The McGraw-Hill Companies

Test Your Skills in C

Second Edition

S THAMARAI SELVI
R MURUGESAN

The McGraw-Hill Companies

Test Your Skills in C

Second Edition

S. Thamarai Selvi
Professor & Head
Department of Information Technology
MIT Campus, Anna University, Chennai
Tamil Nadu

R. Murugesan
Reader
Department of Mathematics
Thiruvalluvar College, Vickramasingapuram
Tamil Nadu

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

The McGraw·Hill Companies

Tata McGraw Hill

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

ISBN (13 digits) : 978-0-07-0145856
ISBN (10 digits) : 0-07-0145857

Managing Director: *Ajay Shukla*

General Manager: Publishing—SEM & Tech Ed: *Vibha Mahajan*
Manager—Sponsoring: *Shalini Jha*
Associate Sponsoring Editor: *Nilanjan Chakravarty*
Development Editor: *Surbhi Suman*
Junior Executive—Editorial Services: *Dipika Dey*
Junior Manager—Production: *Anjali Razdan*

General Manager: Marketing—Higher Education: *Michael J Cruz*
Senior Product Manager: SEM & Tech Ed: *Biju Ganesan*

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

Typeset at Print-O-World, 2579, Mandir Lane, Shadipur, New Delhi 110 008, and printed at Ram Book Binding, C-114, Okhla Industrial Area, Phase-I, New Delhi-110020

Cover : SDR Printers

RALCRRAFRCQZC

The McGraw Hill Companies

Contents

The McGraw·Hill Companies

Preface to the Second Edition

The revised edition of Test your Skills in C based on the ANSI C standards, is a complete self-taught book with focus on the 'C' language. This book refreshes C programming knowledge of readers in short span, thereby equipping them to thoroughly prepare for various screening tests and campus interviews. The new edition retains its appeal as a handy text to students as well as a guide for aspiring IT professionals.

C is a structured programming language developed by Dennis Ritchie at AT&T Bell Laboratories in New Jersey. It has excellent support for high-level as well as low-level functionality, which makes it suitable for many applications. Since knowledge of C is essential, it has been introduced as a core subject in various engineering disciplines and as an elective subject in other disciplines. *This book is well suited for undergraduate (final year) students of CSE, IT, ECE, EEE, Electronics & Instrumentation, BCA, MCA, DOEACC courses, and B.Sc/M.Sc (Computer Science/lT).*

Due to the growing influence of IT in several fields, C programming skills are highly sought after. Invariably, all aptitude tests in the recruitment process include testing C Skills. *This book will be an excellent guide for aspiring IT professionals and a valuable resource bank, for IT companies, to frame entrance tests and technical interviews.*

New to the Edition

The book has been revised and restructured based on the feedback received from readers. A separate chapter 'Strings' has been included along with coverage of string function implementation details. Additional programming examples covering important data structures have been included—these programming skills will be helpful in solving puzzles asked in technical interviews.

Salient features of the book

The book has been organised to cover C concepts using a structured format—each chapter has an introduction giving an overview followed by short answer type questions and objective questions. Additionally, 'Fill in the blanks' and 'Match the following' enable better comprehension of key concepts.

- Practice questions covering C language concepts in eleven chapters.
- Special section titled 'Crack the Tough Nuts' containing unique and thought-provoking programming exercises. Programming tips and solutions are provided for these exercises.
- Special notes and tips for tricky questions.
- Concepts are introduced to readers with brief theory/definition with a simple program example.
- Syntax along with the concepts has been highlighted or provided.
- More than 50 additional programming examples covering C programming and data structure concepts included in a separate chapter will help the readers in enhancing their programming skills.
- Four indispensable appendices on ASCII table, Operator Precedence Table, Timings of basic C operations and ANSI C functions.

- **Pedagogy refreshed and increased.**
  - Definitions - 59
  - Short-answer question (with answers) - 410
  - Fill in the blanks - 131
  - True or false - 138
  - Match the following - 55
  - Objective type questions - 766

Acknowledgement

We sincerely thank our students, M Raghavan, Divya Venkataraman, R Parthiban and R Sharadha for verification of the programming examples. We thank our daughters, M Poorani and M Brindha for their love and understanding during the revision of the book. We extend our appreciation to Tata McGraw-Hill Education editorial and production team for their enthusiastic support and guidance. In particular, Vibha Mahajan encouraged us to revise the project and set the publication process in motion, Nilanjan Chakravarty gave useful suggestions and Surbhi Suman helped us in updating the book to enrich its content.

A note of acknowledgement is due to the following reviewers for their valuable suggestions.

Kalpana Sharma
Sikkim Manipal Institue of Technology, Sikkim

N Lalitha
DJ Academy of Managerial Excellence, Coimbatore.

We would be grateful if readers send their valuable suggestions for further improvement of the book at the following email id—tmh.csefeedback@gmail.com (kindly mention the title and author name in the subject line).

S Thamarai Selvi
R Murugesan

The McGraw·Hill Companies

Preface to the First Edition

C is a popular structured programming language. It is a general purpose programming language with many powerful features such as pointers. Because of its portability, it is used for developing software in many applications. It is especially suitable for system programming.

This book is meant to help the readers test their skills in C. Basic understanding of C programming is a prerequisite to use this book. The book will help the reader in further analyzing and understanding the concepts of C. This book is organized to contain questions with short answers, fill in the blanks, true or false and objective type questions. Every care has been taken to ensure that the short answer type questions cover all the concepts of C language, including the frequently asked questions (FAQs). The objective type questions are given in such a way that the reader can analyze the different possibilities carefully and then select the correct answer among the given choices. Solving these questions will help the readers judge their strengths and weaknesses in C.

Most of the questions are based on programming practice, which will help the readers debug C programs and write correct and efficient codes. The questions are selected in such a way that the reader is prepared to face any technical written test in C confidently.

To enable the reader acquire knowledge step by step, this book is organized in ten chapters systematically. For each chapter, the questions are given separately and the entire C language is covered.

As a practice for written tests, two model tests have been included in Chapter 11. By attempting these tests, the readers can improve their speed in solving the problems. Chapter 12 **Crack the Tough Nuts** will enable the readers to develop logical thinking in solving challenging problems and puzzles.

ASCII table and operator precedence table are provided in Appendices A and B respectively to help the reader in solving problems. Appendix C provides the timing of basic C operations. Most of the ANSI C library functions are given in Appendix D for the reader's ready reference. We hope that the readers, be it students, teachers or even professionals, will find this book extremely useful.

The reader may refer to our book titled *C for All* to learn C programming.

We would like to express our sincere thanks to all the people involved, directly or indirectly, in bringing out this book.

S Thamarai Selvi
R Murugesan

The McGraw·Hill Companies

Elements of C Language

1

C was developed by Dennis Ritchie in 1972. This chapter discusses the basic elements of C with the help of short answer type questions and objective type questions. C has got its own character set and reserved words. It has four basic data types, viz. **int, char, float** and **double**, and derived data types. Qualifiers such as short, long, signed and unsigned may precede basic data types to specify memory representation of a data type. C supports numeric constants, character constants and string constants. Symbolic constants are used to give names for constants. An escape sequence is a special character starting with a backslash where the normal interpretation of the character following the backslash is made to escape; it can be represented in a character constant or in a string constant. In C, variable is declared before use and can be initialised in the declaration itself. Comments can also be included in a program to improve the readability of the program. Each individual unit in a C program is called a token. A separator is used to separate tokens, whereas, a semicolon is used as a statement terminator.

Some of the important definitions related to the programming languages are defined here:

IDENTIFIER

An identifier is a symbolic name used to refer to an entity such as a data type, constant, variable, function, array, etc. in a program. It is a sequence of characters starting with an alphabet or underscore that may be followed by alphanumeric characters and underscores.

VARIABLES AND CONSTANTS

A variable is an identifier used for storing data in a program. The value stored in a variable may be changed during the execution of a program. A constant is a fixed value directly used in any program and it is unchanged during the execution of the program.

SCALAR DATA TYPE

A scalar data type is used for representing a single value only. For example, int, char, float and double.

DERIVED DATA TYPE

Derived data types are derived from the scalar data types with additional relationships between the various elements of that scalar data type. They are also known as structured data types. Derived data type may be used for representing a single value or multiple values.

ESCAPE SEQUENCE

An escape sequence is used to escape from the normal meaning of a character in a C program. It is used for editing non-graphic characters in a program. It is a character representation that may appear in a character constant or in a string constant. For example, \n \t \a \" \\.

TOKEN

A token is an individual entity of a program. A compiler identifies and splits a program into a number of tokens. It always constitutes the largest possible token.

ANSI C STANDARD

American National Standards Institute (ANSI) established a committee X3J 11 in 1983 to standardize C language. The committee's work was finally ratified as ANSI X3.159-1989 on 14th December 1989 and published in 1990. The ANSI C standard also formalizes the C run-time library support routines. The standard has been adopted as an international standard ISO/IEC 9899:1990 and this replaces the earlier X3.159. Now a new Standard is developed which is nicknamed C9X. It is generally known as C99 because ISO standard was formally adopted in the year 1999. The previous standard ratified in 1989 is called C89.

SHORT ANSWER QUESTIONS

1.1 Who developed C language?

Dennis Ritchie at AT&T Bell Laboratory, Murray Hill, New Jersey developed C in 1972.

1.2 Why is C named so?

C is named so to present it as the successor of B language.

1.3 Who developed B language?

Ken Thompson in 1970 developed B for the first UNIX system on the DEC PDP-7 computer.

1.4 Give the salient features of C.

C is a general-purpose, free format and structured programming language. It has a rich set of operators, and uses more control structures. Memory addresses are directly accessed by using pointers. It is quite suitable for system programming. It is a flexible and powerful language. It is a fast running, machine independent and efficient language.

1.5 What are the limitations of C?

There is no uniformity in associativity and no direct I/O facility.

1.6 Name any five C compilers.

1. Turbo C 2. Microsoft C 3. Quick C
4. Lattice C 5. Power C

1.7 Give the character set of C.

Alphabets	:	A to Z and a to z
Digits	:	0 to 9
Special characters	:	+ – * / % = < > _ blank : ; , . ' " ? ! # \ ( ) [ ] { } & | ^ ~

1.8. Name any four reserved words.

ANSI C keywords are listed below.
(any four keywords may be selected).

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

1.9 What is the purpose of a variable and a constant in a programming language?

The data are represented using variables or constants in a programming language.

1.10 Name the scalar data types in C.

int, char, float, and double.

1.11 Name the derived data types in C.

Arrays, functions, pointers, structures, and unions.

1.12 What are qualifiers in C?

Qualifiers or modifiers are identifiers that may precede the scalar data types (except float) to specify the number of bits used for representing the respective type of data in memory. The qualifiers in C are **short, long, signed,** and **unsigned**.

1.13 Name the type specifiers in C.

char, int, float, double, short, long, signed, and unsigned.

1.14 Name the type of constants.

Integer constants, single and double precision constants.

1.15 How are octal and hexa decimal constants represented?

Octal constants are preceded by zero and they are formed by using digits 0 to 7. For example, 053 and 065.
Hexa constants are preceded by Ox or OX. They are formed by using digits 0 to 9 and characters A to F or a to f. For example, Ox532, OX6.26, and Oxalc3.

1.16 How is a character constant represented?

A character constant is written within single quotes. For example, 'K', '\', and '5'

1.17 How is a string constant represented?

A string constant is a sequence of zero or more characters enclosed within double quotes. For example, "X", "626", "", and "AREA"

1.18 List the printable escape sequences.

\\, \", \' > and \?

1.19 List the non-graphic escape sequences.

\0 null character
\a alert
\b backspace
\t horizontal tab

\n new line
\v vertical tab
\f form feed
\r carriage return

1.20 What is a symbolic constant?

If a constant is given a name, it becomes a symbolic constant or manifest. For example,

```
#define MAX 20
```

defines the symbolic constant MAX to represent the constant value 20.

1.21 How is a variable declared?

Format:

```
data_type varl, var2, var3 ,..., varN;
e.g., int a, b; float x;
```

1.22 What is initialization of a variable?

If a variable is assigned a value in the declaration itself, it is known as initialization. For example, `int x = 10, y = 5;`

1.23 How are comments included in a C program?

Single line or multiline comments are included between /* and */. Nested comments are not allowed in ANSI C. For example,
/* FINDING AREA OF A TRIANGLE */

1.24 List white space characters in C.

Blank space, new line, horizontal tab, vertical tab, carriage return and form feed are white space characters in C.

1.25 What is a statement terminator? What is a separator?

Each statement is ended with a semicolon, so a semicolon is a statement terminator, whereas, a white space character or a comment is a separator used to separate tokens.

1.26 List the tokens in C.

Identifiers, keywords, constants, string constants, operators and separators are C tokens.

1.27 Identify the tokens in the expression **if(mark>=50).**

Here:
the key word **if** is a token,
the character (is the next token,
the variable **mark** is the subsequent token,
the operator **>=** is the next token (largest possible token),
the constant **50** is the next token, and
the character) is the next token.

1.28 What are Ivalue and rvalue?

Each variable has got two values, which are left value and right value whose short names are **lvalue** and **rvalue**. The **lvalue** denotes an object that is the address of the data object. The **rvalue** is the value residing in that address. An lvalue represents a memory location where a value may be stored; it denotes an object that refers to a storage location of the data object. The term lvalue is used, based on the principle of its appearance, on the left side of an assignment statement. Consider the following statements:

```
int x,y;
```

```
x = 5;
y = x ;
```

In the first assignment statement, x is an lvalue. Hence, x is treated as a name for a particular memory location and the value 5 is stored in that memory location. In the second assignment statement, x is not an Ivalue and hence the value stored in the memory location is referred to, by x. Thus, in some statements an operand must be an Ivalue. If an lvalue appears in any other context, it is replaced by the value stored in that memory location.

1.29 What is the value of sizeof (char)?

The value returned by sizeof (char) is always 1 since char uses only one byte in any machine. For other data types, the number of bytes used to represent a data type depends on the implementation of a compiler.

1.30 What is the current version of C?

C99 is the current version of C. The character set is changed by supporting Unicode characters in this version. It includes additional data types, such as **bool, complex, variable-length arrays, and variable structure members** and some more new syntax also. Since there is a substantial change, most compilers have not yet implemented C99. The specification of new C99 may be obtained from ISO web site.

FILL IN THE BLANKS

1. C is a _____ programming language.
2. C supports ____ number of primitive data types.
3. A _____ may precede a data type in a data type declaration.
4. A hexadecimal constant is preceded by _____.
5. Null string is represented by _____.
6. A constant is given a name using a _____.
7. Variables are also known as _____ in C.
8. Nested comments are _____ in ANSI C.
9. A compiler splits a program into a number of _____.
10. Assigning a value to a variable in a declaration, is known as _____ .
    The value used for the assignment, is known as _____ .
11. A variable declaration fixes_____and_____of the variables.
12. C99 introduced support for characters from _____.

TRUE OR FALSE

1. C is not an object oriented programming language.
2. C allows long float type.

3. The first character of an identifier may be an undescore.
4. A declaration without any variable name is also a valid declaration.
5. Blank space is a white space character.
6. Octal constant is preceded by the alphabet o or O.
7. The data type char is an integral data type.
8. The following manifest is a valid one.
   #define PI 3.141592 ROW 10
9. The internal representation of a string constant includes NUL character at the end.
10. C99 supports boolean data type.
11. The constant 243 is equal to the constant 0243.
12. The constant 048 is a valid constant.

MATCH THE FOLLOWING

1	Escape sequence	Individual unit
2	Qualifier	Value changes during execution
3	Token	Value does not change during execution
4	Constant	Precedes a datatype
5	Object	Character representation

OBJECTIVE TYPE QUESTIONS

1.1 Who developed the C language?
(a) Ken Thompson (b) Bjarne Stroutstrup (c) Dennis Ritchie (d) Kernighan

1.2 When was the C language developed?
(a) 1970 (b) 1972 (c) 1975 (d) 1976

1.3 Where was the C language developed?
(a) Microsoft Corporation (b) Sun Microsystem
(c) AT&T Bell Laboratory (d) CERN, European Particle Physics Laboratory

1.4 Who developed the language B?
(a) Pascal (b) Bjarne Stroutstrup (c) Kernighan (d) Ken Thompson

1.5 Which language was the predecessor of C?
(a) A (b) BCPL (c) B (d) CPL

1.6 Which is not a character of C?
(a) $ (b) ^ (c) ~ (d) |

1.7 An identifier cannot start with
(a) _ (b) uppercase alphabet (c) lowercase alphabet (d) #

1.8 Which is not a keyword in C?
(a) const (b) main (c) sizeof (d) void

1.9 Identify the scalar data type in C.
(a) double (b) union (c) function (d) array

1.10 Identify the derived data type in C.
(a) int (b) float (c) union (d) char

1.11 Which data type can either be used to represent a scalar data type or derived data type?
(a) pointer (b) double (c) structure (d) union

1.12 The qualifier that may precede float is
(a) signed (b) unsigned (c) long (d) none of the above

1.13 The qualifier that may precede char is
(a) signed (b) unsigned (c) options a and b (d) none of the above

1.14 The qualifier that may precede double is
(a) signed (b) unsigned (c) short (d) long

1.15 Printable characters always use
(a) negative integers (b) positive integers
(c) both positive and negative integers (d) –1

1.16 Integral data type includes
(a) enum (b) int (c) char (d) all the above

1.17 Which is not a valid integer constant ?
(a) 600000 u (b) 534878 ul (c) 0Xabpq (d) 0X625

1.18 Which is not a valid floating constant?
(a) 4E-6f (b) 4E 12 (c) 0.08e–4 (d) 1.3345F

1.19 Identify the Octal constant
(a) 627 (b) OX25 (c) –0756 (d) 06.52

1.20 An octal constant is preceded by
(a) X (b) OX (c) 0 (alphabet) (d) 0 (zero)

1.21 A hexa constant is preceded by
(a) OX (b) 0 (c) HX (d) 0

1.22 A character constant is written within
(a) double quotes (b) single quotes
(c) options a and b (d) none of the above

1.23 String constants are represented within
(a) single quotes (b) double quotes
(c) options a and b (d) * and *\

1.24 Identify the invalid string constant.
(a) “A + B” (b) “ ” (c) “ ‘ “ (d) ‘A’

1.25 Which is not a character constant?
(a) '\60' (b) '\012' (c) '\x24' (d) 'sum'

1.26 Identify the invalid constant.
(a) " " (b) ' ' (c) ' in ' (d) ' \b'

1.27 Identify the escape sequence(s).
(a) \0 (b) \n (c) \f (d) all the above

1.28 Which is not a white space character?
(a) \f (b) \v (c) \0 (d) blank

1.29 Identify the white space character (s).
(a) blank (b) \f (c) \r (d) all the above

1.30 Identify the invalid string constant.
(a) "5\t 10\t 15 \t" (b) "cost = 90\x24\n"
(c) "\n Don't care condition" (d) ' C is flexible'

1.31 If a constant is given a name it becomes
(a) a string constant (b) manifest (c) const declaration (d) invalid

1.32 Symbolic constants are defined as
(a) # define S1 S2 (b) # define S1 S2;
(c) # define S1 = S2 (d) # define S1 = S2;

1.33 The symbol # in the # define statement must commence from
(a) anywhere in a line (b) the first column of a line
(c) the first column of next line (d) the first line

1.34 What is an object in C?
(a) constant (b) variable (c) identifier (d) keyword

1.35 Identify the C token(s).
(a) keywords (b) constants (c) operators (d) all the above

1.36 Statement terminator is represented by
(a) : (b) blank (c) ; (d) \n

1.37 Which is true in case of ANSI C?
(a) Comments are represented in between /*and */.
(b) Nested comments are not allowed.
(c) Nested comments are allowed.
(d) Comments are not allowed within a string constant.

1.38 Identify the correct statement(s) related to token.
(a) Token is an individual entity of a program.
(b) Tokens are identified during the compilation.
(c) Compiler always constitutes the longest possible token.
(d) All the above.

1.39 Identify the number of tokens in the following: **if(age = 21)**
(a) 4 (b) 5 (c) 6 (d) 7

1.40 Identify the separator(s) in C.

(a) white space character (b) comment
(c) options a and b (d) semicolon

1.41 Identify the wrong statement.

(a) # define /*symbolic constant */MAX 100 (b) int/*declaration*/a,b;
(c) char cl, c2; (d) # define MAX 25;

1.42 Which is an invalid variable name?

(a) Int (b) Xx (c) net-salary (d) floating

1.43 Identify the correct statement.

(a) The variable names VOLUME and volume are identical.
(b) The variable names Sum and sum are identical.
(c) Variables are not declared before use.
(d) Variable names may be absent in a declaration.

1.44 Identify the invalid identifier.

(a) NET_ $ (b) BINGO (c) _ACCOUNT (d) _ 4

1.45 Which is the most appropriate variable initialization?

(a) # define MAX 100 (b) int x, y; y = 15;
(c) float y = 2.14; (d) char c; c =`O';

1.46 Tokens are separated by using

(a) : (b) . (c) ; (d) separator

1.47 An escape sequence commences with

(a) \ (b) / (c) ? (d) #

1.48 Identify the wrong declaration.

(a) int n = {7}; (b) char c2 = 'A' + 25, cl = 'Z';
(c) int x ; y ; (d) int x = 10, y = x * 20, year;

1.49 Identify the wrong statement.

(a) unsigned long int yl, y2; (b) long float fl;
(c) long double ld; (d) signed n;

1.50 Identify the wrong sentence.

(a) # define is a preprocessor facility.
(b) # define aids in modifying a constant value throughout the program.
(c) # define uses a statement terminator.
(d) # define improves the readability of the program.

1.51 1. A definition allocates memory space for a variable.
2. A declaration allocates memory space for a variable.
3. A definition can occur multiple times.
4. A declaration can occur multiple times.

Regarding the difference between a declaration and a definition of a variable, which of the above statements are true?

(a) options 1 and 3 only (b) options 1 and 4 only
(c) options 2 and 3 only (d) options 2 and 4 only

1.52 What is a variable declaration?

(a) The assignment of properties to a variable.
(b) The assignment of memory space to a variable.
(c) The assignment of properties and memory space to a variable.
(d) The assignment of properties and identification to a variable.

1.53 What are strings?

(a) They are contiguous blocks of characters and a terminating NUL.
(b) They are individual characters linked together to form a string.
(c) Both options (a) and (b).
(d) None of the above.

1.54 In addition to the length of the string, what must be considered while determining the minimum number of characters required to store a C string?

(a) An EOF at the end must be considered.
(b) A '\0' at the end must be considered.
(c) Apart from the string itself, there is nothing else to be considered when determining the length of the string.
(d) A '\0\n' at the end must be considered.

1.55 Where does the execution of every C program starts?

(a) Every C program starts in the main () function.
(b) Every C program starts in the begin () function.
(c) Every C program starts in the initialize () function.
(d) Every C program starts in the start () function.

1.56 Regarding real values in C, which of the following is TRUE?

1. A float occupies less memory than a double.
2. The range of real numbers that can be represented by a double is less than those represented by a float.

(a) only option I (b) only option 2 (c) both options 1 and 2 (d) Neither option 1 nor 2

1.57 What is the maximum value of a signed data type that is 8 bits in size?

(a) 2 to the power of 7 (b) 2 to the power of 8
(c) (2 to the power of 7) minus 1 (d) (2 to the power of 8) minus 1

1.58 What is the largest value an integer can hold in an ANSI C compiler?

(a) 65536 (b) 2147483647 (c) INT_ MAX (d) 1« INT_BITS

1.59 Among the following, which escape sequence does not have any specific meaning?

(a) ' \t' (b) '\a' (c) ' \b' (d) '\c'

1.60 The escape sequence character '\x07' is equivalent to the character.

(a) '\a' (b) '\b' (c) '\r' (d) ' \f '

1.61 Which of these is an invalid identifier?

(a) wd-count (b) wd_ count (c) w4count (d) wdcountabcd

1.62 In a compiler there are 36 bits for a word and to store a character, 8 bits are needed. In this to store a character two words are appended. Then for storing k characters string, how many words are needed?

(a) 2k/9 (b) (2k+8)/9 (c) (k+8)/9 (d) 2*(k+8)/9 (e) none

ANSWERS

Fill in the Blanks

1. structured 2. four 3. qualifier 4. Ox or OX
5. " " 6. manifest 7. objects 8. not allowed
9. tokens 10. initialization, initializer 11. name, data type
12. unicode character set

True or False

1. True 2. False 3. True 4. True 5. True 6. False
7. True 8. False 9. True 10. True 11. False 12. False

Match the Following

1. 5 2. 4 3. 1 4. 3 5. 2

Objective Type Questions

1.c	11.a	21.a	31.b	41.d	51.b	61.a
2.b	12.d	22.b	32.a	42.c	52.d	62.a
3.c	13.c	23.b	33.a	43.d	53.c	
4.d	14.d	24.d	34.b	44.a	54.b	
5.c	15.b	25.d	35.d	45.c	55.a	
6.a	16.d	26.c	36.c	46.d	56.a	
7.d	17.c	27.d	37.b	47.a	57.c	
8.b	18.b	28.c	38.d	48.c	58.c	
9.a	19.c	29.d	39.c	49.b	59.d	
10.c	20.d	30.d	40.c	50.c	60.a	

C Operators and Expressions

2

C is operators-rich programming language. It supports arithmetic, relational, equality and logical operators like other programming languages. C also supports auto increment and auto decrement operators for faster execution. Bitwise operators are also supported in C, which are not available in other structured languages. In addition to simple assignments, C provides compound assignments, which is a shorthand notation for some of the operations performed on a variable, this is the lvalue in an assignment. Expressions are formulated with the help of operators and are evaluated to get the desired result. In evaluating mixed mode expressions, implicit type conversion rules are followed. Explicit type conversions are possible with coercion or type casting. Expressions are evaluated according to the precedence levels of the operators and their associativity. Important definitions are given next.

OPERATOR AND OPERAND

An operator is a symbol used to manipulate the data. The data items that the operators act upon are called operands. In a + b, + is an operator and a and b are operands.

EXPRESSION

A valid combination of constants, variables and operators constitutes an expression.

TYPE CONVERSION

Converting data type of one operand to the data type of another operand in an expression is known as type conversion. Type conversion can be achieved by either implicit type conversion or explicit type conversion.

OPERATOR PRECEDENCE LEVEL

When several operators appear in one expression, evaluation takes place according to certain predefined rules called hierarchy rules. These rules specify the order of evaluation called precedence level or priority of operators.

ASSOCIATIVITY

Associativity refers to the order in which a language evaluates the operations involving more than one operator having the same precedence level in an expression.

SHORT ANSWER QUESTIONS

2.1 Classify the types of operators based on the number of operands.

Unary : The operator that uses a single operand is a unary operator.
Binary : The operator that uses two operands is a binary operator.
Ternary : The operator that uses three operands is a ternary operator.

2.2 List the types of operators supported in C.

Arithmetic, relational, equality, logical, bitwise, assignment and type conversion.

2.3 Give the symbols of arithmetic operators.

+	addition
–	subtraction
*	multiplication
/	division
%	modulus
–	unary minus
+	unary plus
++	increment
– –	decrement

2.4 What is a modulus operator? What are the restrictions of a modulus operator?

A modulus operator gives the remainder value. The result of x % y is obtained by (x – (x/y) * y). This operator is applied only to integral operands and cannot be applied to float or double.

2.5 List the rules for using + + and – – operators.

1. The operand must be a variable, but not a constant or an expression.
2. The operator + + and – – may precede or succeed the operand.

2.6 Why n++ executes faster than n + 1?

The expression n + + requires a single machine instruction such as INR to carry out the increment operation whereas, n + 1 requires more instructions to carry out this operation.

2.7 Name the relational operators.

>	greater than
<	less than
> =	greater than or equal to
<=	less than or equal to

2.8 Name the equality operators.

Equality = = and Not equal to !=

2.9 What is a logical expression?

A logical expression is any valid combination of logical values, variables and logical operators. The logical values may be yielded by a relational expression.

2.10 What is a relational expression?

A relational expression is formed by the valid combination of numeric variables, numeric constants and relational or equality operators.

2.11 Differentiate between relational and logical expressions.

Relational expressions use numeric data and relational operators whereas, logical expressions use logical values and logical operators. The logical values may be obtained using relational expressions also. Hence, a logical expression contains relational expressions also whereas, a relational expression contains numeric expressions and not logical expressions.

2.12 List the logical operators.

&& Logical AND
|| Logical OR
! Logical NOT

2.13 List the bitwise operators.

& bitwise AND
| bitwise OR
^ bitwise XOR
~ one's complement
< < left shift
> > right shift

2.14 What is masking?

Masking is an operation in which the desired bits of a binary number or bit pattern is set to zero.

2.15 What is a logical right shift and what is an arithmetic right shift?

In the logical right shift, zero is filled with the leftmost bit position whereas in the arithmetic right shift, the bit shifted in from the left is the same as the bit value formally occupying the leftmost bit position.

2.16 What are the uses of shift operators?

To divide an integer by 2^n, a right shift by n bit positions is applied. To multiply integer by 2^n a left shift by n positions is applied.

2.17 What are the assignments possible in C?

1. Simple assignment
2. Compound assignment
3. Assignment as expressions

For example,

```
a = 10;            /* Simple assignment */
a += 5;            /* Compound assignment */
a = b = c = 0;     /* Assignment as expression */
a = 5 + (b = 8 + (c = 2)) - 4;
```

2.18 What are the operators used in compound assignments?

+, – , *, /, %, < <, > >, &, ^ , | are the operators used in compound assignments.

2.19 What is integral promotion?

If an operand in an expression or a statement is converted to a higher rank data type causing upward type conversion, it is known as integral promotion.

2.20 What is downward type conversion?

If an operand in an expression or a statement is converted to a lower rank data type, it is known as downward type conversion. It may occur when the left-hand side of an assignment has a lower rank data type compared to the values in right-hand side.

2.21 What is type casting or coercion?

Converting one data type to another data type using cast operator is known as coercion. For example, (float) 1/3 and 1/(float)3 represent 1.0/3 and 1/3.0 respectively.

2.22 List the operators having right to left associativity.

! ~ – (unary minus) + + – – & * cast sizeof ?: compound assignment.

2.23 Arrange the operators &, &&, | and || according to the precedence level.

```
    &          |          &&          ||
(high)                                (low)
```

2.24 List the operators having left to right associativity.

( ) [ ] . –> * / % + – << >> < <= >= == != & ^ | && ||

2.25 List the operators having highest priority and lowest priority.

```
( ) [ ] . –>        highest priority
                    lowest priority
```

2.26 What are the exceptions in the evaluation of logical expressions involving && and || ? Give an example.

If the left operand yields false value, the right operand is not evaluated by a compiler in a logical expression using &&. If the left operand yields true value, the right operand is not evaluated by the compiler in a logical expression with the operator ||. If one operand is false in logical AND, the result is false and if one operand is true in logical OR, the result is true. The operators && and || have left to right associativity. Hence, the left operand is evaluated first and based on the output, the right operand may or may not be evaluated.

For example, the following code

```
int i = 0, j = 1;
printf("%d%d%d\n", i++ && ++j, i, j);
```

outputs the values 001. Since, i is 0, the right operand of logical expression is not evaluated and hence j is 1. The value of i is incremented after its value 0 is used in this expression. The value of i will be 1 if printed by another printf() statement following the above given code.

2.27 What is the type of value returned by a cast operator?

Cast operator returns a rvalue of the converted type. Hence, it cannot be assigned to, or incremented/decremented with + +/– –.

2.28 Will the following code work? Justify.

```
int x = 1000000, y = 1000000;
long int z = x*y;
```

The code will work and may produce an unexpected result because the result may exceed the maximum limit. After evaluation of the right-hand side of the assignment to z, the multiplied integer value is converted to long int before assignment.

2.29 Write a program to convert an alphabet from uppercase to lowercase.

```
main( )
     {
     char c;
     c = getchar( ); /* Read an alphabet */
     putchar(c|32); /* conversion to lowercase using bitwise OR */
   }
```

2.30 Write the equivalent expression for x%8.

The expression x%8 is equivalent to x&7.

FILL IN THE BLANKS

1. The second operand of the operators % and / must be _____ .
2. The operator / can be applied to both _____ and _____ types.
3. The modulus operator can be applied only to _____ types.
4. The bits shifted in from the left are the same as the bit value formerly occupying the leftmost bit position in _____ shift.
5. The symbol _____ is one's complement operator.
6. The symbol____ is logical AND operator and the symbol_____ is bitwise AND operator.
7. x + + + + + y is evaluated as _____.
8. Cast operator uses the format _____ .
9. The sizeof () is an _____.
10. Assignment operators use the associativity of_____.
11. The expression (float)(22./7) yields_____ .
12. The expression (double)(22/7) yields_____.

TRUE OR FALSE

1. The expression ++(a+b) is a valid one.
2. The expression a &&= b is a valid one.
3. The expression (long double)(10+25) converts integer constant 35, to long double.
4. The expression – –10 is a valid one.
5. Cast operator returns an rvalue.
6. Some of the unary operators may precede or follow the operand.
7. The assignment statement **a+b = c;** is a valid statement.
8. The exponentiation operator in C is ** .
9. The logical constant false is represented by zero and true by non-zero.
10. A constant or a variable alone may also be treated as an expression.
11. The one's complement operator ~ is a binary operator.
12. Assignment operation is performed during execution of the program.
13. If both the operands of the operator / are integers, the result is a float value.
14. The sizeof operator determines the memory space required by its operand.
15. A char data type always occupies one byte.

MATCH THE FOLLOWING

1	x%y	Relational expression
2	x&y	Operator
3	,	Masking
4	x>>=y	(x–(x/y)*y)
5	x>=y	assignment

OBJECTIVE TYPE QUESTIONS

2.1 The number of binary arithmetic operators in C is

(a) 5 (b) 4 (c) 6 (d) 7

2.2 The number of unary arithmetic operators in C is

(a) 1 (b) 4 (c) 3 (d) 2

2.3 Identify the operator not used in C.

(a) ~ (b) % (c) ^ (d) **

2.4 The operator % yields

(a) quotient value (b) remainder value
(c) percentage value (d) fractional part of the division

2.5 The operator / when applied to floating values yields

(a) remainder value (b) negative value of the remainder
(c) quotient including fractional part (d) integer quotient value only

2.6 The operator % can be applied only to

(a) float values (b) double values (c) options a and b (d) integral values

2.7 x % y is equal to

(a) (x – (x/y)) (b) (x – (x/y) * y) (c) (y – (x/y)) (d) (y – (x/y) * y)

2.8 The operator / can be applied to

(a) integral values (b) float values (c) double values (d) all the above

2.9 The second operand of the operator % must always be

(a) negative value (b) non-zero (c) zero (d) positive value

2.10 If both the operands of the operator / are integers, the result is

(a) a float value (b) an integer value (c) option a or b (d) undefined

2.11 Integer division results in

(a) rounding of the fractional part of the quotient
(b) truncating the fractional part of the quotient
(c) floating value
(d) syntax error

2.12 Assume c1 and c2 as char variables. If c1='A' ; c2 = '2'; what are the results of the statements putchar (cl + 3) and putchar (c2 – 1)?
(a) D, 1 (b) C, 2 (c) d, 1 (d) C, I

2.13 Which is not a valid expression?
(a) +0XAB5 (b) –0525 (c) 15– (d) +a

2.14 Which is not a valid expression?
(a) – p++ (b) ++p – – (c) ++6 (d) ++ x ++

2.15 Which is not a valid expression?
(a) ++(a + b) (b) y– – (c) – – x– – (d) ++p + q

2.16 The equality operator is represented by
(a) := (b) .EQ. (c) = (d) = =

2.17 Identify the logical operator.
(a) ! (b) != (c) ~ (d) = =

2.18 Identify the relational operator.
(a) && (b) > (c) || (d) !

2.19 The number of binary bitwise operators in C is
(a) 3 (b) 4 (c) 5 (d) 6

2.20 The number of bitwise operators in C is
(a) 4 (b) 5 (c) 6 (d) 7

2.21 The symbol of exclusive OR operator is
(a) ^ (b) ~ (c) & (d) |

2.22 The symbol of one's complement operator is
(a) & (b) ^ (c) ~ (d) |

2.23 The symbol of bitwise AND operator is
(a) < < (b) & = (c) && (d) &

2.24 The symbol of bitwise OR operator is
(a) | (b) || (c) ! = (d) > >

2.25 The symbol of left shift operator is
(a) < (b) < < (c) < = (d) <<<

2.26 The symbol of right shift operator is
(a) > = (b) >>> (c) > > (d) >

2.27 The bitwise AND is used for
(a) masking (b) comparison (c) division (d) shifting bits

2.28 The bitwise OR is used to
(a) set the desired bits to 1 (b) multiply numbers
(c) divide numbers (d) set the desired bits to 0

2.29 The bitwise XOR is used to
(a) complement the desired bits (b) multiply the numbers
(c) divide the numbers (d) mask the bits

2.30 Logical right shift results in
(a) maintaining the leftmost bit value
(b) zero is shifted to the leftmost bit position
(c) one is shifted to the rightmost bit value
(d) zero is shifted to the rightmost position

2.31 Arithmetic right shift results in
(a) zero is shifted to the leftmost bit position
(b) one is shifted to the rightmost bit value
(c) maintains the leftmost bit value
(d) zero is shifted to the rightmost bit position

2.32 The result of the expression ~ ~ 7 is
(a) 7 (b) 1 (c) 0 (d) invalid expression

2.33 The operator that can be used for simple encryption is
(a) ~ (b) & (c) ^ (d) all the above

2.34 Identify the invalid compound assignment.
(a) + = (b) ^ = (c) < < = (d) ~ =

2.35 Identify the valid compound assignment.
(a) <<= (b) >>= (c) options a and b (d) >>>=

2.36 Identify the valid expression(s).
(a) a = 0 (b) a = b = 0 (c) a%= (x % 10) (d) all the above

2.37 Explicit type conversion is known as
(a) casting (b) coercion
(c) options a and b (d) upward type conversion

2.38 Which expression yields correct value for the expression 1/3?
(a) 1• / 3 (b) (float) 1/3 (c) options a and b (d) float(1/3)

2.39 The associativity of + + operator is
(a) right to left
(b) left to right
(c) a for arithmetic expression and b for pointer expression
(d) a for pointer expression and b for arithmetic expression

2.40 The associativity of assignment operators is.
(a) right to left
(b) left to right
(c) a for arithmetic expression and b for pointer expression
(d) a for pointer expression and b for arithmetic expression

2.41 The associativity of comma operator is
(a) right to left
(b) left to right
(c) option a for arithmetic expression and b for pointer expression
(d) option a for pointer expression and b for arithmetic expression

2.42 The associativity of ! operator is
(a) right to left
(b) left to right
(c) option a for arithmetic expression and b for pointer expression
(d) option a for pointer expression and b for arithmetic expression

2.43 The associativity of ~ operator is
(a) right to left
(b) left to right
(c) option a for arithmetic expression and b for pointer expression
(d) option a for pointer expression and b for arithmetic expression

2.44 The associativity of indirection operator is
(a) right to left
(b) left to right
(c) option a for arithmetic expression and b for pointer expression
(d) option a for pointer expression and b for arithmetic expression

2.45 The associativity of bitwise AND, OR, XOR is
(a) right to left
(b) left to right
(c) option a for arithmetic expression and b for pointer expression
(d) option a for pointer expression and b for arithmetic expression

2.46 The associativity of logical AND, OR is
(a) right to left
(b) left to right
(c) option a for arithmetic expression and b for pointer expression
(d) option a for pointer expression and b for arithmetic expression

2.47 Which operator has the lowest priority?
(a) Assignment (b) Division (c) Comma (d) Conditional operator

2.48 Which operator has the highest priority?
(a) ( ) (b) → (c) . (d) all the above

2.49 Which operator has the highest priority?
(a) + + (b) % (c) + (d) /

2.50 Which operator has the lowest priority?
(a) & (b) + (c) < = (d) | |

2.51 Which is executed quickly?
(a) p++ (b) ++p (c) options a and b (d) p + 1

2.52 p++ executes faster than p + 1 since
(a) p uses registers
(b) single machine instruction is required for p+ +
(c) options a and b
(d) none of the above

2.53 Of the following, which are logical operators?
(a) && (b) ! (c) | | (d) options a, b, and c

2.54 In a C expression, how is a logical AND represented?
(a) | | (b) .AND. (c) @@ (d) &&

2.55 Which of the following is a ternary operator?.
(a) ? : (b) * (c) sizeof (d) ^

2.56 Which of the following is NOT a bitwise operator?
(a) & (b) ^ (c) | (d) | |

2.57 The type cast operator is
(a) (type) (b) cast( ) (c) // (d) " "

2.58 Which of the following statements is/are TRUE?
1. A char data type variable always occupies one byte independent of the system architecture.
2. The sizeof operator is used to determine the amount of memory occupied by a variable.
(a) both 1 and 2 (b) 1 only (c) 2 only (d) neither 1 nor 2

2.59 What might be the minimum value for the data type int, for 32 bit compiler?
(a) 0 (b) –2,147,483,648 (c) –2,147,483,647 (d) –32768

2.60 If the value of a = 10 and b = –1, the value of x after executing the following expression is
x = (a! = 10) && (b = 1)
(a) 0 (b) 1 (c) –1 (d) none

2.61 What would be the output of the expression 11 ^ 5?
(a) 6 (b) 8 (c) 14 (d) 15

2.62 What is the value of x after executing the following statement?
int x = 011 | 0x10;
(a) 13 (b) 19 (c) 25 (d) 27

2.63 What is the function of the modulus operator in most languages?
(a) Sets a system environmental value to either base 10, base 8, or base 16.
(b) Returns the first argument raised to the second argument power.
(c) Prints out the actual code written to standard output rather than executing the code
(d) Returns the remainder after dividing one number by another.

