
void main()
{
struct comp num1, num2, sum;
clrscr();
     printf("\n Input 2 complex numbers for addition ");
     printf("\n Real part of 1st No:");
     scanf("%lf",&num1.real);
     printf("\n Imaginary part of 1st No:");
     scanf("%lf",&num1.imagi);
     printf("\n Real part of 2nd No:");
     scanf("%lf",&num2.real);
     printf("\n Imaginary part of 2nd No :");
```

```
    scanf("%lf",&num2.imagi);
  sum.real = num1.real+num2.real;
       sum.imagi = num1.imagi+num2.imagi;
  printf("\n Sum of two complex numbers is = %3.1lf +%3.1lfi",sum.real,sum.imagi);
 getch();
}
Output
 Input 2 complex numbers for addition
 Real part of 1st No:3
 Imaginary part of 1st No:5
 Real part of 2nd No:1.5
 Imaginary part of 2nd No :11
 Sum of two complex numbers is = 4.5 +16.0i
```

Fig. A.14 **Program to find the sum of two complex numbers**

A.15 Write a program to find the final velocity of a moving object using the following equation:

$$v = u + a^*t$$

```
Program
#include <stdio.h>
#include <math.h>
void main()
{
float acceleration, time, final_velo, initial_velo;
clrscr();
printf("\n\nEnter the values for calculating final velocity of a moving object");
printf("\nEnter the initial velocity in metre per second: ");
scanf("%f",&initial_velo);
printf("\nEnter the acceleration in metre per second square: ");
scanf("%f",&acceleration);
printf("\nEnter the time interval in second : ");
scanf("%f",&time);
final_velo=initial_velo+ acceleration*time;
printf("\nThe final velocity of the moving object is = %4.2f", final_velo);
getch();
}
Output
Enter the values for calculating final velocity of a moving object
Enter the initial velocity in metre per second: 5
Enter the acceleration in metre per second square: 6
Enter the time interval in second : 20
The final velocity of the moving object is = 125.00
```

Fig. A.15 **Solving the equation, v=u + at**

A.16 Write a program to find out whether the given number is even or odd.

```
Program
#include<stdio.h>
#include<conio.h>
void main( )
{
int num;
clrscr();
printf("Enter the number");
scanf("%d",&num);
if (num%2==0) /* remainder after division by 2*/
     printf("\nThe number is even");
else
     printf("\nThe number is odd");
getch();
}
Output
Enter the number
4
The number is even
```

Fig. A.16 To identify even and odd numbers

A.17 Write a program to determine whether the year entered by the user is a leap year or not.

```
Program
#include<stdio.h>
#include<conio.h>
void main()
{
int yr;
clrscr();
printf("Enter the year");
scanf("%d",&yr);
if(yr%100==0)
{
     if(yr%400==0)
          printf("\n Year is leap year");
     else
          printf("\n Year is not leap year");
}
else
{
```

```
     if(yr%4==0)
          printf("\n Leap year");
     else
          printf("\n Not a leap year");
}
getch();
}
Output
Enter the year
2004
Year is leap year
```

Fig. A.17 **To identify even and odd numbers**

APPENDIX B

UNIT III: TOP DOWN DESIGN, TYPE QUALIFIER, USING LOOPS FOR SEUENCE ACCESS.

B.1 TOP-DOWN DESIGN

The approach to systematic and managed software development involves breaking up the entire problem into a set of small focus areas with each serving a definite purpose. The software solutions (modules) pertaining to each of these focus areas are then interlinked together to arrive at one common solution that serves a large set of requirements.

Top-down software design is one such approach to software design and implementation that starts with the primary module at the top (root) and moves in a top-down fashion breaking the primary module into a series of sub-modules, each catering to a specific set of system requirements. The sub-modules are created in such a fashion that they represent some real-world action-oriented task within the application scope. Further, the size of the sub-modules is kept small and thus manageable from both development and implementation perspectives. The fully functional modules, well interacting with each other serves the overall application objectives.

While working in C, we can relate the concept of modules with that of subroutines or functions. The various functions in a program call or interact with each other to serve the overall program objectives.

Figure B.1 shows a top-down hierarchy chart for a payroll application.

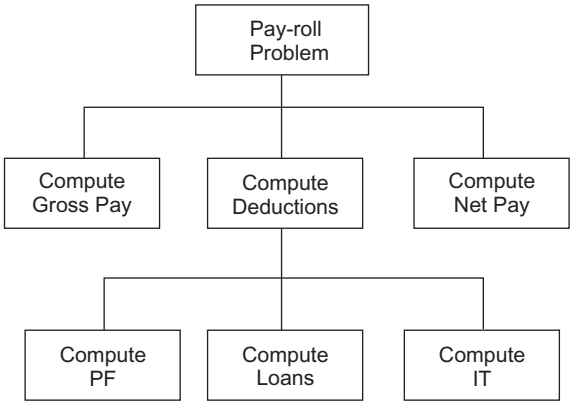


Fig. B.1 **Hierarchy chart for pay-roll problem**

B.1.1 Structure Charts

The best way to create a top-down design for an application is through structure chart. It is a diagrammatic representation consisting of rectangular boxes to represent application modules. It also contains arrows to signify inter module communication. The hierarchically arranged modules and their interfaces present a complete picture of the overall application design.

Table A.1 shows some of the standard symbols used while creating structure charts:

Table B.1 Structure Chart Symbols

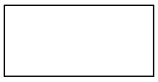
Symbol	Description
	Represents a module. It contains the name of the module
	Represents a function call. The arrow points from the calling module to the called module.
	Represents passing of a data field from one module to another
	Represents passing of a control field (flag) from one module to another

Figure A.2 shows a sample structure chart:

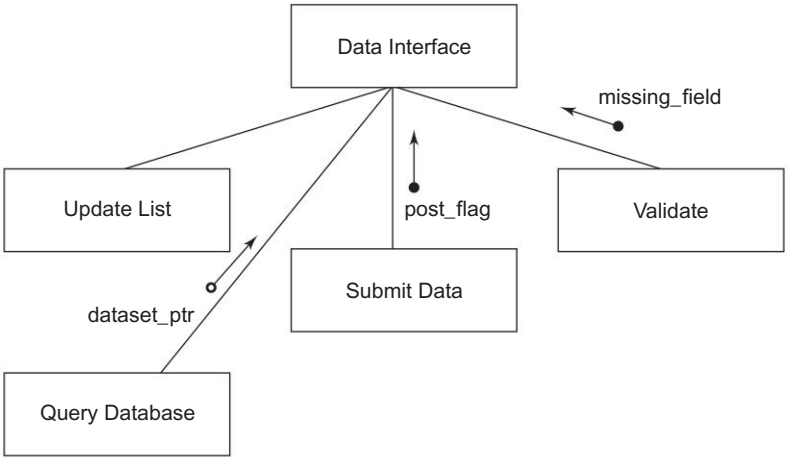


Fig. B.2 **A sample structure chart**

B.2 DYNAMIC MEMORY ALLOCATION

C language requires the number of elements in an array to be specified at compile time. But we may not be able to do so always. Our initial judgement of size, if it is wrong, may cause failure of the program or wastage of memory space.

Many languages permit a programmer to specify an array's size at run time. Such languages have the ability to calculate and assign, during execution, the memory space required by the variables in a program.

The process of allocating memory at run time is known as *dynamic memory allocation*. Although C does not inherently have this facility, there are four library routines known as "memory management functions" that can be used for allocating and freeing memory during program execution. They are listed in Table B.2. These functions help us build complex application programs that use the available memory intelligently.

Table B.1 Structure Chart Symbols

Function	Task
malloc	Allocates request size of bytes and returns a pointer to the first byte of the allocated space.
calloc	Allocates space for an array of elements, initializes them to zero and then returns a pointer to the memory.
free	Frees previously allocated space.
realloc	Modifies the size of previously allocated space.

Memory Allocation Process

Before we discuss these functions, let us look at the memory allocation process associated with a C program. Figure B.3 shows the conceptual view of storage of a C program in memory.

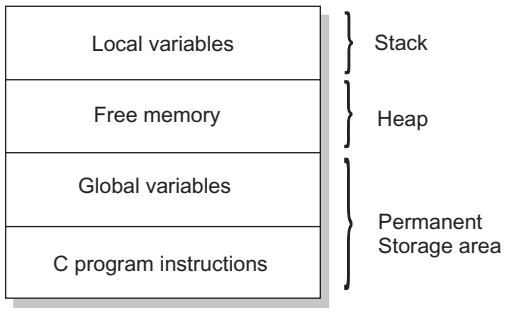


Fig. B.3 Storage of C program

The program instructions and global and static variables are stored in a region known as *permanent storage area* and the local variables are stored in another area called *stack*. The memory space that is located between these two regions is available for dynamic allocation during execution of the program. This free memory region is called the *heap*. The size of the heap keeps changing when program is executed due to creation and death of variables that are local to functions and blocks. Therefore, it is possible to encounter memory "overflow" during dynamic allocation process. In such situations, the memory allocation functions mentioned above return a NULL pointer (when they fail to locate enough memory requested).

B.3 ALLOCATING A BLOCK OF MEMORY: MALLOC

A block of memory may be allocated using the function malloc. The malloc function reserves a block of memory of specified size and returns a pointer of type void. This means that we can assign it to any type of pointer. It takes the following form:

```
ptr = (cast-type *) malloc (byte-size);
```

ptr is a pointer of type *cast-type*. The **malloc** returns a pointer (of *cast-type*) to an area of memory with size *byte-size*.

Example

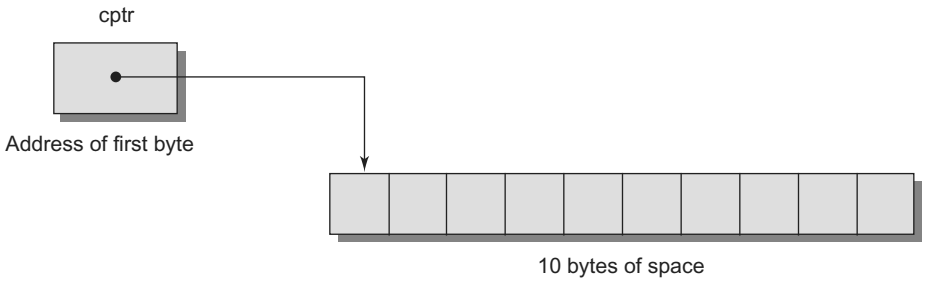
```
x = (int *) malloc (100 *sizeof(int));
```

On successful execution of this statement, a memory space equivalent to "100 times the size of an **int**" bytes is reserved and the address of the fist byte of the memory allocated is assigned to the pointer **x** of type of **int**.

Similarly, the statement

```
cptr = (char*) malloc(10);
```

allocates 10 bytes of space for the pointer cptr of type char. This is illustrated as:



Note that the storage space allocated dynamically has no name and therefore its contents can be accessed only through a pointer.

We may also use **malloc** to allocate space for complex data types such as structures.

Example

```
st-var = (struct store *)malloc(sizeof(struct store));
```

where, **st_var** is a pointer of type **struct store**

Remember, the **malloc** allocates a block of contiguous bytes. The allocation can fail if the space in the heap is not sufficient to satisfy the request. If it fails, it returns a NULL. We should therefore check whether the allocation is successful before using the memory pointer. This is illustrated in the program in Fig. B.3.

EXAMPLE B.1

Write a program that uses a table of integers whose size will be specified interactively at run time.

The program is given in Fig. B.3. It tests for availability of memory space of required size. If it is available, then the required space is allocated and the address of the first byte of the space allocated is displayed. The program also illustrates the use of pointer variable for storing and accessing the table values.

Program

```
     #include <stdio.h>
     #include <stdl i b.h>
     #define NULL 0
main()
{
          int *p, *table;
          int size;
          pr~ntf("\nWhat i s the size of table?");
          scanf("%d",size)
          printf ("\no)
            /*---------Memory allocation ----------------- */
          if((table=(int*)malloc(size*sizeof(int)))==NULL)
            {
              printf("NO space availab1e\n");
              exit(1);
            }
            printf("\n Address of the first byte is %u\n", table);
            /* Reading table values*/
            printf("\nInput table values\nl');

            for (p=table; pctable + size; p++)
              scanf ("%dl,1p);
            /*Printing table values in reverse order*/
            for (p = table + size -1; p >= table; p --)
            printf("%d is stored at address %u \n",*p,p);
}
```

Output

```
What is the size of the table? 5
Address of the first byte is 2262
Input table values t
11 12 13 14 15 15
15 is stored at address 2270
14 is stored at address 2268
13 is stored at address 2266
12 is stored at address 2264
11 is stored at address 2262
```

Fig. B.3 **A sample structure chart**

B.4 ALLOCATING A BLOCK OF MEMORY: MALLOC

calloc is another memory allocation function that is normally used for requesting memory space at run time for storing derived data types such as arrays and structures. While **malloc** allocates a single block of storage space, **calloc** allocates multiple blocks of storage, each of the same size, and then sets all bytes to zero. The general form of **calloc** is:

```
ptr = (cast-type *) calloc (n. elem-size);
```

The above statement allocates contiguous space for n blocks, each of size elem-size bytes. All bytes are initialized to zero and a pointer to the first byte of the allocated region is returned. If there is not enough space, a NULL pointer is returned.

The following segment of a program allocates space for a structure variable:

```
. . . .
. . . .
struct student
{
char name[25] ;
float age;
long int id_num;
};
```

Appendix C
SOLVED PROGRAMMING EXERCISES

EXAMPLE C.1 *Write a program to determine the Greatest Common Divisor (GCD) of two numbers.*

Solution

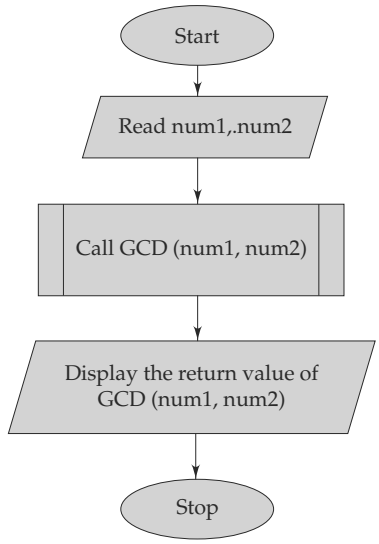
Algorithm

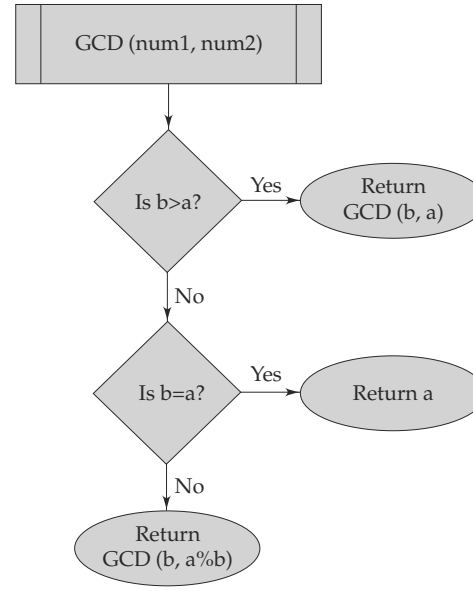
```
Step 1 – Start
Step 2 – Accept the two numbers whose GCD is to be found (num1, num2)
Step 3 – Call function GCD(num1,num2)
Step 4 – Display the value returned by the function call GCD(num1,num2)
Step 5 – Stop

GCD(a,b)
Step 1 – Start
Step 2 – If b > a goto Step 3 else goto Step 4
Step 3 – Return the result of the function call GCD(b,a) to the calling function
Step 4 – If b = 0 goto Step 5 else goto Step 6
Step 5 – Return the value a to the calling function
Step 6 – Return thFCe result of the function call GCD(b,a mod b) to the calling function
```

Flowchart

Program

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

int GCD(int m, int n);
void main()
```






```
{
    int num1,num2;
    clrscr();
    printf("Enter the two numbers whose GCD is to be found: ");
    scanf("%d %d",&num1,&num2);

    printf("\nGCD of %d and %d is %d\n",num1,num2,GCD(num1,num2));
    getch();
}

int GCD(int a, int b)
{
   if(b>a)
      return GCD(b,a);
   if(b==0)
      return a;
   else
      return GCD(b,a%b);
}
```

Output

```
Enter the two numbers whose GCD is to be found: 18 12

GCD of 18 and 12 is 6
```