2.64 What is the value of x in the sample code below?

```
double x; x = (2 + 3) * 2 + 3;
```

(a) 10 (b) 13 (c) 25 (d) 28

2.65 What value will be stored in z if the following code is executed?

```
main()
    {
      int x = 5, y = -10, z;
      int a = 4, b = 2;
      z = x+++++y * b / a;
    }
```

(a) –2 (b) 0 (c) 1 (d) 2

2.66 Which of the following expressions will correctly swap two integers without using a temporary variable?
(a) (x ^ = y), (y ^ = x) (b) (x = y), (y = x)
(c) (x ^ = y), (y ^ = x), (x ^ = y) (d) x ^ = (y ^ = x)

2.67 What is the correct and fully portable way to obtain the most significant byte of an unsigned integer x?
(a) x & 0xFF00
(b) x>>24
(c) x > > (CHAR - BIT * (sizeof(int) – 3))
(d) x > > (CHAR_BIT * (sizeof(int) – 1))

2.68 What is the value of the following expression?

```
i = 1;
i << 1 % 2
```

(a) 2 (b) –2 (c) 1 (d) 0

2.69. What is the value of the following expression?

```
i = 1;
i = (i<<= 1 % 2)
```

(a) 2 (b)1 (c) 0 (d) syntax error

2.70 For the following statements find the values generated for p and q?

```
int p = 0, q =1;
p = q++;
p = ++q;
p = q – –;
p = – –q;
```

The value of p and q are
(a) 1, 1 (b) 0, 0 (c) 3, 2 (d) 1, 2

2.71 Which of the following is a better approach to do the operation **i = i* 16;** ?
(a) multiply i by 16 and keep it (b) shift left by 4 bits
(c) add i 16 times (d) none of the above

ANSWERS

Fill in the Blanks

1. non-zero 2. floating and integral 3. integral
4. arithmetic right 5. ~ 6. && &
7. (x++)+(++y) 8. (data_ type) 9. operator 10. right to left
11. 3.142857 12. 3.000000

True or False

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

Match the Following

1. **4** 2. **3** 3. **2** 4. **5** 5. **1**

Objective Type Questions

1.**a**	11.**b**	21.**a**	31.**c**	41.**b**	51.**c**	61.**c**
2.**b**	12.**a**	22.**c**	32.**a**	42.**a**	52.**b**	62.**c**
3.**d**	13.**c**	23.**d**	33.**d**	43.**a**	53.**d**	63.**d**
4.**b**	14.**c**	24.**a**	34.**d**	44.**a**	54.**d**	64.**b**
5.**c**	15.**a**	25.**b**	35.**c**	45.**b**	55.**a**	65.**c**
6.**d**	16.**d**	26.**c**	36.**d**	46.**b**	56.**d**	66.**c**
7.**b**	17.**a**	27.**a**	37.**c**	47.**c**	57.**a**	67.**d**
8.**d**	18.**b**	28.**a**	38.**c**	48.**d**	58.**a**	68.**a**
9.**b**	19.**c**	29.**a**	39.**a**	49.**a**	59.**b**	69.**a**
10.**b**	20.**c**	30.**b**	40.**a**	50.**d**	60.**a**	70.**a**
						71.**b**

Simple Input / Output Facilities

3

In C language input and output functions are available as **C library functions** provided with each C compiler implementation. The character oriented I/O functions getchar( ) and putchar( ) are used to read / display one character and the formatted I/O functions scanf( ) and printf( ) may be used to read / display one or more characters. Conversion specification is used in scanf( )/printf( ) to convert the next I/O data. The functions scanf() and printf( ) make use of placeholders to read/display the I/O in a particular format. The input function scanf() fails whenever it is not possible to read data; sometimes, the failure may be due to non-matching of any character while using special placeholders. The input function scanf( ) returns the number of inputs successfully read or EOF. The output function printf( ) returns the number of output data successfully displayed or a negative value, when an error is encountered. Simple programs using operators are illustrated ahead.

SHORT ANSWER TYPE QUESTIONS

3.1 What is the need for I/O library functions?

C does not have inherent I/O facilities for data transfer. I/O facilities are available as C library functions. Hence, there is a need for I/O library functions.

3.2 What is a header file?

Header files provide the definitions and declarations for the library functions. Thus, each header file contains the library functions along with the necessary definitions and declarations. For example, stdio.h, math.h, stdlib.h, string.h, etc.

3.3 How can you use I/O library functions in a program?

I/O functions can be used in a program by including **stdio.h** in a program as

```
#include <stdio.h>
```

3.4 Name the character oriented I/O functions.

1. getchar( ) 2. putchar( )

3.5 Name the formatted I/O functions.

1. scanf( ) 2. printf( )

3.6 Give the syntax of getchar( ) and putchar( ).

The function getchar( ) uses no argument and returns the ASCII value of the character read. The function putchar( ) uses the ASCII value as its argument and displays the character.

Format:

```
variable = getchar ();
putchar(ASCII value);
```

For example,

```
char c;
c = getchar();
putchar(c);
```

3.7 What is the syntax of scanf ()?

```
scanf ("format_ string", address_ list);
```

where, **format_string** contains the required formatting specifications and **address_ list** contains the addresses of the memory locations where the input data are stored. These addresses are separated by commas.

For example,

```
int x, y;
float z;
scanf("%d %d %f",&x &y &z);
```

3.8 List the conversion specification of scanf ().

The list is %c %i %d %e %f %g %n %o %p %s %u %x %%.

3.9 What is the effect of using %n in scanf ()?

It assigns the number of characters read so far to the matching argument given in the address list.

3.10 Give the format of printf().

1. printf(" Any string");
2. printf(" format_ string", list);

where **format_string** contains the conversion specifications and **list** contains the expressions separated by commas yielding the values to be displayed.

3.11 Give the conversion specifications used in printf ().

That is %c %d %i %e %f %g %E %G %n %o %p %s %u %x %X %%.

3.12 What is EOF? How is it entered from keyboard?

EOF refers to end of file. It is entered as Ctrl z in DOS environment and Ctrl d in UNIX environment.

3.13 What is a suppression character?

Whenever a character * appears between the percentage sign (%) and conversion character, it is known as suppression character. This character * suppresses the assignment by skipping the input data for it and hence, it is optional to give an argument variable for this data.

3.14 How is fieldwidth included in scanf ()?

Format: %[fieldwidth]conversion_ character

- For example, scanf(**"%2d %3d", &p, &q**);
- assigns 35 to p and 678 to q, for the given input 356783.

3.15 What is a place holder?

A place holder is a conversion specification, that tells how to interpret the next input field. A complete place holder may include, a suppression character and fieldwidth as given below:

%[suppression_ char][fieldwidth]conversion_char

For example, `scanf("%2d %*lc", &x, &y);`

3.16 What is the value returned by scanf ( )?

The function scanf ( ) returns the number of input data, that have been successfully read and assigned on successful action. If an error occurs, EOF is returned.

3.17 Give the complete place holder of printf ( ).

%[flags] [fieldwidth] [.precision] [modifier] conversion character

Flag represents the formatting, fieldwidth specifies the number of columns used for printing, precision specifies the number of digits preceded by dot (.) and modifier specifies the qualifier such as short, long or unsigned.

3.18 List the flags used in printf ( ).

–	Left justified.
+	A sign + or – precedes the signed number.
Blank	A sign – precedes negative value and blank precedes positive value.
#	0 and 0x precede non-zero octal and hexa values respectively, when the conversion characters are 0 and x.
0	Leading zero appears instead of leading blanks.

3.19 What is the value returned by printf ()?

The printf () function returns the number of characters printed. If an error occurs it returns a negative value.

3.20 What is an input field?

A string of non-white space input characters defines input field. The number of characters read from the input field never exceeds the specified fieldwidth.

3.21 Write the code to read double and float data types?

```
float f; double d;
scanf ("%f%lf", &f, &d);
```

The function scanf() uses %f for float and %lf for double as, conversion specifications.

3.22 Why is & used in the arguments of scanf()?

The general format of scanf() is

```
scanf ("format_ string", address_list);
```

where format string gives conversion specifications and address list specifies the list of addresses where the values read are to be stored. The second argument gives the location where the data to be read are to be stored. The operator & yields an address of a variable and hence & precedes the argument. But, if an array or a pointer is used in the argument, it is not preceded by & because they represent a valid address.

3.33 What is the output of the following code?

```
main()
  {
     int i;
     i =1;
     i = i+2*i++;
     printf("%d",i);
  }
```

Output:

4

3.34 What is the output of the following code?

```
#define scanf "%s is string"
main()
    {
      printf(scanf,scanf);
    }
```

Output:
%s is string is string

3.35 What is the output of the following code?

```
i = 2+3, 4>3,1;
printf("%d",i);
```

Output:
5

3.36 What is the output of the following code?

```
main()
  {
    int x = 5;
    printf("%d %d %d\n",x,x<<2,x>>2);
  }
```

Output:
5 20 1

3.37 What is the output of the following code?

```
main()
  {
   int x=20,y=35;
   x = y++ + x++;
   y = ++y + ++x;
   printf("%d %d\n",x,y);
  }
```

Output:
57 94 (Turbo C) 56 93 (ANSI C)

3.38 Supposing that each integer occupies 4 bytes and each character 1 byte, what is the output of the following program?

```
#include<stdio.h>
main()
  {
   int a[] = { 1,2,3,4,5,6,7);
   char c[] = {'a','x','h','o','k' };
   printf("%d\t %d",(&a[3]-&a[0]),(&c[3]-&c[0]));
  }
```

Output:
33

3.39 What is the output of the following code?

```
int x=2;
x=x«2;
printf("%d",x);
```

Output:

8

3.40 What is the output of the program?

```
# define infiniteloop while(1)
main()
  {
   infiniteloop;
   printf("DONE");
  }
```

Output:

none

Explanation: infiniteloop in main ends with ";" . So the loop will not reach the end; and the DONE also will not be displayed.

3.41 What is the output of the following code?

```
#define PRINT(int) printf("%d ",int)
main()
  {
    int x,y,z;
    x=03;y=-1;z=01;
    PRINT(x^x);
    z«=3;PRINT(z);
    y»=3;PRINT(y);
  }
```

Output:

08–1

3.42 If '–' is 45 and '/' is 47 what is the output of the following statement?

```
printf("%d%d%d%d%d%d%d\n",'-','-','-','-'/','/','/');
```

Output:

45454545474747

FILL IN THE BLANKS

1. The character____ is used as suppression character in conversion specification.
2. The function putchar() uses_____, as its argument.
3. The statement_____reads the value of an int variable a.
4. Execution of **scanf("%[^\n] ",str);** reads the input, to store in a char array str, until a _____ is encountered.

5. Execution of **scanf("%*[ ]%c ",&p,&q);** skips_____, if any, while reading the next input.
6. A string of non-whitespace characters, given as input for scanf(), is called_____.
7. Execution of **printf("%d", 'A');** displays_____and **printf("%c", 'A');** displays ____.
8. Execution of **printf("%–5d", 10);** displays the output as ☐☐☐☐☐☐.
9. Execution of **printf("%+d", 25);** displays the output as_____.
10. Execution of **printf("%4.2f", 3.1478);** displays the output as_____.
11. The library functions are made available to a program, by using_____.
12. Conversion specification is represented by_____and a_____.

TRUE OR FALSE

1. C doesn't have inherent I/O facilities.
2. The function scanf() returns the number of input data, that has been read successfully.
3. The function getchar() uses one parameter only.
4. Conversion specification starts with \.
5. If an error occurs, printf() returns a negative value.
6. The format specification %lc is used to print a character using printf().
7. The symbol % is printed using **printf("\%");** in a program.
8. The conversion specification %f is equivalent to %F.
9. The conversion specification %h is used to read hexadecimal values.
10. The conversion specification %0(i.e.,%zero) is used to print octal values.

MATCH THE FOLLOWING

1	getchar()	Charater oriented output function.
2	putchar()	Reads input from keyboard.
3	Scanf()	Displays output on the screen.
4	printf()	Charater oriented input function.

OBJECTIVE TYPE QUESTIONS

3.1 Identify the correct statement.

(a) C library functions provide I/O facilities.
(b) C has inherent I/O facilities.
(c) C doesn't have inherent I/O facilities.
(d) options a and c

3.2 Header files in C contain

(a) compiler commands (b) library functions
(c) header information of C programs (d) operators for files

3.3 How are the library functions made available to a program?

(a) by using # define statements (b) by linking loader to the program
(c) by using # include statements (d) by using function declarations

3.4 Identify the character-oriented console I/O functions.

(a) getchar() and putchar() (b) gets() and puts()
(c) scanf() and printf() (d) fgets() and fputs()

3.5 Identify the formatted console I/O functions.

(a) getchar() and putchar() (b) gets() and puts()
(c) scanf() and printf() (d) fgets() and fputs()

3.6 What is the value returned by getchar() when an alphabet key is pressed?

(a) the alphabet entered from the keyboard
(b) the ASCII value of the alphabet entered from the keyboard
(c) 0
(d) 1

3.7 The function putchar() uses

(a) no argument
(b) one argument that is an ASCII value of a character
(c) two arguments; first one is ASCII value and the second one is number of characters
(d) one argument, that is a string

3.8 The function getchar() uses

(a) no argument
(b) one argument that is a character variable
(c) one argument that is the ASCII value of a character
(d) one argument, that is a string

3.9 Identify the wrong statement.

(a) putchar(65) (b) putchar(‘ x’) (c) putchar(“x”) (d) putchar(‘ \n’)

3.10 The function scanf() returns

(a) the actual values read for each argument
(b) the number of successfully read input values
(c) no value (void)
(d) ASCII values of the characters read

3.11 The function printf() returns

(a) the actual values displayed for each argument
(b) no value (void)
(c) the number of characters displayed
(d) ASCII values of the characters read

3.12. Conversion specification includes

(a) \ and a conversion character (b) / and a conversion character
(c) & and a conversion character (d) % and a conversion character

3.13 Identify the correct statement given **double x;**

(a) `scanf("%d", &x);` (b) `scanf("%f", x);`
(c) `scanf("%d", *x);` (d) `scanf("%|f", &x);`

3.14 Identify the correct statement(s) given **float y;**

(a) `scanf ("%e", &y);` (b) `scanf ("%i", &y);`
(c) `scanf ("% g", &y);` (d) `options a and c`

3.15 For the given scanf() statement, what is the form of input expected?

```
scanf ("%d. %d. %d", &date, &month, &year);
```

(a) 22. 4. 1972 (b) 22 4 1972 (c) 22/ 4/ 1972 (d) options a and b

3.16 Identify the wrong statement given, **int x; float y; char z;**

(a) scanf("%d, %f, %c", &x, &y, &z);
(b) scanf("%d, %f, %c", &x, &y, z);
(c) scanf("Rs. %d, %f, %c", &x, &y, &z);
(d) options a and c.

3.17 What are the values of x, y and z, if 25 20 30 are given as inputs for

scanf("%d %* d %d", &x, &y, &z); ?

(a) x = 25, y is not assigned, z = 30
(b) x = 25, y = 20, z = 30
(c) x = 25, y = 30, z is not assigned
(d) x = 25, y is not assigned, z = 20

3.18 What are the values of a, b and c, given **int a, c; float b;** with inputs as 50, 3.52, 20 for the statement **scanf("%d%*c% f%*c%d", &a, &b, &c);** ?

(a) a = 50 b = , c = 3.52 (b) a = 50 b is not assigned, c = 3
(c) a = 50 b = 3.52 c = 20 (d) a = 50 b and c are not assigned

3.19 What are the values of p and q (integer variables) with inputs given as 50356 for the statement **scanf("%2d %2d", &p, &q);** ?

(a) p = 50q = 35 (b) p = 50q = 356
(c) p = 50 q is not assigned (d) p = 50356 q is not assigned

3.20 What are the values of x and y (integer variables) with inputs given as 356 47 3 for the statement **scanf("%2d %2d", &x, &y);** ?

(a) x = 356 y = 47 (b) x = 35 y = 6 (c) x = 35 y = 4 (d) x = 35 y = 47

3.21 char p, q; scanf("%c%ls", &p, &q);

What are the values assigned, if x and y are given as inputs?

(a) p = x and q = blank (b) p = x and q is not assigned
(c) p = x and q = y (d) p = x and q = x

3.22 Identify the correct statement(s) given the declaration **char x, y**;

(a) `scanf("%c%c", &x, &y);` (b) `scanf("%c%ls", &x, &y);`
(c) `scanf ("%* [ ] %c", &x, &y);` (d) `all the above`

3.23 What is the output of the statement

scanf ("%0%d %f%ls", &a, &b, &c, &d);

for the inputs **45 25.32 20 A** given the declarations **int a,b; float c; char d;**?

(a) a = $(45)_8$ b is not assigned c = 25.32 d = A
(b) a=$(45)_8$ b = 25 c = 0.32 d = 2
(c) a=$(45)_8$ b= 25 c = 20.0 d = A
(d) all the above

3.24 The statement **printf("%*d %*.*f", a, b, c, d, e)**; is interpreted for a = 5, c = 10 and d = 5 as

(a) `printf ("%5d %10.5f ", b, e);`
(b) `printf ("%d %5.5f ", b, e);`
(c) `printf ("%5d %10*f ", c, e);`
(d) `printf ("%*d %10.5f ", a, e);`

3.25 What is the value returned by printf() function, if an error occurs?

(a) Positive value (b) 0 (c) Negative value (d) 1

3.26 What is the purpose of the flag – in printf() function?

(a) Centered (b) Right justified (c) Putting sign (d) Left justified

3.27 What is the purpose of the flag + used in printf() function?

(a) Left justified (b) + or – precedes the signed numbers
(c) Right justified (d) Centered

3.28 What is the purpose of the flag # used in printf() function?

(a) Left justified
(b) Right justified
(c) 0 and 0x precede non-zero octal and hexa values respectively
(d) Leading zeros appear

3.29 What is the purpose of the flag 0 used in the function printf()?

(a) Leading blanks are displayed (b) Leading zero appears
(c) – sign precedes (d) + or – sign precedes

3.30 What is the purpose of the flag blank used in the function printf()?

(a) – precedes negative value and blank precedes positive value.
(b) – precedes negative value and + precedes positive value.
(c) – sign precedes negative value and nothing precedes positive value.
(d) Blank precedes negative and positive numbers.

3.31 What is the output of the following code?

```
main()
 {
   printf("%c", 'A');
 }
```

(a) 65 (b) a (c) A (d) Error

3.32 What is the output of the following code?

```
main()
 {
   printf("%d", 'A');
 }
```

(a) 65 (b) a (c) A (d) Error

3.33 What is the output of the following code?

```
main()
   {
        char name[10];
        strcpy(name, "Poorani");
        printf("%c%c%c", name[4], name[5], name[6]);
    }
```

(a) rani (b) ran (c) ani (d) orani

3.34 What format is used to print a character with the printf function?

(a) %s (b) %c (c) %char (d) %lc

3.35 "The stock's value increased by 15%."

Select the statement, which will exactly reproduce the line of text above.

(a) printf("\"The stock\'s value increased by %d%%.\"\n", 15);
(b) printf("The stock\'s value increased by %d%.\n", 15);
(c) printf("\"The stock\'s value increased by %d%%.\"\n", 15);
(d) printf("\"The stock\'s value increased by %d%.\"\n");

3.36

```
main()
    {
     double x = ½.0 - 1/2 ;
     printf("x=%.2f\n", x);
    }
```

What will be printed when the above code is executed?

(a) x=0.00 (b) x=0.25 (c) x=0.50 (d) x= 1.00

3.37

```
main()
  {
    int x = 5,y = 10;
    double z;
    z = (double) (x / y);
    printf("z = %.2f\n", z);
  }
```

What will be printed when the above code is executed?

(a) z = 0.00 (b) z = 0.50 (c) z = 1.00 (d) 2.00

3.38 What will be printed when the following code is executed?

```
main()
  {
    int i = 4, a = 6;
    double z;
    z = a /i;
    printf("z=%.2f\n", z);
  }
```

(a) z =0.00 (b) z =1.00 (c) z =1.50 (d) z =2.00

3.39
```
main()
    {
     int x = 3, y = 3, z = 3;
     z - =  x-- - --y;
     printf( "x = %d y = %d z = %d", x, y, z );
     }
```
What will be the values of x, y, and z?
(a) x = 2 y = 2 z = 2 (b) x = l y = 2 z =2 (c) x = 2 x = 2 y = 0 (d) x = 2 y = 2 z = 1

3.40
```
main()
    {
     int x, y, z ;
     printf(" %d\n", ( z>= y>= x ? 10 : 20 ));
    }
```
What might be the value of x, y, and z, if 20 has to be printed?
(a) 0, 0, 0 (b) –1, –1, –2 (c) –1, 0, 1 (d) none

3.41
```
# include <stdio.h>
main( )
  {
   int a = 0xff;
   (a << 4 >> 12)? printf("Sandeep"):printf("Ashok");
  }
```
What will be the output of the program?
(a) Ashok (b) Sandeep
(c) Error: Incorrect assignment (d) None

3.42
```
int x = 0x1234, y = 0x5555;
printf("0x%04.4x\n", x | y);
printf("0x%04.4x\n", x & y);
printf("0x%04.4x\n", -x ^ y);
printf("0x%04.4x\n", ~x);
```
What is output from the above sample code?
(a) The code will not compile.
(b) 0x5775
0x1014
0x4761
0xffffedcb
(c) 0x15f3
0x0492
0x1161
0xfffffb2d
(d) 0x55d7
0x0450
0x5187
0xfffffb2d

3.43
```
printf("Daily Click!");
```
In the code segment above, the words Daily Click! will be printed to what output device?
(a) to an E-mail message
(b) to the console
(c) to the default printer
(d) to wherever standard output has been directed to

3.44 If x = 2,y = 6, and z = 6 then, what will be the output of
```
x = y = = z;
printf(%d",x);
```
(a) 0 (b) 1 (c) 2 (d) 6

3.45
```
int i = 20;
printf ("%x", i);
```
What is the output?
(a) x 14 (b) 14 (c) 20 (d) none of the above

3.46
```
float x, y, z;
scanf("%f%f", &x, &y); printf("%.2f,%.2f", x,y);
```
if input stream contains "4.2 3.2 3..." what will be the output?
(a) 4.20, 3.00 (b) 4.2, 2.3 (c) 4.20, 2.30 (d) 4.20 3.20

3.47 What will be the result of executing the following statement?
```
int i=10;
printf("%d %d %d",i, ++i, i++);
```
(a) 10 11 12 (b) 10 11 11
(c) result is operating system dependent (d) result is compiler dependent

3.48 What will be the result of the following program if the inputs are 2 3?
```
main()
  {
   int a,b;
   printf("enter two numbers :");
   scanf("%d%d",a,b);
   printf("%d+%d=%d",a,b,a+b);
  }
```
(a) 2+3=5 (b) syntax error
(c) will generate run time error /core dump (d) a+b=5

3.49
```
main()
  {
   int a=10,b=5, c=3,d=3;
   if((a<b)&&(c=d++)) printf("%d %d %d %d", a,b,c,d);
   else printf("%d %d %d %d", a,b,c,d);
  }
```
(a) 10 5 3 4 (b) 10 5 4 4 (c) 10 5 3 3 (d) none of the above

3.50 What is the output of the following statement?
```
printf("%u",-1);
```
(a) –1 (b) minimum int value (c) maximum int value (d) error

ANSWERS

Fill in the Blanks

1. * 2. ASCII value 3. scanf("%d",&a); 4. new line
5. blank space 6. input field 7. 65, A 8. 1 0
9. +25 10. 3.15 11. # include facility
12. % , conversion character

True or False

1. True 2. True 3. False 4. False 5. True 6. False
7. False 8. False 9. False 10. False

Match the Following

1. 4 2. 1 3. 2 4. 3

Objective Type Questions

1. d	11. c	21. c	31. c	41. a
2. b	12. d	22. d	32. a	42. b
3. c	13. d	23. b	33. c	43. b
4. a	14. d	24. a	34. b	44. b
5. c	15. a	25. c	35. a	45. b
6. b	16. b	26. d	36. c	46. d
7. b	17. a	27. b	37. a	47. d
8. a	18. c	28. c	38. b	48. c
9. c	19. a	29. b	39. a	49. c
10. b	20. b	30. a	40. d	50. c

Control Flow Constructions

4

C is a structured programming language. In this, a null statement is represented by a semicolon and a simple statement is terminated by a semicolon. Compound statements are written within braces and are known as blocks. C supports many control constructs broadly classified under conditional execution and unconditional execution. Selective and loop constructs are conditional execution type and **goto, break** and **continue** are unconditional execution statements. Selective control structures may be written using a conditional expression or an **if-else** construct or an **if-else if** construct or a **switch-case** construct. A loop control structure may use **for** or **while** or **do-while** construct. **Break, continue** or **goto** may be used for earlier exit from the control constructs. Null statements are useful to create time delay and to end a label statement where no useful operation is to be performed.

SHORT ANSWER TYPE QUESTIONS

4.1 Differentiate between an expression and a statement.

An expression does not end with a semicolon whereas, a statement ends with a semicolon.

4.2 What is a block ?

A block is a compound statement. A group of statements and declarations surrounded by the opening brace { and closing brace } forms a block.

4.3 What is conditional execution?

The flow of execution may be transferred from one part of a programme to another part based on the output of the conditional test carried out It is known as conditional execution.

4.4 What is unconditional execution?

If the flow of execution is transferred from one part of a program to another part without carrying out any conditional test, it is known as unconditional execution.

4.5 Name the selective control structures.

1. if-else 2. if-else-if 3. switch-case 4. conditional expression

4.6 Name the loop constructs.

1. for 2. while 3. do-while

4.7 Name the unconditional constructs.

1. goto 2. break 3. continue

4.8 Give the syntax of if-else construct.

```
if(expression)
 {
   statement 1;
 }
 else
 {
   statement2;
 }
```

For example,

```
  if(a < b)
  {
   printf (" %d\n", a);
  }
 else
   {
  printf ("%d\n", b);
   }
```

4.9 Give the syntax of conditional expression.

Format:

```
condition ? expressionl : expression2;
```

If **condition** is true **expressionl** is returned, else **expression2** is returned.

For example, x < 0? - x : x;

4.10 Give the syntax of if-else-if structure.

Format:

```
if (expression1)
    {
     statement(s) ;
    }
  else if (expression2)
    {
  statement(s);
  }
 else
  {
    statement(s);
  }
```

The else part is associated with the closer if. If the expression1 is true, its following block is executed; otherwise expression2 is evaluated. If expression2 is true, its following block is executed; otherwise the block following else is executed. Any number of else-if and else may be added according to the requirement.

4.11 Write a program to check whether an integer is odd or even.

```
main()
    {
      int x;
    scanf ("%d", &x); /* read value */
    if(x % 2)
        {
         printf("The number %d is ODD\n",x);
        }
      else
        {
        printf("The number %d is EVEN\n",x);
        }
    }
```

4.12 Write a program to get the absolute value of a number.

```
main()
  {
    int x, y;
    scanf("%d", &x);
    y = (x < 0)? -x : x;
    printf("Absolute value of %d is %d\n", x, y);
  }
```

4.13 Write a program to convert the entered character into uppercase letter.

```
#include <ctype.h>
main()
  {
   char c;
   c = getchar();
   printf("%c\n", toupper(c) );
  }
```

4.14 Give the syntax of while structure.

Format:

```
while(expression)
      {
        statements;
      }
```

For example,

```
while(a < 10)
      {
       printf("%d\n", a);
       a++;
      }
```

4.15 Give examples of infinite loop.

```
while(1) {...}
      Or
while(!0) (...}
```

4.16 Give the syntax of do-while loop.

Format:

```
do
 {
  statements;
 }
while ( expression );
```

For example,

```
a=1;
do
 {
  printf("%d\n", a++);
 }
while ( a < 10 );
```

4.17 Give the syntax of for loop.

Format:

```
for( expressionl; expression2; expression3 )
    {
     statements;
    }
```

(1) 'expression1' is executed once. (2) ' expression2' is evaluated and if it is true, ' statements' are executed. (3) ' expression3' is evaluated. Steps (2) and (3) are repeated until ' expression2' becomes false. For example,

```
for( i = 1; i < 10; i++ )
   {
    printf("%d\n", i);
   }
```

4.18 Give the equivalent while structure and do-while structure for the following loop structure.

```
for( i = 0; i < 10; i++ )
   {
    printf("%d\n",i);
   }
```

while structure	do-while structure

`i = 0;`
`while ( i < 10 )`
`{`
`printf("%d\n", i);`
`i++;`
`}`
	`i = 0;`
	`do`
	`{`
	`printf("%d\n", i);`
	`}`
	`while ( i++ < 10 );`

4.19 Write the infinite loops using for loop.

1. for(; ;) {...}
2. for(expressionl ; ; expression3) {...}
3. for(; ; expression3) {...}
4. for(expressionl; ;) {...}

4.20 What is a comma expression?

A comma expression uses comma as its operator. The operator allows multiple expressions to be used where a single expression is normally allowed. It is evaluated from left to right when more than one comma operator exists.

4.21 Where is a comma operator useful? Give a suitable example.

Normally a comma operator is used in for loop to have multiple expressions which are to be evaluated. For example,

```
for ( i = 1, j = 1, j <= n; i += 2, j++ )
  {
    printf("%d%d\n", i, j);
  }
```

4.22 What is a nested loop?

A loop construct, which appears within another loop construct, is known as nested loop structure. The loops should not overlap each other.

4.23 What is the difference between break and continue?

S.No.	Break	Continue
1.	It helps to make an earlier exit from the block where it appears.	It helps in leaving the remaining statements in a current iteration of a loop and continues with the next iteration.
2.	It can be used in all control statements, except if construct.	It can be used only in loop constructs.

4.24 What are called structured goto statements?

The statements break and continue are known as structured goto statements.

4.25 Give the syntax of switch case structure.

Format:

```
switch ( expression )
   {
    case constl:   statements;
                   break;
    case const2:   statements;
                   break;
    case constN:   statements;
                   break;
        default:   statements;
                   break;      }
```

The expression must be an integer expression. Each case label following case is an integer value. The last statement is default statement. If break is not used in each case statement, all the statements following the matched case statement will be executed.

4.26 Give the syntax of goto statement.

```
goto label;
...
...
label: statements;
```

For example,

```
if( x <0 )
goto end;
...
...
end: printf ("Program terminated/n");
}
```

4.27 What are the two types of goto statements?

1. Forward goto:
If the statements along with the label appear after the goto statement, it is a forward goto.

2. Backward goto:
If the statements along with the label appear below the goto statement, it is a backward goto.

4.28 Which of the following code is preferred? Why?
l. if(x==0) 2. if(0==x)

The second code is preferred. By mistake, if (0 = x) is written, this will produce error whereas the first one will accept.

4.29 What is the output of the program?

```
main()
    {
     int a=2, b=3;
     printf(" %d ", a+++b); }
```

Here a+++b is evaluated as a++ + b.

Output:
5

4.30 What are to be replaced in ??? to get the sum of the series

sum(x)=1 + (1+2) + (1+2+3) + ... + (1+2+3+4+...+ x)

in the following code?

```
sum(int x)
  {
  int i, term = 1, s=1;
  for(i=2;i <= ???;i++)
   {
    ???;
    s+=term ;
   }
  return s ;
  }
```

The correct code for the loop is given below.

```
for(i=2;i<= x;i++)
    {
      term +=i;
      s+=term;
    }
```

4.31 What is the output of the program?

```
main()
  {
    int a=0;
    if (a=0); printf("Ramco Systems\n");
    printf("Ramco Systems\n");
  }
```

Output:
"Ramco Systems"
will be printed once since = is used instead of = = in if(a=0).

4.32 What is the output of the program?

```
int count = 11;
while (--count+l);
      printf("count down is %d \n",count);
```

Output:
countdown is –1

4.33 What is the output of the following code?

```
# define TRUE 0
main()
   {
    while(TRUE)
     {
     ...
     }
  }
```

Output:
This won't go into the loop as TRUE is defined as 0

4.34 What is the output of the program?

```
#include <stdio.h>
main()
   {
    int y,z;
    int x=y=z=10;
    int f=x;
    float Output=0.0;
    f *=x*y;
    Output=x/3.0+y/3;
```

```
    printf("%d %.2f",f,Output);
   }
```

Output:
1000 6.33

4.35 What is the output of the program?

```
#include <stdio.h>
main()
    {
     int i=100, j=20;
     i++=j;
     i*=j;
     printf("%d\t%d\n",i,j);
    }
```

Output:
Error i++=j not allowed. If it is made as i+=j output is 2400 20

4.36 What is the output of the program?

```
#include <stdio.h>
main()
    }
      int varl,var2,var3,minmax;
      varl=5;  var2=5;  var3=6;
      minmax=(varl)?(var3)?varl:var3:(var2)?var2:var3;
      printf("%d\n",minmax);
    }
```

Output:
5 [**Hint:** Ternary operator is evaluated from right to left]

4.37 What is the output of the program?

```
#include <stdio.h>
const int k=100;
main()
    {
      int a[100]; int sum=0;
      for(k=0;k<100;k++)*(a+k)=k;
      sum+=a [--k] ;
      printf("%d",sum);
    }
```

Output:
Error, since const value k cannot be modified.

4.38 How is a nested ternary expression evaluated?

A nested ternary expression is evaluated from right to left. The evaluated value is substituted in the place of ternary expression and the evaluation continues until all ternary expressions are evaluated. For example,

3>4?5<6?1:2:10>8?3:4

is evaluated as

3>4?5<6?1:2:3

and then evaluated as
3>4?1:3
and the result is 3.

4.39 Write an efficient program to print an NXN identity matrix.

The program given below is not efficient.

```
for(i=0;i<N;i++)
   for(j=0;j<N;j++)
      a[i] [j] = 0;
   for(i=0;i<N;i++)
      a[i] [i] = 1;
```

The same program can be written efficiently as follows.

```
for(i=0;i<N;i++)
{
 for(j=0;j<N;j++)
    a[i] [j] = 0;
    a[i] [i] = 1;
}
```

If the adjacent loops are combined as given above, it is known as **loop jamming**. The efficient way of writing this program uses loop jamming.

4.40 What is loop inversion?

If a loop is to be executed for a specified number of times and the counter variable is not used within the loop, the loop may be written as given below using count down instead of count up. This type of looping is known as loop inversion.

For example,

The loop inversion of the following for loop

```
for(i=1;i<=10;i++) {...}
```

is written as

```
i=11;
while (--i ) {...}
```

FILL IN THE BLANKS

1. A null statement is represented by a _____.
2. The minimum number of times a while loop is executed is_____.
3. The expression **5< 10?6> 8?5:3:7** is reduced to_____after evaluating the first expression in it.
4. The do-while loop is also known as_____loop.
5. for(; ;) {...} forms an_____loop.
6. After evaluation of the expression x= (a= 10,3*a) the value of a is_____.
7. The case label in a switch-case must be an_____.

8. The label in a goto is an_____.
9. Structured goto statements are_____and_____statements.
10. while(i=10) {...} results in an_____loop.
11. A group of declarations and statements surrounded by { and } form a _____.
12. The_____ statement in a switch case statement skips all the remaining statements and exits the switch block.

TRUE OR FALSE

1. A null block contains {} only.
2. The minimum number of times a for loop is executed is zero.
3. The continue statement can be used both in selective and loop constructs.
4. Ternary expression is evaluated from right to left.
5. The do-while loop is terminated when its conditional expression returns true value.
6. Ternary expression is a selective control construct.
7. The conditional expression need not be written within the parentheses in any control construct.
8. A semicolon must be placed after closing the conditional expression in a do-while construct.
9. One type of loop can be nested in another type of loop.
10. A break statement can be used both in loop and selective control construct.
11. Compound statement in **case label** must be enclosed within braces.
12. In a switch-case statement, labels must be declared in ascending order.

MATCH THE FOLLOWING

1	goto	Loop construct
2	while	Selective construct
3	ternary	Unconditional statement
4	break	Three operands
5	switch-case	Earlier exit

OBJECTIVE TYPE QUESTIONS

4.1 Structural programming approach makes use of
(a) modules (b) control structures
(c) user defined data types (d) all the above

4.2 A statement is differentiated from an expression
(a) by terminating it by a semicolon (b) by terminating it by a newline character
(c) by terminating it by a NULL (d) by terminating it by a blank space

4.3 A null statement can be represented by a
(a) newline (b) blank space (c) semicolon (d) colon

4.4 A block is enclosed within a pair of
(a) { } (b) ( ) (c) [ ] (d) < >

4.5 Identify the unconditional control structure.
(a) do-while (b) switch-case (c) goto (d) if

4.6 Identify the loop construct.
(a) if-else (b) switch-case (c) goto (d) while

4.7 The number of loop constructs in C is
(a) 2 (b) 3 (c) 4 (d) 5

4.8 Identify the wrong statement.
(a) `if (a<b);` (b) `if a<b;` (c) `if (a<b){;}` (d) options b and c.

4.9 Which is syntactically correct?
(a) if (a := 10) { ... } else if (a< 10) { ... }
(b) if (a == 10) { ... } else if (a< 10) { ... }
(c) if (a eq 10) { ,.. } else if (a< 10) { ... }
(d) if (a .eq. 10) { ... } else if (a< 10) { ... }

4.10 Which is the correct statement?
(a) printf ("Maximum = % d\n",(x.y)? x:y);
(b) `printf ("%s\n", (mark > = 60)?"FIRST CLASS":"NOT FIRST CLASS");`
(c) `printf("%s \n", "PASS");`
(d) all the above

4.11 The minimum number of times the while loop is executed is
(a) 0 (b) 1 (c) 2 (d) cannot be predicted

4.12 The minimum number of times the for loop is executed is
(a) 0 (b) 1 (c) 2 (d) cannot be predicted

4.13 The minimum number of times the do-while loop executed is
(a) 0 (b) 1 (c) 2 (d) cannot be predicted

4.14 The do-while loop is terminated when the conditional expression returns
(a) zero (b) 1 (c) non-zero (d) –1

4.15 Which conditional expression always returns false value?
(a) if (a = = 0) (b) if (a = 0) (c) if (a = 10) (d) if (10 == a)

4.16 Which conditional expression always returns true value?
(a) if (a = 1) (b) if (a = = 1) (c) if (a = 0) (d) if (1 == a)

4.17 What is (are) the statement(s) which results in infinite loop?
(a) for (i = 0; ; i++); (b) for (i = 0; ;); (c) for (; ;); (d) all the above

4.18 In the following loop construct, which one is executed only once always.

```
for (expr1; expr2; expr3) { ... }
```

(a) expr1 (b) expr3
(c) expr1 and expr3 (d) expr1, expr2 and expr3

4.19 Identify the wrong construct.
(a) for (exprl; expr2;) (b) for(exprl; expr3) (c) for (; expr;) (d) for (; ; expr3)

4.20 Infinite loop is
(a) useful for time delay (b) useless
(c) used to terminate execution (d) not possible

4.21 What is the value of x in the expression x = (a = 10, a*a) ?
(a) invalid expression (b) 0 (c) 10 (d) 100

4.22 Comma is used as
(a) a separator in C (b) an operator in C (c) a delimiter in C (d) terminator in C

4.23 What is the result of a multiple expression separated by commas?
(a) result of the leftmost expression after evaluation
(b) result of the rightmost expression after evaluating all other previous expressions
(c) no value is returned
(d) result of arithmetic operations, if any

4.24 Which is the correct statement?
(a) while loop can be nested
(b) for loop can be nested
(c) options a and b
(d) one type of a loop cannot be nested in another type of loop

4.25 The break statement is used to
(a) continue the next iteration of a loop construct
(b) exit the block where it exists and continues further sequentially
(c) exit the outermost block even if it occurs inside the innermost block and continues further sequentially
(d) terminate the program

4.26 The continue statement is used to
(a) continue the next iteration of a loop construct
(b) exit the block where it exists and continues further
(c) exit the outermost block even if it occurs inside the innermost
(d) continue the compilation even an error occurs in a program

4.27 The continue statement is used in
(a) selective control structures only (b) loop control structures only
(c) options a and b (d) goto control structure only

4.28 The break statement is used in
(a) selective control structures only (b) loop control structures only
(c) options a and b (d) switch-case control structures only

4.29 If break statement is omitted in each case statement
(a) The program executes the statements following the case statement where a match is found and exits the switch-case construct.
(b) The program executes the statements following the case statement where a match is found and also all the subsequent case statements and default statements.
(c) The program executes default statements only and continues with the remaining code.
(d) Syntax error is produced.

4.30 If default statement is omitted and there is no match with case labels

(a) No statement within switch-case will be executed.
(b) Syntax error is produced.
(c) Executes all the statements in the switch-case construct.
(d) Executes the last case statement only.

4.31 When is default statement executed in switch-case construct?

(a) Whenever there is exactly one match.
(b) Whenever break statement is omitted in all case statements.
(c) Whenever there is no match with case labels.
(d) options b and c.

4.32 Identify the wrong statement where – – – represents C code.

(a)
```
label : {- - - } - - - goto label;
```
(b)
```
goto end; - - -
        end: {- - -}
```
(c)
```
goto 50; - - -
        50: {- - -}
```
(d)
```
begin: {- - -}
        if (a < 10) goto begin;
```

4.33 C is an example of

(a) object oriented language
(b) structured programming language
(c) object based language
(d) component based language

4.34 The syntax of if statement is

(a) if expression then program-statement
(b) if (expression) program-statement
(c) if (expression) then program-statement
(d) if expression {program-statement}

4.35
```
main ( )
    {
      int a = 2, b = 4,  c = 8, x = 4;
      if (x == b) x = a;  else x = b;
      if (x ! = b) c = c + b;  else c = c + a;
      printf ("c = %d\n", c);
     }
```

What will be printed when the above code is executed?

(a) c = 4 (b) c = 8 (c) c = 10 (d) c = 12

4.36
```
main ()
    {
      unsigned int x = -10; int y = 10;
      if (y <= x) printf ("He is good\n");
      if  (y = =  (x  =  -10))  printf ("She is better\n");
      if ((int) x== y) printf ("It is the best\n");
    }
```

What will be the output of above sample code ?

(a) She is better (b) He is good
It is the best

(c) It is the best (d) He is good

4.37
```
if  (! 3.14) printf  ("The value of exponent\n");
else printf  ("The value of PIE is used in conditional part\n ");
```
What might be the result?

(a) Float value can't be given in conditional part.

(b) The value of PIE is used in conditional part.

(c) The value of exponent.

(d) None of the above.

4.38
```
main ( )
  {
  int x;
  if( x > 4 ) printf("Brindha");
  else if( x > 10 ) printf("Karthik");
       else if( x > 21 ) printf("Pradeep");
            else printf("Sandeep");
  }
```
What will be the value of x so that "Karthik" will be printed ?

(a) from 10 to 21

(b) from 11 to 21

(c) greater than 10

(d) none.

4.39
```
main()
   {
    i = 0; j = 0;
    for(j=1;j<10;j++)
    { i = i+1; }
   }
```
In the (generic) segment above, what will be the value of the variable i on completion?

(a) 1 (b) 3 (c) 9 (d) 10

4.40
```
main()
   {
    int x = 0;
    for (x=1; x<4; x++);
    printf("x=%d\n", x);
   }
```
What will be printed when the above sample is executed?

(a) x=0 (b) x=1 (c) x=3 (d) x=4

4.41
```
main()
    {
     int x = 0;
     for(; ;)
      {
       if (x++ = 4) break;
       continue;
      }
    printf("x=%d\n",x);
    }
```
What will be printed when the above code is executed?
(a) x=0 (b) x= 1 (c) x=4 (d) x=5

4.42
```
int y;
for (Y=0; y<3; y++);
printf("y=%d\n", y);
```
What will be printed when the sample code above is executed?
(a) y=0 (b) y=0 y=1 y=2 (c) y=2 (d) y=3

4.43
```
main ( )
    {
     int i=0;
     for (;++i;)
     printf ("%d",i );
    }
```
How many times the for loop will be executed?
(a) 0 (b) infinite (c) 32767
(d) The maximum integer value that can be represented in the machine.

4.44
```
main ()
    {
     unsigned int i = 10;
     while (i-->= 0);
    }
```
How many times the loop will be executed?
(a) 10 (b) zero time (c) one time (d) none

4.45
```
int x = 10;
```
Which of the following is the finite loop?
(a)
```
while (1)
     {
      if ( !( x != 10 ))
      continue;
      ....
     }
```

(b)
```
while (! 0)
     {
      if ( x==10)
      continue;
      ....
     }
```

(c)
```
do
   {
   ....
   }while( !( x > 10));
```

(d)
```
while( x > 10 )
     { ... } ;
```

4.46
```
main()
   {
     int score = 4;
     switch (score)
        {
          default:
                  ;
          case 3:
                 score += 5;
                 if ( score == 8)
                 {
                   score++;
                   if (score == 9) break;
                   score *= 2;
                 }
                 score -=4;
                 break;
            case 8:
                 score +=5;
                 break;
     }
     printf("SCORE = %d\n", score);
   }
```

What will be the output of the above code?

(a) SCORE = 5 (b) SCORE = 4 (c) SCORE = 9 (d) SCORE = 10

4.47
```
switch (s)
  {
    case 1 : printf("Balan")
    case 2 :
    case 3 : printf("Elango")
    case 4 : printf("Thiruvalluvan")
    default : printf("Kamban")
  }
```

To print only Kamban, what will be the value of s?

(a) 2 (b) 1 or 3 or 4 (c) any int value other than 1,2,3 and 4 (d) 4

4.48
```
main ()
    {
     int a=-4, r, num = 1;
     r = a% -3;
     r = ( r ? 0 : num * num ) ;
     printf ("%d",r);
    }
```

What will be the output of the program?

(a) 0 (b) 1

(c) –ve is not allowed in the mod operands. (d) none

4.49 What is the value of int variable **result** if x = 50, y = 75 and z = 100?

```
result = ( x + 50 ? y >= 75 ? z > 100 ? 1 : 2 : 3 : 4);
```

(a) 1 (b) 2 (c) 3 (d) 4

4.50 Omitting the break statement from a particular case

(a) leads to a syntax error

(b) causes execution to terminate after that case

(c) causes execution to continue all subsequent cases

(d) causes execution to branch to the statement after the switch statement

4.51
```
int i;
i=2;
i++;
if (i=4)
    {
      printf("i=4");
    }
      else
       {
        printf("i=3");
       }
```

What is the output of the program?

(a) i =4 (b) i =3 (c) unpredictable (d) none

4.52
```
int a = 0, b = 2;
if  (a = 0)  b = 0;
else b *= 10;
```

What is the value of b?

(a) 0 (b) 20 (c) 2 (d) none

4.53
```
int x = 2, y = 2, z = 1;
```

What is the value of x after the following statements?

```
if (x = y%2) z +=10;
else z +=20;
```

(a) 0 (b) 2 (c) 1 (d) none

4.54 What is the result?

```
main ()
  {
    char c =-64; int i =-32;
    unsigned int u =-16;
    if(c>i)
    {
         printf ("passl");
         if(c<u) printf ("pass2");
         else printf ("Fai12");}
         else printf ("Faill");
         if (i<u) printf("pass2");
         else printf("Fail2");
  }
```

(a) pass1pass2 (b) pass1Fail2 (c) Fail1pass2 (d) Fail1Fai12

4.55 How many x are printed?

```
for (i = 0, j = 10; i < j; i++, j--)
printf ("x");
```

(a) 10 (b) 5 (c) 4 (d) none

4.56 What is the output of the following program?

```
main ()
  {
    unsigned int i;
    for (i = 10; i >= 0; i--)
       printf ("%d", i);
  }
```

(a) prints numbers 10 to 0 (b) prints numbers 10 to 1
(c) prints numbers 10 to –1 (d) goes into infinite loop

4.57

```
unsigned char c;
for ( c=0;c!=256;c+2)
printf("%d",c);
```

How many times the loop is executed?

(a) 127 (b) 128 (c) 256 (d) Infinitely

4.58 For the following code how many times the printf () function is executed?

```
int i,j ;
for(i=0;i<=10;i++);
for (j=0;j<=10;j++);
       printf ("i=%d,j=%d\n",i,j);
```

(a) 121 (b) 11
(c) 10 (d) none of the above

4.59 What is output of the following code?

```
main ()
  {
   int i=3;
   while(i--)
      {
       int i=100;
       i--;
       printf ("%d..",i);
      }
  }
```

(a) Infinite loop
(b) Error
(c) 99..99..99..
(d) 3..2..1..

4.60 What will be the output of the following code?

```
{
    ch='A';
    while (ch<='F')
      {
          switch (ch)
           {
              case'A':case'B':case'C':case'D':ch++;continue;
              case'E':case'F':ch++;
           }
              putchar(ch);
      }
}
```

(a) ABCDEF (b) EFG (c) FG (d) Error

4.61

```
main ()
  {
     int, ones, twos, threes, others; int c;
     ones = twos = threes = others = 0;
     while ((c = getchar ()) !_ '\n')
       {
     switch (c)
       {
     case '1': ++ones;
     case '2': ++twos;
     case '3': ++threes;
               break;
     default: ++others;
              break;
     }
```

```
    }
    printf ("%d%d", ones, others);
  }
```

If the input is "lalblc" what is the output?

(a) 13 (b) 34 (c) 33 (d) 31

4.62 Result of the following program is

```
main()
  {
    int i=0;
    for(i=0;i<20;i++)
      {
        switch(i)
            {
              case O:i+=5;
              case l:i+=2;
              case 5:i+=5;
              default i+=4;
              break;
            }
        printf("%d,",i);
      }
    }
```

(a) 0,5,9,13,17 (b) 5,9,13,17
(c) 12,17,22 (d) 16,21

4.63 What is the output of the following code?

```
main()
  {
    int i = 0;
    switch(i)
     {
      case 0 : i++;
      case 1 : i+++2
      case 2 : ++i;
    }
    printf ("%d",i++);
  }
```

(a) 1 (b) 3 (c) 4 (d) 5

ANSWERS

Fill in the Blanks

1. semicolon	2. zero	3. 5<10?3:7	4. post tested
5. infinite	6. 30	7. integral constant	8. identifier
9. break, continue	10. infinite	11. block	12. break

True or False

1. True	2. True	3. False	4. True	5. False	6. True
7. False	8. True	9. True	10. True	11. False	12. False

Match the Following

1. 3	2. 1	3. 4	4. 5	5. 2

Objective Type Questions

1. d	11. a	21. d	31. d	41. d	51. a	61. c
2. a	12. a	22. b	32. c	42. d	52. b	62. d
3. c	13. b	23. b	33. b	43. d	53. a	63. b
4. a	14. a	24. c	34. b	44. d	54. c	
5. c	15. b	25. b	35. d	45. d	55. b	
6. d	16. a	26. a	36. d	46. a	56. a	
7. b	17. d	27. b	37. b	47. c	57. d	
8. b	18. a	28. c	38. d	48. a	58. a	
9. b	19. b	29. b	39. c	49. b	59. c	
10. d	20. a	30. a	40. d	50. c	60. c	

Storage Classes of Variables

5

Variables in C may possess an attribute called storage class in addition to the data type. A storage class controls the lifetime, scope and linkage of a variable. There are five types of storage classes, viz. auto, register, static, extern and typedef.

Auto variables appear within a block and they have block scope, temporary storage and no linkage. If a variable declaration appears in a block without auto, the default storage class auto is assigned.

Register variables have scope, longevity and linkage like auto variables. But the values are stored in the CPU registers instead of the main memory. Since, only a few values can be placed in the CPU registers, frequently used variables may be declared as register variables. Due to the non-availability of CPU registers, the register variables may be treated as auto variables. Register variables have faster access than the other variables resulting in faster execution of the program.

A static variable may be an internal static or an external static. Internal static variables have block scope, persistent storage and no linkage as they are declared inside a block. External static variables are declared outside the outermost block of a program and hence they have file scope, persistent storage and internal linkage. If a static variable has an initializer, it is initialized before the execution of the program, only once. Unlike an auto variable, it will not be initialized each time the execution enters the block.

External variables must be defined and declared. A declaration gives the compiler information about the type of an identifier. A definition is a declaration, but a declaration is not necessarily a definition. The declaration contains the keyword extern. The definition is outside the outermost block and it reserves memory for storage. Only one definition is allowed for an external variable. They have file scope and persistent storage. If the definition of an external variable is a static declaration, it has internal linkage; otherwise, it has external linkage.

Another storage class is typedef, which is different from the other storage classes and is used for creating user defined data types. The typedef statement does not introduce a new data type, but it only renames the existing data type. It helps in easier modification of a program, when a program has to be run on different machines and it aids in giving meaningful names to the data types.

SHORT ANSWER TYPE QUESTIONS

5.1 What is a storage class?

A storage class is an attribute that changes the behaviour of a variable. It controls the lifetime, scope and linkage.

5.2 What is a block scope? or What is a local variable?

If a variable is recognized only within a particular block, it is said to have a block scope. A variable having a block scope is known as a local variable or private variable or internal variable.

5.3 What is a file scope? or What is an external variable?

If a variable is recognized throughout a program written in a single file, it is said to have a file scope. The definition of such a variable must be outside all the blocks in the program. Such a variable is also known as a global variable or public variable. The variable declared by an extern declaration is also a global variable, irrespective of the place of declaration and is known as external variable.

5.4 What is known as the side effect of a variable?

A change in the value of a variable in a block has its effect propagated to all places outside the block, whenever the same variable is used. This emerging effect is known as side effect.

5.5 What is an external linkage?

If a variable refers to the same value in every file associated with the same source program because of the external declaration, it has an external linkage.

5.6 What is an internal linkage?

If a variable refers to the same value in the same file irrespective of the position, it has internal linkage. External static variable has internal linkage.

5.7 List the types of storage classes.

1. auto 2. static 3. extern 4. register 5. typedef

5.8 What are the advantages of auto variables?

1. The same auto variable name can be used in different blocks.
2. There is no side effect by changing the values in the block.
3. The memory is economically used.
4. Auto variables have inherent protection because of local scope.

5.9 What is a register variable?

If a variable is declared with a register storage class, it is known as a register variable. The register variable is stored in the CPU registers instead of the main memory. If registers are not free, the register variable is treated as an auto variable.

5.10 Differentiate between an internal static and an external static variable.

An internal static variable is declared inside a block with static storage class whereas an external static variable is declared outside all the blocks in a file. An internal static variable has persistent storage, block scope and no linkage. An external static variable has permanent storage, file scope and internal linkage.

5.11 What is a tentative definition?

An external definition without an initializer is known as a tentative definition.

5.12 Differentiate between an external variable definition and external variable declaration.

S.No.	External Variable Definition	External Variable Declaration
1.	It creates variables.	It refers to the variable already defined.
2.	It allocates memory.	It does not allocate memory.
3.	The keyword extern is not used.	The keyword extern is used.
4.	It appears only once.	It can be declared in many places.
5.	It can be initialized.	It cannot be initialized.
6.	It must be outside all the blocks.	It can appear wherever a declaration is allowed.

5.13 What are the advantages of external storage class?

1. Persistent storage of a variable retains the latest value.
2. The value is globally available.

5.14 What are the disadvantages of external storage class?

1. The storage for an external variable exists even when the variable is not needed.
2. The side effect may produce surprising output.
3. Modification of the program is difficult.
4. Generality of a program is affected.

5.15 What is typedef?

A typedef is a storage class that creates user-defined data type. It renames the existing data type. For example,

```
typedef float REAL;
REAL area, volume;
```

5.16 What are the advantages of typedef ?

The typedef helps in easier modification when the programs are ported to another machine. A descriptive new name given to the existing data type may be easier to understand the code.

5.17 Give the scope, storage and linkage of an auto storage class.

An auto variable has temporary storage, block scope and no linkage.

5.18 Give the scope, storage and linkage of a register storage class.

A register variable has temporary storage, block scope and no linkage.

5.19 Give the scope, storage and linkage of an internal static variable.

An internal static variable has persistent storage, block scope and no linkage.

5.20 Give the scope, storage and linkage of an external static variable.

An external static variable has persistent storage, file scope and internal linkage.

5.21 Give the scope, storage and linkage of an external variable.

An external variable has persistent storage, file scope and external linkage.

5.22 Differentiate between a linker and linkage.

A linker converts an object code into an executable code by linking together the necessary built-in functions. The form and place of declaration where the variable is declared in a program determine the linkage of variable.

5.23 What is the scope of a variable?

The scope of a variable is the portion of a program in which the variable may be visible or available.

5.24 What is a forward declaration?

If a declaration of a variable appears before its definition, it is known as a forward declaration. It is applicable only for the entities having definition and declaration. Forward declaration is possible with extern variables.

5.25 What is a forward reference?

Certain entities have definitions and declarations. In such cases, if the declaration appears before its definition and the reference of such a variable is made, it is known as a forward reference. Forward reference is possible with external variables.

5.26 Where can auto be used?

The keyword auto can be used only in the declarations within a block. The keyword auto need not be used because if any variable declared within a block does not include any other storage class in its declaration, it is auto by default.

5.27 How is an existing data type renamed in C?

An existing data type is renamed using typedef statement.

5.28 What is the output of the program?

```
main()
    {
   int i=10;
printf("%d",i);
    {
     int i=20;
     printf("%d",i);
    }
printf("%d",i);
}
```

Output: 10 20 10

5.29 What is the output of the program?

```
int i_value=100;
main()
    {
     void func(void);
     i_value=50;
     printf("%d ",i value);
     func();
     printf("%d ",i value);
    }
void func()
       {
        i_value=25;
        printf("%d ",i_value);
       }
```

Output 50 25 25

5.30 What is the output of the program?

```
main()
    {
        char *p1="Name";
        char *p2;
        p2=(char *)malloc(20);
        while(*p2++=*p1++);
        printf("%s\n",p2);
    }
```

Output:

An empty string since p2 points to the character beyond the string value because of the expression *p2++.

5.31 What is the output of the program?

```
#include <stdio.h>
static int i=5;
main()
  {
    int sum=0;
    do
    {
      sum+=(1/i);
    }while(0<i--);
  }
```

Output:

Floating exception-core dumped. Post decrement uses the current value and then decrements it. When i becomes 1, 0< 1 is true. But now 1 is decremented and becomes 0 before entering the loop and the expression 1/i becomes 1/0 and hence the error.

FILL IN THE BLANKS

1. The _____ of a variable is the portion of a program in which the variable is visible.
2. A variable having file scope is known as _____ variable.
3. The auto variables have _____ scope.
4. If the definition of a variable does not contain an initializer, it is known as_____.
5. If the reference of a variable is done before its definition, it is known as a_____.
6. The storage class controls the_____, _____, and _____.
7. Every occurrence of a variable in a particular source file refers to the same value, if it has an_____.
8. _____variable declaration does not allocate memory for the variable.
9. An external variable declaration begins with the storage class specifier_____.
10. The storage class_____renames the existing data types.

TRUE OR FALSE

1. Variables having block scope have no linkage.
2. Definition of an external variable may appear many times.
3. Internal static variables have temporary storage.
4. Typedef is a storage class.
5. Register variables are treated as static variable if CPU registers are not free.
6. External variables retain their values throughout the life of a program.
7. The value of an auto variable exists though the execution leaves the block.
8. The variables declared as register type may be stored in the CPU registers instead of in the main memory.
9. An external static variable is a global variable and an internal static variable is a local variable.
10. Whenever an external variable is modified in a block the effect is propagated to all places wherever it is used.
11. External variable definition allocates memory.
12. Scope of a variable refers to the duration for which the variable retains a given value during the execution of a program.
13. More than one storage class is allowed in a declaration.
14. Register storage class is applicable only to scalar data types.

MATCH THE FOLLOWING

1	External variable	Block scope
2	Internal static variable	Executes faster
3	External static variable	External linkage
4	Register variable	Initialized once only
5	Automatic variable	No linkage

OBJECTIVE TYPE QUESTIONS

5.1 Storage class controls
(a) lifetime of a variable
(b) linkage of a variable
(b) scope of a variable
(d) all the above

5.2 Longevity of a variable refers to
(a) the duration for which the variable retains a given value during the execution of a program.
(b) the portion of a program in which the variable may be visible.
(c) internal linkage of a variable.
(d) external linkage of a variable.

5.3 Scope of a variable refers to

(a) the duration for which the variable retains a given value during the execution of a program.
(b) the portion of a program in which the variable may be visible.
(c) the value of the variable.
(d) linkage of a variable.

5.4 A variable with external linkage refers to

(a) the duration for which the variable retains a given value during the execution of a program.
(b) the same value for every occurrence of that variable in a particular file.
(c) the same value in every source file where source program spans over multiple files.
(d) block scope.

5.5 A variable with internal linkage refers to

(a) the duration for which the variable retains a given value during the execution of a program.
(b) the same value for every occurrence of that variable in a particular file.
(c) the same value in every source file where source program spans over multiple files.
(d) block scope.

5.6 A variable with no linkage refers to

(a) the duration for which the variable retains a given value during the execution of a program.
(b) the same value for every occurrence of that variable in a particular file.
(c) the same value in every source file where source program spans over multiple files.
(d) block scope.

5.7 Which is not a storage class?

(a) auto (b) struct (c) typedef (d) static

5.8 The automatic storage class has

(a) temporary storage (b) block scope
(c) persistent storage (d) options a and b

5.9 The register storage class has

(a) temporary storage (b) block scope (c) persistent storage (d) options a and b

5.10 The storage class type of internal static has

(a) persistent storage (b) block scope (c) file scope (d) options a and b

5.11 The storage class type of external static has

(a) persistent storage (b) block scope (c) file scope (d) options a and c

5.12 The storage class type of external has

(a) persistent storage (b) block scope (c) file scope (d) options a and c

5.13 Which storage class may help in faster execution?

(a) static (b) extern (c) register (d) auto

5.14 The typedef statement is used to

(a) create a new data type (b) rename the existing data type
(c) to define a storage class (d) create a structure

5.15 Identify the correct statement.

(a) `typedef int HOST` (b) `typedef int HOST;`
(c) `typedef int = HOST` (d) `typedef int = HOST;`

5.16 The typedef statement does not
(a) create new data type (b) reserve storage
(c) options a and b (d) rename the data type

5.17 Which storage class specifies local variables?
(a) auto (b) register (c) internal static (d) all the above

5.18 Which storage class specifies global variables?
(a) extern (b) external static (c) options a and b (d) typedef

5.19 External variable declaration uses
(a) the keyword external (b) the keyword extern
(c) no keyword such as extern or external (d) the keyword register

5.20 External variable definition
(a) is specified outside main() (b) reserves memory for storage
(c) is defined only once (d) all the above

5.21 External variable definition uses
(a) the keyword external (b) the keyword extern
(c) no keyword (d) the keyword auto

5.22 External variable requires
(a) declaration (b) definition (c) options a and b (d) initialization only

5.23 If the definition of an external variable does not contain an initializer, it is known as
(a) forward declaration (b) tentative definition
(c) backward reference (d) backward definition

5.24 External variable definition
(a) creates the variable (b) allocates memory
(c) does not use extern keyword (d) all the above

5.25 External variable declaration
(a) does not allocate memory (b) uses keyword extern
(c) cannot be initialized (d) all the above

5.26 The data type that cannot be renamed by typedef is
(a) function (b) pointer (c) options a and b (d) double

5.27 What are the valid C scopes of identifiers?
(a) external and local (b) project, file, and block
(c) global, file, and local (d) file and block

5.28 How is a variable accessed from another file?
(a) The global variable is referenced via the global specifier.
(b) The global variable is referenced via the extern specifier.
(c) The global variable is referenced via the auto specifier.
(d) The global variable is referenced via the pointer specifier.

5.29 Which of the following is/are achieved using typedef facility?
(a) increase the portability of code (b) write more compact code
(a) both options a and b (b) only option a
(c) only option b (d) neither option a nor b

5.30 What does invoking the C compiler accomplish?

(a) It links together object files.
(b) It creates machine code.
(c) It only provides code evaluation. You must use the linker to assemble and link programs.
(d) It interprets files at run-time.

5.31 What does extern means in a function declaration?

(a) The function has global scope.
(b) The function need not be defined.
(c) Nothing really.
(d) The function has local scope only to the file it is defined in.

5.32
```
extern int s;
int t;
static int u;
main ()
{}
```
Which of s, t and u are available to a function present in another file?

(a) only s (b) s & u (c) s, t, u (d) none

5.33 What will be the output of the following code?
```
static int i = 5;
main()
    {
int sum=0;
do
 {
  sum+=(1/i);
}while(0<i--);
    printf("sum of the series is %d",sum);
        }
```
(a) It will print the sum of the series 1/5+1/4+....+1/1.
(b) It will produce a compilation error.
(c) It will produce a runtime error.
(d) None of the above.

ANSWERS

Fill in the Blanks

1.**scope**	2.**global variable**	3.**block**	4.**tentative definition**
5.**forward reference**		6.**lifetime, scope, linkage**	
7.**internal linkage**		8.**external**	9.**extern**
10.**typedef**			

True or False

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

Match the Following

1. **3** 2. **5** 3. **4** 4. **2** 5. **1**

Objective Type Questions

1. **d** 11. **d** 21. **c** 31. **c**
2. **a** 12. **d** 22. **c** 32. **a**
3. **b** 13. **c** 23. **b** 33. **c**
4. **c** 14. **b** 24. **d**
5. **b** 15. **b** 25. **d**
6. **d** 16. **c** 26. **a**
7. **b** 17. **d** 27. **d**
8. **d** 18. **c** 28. **b**
9. **d** 19. **b** 29. **a**
10. **d** 20. **d** 30. **b**

Arrays

6

An array is a derived data type. It is an ordered sequence of finite data items of the same data type that shares a common name. The common name is the **array name** and each individual data item is known as an **element** of the array. The elements of the array are stored in the subsequent memory locations starting from the memory location given by the array name.

An array may be one-dimensional or multidimensional. A one-dimensional array can be used to represent a list of data items and it is also known as a **vector**. A two-dimensional array can be used to represent a **table** of data items consisting of rows and columns and is also known as a **matrix**. A three-dimensional array can be used to represent a collection of tables. It can be thought of as a **book** where each page is equivalent to a table and the page number represents the third dimension. This concept can be extended to arrays with more than three-dimensions also.

Like scalar variables, array variables must also be declared before use. In the declaration, the name of the array and the size of the array are given. The declaration of array facilitates the compiler to reserve enough memory for storage, based on the size mentioned and no value is assumed for the individual elements. In the declaration, the value within the square brackets [] mentions the size of the array and in the other places it represents the subscript value. The array name along with the subscripts is known as the subscripted variable and it can be used as a scalar variable after reading the array.

The individual elements are read using the I/O functions or initialized in the declaration itself. Numeric arrays have elements of int, float or double data type whereas character array (string) contains character elements in it. A one-dimensional array must have one set of square brackets [ ] whereas multidimensional arrays must have as many sets of square brackets equal to the dimensionality of the arrays. Numeric array elements cannot be read as a single entity, but the individual elements are read. Loop structure is used to simplify the coding to read/display the array. One-dimensional character arrays can be read as a whole whereas two-dimensional character arrays can be read one row at a time.

Initialization of arrays are made by enclosing the values within braces and nested braces for one-dimensional and multidimensional arrays respectively and placing a semicolon after the rightmost closing brace. This semicolon differentiates a declaration from a block containing executable statements. String constants may be used for initializing character arrays. Missing elements of partly initialized arrays are set to zero. If the size of the array is less than the number of initializers, it is an error. The size of the array in one dimensional array and the leftmost dimension of a multidimensional array may be omitted in the initialization; but care must be taken to put internal braces properly to fix the size of the dimension appropriately. The array elements can be accessed and manipulated at random, after the elements are read out or initialized.

SHORT ANSWER TYPE QUESTIONS

6.1 Define an array.

An array is an ordered sequence of finite data items of the same data type that shares a common name. The common name is the array name and each individual data item is known as an element of the array.

6.2 How are arrays declared?

One-dimensional arrays are declared with a pair of square brackets, two-dimensional arrays with 2 pairs of square brackets, and so on. For example,

```
int x[10];          /* One-dimensional array */
float a[5][5];      /*Two-dimensional array */
int  p[6][4][3];    /* Three-dimensional array */
```

6.3 How are the two-dimensional array elements stored in memory?

Two-dimensional arrays follow row major order storage representation. The elements are stored in row by row in the subsequent memory locations.

6.4 What is the difference between a[i] [j] and a[i, j] ?

The variable a[i] [j] represents the element in the j^{th} column and the i^{th} row. The expression a[i,j] is interpreted as a[j] because the comma is an operator, so the expression i, j is evaluated and j is returned.

6.5 Is it possible to have negative index?

Yes, it is possible to index with negative value provided there are data stored in these memory locations. Even though it is illegal to refer to the elements that are out of array bounds, the compiler will not produce error because C has no checks on the bounds of an array.

6.6 What is zero based addressing?

The array subscripts always start at zero. The compiler makes use of subscript values to identify the elements in the array. Since subscripts start at 0, it is said that array uses zero-based addressing.

6.7 What is array initialization?

If the elements of an array are initialized in the declaration itself, it is known as array initialization. The values are enclosed within { } and all the rows are enclosed within { and }. For example,

```
int a[4] = { 3, 5, -8, 10 };
int x[3][2] = {{20, 5}, {-8, 5), {5, 7}};
```

The elements that are not initialized are set to zero.

6.8 What is a string?

A string is a sequence of characters ending with NUL. It can be treated as a one-dimensional array of characters terminated by a NUL character.

6.9 Differentiate between gets() and scanf() using %s conversion specification?

Even though both are used for reading strings, gets() reads a string until a new line or EOF is read. The function scanf() reads a string until a white space character is read. NUL is automatically appended to string while reading by both functions.

6.10 How to initialize a character array?

For example,

1. `char t[] = {'a', 'e', 'i', 'o', 'u', '\0'};` /* One by six array */

or

```
char t[] = "aeiou"
```

2. `char winter[3][10] ={"OCT", "NOV", "DEC"};`
3. `char v[][5]={{'a'),{'e'),{'i'},('o'),{'u'}};` /* Five by five array */

6.11 What is a subscripted variable?

In an array, the elements are accessed using the subscripted variables or array variables.

```
int a[10];
```

Here, a[0], a[1],..., a[9] are subscripted variables.

6.12 What are the characteristics of arrays in C?

1. An array holds elements that have the same data type.
2. Array elements are stored in subsequent memory locations.
3. Two-dimensional array elements are stored row by row in subsequent memory locations.
4. Array name represents the address of the starting element.
5. Array size should be mentioned in the declaration. Array size must be a constant expression and not a variable.

6.13 What is the allowed data type for subscript?

The subscript must be an integer value or an integer expression.

6.14 Why is it necessary to give the size of an array in an array declaration?

When an array is declared, the compiler allocates a base address and reserves enough space in memory for all the elements of the array. The size is required to allocate the required space and hence size must be mentioned.

6.15 How to get the size of an array in a program?

```
int a[10];
printf("%d\n", sizeof(a)/sizeof(a[0]));
```

The above code prints the size of the integer array a[10].

6.16 When does the compiler not implicitly generate the address of the first element of an array?

Whenever an array name appears in an expression such as,

1. array as an operand of the sizeof operator,
2. array as an operand of & operator, or
3. array as a string literal initializer for a character array,

then the compiler does not implicitly generate the address of the first element of an array.

6.17 Does the following code work? Justify.

```
const int n = 10; int x[n];
```

No, the code gives an error. The const qualifier really means read-only; it is a run-time object which cannot be assigned to. The value of const qualifier object is not a constant expression. The array size must be a constant expression because the size is needed by a compiler to allocate space. Run-time object yields the value only at the execution time. Hence, the above code does not work.

6.18 What is the output of the program?

```
main()
  {
   int a[]=(0,0X4,4,9};
   int i=2;
   printf("%d %d",a[i],i[a]);
  }
```

Output:
4 4 /* a[i] and i[a] are identical */

6.19
```
#define void int
int i=300;
void main()
       {
         int i=200;
           {
             int i=100;
             printf("%d ",i);
                 }
             printf("%d ",i);
           }
```
What is the output of the above program?

Output:
100 200

6.20 How many bytes of memory will the following arrays need?

(a) char s[80]; /*80 */
(b) char s[80][10]; /*800 */
(c) int d[10]; /* 10*sizeof(int) */
(d) float d[10][5]; /* 50 * sizeof(float) */

6.21 What is the output of the program?
```
main()
    {
      int i,j;
      int mat[3][3] ={1,2,3,4,5,6,7,8,9};
      for (i=2;i>=0;i--)
        for (j=2;j>=0;j--)
          printf("%d ",*(*(mat+j)+i));
    }
```

Output:
9 6 3 8 5 2 7 4 1

6.22 What is the output of the program?
```
#include <stdio.h>
main()
    {
     char s1[]="Ramco";
     char s2[]="Systems";
     s1=s2;
     printf("%s",s1);
    }
```

Output:
Compilation error since s1 is the array name and not a variable.

6.23 What is the output of the program?

```
#include <stdio.h>
main()
    {
      int a[10];
      printf("%d",((a+9) + (a+l)));
    }
```

Output:
Error(invalid pointer arithmetic).

6.24 What is the output of the program?

```
#include <stdio.h>
main()
     {
       char numbers[5][6]={"Zero","One","Two","Three","Four"};
       printf("%s is %c",&numbers[4][0],numbers[0][0]);
     }
```

Output:
Four is Z

6.25 Write a program to reverse a string using the operator.^

```
#include <stdio.h>
int main()
{
  char s[20];
  void revstr( char * );
  printf("Enter the string to be reversed\n");
  scanf("%s", s);
  revstr(s);
  printf("Reverse string is %s \n", s);
  return 0;
}
void revstr(char *s)
  {
  int length = strlen(s);
  int i,j;
  if(length > 0)
   for(i=0,j=length-1; i<j;i++,j--)
    s[i] ^= s[j] , s[j] ^= s[i], s[i] ^=s[j];
  }
```

Output
Enter the string to be reversed
madam
Reverse string is madam

FILL IN THE BLANKS

1. A vector is a _____ array.
2. Array name represents the _____ of the starting element.
3. _____storage representation is used in a matrix.
4. Number of elements in a two-dimensional array having 5 rows and 3 columns is____.
5. ANSI C compiler supports at least_____dimensions of array.
6. Missing elements of partially initialized arrays are set to_____.
7. The array subscript must always start at_____.
8. In an array declaration, the value enclosed in [] specifies the_____of the array.
9. The_____of an array are stored in the consecutive memory locations.
10. The initializer **char s[]= {‘ a’ ,‘ e’ , ‘i’ , ‘o’ , ‘u’ };** is different from **char s[]= “aeiou” ;** since the first assignment does not include_____character.
11. Whenever an error occurs or end of file reaches the function gets() returns_____.
12. Whenever an error occurs the function puts() returns_____.
13. The function scanf() is used to read a string by using a conversion specification_____.
14. The function printf() is used to display a string by using a conversion specification_____.
15. In a one-dimensional array declaration, the value within_____mentions the size of the array and in other places it represents the subscript value.

TRUE OR FALSE

1. Array is an ordered sequence of finite data items of different data types.
2. Matrix is an array of one-dimensional array.
3. Arrays must be declared before use.
4. If the size of an array is less than the number of initializers, the size grows dynamically.
5. C has no checks on array bounds.
6. A single array can be used to represent both integer and real numbers.
7. The values of an array **b** can be assigned to another array **a** by the statement **a = b**;.
8. A list of strings can be stored in within a two dimensional array.
9. In a two-dimensional array **a**, the individual elements **a[i]j]** and **a[ij]** are the same.
10. The size of an array which is initialized with a string constant is equal to the number of characters in the string.
11. Numeric array elements can be read as a single entity.
12. One-dimensional character array can be read as a single entity.
13. Array name and subscripted variables are different.
14. The expressions x[5] and 5[x] are identical.