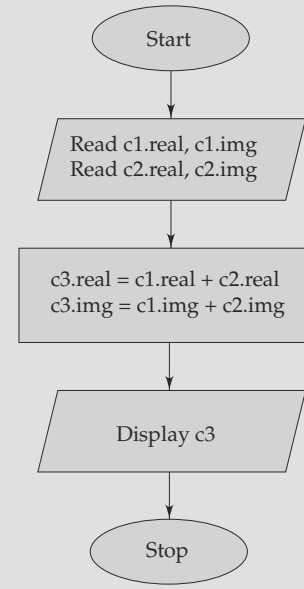
EXAMPLE C.2 *Write a program to accept two complex numbers and find their sum.*

Solution

Algorithm

```
Step 1 – Start
Step 2 – Define a structure to represent a complex number
          STRUCTURE complex
          REAL real
          REAL img
          END STRUCTURE
          STRUCTURE complex c1, c2
Step 3 – Read the real and imaginary parts of the first complex number (c1.real, c1.img)
Step 4 – Read the real and imaginary parts of the second complex number (c2.real, c2.img)
Step 5 – Calculate c3.real=c1.real+c2.real
Step 6 – Calculate c3.img=c1.img+c2.img
Step 7 – Display c3
Step 8 – Stop
```

Flowchart



Program

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

void main()
{
struct complex
{
    double real;
    double img;
};

struct complex c1, c2, c3;
clrscr();
printf("\n Enter two Complex Numbers (x+iy):\n\n Real Part of First Number: ");
scanf("%lf",&c1.real);
printf("\n Imaginary Part of First Number: ");
scanf("%lf",&c1.img);
printf("\n Real Part of Second Number: ");
scanf("%lf",&c2.real);
```

```
printf("\n Imaginary Part of Second Number: ");
scanf("%lf",&c2.img);
c3.real=c1.real+c2.real;
c3.img=c1.img+c2.img;
printf("\n\n%.2lf+(%.2lf)i + %.2lf+(%.2lf)i = %.2lf+(%.2lf)i", c1.real, c1.img, c2.real,
c2.img, c3.real, c3.img);
getch();
}
```

Output

```
Enter two Complex Numbers (x+iy):

Real Part of First Number: 22

Imaginary Part of First Number: 4

Real Part of Second Number: 5

Imaginary Part of Second Number: 3

22.00+(4.00)i + 5.00+(3.00)i = 27.00+(7.00)i
```

EXAMPLE C.3 *Write a program to simulate a simple calculator for performing basic arithmetic operations.*

Solution

Algorithm

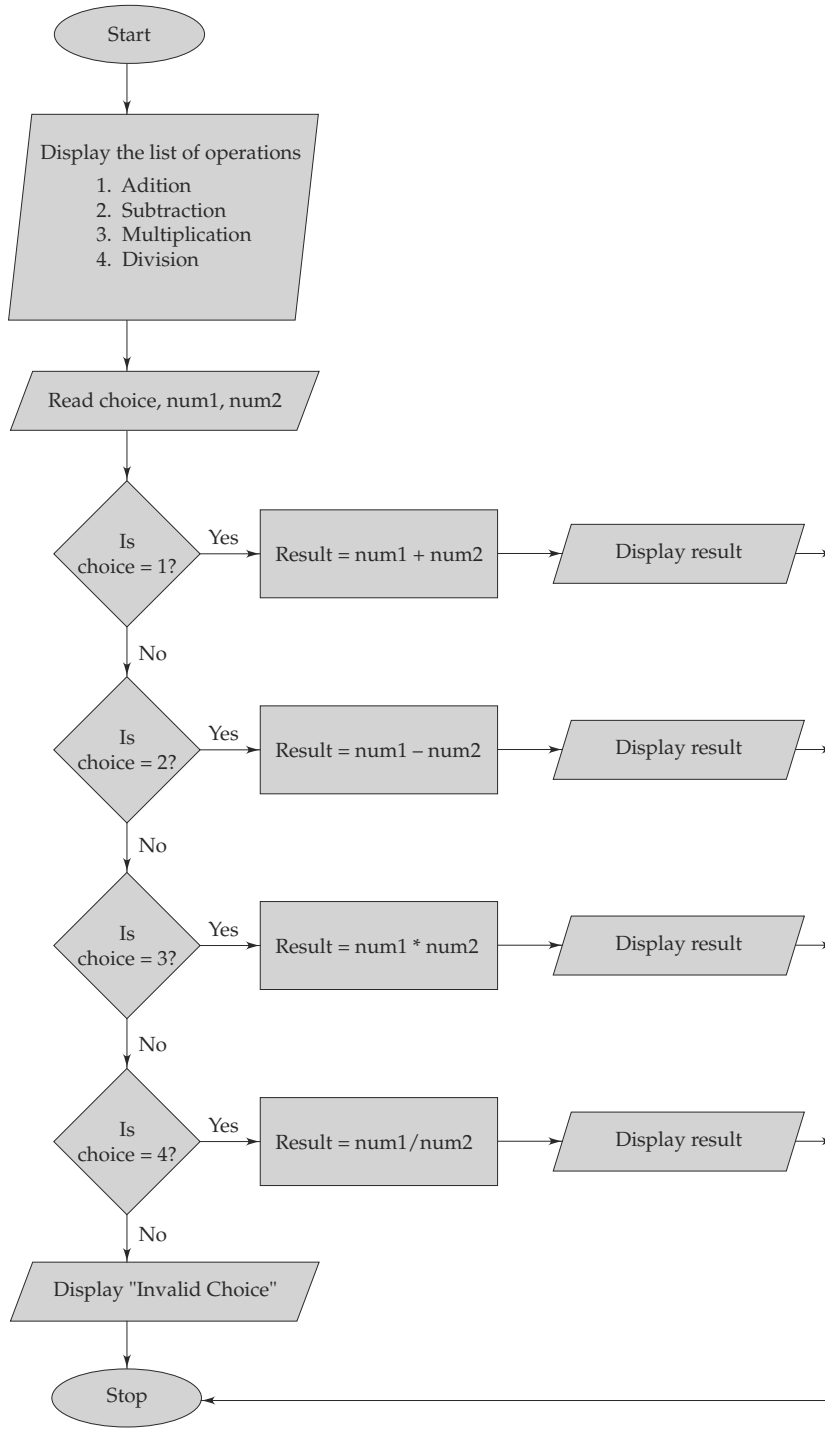
```
Step 1 – Start
Step 2 – Display a list of operations for the user to choose from
          1. Addition
          2. Subtraction
          3. Multiplication
          4. Division
Step 3 – Read the choice entered by the user (choice)
Step 4 – Read the two operands (num1, num2)
Step 5 – If choice = 1 goto Step 6 else goto Step 7
Step 6 – Calculate num1 + num2, display the result and goto Step 14
Step 7 – If choice = 2 goto Step 8 else goto Step 9
Step 8 – Calculate num1 - num2, display the result and goto Step 14
Step 9 – If choice = 3 goto Step 10 else goto Step 11
Step 10 – Calculate num1 X num2, display the result and goto Step 14
```

```
Step 11 – If choice = 4 goto Step 12 else goto Step 13
Step 12 – Calculate num1 / num2, display the result and goto Step 14
Step 13 – Display the message "Invalid Choice"
Step 14 – Stop
```

Flowchart

Program

```
#include <stdio.h>
#include <conio.h>

void main()
{
int choice;
float num1, num2;
clrscr();
printf("**********Simple Calc**********");
printf("\n\nChoose a type of operation from the following: ");
printf("\n\t1.  Addition");
printf("\n\t2.  Subtraction");
printf("\n\t3.  Multiplication");
printf("\n\t4.  Division\n");
scanf("%d", &choice);
printf("\n\nEnter the two operands: ");
scanf("%f %f", &num1, & num2);

switch (choice)
{
case 1:
printf("\n%.2f + %.2f = %.2lf", num1, num2, num1+num2);
break;
case 2:
printf("\n%.2f - %.2f = %.2lf", num1, num2, num1-num2);
break;
case 3:
printf("\n%.2f * %.2f = %.2lf", num1, num2, num1*num2);
break;
case 4:
printf("\n%.2f / %.2f = %.2lf", num1, num2, num1/num2);
break;
```


Start
Display the list of operations
1. Adition
2. Subtraction
3. Multiplication
4. Division
Read choice, num1, num2
Is choice = 1?
Yes
Result = num1 + num2
Display result
No
Is choice = 2?
Yes
Result = num1 – num2
Display result
No
Is choice = 3?
Yes
Result = num1 * num2
Display result
No
Is choice = 4?
Yes
Result = num1/num2
Display result
No
Display "Invalid Choice"
Stop

```
default:
printf(
"\nIncorrect Choice!");
}
getch();
}
```

Output

```
**********Simple Calc***********

Choose a type of operation from the following:
          1.  Addition
          2.  Subtraction
          3.  Multiplication
          4.  Division
3

Enter the two operands: 18.25 2.23

18.25 * 2.23 = 40.70
```

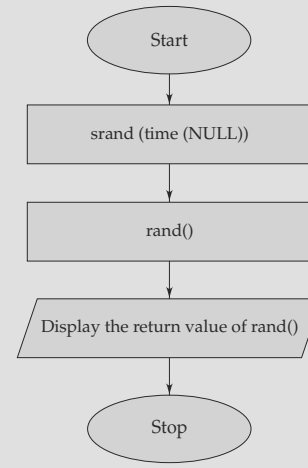
EXAMPLE C.4 *Write a program that generates random numbers.*

Solution

Algorithm

```
Step 1 – Start
Step 2 – Pass the system generated time value as a seed to the srand function,
         srand(time(NULL))
Step 3 – Call the rand function to generate a random number, rand()
Step 4 – Display the generated random number value
Step 5 – Stop
```

Flowchart



Program

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
void main()
{
clrscr();
```

```
srand(time(NULL));
printf("The system generated random number is: %d", rand());
getch();
}
```

Output

```
The system generated random number is: 23176
```

EXAMPLE C.5 *Write a program to display the Pascal's triangle.*

Solution

Algorithm

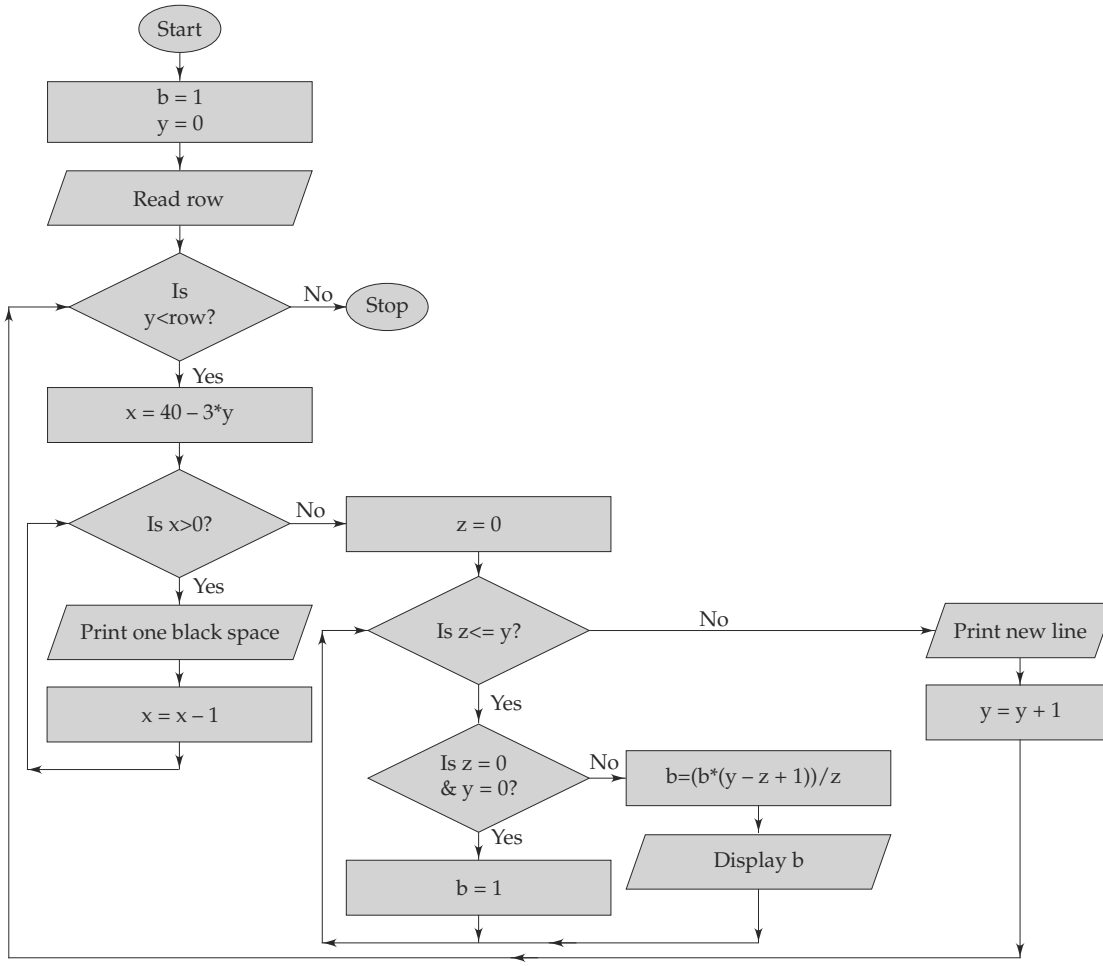
```
Step 1 – Start
Step 2 – Set b = 1 and y = 0
Step 3 – Read the number of rows for the Pascal's triangle (row)
Step 4 – Repeat Steps 5-17 while y < row
Step 5 – Initialize the looping counter x = 40-3*y
Step 6 – Repeat Steps 7-8 while x > 0
Step 7 – Print a blank space on the output screen
Step 8 – x = x - 1
Step 9 – Initialize the looping counter z = 0
Step 10 – Repeat Steps 11-15 while z <= y
Step 11 – If z = 0 OR y = 0 goto Step 12 else goto Step 13
Step 12 – b = 1
Step 13 – b= (b*(y-z+1))/z
Step 14 – Display the value of b in a field width of 6 characters
Step 15 – z = z + 1
Step 16 – Print a new line character
Step 17 – y = y + 1
Step 18 – Stop
```

Flowchart

Program

```
#include <stdio.h>
#include <conio.h>

void main()
{
  int b,row,x,y,z;
  clrscr();
```

```
b=1;
y=0;
printf("Enter the number of rows for the Pascal's triangle:");
scanf("%d",&row);
printf("\n******Pascal's Triangle******\n");

while(y<row)
{
    for(x=40-3*y;x>0;--x)
    printf(" ");
    for(z=0;z<=y;++z)
```



```
      {
         if((z==0)||(y==0))
            b=1;
         else
            b=(b*(y-z+1))/z;
            printf("%6d",b);
      }
   printf("\n");
   ++y;
   }
 getch();
 }
```

Output

```
Enter the number of rows for the Pascal's triangle: 6

******Pascal's Triangle******
                                        1
                                    1   1
                                1   2   1
                            1   3   3   1
                        1   4   6   4   1
                    1   5   10  10  5   1
```

EXAMPLE C.6 *Write a program to display a pyramid.*

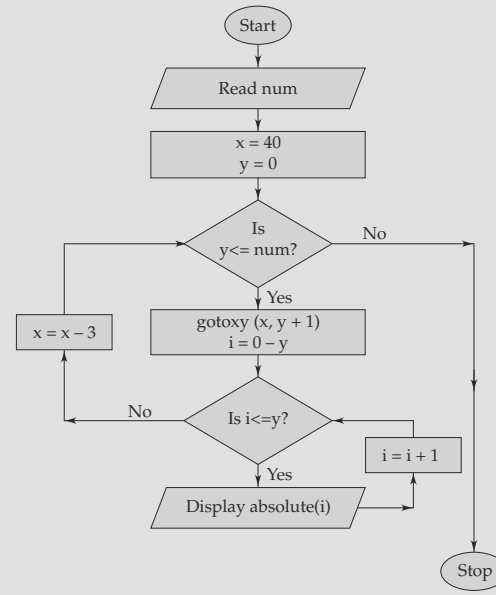
Solution

Algorithm

```
Step 1 – Start
Step 2 – Read a value for generating the pyramid (num)
Step 3 – Set x = 40
Step 4 – Initialize the looping counter y=0
Step 5 – Repeat Steps 6-12 while y <= num
Step 6 – Move to the coordinate position (x, y+1)
Step 7 – Initialize the looping counter i=0-y
Step 8 – Repeat Steps 9-10 while i <= y
Step 9 – Display the absolute value of i, abs(i)
Step 10 – i = i + 1
```

```
Step 11 – x = x – 3
Step 12 – y = y + 1
Step 13 – Stop
```

Flowchart



Program

```
#include <stdio.h>
#include <conio.h>

void main()
{
  int num,i,y,x=40;
  clrscr();
  printf("\nEnter a number for \ngenerating the
  pyramid:\n");
  scanf("%d",&num);
  for(y=0;y<=num;y++)
  {
     gotoxy(x,y+1);
     for(i=0-y;i<=y;i++)
     printf("%3d",abs(i));
     x=x-3;
  }
  getch();
}
```