MATCH THE FOLLOWING

1	Array	Subscripted variable
2	Array elements	Allocation of memory at compilation time
3	Declaration of array	String
4	Character array	May be omitted in the declaration
5	Leftmost dimension in a multidimensional array	Derived data type

OBJECTIVE TYPE QUESTIONS

6.1 Array is used to represent

(a) a list of data items of integer data type
(b) a list of data items of real data type
(c) a list of data items of different data type
(d) a list of data items of same data type

6.2 Array name is

(a) an array variable
(b) a keyword
(c) a common name shared by all elements
(d) not used in a program

6.3 One-dimensional array is known as

(a) vector (b) table (c) matrix (d) an array of arrays

6.4 The array elements are represented by

(a) index values (b) subscripted variables
(c) array name (d) size of an array

6.5 Array elements occupy

(a) subsequent memory locations
(b) random location for each element
(c) varying length of memory locations for each element
(d) no space in memory

6.6 The address of the starting element of an array is

(a) represented by subscripted variable of the starting element
(b) cannot be specified
(c) represented by the array name
(d) not used by the compiler

6.7 Identify the wrong statement.

(a) Subscripts are also known as indices.
(b) Array variables and subscripted variables are same.
(c) Array name and subscripted variables are different.
(d) Array name and subscripted variables are same.

6.8 Array subscripts in C always start at
(a) –1 (b) 0 (c) 1 (d) any value

6.9 Identify the correct declaration.
(a) `int a[10][10];` (b) `int a[10,10];`
(c) `int a(10)(10);` (d) `int a(10,10);`

6.10 Maximum number of elements in the array declaration **int x[10]**; is
(a) 9 (b) 10 (c) 11 (d) undefined

6.11 The elements of the following array x are

float x[5];

(a) x[0], x[1], x[2], x[3], x[4] (b) x[1], x[2], x[3], x[4], x[5]
(c) x(0), x(1), x(2), x(3), x(4) (d) x(1), x(2), x(3), x(4), x(5)

6.12 Maximum number of elements in the declaration **int y[5][8];** is
(a) 28 (b) 32 (c) 35 (d) 40

6.13 Two-dimensional array elements are stored in
(a) column major order (b) row major order
(c) both options a and b (d) random order

6.14 Two-dimensional array elements are stored
(a) row by row in the subsequent memory locations
(b) column by column in the subsequent memory locations
(c) row by row in scattered memory locations
(d) column by column scattered memory locations

6.15 ANSI C recommends a compiler to support at least____dimensions of an array.
(a) 4 (b) 5 (c) 6 (d) 7

6.16 Array declaration
(a) requires the number of elements to be specified
(b) does not require the number of elements to be specified
(c) assumes default size as 0
(d) is not necessary

6.17 Identify the wrong expression given **int a[10];**
(a) a[–1] (b) a[10] (c) a[0] (d) ++ a

6.18 What is the value of a[0][2] in **int a[3][4] ={{1,2}, {4,8,15}}?**
(a) 4 (b) 2 (c) 0 (d) not defined

6.19 To initialize a 5 element array all having value 1 is given by
(a) `int num[5] = {1};`
(b) `int num[4] = {1,1,1,1,1};`
(c) `int num[5] = {1,1,1,1,1};`
(d) `int num[ ] = {1};`

6.20 To initialize a 5 element array all having value 0 is given by
(a) `int num[5] = {0};`
(b) `int num[5] = {0, 0, 0, 0, 0,};`
(c) options a and b
(d) `int num[5] = (1);`

6.21 The declaration float f[ ][3] = {{1.0}, {2.0}, {3.0}}; represents
(a) a one-by-three array (b) a three-by-one array
(c) a three-by-three array (d) a one-by-one-array

6.22 A char array with the string value "aeiou" can be initialized as
(a) char s[ ] = {'a', 'e', 'i','o', 'u'}; (b) char s [ ] = "aeiou" ;
(c) char s[ ] = {'a', 'e', 'i', 'o', 'u', '\0'}; (d) options b and c

6.23 If the size of an array is less than the number of initializers,
(a) the extra values are neglected (b) it is an error
(c) the size is automatically increased (d) the size is neglected

6.24 Identify the correct statement.
(a) Float array can be read as a whole. (b) Integer array can be read as a whole
(c) Char array can be read as a whole. (d) Double array can be read as a whole.

6.25 Identify the correct statement.
(a) Float array can be displayed by using single printf() without using a loop.
(b) Integer array can be displayed by using single printf() without using a loop.
(c) Char array can be displayed by using single printf() without using a loop.
(d) Double array can be displayed by using single printf() without using a loop.

6.26 Missing elements of partly initialized arrays, are
(a) set to zero (b) set to one (c) not defined (d) invalid

6.27 The declaration of array main() { int a[10]; – – –}
(a) reserves the space required for 11 elements
(b) reserves space for 9 elements
(c) does not initialize any element
(d) initializes the values of all elements to 0

6.28 The value within the [ ] in an array declaration specifies
(a) subscript value (b) index value
(c) size of an array (d) value of the array element

6.29 The value within the [ ] in an array variable specifies
(a) subscript value (b) size of the array
(c) value of the array element (d) array bound

6.30 In a multi-dimensional array with initialization
(a) The rightmost dimension may be omitted.
(b) The leftmost dimension may be omitted.
(c) Nothing must be omitted.
(d) All may be omitted.

6.31 When should an array be used?
(a) When we need to hold variable constants.
(b) When we need to hold data of the same type.
(c) When we need to obtain automatic memory cleanup functionality.
(d) When we need to hold data of different types.

6.32
```
x[2] = 5;
2[x] = 5;
```
Are x[2] and 2[x] identical in the sample code above? Why or why not?
(a) No. Both variable assignments have invalid syntax.
(b) No. x[2] is correct, but 2[x] is invalid syntax.
(c) Yes. Both are identical because they are resolved to pointer references.
(d) No. 2[x] is correct, but x[2] is invalid syntax.

6.33
```
main()
    {
     int a[100], i;
     for( i = 1 ; i <= 100; ++i)   { ... ; }
    }
```
(a) The above loop statement is incorrect since the last valid subscript of a is 99
(b) The above loop statement is incorrect since i is not initialized to 0.
(c) The above loop statement is correct.
(d) The above loop statement is incorrect since i is pre-incremented.

6.34
```
char sub [ 10 ] = ????;
```
Which of the following statements cannot be used to replace the ???? in the above syntax to initialize sub with the string "Maths"?
(a) { "Maths" } (b) { 'M', 'a', 't', 'h', 's', '\0' }
(c) { 'M' "aths" } (d) { "Mat" "hs" }

6.35
```
main()
    {
      const int size = 5;
      int i, n, line[size];
      for (i = 0; i< n; i++)
        {
         line[ i ] = i;
        }
     }
```
What will happen when the sample code fragment above is executed?
(a) The array will be initialized with the numbers 0 through 39.
(b) The code will not compile.
(c) The code will compile with warnings.
(d) The code will compile, but not link.

6.36 Which of the following macros would properly return the number of elements in an array (not a pointer, an actual array)?
(a) #define NUM_ELEM(x) (sizeof(x)/sizeof(x[0]))
(b) #define NUM ELEM(x) (sizeof(x))
(c) #define NUM ELEM(x) (sizeof(x[0])/sizeof(x))
(d) #define NUM ELEM(x) (sizeof(x)/sizeof(x[O]))
(e) #include <stdio.h>

6.37
```
main()
    {
      char sl[100]; char s2[100];
      gets( sl );
      fgets( s2, sizeof(s2), stdin);
      printf( "%d\n", strlen( sl ) - strlen( s2 ));
    }
```
What will be printed when the above code is executed and the string "abcd" is entered twice on stdin?

(a) –1 (b) 0 (c) 1 (d) 4

6.38 `int booklet[3][2][2] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};`

What value does booklet[2] [1][0] in the sample code above contain?

(a) 5 (b) 7 (c) 9 (d) 11

6.39
```
int matrix[10] [4] ;
```
A. `memset(matrix, -1, sizeof(matrix));`
B. `memset(matrix, 0, sizeof(matrix));`
C. `memset(matrix, 1, sizeof(matrix));`

Given the above choices, which one will fill the matrix with an integer value?

(a) only A (b) only B (c) A and B (d) A, B and C

6.40
```
main()
  {
    char s[] = "Focal Point";
    printf( "%d\n", ???? );
  }
```
Which of the following could replace the ???? in the code above to print the index of the first occurrence of the letter o in s (in this case the value would be 1) ?

(a) strchr(s, 'o') - s (b) strchr(s, "o") (c) strchr(s, 'o') (d) strchr(s, "o") – s

6.41
```
main()
  {
    char s[6]= " HELLO";
    printf("%s",s[5]);
  }
```
What is the output of the above program?

(a) 0 (b) 48 (c) nothing (d) unpredictable

6.42 What is the result of the following declaration?
```
int A[] ={1,2,3,4,5};
&A[5] - &A[1] ;
```
(a) 8 (b) –4 (c) 4 (d) –8

6.43
```
char name[] = {'n', 'a', 'm', 'e'};
printf ("name = %s\n", name);
```
What will be the output?

(a) name = name (b) name = name followed by junk characters
(c) name = \nname (d) option a or b

6.44 What would be the result of the following program?

```
main()
  {
    char p[]="string";
    char t;
    int i,j;
    for(i=0,j=strlen(p);i<j;i++)
      {
        t=p[i];
        p[i]= p[j-i];
        p[j-i]=t;
      }
    printf("%s",p);
  }
```

(a) will print:string
(b) will not print anything since p will be pointing to a null string
(c) will print:gnirts
(d) will result in a complication error

6.45 What is the output of the program?

```
main()
    { int rows=3,columns=4;
      int a[rows][columns]={1,2,3,4,5,6,7,8,9,10,11,12};
      i=j=k=99;
      for(i=0;i<rows;i++)
        for(j=0;j<columns;j++)
          if(a[i] [j]<k) k=a[i][j];
          printf("%d\n",k); }
```

(a) syntax error (b) runtime error (c) 1 (d) none of the above

6.46 What is the value of i in the following code?

```
main()
  {
   int i=strlen("BLUE")+strlen("purple")/strlen("red")-strlen("green");
   printf("%d",i);
  }
```

(a) –2 (b) 1 (c) –1.666667 (d) –1

6.47 What is the output of the following code?

```
int cap(int);
main()
  {
    int n; n=cap(6);
    printf("%d",n);
  }
int cap(int n)
{
  if(n<=1) return 1;
```

```
  else return(cap(n-3)+cap(n-1));
}
```

(a) 7 (b) 8 (c) 9 (d) 10

ANSWERS

Fill in the Blanks

1. **one-dimensional array**		2. **address**		3. **row major order**
4. **15**	5. **6**	6. **zero**	7. **zero**	
8. **size**	9. **elements**	10. **NUL**	11. **NULL**	
12. **EOF**	13. **%s**	14. **%s**	15. **[ ]**	

True or False

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

Match the Following

1. **5**	2. **1**	3. **2**	4. **3**	5. **4**

Objective Type Questions

1. **d**	11. **a**	21. **c**	31. **b**	41. **c**
2. **c**	12. **d**	22. **d**	32. **c**	42. **c**
3. **a**	13. **b**	23. **b**	33. **c**	43. **d**
4. **b**	14. **a**	24. **c**	34. **c**	44. **b**
5. **a**	15. **c**	25. **c**	35. **b**	45. **a**
6. **c**	16. **a**	26. **a**	36. **d**	46. **b**
7. **d**	17. **d**	27. **c**	37. **a**	47. **c**
8. **b**	18. **c**	28. **c**	38. **d**	
9. **a**	19. **c**	29. **a**	39. **c**	
10. **b**	20. **c**	30. **b**	40. **a**	

Functions

7

A program can be used for solving a simple problem or a large and complex problem. Programs solving simple problems are easier to understand and identify mistakes, if any, in them whereas, complex programs are usually larger. If a program is large, it is difficult to understand the steps involved in it. Hence, it is subdivided into a number of smaller programs called **subprograms** or **modules**. Each subprogram specifies one or more actions to be performed for the larger program. Such subprograms are called as **functions / subroutines** in FORTRAN and **functions / procedures** in PASCAL. But in C, subprograms are called only as **functions**. Large C programs are divided into many smaller functions. C has been designed in such a way that its functions are efficient and easy to use.

Larger programs usually consist of several subprograms or modules. Each module may be stored in a separate file called **source module** or **source file**. These are the input files for the compiler. It is convenient to name the source module reflecting the name of the function in it. Thus, it is easier to identify any function in a set of source files. The source modules are compiled separately and the **object files** are created. Then these object files are combined into one **executable** program.

Modular programming is achieved with the help of functions. Functions are helpful to hide the internal details of a program and improve its readability. Debugging is also easy by using functions in a program. To make use of these functions, the functions must be defined and declared.

Function is classified as user-defined or built-in function, based on the nature of creation. It is classified as function returning single value and function returning no value on the basis of the returned value of the function. With respect to the function call, the function may be recursive or non-recursive. The defined function is invoked to perform its action, which is called as function call. By default function has **extern** storage class but it can also have **static** storage class. The function main( ) is the starting function (startup code) of a C program. C makes use of **call by value** method of parameter passing. This method of parameter passing does not allow the modified values of the parameters in the called function to be available in the calling function.

SHORT ANSWER TYPE QUESTIONS

7.1 What is modular programming?

If a program is large, it is subdivided into a number of smaller programs that are called modules or

subprograms. If a complex problem is solved using more modules, this approach is known as modular programming.

7.2 What is a function?

A large program is subdivided into a number of smaller programs or subprograms. Each subprogram specifies one or more actions to be performed for the larger program. Such subprograms are called functions.

7.3 Classify the functions based on the nature of creation.

1. User-defined functions 2. Built-in functions

7.4 What is an argument?

An argument is an entity used to pass data from the calling function to a called function.

7.5 What is the purpose of main() function?

The function main() invokes other functions within it. It is the first function to be called when the program starts execution.

7.6 Differentiate between formal arguments and actual arguments.

Formal arguments are the arguments available in the function definition. They are preceded by their own data types. Actual arguments are available in the function call. These arguments are given as constants or variables or expressions to pass the values to the function.

7.7 What are built-in functions?

The functions that are predefined and supplied along with the compiler are known as built-in functions. They are also known as library functions.

7.8 What is the parameter passing mechanism used in C?

C uses call by value method of parameter passing. Any change in the value of the formal parameter within the function is not visible in the calling function.

7.9 What is a function definition?

A function definition has a function header and a function body.

Format:

```
return_type function_name( formal arguments )
               {
                 statements;
                 return statement;
               }
```

For example,

```
float max( float x, float y)
         {
           if (x > y)
           return x;
          }
```

7.10 What is a function prototype?

A function must be declared before use. Such a declaration is known as function declaration or function prototype. It ends with semicolon.

Format:

```
return_type function_name( parameters );
```

For example,

```
float max(float x, float y);
```

7.11 Is it possible to have more than one main() function in a C program? Why?

The function main() can appear only once in a C program, because execution commences from the first statement in the main() function. If there is more than one main() function, there will be a confusion while commencing the execution. Hence, only one main() function is allowed.

7.12 Differentiate between function definition and function declaration.

S.No.	Function Definition	Function Declaration
1.	There is no semicolon at the end of the header.	There is a semicolon at the end of the declaration.
2.	The function body follows the header.	There is no function body.

7.13 What is a dummy function?

A function that performs no action is known as a dummy function. It is a valid function. Dummy function may be used as a place-holder, which facilitates adding new functions later on. For example,

```
dummy () { }
```

7.14 What is recursion?

If a function calls itself in the function body of its function definition, it is known as recursion. Indirect recursion is possible if a function, say **func** calls another function which in turn calls **func**.

7.15 What are the advantages of recursion?

1. Easier understanding of program logic.
2. Writing compact codes.
3. Implementing recursively defined data structures.

7.16 Give the features of main().

1. It is the starting function.
2. It returns an int value to the environment that called the program.
3. Recursive call is allowed for main() also.
4. It is a user-defined function.
5. Program execution ends when the closing brace of the function main ( ) is reached.

7.17 What are the advantages of functions?

1. Debugging is easier.
2. It is easier to understand the logic involved in the program.
3. Testing is easier.
4. Recursive call is possible.
5. Irrelevant details in the user point of view are hidden in functions.
6. Functions are helpful in generalizing the program.

7.18 What are the storage classes supported by functions?

The functions support only

- static and
- extern

storage classes. By default, function assumes extern storage class. Functions have global scope.

For example,

```
extern float func()
```

and

```
float func()
```

make no difference.

7.19 What are the storage classes permitted in the function parameters?

Only register or auto storage class is allowed in the function parameters.

7.20 How are functions classified based on function call?

1. Recursive 2. Non recursive

7.21 What for is a return statement used?

1. A return statement is used for returning a value from function by appending an expression to return.
2. It also helps in earlier exit from a function.

7.22 Name any four string manipulation library functions.

1. strcpy(sl, s2) : Copying string s2 to string sl.
2. strcmp(sl, s2) : Comparing strings. If both are equal it returns 0.
   If s1< s2, it returns negative value.
   If s1> s2, it returns positive value.
3. strlen(s1) : Length of string s l.
4. strcat(sl, s2) : Appending string s2 to end of string sl.

7.23 Give any four trigonometric functions defined in <math.h> header file.

sin(x), cos(x), tan(x) and asin(x).
The angle x is expressed in radians of double data type.

7.24 Name any four functions in <ctype.h> header file.

isupper(c) islower(c) toupper(c) tolower(c)
c is a char data type.

7.25 What are the values returned by ceil(x) and floor(x)?

ceil(x) : It returns the value rounded up to the next higher integer. For example, ceil(3.1) is 4.
floor(x): It returns the value rounded down to the next lower integer. For example, floor(3.9) returns 3.

7.26 Does C permit **call by reference**? Justify.

No, C permits only **call by value** method of parameter passing. The effect of call by reference may be achieved by passing parameters as pointers.

7.27 Give the name of the standard library function for the following

(a) string length : strlen(s)
(b) string compare : strcmp(sl, s2)
(c) string copy : strcpy(sl, s2)
(d) string concatenation : strcat(sl, s2)

7.28

```
main()
   {
     func();
     func();
   }
func()
   {
     static int   i = 10;
```

```
    printf("%d ",i)
    i ++;
  }
```

What is the value of i if the function is called twice?

Output: 10 11

7.29 What is the output generated by the following program?

```
main()
  {
    int n=10;
    printf("%d",func(n));
  }
int func(int n)
    {
      if(n>0) return(n+func(n-2));
      else return 0;
    }
```

Output:
30

7.30 What is the output of the program?

```
#include <stdio.h>
int newval(int);
main(void)
          {
            int ia[]={12,24,45,0};
            int i;
            int sum=0;
            for(i=0;ia[i];i++)
              {
                sum+=newval(ia[i]);
              }
            printf("Sum=%d",sum);
          }
int newval(int x)
     {
         static int div=1;
         return(x/div++);
     }
```

Output:
Sum= 39

7.31 What is the output of the program?

```
#include <stdio.h>
double dbl=20.4530,d=4.5710,dblvar3;
main(void)
          {
            double dbln(void);
            dblvar3=dbln();
```

```
            printf("%.2f\t%.2f\t%.2f\n", db1,d,db1var3);
        }
    double dbln(void)
        {
         double dblvar3;
         dbl=dblvar3=4.5;
         return(dbl+d+dblvar3);
        }
```

Output:
4.50 4.57 13.57

7.32 What is the output of the program?

```
#include <stdio.h>
int SumElement(int *,int);
main(void)
    {
     int x[10];
     int i=10;
     for(; i;)
      {
        i--;
        *(x+i)=i;
      }
     printf("%d",SumElement(x,10));
    }
int SumElement(int array[],int size)
      {
       int i=0;
       float sum=0;
       for(;i<size;i++)
       sum+=array [i];
       return sum;
      }
```

Output:
45

7.33 What is the output of the program?

```
#include <stdio.h>
main(void)
    {
     void print(void);
     print();
    }
void fl(void)
     {
      printf("\nfl():"),
```

```
    }
#include "32.c"
void print(void)
     {
      extern void f1(void);
      f1();
     }
static void f1(void)
      {
       printf("\n static f1().");
      }
static int i=50;
int print(int i);
main(void)
      {
        static int i=100;
        while(print(i))
            {
                printf("%d\n",i);
                i--;
            }
       }
int print(int x)
         {
          static int i=2;
          return (i--);
         }
```

Output:

100 99

7.34 What is the output of the program?

```
main()
   {
    int i = 2;
    twice (2);
    printf ("%d", i);
   }
    twice (int i)
       {
        i = i*2;
       }
```

Output:

2

7.35 What is the output of the program?

```
main()
    {
     fl();
    }
  fl()
   { f(3); }
  f(int t)
   {
    switch(t)
         {
           case 2:c=3;
           case 3:c=4;
           case 4:c=5;
           case 5:c=6;
           default:c=0;}
             printf("%d ",c);
         }
```

Output:
0

7.36 What is the output of the program?

```
main()
    {
      int *fl();
      f();
    }
int *fl()
      {
        int a=5;
        return &a;
      }
f()
{
 int *b=f1();
 int c=*b;
 printf("%d ",c);
}
```

Output:
5

7.37 Write a program to find the day of week, for the given date.
Tomshiko Sakamota's code to find the day of the given date, which is given below.

```
main( )
       {
        int y,m,d;
```

```
        clrscr( );
        y = 2000;m = 12;d=23;
       day(y,m,d);
       }
void day(int y,int m,int d)
    {
       int h;
      static int t[] = {0,3,2,5,0,3,5,1,4,6,2,4};
      y -= m<3;
      h=(y + y/4 - y/100 + y/400 + t[m-1] + d) % 7;
      printf("%d" ,h);
    }
```

FILL IN THE BLANKS

1. _____are used to communicate the values between the functions.
2. If a function returns no value, ___ is specified in the place of return type.
3. A function returning a_____may be used in an expression.
4. A function assumes_____data type as return type if no return type is specified in the function header.
5. Header files include_____functions.
6. Function names starting with underscores are meant for_____function names.
7. Function declaration ends with a_____.
8. Parameters can use_____storage class.
9. A function uses_____parameter passing mechanism.
10. The storage classes_____and_____are allowed in functions and_____is assumed to be the default one.
11. _____functions are predefined and supplied along with the compiler.
12. The function_____is the first calling function in any C program.
13. An identifier(other than keywords) followed by an open parenthesis is recognized as a _____ by the compiler.
14. Function_____is mandatory for all functions.
15. All the string manipulation functions have been included in the header file_____.
16. The function main() can use only_____data type as its return data type.

TRUE OR FALSE

1. Linker converts the object code into executable code.
2. The function main() is a built-in function.
3. The function memcpy() is faster than the function memmove().

4. One function cannot be defined within another function definition.
5. Recursive call is not allowed for main() function.
6. A function can return more than one value.
7. There is a limit on the number of times a function is called.
8. A function can be called from more than one place within a program.
9. The function call floor(5.9) returns 5 and floor(5.1) also returns 5.
10. The function call ceil(6.9) returns 7 and ceil(6.1) returns 6.
11. Two functions can have the same name in a single program.
12. A function with no action is an invalid function.

MATCH THE FOLLOWING

1	Function	Calling function
2	Startup code	Called function
3	Formal parameter	Subprogram
4	Actual parameter	No action
5	Dummy function	main()

OBJECTIVE TYPE QUESTIONS

7.1 The program execution starts from

(a) the function which is first defined (b) main() function
(c) the function which is last defined (d) the function other than main()

7.2 How many main() functions can be defined in a C program?

(a) 1 (b) 2 (c) 3 (d) any number of times

7.3 A function is identified by an open parenthesis following

(a) a keyword (b) an identifier other than keywords
(c) an identifier including keywords (d) an operator

7.4 A function with no action

(a) is an invalid function (b) produces syntax error
(c) is allowed and is known as dummy function (d) returns zero

7.5 Parameters are used

(a) to return values from the called function (b) to send values from the calling function
(c) options a and b (d) to specify the data type of the return value

7.6 Identify the correct statement.

(a) A function can be defined more than once in a program.
(b) Function definition cannot appear in any order.
(c) Functions cannot be distributed in many files.
(d) One function cannot be defined within another function definition.

7.7 The default return data type in function definition is
(a) void (b) int (c) float (d) char

7.8 The parameters in a function call are
(a) actual parameters (b) formal parameters
(c) dummy parameters (d) optional

7.9 The parameters in a function definition are
(a) actual parameters (b) formal parameters
(c) dummy parameters (d) optional

7.10 The parameters passing mechanism used in C is
(a) call by reference (b) call by name (c) call by value (d) options a and b

7.11 The storage class that can precede return data type in function declaration is
(a) extern (b) static (c) options a and b (d) register

7.12 Recursive call results when
(a) a function calls itself
(b) a function1 calls another function, which in turn calls the function1
(c) options a and b
(d) a function calls another function

7.13 The main() function calls in a C program
(a) allows recursive call (b) does not allow recursive call
(c) is optional (d) is a built-in function

7.14 The function main() is
(a) a built-in function (b) a user-defined function
(c) optional (d) all the above

7.15 The storage class allowed for parameters is
(a) auto (b) static (c) extern (d) register

7.16 Functions are assigned by default
(a) auto storage class (b) static storage class
(c) extern storage class (d) register storage class

7.17 Functions have
(a) file scope (b) local scope (b) block scope (d) function scope

7.18 The function defined in math.h file for returning the arc tangent of x is
(a) tan_ 1(x) (b) atan(x) (c) tanh(x) (d) arctan(x)

7.19 The function cell(x) defined in math.h
(a) returns the value rounded down to the next lower integer
(b) returns the value rounded up to the next higher integer
(c) the next higher value
(d) the next lower value

7.20 The function floor(x) in math.h
(a) returns the value rounded down to the next lower integer
(b) returns the value rounded up to the next higher integer
(c) the next lower value
(d) the next lower value

7.21 The function strcpy(sl, s2) in string.h
(a) copies s1 to s2 (b) copies s2 to s1
(c) appends s1 to end of s2 (d) appends s2 to end of s1

7.22 The function strcat(sl, s2) in string.h
(a) copies s1 to s2 (b) copies s2 to s 1
(c) appends s1 to end of s2 (d) appends s2 to end of s1

7.23 The function strcmp(sl, s2) returns zero
(a) if s1 is lexicographically less than s2
(b) if s1 is lexicographically greater than s2
(c) if both s1 and s2 are equal
(d) if s1 is empty string

7.24 The function toupper(ch) in ctype.h
(a) returns the upper case alphabet of ch
(b) returns the lower case alphabet of ch
(c) returns upper case if ch is lower case, and lower case if ch is upper case
(d) is a user-defined function

7.25 The function tolower(ch) in ctype.h
(a) returns the uppercase alphabet of ch
(b) returns the lowercase alphabet of ch
(c) returns uppercase if ch is lowercase, and lowercase if ch is uppercase
(d) is a user-defined function

7.26 What function must all C programs have?
(a) start() (b) main() (c) return() (d) exit()

7.27 In C, which of the following statement(s) is (are) TRUE about the way the arguments are passed?
1. The order of evaluation of arguments is compiler dependant.
2. Arguments are passed by reference.
(a) both options 1 and 2 (b) only option 1
(c) only option 2 (d) neither option 1 nor 2

7.28 Which of the following statements in C is/are TRUE?
1. Two functions can have the same name in a single program.
2. Function calls can be recursive.
(a) both options 1 and 2 (b) only option 1
(c) only option 2 (d) neither option 1 nor 2

7.29 Which of the following would compute the square of x in C?
(a) pow(2, x); (b) pow(x, 2) (c) x**2; (d) power(x, 2);

7.30 All standard C library <math.h> functions return what data type?
(a) decimal (b) float (c) double (d) int

7.31
```
void show();
main()
       {
         show();
```

```
}
void show( char *s )
        {
  printf( "%s\n", s );
}
```

What will happen when the above code is compiled and executed using a strict ANSI C compiler?

(a) It will compile and nothing will be printed when it is executed.
(b) It will compile, but not link.
(c) The compiler will generate an error.
(d) The compiler will generate a warning.

7.32
```
void display(int x)
      {
        if (x ! = 0)
        {
          display(x / 16);
          putchar("0123456789ABCDEF"[x % 16]);
        }
       else
        putchar('\n');
      }
```

What will be output if the above function is called with x = 1234?

(a) 1234 (b) 4321 (c) 4D2 (d) It will not compile

7.33
```
int i=3, a=1, b=1;
void func()
      {
        int a=0;
        static int b = 0;
        a++; b++;
      }
int main()
    {
      for ( i=0; i < 5; i++)
        {
          func ();
          a++; b++;
        }
      printf ("a=%d b=%d\n", a, b);
    }
```

What will be printed from the sample code above?

(a) a=0 b=6 (b) a = 5=5 (c) a=5 b=6 (d) a=6 b=6

7.34 long factorial (long x)

```
{
  ????
  return x * factorial(x - 1);
}
```

What would you replace the ???? with, to make the function shown above, return the correct answer?

(a) if (x == 0) return 0;
(b) if (x == 0) return 1;
(c) if (x < = 1) return 1;
(d) return 1;

7.35 `if ( x ? y : z) do Something();`

Referring to the sample above, which statement correctly identifies when y is evaluated?

(a) y is evaluated only when x = = 1.
(b) y is evaluated only when x > = 1.
(c) y is evaluated only when x!= 0.
(d) y is evaluated only when x== 0.

7.36
```
const int varl = 1;
static int var2 = 2;
void func()
        {
          int var3 = 3;
          static int var4 = 4;
        }
```

Which of the above variables exist after the function returns and cannot be accessed from another file?

(a) var2 only
(b) var4 only
(c) var2 and var4 only
(d) var2, var3, and var4 only

7.37
```
void func(int x)
        {
          if (x > 0) func(--x);
          printf("%d, ", x);
        }
int main()
        {
          func(5);
          return 0;
        }
```

What will the above sample code produce when executed?

(a) 0, 1, 2, 3, 4, 5
(b) 0, 0, 1, 2, 3, 4
(c) 4, 3, 2, 1, 0
(d) 5, 4, 3, 2, 1, 0,

7.38
```
#include <stdio.h>                /* show.c */
void show()
        {
          printf( "Water\n" );
        }
```

```
/* main.c */
#include <stdio.h>
static void show()
              {
                printf("Drink\n");
              }
int main()
       {
        show();
        return 0;
       }
```

Referring to the program given by the 2 source files defined in the code above, what will happen when you try to build and run the program?

(a) It will print Water.
(b) It will print Drink.
(c) It will print:
Drink
Water
(d) It will not compile.

7.39
```
char varl [10] ;
char var2[5] = "Angel";
strcpy( varl, var2 );
```

What will happen when the above code is executed?

(a) The linker will reject this as an array overflow error.
(b) The variable varl and var2 will contain the string "Angel" and no problems will occur.,
(c) The compiler will reject it because only string pointers can be initialized this way.
(d) The variable var2 will contain the word "Angel", but the behaviour of strcpy is undefined.

7.40
```
void display(????)
            {
              printf( "%d\n", list[1][0] );
            }
int main()
       {
         int ary[2][2] = (1,2,3,4};
         display( (void *)ary);
         return 0;
       }
```

Which of the following when substituted for the ???? above, will NOT correctly print out the number 3?

(a) int list[][2]
(b) int list[2][2]
(c) int (*list)[2]
(d) int *list[2]

7.41 What does strncat append to the target string after the specified number of characters have been copied?

(a) Nothing
(b) NULL
(c) \0
(d) –0

7.42 int strcmp(sl, s2) compares strings sl and s2 and

(a) returns a value less than zero if s2 is lexicographically greater than sl
(b) returns zero if sl is lexicographically less than s2.

(c) returns 1 if sl is equal to s2.
(d) returns a value less than zero if sl is lexicographically greater than s2.

7.43
```
int func( char *sv )
       {
         static char info[]= "If anything can go wrong, it will"; ????
       }
```
From the sample above, which of the following could replace the ???? to correctly copy text from info into the passed buffer without any illegal memory accesses?
(a) `sv = info;`
(b) `strncpy( sv, info, sizeof( info ) );`
(c) `strncpy( sv, info, strlen( info ) + 1);`
(d) `strncpy( sv, info, strlen( sv ) + 1);`

7.44 `return (x=((((y=2)*z)+((a|b)*(2+s)))));`
Which of the following expressions use the minimum number of parentheses and is equivalent to the expression above?
(a) `return (x=(y=2)*z+(a|b)*(2+s));`
(b) `return (x=((y=2)*z+(a|b)*(2+s)));`
(c) `return x=(y=2)*z+a|b*(2+s);`
(d) `return x=(y=2)*z+(a/b)*(2+s);`

7.45 When a function is recursively called all automatic variables are
(a) stored in stack (b) stored in queue (c) stored in array (d) stored in linked list

7.46
```
main()
    {
      int n =10;
      fun(n);
    }
int fun( int n)
  {
    int i;
    for(i=0;i<=n;i++)
    fun(n-i);
    printf(" Kodeeswaran");
  }
```
How many times is the printf statement executed for n=10?
(a) 1 (b) Infinity (c) Zero (d) 10

7.47 C allows
(a) only call by value (b) only call by reference
(c) both
(d) only call by value and sometimes call by reference

7.48
```
main ()
 { }
    int a;
    f1(){}
```

```
f2(){}
```

To which of the above functions, is **int a** available for?

(a) all of them (b) onlyf2 (c) only fl (d) fl and f2 only

7.49 What will be result of the following program?

```
main()
    {
      void f(int,int);
      int i=10;
      f(i,i++);
   }
void f(int i, int j)
   {
     if (i>50)
     return;
     i+=j;
     f(i,j);
      printf("%d,",i);
   }
```

(a) 85,53,32,21 (b) 10,11,21,32,53 (c) 32,21,11,10 (d) none of the above

7.50 What is the output of the following code?

```
main()
    {
      printf("%d\n", sum(5));
    }
int sum(int n)
      {
        if(n<1) return n;
        else return(n+sum(n-1));
      }
```

(a) 10 (b) 16 (c) 14 (d) 15

7.51 What is the output generated by the following program?

```
#include <stdio.h>
main()
   {
     int i,x;
     for(i=1;i<=5;i++)
       {
         x = sq(i);
         printf("%d",x);
       }
   }
sq(int x)
   {
     return x*x;
   }
```

(a) 1234567 (b) 2516941 (c) 9162514 (d) 1491625

7.52 What is the output of the following code?

```
int n = -24;
main()
  {
    printd(n);
  }
printd(int n)
    { if (n < 0)
         {
           printf ("-");
           n = -n;
    }
      if (n % 10) printf ("%d", n);
      else printf ("%d", n/10);
      printf ("%d", n);
}
```

(a) –24 (b) 24 (c) –2424 (d) –224

7.53 What is the output of the following code?

```
main()
  {
     int x = 80, a = -80;
     void swap(int, int);
     swap(x, a);
     printf("numbers are %d\t%d",a,x),
  }
void swap(int y, int b)
         {
           int t=y;
           y=b;
           b=t;
         }
```

(a) numbers are 80 –80
(b) numbers are 80 80
(c) numbers are –80 80
(d) numbers are –80 –80

ANSWERS

Fill in the Blanks

1. parameters 2. void 3. single value 4. int
5. library 6. library 7. semicolon 8. register
9. call by value 10. extern, static, extern
11. built in functions 12. main() 13. function name
14. definition 15. string.h 16. int

True or False

1. True 2. False 3. True 4. True 5. False 6. False
7. False 8. True 9. True 10. False 11. False 12. False

Match the Following

1. 3 2. 5 3. 2 4. 1 5. 4

Objective Type Questions

1. b	11. c	21. b	31. a	41. c	51. d
2. a	12. c	22. d	32. c	42. a	52. c
3. b	13. a	23. c	33. d	43. c	53. c
4. c	14. b	24. a	34. c	44. b	
5. c	15. d	25. b	35. c	45. a	
6. d	16. c	26. b	36. c	46. c	
7. b	17. a	27. b	37. b	47. a	
8. a	18. b	28. c	38. b	48. d	
9. b	19. b	29. b	39. d	49. d	
10. c	20. a	30. c	40. d	50. d	

Pointers

8

C language supports pointers, and they play an important role in it. As most of the other languages do not support the use of pointers directly, it seems to be a new concept for the beginners. Pointers are very simple to use, provided the basic concept is understood properly. Careless use of pointers causes unexpected errors or difficulties in the execution of programs. Programs can be written efficiently and compactly with the help of pointers. Also, certain operations can be performed by using pointers only, in C.

C uses byte as the basic unit of memory. The bytes required to store a value depend on the data type of the object used. Each byte is numbered by an address for its reference. Generally, 1 byte is used for char, 2 bytes for int, 4 bytes for float and 8 bytes for double. It is possible to directly access the addresses in C by using pointers. A pointer is a valid address, which is stored in a pointer variable. A pointer variable is declared like an ordinary variable, with * preceding each variable name. The address of operator & and the indirection operator * are exclusively used in pointers, they precede their operands. The operator & requires a lvalue like a variable as its operand and it returns the address of its operand. The indirection operator * requires a pointer expression as its operand and it returns the data object stored in its operand.

The execution of the declaration statement results in allocating storage spaces only and the data objects are not assigned automatically. Hence, after the declaration of the pointer variables, they must be initialized. A pointer variable may be initialized by using static or dynamic memory allocation. In static memory allocation, the space reserved by the compiler is assigned to a pointer variable. Dynamic memory allocation is obtained by using the built-in functions like malloc( ) and calloc( ). A dynamically created memory may be freed using the function free( ).

After assigning proper values to the pointer variables, it is possible to manipulate the data objects stored. It is also possible to increment/decrement the pointer variables. Since the operator precedence levels of the operators &, *, + +, and – – are the same, the associativity of these operators are carefully followed to write expressions using these operators.

There is a strong relationship between arrays and pointers. The array variable represented by a subscript expression may also be written using a pointer expression. The representation of the pointer expression helps in faster and easier execution.

The concept of pointer to pointer helps in storing a chain of pointers. It is possible to have an array of pointers and a pointer to an array. An array of pointers contains pointers as its elements and a pointer to an array is used to store arrays. The syntax of declaring a pointer to an array differs from the array of pointers by enclosing * and the array name within the parentheses. Character arrays may also be manipulated using pointers. String constants return a pointer that can be used in pointer expressions.

It is possible to pass string variables to the function main( ) using the standard parameters argc and *argv[ ] in it. It is also possible to define a pointer to a function, for manipulating the function as an ordinary variable. Only operations such as addition / subtraction of an integer with pointers, increment/decrement of the pointers and comparison of pointers are allowed for pointers. Pointer declarations may be made complex by combining pointer to a function, pointer to an array, array of pointers and function returning a pointer.

SHORT ANSWER TYPE QUESTIONS

8.1 What is a pointer value?

A pointer value is a data object that refers to a memory location.

8.2 What is an address?

Each memory location is numbered in the memory. The number attached to a memory location is called the address of that location.

8.3 What is a pointer variable?

A pointer variable is a variable that may contain the address of another variable or any valid address in the memory.

8.4 How is a pointer variable declared?

Format:

```
data_type *varl, *var2, ..., *varN;
```

For example,

```
int *pl, *p2;
```

Here, p 1 and p2 are pointer variables.

8.5 Which are the operators exclusively used with pointers?

Address of operator (&) and indirection operator (*). The operator & returns the address of its operand. The operator * returns the value pointed by its operand.

8.6 Give the syntax for using an address of operator.

The address of operator returns the address of its operand. The operand must be a named region of storage like int variable, float variable, etc. for which a value may be assigned. It cannot be a constant or an expression or a register type variable.

Format : ptr variable = &named_region

For example,

```
int num, *p;
p = &num;
```

8.7 Give the syntax for using an indirection operator.

The operand must be a pointer expression. The value returned is an lvalue.

Format : *ptr expression

For example,

```
int *p, num = 10;
p = &num;
printf ( "%d\n", *p ) ;          /* Prints the value 10 */
```

8.8 Compare the values returned by & and *.

The address of operator & requires an lvalue as its operand and it returns the actual address of its operand. It does not return lvalue. The indirection operator * returns an lvalue. It returns the value to which its operand points to.

8.9 Are pointers integers?

No, pointers are not integers. A pointer is an address. It is merely a positive number and not an integer.

8.10 What must be done to a pointer variable before it can be put to use?

The pointer variable must be initialized with appropriate pointer value before it is put to use.

8.11 Why is it necessary to declare pointer variables?

Pointer variables are also like other variables for which memory locations are allocated. Hence, pointer variables are also created like other variables by declaring them.

8.12 How are pointer variables initialized?

Pointer variables are initialized by one of the following two ways:

- Static memory allocation.
- Dynamic memory allocation.

8.13 What is a static memory allocation?

The compiler allocates the required memory space for a declared variable. By using the address of operator, the reserved address is obtained and this address may be assigned to a pointer variable. Since most of the declared variables have static memory, this way of assigning pointer value to a pointer variable is known as static memory allocation. Memory is assigned during compilation time.

8.14 What is a dynamic memory allocation?

A dynamic memory allocation uses functions such as malloc() or calloc() to get memory dynamically. If these functions are used to get memory dynamically and the values returned by these functions are assigned to pointer variables, such assignments are known as dynamic memory allocation. Memory is assigned during run time.

8.15 What is the operator used to find the number of locations needed for a data type or a variable?

The size of operator is used to obtain the number of locations needed for a data type.
For example,

```
int x, *p;
p = ( int * ) malloc(sizeof(int));
```

The operand of the size of operator is enclosed within parentheses. It may be a data type or a variable.

8.16 Name any two functions of dynamic memory allocation.

1. malloc(n) 2. calloc(n, m)
malloc() allocates n number of bytes mentioned in its argument.
calloc() allocates the space required for storing n elements, each element occupying m bytes, i.e. n * m bytes.

8.17 How to destroy a dynamically allocated memory?

The function free(ptr) is used to deallocate the memory space pointed to by ptr. It is equivalent to calloc(ptr, 0).

8.18 What is the purpose of realloc()?

The function realloc(ptr, n) uses two arguments. The first argument ptr is a pointer to a block of memory for which the size is to be altered. The second argument n specifies the new size. The size may be increased or decreased. If n is greater than the old size and if sufficient space is not available

subsequent to the old region, the function realloc() may create a new region and all the old data are moved to the new region.

8.19 How can pointer variables be incremented and decremented?

Pointer variables can be incremented or decremented using + + or – –. Pointer operators & and * may be involved while using + + or – –. The operators &, *, + + and – – have same precedence level and the associativity is from right to left. Care must be taken to evaluate the expression for the intended purpose.

8.20 Differentiate between an array name and a pointer variable.

A pointer variable is a variable whereas an array name is a fixed address and is not a variable. A pointer variable must be initialized but an array name cannot be initialized. An array name being a constant value, + + and – – operators cannot be applied to it.

8.21 Compare arrays and pointers.

Pointers are used to manipulate data using the addresses. Arrays use subscripted variables to access and manipulate data. Pointers use * operator to access the data pointed to by them. Array variables can be equivalently written using pointer expressions.

8.22 Represent a two-dimensional array using pointer.

Address of a[i][j]	Value of a[i][j]
&a[i][j] or a[i] + j or *(a+i)+j	*&a[i][j] or a[i][j] or *(a[i] + j) or *(*(a+i)+j)

8.23 What is a pointer to a pointer?

If a pointer variable points to another pointer value, such a situation is known as a pointer to a pointer. For example,

```
int *pl, **p2, v = 25;
pl = &v; p2 = &pl;
```

Here, p2 is a pointer to a pointer.

8.24 What is an array of pointers?

If the elements of an array are addresses, such an array is called an array of pointers.

8.25 How is an array of pointers declared?

Format:

```
data_ type *array_name[size];
```

For example,

```
int *p[5];    /* Array of pointers */
```

8.26 How is a pointer to an array is declared?

Format:

```
data_type ( *array_name)[size];
```

For example,

```
float ( *y ) [25] ; /* Pointer to an array */
```

8.27 Differentiate between an array of pointers and a pointer to an array.

S.No.	Array of pointers	Pointer to an array
1.	Declaration is **data-type *array_name[size];**	Declaration is **data-type (*array_ name)[size];**
2.	Size represents the row size.	Size represents the column size.
3.	The space for columns may be dynamically allocated.	The space for rows may be dynamically allocated.

8.28 What are the arguments possible in main()?

Two arguments, viz.

1. argument count — representing the number of arguments passed and
2. argument vector — representing the strings passed

are possible as main(int argc, char *argv[]). Here argc represents argument count and argv represents argument vector. Any user-defined name can also be used as parameters for main() instead of argc and argv.

8.29 What are command line arguments?

The arguments passed through the main() function are called command line arguments. They are also known as program parameters.

8.30 How is a pointer to a function declared?

Format:

```
data_type ( *function_name )(argl, arg2,..., argN );
```

For example,

```
double ( *pf )( int x, float y);
```

8.31 How is a function returning pointer is declared?

Format:

```
data_type *function_name( argl, arg2,..., argN );
```

For example,

```
double *f ( double x, double y);
```

8.32 Give an example of function returning a pointer to an array of float.

```
float ( *f() )[5];
```

8.33 Give an example of function fp returning a pointer to a function returning float.

```
float ( *fp() )();
```

8.34 What does the following declaration mean?

```
char ( *( *f() ) [ ] ) ();
```

Function returns a pointer to an array of pointers to a function returning character.

8.35 What are the advantages of pointers?

- Pointers can be used as scalar data type or derived data type.
- Arrays can be easily accessed using pointers.
- Call by reference is achieved using pointers.
- Complex declarations are possible in pointers.

8.36 What are the pointer declarations used in C?

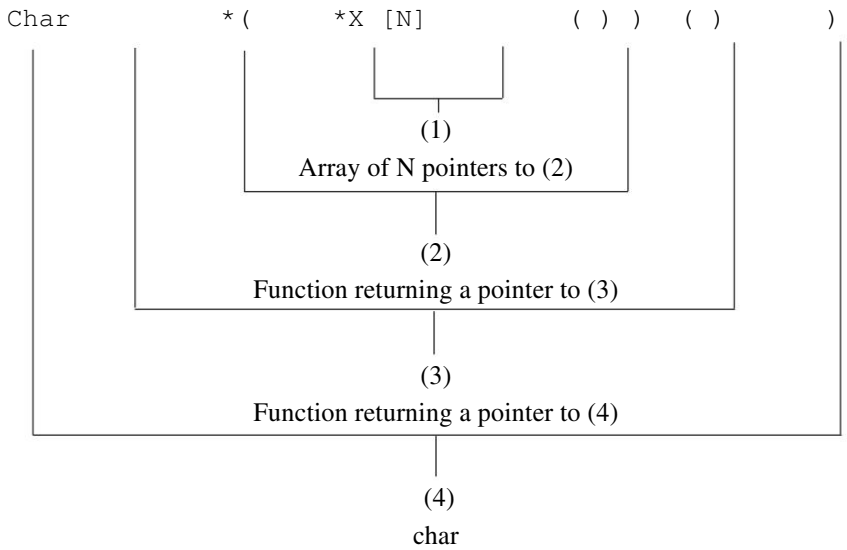
1.	Array of pointers, e.g.,	`int*a[10];`	: Array of pointers to int.
2.	Pointer to an array, e.g.,	`int(*a)[10];`	: Pointer to an array of int.
3.	Function returning a pointer, e.g.,	`float *f( );`	: Function returning a pointer to float.
4.	Pointer to a pointer, e.g.,	`int **x;`	: Pointer to pointer to int.
5.	Pointer to a data type, e.g.,	`char *p;`	: Pointer to char.

8.37 Declare an array of N pointers to a function returning pointers to a function returning a pointer to a character.

It is a complex declaration.

```
char *(*(*x[N]( ))( ));
```

Read the declaration in the following way from inner to outer.



8.38 Is a null pointer same as uninitialized pointer? Justify.

A null pointer is conceptually different from an uninitialized pointer. An uninitialized pointer may point to anywhere, whereas a null pointer doesn' t point to any object or function.

8.39 What is a null pointer?

C defines a special value called null pointer for each pointer type. It is used to compare a pointer to any object or function. The internal representation of null pointer for different data types may be different.

8.40 When is explicit cast operator required for a null pointer?

An explicit cast operator is required for a null pointer in a function call with variable arguments where no prototype is in scope.

8.41 How to cast a null pointer in a function call?

Only 3 ways are given below:

1. `# define NULL ((void*)0).`

It automatically converts 0 to the respective pointer type.

```
2. # define NULL(type) (type *)0
```

It passes the null pointer type needed in the function call.

3. Explicitly type cast the null pointer in the respective function call.

8.42 What does the run-time error message **null pointer assignment** mean?

Accessing an uninitialised pointer or an invalid location may cause such errors.

8.43 Differentiate between pointers and arrays.

	Array	Pointer
1.	Array allocates space automatically.	Explicitly assigned to point to an allocated space.
2.	It cannot be resised.	It can be resized using realloc( ).
3.	It cannot be reassigned.	Pointer can be reassigned.
4.	sizeof(arrayname) gives the number of bytes occupied by the array.	sizeof(p) returns number of bytes used to store the pointer variable p.

8.44. How to allocate memory for an array at run time?

The code

```
int *a;
a = malloc(5*sizeof(int));
```

allocates memory at run time to hold 5 integers in the array.

8.45 How to dynamically allocate a two-dimensional array?

The following code is an example of dynamic allocation of memory for a two-dimensional array.

```
# include <stdlib.h>
main()
   {
    int row = 5, col = 5,i;
    int **matrix = malloc(row*sizeof(int *));
    for (i = 0; i< row; i++)
      matrix[i] = malloc(col*sizeof(int));
  }
```

8.46 How to pass a two-dimensional array to a function?

```
main()
   {
    int matrix[5] [5];
     :
    init(matrix);
     .
     :
  }
void init(int matrix[][5]){...}
  (or)
void init(int (*matrix)[5]) {...}
```

8.47 How to pass a two-dimensional array to a function when the order of matrix is not known at compile time?

```
main()
   {
     int matrix[0][0], row, col;
     ...
     init(&matrix[0][0],row,col);
     ...
}
void init(int *mat, int row, int col)
      { ...
        mat [i * col + j] = x++; /* access matrix[i][j] */
        ...
       }
```

8.48 What happens to pointer p after calling free(p);?

Strictly speaking, a pointer value which has been freed is invalid. It is of no use and this pointer becomes an uninitialized pointer. However, sometimes it is unsafe to use a pointer value after it has been freed since the value of the pointer remains unchanged because of the call by value parameter passing mechanism. It is implementation dependant.

8.49 Compare calloc() and malloc().

	calloc()	malloc()
1.	The function calloc() takes 2 arguments. For example, calloc(n,m); n – number of objects m – size of each object	It takes only one argument. For example, malloc(nb); nb — number of byters.
2.	It initializes the contents of the block of memory to zero.	It doesn't initialize.
3.	It uses free() to deallocate.	It also uses free().
4.	Can be resized by realloc() function.	It can be resized by realloc();

8.50 Differentiate between a constant pointer and pointer to a constant.

The following declarations will differentiate the constant pointer from a pointer to constant.

```
const char *p;              // Pointer to a const character.
char const *p;              // Pointer to a const character.
char * const p;             // const pointer to a char variable.
const char *const p;        // const pointer to a const character.
```

It is very important to place * infront of const to define a const pointer.

8.51 What are the different versions of main()?

1. `main()`
2. `int main(void)`
3. `int main(int argc, char **argv)`
4. `int main(int argc, char *argv[])`

The arguments argc and argv may also be given any user-defined name.

8.52 What is the difference between memmove() and memcpy()? Which one is preferred?

memcpy()	memmove()

1. The function memcpy() uses three arguments. For example,
`void *s;`
`const void *t;`
`int n;`
`void *memcpy`
`...`
`memcpy(s,t,n);`
	The function memmove() uses three arguments. For example,
	`void *s`
	`const void *t;`
	`int n;`
	`void *memmove`
	`...`
	`memmove (s,t,n);`

2. It copies the first n characters from t to s, and return s.	It also copies the first n characters from t to s, and return s. It works even if the objects s and t overlap.
3. It is not safer to use when there is an overlap.	It is safer to use because of its guaranteed behaviour even if there is an overlap.
4. It is more efficiently implementable.	It is preferable compared to memcpy.

8.53 Compare memchr() and memset().

memchr()	memset()

1. The function memchr() uses three arguments. For example,
`const void *cs;`
`int c,n;`
`void *memchr()`
`...`
`memchr (cs,c,n);`
	The function memset() also uses three arguments. For example,
	`const void *s;`
	`int c,n;`
	`void *memset()`
	`...`
	`memset (s,c,n);`

2. It returns pointer to first occurrence of character c in cs or NULL if c is not present among first n characters.	It places character c into first n characters of s and returns s.
Note: c is an int converted to an unsigned char.	

8.54 What is a generic pointer in C?

In C, void* acts as a generic pointer. When other pointer types are assigned to generic pointer, conversions are applied automatically. However these conversions cannot be performed explicitly. Also it is very important to note that there is no generic pointer to pointer type in ANSI C.

8.55 Where can 0 be used to represent a null pointer?

The value zero can be used to represent a null pointer in

- an assignment and initialization,
- a comparison, and
- a function call having fixed argument with prototype in scope.

8.56 Is the allocated space within a function automatically deallocated when the function returns?

No. Pointer is different from what it points to. Local variables including local pointer variables in a function are deallocated automatically when function returns. But, in case of a local pointer vari-

able, deallocation means that the pointer is deallocated and not the block of memory allocated to it. Memory dynamically allocated always persists until the allocation is freed or the program terminates.

8.57
```
int size ,*int_ptr,table[20];
char ch,*char ptr;
double d,grid[20];
```
Find out the value for the following statements?

(a) `size=sizeof(int)`	Output = 2
(b) `size=sizeof(ch)`	Output = 1
(c) `size=sizeof(size)`	Output = 2
(d) `size=sizeof(table)`	Output = 40
(e) `size=sizeof(grid)`	Output = 160 /* sizeof (d) is 8*/
(f) `size=sizeof(char_ ptr)`	Output = 4

8.58 What is the output of the program?
```
include <malloc.h>
char *f()
      {
        char *s=malloc(8);
        strcpy(s,"goodbye")
      }
main()
    {
      char *f();
      printf("%c",*f()='A');
    }
```
Output:
A

8.59 What is the output of the program?
```
typedef struct node
       {
         char s[15] ;
         struct node *next;
         } *NODEPTR;
```
what does NODEPTR stand for?

NODEPTR stands for pointer to struct node.

8.60 What is the output of the program?
```
main()
   {
     int a[5],*p;
     for(p=a;p<&a[5];p++)
       {
         *p=p-a;
         printf("%d ",*p);
       }
    }
```

Output:

0 1 2 3 4

8.61 What is the output of the program?

```
main()
   {
     void x(void);
     x();
   }
void x(void)
    {
      char a[]="HELLO";
      char *b="HELLO";
      char c[10]="HELLO";
      printf("%s %s %s\n",a,b,c);
      printf("%d %d %d\n",sizeof(a),sizeof(b),sizeof(c));
    }
```

Output:

HELLO HELLO HELLO

6 4 10

sizeof(b) gives the bytes required for storing the pointer b. Other two are the array sizes.

8.62 What is the output of the program?

```
char *cp;
int *ip;
cp=(char *)0x100;
ip=(int *)cp;
ip++;
cp++;
printf("cp = %x ip = %x", cp, ip);
```

Output:

```
cp = 101 ip = 102
```

8.63 Write an appropriate declaration for the following situations.

(a) x: function returning pointer to array[] of pointer, to function returning char.

Output:

```
char (**x() []) ();
```

(b) Declare a function func that accepts two integer arguments and return a pointer to a long integer.

Output:

```
long int *func(int,int);
```

8.64 What is the output of the program?

```
main()
  {
    char *ptr = "Ramco Systems";
    (*ptr)++;
    printf("%s\n",ptr);
```

```
    ptr++;
    printf("%s\n",ptr);
  }
```

Output:
Samco Systems
amco Systems

8.65 What is the output of the program?

```
#include <stdio.h>
main()
  {
    char *pl;
    char *p2;
    pl=(char *) malloc(25);
    p2=(char *) malloc(25);
    strcpy(pl,"Ramco");
    strcpy(p2,"Systems");
    strcat(pl,p2);
    printf("%s",pl);
  }
```

Output:
Ramco Systems

8.66 What is the output of the program?

```
#include <stdio.h>
int printf(const char*,...);
main(void)
        {
          int i=100,j=10,k=20;
          int sum;
          float ave;
          char myformat[]="ave=%.2f";
          sum=i+j+k;
          ave=sum/3.0;
          printf(myformat,ave);
        }
```

Output:
ave = 43.33

8.67 What is the output of the program?

```
#include <stdio.h>
int fn(void);
void print(int,int(*)());
int i=10;
main(void)
        {
          int i=20;
          print(i,fn);
```

```
        }
void print(int i,int (*fnl)())
        {
          printf("%d\n",(*fnl)());
        }
int fn(void)
     {
       return(i-=5);
     }
```

Output:
5

8.68 What is the output of the program?

```
int bags[5]={20, 5, 20, 3, 20};
main(void)
      {
       int pos=5,*next();
       *next()=pos;
       printf("%d %d %d",pos,*next(),bags[0]);
      }
int *next ()
       {
         int i;
         for(i=0;i<5;i++)
           if (bags[i]==20)
              return(bags+i);
         printf("Error!");
         exit(0);
       }
```

Output:
5 20 5

8.69 What is the output of the program?

```
#include <stdio.h>
main(void)
       {
         void pa(int *a,int n);
         int arr[5]={5, 4, 3, 2, 1};
         pa(arr,5);
       }
void pa(int *a,int n)
       {
         int i;
         for(i=0;i<n;i++)
         printf("%d ",*(a++)+i);
       }
```

Output:
5 5 5 5 5

8.70 What is the output of the program?

```
int i, b[] ={1, 2, 3, 4, 5}, *p;
p = b;
++*p;
printf("%d ",*p);
p+=2;
printf("%d ",*p);
```

Output:
2 3

8.71 What is the output of the program?

```
int num[ ]={10,1,5,22,90};
main()
    {
      int *p,*q;
      int i;
      p=num;
      q=num+2;
      i =*p++;
      printf("%d %d\n",i,q-p);
    }
```

Output
10 1

8.72 Write a program which produces the source code assigned to a character pointer as its output.

```
char *s = "char *s = %c%s%c;main( ){printf(s,'\"',s,'\"');}";
main( )
    {
      clrscr( );
      printf(s,'\"',s,'\"');
    }
```

8.73 Write a program to find whether an integer is odd or even without using control statements.

```
main( )
    {
      int n;
      char *a[2] = {"EVEN","ODD"};
      scanf("%d",&n);
      printf("%d is %s\n",n,a[n%2]);
    }
```

8.74 Write a program to find the machine's byte order as big-endian or little-endian.

```
main( )
   {
     int a = 1;
     if(*(char *)&a ==1)
       printf("Little-Endian\n");
     else
       printf("Big-Endian\n");
    }
```

8.75 Discuss on pointer arithmetic.

Pointers are not integers. Certain arithmetic operations on pointers are allowed in C. This provision facilitates the manipulation of pointers and it is called **pointer arithmetic** or **address arithmetic**. The following are **valid** pointer operations.

1. Assignment of pointers to the same type of pointers:
   The assignment of pointers is done symbolically. Hence no integer constant except 0, can be assigned to a pointer.
2. Adding or subtracting a pointer and an integer.
3. Subtracting or comparing two pointers (within the array limits) which are pointing to the elements of an array.
4. Incrementing or decrementing the pointers (within the array limits) pointing to the elements of an array. When a pointer to an integer is incremented by one, the address is incremented by 2 (as 2 bytes are used for int). Such scaling factors necessary for the pointer arithmetic are taken care of automatically by the compiler.
5. Assigning the value 0 to the pointer variable and comparing 0 with the pointer. The pointer having address 0 points to nowhere at all.

8.76 What are the undefined and invalid pointer arithmetic?

The following pointer operations are **undefined**:

1. Comparisons of pointers that do not point to the elements of the same array.
2. Arithmetic operations such as addition/subtraction of an integer with pointers and increment / decrement of pointers that do not point to the elements of the same array.

There is an exception for the above two cases. The address of the first element past the end of an array can be used in the pointer arithmetic. The following operations are **invalid** pointer arithmetic.

(i) Adding, multiplying and dividing two pointers.
(ii) Shifting or masking pointers.
(iii) Addition of float or double to pointers
(iv) Assignment of a pointer of one type to a pointer of another type.

8.77 What are the advantages of using array of pointers to string instead of an array of strings?

1. Efficient use of memory.
2. Easier to exchange the strings by moving their pointers while sorting.

FILL IN THE BLANKS

1. Pointers allow the direct access of_____.
2. A variable is used as operand for_____pointer operator.
3. A_____memory allocation is done at compilation time.
4. The number of arguments used in the function malloc() is_____.
5. The memory allocated by calloc() function contains the storage initialized to_____.
6. Pointer operators are_____and_____.
7. Indirection operator * returns_____.

8. The expression *p++ is evaluated from_____.
9. The expression a[i] + j is equivalent to_____.
10. The size of the array of pointers in the declaraton **float *x[15]**; represents the number of ____ in the two dimensional array x.
11. A pointer variable must be asigned a valid_____before using it.
12. C provides an operator_____to find out the required space to store the data object of a particular data type.
13. The function_____is capable of increasing the memory space already allocated.
14. A function name itself represents the_____of that function.
15. The recursive use of a pointer declaration makes it a_____.

TRUE OR FALSE

1. A pointer always refers to a scalar data type.
2. Using pointers as parameters allow call by reference in C.
3. Dynamic memory allocation is done at execution time.
4. Adding two pointers is an invalid operation.
5. The operand of address of operator & may be a register variable.
6. An integer can be added to a pointer.
7. Pointers can be multiplied.
8. The name of an array itself is a fixed address and it is not a variable.
9. An array name can be used at the lefthand side of an assignment.
10. A two-dimensional array can be defined as a one-dimensional array of pointers.
11. In array of pointers, the rows of the array may be of different length.
12. In an array of pointers, the column size is mentioned in its declaration whereas in a pointer to an array the row size is mentioned.
13. A function is not a variable, but it has an address.
14. The execution of programs using pointers is faster than the programs without using pointers.
15. A pointer can be used as a scalar data type as well as a derived data type.
16. Address of operator can be applied to pointer variables only.
17. If x is an int variable, **sizeof x** may be used to find the number of bytes used to hold an int data type.

MATCH THE FOLLOWING

1	data_type (*name[SIZE])() ;	Array of pointers
2	data_type *arrayname[SIZE];	Pointer to a function
3	data_type (*name)();	Array of pointer to a function
4	data type (*arrayname)[SIZE];	Pointer to pointer to pointer
5	char ***c;	Pointer to an array

OBJECTIVE TYPE QUESTIONS

8.1 Pointers are supported in

(a) FORTRAN (b) PASCAL (c) C (d) both options b and c

8.2 Pointer variable may be assigned

(a) an address value represented in hexadecimal
(b) an address value represented in octal
(c) the address of another variable
(d) An address value represented in binary

8.3 A pointer value refers to

(a) an integer constant (b) a float value
(c) any valid address in memory (d) any ordinary variable

8.4 Identify the correct declaration of pointer variables pl, p2.

(a) `int pl, p2;` (b) `int *pl, p2;` (c) `int pl,*p2;` (d) `int *pl, *p2;`

8.5 The operators exclusively used in connection with pointers are

(a) * and / (b) & and * (c) & and | (d) – and >

8.6 Identify the invalid expression for given **register int r = 10;**

(a) r = 20 (b) &r (c) r + 15 (d) r/10

8.7 Identify the invalid expression.

(a) &274 (b) &(a + b) (c) &(a*b) (d) all the above

8.8 Identify the wrong declaration statement.

(a) int *p, a = 10; (b) int a = 10, *p = &a;
(c) int *p = &a, a = 10; (d) options a and b

8.9 Identify the invalid expression given

```
int num = 15, *p = &num;
```

(a) *num (b) *(&num) (c) *&*&num (d) **&p

8.10 Identify the invalid expression for given float x = 2.14, *y =&x;

(a) &y (b) *&x (c) **&y (d) (*&)x

8.11 The operand of the address of operator is

(a) a constant (b) an expression
(c) a named region of storage (d) a register variable

8.12 How does compiler differentiate address of operator from bitwise AND operator?

(a) by using the number of operands and position of operands
(b) by seeing the declarations
(c) both options a and b
(d) by using the value of the operand

8.13 How does compiler differentiate indirection operator from multiplication operator?

(a) by using the number of operands (b) by seeing the position of operand
(c) both options a and b (d) by using the value of the operand

8.14 The address of operator returns
(a) the address of its operand (b) Ivalue
(c) both options a and b (d) rvalue

8.15 The indirection operator returns
(a) the data object stored in the address represented by its operand
(b) lvalue
(c) both options a and b
(d) rvalue

8.16 The operand of indirection operator is
(a) pointer variable (b) pointer expression
(c) both options a and b (d) ordinary variable

8.17 The operand of address of operator may be
(a) an ordinary variable (b) an array variable
(c) a pointer variable (d) Any one of the above

8.18 Identify the invalid lvalue given int x, *p =&x;
(a) *(p + 1) (b) *(p – 3) (c) both options a and b (d) &x

8.19 After the execution of the statement int x; the value of x is
(a) 0 (b) undefined (c) 1 (d) –1

8.20 Pointer variable may be initialized using
(a) static memory allocation (b) dynamic memory allocation
(c) both options a and b (d) a positive integer

8.21. Identify the invalid pointer operator.
(a) & (b) * (c) > > (d) none of the above

8.22 Assume 2 bytes for int, 4 bytes for float and 8 bytes for double data types respectively, how many bytes are assigned to the following pointer variables?

```
int *ip; float *fp; double *dp;
```

(a) 2 bytes for ip, 4 bytes for fp and 8 bytes for dp
(b) 2 bytes for all pointer variables ip, fp and dp
(c) one byte for ip, 2 bytes for fp and 4 bytes for dp
(d) 2 bytes for ip and 8 bytes for each fp and dp

8.23 The number of arguments used in malloc() is
(a) 0 (b) 1 (c) 2 (d) 3

8.24 The number of arguments used in calloc() is
(a) 0 (b) 1 (c) 2 (d) 3

8.25 The number of arguments used in realloc() is
(a) 0 (b) 1 (c) 2 (d) 3

8.26 The function used for dynamic deallocation of memory is
(a) destroy() (b) delete() (c) free() (d) remove()

8.27 The function call realloc(ptr, 0) is
(a) same as free(ptr)
(b) used to clear the values in the address represented by ptr

(c) used to set the value of ptr to be 0
(d) invalid

8.28 In the expression *cp++
(a) *cp is evaluated first and *cp is incremented by 1.
(b) *cp is evaluated first and cp is incremented by 1.
(c) cp is incremented by 1 first and * is applied.
(d) cp is incremented by 1 first and * is applied to the previous value of cp.

8.29 The pointers can be used to achieve
(a) call by function (b) call by reference (c) call by name (d) call by procedure

8.30 The operators &, *, + + and – – have
(a) same precedence level and same associativity
(b) same associativity and different precedence level
(c) different precedence level and different associativity
(d) different precedence level and same associativity

8.31 Identify the invalid expression for given syntax:

```
float fnum[10], *fptr = fnum;
```

(a) fnum + 4 (b) fptr[4] (c) fnum = + +fptr (d) &fnum[4]

8.32 Identify the correct statement for given expression

```
float fnum[10];
```

(a) fnum is a pointer variable.
(b) fnum is a fixed address and not a variable.
(c) fnum is an array variable.
(d) fnum is an address that can be modified.

8.33 Given the declaration **double prec[5]**; the address of the element prec[2] is obtained by:
(a) &prec[2] (b) prec + 2
(c) both options a and b (d) *(prec + 2)

8.34 Given the declaration **int prec[5];** the element prec[2] is accessed by
(a) prec[2] (b) prec + 2 (c) *(prec + 2) (d) both options a and c

8.35 Given **float x[5][5];** the address of the element x[2][3] is obtained by
(a) &x[2][3] (b) x[2] + 3 (c) *(x + 2) + 3 (d) all the above

8.36 Given **char s[4][10];** the element s[2][4] is accessed by
(a) s[2][4] (b) *(s[2] + 4) (c) *(*(s + 2) + 4) (d) all the above

8.37 Given **int a[5][5];** identify the correct expression, yielding the starting element.
(a) *a[0] (b) **a (c) a[0][0] (d) all the above

8.38 Given **int x[5][10][8];** find the address of the element x[2][3][5].
(a) x[2][3] + 5 (b) *(x[2] + 3) + 5
(b) *(*(x + 2) + 3) + 5 (d) all the above

8.39 Given **int x[5][5][5];** find the value of the element x[2][3][4]
(a) *(x[2][3] + 4) (b) *(*(x[2] + 3) + 4)
(c)*(*(*(x+2)+3)+4) (d) all the above

8.40 Given **int a[5];** how to declare array in the function definition if the function call is sort(a).
(a) sort(int *a) (b) sort(int a[5])
(c) both options a and b (d) sort(int a)

8.41 Given **int *pl, **p2, ***p3, v = 25;**
```
pl = &v; p2 = &pl; p3 = &p3;
```
how to obtain the value of v using pointer variable?
(a) *pl (b) **p2 (c) *** p3 (d) all the above

8.42 The declaration **float *a[5];** is
(a) an ordinary array (b) a pointer to an array
(c) an array of pointers (d) pointer to an array

8.43 The declaration **int (*p)[8];** is
(a) an array of pointers (b) a pointer to an array
(c) pointer to function (d) function returning pointer

8.44 Array of pointers such as **int *p[5];** is used for
(a) fixed row size and varying column size
(b) fixed row size and fixed column size
(c) varying row size and fixed column size
(d) varying row size and varying column size

8.45 Pointer to array such as **int (*a)[8];** is used for
(a) fixed row size and varying column size
(b) fixed row size and fixed column size
(c) varying row size and fixed column size
(d) varying row size and varying column size

8.46 Given **float x[5][5]; float (*y)[5];** the assignment of the array x to the pointer variable y may be done as
(a) `y = &x;` (b) `y = x ;` (c) `y[0] = x ;` (d) `*y=x;`

8.47 Given **float x[5][10], *y[5];** the assignment of the array x to y is done as
(a) `y = x ;`
(b) `y[0] = x; y[1] = x[1]; ... ; y[4] = x[4];`
(c) `y[0] = x[0]; y[1] = x[1]; y[4] = x[4];`
(d) both options b and c

8.48 Given **char *p = "ANSI C";** identify the expression returning the value C.
(a) p[5] (b) *("ANSI C" + 5) (c) "ANSI C"[5] (d) all the above

8.49 Given **char *t[10];** identify the correct statement.
(a) `strcpy(t[0], "BASIC");` (b) `t[0] = "JAVA";`
(c) `both options a and b` (d) none of the above

8.50 Given **char (*t)[10]; t = (char *)malloc(50);** identify the illegal statement.
(a) `t[0] = "BASIC";` (b) `strcpy(t[4],"FORTRAN");`
(b) `strcpy (t[0],"JAVA")` (d) both options b and c

8.51 Identify the arguments of main().
(a) `int argc` (b) `char *argv [ ]`
(c) both options a and b (d) no arguments are allowed

8.52 Which is the correct function header for function main()?
(a) main(int argc, char *argv [ ])
(b) main(int argc, char **argv)
(c) main(int argc, char *av[ ])
(d) all the above

8.53 The first argument argc in main() counts
(a) the number of command line strings including the execution command
(b) the number of command line strings excluding the execution command
(c) the number of lines in a program
(d) the number of characters in a program

8.54 The arguments in main() function are known as
(a) program parameters
(b) command line arguments
(c) both options a and b
(d) memory in-line format conversion

8.55 The argument argv[ ] is used to
(a) count the number of command line arguments
(b) pass strings to the programs including the excecution command
(c) both options a and b
(d) pass strings to the programs excluding the excecution command

8.56 The argc in main() is used to
(a) count the number of command line arguments
(b) pass strings to the programs
(c) both options a and b
(d) count the number of lines in a program

8.57 In the declaration **double (*pf)()**;
(a) pf is a pointer to a function
(b) pf is a function returning pointer
(c) pf is a pointer to array
(d) pf is an array of pointers

8.58 In the declaration **double *f()**;
(a) f is a pointer to a function
(b) f is a function returning a pointer
(c) f is an array of function
(d) f is an pointer to array

8.59 Identify the invalid assignment, for given **int *p, x**;
(a) `p = 0;`
(b) `p = 255864u;`
(c) `p = &x;`
(d) `*p = 10;` /* assume valid pointer is already assigned to p*/

8.60 The operations that can be performed on pointers are
(a) addition or subtraction of a pointer and an integer
(b) assignment of pointer using pointer expression
(c) assignment of the value 0 to a pointer
(d) all the above

8.61 The invalid address arithmetic is (are)
(a) adding two pointers
(b) multiplying two pointers
(c) dividing two pointers
(d) all the above

8.62 The invalid pointer arithmetic is (are)
(a) shifting pointers using shift operators
(b) addition of float or double values to printers
(c) both options a and b
(d) incrementing pointers

8.63 The declaration **float(*f[5])()**; is
(a) an array of pointers to a function returning float
(b) pointer to an array
(c) function returning pointer to array
(d) pointer to a function returning an array of pointers to float

8.64 The declaration **float (*fp())()**; is
(a) pointer fp returning float
(b) function fp returning a pointer to a function returning float
(c) function fp returning a pointer
(d) pointer to a function

8.65 When applied to a variable, what does the unary "&" operator yield?
(a) the variable's value (b) the variable's address
(c) the variable's format (d) the variable's right value

8.66 Which of the following is the most accurate statement about runtime memory?
1. Memory is allocated by using malloc()
2. Memory is freed by a garbage collector
3. Allocated memory can be freed by calling free()

(a) option 1 only (b) option 2 only
(c) both options 1 and 2 (d) both options 1 and 3

8.67 Which of the following is True about pointers?
1. Pointers do not require memory storage.
2. The size of a pointer depends on the type of the variable it points to.

(a) option 1 only (b) option 2 only
(c) options 1 and 2 (d) neither option 1 nor 2

8.68 If ptrl and ptr2 are valid pointers in the same array, then which of the following statements is valid?
(a) ptrl+ 2 (b) ptrl- ptr2 (c) ptrl * ptr2 (d) both options a and b

8.69
```
char *x;
x = "AMMA";
```
Is the above code valid?
(a) Yes. A new memory space will be allocated to hold the string " AMMA".
(b) No. It would assign the string " AMMA " to an unallocated space in memory.
(c) Yes. The pointer x will point to the memory location created for the constant " AMMA ".
(d) No. This syntax is not allowed.

8.70
```
char *x = NULL;
printf("%s\n", x);
```
What will be the behaviour of the sample code above?
(a) This will not compile. A pointer cannot be initialized to NULL.

(b) The result will fall into "undefined behaviour" and be unpredictable.
(c) A "0" will be printed.
(d) Nothing will be printed.

8.71
```
char base[5] = "a";
const char *ptr;
```
Given the above, which of the following statements is legal?
(a) `ptr = base;` (b) `*ptr = "a";` (c) `*ptr=base;` (d) `ptr = 'a';`

8.72
```
char *ptr;
char string[] = "project";
ptr = string;
ptr += (ptr + 5);
```
What string does ptr contain in the sample code above?
(a) ct (b) ject
(c) oject (d) Unknown. This code is incorrect.

8.73
```
char ptrl[] = "Drama Actor";
char *ptr2 = malloc( 5 );
ptr2 = ptrl;
```
What is wrong with the above code (assuming the call to malloc does not fail)?
(a) There will be a memory overwrite.
(b) It will not compile.
(c) There will be a segmentation fault.
(d) Not enough space is allocated by the malloc.

8.74
```
char echo[ 50 ]= "Brain Power";
char *ptr = echo + 5;
```
From the sample above, which would be the proper way to copy 20 bytes from the location pointed to by ptr to the beginning of echo?
(a) `memcpy( echo, ptr, 20 );`
(b) It cannot be done, because the source and destination overlap.
(c) `strncpy( echo, ptr, 20 );`
(d) memmove (echo, ptr, 20);

8.75
```
char subl[100] = "Tamil";
char sub2[100] = "Hindi";
char *strptrl = subl + 2;
char *strptr2 = sub2 + 3;
strcpy( strptrl, sub2 );
strcpy( strptr2, subl );
printf( "%s\n",subl );
printf( "%s\n",sub2 );
```
Given the sample code above, which of the following string values will be printed when the code is executed?
(a) TaHinTamil
HinTamil
(b) TamHindi
HiTamHindi

(c) TaHindi
HinTaHindi
(d) TaHindi
HinTamil

8.76 Scrutinize the following code in C.

```
char fruit[] = "Orange";
char *ptr;
ptr = fruit;
```

What will be the output for the following expression?

```
printf("%c",*( ptr + 2 ));
```

(a) r (b) a (c) n (d) none

8.77 In the given code:

```
int a[50];
int *ptr;
ptr = a;
```

To access the 7th element of the array which of the following is incorrect?

(a) *(a+6) (b) a[6] (c) ptr[6] (d) *(*ptr + 6)

8.78

```
main()
  {
    int a[5] = {-2, -1, 3, 4, 5};
    int *b;
    b=&a [2] ;
  }
```

Then the value of b[–1] is:

(a) 4 (b) 3 (c) –1 (d) –2

8.79

```
main()
  {
    int a = 5, *ptr; ptr = &a;
    printf ("%d", ++*ptr);
  }
```

The output might be:

(a) 6 (b) 5 (c) 0 (d) none

8.80 int x, array[10];
From the sample above, which of the following is not a valid initialization for ptr?