Output

```
Enter a number for
generating the pyramid:
7
                                        0
                                     1  0  1
                                  2  1  0  1  2
                               3  2  1  0  1  2  3
                            4  3  2  1  0  1  2  3  4
                         5  4  3  2  1  0  1  2  3  4  5
                      6  5  4  3  2  1  0  1  2  3  4  5  6
                   7  6  5  4  3  2  1  0  1  2  3  4  5  6  7
```

EXAMPLE C.7 *Write a program to find the one's compliment of a binary number.*

Solution

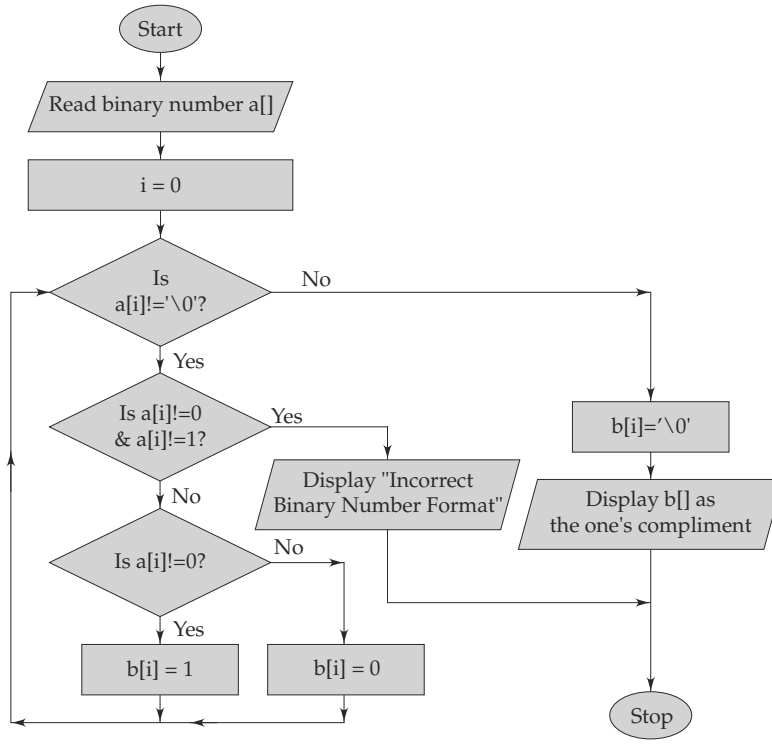
Algorithm

```
Step 1 – Start
Step 2 – Read a binary number string (a[])
Step 3 – Initialize the looping counter i=0
Step 4 – Repeat Steps 5-9 while a[i] != '\0'
Step 5 – If a[i]!= 0 AND a[i]!= 1 goto Step 6 else goto Step 7
Step 6 – Display error "Incorrect binary number format" and terminate the program
Step 7 – If a[i] = 0 goto Step 8 else goto Step 9
Step 8 – b[i]='1'
Step 9 – b[i]='0'
Step 10 – b[i] = '\0'
Step 11 – Display b[] as the one's compliment of the binary number a[]
Step 12 – Stop
```

Flowchart

Program

```
#include <stdio.h>
#include <conio.h>

void main()
{
  char a[16],b[16];
  int i;
  clrscr();
  printf("Enter a binary number: ");
  gets(a);
  for(i=0;a[i]!='\0'; i++)
  {
     if (a[i]!='0' && a[i]!='1')
     {
        printf("\nIncorrect binary number format...the program will quit");
        getch();
        exit(0);
     }
```




```
        if (a[i]=='0')
         b[i]='1';
        else
         b[i]='0';
   }
   b[i]='\0';
   printf("\nThe 1's compliment of %s is %s", a,b);
   getch();
}
```

Output

```
Enter a binary number: 11001210
Incorrect binary number format...the program will quit

Enter a binary number: 1101101
The 1's compliment of 1101101 is 0010010
```

EXAMPLE C.8 *Write a program to find the two's compliment of a binary number.*

Solution

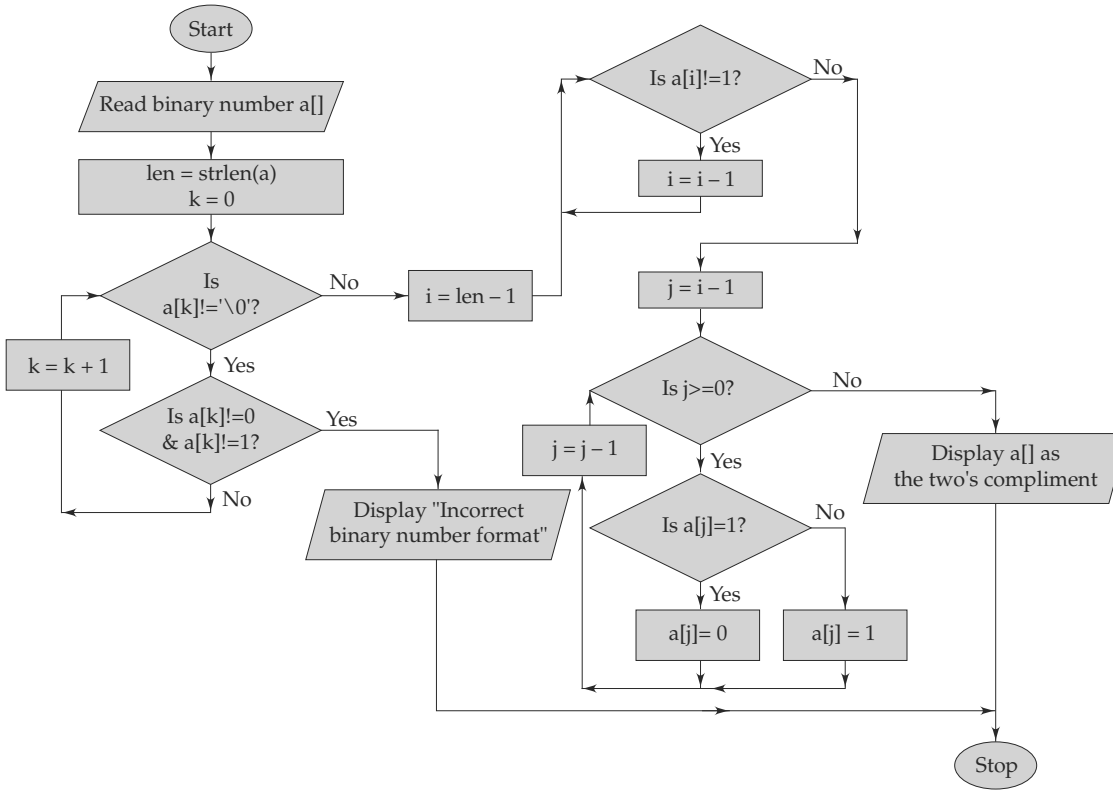
Algorithm

```
Step 1 – Start
Step 2 – Read a binary number string (a[])
Step 3 – Calculate the length of string str (len)
Step 4 – Initialize the looping counter k=0
Step 5 – Repeat Steps 6-8 while a[k] != '\0'
Step 6 – If a[k]!= 0 AND a[k]!= 1 goto Step 7 else goto Step 8
Step 7 – Display error "Incorrect binary number format" and terminate the program
Step 8 – k = k + 1
Step 9 – Initialize the looping counter i = len - 1
Step 10 – Repeat Step 11 while a[i]!='1'
Step 11 – i = i - 1
Step 12 – Initialize the looping counter j = i - 1
Step 13 – Repeat Step 14-17 while j >= 0
Step 14 – If a[j]=1 goto Step 15 else goto Step 16
Step 15 – a[j]='0'
Step 16 – a[j]='1'
Step 17 – j = j - 1
Step 18 – Display a[] as the two's compliment
Step 19 – Stop
```

Flowchart

Program

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

void main()
{
  char a[16];
  int i,j,k,len;
  clrscr();
  printf("Enter a binary number: ");
  gets(a);
  len=strlen(a);
  for(k=0;a[k]!='\0'; k++)
```




```
{
  if (a[k]!='0' && a[k]!='1')
    {
      printf("\nIncorrect binary number format...the program will quit");
      getch();
      exit(0);
    }
}
for(i=len-1;a[i]!='1'; i--)
;
for(j=i-1;j>=0;j--)
{
if(a[j]=='1')
a[j]='0';
else
a[j]='1';
```

```
    }
    printf("\n2's compliment = %s",a);
    getch();
}
```

Output

```
Enter a binary number: 01011001001

2's compliment = 10100110111
```

EXAMPLE C.9 *Write a program to find the number of instances of different digits in a given number.*

Solution

Algorithm

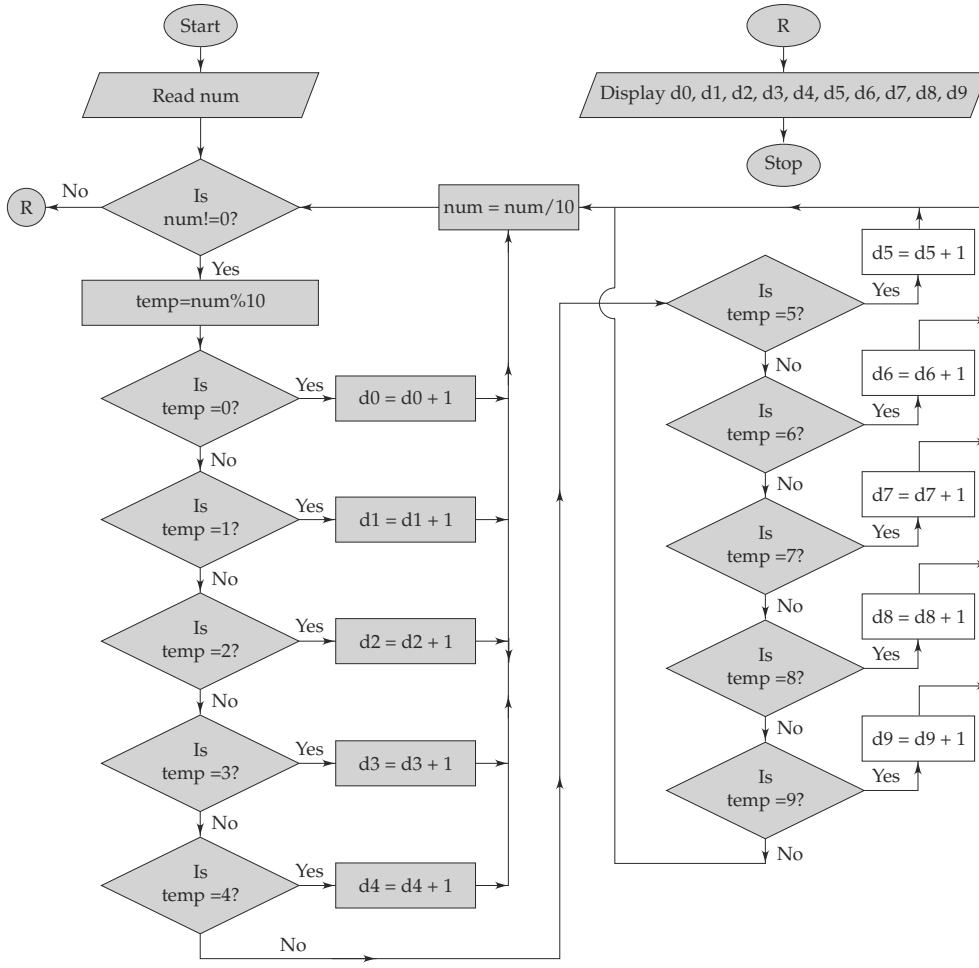
```
Step 1 – Start
Step 2 – Read an integer number (num)
Step 3 – Repeat steps 4-25 while (num!=0)
Step 4 – Calculate temp = num % 10
Step 5 – If temp = 0 goto Step 6 else goto Step 7
Step 6 – Increment the 0-digit counter by 1 (d0=d0+1)
Step 7 – If temp = 1 goto Step 8 else goto Step 9
Step 8 – Increment the 1-digit counter by 1 (d1=d1+1)
Step 9 – If temp = 2 goto Step 10 else goto Step 11
Step 10 – Increment the 2-digit counter by 1 (d2=d2+1)
Step 11 – If temp = 3 goto Step 12 else goto Step 13
Step 12 – Increment the 3-digit counter by 1 (d3=d3+1)
Step 13 – If temp = 4 goto Step 14 else goto Step 15
Step 14 – Increment the 4-digit counter by 1 (d4=d4+1)
Step 15 – If temp = 5 goto Step 16 else goto Step 17
Step 16 – Increment the 5-digit counter by 1 (d5=d5+1)
Step 17 – If temp = 6 goto Step 18 else goto Step 19
Step 18 – Increment the 6-digit counter by 6 (d6=d6+1)
Step 19 – If temp = 7 goto Step 20 else goto Step 21
Step 20 – Increment the 7-digit counter by 1 (d7=d7+1)
Step 21 – If temp = 8 goto Step 22 else goto Step 23
Step 22 – Increment the 8-digit counter by 1 (d8=d8+1)
Step 23 – If temp = 9 goto Step 24 else goto Step 25
Step 24 – Increment the 9-digit counter by 1 (d9=d9+1)
```

```
Step 25 – Set num = num / 10
Step 26 –  Display the number of instances of digits (0-9) present in the number num (d0,
           d1, d2, d3, d4, d5, d6, d7, d8, d9)
Step 27 – Stop
```

Flowchart

Program

```
#include <stdio.h>
#include <conio.h>
void main()
```



```
{
  long int num1,num2;
  int temp,d1=0,d2=0,d3=0,d4=0,d5=0,d6=0,d7=0,d8=0,d9=0,d0=0;
  clrscr();
  printf("\nEnter the number:");
  scanf("%ld",&num1);
  num2=num1;
  while(num1!=0)
  {
     temp=num1%10;
     switch(temp)
     {
        case 0:
           d0++;
           break;
        case 1:
           d1++;
           break;
        case 2:
           d2++;
           break;
        case 3:
           d3++;
           break;
        case 4:
           d4++;
           break;
        case 5:
           d5++;
           break;
        case 6:
           d6++;
           break;
        case 7:
           d7++;
           break;
        case 8:
           d8++;
           break;
```

```
        case 9:
           d9++;
           break;
      }
      num1=num1/10;
   }
   printf("\nThe no of 0s in %ld are %d",num2,d0);
   printf("\nThe no of 1s in %ld are %d",num2,d1);
   printf("\nThe no of 2s in %ld are %d",num2,d2);
   printf("\nThe no of 3s in %ld are %d",num2,d3);
   printf("\nThe no of 4s in %ld are %d",num2,d4);
   printf("\nThe no of 5s in %ld are %d",num2,d5);
   printf("\nThe no of 6s in %ld are %d",num2,d6);
   printf("\nThe no of 7s in %ld are %d",num2,d7);
   printf("\nThe no of 8s in %ld are %d",num2,d8);
   printf("\nThe no of 9s in %ld are %d",num2,d9);
   getch();
}
```

Output

```
Enter the number:28544401

The no of 0s in 28544401 are 1
The no of 1s in 28544401 are 1
The no of 2s in 28544401 are 1
The no of 3s in 28544401 are 0
The no of 4s in 28544401 are 3
The no of 5s in 28544401 are 1
The no of 6s in 28544401 are 0
The no of 7s in 28544401 are 0
The no of 8s in 28544401 are 1
The no of 9s in 28544401 are 0
```

EXAMPLE C.10 *Write a program to find the number of vowels and consonants in a text string.*

Solution

Algorithm

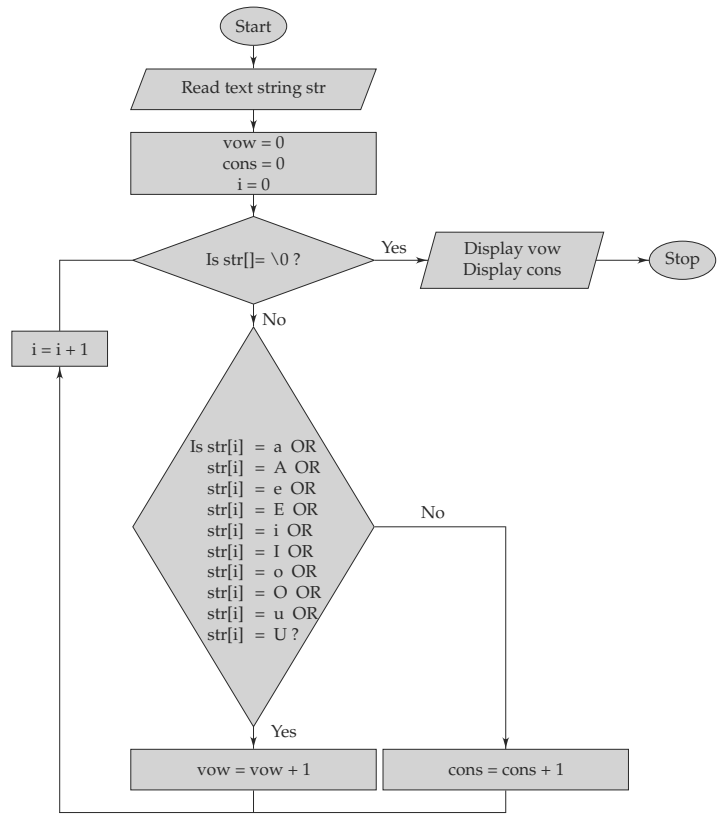
```
Step 1 – Start
Step 2 – Read a text string (str)
Step 3 – Set vow = 0, cons = 0, i = 0
```

```
Step 4 – Repeat steps 5-8 while (str[i]!='\0')
Step 5 – if str[i] = 'a' OR str[i] = 'A' OR str[i] = 'e' OR str[i] = 'E' OR str[i] = 'i' OR
         str[i] = 'I' OR str[i] = 'o' OR str[i] = 'O' OR str[i] = 'u' OR str[i] = 'U' goto
         Step 6 else goto Step 7
Step 6 – Increment the vowels counter by 1 (vow=vow+1)
Step 7 – Increment the consonants counter by 1 (cons=cons+1)
Step 8 – i = i + 1
Step 9 – Display the number of vowels and consonants (vow, cons)
Step 10 – Stop
```

Flowchart

Program

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
```



```
{
      char str[30];
      int vow=0,cons=0,i=0;
      clrscr();

      printf("Enter a string: ");
      gets(str);

      while(str[i] != '\0')
      {
        if(str[i] == 'a' || str[i] == 'A' || str[i] == 'e' || str[i] == 'E' || str[i] == 'i'
|| str[i] == 'I' || str[i] == 'o' || str[i] == 'O' || str[i] == 'u' || str[i] == 'U')
            vow++;
        else
            cons++;
        i++;

      }

        printf("\nNumber of Vowels = %d",vow);
        printf("\nNumber of Consonants = %d",cons);

      getch();
}
```

Output

```
Enter a string: Chennai

Number of Vowels = 3
Number of Consonants = 4
```

EXAMPLE C.11 *Write a program that uses a simple structure for storing different students' details.*

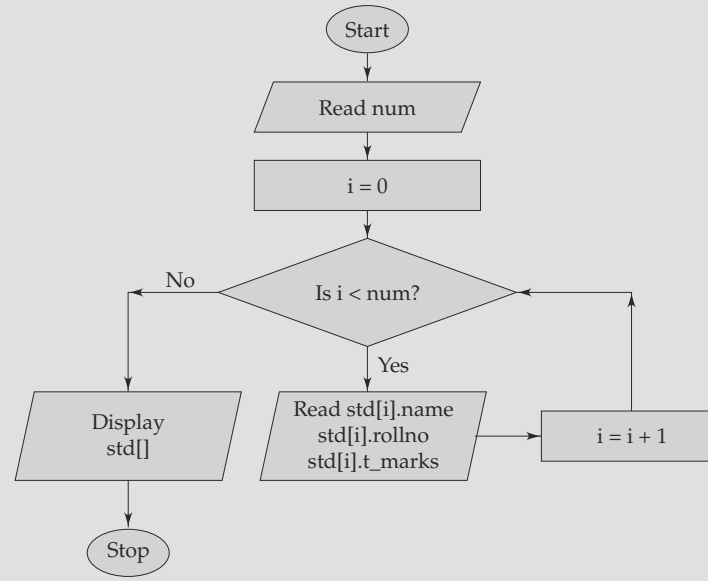
Solution

Algorithm

```
Step 1 – Start
Step 2 – Define a simple structure to store student details
          STRUCTURE student
          STRING name
          INTEGER rollno
          INTEGER t_marks
          END STRUCTURE
```

```
          STRUCTURE student std[]
Step 3 – Read the number of students for which details are to be entered (num)
Step 4 – Initialize looping counter i = 0
Step 5 – Repeat Steps 6=8 while i < num
Step 6 –  Read student's name, roll no and total marks (std[i].name, std[i].rollno, std[i].t_
          marks)
Step 7 – i = i + 1
Step 8 – Display the different students' details stored in structure array std[]
Step 9 – Stop
```

Flowchart



Program

```
#include <stdio.h>
#include <conio.h>
void main ()
{
int num, i=0;
struct student
{
char name[30];
int rollno;
int t_marks;
};
struct student std[10];
clrscr();
printf("Enter the number of students: ");
scanf("%d",&num);

for(i=0;i<num;i++)
{
printf("\nEnter the details for %d student",i+1);
printf("\n\n Name ");
scanf("%s",std[i].name);
printf("\n Roll No. ");
scanf("%d",&std[i].rollno);
printf("\n Total Marks ");
scanf("%d",&std[i].t_marks);
}
printf("\n Press any key to display the student details!");
getch();
```

```
for(i=0;i<num;i++)
printf("\nstudent %d \n Name %s \n Roll No. %d \n Total Marks
%d\n",i+1,std[i].name, std[i].rollno, std[i].t_marks);
getch();
}
```

Output

```
Enter the number of students: 3
Enter the details for 1 student
  Name Arjun
  Roll No. 1
  Total Marks 399
Enter the details for 2 student
  Name Binoy
  Roll No. 2
  Total Marks 432
Enter the details for 3 student
  Name Chitra
  Roll No. 3
  Total Marks 402
  Press any key to display the student details!
student 1
  Name Arjun
  Roll No. 1
  Total Marks 399
student 2
  Name Binoy
  Roll No. 2
  Total Marks 432
student 3
  Name Chitra
  Roll No. 3
  Total Marks 402
```

EXAMPLE C.12 *Write a program to find the sum of the following series:*

$$1 + x + x^2 + x^3 + \ldots\ldots\ldots + x^n$$

Solution

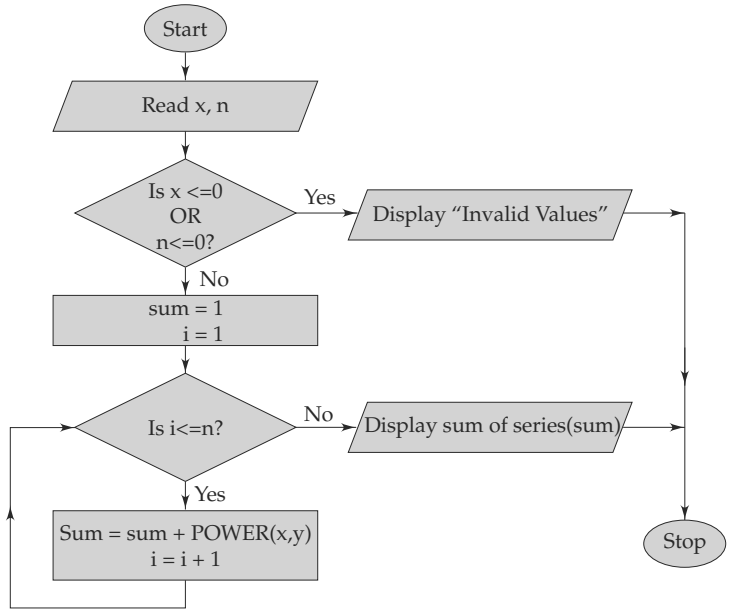
Algorithm

```
Step 1 – Start
Step 2 – Read the values of x and n
Step 3 – If n <= 0 OR x <=0 goto Step 4 else goto Step 5
Step 4 – Display error "Invalid values" and terminate the program
Step 5 – Set sum = 1
Step 6 – Initialize the looping counter i = 1
Step 7 – Repeat Steps 8-9 while i<=n
Step 8 – sum = sum + POWER(x,i)
Step 9 – i = i + 1
Step 10 – Display sum as the resultant sum of the series
Step 11 – Stop
```

Flowchart

Program

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

void main()
{

long sum;
int n,x,i;
clrscr();

printf("Enter the values of x and n:");
scanf("%d %d",&x,&n);

if(n<=0 || x<=0)
{
printf("The values must be positive integers. Please try again\n");
getch();
}
else
{
```




```
    sum=1;
    for(i=1;i<=n;i++)
    {
        sum=sum+pow(x,i);
    }
    printf("Sum of series=%ld\n",sum);
}
getch();
}
```

Output

```
Enter the values of x and n:2
5
Sum of series = 63
```

EXAMPLE C.13 *Write a program to find the sum of the following series:*

$$1 + 2 + 3 + \ldots + n$$

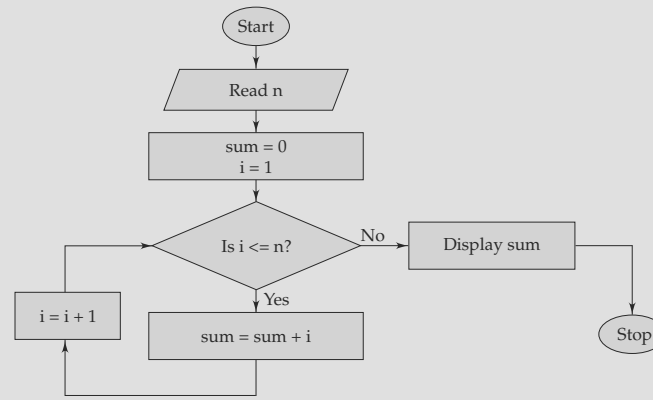
Solution

Algorithm

```
Step 1 – Start
```

```
Step 2 – Read n
Step 3 – Set sum = 0
Step 4 – Initialize the looping counter i = 1
Step 5 – Repeat Steps 6-8 while i<=n
Step 6 – sum = sum + i
Step 7 – i = i + 1
Step 8 – Display sum as the resultant sum of the series
Step 9 – Stop
```

Flowchart



Program

```
#include <stdio.h>
#include <conio.h>

void main()
{
int i, n;
long sum=0;
clrscr();
printf("Enter the value of n");
scanf("%d",&n);

for(i=1;i<=n;i++)
sum = sum + i;
printf("\nThe Sum of the series 1 + 2 + .... + n (for n = %d) is %ld",n,sum);
getch();
}
```

Output

```
Enter the value of n 6

The Sum of the series 1 + 2 + .... + n (for n = 6) is 21
```

EXAMPLE C.14 *Write a program to print the value and address of variables.*

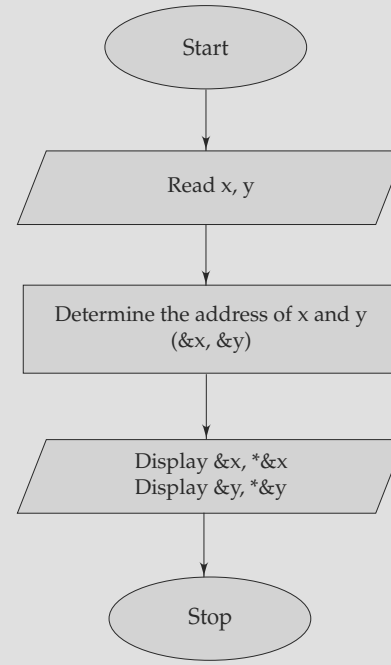
Solution

Algorithm

```
Step 1 – Start
Step 2 – Read the values of x and y
Step 3 – Determine the addresses of x and y using ampersand (&) operator (&x, &y)
Step 4 – Print the address and value of x (&x, *&x)
```

```
Step 5 – Print the address and value of y (&y, *&y)
Step 6 – Stop
```

Flowchart



Program

```
#include <stdio.h>
#include <conio.h>
void main ()
{
int x,y;
clrscr();
printf("Enter the values of x and y ");
scanf("%d %d",&x,&y);
printf("Address of x is %u",&x);
printf("\nValue of x is %d",*(&x));
printf("\nAddress of y is %u",&y);
printf("\nValue of y is %d",*(&y));
getch();
}
```

Output

```
Enter the values of x and y 22
44
Address of x is 65524
Value of x is 22
Address of y is 65522
Value of y is 44
```

EXAMPLE C.15 *Write a program to copy the contents of one file into another.*

Solution

Algorithm

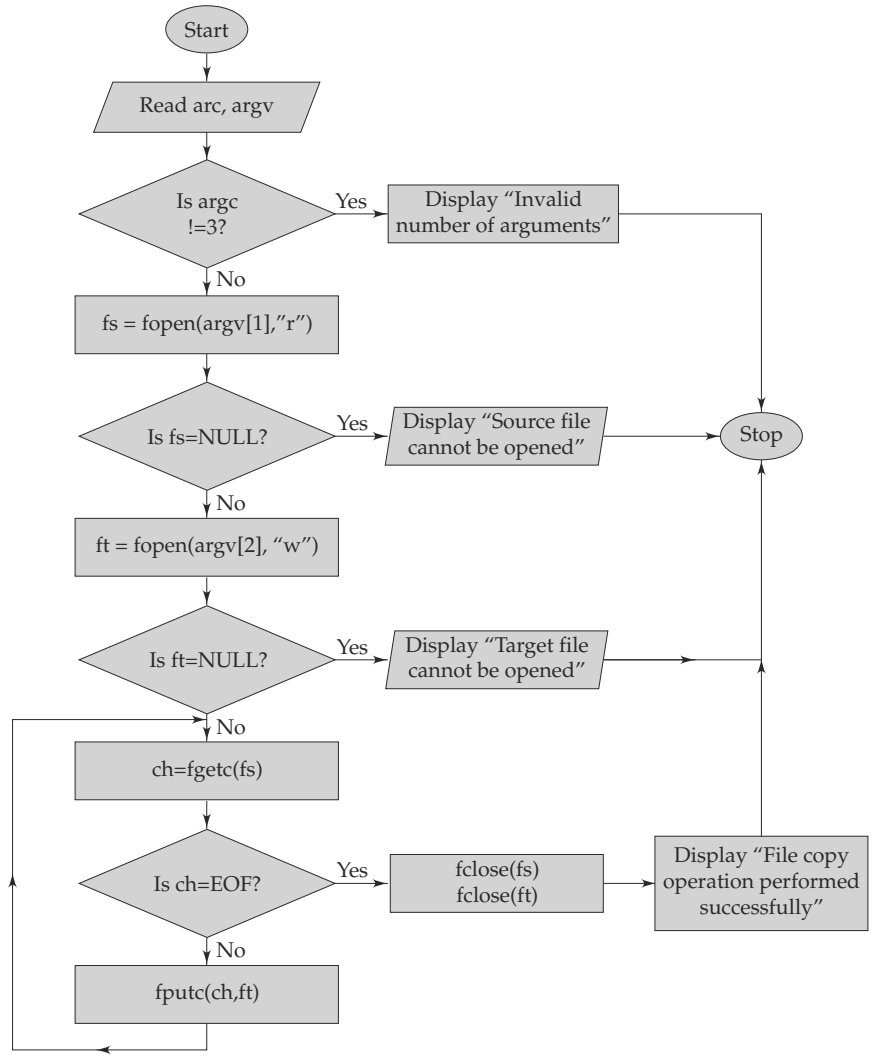
```
Step 1 – Start
Step 2 – Read the command line arguments (argc, argv)
Step 3 – If argc !=3 goto Step 4 else goto Step 5
Step 4 – Display "Invalid number of arguments" and terminate the program
Step 5 – Open the source file specified by argv[1] in read mode and assign its starting location
         to file pointer fs (fs = fopen(argv[1],"r"))
Step 6 – If fs=NULL goto Step 7 else goto Step 8
Step 7 – Display "Source file cannot be opened" and terminate the program
```

```
Step 8 – Open the target file specified by argv[2] in write mode and assign its starting
         location to file pointer ft (ft = fopen(argv[2],"w"))
Step 9 – If ft=NULL goto Step 10 else goto Step 11
Step 10 – Display "Target file cannot be opened" and terminate the program
Step 11 – Repeat Steps 12-14 indefinitely
Step 12 – Read the first character of the source file (ch)
Step 13 – If ch = EOF goto Step 15 else goto Step 14
Step 14 – Copy character ch into the target file
Step 15 – Close the file pointers fs and ft
Step 16 – Display "Files copied successfully"
Step 17 – Stop
```

Flowchart

Program

```
#include <stdio.h>
#include <conio.h>

void main(int argc, char *argv[])
{
  FILE *fs,*ft;
  char ch;
  clrscr();
  if(argc!=3)
  {
     printf("Invalid number of arguments.");
     exit(0);
  }
  fs = fopen(argv[1],"r");
  if(fs==NULL)
  {
  printf("Source file cannot be opened.");
  exit(0);
  }
  ft = fopen(argv[2],"w");
  if (ft==NULL)
  {
  printf("Target file cannot be opened.");
  fclose(fs);
  exit(0);
```