(a) `int *ptr = *array;` (b) `int *ptr = 9 + array;`
(c) `int *ptr = &x;` (d) `int *ptr = (int *)0x1000;`

8.81 `int y[4] = {5,6,7,8};`
Which of the following will declare a pointer variable that points to the array in the sample code above?

(a) `int *(yptr[4]) = &y;` (b) `int (yptr *)[4] = &y;`
(c) `int (*yptr)[4] = &y;` (d) `int *yptr[4] = &y;`

8.82
```
int x[] = {1, 2, 3, 4, 5};
int *ptr, **ptr2;
ptr = x;
ptr2 = &ptr;
```
Referring to the sample code above, how would you update x[2] to 10 using ptr2?

(a) `**(ptr2 + 2) = 10;` (b) `*(&ptr2 + 2) = 10;`

(c) `*(*ptr2 + 2) = 10;` (d) `(**ptr2 + 2) = 10;`

8.83
```
main()
{
  char x[25], y[25], *p = y;
  strcpy(x, "BIRTHDAY");
  strcpy(y, "HAPPY");
  p=x;
  strcpy(x, "LOVER");
  *p = 'D';
  printf("p=%s\n", p);
}
```
What will be the output when the sample code above is executed?

(a) p=DOVERDAY (b) p=LOVERDAY (c) p=DOVER (d) p=DIRTHDAY

8.84
```
int *x;
x = (int *) 15;
```
Is the above code legal?

(a) Yes. A new memory space will be allocated to hold the number 15.

(b) No. This would assign the number 15 to an unallocated space in memory.

(c) Yes. Upon initialization, the number 15 will be stored in a special pointer memory address space.

(d) Yes. The pointer x will point at the integer in memory location 15.

8.85 Which statement correctly defines a character string with a length of 4 plus 1 for the terminating NUL?

(a) `char string[];` (b) `char* string;`

(c) `char* string [5];` (d) `char string [5];`

8.86 Given that **a** is an array of elements of type **t**, val is a lvalue expression of type "pointer to t" that points to elements in **a** then * + +val.

(a) sets val to point to the next element in a.

(b) increments val and then references the value in a, that val points to.

(c) references the value in a, that val points to and then increments val.

(d) references the element of a that val points to.

8.87
```
int y[4] = {6, 7, 8, 9};
int *ptr = y + 2;
printf("%d\n", ptr[ 1 ] );
```
What is printed when the sample code above is executed?

(a) 7 (b) 8

(c) 9 (d) The code will not compile.

8.88
```
printf("%s\n", string );
```
Which of the following initializations for the string variable will cause the code above to print the string, "First grade actor" when executed without memory leaks or memory overwrite?

(a)
```
char *string = malloc( 100 );
string = "First grade actor ";
```
(b)
```
char string[] = "Hello again";
string = "First grade actor";
```
(c)
```
char *string = "Hello again";
string = "First grade actor";
```
(d)
```
char string [100];
string = "First grade actor";
```

8.89
```
# include <ctype.h>
char s[] = "Photo Flash";
char *ptr = s;
```
Referring to the code above, which of the following code is the best way to convert the string s to all lower case letters?

(a)
```
for( ; *ptr; ptr++ )
    {
      if( isalpha( *ptr ) && isupper( *ptr ) ) *ptr = tolower(*ptr);
    }
```
(b)
```
tolower( ptr );
```
(c)
```
for( ; *ptr; ptr++ )
    {
      if( isupper ( *ptr ) ) *ptr = tolower( *ptr );
    }
```
(d)
```
for( ; *ptr; ptr++ )
    {
      *ptr = tolower( *ptr );
    }
```

8.90
```
extern void *ptrl;
extern void *ptr2;
int compare( int n )
       {
         return ????;
       }
```
What should replace the ???? in the code above to compare the first n bytes of the memory pointed to by ptrl and ptr2 and return a non-zero value if they are equal?

(a) `memcmp( ptrl, ptr2, n )`
(b) `strncmp( ptrl, ptr2, n )`
(c) `!strncmp( ptrl, ptr2, n )`
(d) `!memcmp( ptrl, ptr2, n )`

8.91 What memory function should be used to allocate memory in which all bits are initialized to 0?

(a) calloc (b) malloc (c) alloc (d) memalloc

8.92 What is the primary difference between the malloc and calloc functions?
(a) Memory allocated by calloc does not need to be freed and memory allocated by malloc does.
(b) calloc returns a pointer to char and malloc returns a void pointer.
(c) calloc initializes the memory returned and malloc does not.
(d) calloc can allocate memory for an array and malloc cannot.

8.93
```
int *const size = 10;
```
If the address of size is 3024, then size++ is
(a) 11 (b) 3025 (c) 3026 (d) invalid

8.94
```
int *ptr = (int *)malloc(sizeof(int));
ptr += 3;
```
If ptr points to the memory location 1000 and after execution of the statement ptr += 3, ptr will point to the memory location_____ (Assume 32 bits for int).
(a) 1006 (b) 1003 (c) 1010 (d) none

8.95 The amount of memory to be allocated for the following array of pointers.
```
short *p[4];
```
(a) no memory (b) 4 bytes (c) 6 bytes (d) 16 bytes

8.96
```
int x = 1;
int *ptr = malloc(sizeof(int));
ptr = &x;
x = 2;
*ptr = 3;
```
Is there anything wrong with the above code?
(a) No, x will be set to 2. (b) No, x will be set to 3.
(c) Yes, There will be a memory overwrite. (d) Yes, There will be a memory leak.

8.97
```
int *ptr = malloc(5 * sizeof(int));
realloc(ptr, 10 * sizeof(int));
for (i=0; i< 10; i++) { ptr[i] = 0; }
```
Assuming realloc succeeds, what effect will the above sample have on the rest of the program?
(a) The pointer "ptr" will contain an array of 10 that is initialized to 0 with no issues.
(b) This will result in a memory overwrite but no memory leak.
(c) This will result in a memory leak but no memory overwrite.
(d) This will result in both a memory leak and memory overwrite.

8.98
```
int *array [3];
int (*ptr) [] = array;
int x = 2, y = 3, z = 4;
```
Referring to the sample code above, how would you assign the second pointer in the array 'ptr' to point to the value of y?
(a) `(*ptr)[1]=y;` (b) `(*ptr)[2]=&y;` (c) `*ptr[2]=y;` (d) `(*ptr)[1]=&y;`

8.99
```
void *ptr;
```
What would be the correct way to cast ptr in the code above to be a pointer to a 3 element array of function pointers with a return value of int and a single parameter of a pointer to char?
(a) (int ((*)[3])(char *))ptr (b) ((int *)(*[3])(char *))ptr
(c) (int (*(*)[3])(char *))ptr (d) (int *(*)[3](char *))ptr

8.100
```
main()
    {
      int count = 0;
      int matrix[5][5];
     register int  *ptr;
    int i, j;
    for (i=0; i<5; i++)
     for (j=0; j<5; j++) matrix[i][j] = count++;
      ptr = &matrix[1][1];
      printf("%d\n", ptr[2]);
}
```
Referring to the sample above, what will be the value of "ptr[2]", after execution?

(a) 6 (b) 7 (c) 8 (d) 9

8.101
```
main()
   {
    int  a[4][2];
    int b = 0, x; _
    int i, y;
    for (i = 0; i < 4; i++)
        for (y=0; y< 2; y++)
          a[i][y]= b++;
        x = *(*(a+2) - 1);
    }
```
What is the value of x in the above sample?

(a) 2 (b) 3 (c) 4 (d) 5

8.102
```
void display(int *list)
        {
          printf("element=%d\n", *(list + 3));
        }
int main()
       {
         int ary[3] [3] = { { 0 }, { 2, 3 }, { 4, 5, 6 }};
         display((int *) ary);
         return 0;
       }
```
What will be printed when the sample code above is executed?

(a) element = 0 (b) element = 2 (c) element = 3 (d) element = 4

8.103
```
#include <math.h>
static double ( *funcs[] )( double ) =
                                    {
                                      sin, cos, tan, asin, acos,
                                      atan, sinh, cosh, tanh
                                    };
```

```
double computeTrigFunction(int index, double argument)
                                    {
                                      return ????;
                                    }
```

Referring to the sample code above, which should compute the value of a trigonometric function based on its index, what would be a replacement for the ???? to make the correct call?

(a) `(*funcs)[ index ]( argument )`
(b) `funcs[index] (argument)`
(c) `funcs (argument) [index]`
(d) `*funcs[ index ]( argument )`

8.104 **float (*f[5]());** This declaration represents

(a) pointer to function returning array of float
(b) pointer to array of pointer to function returning float
(c) array of pointers to function returning array of float
(d) array of pointers to function returning float

8.105 **float (*f())[5];** This declaration represents

(a) function returning a pointer to the array of float
(b) pointer to function returning array of float
(c) array to function returning float
(d) all of the above

8.106 **char (*(*f()[])();** This pointer declaration represents

(a) function returning array of pointer to a function
(b) pointer to array of pointer to a function return char
(c) function returning a pointer to array of pointer to a function returning char
(d) function returning a pointer to returning char

8.107 **float (*(*f[5])())[10];** This declaration represents

(a) array[5] of function returning a pointer to array[10] of float
(b) array of pointer returning pointer to array[10] of float
(c) array[5] of pointer to a function returning a pointer to array[10] of float
(d) none

8.108
```
# include <stdio.h>
char *format = "%d";
int main()
       {
         int x; void func();
         func( scanf, &x );
         printf("%d\n", x );
         return 0;
       }
```

Referring to the sample code above, which of the following would be a correct implementation for func?

(a) `void func( int *y(const char*, ... ), int *x )`
`{ (*y)( format, &x ); }`

(b)
```
void func(int(*y)(const char*, ... ), int *x )
                { (*y)( format, x ); }
```
(c)
```
void func( int (*y)(const char*, ... ), int *x )
                { (*y)( format, &x ); }
```
(d)
```
void func( (int *)y(const char*, ... ), int *x )
                { (*y)( format, x ); }
```

8.109 In the declaration **int x[3][2][2] ={ 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 };**
((*(x+2) + 1) + 1) represents
(a) the element 13 (b) the element 9
(c) the address of the element of 8 (d) the address of the element of 9

8.110
```
int a=1; b=2; c=3; *pointer; pointer=&c;
a=c/*pointer;
b=c;
printf("a=%d b=%d",a,b);
```
What will be the output?
(a) a=1 b=3 (b) a=3 b=3 (c) 3 2 (d) Error

8.111
```
void fn(int *a, int *b)
                {
                  int *t; t=a;
                  a=b;
                  b=t;
                }
main()
    {
      int a=2; void fn();
      int b=3;
      fn(&a,&b);
      printf("%d %d\n",a,b);
    }
```
What will be the output?
(a) Error at runtime (b) Compilation error (c) 2 3 (d) 3 2

8.112
```
main()
  {
    char *a[]={"jaya","mahe","chandra","buchi"};
    printf("%d ",sizeof(a)/sizeof(char *));
  }
```
(a) 4 (b) bytes for char (c) bytes for char * (d) error

8.113
```
main()
  {
    char *p="abc";
    char *q="abc123";
    while(*p++=*q++)
```

```
        {
          printf("%c%c",*p,*q);
        }
    }
```

What will be the output of the above?

(a) aabbcc (b) aabbccl23 (c) abcabc123 (d) bbccla2b3c

8.114 What is the result of the expression for the following declaration?

```
int A[] ={1,2,3,4,5};
*A + 1 - *A + 3
```

(a) 2 (b) –2
(c) 4 (d) none of the above

8.115 Which are valid?

(i) Pointers can be added.
(ii) Pointers can be subtracted.
(iii) Integers can be added to pointers.

(a) all correct (b) only options (i) and (ii)
(c) only option (iii) (d) only options (ii) and (iii)

8.116 x = malloc (y). Which of the following statements is correct?

(a) x is the size of the memory allocated
(b) y points to the memory allocated
(c) x points to the memory allocated
(d) none of the above

8.117 p and q are pointers to the same type of data items. Which of these are valid?

(i) *(p+q) (ii) *(P–q) (iii) *p – *q
(a) all (b) options (i) and (ii)
(c) option (iii) is valid sometimes (d) none of the above

8.118
```
int *i;
float *f;
char *c;
```

Which are the valid castings?

(i) (int *) &c
(ii) (float *) &c
(iii) (char *) &i

(a) options (i), (ii) and (iii) (b) only options (i) and (iii)
(c) only option (iii) (d) only options (i) and (ii)

8.119
```
int a = 'a', d = 'd';
char *b = "b", **c = "cr";
main ()
  {
    mixup (a, b, &c);
  }
mixup (int pl, char *p2, char **p3)
    {
```

```
    int *temp;
    …
  }
```

What is the value of a and b in mixup at the beginning?

(a) a address of the variable b (b) 'a ' ' b'

(c) 'a' address of the value b in "b" (d) a b

8.120
```
main()
    {
      char s[] = "S.T.S.", *A;
      print(s);
    {
      print(char *p)
        {
          while (*p !='\0')
            {
              if (*p !='.') printf ("%c", *p);
              p++;
            }
        }
```

What is the output?

(a) S.T.S. (b) STS (c) ST (d) S.T.

8.121 What will be the result of the following segment of the program using ANSI C compiler?

```
  main()
      {
        char *s="hello world";
        int i=7;
        printf("%.*%s",s);
      }
```

(a) Syntax error (b) hello w (c) hello world (d) %s

8.122
```
main ()
  {
    char *name = "name";
    change (name);
    printf ("%s", name);
  }
change (char *name)
    {
     char *nm = "newname";
     name = nm;
    }
```

What is the output?

(a) name (b) newname

(c) name = nm not valid (d) function call invalid

8.123 Select the explanation for the following declaration:

```
int (*(*ptr)(int)) (void)
```

(a) ptr is a pointer pointing to an integer function that takes an int argument returning an integer which points to a function with no argument.
(b) ptr is pointer to a function that takes an int argument returning a pointer to a function with no argument which returns an integer.
(c) This is not a valid C statement.
(d) None of the above.

8.124 Which of the following statement when replaces ??? results in error?

```
main ()
  {
    int *i = 10 , *j=20;
    ???
  }
```

(a) `i=(int) ((int) i * (int)j);` (b) `i = i*j;`
(c) `i = (int*) ((int)i*(int)j);` (d) `i = (int) i* (int)j;`

8.125 What will be the result of the following program?

```
main()
   {
     char *x="String";
     char y[] = "add";
     char *z;
     z=(char *) malloc(sizeof(x)+sizeof(y)-1);
     strcpy(z,y);
     strcat(z,y);
     printf("%s+%s=%s",y,x,z);
   }
```

(a) add+ String =AddString (b) syntax error during compilation
(c) run time error/core dump (d) add+String=addadd

8.126 What will be the result of executing following program?

```
main()
    {
      char *x="New";
      char *y="Dictionary";
      char *t;
      void swap (char * , char *);
      swap (x,y);
      printf("(%s, %s)",x,y);
      t=x;
      x=y;
      y=t;
      printf("-(%s, %s)",x,y);
    }
void swap (char *x,char *y)
```

```
{
    char *t;
    t=x;
    x=y;
    y=t;
}
```

(a) (New, Dictionary) – (New, Dictionary)
(b) (Dictionary, New) – (New, Dictionary)
(c) (New, Dictionary) – (Dictionary, New)
(d) (Dictionary, New) – (Dictionary, New)

8.127 What is the output of the following program?

```
char *c[] ={"COMPILE","EDIT","FILE","SEARCH",};
char **cp[] = {c+3,c+2,c+1,c};
char ***cpp = cp;
main()
  {
   printf("%s", **cpp);
   printf("%s", *--*++cpp+3);
   printf("%s", *cpp[-1)+3);
   printf("%s",cpp[-1][-1]+1);
  }
```

(a) SEARCHFILEEDITCOMPILE
(b) SEARCHCOMPILEEDIT
(c) SEARCHEPILEDIT
(d) SEARCHTRCHILE

8.128 What is the output of the following program?

```
main()
  {
   char buff[] = "This is a model test";
     int i, *ptr;
   ptr = buff;
  for (i=0;*ptr; i++)
  printf("%c",*ptr++);
  }
```

(a) This is a test
(b) It'll print junk
(c) Compilation error
(d) none of the above

ANSWERS

Fill in the Blanks

1. **memory**	2. **address of**	3. **static**	4. **one**
5. **zero**	6. *** and &**	7. **rvalue**	8. **right to left**
9. ***(a+i)+j**	10. **rows**	11. **address**	12. **sizeof**
13. **realloc()**	14. **address**	15. **complex declaration**	

True or False

1. **False**	2. **False**	3. **True**	4. **True**	5. **False**	6. **True**
7. **False**	8. **True**	9. **False**	10. **True**	11. **True**	12. **False**
13. **True**	14. **True**	15. **True**	16. **False**	17. **True**	

Match the Following

1. **3**	2. **1**	3. **2**	4. **5**	5. **4**

Objective Type Questions

1.**d**	21.**c**	41.**d**	61.**d**	81.**c**	101.**b**	121.**d**
2.**c**	22.**b**	42.**c**	62.**c**	82.**c**	102.**b**	122.**a**
3.**c**	23.**b**	43.**b**	63.**a**	83.**c**	103.**b**	123.**b**
4.**d**	24.**c**	44.**a**	64.**b**	84.**d**	104.**d**	124.**c**
5.**b**	25.**c**	45.**c**	65.**b**	85.**d**	105.**a**	125.**d**
6.**b**	26.**c**	46.**b**	66.**d**	86.**b**	106.**a**	126.**c**
7.**d**	27.**a**	47.**d**	67.**d**	87.**c**	107.**c**	127.**d**
8.**c**	28.**d**	48.**d**	68.**d**	88.**a**	108.**b**	128.**d**
9.**a**	29.**b**	49.**b**	69.**c**	89.**d**	109.**a**	
10.**d**	30.**a**	50.**a**	70.**b**	90.**d**	110.**d**	
11.**c**	31.**c**	51.**c**	71.**a**	91.**a**	111.**c**	
12.**a**	32.**b**	52.**d**	72.**d**	92.**c**	112.**a**	
13.**c**	33.**c**	53.**a**	73.**a**	93.**d**	113.**d**	
14.**a**	34.**d**	54.**c**	74.**d**	94.**d**	114.**c**	
15.**c**	35.**d**	55.**b**	75.**d**	95.**d**	115.**c**	
16.**c**	36.**d**	56.**a**	76.**b**	96.**c**	116.**c**	
17.**d**	37.**d**	57.**a**	77.**d**	97.**d**	117.**c**	
18.**d**	38.**d**	58.**b**	78.**c**	98.**a**	118.**a**	
19.**b**	39.**d**	59.**b**	79.**a**	99.**c**	119.**c**	
20.**c**	40.**c**	60.**d**	80.**a**	100.**C**	120.**b**	

Strings

9

A sequence of characters constitutes a string. We use strings in many applications and manipulate them to perform useful tasks. In C, a *string* is a *NULL terminated array* of *characters*. C does not have a type called string. However, a string can be defined in two ways—by declaring a character array and declaring pointer to character type.

Example:

1. *char str[30];*
2. *char *str;*

where *str* represents the name of the string.

The first declaration reserves memory for a string holding 30 characters. The second one declares a *character pointer*. This declaration of *character pointer* does not actually allocate memory for the string. We have to allocate memory by using a dynamic memory allocation function or assigning a valid address of another string.

STRING MANIPULATION

There are many string functions defined by library functions and they are helpful in manipulating strings for the purpose required. **Appendix D2** provides a table of different functions supported by ANSI C. The demonstrations of the string functions are given next.

strcat(*s*,*ct*)— concatenates string *ct* to end of string *s*; returns

Implementation of strcat() functions:

```
void strcat(char s[], char t[])
  {
     int i, j;
     i = j = 0;
     while (s[i] != '\0')                    /* find end of s */
        i++;
     while ((s[i++] = t[j++]) != '\0')       /* copy t */
        ;
  }
```

strcpy(*s*,*ct*)—copies string *ct* to string *s* including '\o'; returns *s*

Implementation of strcpy() function:

```
char *strcpy(char dest[], const char source[])
{
  int i = 0;
  while (source[i] != '\0')
  {
    dest[i] = source[i];
    i++;
  }
  dest[i] = '\0';
  return(dest);
}
```

strncmp(*cs*,*ct*,*n*)—compares at most *n* characters of string *cs* to string *ct*; returns a negative value if *cs*<*ct*, 0 if *cs*==*ct*, or positive value if *cs*>*ct*

Implementation of strncmp()

```
Int strncmp(const char *s1,const char *s2, register size_t n)
    {
  register unsigned char u1, u2;

  while (n– – > 0)
    {
     u1= (unsigned char) *s1 ++;
     u2= (unsigned char) *s2 ++;
     if (u1!=u2)
            return u1-u2;
     if (u1 == '\0')
            return 0;
    }
  return 0;
}
```

strchr(*cs*,*c*)—returns pointer to first occurrence of *c* in string *cs* or NULL if not present

Implementation of strchr()

```
char * strchr (register const char *s, int c)
  {
  do {
    if (*s == c)
      {
            return (char*)s;
      }
```

```
  } while (*s++);
  return (0);
}
```

strrchr(*cs*,*c*) —returns pointer to last ccurrence of *c* in string *cs* or NULL if not present

Implementation of strrchr()

```
char * strrchr (register const char *s, int c)
{
  char *rtnval = 0;

  do {
    if (*s == c)
      rtnval = (char*) s;
  } while (*s++);
  return (rtnval);
```

strlen(*cs*)—returns length of string *cs*

Implementation of strlen function

```
int my_strlen(char *s)
{
  char *p=s;

  while(*p!='\0')
    p++;

  return(p-s);
}
```

strspn(*cs*,*ct*)—returns length of prefix of string *cs* consisting of characters found in string *ct*

Implementation of strspn function

```
size_t (strspn)(const char *s1, const char *s2)
{
    const char *sc1;
    for (sc1 = s1; *sc1 != '\0'; sc1++)
        if (strchr(s2, *sc1) == NULL)
            return (sc1 – s1);
    return sc1 – s1;
}
```

strtok(*s*,*ct*)—searches *s* for tokens delimited by characters from *ct*.

Implementation of strtok function

```
char *(strtok_r)(char *s, const char *delimiters, char **lasts)
{
   char *sbegin, *send;
   sbegin = s ? s : *lasts;
   sbegin += strspn(sbegin, delimiters);
   if (*sbegin == '\0') {
      *lasts = "";
      return NULL;
   }
   send = sbegin + strcspn(sbegin, delimiters);
   if (*send != '\0')
      *send++ = '\0';
   *lasts = send;
   return sbegin;
}
/* strtok */
char *(strtok)(char *restrict s1, const char *restrict delimiters)
{
   static char *ssave = "";
   return strtok_r(s1, delimiters, &ssave);
}
```

strdup(*cs*)—creates duplicate of string *cs*

Implementation of strdup()

```
char * strdup (char *s)
    {
   char *result = (char*)malloc (strlen (s) + 1);
   if (result == (char*)0)
           return (char*)0;
   strcpy(result, s);
   return result;
}
```

strstr(*cs*,*ct*)—returns pointer to first occurrence of *ct* in string *cs*, or NULL if not present.

Implementation of strstr()

```
char * strstr (char *s1,char *s2)
  {
  register char *p = s1;
  extern char *strchr ();
  extern int strncmp ();
```

```
register int len = strlen (s2);
  for (; (p = strchr (p, *s2)) != 0; p++)
    {
      if (strncmp (p, s2, len) == 0)
              {
                return (p);
              }
    }
  return (0);
}
```

strlwr(*cs*)—translates alphabetic characters in string *cs* into lower case.

Implementation of strlwr()

```
char * strlwr (char *a)
{
  char *ret = a;

  while (*a != '\0')
    {
      if (isupper (*a))
              *a = tolower (*a);
      ++a;
    }
  return ret;
}
```

strupr(*cs*)—converts alphabetic characters in string *cs* to uppercase.

Implementation of strupr()

```
char *strupr (char *a)
     {
  char *ret = a;

  while (*a != '\0')
    {
      if (islower (*a))
              *a = toupper (*a);
      ++a;
    }

  return ret;
}
```

SHORT ANSWER QUESTIONS

9.1 What is a string?

The string is a sequence of a character terminated by NULL character.

9.2 What is the length of a string in C?

The length of a string is the number of non-null characters available in it.

9.3 How can you express a string in C?

You can express a string in C either by declaring a character array or by initializing a character pointer with a string constant.

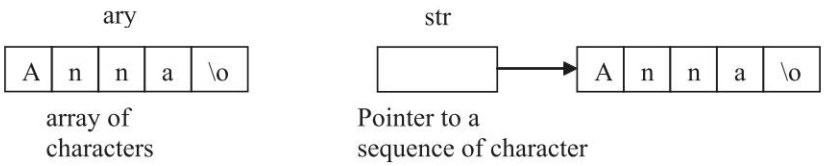
9.4 How is the end of a string recognized in C?

The null character '\o' is the end of the string or delimiter for a string in C.

9.5 What is the difference between string and array of character?

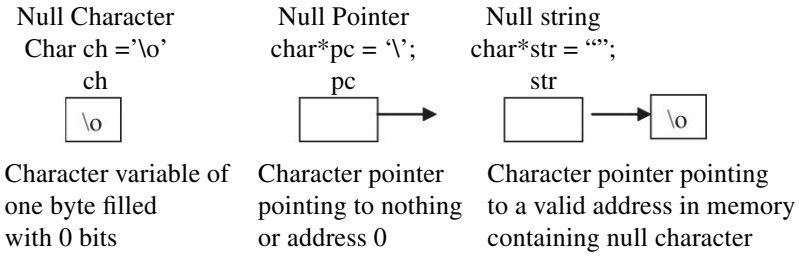
Consider the following declarations.

```
Char ary[] = "Anna"; /* 5 bytes long*/
Char *str = "Anna"; /* 9 bytes long, 4 for pointer, 5 for chars */
```



Character array holds only the characters in the string literal. Pointer to character holds the required bytes of memory for pointer variable plus the number of characters including NULL in the literal.

9.6 What is the difference between null character, null pointer and null string?



9.7 What are the ways to read a string into memory?

```
char ary[8];
scanf("%7s", ary);
```

```
char c[8];
char *str = c;
scanf("%7s", ary);
```

```
char *st;
st = (char *) malloc(8);
scanf(%7s", st);
```

9.8 Differentiate between gets() and scanf().

scanf() skips leading whitespace characters and stops at the first whitespace character. gets() reads and stores any character, including whitespace, up to the first new line character.

9.9 What is ragged array?

A ragged array is an array of strings that do not have equal number of columns for each row.

9.10 List out the applications of strings.

(1) Text formatting (2) String formatting
(3) Menu processing (4) password validation
(5) Searching keywords (6) Message construction, etc.

9.11 Write a program to count digits, white spaces and others.

```
#include <stdio.h>
   int main()
   {
        int c, i, nwhite, nother, ndigit[10];
        nwhite = nother = 0;
        for (i = 0; i < 10; i++)
            ndigit[i] = 0;
        while ((c = getchar()) != EOF) {
            switch (c) {
            case '0': case '1': case '2': case '3': case '4':
            case '5': case '6': case '7': case '8': case '9':
                ndigit[c-'0']++;
                break;
            case ' ':
            case '\n':
            case '\t':
                nwhite++;
                break;
            default:
                nother++;
                break;
            }
        }
        printf("digits =");
        for (i = 0; i < 10; i++)
            printf(" %d", ndigit[i]);
        printf(", white space = %d, other = %d\n",
            nwhite, nother);
        return 0;
   }
```

9.12 Write a program to reverse a string.

```
#include <string.h>
   void reverse(char s[])
   {
```

```
        int c, i, j;

        for (i = 0, j = strlen(s)-1; i < j; i++, j--) {
            c = s[i];
            s[i] = s[j];
            s[j] = c;
        }
    }
```

9.13 Write a program to remove trailing tabs, blanks and new lines from the end of a string.

```
int trim(char s[])
    {
        int n;
        for (n = strlen(s)-1; n >= 0; n--)
            if (s[n] != '' && s[n] != '\t' && s[n] != '\n')
                break;
        s[n+1] = '\0';
        return n;
    }
```

9.14 Write a program to left justify a string.

```
#include<stdio.h>
char* ljust(char *s)
{
static char temp[20];
int i,j;
i = j = 0;
for(i=0;s[i] != '\0';i++)
{
if(s[i] == '')
{
continue;
}
else
{
temp[j] = s[i];
j++;
}
}
temp[j] = '\0';
i = 0;
j = strlen(temp);
while(s[i] == '')
{
temp[j++] = s[i++];
```

```
}
temp[j] = '\0';
return temp;
}
int main()
{
char str[] = "Example..............."; // ....... indicate spac-
es..
printf("After left justification the string is %s",ljust(str));
}
```

9.15 Write a C program to reverse the words in a sentence.

```
int main(int argc, char *argv[])
{
   char buf[] = "the world will go on forever";
   char *end, *x, *y;
   for(end=buf; *end; end++);
   rev(buf,end-1);
   x = buf-1;
   y = buf;
   while(x++ < end)
   {
      if(*x == '\0' || *x == ' ')
      {
        rev(y,x-1);
        y = x+1;
      }
   }
  printf("%s\n",buf);
  return(0);
}
void rev(char *l,char *r)
{
   char t;
   while(l<r)
   {
      t    = *l;
      *l++ = *r;
      *r-- = t;
   }
}
```

9.16 Write a function to check if a word is a palindrome or not.

```
void isPalindrome(char *string)
{
```

```
    char *start, *end;
    if(string)
    {
       start = string;
       end   = string + strlen(string) - 1;
       while((*start == *end) && (start!=end))
       {
         if(start<end)start++;
         if(end>start)end--;
       }
       if(*start!=*end)
       {
          printf("\n[%s] - This is not a palidrome!\n", string);
       }
       else
       {
          printf("\n[%s] - This is a palidrome!\n", string);
       }
    }
    printf("\n\n");
}
```

9.17 Write a program using strcpy().

```
#include <stdio.h>
#include <string.h>
int main() {
  char input_str[] = "Hello";
  char *output_str;
  output_str = strcpy(input_str, "World");
  printf("input_str: %s\n", input_str);
  printf("output_str: %s\n", output_str);
  return 0;
}
```

9.18 Write a program using strncpy().

```
#include <stdio.h>
#include <string.h>
int main() {
  char str1[10] = "";
  char str2[10] = "";
  char str3[15] = "";
  char str4[10];
  char *str5;
  char str6[10] = "";
  char str7[10] = "123456789";
  char *source = "www.iota-six.co.uk";
```

```
  strncpy(str1, source, 9);
  strncpy(str2, source, 10);
  strncpy(str3, "www.iota-six.co.uk", 10);
  strncpy(str4, source, 9);
  strncpy(str5, source, 9);
  strncpy(str6 + 2, source, 7);
  strncpy(str7 + 2, source, 7);

  printf("%s\n", str1);
  printf("%s\n", str2);
  printf("%s\n", str3);
  printf("%s\n", str4);
  printf("%s\n", str5);
  printf("%s\n", str6);
  printf("%s\n", str7);

  return 0;
}
```

9.19 Write a program using strcmp() to compare strings.

```
#include <stdio.h>
#include <string.h>

int main() {
  char *name = "EDDIE";
  char *guess;
  /* char guess[50]; also works */
  int correct = 0;
  printf("Enter a name in uppercase: ");

  while(!correct) {
    gets(guess);
    if(strcmp(name, guess)==0) {
      printf("Correct!\n");
      correct = 1;
    }
    else {
      printf("Try again: ");
    }
  }
  return 0;
}
```

9.20 Write a program using strstr().

```
#include <stdio.h>
```

```
#include <string.h>
int main() {
  char *url = "http://www.iota-six.co.uk";
  printf("%s\n", strstr(url, "iota-six"));
  printf("%s\n", strstr(url, "iota-6"));
  if(strstr(url, "iota-seven")==NULL) {
    printf("iota-seven not found in %s\n", url);
  }
  printf("Length of %s is %d\n", url, strlen(url));
  return 0;
}
```

9.21 Write a program using strchr() and strrchr().

```
#include <stdio.h>
#include <string.h>
int main() {
  char *title = "Eddie's Basic Guide to C";
  char search_char = 'e';
  char *position;
  int index;

  position = strchr(title, search_char);
  /* search for first occurrence ... */

  if(position==NULL) {
    printf("%c not found in %s\n", search_char, title);
  }
  else {
    index = title - position;
    /* pointer arithmetic to work out index */

    if(index<0) {
      index *= -1;  /* take absolute value */
    }

    printf("First occurrence of %c in %s", search_char, title);
    printf(" is at index %d\n", index);

    position = strrchr(title, search_char);
    /* search for last occurrence */

    index = title - position;
```

```
    if(index<0) {
      index *= -1;
    }
    printf("Last occurrence of %c in %s", search_char, title);
    printf(" is at index %d\n", index);
  }
  return 0;
}
```

9.22 Write a program using strcat().

```
#include <stdio.h>
#include <string.h>

int main() {
  char str1[50] = "Hello ";
  char str2[] = "World";

  strcat(str1, str2);

  printf("str1: %s\n", str1);

  return 0;
}
```

9.23 Write a program using strncat().

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main() {
  char *str1 = "It is ";
  char *str2 = "raining sunny snowing foggy";
  char *str3;

 str3 = (char *)calloc(strlen(str1) + strlen(str2), sizeof(char));

  strcpy(str3, str1);
  strncat(str3, str2+8, 5);

  printf("str3: %s\n", str3);

  free(str3);
  return 0;
}
```

9.24 Write a program using strlen() and strcpy().

```
#include <stdio.h>
```

```
#include <string.h>   /* required for strlen and strcpy */

int main() {
  char array1[50];
  char array2[50] = "Boring!";
  int size;

  printf("Enter a string less than 50 characters: \n");
  gets(array1);

  size = strlen(array1);  /* work out the length */

  printf("\nYour string is %d byt%s long...\n",
          size, (size==1 ? "e" : "es"));
  printf("   ... and contains %d characters\n\n", size);
  printf("Before copying, array2[] contains \"%s\"\n", array2);
  strcpy(array2, array1);
  printf("Now array2[] contains \"");
  puts(array2);
  printf("\"\n");
  return 0;
}
```

9.25 Write a code to split a string at equal intervals.

```
#define maxLineSize 20
split(char *string)
{
  int i, length;
  char dest[maxLineSize + 1];
  i = 0;
  length= strlen(string);
  while((i+maxLineSize) <= length)
  {
     strncpy(dest, (string+i), maxLineSize);
     dest[maxLineSize - 1] = '\0';
     i = i + strlen(dest) - 1;
     printf("\nChunk : [%s]\n", dest);
  }
  strcpy(dest, (string + i));
  printf("\nChunk : [%s]\n", dest);
}
```

FILL IN THE BLANKS

1. A string is an array of_____.
2. The number of locations needed to store a string is_____more than the string's length.
3. A pair of double quotes with nothing between them is called_____.
4. A string literal is enclosed within_____.
5. The length of a null string is_____.

TRUE OR FALSE

1. There is no string type in C.
2. The no. of locations needed to store a string is the length of the string.
3. The quotes are part of the literals in case of string constants.
4. A double quote included in the string literal as \" is recognized as part of the literal.
5. The length of null string is one.
6. The declaration
   char string[] = "welcome";
   is equivalent to the declaration
   char string[] = {'w', 'e', 'l', 'c', 'o', 'm', 'e', '\o'};
7. Pointer to char and character array are one and the same as far as memory allocation is concerned.
8. Null string points to same byte filled with 0 bits.
9. The functions gets(), puts() are string functions to read and write.
10. String parsing is an application of string.

MATCH THE FOLLOWING

1.	char	1	256 symbols
2.	strlen	2	512 symbols
3.	ASCII	3	Distance between base address of array and null termination.
4.	sizeof	4	One byte
5.	unicode	5	Actual size of the array in bytes.

OBJECTIVE TYPE QUESTIONS

9.1 The number of bits used in extended ASCII is

(a) 7 (b) 8 (c) 16 (d) 32

9.2 Given a declaration and initialization statement as shown below,

```
char a[200]={'0'};
```

What is the output of the following printf statement?

```
printf("%d %c %c %d",a[0],a[0],a[3],a[3]);
```

(a) 48 0 0 (b) 44 0 0 (c) 46 0 (d) 49 0 49 0

9.3 Given a declaration and initilization as shown below,

```
char a[6] = {'1','2','3'};
```

What is the output of the following printf statement?

```
printf("%c %c %c %c %c %c ",a[0],a[3],a[4],a[2],a[4],a[1]);
```

(a) 1 2 3 4 5 6 (b) 1 2 3 4 5 (c) 1 3 2 (d) 1 2 3 0 0 0

9.4 char *a="23422";

printf("%d %d", sizeof(a),strlen(a));

From the above, what is the output of the printf function?

(a) 4 4 (b) 5 5 (c) 5 4 (d) 4 5

9.5

```
char name[]="MIT"
printf("%sisdepartment%s%dof%s/n",&name[1],"no",('A''a')/",name);
```

From the above, what is the output of the printf statement?

(a) IT is department no 1 of MIT
(b) L value required RUNTIME ERROR
(c) IT is department no –1 of MIT
(d) compilation error

9.6

```
char *p;
char name[25]="MIT-IT";
strcpy(p,name);
printf("%s",p);
```

What will be the output?

(a) segmentation fault because pointer p is only declared but no memory allocated
(b) compilation error
(c) MIT-IT
(d) none of the above

9.7

```
char name[10];
scanf("%s",name);
if(strcmp(name,"chennai")==0)
 {
  char a[10]="600028";
  printf("Mandaveli has the pin code: %s\n",a);
 }
```

What will be the amount of memory allocated in the following cases'?
case 1 input is not chennai,
case 2 input is chennai
(a) in both cases, the total amount of memory allocated will be the same
(b) the total amount of memory allocated differs based on the input
(c) based on the compiler options
(d) both (a) and (c)

9.8
```
char *name ="Delhi";
printf("%d %d ",sizeof(name),sizeof((*name));
```
What will be the output?
(a) 4 1 (b) 4 4 (c) 5 1 (d) 5 4

9.9 Which of the following is the correct output for the program given?
```
main()
{
    char str[]="Sales\0man\0";
    printf("%d %s %d \n",sizeof(str),str,strlen(str));
}
```
(a) 5 Sales 5
(b) 11 Sales 9
(c) 11 Sales man 9
(d) 11 Sales 5

9.10 Which of the following is the correct output for the program given?
```
char str[7]="strings"; printf("%s",str);
```
(a) error (b) strings (c) cannot predict (d) none of the above

9.11 If sizes of a char, an int, a float and a double are 1, 4, 4 and 8 bytes respectively, which of the following is the correct output for the program given below?
```
{
char ch='A';
printf("%d %d %d", sizeof(ch), sizeof(sizeof('A')), sizeof(3.14));
}
```
(a) 1 4 2 (b) 1 4 8 (c) 2 2 4 (d) 2 4 8

9.12 What is the value of str3 and strl?
```
void main()
{
  char strl[100]="united";
  char *str2="front";
  char *str3;
  str3=strcat(strl, str2)
  printf("\n%s", str3);
}
```
(a) compilation error because strcat does not return a value
(b) united front

(c) segmentation fault because str3 might be pointing to a garbage value
(d) united

9.13 Consider the following declarations:

```
char *a;
char a[5];
```

Which of the following statements is/are true?
1. both are of pointer type
2. memory allocation is different in both the cases
3. the value of a in declaration char a[5] can be changed later

(a) (2) only (b) (1) only (c) (1) and (2) (d) (2) and (3)

9.14 What will be the output?