```
}
while(1)
{
ch=fgetc(fs);
if (ch==EOF)
break;
else
fputc(ch,ft);
}
```

```
    fclose(fs);
    fclose(ft);
    printf("\nFile copy operation performed successfully");
    getch();
}
```

Output

```
D:\TC\BIN>15.exe s1.txt t1.txt
File copy operation performed successfully
```

EXAMPLE C.16 *Write a program to count the number of characters in a file.*

Solution

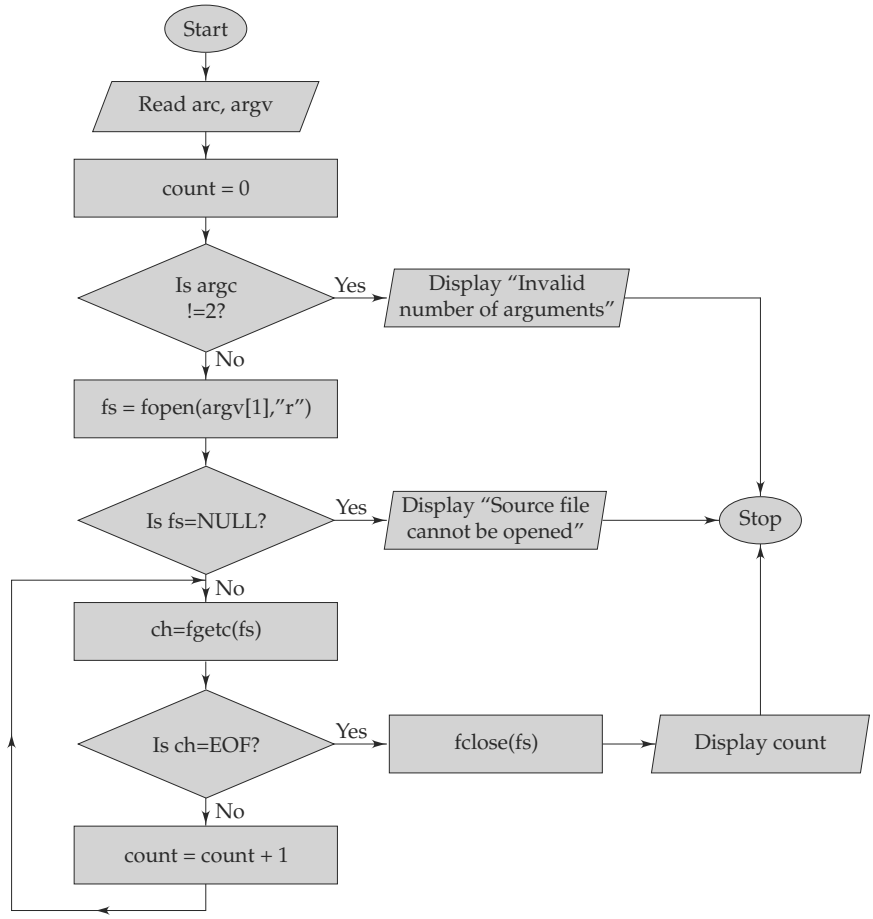
Algorithm

```
Step 1 – Start
Step 2 – Read the command line arguments (argc, argv)
Step 3 – Initialize count = 0
Step 4 – If argc !=2 goto Step 5 else goto Step 6
Step 5 – Display "Invalid number of arguments" and terminate the program
Step 6 – Open the source file specified by argv[1] in read mode and assign its starting location
         to file pointer fs (fs = fopen(argv[1],"r"))
Step 7 – If fs=NULL goto Step 8 else goto Step 9
Step 8 – Display "Source file cannot be opened" and terminate the program
Step 9 – Repeat Steps 10-12 indefinitely
Step 10 – Read the first character of the source file (ch)
Step 11 – If ch = EOF goto Step 13 else goto Step 12
Step 12 – count = count + 1
Step 13 – Close the file pointer fs
Step 14 – Display count as the number characters contained in the source file
Step 15 – Stop
```

Flowchart

Program

```
#include <stdio.h>
#include <conio.h>

void main(int argc, char *argv[])
{
  FILE *fs;
   char ch;
```




```
 long count=0;
 clrscr();
 if(argc!=2)
 {
   printf("Invalid number of arguments.");
   exit(0);
 }
fs = fopen(argv[1],"r");
if(fs==NULL)
{
printf("Source file cannot be opened.");
exit(0);
```

```
 }
 while(1)
 {
   ch=fgetc(fs);
   if (ch==EOF)
   break;
   else
   count=count+1;
 }
 fclose(fs);
 printf("\nThe number of characters in %s is %ld",argv[1],count);
 getch();
}
```

Output

```
D:\TC\BIN>16.exe s1.txt

The number of characters in s1.txt is 15
```

EXAMPLE C.17 *Write a program to find the transpose of a matrix.*

Solution

Algorithm

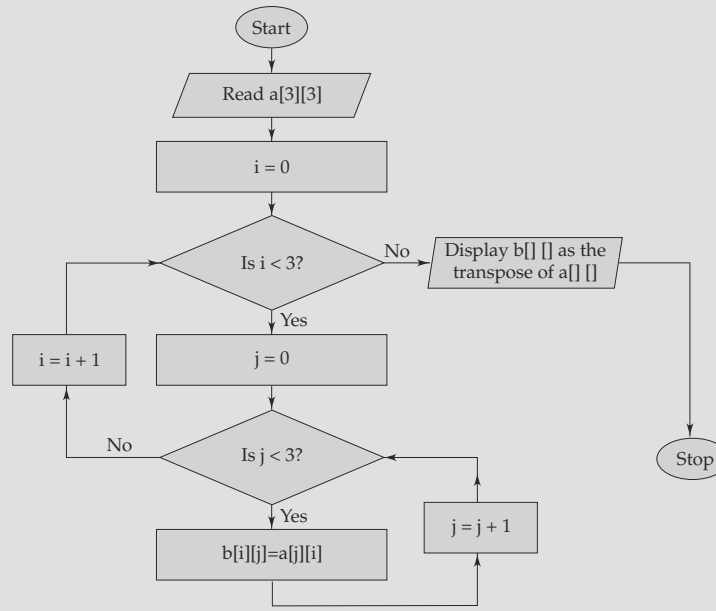
```
Step 1 – Start
Step 2 – Read a 3 X 3 matrix (a[3][3])
Step 3 – Initialize the looping counter i = 0
Step 4 – Repeat Steps 5-9 while i<3
Step 5 – Initialize the looping counter j = 0
Step 6 – Repeat Steps 7-8 while j<3
Step 7 – b[i][j]=a[j][i]
Step 8 – j = j + 1
Step 9 – i = i + 1
Step 10 – Display b[][] as the transpose of the matrix a[][]
Step 11 – Stop
```

Flowchart

Program

```
#include <stdio.h>
#include <conio.h>
```

```
void main()
{
int i,j,a[3][3],b[3][3];
clrscr();
printf("Enter a 3 X 3 matrix:\n");
for(i=0;i<3;i++)
{
  for(j=0;j<3;j++)
  {
  printf("a[%d][%d] = ",i,j);
  scanf("%d",&a[i][j]);
  }
}
printf("\nThe entered matrix
is: \n");

for(i=0;i<3;i++)
{
  printf("\n");
  for(j=0;j<3;j++)
  {
  printf("%d\t",a[i][j]);
  }
}
for(i=0;i<3;i++)
{
  for(j=0;j<3;j++)
  b[i][j]=a[j][i];
}
printf("\n\nThe transpose of the matrix is: \n");

for(i=0;i<3;i++)
{
  printf("\n");
  for(j=0;j<3;j++)
  {
  printf("%d\t",b[i][j]);
  }
}
```



```
getch();
}
```

Output

```
Enter a 3 X 3 matrix:

a[0][0] = 1
a[0][1] = 2
a[0][2] = 3
a[1][0] = 4
a[1][1] = 5
a[1][2] = 6
a[2][0] = 7
a[2][1] = 8
a[2][2] = 9

The entered matrix is:

     1    2    3
     4    5    6
     7    8    9

The transpose of the matrix is:

     1    4    7
     2    5    8
     3    6    9
```

EXAMPLE C.18 *Write a program to add two matrices.*

Solution

Algorithm

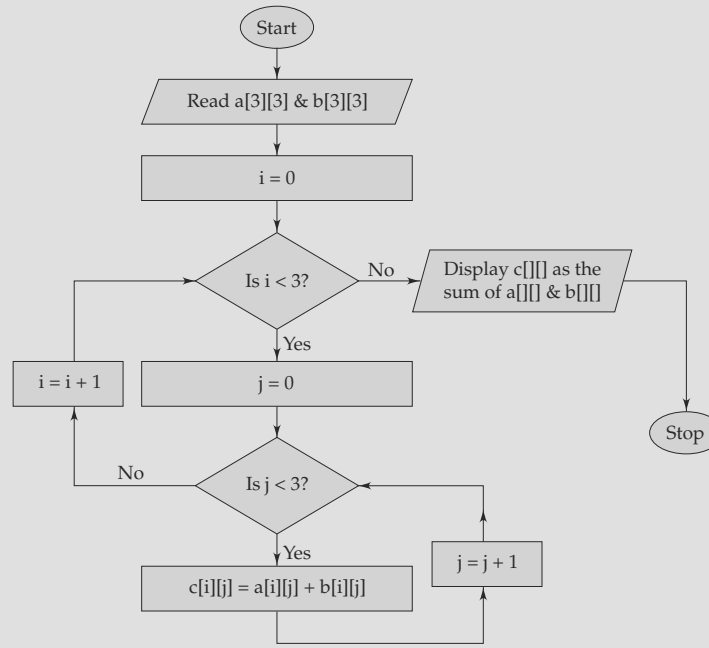
```
Step 1 – Start
Step 2 – Read two 3 X 3 matrices (a[3][3], b[3][3])
Step 3 – Initialize the looping counter i = 0
Step 4 – Repeat Steps 5-9 while i<3
Step 5 – Initialize the looping counter j = 0
Step 6 – Repeat Steps 7-8 while j<3
Step 7 – c[i][j] = a[i][j] + b[i][j]
Step 8 – j = j + 1
Step 9 – i = i + 1
Step 10 – Display c[][] as the resultant sum of the two matrices
Step 11 – Stop
```

Flowchart

Program

```
#include <stdio.h>
#include <conio.h>

void main()
{
int i,j,a[3][3],b[3][3],c[3][3];
clrscr();
printf("Enter the first 3 X 3 matrix:\n");
for(i=0;i<3;i++)
{
  for(j=0;j<3;j++)
  {
  printf("a[%d][%d] = ",i,j);
  scanf("%d",&a[i][j]);
  }
}
printf("Enter the second 3 X 3 matrix:\n");
for(i=0;i<3;i++)
{
  for(j=0;j<3;j++)
  {
  printf("b[%d][%d] = ",i,j);
  scanf("%d",&b[i][j]);
  }
}
printf("\nThe entered matrices are: \n");

for(i=0;i<3;i++)
{
  printf("\n");
  for(j=0;j<3;j++)
  {
  printf("%d\t",a[i][j]);
  }
  printf("\t\t");
  for(j=0;j<3;j++)
```

```
  {
  printf("%d\t",b[i][j]);
  }
}
for(i=0;i<3;i++)
{
  for(j=0;j<3;j++)
  c[i][j] =a[i][j]+b[i][j];
}
printf("\n\nThe sum of the two matrices is shown below: \n");

for(i=0;i<3;i++)
{
  printf("\n\t\t ");
  for(j=0;j<3;j++)
  {
  printf("%d\t",c[i][j]);
  }
}
getch();
}
```



Output

```
Enter the first 3 X 3 matrix:

a[0][0] = 1
a[0][1] = 1
a[0][2] = 1
a[1][0] = 1
a[1][1] = 1
a[1][2] = 1
a[2][0] = 1
a[2][1] = 1
a[2][2] = 1
Enter the second 3 X 3 matrix:

b[0][0] = 2
b[0][1] = 2
b[0][2] = 2
b[1][0] = 2
b[1][1] = 2
```

```
b[1][2] = 2
b[2][0] = 2
b[2][1] = 2
b[2][2] = 2

The entered matrices are:

     1    1    1            2    2    2
     1    1    1            2    2    2
     1    1    1            2    2    2

The sum of the two matrices is shown below:

     3    3    3
     3    3    3
     3    3    3
```

EXAMPLE C.19 *Write a program to multiply two matrices.*

Solution

Algorithm

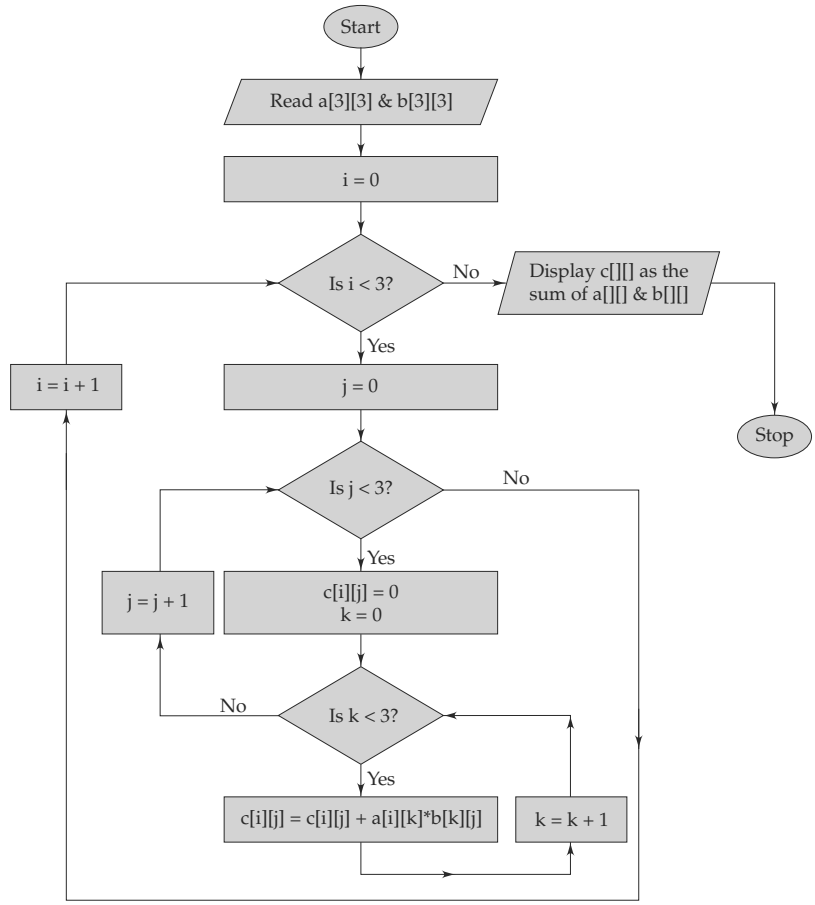
```
Step 1 – Start
Step 2 – Read two 3 X 3 matrices (a[3][3], b[3][3])
Step 3 – Initialize the looping counter i = 0
Step 4 – Repeat Steps 5-13 while i<3
Step 5 – Initialize the looping counter j = 0
Step 6 – Repeat Steps 7-12 while j<3
Step 7 – c[i][j]=0
Step 8 – Initialize the looping counter k = 0
Step 9 – Repeat Steps 10-11 while k<3
Step 10 – c[i][j]=c[i][j]+a[i][k]*b[k][j]
Step 11 – k = k + 1
Step 12 – j = j + 1
Step 13 – i = i + 1
Step 14 – Display c[][] as the resultant product of the two matrices
Step 15 – Stop
```

Flowchart

Program

```
#include <stdio.h>
#include <conio.h>
```

```
void main()
{
int i,j,k,a[3][3],b[3][3],c[3][3];
clrscr();
printf("Enter the first 3 X 3 matrix:\n");
for(i=0;i<3;i++)
{
  for(j=0;j<3;j++)
  {
  printf("a[%d][%d] = ",i,j);
  scanf("%d",&a[i][j]);
  }
}
printf("Enter the second 3 X 3 matrix:\n");
for(i=0;i<3;i++)
{
  for(j=0;j<3;j++)
  {
  printf("b[%d][%d] = ",i,j);
  scanf("%d",&b[i][j]);
  }
}
printf("\nThe entered matrices are: \n");

for(i=0;i<3;i++)
{
  printf("\n");
  for(j=0;j<3;j++)
  {
  printf("%d\t",a[i][j]);
  }
  printf("\t\t");
  for(j=0;j<3;j++)
  {
  printf("%d\t",b[i][j]);
  }
}
for(i=0;i<3;i++)
```




```
  for(j=0;j<3;j++)
  {
     c[i][j]=0;
     for(k=0;k<3;k++)
     c[i][j]=c[i][j]+a[i][k]*b[k][j];
  }
printf("\n\nThe product of the two matrices is shown below: \n");

for(i=0;i<3;i++)
{
  printf("\n\t\t ");
  for(j=0;j<3;j++)
```

```
  {
  printf("%d\t",c[i][j]);
  }
}
getch();
}
```

Output

```
Enter the first 3 X 3 matrix:

a[0][0] = 1
a[0][1] = 2
a[0][2] = 3
a[1][0] = 4
a[1][1] = 5
a[1][2] = 6
a[2][0] = 7
a[2][1] = 8
a[2][2] = 9
Enter the second 3 X 3 matrix:

b[0][0] = 1
b[0][1] = 1
b[0][2] = 1
b[1][0] = 2
b[1][1] = 2
b[1][2] = 2
b[2][0] = 3
b[2][1] = 3
b[2][2] = 3

The entered matrices are:

     1    2    3          1    1    1
     4    5    6          2    2    2
     7    8    9          3    3    3

The product of the two matrices is shown below:

     14   14   14
     32   32   32
     50   50   50
```

EXAMPLE C.20 *Write a program that uses insertion sort technique to sort an array of ten elements.*

Solution

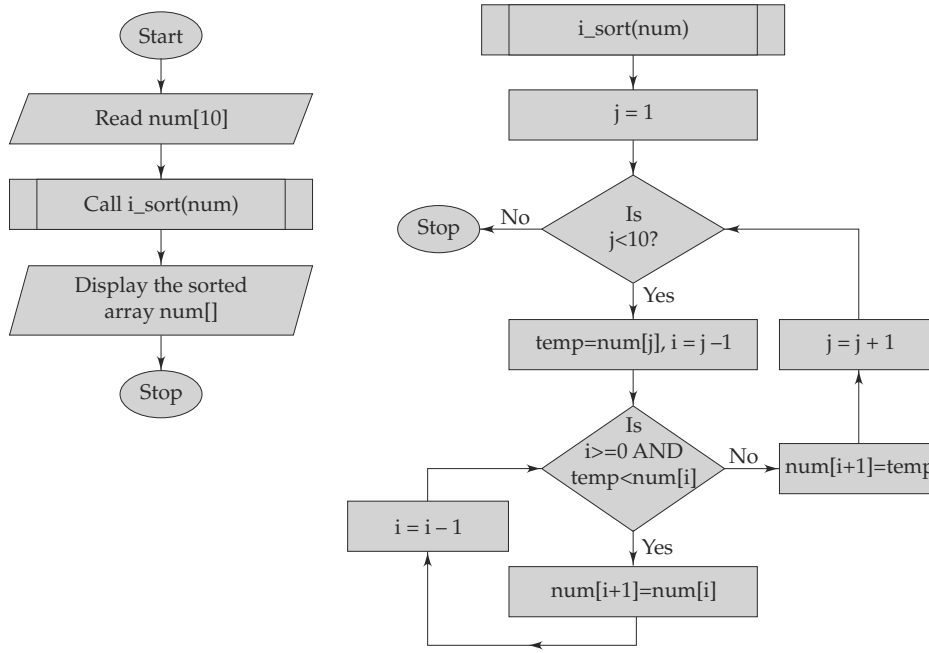
Algorithm

```
Step 1 – Start
Step 2 – Accept a ten element array which needs to be sorted (num[])
Step 3 – Call function i_sort(num)
Step 4 – Display the sorted array num[]
Step 5 – Stop

i_sort(num[])
Step 1 – Start
Step 2 – Initialize the looping counter j = 1
Step 3 – Repeat Steps 4–10 while j<10
Step 4 – Set temp = num[j]
Step 5 – Initialize the looping counter i = j-1
Step 6 – Repeat Steps 7–8 while i>=0 AND temp<num[i]
Step 7 – num[i+1]=num[i]
Step 8 – i = i - 1
Step 9 – num[i+1]=temp
Step 10 – j = j + 1
Step 11 – Stop
```

Flowchart

Program

```
#include <stdio.h>
#include <conio.h>

void main()
{
  void i_sort(int []);
  int num[10],i;
  clrscr();
     printf("\nEnter the ten elements to sort:\n");

  for (i=0;i<10;i++)
     scanf("%d",&num[i]);
     i_sort(num);
```




```
  printf("\n\nThe sorted elements are:\n");
  for(i=0;i<10;i++)
     printf("%d\n",num[i]);
     getch();
}

void i_sort(int num[])
{
  int i,j,temp;
  for(j=1;j<10;j++)
  {
     temp=num[j];
     for(i=j-1;i>=0 && temp<num[i];i--)
     num[i+1]=num[i];
     num[i+1]=temp;
  }
}
```

Output

```
Enter the ten elements to sort:
22
33
1
2
65
18
7
54
78
5

The sorted elements are:
1
2
5
7
18
22
33
54
65
78
```

EXAMPLE C.21 *Write a program that uses bubble sort technique to sort an array of ten elements.*

Solution

Algorithm

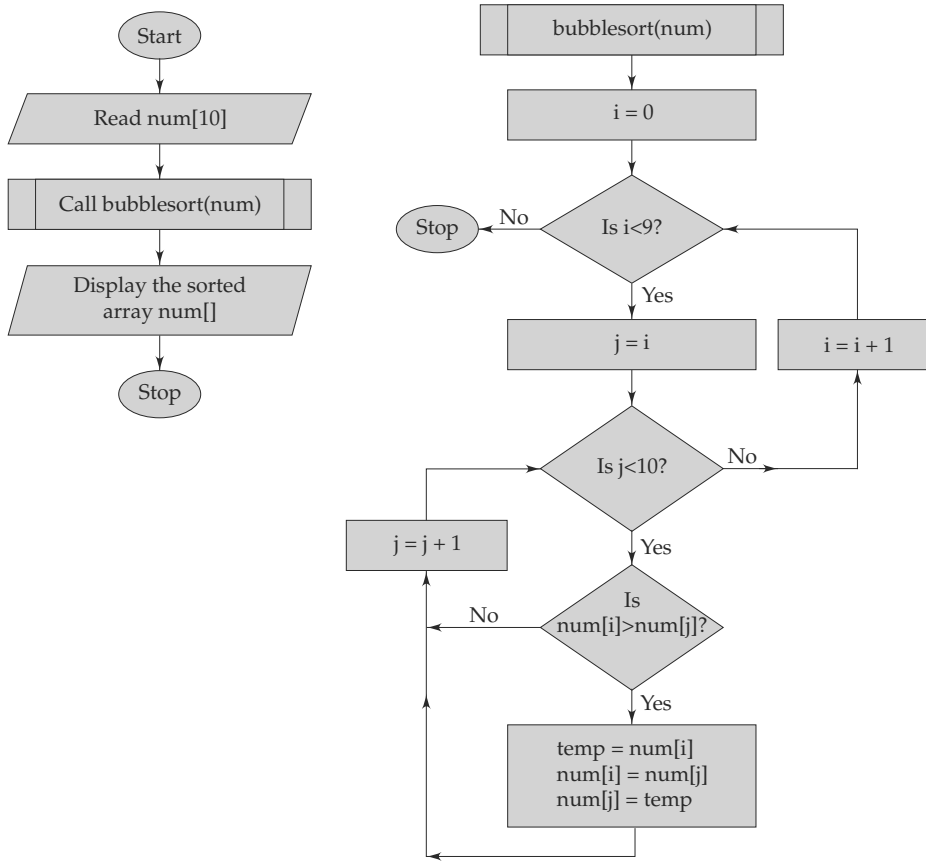
```
Step 1 – Start
Step 2 – Accept a ten element array which needs to be sorted (num[])
Step 3 – Call function bubblesort(num)
Step 4 – Display the sorted array num[]
Step 5 – Stop

bubblesort(num[])
Step 1 – Start
Step 2 – Initialize the looping counter i = 0
```

```
Step 3 – Repeat Steps 4–9 while i<9
Step 4 – Initialize the looping counter j = i
Step 5 – Repeat Steps 6–8 while j<10
Step 6 – If num[i] > num[j] goto Step 7 else goto Step 8
Step 7 – Swap the values of num[i] and num[j]
Step 8 – j = j + 1
Step 9 – i = i + 1
Step 10 – Stop
```

Flowchart

Program

```
#include <stdio.h>
#include <conio.h>
```



```
void main()
{
  void bubblesort(int num[]);
  int i, num[10];
  clrscr();
  printf("\nEnter the 10 elements to be sorted: ");
  for (i = 0; i < 10; i++)
  {
  printf ("\nEnter element %d: ",i+1);
  scanf("%d",&num[i]);
  }
  printf("\n\nThe array elements before sorting are:\n\n");
  for (i = 0; i < 10; i++)
  printf("[%d], ",num[i]);
  bubblesort(num);
  printf("\n\nThe array elements after sorting are:\n\n");
  for (i = 0; i < 10; i++)
  printf("[%d], ", num[i]);
  getch();
  }

  void bubblesort(int num[])
  {
  int i, j, temp;
  for (i = 0; i < 9; i++)
  {
     for (j = i; j < 10; j++)
     {
       if (num[i] > num[j])
       {
           temp = num[i];
           num[i] = num[j];
           num[j] = temp;
       }
     }
  }
}
```

Output

```
Enter the 10 elements to be sorted:
Enter element 1: 1
Enter element 2: 99
Enter element 3: 3
Enter element 4: 85
Enter element 5: 19
Enter element 6: 74
Enter element 7: 5
Enter element 8: 59
Enter element 9: 18
Enter element 10: 33
The array elements before sorting are:
[1], [99], [3], [85], [19], [74], [5], [59], [18], [33],
The array elements after sorting are:
[1], [3], [5], [18], [19], [33], [59], [74], [85], [99],
```

EXAMPLE C.22 *Write a program to implement stack using arrays.*

Solution

Algorithm

```
Step 1 – Start
Step 2 – Reserve a 100 element array in the memory stack[100] and set its top pointer to -1
         (top = -1)
Step 3 – Repeat Steps 4-15 indefinitely
Step 4 – Display a list of stack operations for the user to choose from
         1. Push an element into the stack
         2. Pop out an element from the stack
         3. Display the stack elements
         4. Exit
Step 5 – Read the choice entered by the user (choice)
Step 6 – If choice = 1 goto Step 7 else goto Step 9
Step 7 – Read the element to be pushed (num1)
Step 8 – Call the push function, push(num1) and goto Step 3
```

```
Step 9 – If choice = 2 goto Step 10 else goto Step 12
Step 10 – Call the pop function, pop()
Step 11 – Display the popped element and goto Step 3
Step 12 – If choice = 3 goto Step 13 else goto Step 14
Step 13 – Call the display function, display() and goto Step 3
Step 14 – If choice = 4 goto Step 16 else goto Step 15
Step 15 – Display message "Invalid Choice" and goto Step 3
Step 16 – Stop
```

push(element)

```
Step 1 – Start
Step 2 – If top = 99 goto Step 3 else goto Step 4
Step 3 – Display message "Stack Full" and exit
Step 4 – top = top + 1
Step 5 – Stack[top] = element
Step 6 – Stop
```

pop()

```
Step 1 – Start
Step 2 – If top = -1 goto Step 3 else goto Step 4
Step 3 – Display message "Stack Empty" and exit
Step 4 – Return stack[top] and set top = top - 1
Step 5 – Stop
```

display()

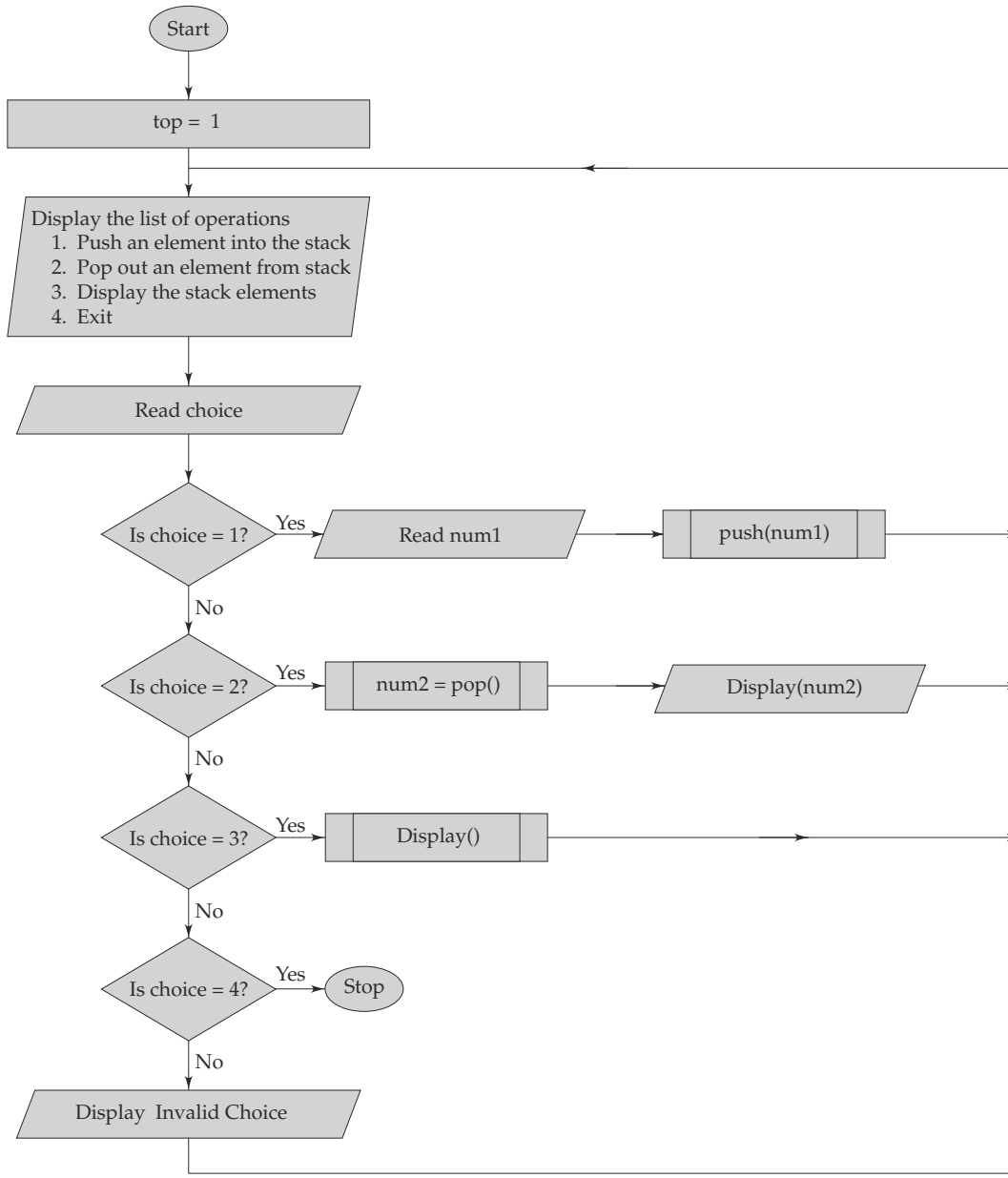
```
Step 1 – Start
Step 2 – Set i = 0
Step 3 – Repeat steps 4-5 while i<=top
Step 4 – Display stack[i]
Step 5 – i = i + 1
Step 6 – Stop
```

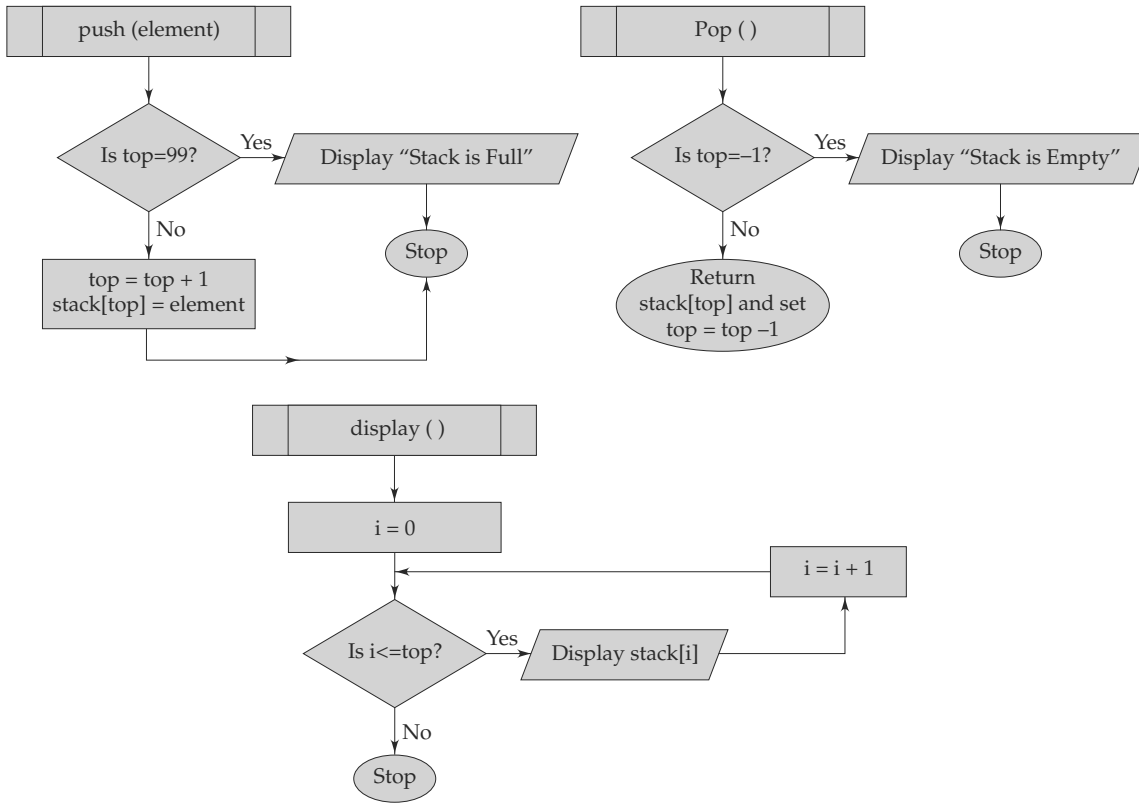
Flowchart

Program