```
char st[]="India";
char *p;
strcpy(p,st);
printf("%s and %s are same",st,p);
```

(a) India and India are same
(b) India and <some Garbage value> are same
(c) compilation error
(d) runtime segmentation fault

9.15

```
*1* char st[]="Mandaveli";
*2* char *p;
*3* strcpy(p,st);
*4* printf("%s and %s are same",st,p);
```

If the code does result in error, what is the solution to be taken?
(a) the code will work fine and print the intended output
(b) remove the statement 3 and edit statement 2 such that char *p = st;
(c) allocate dynamic memory for the pointer p in statement 2 as char *p=(char*)malloc (strlen(st)+1);
(d) both b and c

9.16 What will strcmp return, if both the strings are equal?
(a) boolean TRUE (b) 0 (c) 2 (d) 1

9.17 Which of the following function is used to check the presence of a substring?
(a) strstr() (b) substr()
(c) check string() (d) none of the above

9.18 What does atoi() function do?
(a) convert a string into an integer (b) convert string into a float
(c) convert integer to string (d) convert float to string

9.19 Given an integer 'a' and a float 'b', write a statement that copies the integer and float data into a character array?
(a) `strcat(str,a);strcat(str,b);` (b) `sprintf(str,"%d %f',a,b);`
(c) `sscanf(str,"%d %f',a,b);` (d) none of the above

9.20 What does strlen() return for the following character array?

```
char str[]="MIT\OIT";
```

(a) 5 (b) 6 (c) 3 (d) 7

ANSWERS

Fill in the Blanks

1. characters 2. one 3. null string
4. double quotes 5. zero

True or False

1. True 2. False 3. False 4. True 5. False 6. True
7. False 8. True 9. True 10. True

Match the Following

1. 4 2. 3 3. 1 4. 5 5. 2

Objective Type Questions

1.b	5.c	9.d	13.a	17.a
2.a	6.a	10.c	14.d	18.a
3.c	7.a	11.b	15.d	19.b
4.d	8.a	12.b	16.b	20.c

Structure and Union

10

Arrays and the pointers have been used to represent a group of data items having same data types. Sometimes, it may be necessary to organize a group of related data items having different data types as a single entity. For example, the data items like name, register number, branch and sex may be grouped and organized as a single entity to represent a student record. Each data item in the group is related to a student, but all the data items do not have the same data type. To represent such a complicated data, C provides data types such as structure and union.

A structure is a derived data type used to organize a group of related data items having different data types. The syntax of a structure declaration differs from that of a structure definition. The declaration informs the compiler about the prototype of the structure, whereas the definition creates the structure variable. The declaration alone does not reserve space, whereas the definition allocates space for storing the members. There are different ways to declare and define the structure variables. Sometimes, the definition is combined with the declaration. In such a situation, the memory space is also reserved. The members of the structure are accessed by the structure member operator, which is denoted by a dot (.). The structure variable can also be initialized. The structure cannot be read as a whole, but the individual members are read and displayed. But assignment of one structure variable to another one is possible.

A nested structure allows another structure as its member. An array of structures is used to organise a group of structures. A pointer to a structure is helpful in the creation and manipulation of a structure or a group of structures. To access a member using a pointer to structure, the operator –> is used. A self-referential structure is possible by having a member of a structure as a pointer to itself. It can be achieved directly or indirectly.

A structure or a pointer to a structure may be passed as a parameter to a function. The call by reference effect is achieved by passing a pointer to a structure as a function argument. A member of a structure alone may also be passed as an argument in a function. The parameter must be appropriately declared in the function definition.

A union is similar to a structure, except that only one member may be stored at a time in a union variable. Certain programs may be written using a union to conserve memory. Bit fields are used to pack data in a word of computer memory. They are used to store data in a specific number of bits. However, programs using bit fields are not portable because they are implementation dependent.

SHORT ANSWER TYPE QUESTIONS

10.1 What is a structure?

A structure is a derived data type to organize a group of related data items of different data types.

10.2 How is a structure created?

A structure requires a definition and a declaration. The declaration describes the prototype of the structure. It does not reserve memory. The declaration is followed by a definition that assigns memory for structure variables.

10.3 Give the syntax of structure declaration and structure definition.

Syntax for declaration:

```
struct tag
        {
           declaration of members;
        };
struct tag varl, var2,..., varN;     /* Definition */
```

The declaration and definition may be combined as:

```
struct tag
         {
           declaration of members;
         }varl, var2,..., varN;
```

10.4 What is the purpose of a tag in structure declaration?

The tag is an identifier and it is optional. This tag can be subsequently used as a shorthand notation, for the declaration part within the braces.

10.5 What is the need for the initialization of a structure?

A structure cannot be read as a whole. But by initialization, it can be assigned values for all the members in a single statement.

For example,

```
struct pas
        {
          char name[10];
          int age;
          char place [20]
        };
struct pas = { "POORANI", 10, "TIRUNELVELI" };
                          /*Initialization */
```

10.6 How is a member of a structure accessed?

Each member of a structure variable can be accessed using the structure member operator dot (.) as given below.

Format:

```
Structure_variable . member_variable
```

For example,

```
pas.age      /* For the definition in Question 9.5 */
pas.name
```

10.7 What is a nested structure?

If a structure contains one or more structures as its members, it is known as a nested structure.

For example,

```
struct date
          {
            int day;
            char month [10];
            int year;
          }
struct person                    /* Outer structure declaration */
            {
              char name[20];
              int age;
              struct date dob;  /* Inner structure definition */
};
      struct person man;         /* Outer structure definition */
```

10.8 What is the restriction in a nested structure?

The structure itself cannot be nested.

For example,

```
struct person
            {
              char name[20];
              int age;
              struct person man; /* Nesting the structure itself */
            };
```

The above example is invalid.

10.9 Define an array of structures.

A group of structures may be organized in an array resulting in an array of structures. Each element in the array is a structure.

For example,

```
struct person
       {
         char name[20];
         int age;
       };
struct person emp [10] ;      /* An array of 10 structures */
```

Each structure variable emp[0], emp[1], ... , emp[9] contains structure as its value.

The members are accessed as:

```
emp[0].age
emp[1].age
…
emp[9].age
```

10.10 Is it possible to initialize an array of structures?

Yes, it is possible to initialize an array of structures.
For example, Consider the declaration in Question 9.9.

```
struct person emp ={{ " POORANI", 10 }, {"BRINDHA", 7}};
```

10.11 Define a pointer to a structure.

A pointer to a structure is similar to a pointer to an ordinary variable. It is created in the same way as a pointer to an ordinary variable is created.

```
struct person *sp;  /*Declaration as given in Question 9.9 */
```

Now, sp is a pointer variable pointing to a structure person. This pointer must be initialized statically or dynamically.

For example,

```
struct person
            {
              char name[10];
              int age;
            };
struct person s, *spl, *sp2;
spl = &s;          /* Static allocation */
sp2 = ( struct person* ) malloc ( sizeof (s) ); /* Dynamic allocation*/
```

The individual members can be accessed as:

```
(*spl).name  (*spl).age
(*sp2).name  (*sp2).age
```

Care must be taken to enclose the pointer to structure and * within parentheses since * and . have same precedence and associativity of both is from right to left. To avoid this confusion, an operator -> is used as given below.

```
spl->name     sp1->age
sp2->name     sp2->age
```

10.12 What is a self-referential structure?

If a member of a structure is a pointer to itself, it is a self-referential structure.

For example,

```
struct vertex
            {
              int data;
             struct vertex *np;     /* Pointer to the same structure */
            };                      /* Self-referential structure */
```

A structure itself as a member is not allowed. But a pointer to the same structure is possible.

10.13 How is a structure declaration renamed?

A structure declaration may be used in typedef statement to give a new name as given below.

For example,

```
typedef struct vertex
                    {
                      int data;
```

```
        struct vertex *np;
    }  NODE;
```

Now, NODE is the new data type equivalent to the structure vertex.

```
NODE nl, n2, *n;          /* Structure definition */
```

n 1 and n2 are structure variables and n is a pointer to the structure vertex.

```
n=( NODE* ) malloc( sizeof( NODE ));
    /* Dynamic allocation of memory */
```

n–>data is the syntax for accessing the member **data**.

10.14 How is a structure passed as a function argument?

A function may pass a structure as an argument. If a structure variable is passed, the changes made in the members of the structure in the called function is not available to the calling function. To make these changes available in the calling function, the argument is passed as a pointer to the structure. By passing the pointer to a structure, the effect of call by reference is achieved.

10.15 What is a union? How is it declared?

A union has different types of data items in which each member shares the same block of memory. For example,

```
union mixed
       {
         int  i;
         float f;
         double d;
       } ;            /* union declaration */
union mixed mt;       /* union definition */
```

The dot notation is used to access the member. All the features of the structures are possible with a union.

10.16 What are the features of a structure?

- It permits different data items.
- Array of structures is possible.
- Pointer to a structure is possible.
- Nested structure is possible.
- Self-referential structure is possible.
- Structures can be passed as arguments.
- The members can be accessed and manipulated
- Union can be a member of a structure.

10.17 What are the features of a union?

- Each member of a union shares the same block of memory.
- A union can use a structure as its member.
- Array of unions, pointer to a union and function with union as its arguments are possible.
- Nested union may be used.
- Self referential union is also possible.

10.18 What is a bit field?

A bit field is one bit or a set of adjacent bits within a word. Bit fields can hold data items.

10.19 Declare bit field.

The syntax of a structure is used for defining and declaring bit fields.

Format:

```
struct tag
        {
            data_type mem1 : fieldwidth1;
            data_ type mem2 : fieldwidth2;
            …
            data_type memN : fieldwidthN;
        } varl, var2,..., varN;
```

The data type can be int, signed int or unsigned int only. Fieldwidth specifies the number of bits used by that member to store value.

10.20 What is unnamed bit field?

If a member is not given any name in a bit field declaration, it is known as unnamed bit field.

For example,

```
struct bit
     {
       int first_bit:1;
       unsigned:14;                    /* unnamed bit field */
       int last_bit:1;
     } bf;
bf.first_bit=1; bf.last_bit=1;         /* assignments */
```

10.21 What are the limitations of bit fields?

- Bit fields do not have addresses.
- They cannot be read using scanf() function.
- They cannot be accessed using pointers.
- They are not arrays.
- They cannot store values beyond their limits.

10.22 Compare array and structure.

S.No	Array	Structure
1.	An array is a collection of data items of same data type.	A structure is a collection of data items of different data types.
2.	It has declaration only.	It has declaration and definition.
3.	There is no keyword.	The keyword struct is used.
4.	An array name represents the address of the starting element.	A structure name is known as tag. It is a short hand notation of the declaration.
5.	An array cannot have bit fields.	A structure may contain bit fields.

10.23 Compare structure and union.

S.No	Structure	Union
1.	Every member has its own memory.	All members use the same memory.
2.	Keyword struct is used.	Keyword union is used.
3.	All members may be initialized.	Only its first member may be initialized.
4.	Different interpretations of the same memory location are not possible.	Different interpretations for the same memory location are possible.
5.	Consumes more space compared to union.	Conservation of memory is possible.

10.24 What are the uses of a union data type?

- Memory is economically used.
- If a particular member alone is to be manipulated, a union is a suitable structure.
- Structure may be used as its member.

10.25 What is wrong with the following code?

```
typedef struct
              {
                int id;
                NODE *next;
              } NODE;
```

The identifier NODE has not been defined at the point where the pointer field next is declared. The correct code may be written in different ways as given below.

```
1. typedef struct node NODE;
   struct node
     {
       int id;
       NODE *next;
     };
2. typedef struct node
             {
               int id;
               struct node *next;
             } NODE;
3. struct node
      {
        int id;
        struct node *next;
      };
   typedef struct node NODE;
```

10.26 What is the type of structure defined by the following code?

```
typedef struct first FIRST;
typedef struct second SECOND;
struct first
```

```
        {
          ...
          SECOND *s;
        }
    struct second
        {
          ...
          FIRST *f;
        };
```

The structure is defined by an indirect recursive declaration of a pointer to a structure. It results in a self-referential structure.

10.27 Is the following code legal? Justify.

```
    struct node
        {
          auto int x;
          auto float f;
        };
```

This is not a legal code. The compiler gives syntax error. By removing auto, it will work correctly. The structure members are accessed by structure variable outside this block only. Hence, the variables within a structure definition cannot have block scope by default. Inclusion of the storage class auto forces the variables to possess local scope. Hence, the above code will give syntax error. By removing auto, the code will work.

10.28

```
struct list
     {
       int x;
       struct list *next;
     }*head;
head.x = 100;
```

Whether the above code is correct or wrong?

Output:
Wrong. The field must be accessed as head–>x instead of head.x.

10.29 What is the output of the program?

```
#include <stdio.h>
main()
  {
    struct sl {int i; };
    struct s2 {lint i; };
    struct sl stl;
    struct s2 st2;
    stl. i =5;
    st2 = stl;
    printf(" %d " , st2.i);
  }
```

Output:
Syntax error. One struct variable cannot be assigned to another struct variable.

10.30 What is the output of the program?

```
main()
  {
   struct emp
      {
        char emp[15];
        int empno;
        float sal;
      };
   struct emp member = { "TIGER");
   printf(" %d %f", member.empno,member.sal);
  }
```

Output:
0 0.00

10.31 What is the output of the program?

```
struct xx
      {
        int a;
        long b;
      }s;
union yy
      {
        int a;
        long b;
      }u;
```

Print sizeof(s) and sizeof(u) if sizeof(int)=4 andsizeof(long)=4.

Output:
sizeof(s) 8 sizeof(u) 4

10.32 What is the output of the program?

```
#include <stdio.h>
main()
       {
         struct s
              {
                int x;
                float y;
              }sl={ 25, 45. 00 };
         union u
              {
               int x;
               float y;
```

```
        } ul;
    u1=(union u)s1;
    printf("%d and %f",ul.x,ul.y);
}
```

Output:
Error : illegal cast operation

10.33 For the following declaration

```
union x {
          char ch;
          int i;
          double j;
        }u_var;
```

What is the value of sizeof(u_var)?

The value of sizeof(u_var) is sizeof(double) since double type occupies maximum bytes.

10.34 What is the output of the program?

```
#include <stdio.h>
typedef struct NType
       {
         int i;
         char c;
         long x;
       } NewType;
main
  {
      NewType *c;
      c=(NewType *)malloc(sizeof(NewType));
      c->i=100;
      c->c='C';
      (*c).x=100L;
      printf("(%d,%c,%4Ld)",c->i,c->c,c->x);
  }
```

Output:
(100,C, 100)

FILL IN THE BLANKS

1. Related data items of different types are organized using_____.
2. The keyword_____is used to declare union.
3. Structure declaration describes the_____of a structure.
4. A structure name is represented by its_____.
5. Bit fields are declared in a_____.

6. The_____of a structure not followed by a structure variable does not reserve any storage.
7. The expression **(*ps).x** is equivalent to_____.
8. The variables named in a structure are_____.
9. Initialization of a union variable can initialize its_____ only.
10. The expression ++ps–>a increments_____.

TRUE OR FALSE

1. A structure may have data items of similar data types.
2. Structure declaration does not require a semicolon after closing brace.
3. The declaration **struct first{ int x; };** is equivalent to **struct second**{ int x; };.
4. The declaration and definiton cannot be combined.
5. A structure cannot contain the structure itself as its member.
6. The return statement can be used within a calling function of a structure.
7. The tag name of a structure can be used as an ordinary variable name or a member variable name without any conflict in a program.
8. A structure can be read and displayed as a whole.
9. The square brackets of an array of structures must be associated with the array name and not with any structure member name to access the members of the structure.
10. Unions can be used as members of structures and structures can be used as members of unions.
11. Nested unions and self referential unions are not possible in C.
12. Structures cannot be passed to functions.

MATCH THE FOLLOWING

1	Nested structure	Operator
2	Self-referential structure	Special type of structure.
3	Union	A structure containing a pointer to structure itself as its member.
4	Bit fields	A structure having another structure as its member.
5	Sizeof()	Set of adjacent bits in a computer word of memory.

OBJECTIVE TYPE QUESTIONS

10.1 Structure is a

(a) scalar data type
(b) derived data type
(c) both options a and b
(d) primitive data type

10.2 Structure is a data type in which
(a) each element must have the same data type.
(b) each element must have pointer type only.
(c) each element may have different data type.
(d) no element is defined.

10.3 C provides a facility for user defined data type using
(a) pointer (b) function (c) structure (d) array

10.4 The keyword used to represent a structure data type is
(a) structure (b) struct (c) struc (d) structr

10.5 Structure declaration
(a) describes the prototype
(b) creates structure variable
(c) defines the structure function
(d) is not necessary

10.6 Structure definition
(a) describes the prototype
(b) creates structure variable
(c) defines the structure function
(d) is not necessary

10.7 Identify the wrong syntax.
(a) `typedef struct {member declaration;} NAME; NAME V1,V2;`
(b) `typedef struct tag{member declaration;} NAME; NAME V1,V2;`
(c) `typedef struct {member declaration;} NAME; NAME V1,V2;`
(d) `typedef struct tag {member declaration;}NAME; NAME V1,V2;`

10.8 Identify the wrong syntax.
(a) `struct tag {member declaration;};`
(b) `struct tag {member declaration;}V1,V2;`
(c) `struct tag {member declaration;}`
(d) `struct tag {member declaration;} V1, V2;`

10.9 The operator used to access the structure member is
(a) * (b) . (c) [] (d) &

10.10 A structure
(a) can be read as a single entity.
(b) cannot be read as a single entity.
(c) can be displayed as a single entity.
(d) has member variables that cannot be individually read.

10.11 A structure can have
(a) pointers as its members
(b) scalar data type as its members
(c) structure as its members
(d) all the above

10.12 The structure declaration

```
struct person { char name[20]; int age; struct person woman;};
```

(a) is a valid nested structure
(b) is not a valid nested structure
(c) uses an invalid data type
(d) is a self-referential structure

10.13 A structure
(a) allows array of structure
(b) does not allow array of structures
(c) does not use array as its members
(d) is a scalar data type

10.14 In a structure definition,

(a) initialization of structure members are possible
(b) initialization of array of structures are possible
(c) both options a and b
(d) initialization of array of structures are not possible

10.15 The operator exclusively used with pointer to structure is

(a) . (b) –> (c) [] (d) *

10.16 If one or more members of a structure are pointer to the same structure, the structure is known as

(a) nested structure (b) invalid structure
(c) self-referential structure (d) structured structure

10.17 If one or more members of a structure are other structures, the structure is known as

(a) nested structure (b) invalid structure
(c) self-referential structure (d) unstructured structure

10.18 What type of structure is created by the following definition?

```
struct first {... ; struct second *s;};
struct second {...;  struct first *f;};
```

(a) Nested structure (b) Self-referential structure
(c) Invalid structure (d) Structured structure

10.19 The changes made in the members of a structure are not available in the calling function if

(a) pointer to structure is passed as argument
(b) the members other than pointer type are passed as arguments
(c) structure variable is passed as argument
(d) both options b and c

10.20 The changes made in the members of a structure are available in the calling function if

(a) pointer to structure is passed as argument
(b) structure variable is passed
(c) the members other than pointer type are passed as arguments
(d) both options a and c

10.21 Identify the wrong statement.

(a) Structure variable can be passed as argument.
(b) Pointer to structure can be passed as argument.
(c) Member variable of a structure can be passed as argument.
(d) None of the above.

10.22 Union is

(a) a special type of structure (b) a pointer data type
(c) a function data type (d) not a data type

10.23 The restriction with union is

(a) The last member can only be initialized.
(b) The first member can only be initialized.
(c) Any member can be initialized.
(d) Union cannot be initialized.

10.24 Union differs from structure in the following way.
(a) All members are used at a time. (b) Only one member can be used at a time.
(c) Union cannot have more members. (d) Union initializes all members as structure.

10.25 Identify the correct statement.
(a) Unions can be members of structures. (b) Structures can be members of unions.
(c) Both options a and b. (d) Union can contain bit field.

10.26 What is not possible with union?
(a) Array of union (b) Pointer to union
(c) Self-referential union (d) None of the above

10.27 Structure is used to implement the data structure.
(a) stack (b) queue (c) tree (d) all the above

10.28 The nodes in a linked list are implemented using
(a) self-referential structure (b) nested structure
(c) array of structure (d) ordinary structure

10.29 Identify the wrong statement.
(a) An array is a collection of data items of same data type.
(b) An array declaration reserves memory space and structure declaration does not reserve memory space.
(c) Array uses the keyword array in its declaration.
(d) A structure is a collection of data items of different data types.

10.30 Identify the wrong statement.
(a) An array can have bit fields. (b) A structure may contain bit fields.
(c) A structure has declaration and definition. (d) A structure variable can be initialized.

10.31 A bit field is
(a) a pointer variable in a structure.
(b) one bit or a set of adjacent bits within a word.
(c) a pointer variable in a union.
(d) not used in C.

10.32 A bit field can be of
(a) int (b) float (c) double (d) all the above

10.33 Identify the wrong statement(s).
(a) Bit fields have addresses (b) Bit fields can be read using scanf( )
(c) Bit fields can be accessed using pointer (d) all the above

10.34 Identify the correct statement(s).
(a) Bit fields are not arrays.
(b) Bit fields cannot hold the values beyond their limits.
(c) Bit fields may be unnamed also.
(d) All the above.

10.35 About structures which of the following is true.
1. Structure members are aligned in memory depending on their data type.
2. The size of a structure may not be equal to the sum of the sizes of its members.
(a) only option 1 (b) only option 2
(c) both options 1 and 2 (d) neither option 1 nor 2

10.36
```
struct date
    {
      int day;
      int month;
      int year;
    };
main()
    {
      struct date *d;
      ...
      ++d->day;      /*statementN*/
      ...
    }
```
Then the statement statementN
(a) increments the pointer to point the month (b) increment the value of day
(c) increment d by sizeof(struct date) (d) none

10.37
```
struct cost_record
      {
      long cost_no;
      char cost_name[31];
      double current_bal;
      }   CUST_REC;
```
Is the sample code above a usable structure variable declaration?
(a) Yes. CUST_REC can be used to access its members with the "–>" operator.
(b) No. A typedef must be added before "struct".
(c) Yes. CUST REC can be used to access its members with the "." operator.
(d) No. CUST REC must be in lowercase letters.

10.38
```
struct car
       {
         int speed;
         char type[10];
       } vehicle;
struct car *ptr;
ptr = &vehicle;
```
Referring to the sample code above, which of the following will make the speed equal to 200?
(a) (*ptr).speed = 200; (b) (*ptr)–>speed = 200;
(c) *ptr.speed = 200; (d) &ptr.speed = 200;

10.39 Consider the following structure.
```
  struct numname
      {
        int no;
        char name[25];
      };
```

```
struct numname nl[] ={
                        {12," Raja "},
                        {15,"Selvan"},
                        {18,"Prema"},
                        {21,"Naveen"}
                      };
```

The output for the following statement would be:

```
printf("%d,%d",nl[2].no,( *(nl + 2 )).no);
```

(a) 18,ASCII value of p (b) 18,18
(c) 18,ASCII value of r (d) 18,ASCII value of e

10.40 `struct customer *ptr = malloc( sizeof( struct customer) );`

Given the sample allocation for the pointer "ptr" found above, which statement would be used to reallocate ptr to be an array of 10 elements?

(a) `ptr = realloc( ptr, 10 * sizeof( struct customer ));`
(b) `ptr = realloc( ptr, 9* sizeof( struct customer ));`
(c) `realloc( ptr, 9* sizeof( struct customer ));`
(d) `realloc( ptr, 10 * sizeof( struct customer ));`

10.41 Which of the following will define a type NODE that is a node in a linked list.

(a)
```
struct node
       {
         NODE *next;
         int x;
       };
typedef struct node NODE;
```

(b)
```
typedef struct NODE
{
  struct NODE *next;
  int x;
};
```

(c)
```
typedef struct node NODE;
struct node
{
  NODE *next;
  int x;
};
```

(d)
```
typedef struct
{
  NODE *next;
  int x;
} NODE;
```

10.42 The size of the following union, where an int occupies 4 bytes of memory is

```
union arc
      {
        char x;
```

```
        int y;
        char ax [8];
      }aha;
```

(a) 16 bytes (b) 13 bytes (c) 8 bytes (d) 4 bytes

10.43
```
union rainbow
        {
          int a[ 5 ];
          float x[ 5 ];
        };
union rainbow color[ 20 ];
void *ptr = color;
```
Which of the following is the correct way to increment the variable "ptr" to point to the next member of the array from the sample above?

(a) `ptr = ptr + sizeof(rainbow.a);`
(b) `ptr = (void*) ((union rainbow*) ptr + 1);`
(c) `ptr = ptr + sizeof(*ptr);`
(d) `++(int*)ptr;`

10.44 Which is the valid declaration?

(a) `#typedef struct { int i;}in;` (b) `typedef struct in {int i;};`
(c) `#typedef struct int {int i;};` (d) `typedef struct {int i;} in;`

10.45 The following statement is

"The size of a struct is always equal to the sum of the sizes of its members."

(a) valid (b) invalid
(c) can't say (d) insufficient information

10.46
```
struct adr
    {
      char *name;
      char *city;
      int zip;
    };
struct adr *adradr;
```
Which are the valid references?

(i) `adr->name` (ii) `adradr->name` (iii) `adr. zip` (iv) `adradr.zip`

(a) options (i) and (iii) (b) option (ii) only
(c) options (ii) and (iv) (d) option iv only

10.47
```
struct
    {
      int x;
      int y;
  }abc;
```
you cannot access x by the following:

1. `abc->x;` 2. `abc[0]->x;` 3. `abc.x;` 4. `(abc)->x;`

(a) options 1, 2, and 4 (b) options 2 and 3
(c) options 1 and 2 (d) options 1, 3, and 4

10.48 What is the size of 'q' in the following program?
Assume int takes 4 bytes.

```
union
    {
      int x;
      char y;
      struct
          {
              char x;
              char y;
              int xy;
          }p;
    }q;
```

(a) 11 (b) 6 (c) 4 (d) 5

10.49 What is the output of the following code?

```
union
    {
      int no;
      char ch;
    } u;
  u.ch = '2';
  u.no = 0,
  printf ("%d", u.ch);
```

(a) 2 (b) 0 (c) null character (d) none

10.50 Which of these are valid declarations?

(i)
```
union (
        int i;
        int j;
       };
```

(ii)
```
union u-tag {
              int i;
              int j;
            )u;
```

(iii)
```
union {
        int i;
        int j;
       FILE *k;
       };
```

(iv)
```
union {
        int i;
        int j;
      }U;
```

(a) all are correct (b) options (i), (ii), and (iv)
(c) options (ii) and (iv) (d) option (ii) only

10.51 What is the size of ptrl and ptr2?

```
struct x
   {
     int j;
     char k[100];
```

```
        unsigned i;
        };
    int *ptrl:
    struct x *ptr2;
```

(a) same depending on the model used (b) 2,104
(c) 2, undefined for memory is not allocated (d) 2,4

ANSWERS

Fill in the Blanks

1. **structure** 2. **union** 3. **prototype** 4. **tag**
5. **structure** 6. **declaration** 7. **ps->x** 8. **members**
9. **first member** 10. **a**

True or False

1. **True** 2. **False** 3. **False** 4. **False** 5. **True** 6. **True**
7. **True** 8. **False** 9. **True** 10. **True** 11. **False** 12. **False**

Match the Following

1. **4** 2. **3** 3. **2** 4. **5** 5. **1**

Objective Type Questions

1. **b**	11. **d**	21. **d**	31. **b**	41. **c**	51. **a**
2. **c**	12. **b**	22. **a**	32. **a**	42. **c**	
3. **c**	13. **a**	23. **b**	33. **d**	43. **b**	
4. **b**	14. **c**	24. **b**	34. **d**	44. **d**	
5. **a**	15. **b**	25. **c**	35. **c**	45. **c**	
6. **b**	16. **c**	26. **d**	36. **b**	46. **b**	
7. **d**	17. **a**	27. **d**	37. **c**	47. **a**	
8. **c**	18. **b**	28. **a**	38. **a**	48. **b**	
9. **b**	19. **d**	29. **c**	39. **b**	49. **b**	
10. **b**	20. **a**	30. **a**	40. **a**	50. **c**	

Files and Preprocessors

11

Files are helpful in providing permanent storage of input/output (I/O) that can be accessed at any point of time later. Actually, voluminous data may be stored permanently in the form of files in hard disks or in auxiliary storage devices. A file is a region of memory space in the storage media and it can be accessed using the library functions available in the header file stdio.h or by the system-calls of the operating systems. The files, which are accessed by using the library functions are known as **high level files** and those by system-calls are known as **low level files**. High level files make use of the library functions common to all the operating systems and hence the programs using high level files are portable. Low level files make use of the system-calls of the operating system under which the program is run.

High level files make use of streams to transfer data. A stream is a pointer to a buffer of memory which is used for transferring the data. The I/O streams are useful for performing the I/O operations in a file. An I/O stream may be a text stream or a binary stream. A text stream contains lines of text and the characters in a text stream may be modified to match the environment. But, a binary stream is a sequence of unprocessed bytes without any modification in it.

A file created by using a binary stream is a **binary file** and a file created by using a text stream is a **text file**. A file must be opened before it is processed. File opening links the program with the file. The link is made through a file pointer, a pointer to FILE, which is defined and declared in the header file stdio.h.

There are many built-in functions to read/write using the text streams and binary streams. Functions must be used following the syntax properly for performing the I/O operations. A file may be closed after processing. It is very important to know the value returned by each function on successful action and on error to use it in a program. It is also possible to read from a string and write to a string using the functions sscanf() and sprintf().

A preprocessor is a facility provided for writing portable programs, easier program modifications and easier debugging. One file can be included in another file to provide common declarations and functions. Symbolic constants and macros can be defined and used in a program to improve the readability of the program. Conditional compilation is helpful for program testing and for executing the specific part of a program which enable portability.

An enumerated data type is an integer data type following the syntax of the structure. User defined names may be conveniently created in a program with the help of an enumerated data type.

The type qualifiers const and volatile may be used to indicate the specific properties of the variables declared. To incorporate the entire character set of C in the ISO 646-1983 Invariant Code Set, trigraph se-

quences are used. Functions with variable arguments are defined to pass different numbers of parameters in a function call. Built-in time functions can be used to calculate the execution time of a program.

SHORT ANSWER TYPE QUESTIONS

11.1 What is a file?

A file is a region of storage in hard disks or in auxiliary storage devices. It contains bytes of information. It is not a data type.

11.2 List the two type of files in C.

(a) High level files (stream oriented files): These files are accessed using library functions.
(b) Low level files (system oriented files): These files are accessed using system calls.

11.3 What is a stream?

A stream is a source of data or destination of data that may be associated with a disk or other I/O devices. The source stream provides data to a program and it is known as an input stream. The destination stream receives the output from the program and is known as an output stream.

11.4 Name the type of I/O streams.

(a) Text stream: It is a sequence of lines of text.
(b) Binary stream: It is a sequence of unprocessed bytes.

11.5 What is meant by file opening?

The action of connecting a program to a file is called opening of that file. This requires creating an I/O stream before reading or writing the data.

11.6 What is FILE?

FILE is a predefined data type. It is defined in stdio.h file.

11.7 What is a file pointer?

The pointer to FILE data type is called as a stream pointer or a file pointer. A file pointer points to the block of information of the stream that has just been opened.

11.8 Which header file is required for file operations? How can it be included?

The header file **stdio.h** is necessary for file handling because the required functions, definitions and declarations are available in it. It is included as given below:

```
# include <stdio.h>
```

11.9 How is fopen() used?

The function fopen() returns a file pointer. Hence, a file pointer is declared and it is assigned as given below.

```
FILE *fp;
fp = fopen(filename, mode);
```

filename is the string representing the name of the file and **mode** represents:

"r"	for read operation,
"w"	for write operation,
"a"	for append operation, and
"r+", "w+", "a+"	for update operation.
"r+b" "w+b" "a+b" "rb" "wb" "ab"	} modes for binary stream.

For example,

```
fp = fopen( "inventory", "r" );
```

The file named inventory is opened in read mode. The file pointer returned by fopen() is assigned to fp.

11.10 How is a file closed?

A file is closed using fclose() function.

For example,:

```
fclose(fp);
```

where fp is a file pointer.

11.11 List the common functions used for reading from a text stream.

1. fgetc() 2. Getc()
3. fgets() 4.fscanf() (Refer to Appendix D to know the arguments.)

11.12 List the common functions used for writing from a text stream.

1. fputc() 2. putc()
3. fputs() 4. Fprintf() (Refer to Appendix D to know the arguments.)

11.13 What is a random access file?

A file can be accessed at random using **fseek**() function.

Format:

```
fseek(fp, position, origin);
```

where

fp – file pointer.
position – number of bytes offset from origin.
origin – 0,1,or 2 to denote the beginning, current position or end of file respectively.

10.14 What is the purpose of ftell()?

The function ftell() is used to get the current file position of the file represented by the file pointer.

For example,

```
ftell(fp);
```

returns a long int value representing the current file position of the file pointed by the file pointer fp. If an error occurs, –1 is returned.

11.15 What is the purpose of rewind()?

The function **rewind()** is used to bring the file pointer to the beginning of the file.

Format:

```
rewind(fp);
```

where fp is a file pointer. It is possible to get the same effect by

```
fseek(fp, 0, 0);
```

11.16 What are the I/O functions used for reading and writing binary streams?

The functions **fread()** and **fwrite()** are unformatted I/O functions used for reading and writing using binary streams. The functions **fscanf()** and **fprintf()** are used as formatted I/O functions. The file must be opened in the binary stream mode before using these functions.

Format:

```
fread(bufptr, size, count, fp);
fwrite(bufptr, size, count, fp);
```

where

bufptr	– pointer to a buffer.
size	– size in bytes of one element in buffer.
count	– number of elements in the buffer.
fp	– file pointer.

11.17 What are the functions used to read from and write to strings?

1. sscanf()
2. sprintf()

For example,

1.
```
int x = 45;
char string [10] ;
sprintf( string, "%d", x );
```

The effect of the above statement is the same as strcpy(string, "45"); the data "45" is copied to the character array string.

2.
```
int x;
char ch;
char string[20] = "48B";
sscanf( string, "%d%c", &x, &ch );
```

x will be assigned 48 and ch is assigned the character B.

11.18 What is a preprocessor?

A preprocessor processes the source code program before it passes through the compiler.

11.19 What are the facilities provided by a preprocessor?

- File inclusion
- Substitution facilities
- Conditional compilation

11.20. What are directives in C?

Directives are preprocessor control lines that control the preprocessor facilities. They start with the symbol #.

11.21 What are the two forms of #include directive?

1. `#include "filename"`
2. `#include <filename>`

The first form is used to search the directory that contains the source file. If the search fails in the home directory it searches the implementation defined locations. In the second form, the preprocessor searches the file specified only in the implementation defined locations.

11.22 What are the substitution facilities available?

1. Defining manifests
2. Defining macros

Manifest is defined as:

```
# define NAME value
```

For example,

```
# define MAX 10
```

Macro is defined as:

```
# define NAME(arg) expression
```

For example,

```
# define SQUARE(X) ((X) * (X))
```

11.23 What is a macro?

A macro is a simple function having its own syntax using #define. It is defined with one or a few more statements. It is invoked in the same way as a function call.

For example:

```
# define CUBE(x) ((x) * (x) * (x))
```

11.24 What is conditional compilation?

A section of a source code may be compiled conditionally using the conditional compilation facilities. The directives used for this purpose are:

```
# if
# elseif
# else
# endif
```

These directives behave much like the if-else or if-else-if control structure.

11.25 What are the advantages of a preprocessor?

- A preprocessor improves the readability of programs.
- It facilitates easier modifications.
- It helps in writing portable programs.
- It enables easier debugging.
- It enables testing a part of a program.
- It helps in developing generalized program.

11.26 What is an enumerated data type?

Enumeration is a special integer data type. It associates the integer constants to the identifiers of the programmer's choice.

Format:

```
enum tag
      {
       enumerator_1,
       enumerator_2,
       ...
       enumerator_n
      };
```

where

tag	– name of enumeration
enum	– keyword representing enumerated data type
enumerator 1, enumerator 2,..., enumerator_n	– list of identifiers.

For example,

```
enum flowers
           { ROSE, JASMINE, LOTUS, MEHANDI );
```

ROSE assumes the value 0, JASMINE 1, LOTUS 2 and MEHANDI 3.

11.27 What are the features of enumerators?

- The values of the enumerators need not be distinct in the same enumeration type.
- The names of the enumerators in different enumeration types must be distinct.
- The names of the enumerators must be different from other normal variables.
- Enumerators are treated only as integer constants by the compiler.

11.28 What is the advantage of an enumerated data type?

The program can be written in a readable form so that in the place of integer constants suitable names can be used.

11.29 What are the macros and data type used for functions with variable arguments?

`va_start( ) va_arg( ) va_end( )`

va_list is a predefined data type, which is a pointer to the argument list.

11.30 What is a trigraph sequence?

The characters absent in the invariant code set are represented by trigraph sequences.

Character	Trigraph
#	??=
\	??/
^	??'
[	??(
]	??)
|	??!
{	??<
}	??>
~	??–

11.31 Differentiate between fgets() and gets().

	gets()	fgets()
1.	The function gets() is normally used to read a line of string from keyboard.	The function fgets() is used to read a line of string from a file or keyboard.
2.	It automatically replaces the '\n' by '\0'	It does not automatically delete the trailing '\n'.

3.	It takes one argument.
	`char s[10];`
	`gets(s);`
		It takes three arguments.
		`char s[50]; int n;`
		`FILE *fp;`
		`fgets(s,n,fp);/*` to read from a file*/
		`fgets(s,n,stdin);`
		/* to read from keyboard */

4.	It does not prevent overflow.	It prevents overflow.

11.32 Is it possible to compare string in a #if expression?

No. The # if expression uses only constant integer expressions. Hence, it is not possible to compare strings.

11.33 Does the sizeof operator work in preprocessor #if directives?

No. Since processing is done before parsing the type names, it is not possible to use sizeof operator in #if directive.

11.34. What are the uses of # pragma?

The # pragma directive provides a single, well-defined implementation specific controls and extensions such as source listing control, structure packing and warning suppression. Refer the manual of your compiler for details.

11.35 What is a stringizing operator?

The macro parameters in the strings that appear in the macro definitions are not recognized. To substitute a macro argument in a string constant, a special notation # is used in the macro body. When the # character precedes a parameter in the macro body, a string constant is replaced for both # and the parameter. The notation # in this context is known as a stringizing operator.

For example,

```
# define STRING(x,y) #x"developed by"#y
main()
  { char s[] = STRING( PROGRAM, S.T.SELVI );
    printf("%s\n",s);
  }
```

Output:

PROGRAM developed by S.T. SELVI.

11.36 What is token pasting operator?

A notation # # used in a macro definition concatenates the two tokens on either side of the symbol # # into one token. If the concatenation results in an invalid token, the result is undefined. The notation # # is called as token pasting operator. It cannot appear at the beginning or at the end of the macrobody. For example,

```
#define DECIMAL(X)    3.##X
main()
    { printf("%f\n", DECIMAL(14)); }
```

Output:

3.140000

The following example yields an invalid token. Hence, error.