```
#include <stdio.h>
#include <conio.h>

int stack[100];
int top=-1;

void push(int);
```

```
int pop();
void display();

void main()
{
  int choice;
```






```
char num1=0,num2=0;
while(1)
{
  clrscr();
  printf("Select a choice from the following:");
  printf("\n[1] Push an element into the stack");
  printf("\n[2] Pop out an element from the stack");
  printf("\n[3] Display the stack elements");
  printf("\n[4] Exit\n");
  printf("\n\tYour choice: ");
  scanf("%d",&choice);

  switch(choice)
  {

  case 1:
```

```
     {
      printf("\n\tEnter the element to be pushed into the stack: ");
      scanf("%d",&num1);
      push(num1);
      break;
     }

     case 2:
     {
     num2=pop();
     printf("\n\t%d element popped out of the stack\n\t",num2);
     getch();
     break;
     }

     case 3:
     {
     display();
     getch();
     break;
     }

     case 4:
     exit(1);
     break;

     default:
     printf("\nInvalid choice!\n");
     break;
     }
  }
}

void push(int element)
{
  if(top==99)
  {
  printf("Stack is Full.\n");
  getch();
  exit(1);
  }
```

```
  top=top+1;
  stack[top]=element;
}

int pop()
{
  int element;
  if(top==-1)
  {
  printf("\n\tStack is Empty.\n");
  getch();
  exit(1);
  }
  return(stack[top--]);
}

void display()
{
  int i;
  printf("\n\tThe various stack elements are:\n\t");
  for(i=0;i<=top;i++)
  printf("%d\t",stack[i]);
}
```

Output

```
Select a choice from the following:

[1] Push an element into the stack
[2] Pop out an element from the stack
[3] Display the stack elements
[4] Exit

       Your choice: 1

       Enter the element to be pushed into the stack: 42

Select a choice from the following:

[1] Push an element into the stack
[2] Pop out an element from the stack
[3] Display the stack elements
[4] Exit
```

```
       Your choice: 1

       Enter the element to be pushed into the stack: 2

Select a choice from the following:
[1] Push an element into the stack
[2] Pop out an element from the stack
[3] Display the stack elements
[4] Exit

       Your choice: 3

       The various stack elements are:
       42    2

Select a choice from the following:
[1] Push an element into the stack
[2] Pop out an element from the stack
[3] Display the stack elements
[4] Exit

       Your choice: 2

 2 element popped out of the stack

Select a choice from the following:
[1] Push an element into the stack
[2] Pop out an element from the stack
[3] Display the stack elements
[4] Exit

       Your choice: 4
```

EXAMPLE C.23 *Write a program to implement stack using pointers.*

Solution

Algorithm

```
Step 1 – Start
Step 2 – Define a structure to represent a stack
         STRUCTURE stack
         INTEGER element
         STRUCTURE stack *stptr
         END STRUCTURE
         STRUCTURE stack *top
Step 3 – Repeat Steps 4-X indefinitely
```

```
Step 4 – Display a list of stack operations for the user to choose from
         1. Push an element into the stack
         2. Pop out an element from the stack
         3. Display the stack elements
         4. Exit
Step 5 – Read the choice entered by the user (choice)
Step 6 – If choice = 1 goto Step 7 else goto Step 9
Step 7 – Read the element to be pushed (num1)
Step 8 – Call the push function, push(num1) and goto Step 3
Step 9 – If choice = 2 goto Step 10 else goto Step 12
Step 10 – Call the pop function, pop()
Step 11 – Display the popped element and goto Step 3
Step 12 – If choice = 3 goto Step 13 else goto Step 14
Step 13 – Call the display function, display() and goto Step 3
Step 14 – If choice = 4 goto Step 16 else goto Step 15
Step 15 – Display message "Invalid Choice" and goto Step 3
Step 16 – Stop
```

push(value)

```
Step 1 – Start
Step 2 – Reserve a block of memory of size stack and assign its address to pointer ptr,
         (ptr=(struct stack*)malloc(sizeof(struct stack)))
Step 3 – Set ptr → element = value
Step 4 – Set ptr→stptr=top
Step 5 – top = ptr
Step 6 – Return
```

pop()

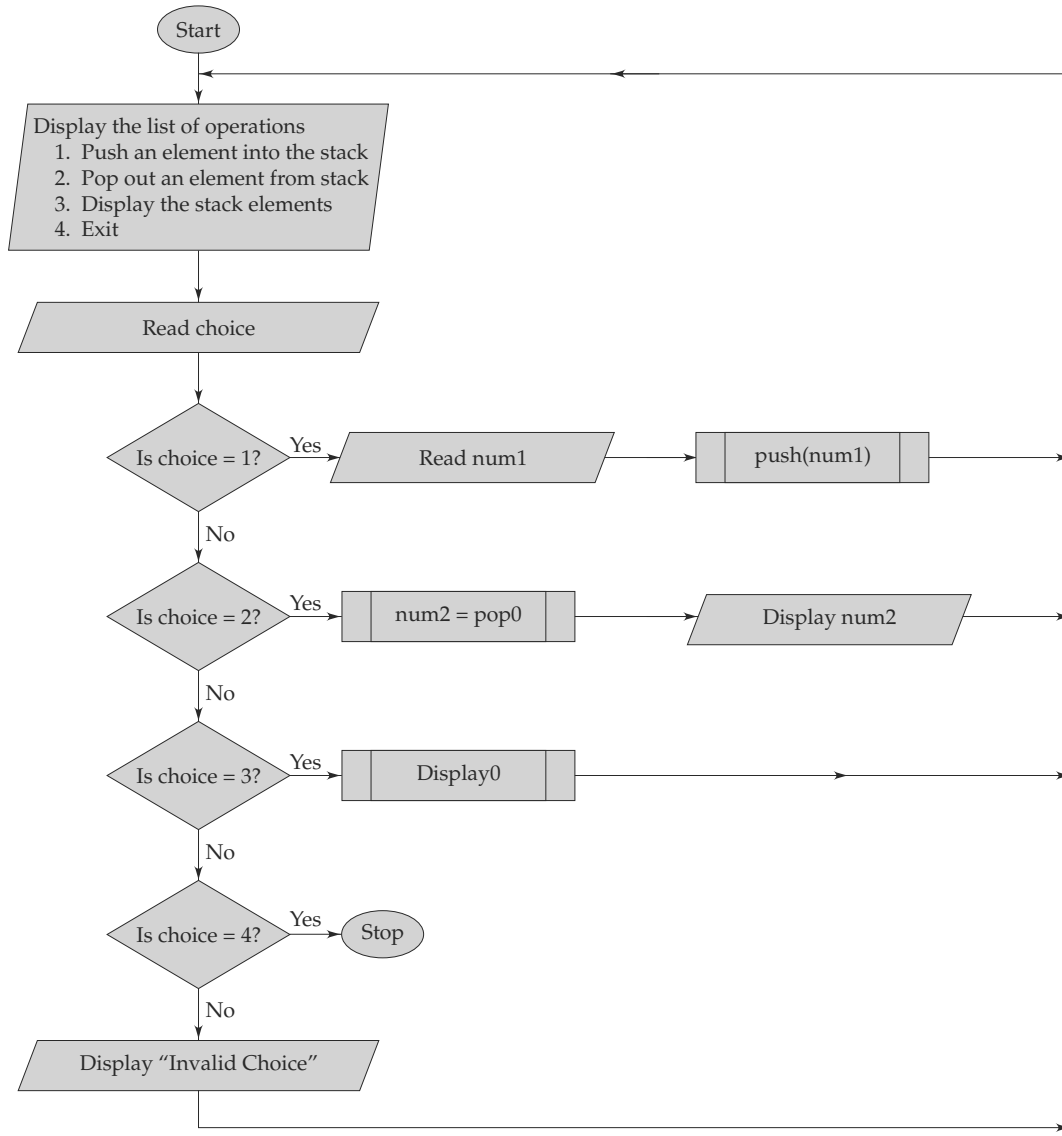
```
Step 1 – Start
Step 2 – If top = NULL goto Step 3 else goto Step 4
Step 3 – Display message "Stack Empty" and exit
Step 4 – Set temp=top→element
Step 5 – Set top=top→stptr
Step 6 - return (temp)
```

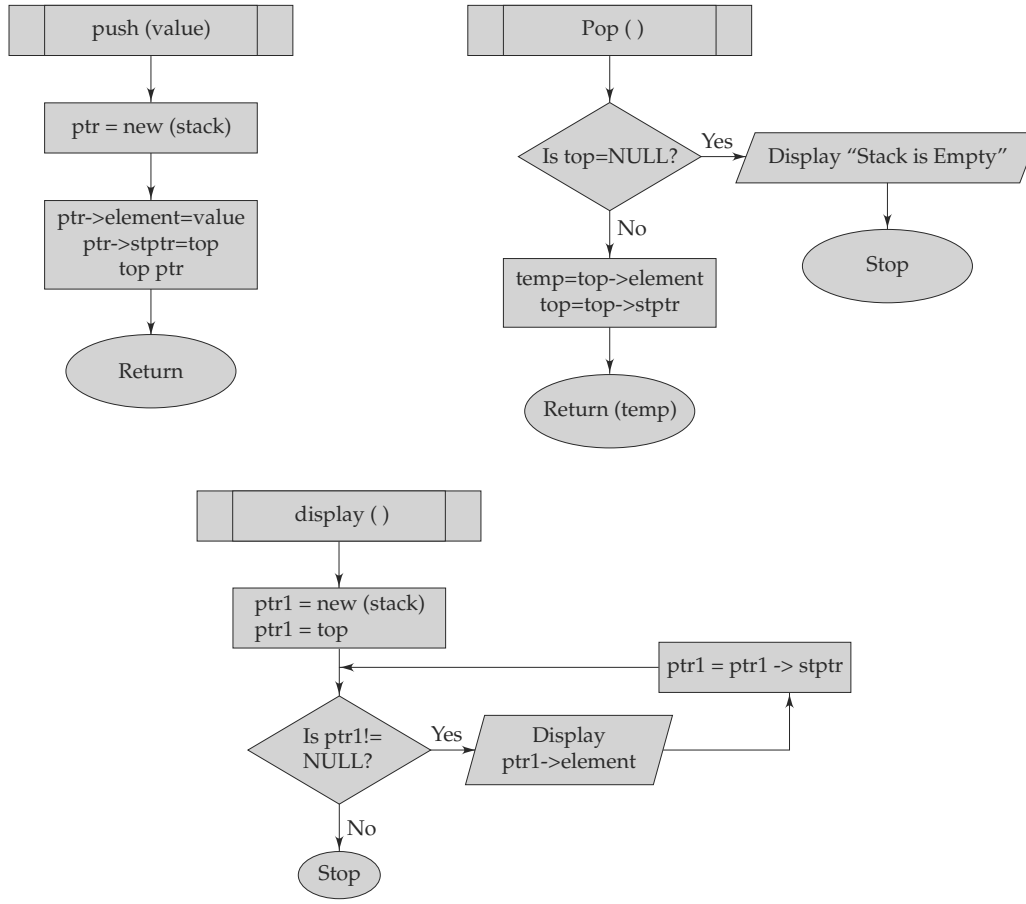
display()

```
Step 1 – Start
Step 2 – Create a pointer (ptr1) of type stack and assign it the value contained in top,
         (struct stack *ptr1=top)
Step 3 – Repeat steps 4-5 while ptr1!=NULL
```

```
Step 4 – Display ptr1→element
Step 5 – ptr1=ptr1→stptr
Step 6 – Stop
```

Flowchart





Program

```
#include <stdio.h>
#include <conio.h>

struct stack
{
  int element;
  struct stack *stptr;
}*top;

int i;

void push(int);
int pop();
```

```
void display();

void main()
{
  int choice;
  char num1=0,num2=0;
  while(1)
  {
    clrscr();
    printf("Select a choice from the following:");
    printf("\n[1] Push an element into the stack");
    printf("\n[2] Pop out an element from the stack");
    printf("\n[3] Display the stack elements");
    printf("\n[4] Exit\n");
    printf("\n\tYour choice: ");
    scanf("%d",&choice);

    switch(choice)
  {

    case 1:
    {
    printf("\n\tEnter the element to be pushed into the stack: ");
    scanf("%d",&num1);
    push(num1);
    break;
    }

    case 2:
    {
    num2=pop();
    printf("\n\t%d element popped out of the stack\n\t",num2);
    getch();
    break;
    }

    case 3:
    {
    display();
    getch();
```

```
    break;
    }

    case 4:
    exit(1);
    break;

    default:
    printf("\nInvalid choice!\n");
    break;
    }
  }
}

void push(int value)
{
  struct stack *ptr;
  ptr=(struct stack*)malloc(sizeof(struct stack));
  ptr->element=value;
  ptr->stptr=top;
  top=ptr;
  return;
}

int pop()
{
  if(top==NULL)
  {
  printf("\n\STACK is Empty.");
  getch();
  exit(1);
  }
  else
  {
    int temp=top->element;
    top=top->stptr;
    return (temp);
  }
}
```

```
void display()
{
  struct stack *ptr1=NULL;
  ptr1=top;
  printf("\nThe various stack elements are:\n");
  while(ptr1!=NULL)
  {
    printf("%d\t",ptr1->element);
    ptr1=ptr1->stptr;
  }
}
```

Output

```
Select a choice from the following:

[1] Push an element into the stack
[2] Pop out an element from the stack
[3] Display the stack elements
[4] Exit

       Your choice: 1

       Enter the element to be pushed into the stack: 66

Select a choice from the following:

[1] Push an element into the stack
[2] Pop out an element from the stack
[3] Display the stack elements
[4] Exit

       Your choice: 1

       Enter the element to be pushed into the stack: 33

Select a choice from the following:

[1] Push an element into the stack
[2] Pop out an element from the stack
[3] Display the stack elements
[4] Exit

       Your choice: 3

The various stack elements are:
33  66
```

```
Select a choice from the following:
[1] Push an element into the stack
[2] Pop out an element from the stack
[3] Display the stack elements
[4] Exit
       Your choice: 2
       33 element popped out of the stack
Select a choice from the following:
[1] Push an element into the stack
[2] Pop out an element from the stack
[3] Display the stack elements
[4] Exit
       Your choice: 4
```

EXAMPLE C.24 *Write a program that uses linear search technique to search an element in an array.*

Solution

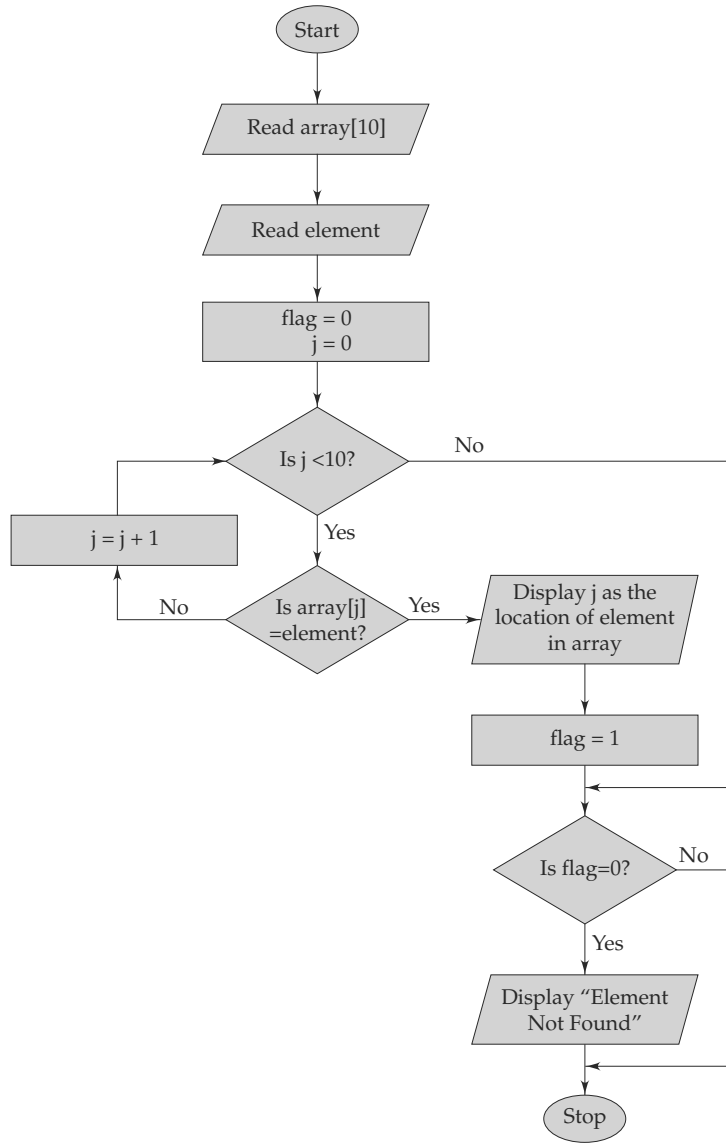
Algorithm

```
Step 1 – Start
Step 2 – Read a 10 element array (array[])
Step 3 – Read the element that needs to be searched (element)
Step 4 – Set flag = 0
Step 5 – Initialize the looping counter j = 0
Step 6 – Repeat Steps 7-9 while j<10
Step 7 – If array[j] = element goto Step 8 else goto Step 9
Step 8 – Display j as the location where element has been found, set flag = 1 and goto Step 10
Step 9 – Set j = j + 1
Step 10 – If flag = 0 goto Step 11 else goto Step 12
Step 11 – Display message "element not found in the array"
Step 12 – Stop
```

Flowchart

Program

```
#include <stdio.h>
#include <conio.h>
```




```
void main()
{
  int array[10], i, j, element;
  int flag=0;
  clrscr();

  printf("Enter the 10 elements of the list:\n");
```

```
  for(i=0;i<10;i++)
  scanf("%d",&array[i]);
  printf("\n\nEnter the element that you want to search: ");
  scanf("%d",&element);
  for(j=0;j<10;j++)
  if( array[j] == element)
  {
    printf("\nThe element %d is present at %d position in the list\n",element,j+1);
    flag=1;
    break;
  }
  if(flag==0)
    printf("\nThe element is %d is not present in the list\n",element);

getch();
}
```

Output

```
Enter the 10 elements of the list:
1
2
3
9
8
7
4
5
6
22

Enter the element that you want to search: 8

The element 8 is present at 5 position in the list
```

EXAMPLE C.25 *Write a program that uses binary search technique to search an element in an array.*

Solution

Algorithm

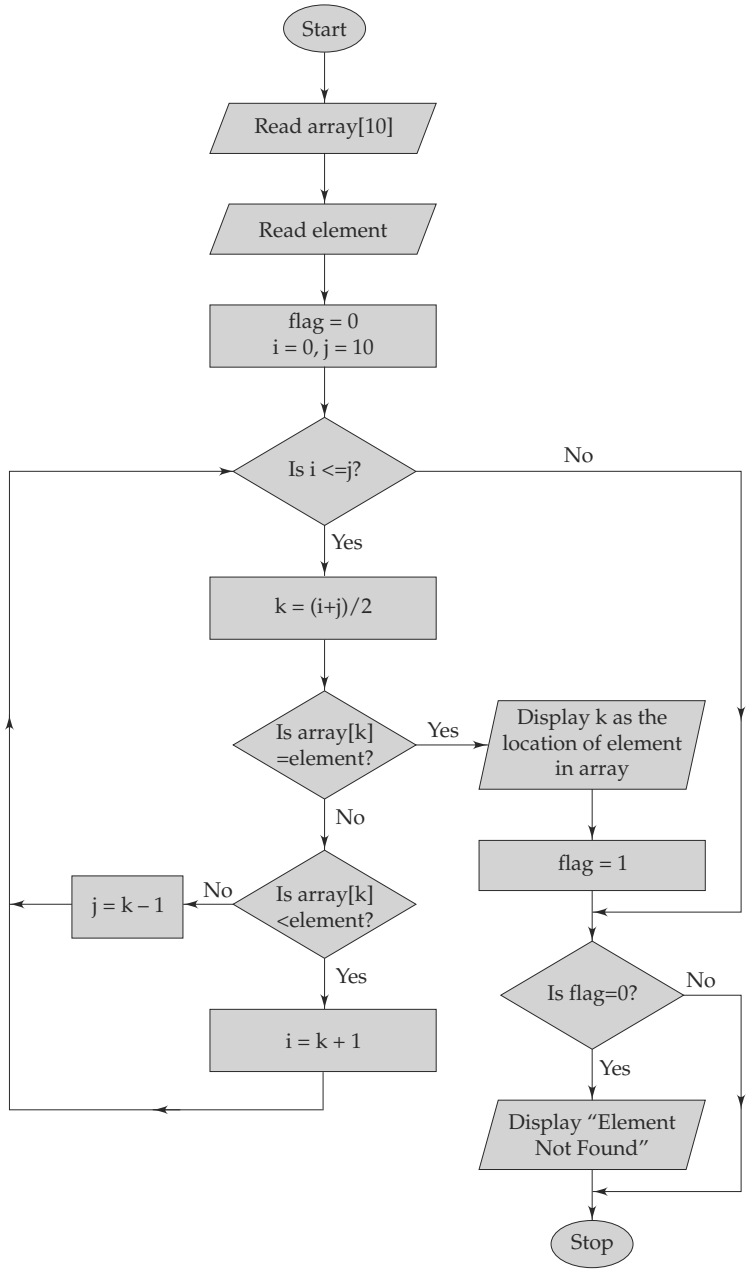
```
Step 1 – Start
Step 2 – Read a 10 element array (array[])
```

```
Step 3 – Read the element that needs to be searched (element)
Step 4 – Set flag = 0
Step 5 – Set i = o, j = 10
Step 6 – Repeat Steps 7-12 while i<=j
Step 7 – k = (i+j)/2
Step 8 – If array[k] = element goto Step 9 else goto Step 10
Step 9 – Display k+1 as the location where element has been found, set flag = 1 and goto Step 13
Step 10 – If array[k] < element goto Step 11 else goto Step 12
Step 11 – i = k + 1
Step 12 – j = k-1
Step 13 – If flag = 0 goto Step 14 else goto Step 15
Step 14 – Display message "Element not found"
Step 15 – Stop
```

Flowchart

Program

```
#include <stdio.h>
#include <conio.h>

void main()
{
  int array[10], i, j, k, element;
  int flag=0;
  clrscr();

  printf("Enter the 10 elements of the list in ascending order:\n");
  for(i=0;i<10;i++)
  scanf("%d",&array[i]);
  printf("\n\nEnter the element that you want to search: ");
  scanf("%d",&element);
  i = 0;
  j = 10;
  while(i <= j)
  {
    k = (i+j)/2;
    if(array[k] == element)
    {
    printf("\nThe element %d is present at %d position in the list\n",element,k+1);
      flag =1;
```




```
    break;
  }
  else
    if(array[k] < element)
```

```
            i = k+1;
         else
            j = k-1;
      }
      if( flag == 0)
      printf("\nThe element %d is not present in the list\n",element);

   getch();
   }
```

Output

```
Enter the 10 elements of the list in ascending order:
1
3
5
6
13
19
27
33
99
102
Enter the element that you want to search: 27
The element 27 is present at 7 position in the list
```

EXAMPLE C.26 *Write a program to solve the following series:*

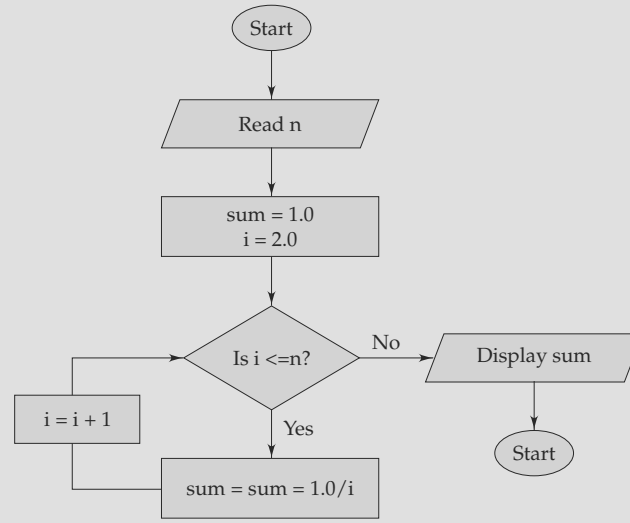
$$1 + 1/2 + 1/3 + 1/4 + \ldots + 1/n$$

Solution

Algorithm

```
Step 1 – Start
Step 2 – Read n
Step 3 – Set sum = 1.0
Step 4 – Set i = 2.0
Step 5 – Repeat Steps 6-7 while i<=n
Step 6 – sum = sum + 1.0/i
Step 7 – i = i + 1
Step 8 – Display sum as the resultant sum of the series
Step 9 – Stop
```

Flowchart



Program

```
#include <stdio.h>
#include <conio.h>

void main()
{
int n;
float i;
double sum;
clrscr();
printf("Enter the value of n: ");
scanf("%d",&n);
sum = 1.0;
for(i=2.0;i<=n;i++)
sum = sum + 1.0/i;
printf("\nThe sum of the series 1 + 1/2 + 1/3 +....+1/n = %.8lf",sum);
getch();
}
```

Output

```
Enter the value of n: 11

The sum of the series 1 + 1/2 + 1/3 + ... + 1/n = 3.01987734
```

EXAMPLE C.27 *Write a program in C to draw a circle.*

Solution

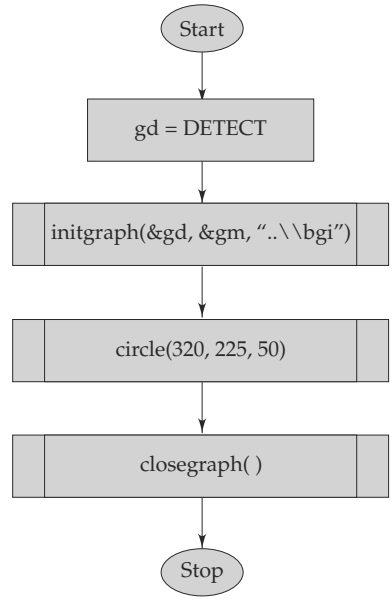
Algorithm

```
Step 1 – Start
Step 2 – Set gd = DETECT
Step 3 – Call in-build function, initgraph(&gd, &gm, "..\\bgi")
Step 4 – Call in-built function, circle(320, 225, 50)
Step 5 – closegraph()
Step 6 – Stop
```

Flowchart

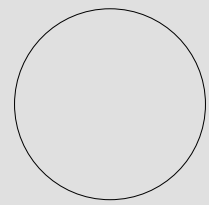
Program

```
#include<conio.h>
#include<graphics.h>
```




```
#include<stdio.h>

void main()
{
int gd = DETECT, gm;
initgraph(&gd, &gm, "..\\bgi");
circle(320, 225, 50);
getch();
closegraph();
}
```

Output



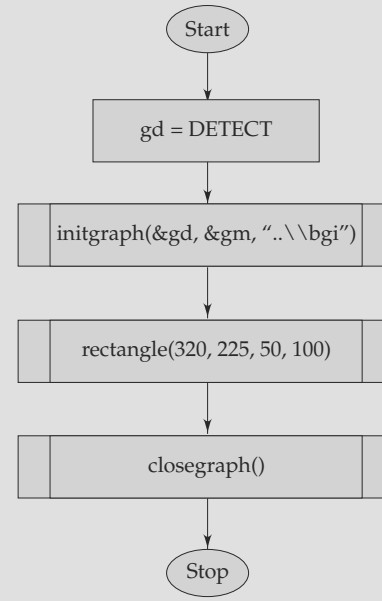
EXAMPLE C.28 *Write a program in C to draw a rectangle.*

Solution

Algorithm

```
Step 1 – Start
Step 2 – Set gd = DETECT
Step 3 – Call in-build function, initgraph(&gd, &gm, "..\\bgi")
Step 4 – Call in-built function, rectangle(320, 225, 50,100)
Step 5 – closegraph()
Step 6 – Stop
```

Flowchart



Program

```
#include<conio.h>
#include<graphics.h>
#include<stdio.h>

void main()
{
int gd = DETECT, gm;
initgraph(&gd, &gm, "..\\bgi");
rectangle(320, 225, 50,100);
getch();
closegraph();
}
```

Output

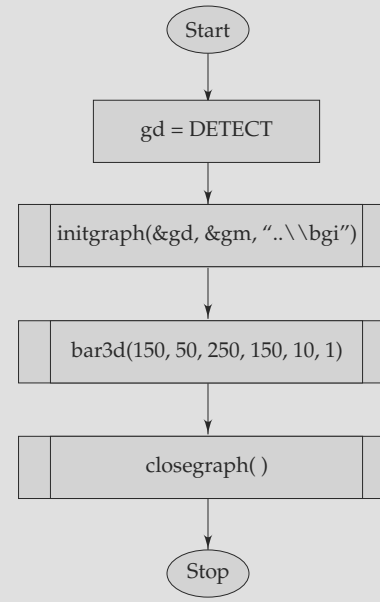
EXAMPLE C.29 *Write a program in C to draw a 3D-bar.*

Solution

Algorithm

```
Step 1 – Start
Step 2 – Set gd = DETECT
Step 3 – Call in-build function, initgraph(&gd, &gm, "..\\bgi")
Step 4 – Call in-built function, bar3d(150, 50, 250,150, 10, 1)
Step 5 – closegraph()
Step 6 – Stop
```

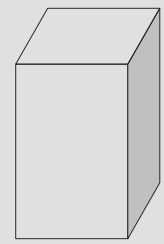
Flowchart



Program

```
#include<conio.h>
#include<graphics.h>
#include<stdio.h>

void main()
{
int gd = DETECT, gm;
initgraph(&gd, &gm, "..\\bgi");
bar3d(150, 50, 250,150, 10, 1);
getch();
closegraph();
}
```

Output



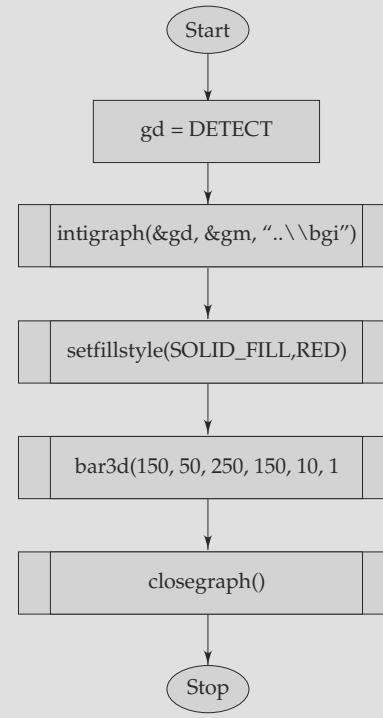
EXAMPLE C.30 *Write a program in C to draw a shape and fill it with color.*

Solution

Algorithm

```
Step 1 – Start
```

```
Step 2 – Set gd = DETECT
Step 3 – Call in-build function, initgraph(&gd, &gm, "..\\bgi")
Step 4 – Call in-build function, setfillstyle(SOLID_FILL,RED)
Step 4 – Call in-built function, bar3d(150, 50, 250,150, 10, 1)
Step 5 – closegraph()
Step 6 – Stop
```

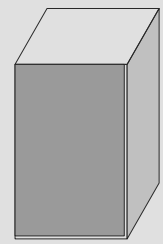


Flowchart

Program

```
#include<conio.h>
#include<graphics.h>
#include<stdio.h>

void main()
{
int gd = DETECT, gm;
initgraph(&gd, &gm, "..\\bgi");
setfillstyle(SOLID_FILL,RED);
bar3d(150, 50, 250,150, 10, 1);
getch();
closegraph();
}
```

Output



INDEX