```
#define TOKEN(X)  1##X
main()
    { printf"%d\n", TOKEN(5));
      printf"%d\n", TOKEN(a));
    }
```

The second printf() results in an invalid token la. Hence, compilation error is produced. The first printf() will yield a valid token 15.

10.37 What is the output of the following code?

```
#define MAN(x,y)  (x)>(y)?(x):(y)
main()
    {
      int i=10,j=5,k=0;
      k= MAN(i++,++j);
      printf("%d %d %d ",i,j,k);
    }
```

The output of the above code is
12 6 11

11.38 In the following enumeration declaration, determine the value of each member.

```
enum compass {north =2, south,east=l,west};
```

Output:
north = 2, south=3, east = 1, west =2.

11.39
```
main()
  {
  enum {ELLIPSE, TRIANGLE, RECTANGLE, SQUARE=100, CIRCLE=5
  printf("%d %d %d \n",TRIANGLE-RECTANGLE, SQUARE*CIRCLE,
                                    SQUARE-RECTANGLE);
  }
```

What is the output?

Output:
–1 500 98

11.40 What is the output of the program?

```
#define prn(a) printf("%d ",a)
#define print(a,b,c) prn(a), prn(b), prn(c)
#define max(a,b) (a<b)? b:a
main()
  {
    int x=1, y=2;
    print(max(x++,y),x,y);
    print(max(x++,y),x,y);
  }
```

Output:
2 2 2 3 4 2

11.41 Debugging is done by finding
(a) logical errors (b) run time errors
(c) both options a and b (d) none of the above

Output:
(c)

11.42
```
#define swapl(a,b) a=a+b;b=a-b;a=a-b;
main()
    {
      int x=5,y=10;
      swapl(x,y);
      printf("%d %d\n",x,y);
      swap2(x,y);
      printf("%d %d\n",x,y);
    }
int swap2(int a,int b)
     { int temp;
```

```
        temp=a; b=a; a=temp;
        return; }
```

Output:

10 5
10 5

FILL IN THE BLANKS

1. A_____is a source or destination of data that may be associated with a disk or an I/O device.
2. An I/O stream may be a_____or a_____.
3. The mode used for opening an existing file for reading a binary stream is_____.
4. The function fopen() returns_____ if the open operation fails.
5. The standard stream pointers are_____,_____and_____.
6. The function getc(stdin) is equivalent to_____.
7. The function call_____gets the current file position of the file represented by the file pointer.
8. The notation_____is called token pasting operator.
9. Enumerations are treated as_____by the compiler.
10. The identifier_____is a predefined data type, which is a pointer to the argument list in functions with variable arguments.
11. When no value is explicitly assigned, the first identifier in an enumeration list represents_____ by default.
12. The_____variable may change the value of it without the program's knowledge or action.

TRUE OR FALSE

1. Macros are not possible in C.
2. The notation # is known as stringizing operator.
3. The names of the enumerators must be different from other normal variables.
4. Declaration of const variables must have initializers.
5. Declaring a structure with the qualifier const makes every element in it constant.
6. The NUL character indicates an end of a string and a file.
7. The EOF character can be included in a file as part of its data.
8. The function call putc(c,stdout) is equivalent to the function call putchar(c) .
9. FILE is a structure data type.
10. The function main() may take a variable number of arguments.

MATCH THE FOLLOWING

1	System oriented files	A region of storage
2	Stream oriented files	Binary stream
3	File	Invariant code set
4	Trigraph	High level files
5	Binary file uses	Low level files

OBJECTIVE TYPE QUESTIONS

11.1 File is

(a) a data type (b) a region of storage
(c) both options a and b (d) a variable

11.2 The way to access file contents from a program is

(a) by using library functions (b) by using system calls
(c) both options a and b (d) to use a linker

11.3 Stream oriented files are accessed through

(a) system calls (b) library functions
(c) linker (d) loader

11.4. Low level files are accessed through

(a) system calls (b) library functions (c) linker (d) loader

11.5 C supports

(a) high level files (b) low level files
(c) both options a and b (d) executable files only

11.6 A stream is

(a) a library function
(b) a system call
(c) a source or destination of data that may be associated with a disk or other I/O devices
(d) a file

11.7 I/O stream can be

(a) a text stream (b) a binary stream
(c) both options a and b (d) an I/O operation

11.8 File opening is

(a) a default action in file processing
(b) an action of connecting a program to a file
(c) not necessary
(d) not using any library function

11.9 FILE defined in stdio.h is

(a) a region of storage (b) a data type (c) not a data type (d) a variable

11.10 A file pointer is
(a) a stream pointer (b) a buffer pointer
(c) a pointer to FILE data type (d) all the above

11.11 The default stream pointers available during execution of a program is
(a) stdin (b) stdout (c) stderr (d) all the above

11.12 The function call fopen("data", "w+b")
(a) is invalid
(b) returns the file pointer pointing to file named data and opens the file for reading and writing using binary stream
(c) returns the file pointer pointing to file named data and opens the file for reading and writing using text stream
(d) does not return file pointer

11.13 If fopen() fails, it returns
(a) –1 (b) NULL (c) 1 (d) the file pointer

11.14 The function fclose() is
(a) used to disconnect a program from file (b) used to close a file logically
(c) both options a and b (d) a mandatory function call in file handling

11.15 The action of connecting a program to a file is obtained by using I
(a) connect() (b) fopen() (c) OPEN() (d) file()

11.16 The action of disconnecting a program from a file is obtained by the function
(a) fclose() (b) delete() (c) fdisconnect() (d) clear()

11.17 The value returned by fclose(), if an error occurs is
(a) 0 (b) 1 (c) EOF (d) –1

11.18 The value returned by fclose() for successful closing of a file is
(a) 0 (b) 1 (c) EOF (d) OK

11.19 The function(s) used for reading a character from a file is (are)
(a) getc() (b) fgetc() (c) both options a and b (d) fgetchar()

11.20 The function(s) used for writing a character to a file is (are)
(a) putc() (b) fputc() (c) both options a and b (d) fputchar()

11.21 The function(s) used for reading formatted input data from a file is (are)
(a) getchar() (b) fscanf() (c) scanf() (d) gets()

11.22 The function used for random access of a file is
(a) fseek() (b) ftell() (c) search() (d) rewind()

11.23 What is the value of origin used in **fseek(fptr, position, origin);?**
to represent end of file
(a) 0 (b) 1 (c) 2 (d) EOF

11.24 What is the value of origin used in **fseek(fptr, position, origin);?**
to represent the beginning of the file
(a) 0 (b) 1 (c) 2 (d) START

11.25 If an error occurs, the function fseek() returns
(a) non-zero (b) zero (c) no value (d) –1

11.26 The value returned by fseek() on successful action is

(a) non-zero (b) zero (c) OK (d) READY

11.27 The function ftell(fptr) returns

(a) the beginning position of the file represented by fptr
(b) the end position of the file represented by fptr
(c) the current position of the file represented by fptr
(d) the middle position of the file represented by fptr

11.28 The value returned by successful action of ftell(fptr) is

(a) –1 (b) 0
(c) long int value representing the current file position (d) MAX_INT

11.29 The value returned by ftell() if an error occurs is

(a) –1 (b) 0 (c) Positive value (d) MIN_INT

11.30 The function call fseek(fp, 0, 0); is same as

(a) `fp = fopen()` (b) `rewind (fp);`
(c) `fclose(fp);` (d) `ftell (fp);`

11.31 The function used for reading binary stream is

(a) read() (b) fread() (c) fscanf() (d) all the above

11.32 The function used for writing to binary streams is

(a) write() (b) fwrite() (c) fprintf() (d) all the above

11.33 Identify the functions for in-memory format conversions

(a) realloc() (b) sprintf() (c) sscanf() (d) both options b and c

11.34 The value of string after executing the following statements is

```
int x = 25; char string[5]; sprintf( string, "%d", x);
```

(a) "25" (b) 25 (c) TWO FIVE (d) 25000

11.35 The value of ch after executing the following statements is

```
int x; char ch; char string[20] = "100 5";
sscanf( string, "%3d%c", &x,&ch );
```

(a) 5 (b) blank (c) "5" (d) NULL

11.36 Identify the correct statement.

(a) `# include "filename"` (b) `#include <filename>`
(c) both options a and b (d) `#include filename`

11.37 The following line in a program

```
#
```

represents

(a) an invalid code (b) a null directive (c) a comment (d) a page number

11.38 Macros

(a) can be recursively defined (b) cannot be recursively defined
(c) cannot be defined (d) are not preprocessor facility

11.39 Identify the stringizing operator.

(a) + (b) :: (c) # (d) ##

11.40 Identify the token pasting operator.
(a) + (b) ++ (c) # (d) ##

11.41 The directive(s) used in conditional compilation is (are)
(a) # if (b) #elif (c) # else (d) all the above

11.42 The default setting of enumerator starts from
(a) 0 (b) 1 (c) –1 (d) any positive integer

11.43 What is the value of CAR in the following statement?

```
enum vehicle
    {BUS, SCOOTER = 2, CAR, TRAIN = 5, AEROPLANE = 6}
```

(a) 0 (b) 1 (c) 3 (d) 4

11.44 Identify the correct statement(s).
(a) the values of an enumerator need not be distinct in the same enumeration type.
(b) the names of an enumerator in different enumeration type must be distinct.
(c) the names of an enumerator must be different from other normal variables.
(d) all the above.

11.45 The qualifier **const**
(a) defines a constant name
(b) keeps the value of a variable constant during execution of the program
(c) both options a and b
(d) does not keep the value of a variable constant.

11.46 Identify the valid constant declaration(s).
(a) `volatile unsigned int clock;` (b) `volatile const unsigned int clock;`
(c) `float const f;` (d) both options b and c

11.47 Functions may use
(a) varying number of parameters. (b) fixed number of parameters.
(c) no argument. (d) at most five arguments.

11.48 va_list is
(a) a macro defined in stdarg.h (b) a predefined data type in stdarg.h
(c) a macro defined in stdlib.h. (d) a predefined data type stdlib.h.

11.49 Identify the macro(s) defined in stdarg.h.
(a) va_start (b) va_end (c) va_arg (d) all the above

11.50 Identify the trigraph sequence for #.
(a) ??/ (b) ??= (c) ??’ (d) ??!

11.51 Identify the trigraph sequence for \.
(a) ??/ (b) ??= (c) ??’ (d) ??!

11.52 Identify the trigraph sequence for ^.
(a) ??/ (b) ??= (c) ??’ (d) ??!

11.53 Which of the following is valid for opening a read-only ASCII file?
(a) `fileOpen (filenm, "read");` (b) `fopen (filenm, "r");`
(c) `fileOpen (filenm, "r");` (d) `fopen (filenm, "read");`

11.54 What are two predefined FILE pointers in C?

(a) stdout and stderr (b) console and error
(c) stdout and stdio (d) stdio and stderr

11.55
```
void listFile( FILE *f )
        {
            int c;
            while( c= fgetc( f )!= EOF )
                   {
                     printf( "%d",c );
                   }
            printf( "\n");
          }
```

What will be printed when the function above is called with pointer to an open file that contains the three characters abc?

(a) The characters ab followed by an infinite number of c's (b) 111
(c) abc (d) 000

11.56
```
FILE *f = fopen( fileName, "r");
readData( f );
if( ???? )
    {
      puts ("End of file was reached");
    }
```

Which of the following can replace the ???? in the above to determine if the end of a file has been reached using ANSI C compiler?

(a) eof(f) (b) feof(f) (c) f == NULL (d) f == EOF

11.57
```
void getFileLength( char *s )
              {
                FILE *f;
                if( f = fopen( s, "r" ) )
                  {
                    fseek(f, SEEK_SET, 2);
                    printf( "%d\n", ????? );
                    fclose( f );
                  }
                }
```

Which of the following could replace the ????? in the code above to cause the function to print the length of the file when a valid file name is passed to it?

(a) ftell(f) (b) fposition(f) (c) tell(f) (d) filelen(f)

11.58 How could stderr be redirected from within a program to force all error messages to be written to the end of the text file "error.log" instead of the location specified by the operating system?

(a) `stderr = fopen( "error.log", "w" );`

(b) `freopen( "error.log", "a", stderr );`

(c) `stderr = freopen( stderr, "a", "error.log" );`
(d) `stderr = fopen( "error.log", "a" );`

11.59 Which of the following would properly define a macro that doubled any expression passed to it?

(a) `# define DOUBLE( x) x* 2`
(b) `# define DOUBLE( x ) ( ( x ) * 2 );`
(c) `# define DOUBLE( x ) ( ( x ) * 2 )`
(d) `# define DOUBLE( x)( x * 2 )`

11.60 What will be the value of j in the following code?

```
#define CATCH( a ) a + 1
int i = 2;
int   j = 4 * CATCH( i * 3);
```

(a) 24 (b) 25 (c) 28 (d) 32

11.61 Which of the following preprocessor directives will make the system-dependent constant INT_MAX available for an ANSI C compiler?

(a) `# include <limits.h>` (b) `# include <int.h>`
(c) `# include <ctype.h>` (d) `# define INT_MAX`

11.62 Which of the following is NOT a valid preprocessor directive?

(a) `#if` (b) `#1ine` (c) `#elseif` (d) `#pragma`

11.63 Which of the following is NOT a pre-processor directive?

(a) `#elif` (b) `#pragma` (c) `#line` (d) `#exclude`

11.64 Which of the following is the correct function prototype for the function main()?

(a) `main(char argc, char *argv)`
(b) `main(int argc, int *argv)`
(c) `main (int argc, int **argv [] )`
(d) `main(int argc,char *argv[])`

11.65 Which is more correct?

`'int main (int argc, char** argv)'`
or `'int main (int argc, char* argv[])'?`

(a) both are equally wrong. (b) both are equally correct.
(c) neither is correct. (d) int main (int argc, char** argv)

11.66 The function used to position the file pointer in C is

(a) seekg() (b) fseekg() (c) fseek() (d) fileseek()

11.67 What file I/O function is used to report the number of bytes from the beginning of the file to the file position indicator?

(a) ftell (b) freport (c) fseek (d) fcount

11.68 1. fcloseall()
2. clearerr()
3. ferror()
Which of the above are valid ANSI C functions?
(a) only option 3 (b) only option 2 (c) only options 2 and 3
(d) options 1, 2, and 3 are all valid ANSI functions.

11.69 Which standard file is to be included to use the memcpy() function without a warning in an ANSI C compiler?

(a) string.h (b) memory.h (c) stdlib.h (d) stdio.h

11.70 If there is a need to see output as soon as possible, which function will force the output from the buffer into the output stream?

(a) write() (b) output() (c) flush() (d) fflush()

11.71 After a library function returns a failure, which of the following will print out the appropriate error message corresponding to error number given by **errno**?

(a) `printf( stderr, "%s\n", geterror() );`
(b) `printerr();`
(c) `perror( errno );`
(d) `strerror( errno );`

11.72 Which of the following function returns calendar time to local time?

(a) gmtime (b) ctime (c) strtime (d) asctime

11.73 What type of call is a system() call?

(a) A user-defined procedure, not a function call
(b) A system inline function call
(c) A standard library call
(d) An application library call

11.74 Which ANSI C standard function could be used to sort a string array?

(a) qsort (b) sort (d) asort (d) bsort

11.75 Which is the fundamental data type used to implement the enum data type?

(a) char (b) int (c) float (d) double

11.76 Which of the following function does not take variable number of arguments?

(a) main() (b) printf() (c) scanf() (d) sprintf()

11.77 Undefined function calls in a C program are detected

(a) by the preprocessor (b) by the assembler
(c) by the linker (d) by the operating system

11.78
```
void test it( struct cust _rec *sptr, char* s)
{...;}
```
Referring to the sample code above, if a calling function has a char array variable **string** of length 10 and a cust _rec structure variable **record**, what would be the correct call to testit?

(a) `testit ( &record, *string );` (b) `testit ( &record, string );`
(c) `testit ( &record, &string );` (d) `testit ( record, *string );`

11.79 What is the output generated by the following code?

```
#define square(a) (a*a)
printf("%d",square(4+5));
```

(a) 81 (b) 4 (c) 29 (d) none of the above

11.80
```
#define max(a,b) (a>b?b:a)
#define squre(x) x*x
int i = 2, j = 3, k = 1;
printf ("%d %d", max(i,j), square(k));
```

11.81. What is the output generated by the above code?

(a) 3 2 (b) 3 1 (c) 2 1 (d) 2 2

11.81 Given the statement **x = fopen (b, c);**
What is b?
(a) pointer to a character array which contains the filename
(b) filename within double quotes
(c) option a or b
(d) none

11.82 What would be the output if the following program (myprog.c) is run using the following command line?

```
myprog jan feb mar apr
main(int size, char *arg[])
  {
    while(size) printf("%s ",arg[--size]);
  }
```

(a) myprog jan feb mar apr (b) apr mar feb jan myprog
(c) jan feb mar apr (d) error

11.83
```
FILE *fpl,*fp2;
fpl=fopen("one","w");
fp2=fopen("one","w");
fputc('A',fpl);
fputc('B',fp2);
fclose(fpl);
fclose(fp2);
fpl=fopen("one","r");
fp2=fopen("one","r");
c = getc(fp2);
putchar(c);
c = getc(fpl);
putchar(c);
```

What is the output of the above code?

(a) error (b) AB (c) BA (d) BB

11.84 What is the output of the following code?

```
#include <stdarg.h>
show(int t,va_list ptrl)
  {
    int a,x,i;
    a=va arg(ptrl,int);
    printf("\n %d",a);
  }
display(char *s,...)
  {
    int x;
```

```
        va_list ptr;
        va _start(ptr,s);
        x=va_arg(ptr,int);
        show(x,ptr);
      }
    main()
       {
        display("hello",4,12,13,14,44)
        }
```

(a) 13 (b) 12 (c) 44 (d) 14

11.85
```
main (int x,char *y[])
  {
    printf ("%d %s", x, y[1]);
  }
```

What is the output when the program is executed as prog argl?

(a) 1 prog (b) l argl (c) 2 prog (d) 2 argl

11.86 Study the following statements.

```
#define DELAYTIME 1000
volatile extern int k;
int j;
for (i=0;i<DELAYTIME;i++);
j=k;
```

State which one of the following is true.

(a) volatile is meaningless for the variable k.
(b) volatile is meaningful for the variable k since k is external and can change.
(c) volatile is meaningless for the variable k since k is loop invariant.
(d) none of the above.

11.87 Which ANSI C function can be used to obtain a date string from structure *tm?

(a) sttime(tm) (b) asctime(tm) (c) gmtime(tm) (d) strtime(tm)

11.88 What is the output of the following program?

```
enum mode {green,red,orange,blue,white};
 main()
    {
      green = green +1;
      printf("%d,%d",green,red );
    }
```

(a) 1,1 (b) 0,1
(c) no output, error in compilation (d) none of the above

11.89 What is the output of the following program?

```
#define MAX(x,y) ((x)>(y) ?(x):(y))
main()
  {
```

```
        int x=5,y=5;
        printf("maximum is %d",MAX(++x,++y));
      }
```

(a) maximum is 7 (b) maximum is 5 (c) maximum is 6 (d) none of the above

11.90 What is the value of CAR in the following enumeration?

```
enum transport
  {
    SCOOTER=1,BUS=4,CAR,TRAIN=8
  };
```

(a) 0 (b) 5 (c) 7 (d) 4

11.91 What is the value of LOTUS in the following enumeration?

```
enum symbol
  {
    HAND,SUN,LOTUS,UMBRELLA
  };
```

(a) 0 (b) 3 (c) 2 (d) none of the above

11.92 What is the output of the following code?

```
enum fruits
{
  APPLE=1,ORANGE=1,GRAPES=3,PINEAPPLE
};
printf("%d",ORANGE);
```

(a) Compilation error (b) Execution error (c) 2 (d) 1

11.93 What is the output of the following code?

```
enum alphanum
{
 VARIABLE=S,DIGIT=10,VARIABLE=11
};
PRINTF("%d",DIGIT);
```

(a) Compilation error (b) Execution error (c) 10 (d) 6

11.94 What is the output of the following code?

```
enum control
{
  on,off,neutral
};
PRINTF("%d",off);
```

(a) Compilation error (b) Execution error (c) 5 (d) 1

11.95 Identify the correct statement.

1. The typedef defines synonyms for an existing data type.
2. The typedef creates a new data type that is not existing in C.
3. The typedef helps in easier modification of a portable program.
4. The typedef gives meaningful names to a data type.

(a) options 1, 3 and 4 (b) options 1 and 4 (c) options 2 and 3 (d) options 2, 3 and 4

11.96 Identify the valid data type of the variable fraction in the following code.

```
typedef float HOST;
HOST fraction;
```

(a) int and HOST (b) struct and HOST (c) enum and HOST (d) float and HOST

11.97 Given the following declaration, identify the correct definition.

```
typedef struct node
  {
    int id;
    char name [20] ;
  };
```

(a) node n; (b) NODE n; (c) typedef NODE n; (d) typedef node n;

11.98 The purpose of the following code is to find

```
#include <time.h>
main( )
  {
    time _t tl,t2;
    clock _t clock(void)
     int i,x=0,y=10;
     tl = clock( );
    for(i=0;i<10000;i++)
      {
        x++; y += 50;
        printf("The value of i= %d, x = %d, y=%d\n",i,x,y);
    }
t2 = clock( );
    printf("Time in seconds :%g\n",difftime(t2,t1)/ CLOCKS_PER_SEC);
  }
```

(a) compilation time (b) execution time (c) difference between GST and IST
(d) difference between compilation and execution times

ANSWERS

Fill in the Blanks

1. **stream**	2. **text stream, binary stream**	3. **"rb"**
4. **NULL pointer**	5. **stdin, stdout and stderr**	6. **getchar()**
7. **ftell(fp)**	8. **##**	9. **integer constants**
10. **va _list**	11. **zero**	12. **volatile**

True or False

1. **False** 2. **True** 3. **True** 4. **False** 5. **True** 6. **False**
7. **False** 8. **True** 9. **True** 10. **False**

Match the Following

1. **5** 2. **4** 3. **1** 4. **3** 5. **2**

Objective Type Questions

1. **b**	11. **d**	21. **b**	31. **b**	41. **d**	51. **a**	61. **a**	71. **d**	81. **c**	91. **c**
2. **c**	12. **b**	22. **a**	32. **b**	42. **a**	52. **c**	62. **c**	72. **b**	82. **b**	92. **d**
3. **b**	13. **b**	23. **c**	33. **d**	43. **c**	53. **b**	63. **d**	73. **c**	83. **d**	93. **a**
4. **a**	14. **c**	24. **a**	34. **a**	44. **d**	54. **a**	64. **d**	74. **a**	84. **b**	94. **a**
5. **c**	15. **b**	25. **a**	35. **b**	45. **b**	55. **b**	65. **b**	75. **b**	85. **d**	95. **a**
6. **c**	16. **a**	26. **b**	36. **c**	46. **d**	56. **b**	66. **c**	76. **a**	86. **b**	96. **b**
7. **c**	17. **c**	27. **c**	37. **b**	47. **a**	57. **a**	67. **a**	77. **c**	87. **b**	97. **b**
8. **b**	18. **a**	28. **c**	38. **b**	48. **b**	58. **d**	68. **c**	78. **b**	88. **c**	98. **b**
9. **b**	19. **c**	29. **a**	39. **c**	49. **d**	59. **c**	69. **a**	79. **c**	89. **a**	
10. **d**	20. **c**	30. **b**	40. **d**	50. **b**	60. **b**	70. **d**	80. **c**	90. **b**	

Additional C Programming Examples

12

PROGRAM 1 — Write a program to print the ASCII equivalent of all characters in the entered string.

SOLUTION

```
#include <stdio.h>
#include <ctype.h>
int ascii_value(char c);
main()
{
 int i,a;
 char c;
// clrscr();
 printf("Please enter a string.");
 printf("\nString will be terminated if you press Ctrl-D (Linux) for EOF.");
 printf("\nSTRING:-  ");
  for (i=0;(c=getchar())!=EOF;i++)//the input character is got until EOF
(Ctrl-D) is met.
 {
  a=ascii_value(c);
  printf("%d%c",a,'');
 }
 printf("\n\t are the ascii values of the characters of the entered string
respectively.");
}
 int ascii_value(char c)
 {
  int a;
  a=(int)c;
  return(a);//The ascii value is returned to the calling function.
 }
```

PROGRAM 2 — Write a program to implement a simple bubble sort.

SOLUTION

```
#include<stdio.h>
int main()
{
            int a[1000];
            int lim;
            int i,j;
            scanf("%d",&lim); // Reading the size of the array
            for(i=0;i<lim;i++)scanf("%d",&a[i]);//reading the input elements
/*For Each Element ,swap the element that is small to the element in cur-
rent position of i */
            for(i=0;i<lim-1;i++)
                    for(j=i+1;j<lim;j++)
                            if(a[i]>a[j])
                            {       a[i]=a[i]+a[j];
                                    a[j]=a[i]-a[j];
                                    a[i]=a[i]-a[j];
                            }

            for(i=0;i<lim;i++)
                    printf("%d\n",a[i]);
}
```

PROGRAM 3 — Write a program to implement the functions of a calculator.

SOLUTION

```
#include<stdio.h>
main()
{
int i,d,k,a,r,h;
char m='y';
long s,n1,n2;
printf("\ncalculation ?");
scanf("%c",&m);
while(m=='Y'||m=='y')
{
/* Create menu giving choice for each operation */
printf("\n1.factorial\n2.addition\n3.subtraction\n4.multiplication\
```

```
n5.division \n6.squares\n7.exit\nenter your choice .......");
scanf("\n%d",&d);
while(d>7||d<1)
scanf("\n%d",&d);
 switch (d)
 {
  case 1:
              printf("enter any positive manageable number");
              scanf("%d",&a);
              while (a<0)
                     scanf("%d",&a);             /* get the input */
              k=1;
              for(i=1;i<=a;i++)
                      k=k*i;
              printf("factorial is %d",k);
             break;
  case 2:

              printf("enter any two numbers");
              scanf(" %ld %ld",&n1,&n2);
              s=n1+n2;                              /* find the summation */
              printf("sum of the input numbers is %ld",s);
           break;
  case 3:
              printf("enter any number");
              scanf("%ld,%ld",&n1,&n2);
               s=n1-n2;                               /* find the difference
between the two inputs */
              printf("the difference between the two numbers is %ld",s);
              break;
  case 4
              printf("enter any numbers which are to be multiplied");
              scanf(" %ld,%ld",&n1,&n2);
              s=n1*n2;                   /*find the multiplication */
              printf("the product is %ld",s)
              break;
  case 5:
              printf("enter dividend");
              scanf("%d",&n1);
              printf("enter divisor ");
              scanf("%d",&n2);
              while (n2==0)
                     {
                     scanf("%d",&n1);
                     scanf("%d",&n2);
```

```
                    }
                s=n1/n2;                          /* find the division of the
two integers */
                printf("the quotient is %d",s);
                break;
  case 6:
                printf("enter the number whose square is to be found out");
                scanf(" %d",&n1);
                s=n1*n1;                            /* square of the number
is found */
                printf("the square of the number is %d",s);
                break;
  case 7:
                return;
  default: printf("\n\nError");
}
printf("\n");
printf("another calculation");
scanf("%c",&m);
}
}
```

PROGRAM 4—Write a program to compute difference between two dates.

SOLUTION

```
#include<stdio.h>
#include<math.h>
void main()
{
int day1,mon1,year1,day2,mon2,year2;
int ref,dd1,dd2,i;
// Reading first date and second date
            printf("Enter first day, month, year");
            scanf("%d%d%d",&day1,&mon1,&year1);
            scanf("%d%d%d",&day2,&mon2,&year2);
            ref = year1;
            if(year2<year1)
                    ref = year2;
            dd1=0;
            dd1=func1(mon1);
for(i=ref;i<year1;i++)
{
            if(i%4==0)
```

```
            dd1+=1;
}
dd1=dd1+day1+(year1-ref)*365;
printf("No. of days of first date from Jan 1 %d= %d",year1,dd1);
/* Count for additional days due to leap years*/
dd2=0;
for(i=ref;i<year2;i++)
{
            if(i%4==0)
            dd2+=1;
}
dd2=func1(mon2)+dd2+day2+((year2-ref)*365);
printf("No. of days from the reference year's first Jan = %d",dd2);
printf("Therefore, diff between the two dates is %d",abs(dd2-dd1));
}
int func1(x)        //x for month y for dd
{  int y=0;
switch(x)
{
case 1: y=0; break;
case 2: y=31; break;
case 3: y=59; break;
case 4: y=90; break;
case 5: y=120;break;
case 6: y=151; break;
case 7: y=181; break;
case 8: y=212; break;
case 9: y=243; break;
case 10:y=273; break;
case 11:y=304; break;
case 12:y=334; break;
default: printf("Error encountered"); exit(1);
}
return(y);
}
```

PROGRAM 5 — Write a program for computing area, volume and perimeter of a rectangle using function with looping.

SOLUTION

```
#include <stdio.h>
void inputoutput ()
                    {
                    int comp,ans;
```

```
                    // Read input choice
                     printf  ("choose  please:  1=perimeter,2=area,3=volume]
?: ");
                    scanf ("%d",&comp);
                    if (comp==1)
                    {
                    int le, wi;
                    printf ("Enter the length: ");
                    scanf ("%d",&le);
                    printf ("Enter the width: ");
                    scanf ("%d",&wi);
// Call perimeter() function to compute perimeter
                    printf ("P=%d",perimeter(le,wi));
                    }
                    else if (comp==2)
                    {
                    int le, wi;
                    printf ("Enter the length: ");
                    scanf ("%d",&le);
                    printf ("Enter the width: ");
                    scanf ("%d",&wi);
// Call perimeter() function to compute area
                    printf ("A=%d", area(le,wi));
                    }
                    else if (comp==3)
                    {
                    int length,width,height;
                    printf ("Enter length: ");
                    scanf ("%d",&length);
                    printf ("Enter width: ");
                    scanf ("%d",&width);
                    printf ("Enter height: ");
                    scanf ("%d",&height);
// Call perimeter() function to compute volume
                    printf ("V=%d",volume (length,width,height));
                    }

                    else inputoutput ();
printf ("\nDo you want to continue? [Yes=1/No=0]: ");
scanf ("%d",&ans);
if (ans==1)
inputoutput ();
else printf ("");
                    }
int perimeter (int l, int w)
```

```
{
                    int per;
// Perimeter = (2l+2b)
                    per=(l*2)+(w*2);
                    return (per);

}
int area (int le, int wi)
{
int area;
// area = lb
area=le*wi;
return (area);
}
int volume (int length, int width, int height)
{
// volume = lbh
int vol;

vol=(length*width*height);
return (vol);
}
main ()
{
input output ();
}
```

PROGRAM 6 — Write a program to convert upper case to lower case or lower case to upper case, depending on the name it is invoked with (as found in argument).

SOLUTION

```
#include <stdio.h>
void lower_to_upper();
void upper_to_lower();
main()
{
 int n;
// Read input choice and input string
 printf("\nPlease enter your choice.");
 printf("\n(1) for upper to lower conversion.");
 printf("\n(2) for lower to upper conversion.");
 printf("\nCHOICE:-  ");
 scanf("%d",&n);
 switch (n)
```

```
 {
  case 1:
                    {
                   printf("Please enter a string in upper case.");
                   printf("\nString will be terminated if you press Ctrl-Z.");
                   printf("\nSTRING:-  ");
// call upper to lower() function
                   upper_to_lower();
                   break;
                    }

  case 2:
                    {
                   printf("Please enter a string in lower case.");
                   printf("\nString will be terminated if you press Ctrl-Z.");
                   printf("\nSTRING:-  ");
// call lower to upper function
                   lower_to_upper();
                   break;
                    }
  default:
                   printf("ERROR");
 }
 printf("");

}
 void upper_to_lower()
 {
  int i,j;
  char c4[80],c3;
/*
           if input is between 'A' and 'Z' , convert it to 'a' between 'z'
*/
  for (i=0;(c3=getchar())!=EOF;i++)
           c4[i]=(c3>='A' && c3<='Z')?('a' + c3 -'A'):c3;
  printf("\nThe lower case equivalent is ");
  for (j=0;j<i;j++)
           putchar(c4[j]);
  return;
 }

 void lower_to_upper()
 {
  int i,j;
  char c2[80],c1;
/*
```

```
            If input is between 'a' and 'z',convert it into between 'A' and
'Z'
*/
  for (i=0;(c1=getchar())!=EOF;i++)
            c2[i]=(c1>='a' && c1<='z')?('A' + c1 -'a'):c1;
  printf("\nThe upper case equivalent is ");
  for (j=0;j<i;j++)
            putchar(c2[j]);
  return;
 }
```

PROGRAM 7 — Write a program to convert roman letter to number. Each character in the input is to be parsed one by one and the corresponding value for each letter in the corresponding positions is assigned and finally added.

SOLUTION

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<malloc.h>
int main()
{

int *a,l,i,j,k;
char s[100];
printf("Enter The Roman Number\n");
scanf("%s",s);
l=strlen(s);
for(i=0;i<l;i++)
{
if(s[i]=='I')
a[i]=1;
else if(s[i]=='V')
a[i]=5;
else if(s[i]=='X')
a[i]=10;
else if(s[i]=='L')
a[i]=50;
else if(s[i]=='C')
a[i]=100;
else if(s[i]=='D')
a[i]=500;
else if(s[i]=='M')
a[i]=1000;
```

```
else
{
printf("Wrong Input");

exit(0);
}
}
k=a[l-1];
for(i=l-1;i>0;i--)
{
if(a[i]>a[i-1])
k=k-a[i-1];
else if(a[i]==a[i-1] || a[i]<a[i-1])
k=k+a[i-1];
}
printf("\n%d",k);
}
```

PROGRAM 8 — Write a program to copy the input to the output replacing string of one or more blanks by a single blank.

SOLUTION

```
#include <stdio.h>
void main()
{
 char c;
 printf("\nEnter a line of text:-  ");
 printf("\nString will be terminated if you press ENTER.");
 printf("\nPress Ctrl-Z & then ENTER to end the task.");
 printf("\nSTRING:-\n  ");
 c=getchar();
 printf("\n\nJustified form:-  ");
 while (c!=EOF)
  {
            if (c==' ')
            putchar(c);
            while (c==' ')
            {
             c=getchar();
            }
            putchar(c);
            c=getchar();
  }
}
```

PROGRAM 9 — Write a program to correct rudimentary syntax errors.

SOLUTION

```
#include <stdio.h>
#define NULL 0
FILE *fpt;
main()
{
 int c1=0,c2=0,c3=0,c4=0,c5=0;
 char c,name[20],z;

 printf("Enter the name of file to be checked :-  ");
 scanf("%s",name);
 fpt=fopen(name,"r");
 if (fpt==NULL)
  printf("\nERROR - can/'t open file %s",name);
 else
 {
  while ((c=getc(fpt))!=EOF)
  {
   if (c=='(')
    c1=c1+1;
   if (c==')')
    c1=c1-1;
   if (c=='[')
            c2=c2+1;
   if (c==']')
            c2=c2-1;
   if (c=='\n')
   {
            if (c1!=0)
             printf("\nERROR - Unbalanced parenthesis ()");
            if (c2!=0)
             printf("\nERROR - Unbalanced brackets []");
   }
   if (c=='{')
            c3=c3+1;
   if (c=='}')
            c3=c3-1;
   if ((int)c==39)
   {
            if (c1!=0)
            {
             if (c4==0)
```

```
              c4=c4+1;
             else
              c4=c4-1;
            }
            else
             printf("\nERROR - Unbalanced ' ");
   }
   if ((int)c==34)
   {
            if (c1!=0)
            {
             if (c5==0)
              c5=c5+1;
             else
              c5=c5-1;
            }
            else
            {
             z=(char)34;
             printf("\nERROR - Unbalanced %c",z);
            }
   }
  }
 }

   if (c1!=0)
            printf("\nERROR - Unbalanced parenthesis ()");
   if (c2!=0)
            printf("\nERROR - Unbalanced brackets []");
   if (c3!=0)
            printf("\nERROR - Unbalanced braces {}");
   if (c4!=0)
            printf("\nERROR - Unbalanced '  ");
   if (c5!=0)
    printf("\nERROR - Unbalanced \" ");

   if (c1==0 && c2==0 && c3==0 && c4==0 && c5==0)
    printf("\nProgram is up to date. WELL DONE!");
 fclose(fpt);

}
```

PROGRAM 10 —Write a program to count no. of characters, no. of blanks, no. of words and no. of lines in a multi-line string.

SOLUTION

```
#include <stdio.h>
#include<string.h>
main()
{
            char str[1000];
            char ch;
            int i=0;
            int lines=1,words=1,charac=0;
            while((ch=getchar())!=EOF)
                    str[i++] = ch;                  /* read input as single
character */
            str[i]='\0';
            for(i=0;i<strlen(str);i++)
            {
                    switch(str[i])
                    {
                            case  '\n':      /* using switch case, when '\n' is encountered */
                                    lines++;  /* increment the 'lines' count.similarly,
when '' */
                            break;           /* is encountered, increment the 'words' count */
                            case ' ':    /* For everything else, increment the charac count */
                                    words++;
                            break;
                            default:
                                    charac++;


                    }
            }
            printf("Lines :%d Words:%d Characters:%d \n",lines,words,charac);
}
```

PROGRAM 11—Write a program to find determinant of a matrix.

SOLUTION

```
#include<stdio.h>
#define LIMIT 10
main()
{
```

```
  int chckdgnl();
  float deter();
  float a[LIMIT][LIMIT],value;
  int i,j,order;
// Read the order of input matrix
  printf("Enter order of determinant :");
  scanf("%d",&order);
// Read the input elements of the square matrix
  for(i=0;i<order;i++)
  {
            for(j=0;j<order;j++)
            {
              printf("Enter (%d,%d) element of the determinant :",i+1,j+1);
              scanf("%f",&a[i][j]);
            }
  }
// Call determinant function
  if(chckdgnl(a,order)==0)
             value=0;
  else
             value=deter(a,order);
  printf("Determinant Value :%f",value);

}

float deter(float a[][LIMIT],int forder)
{
  int i,j,k;
  float mult;
  float deter=1;
  for(i=0;i<forder;i++)
  {
            for(j=0;j<forder;j++)
            {
              mult=a[j][i]/a[i][i];
              for(k=0;k<forder;k++)
              {
                    if(i==j) break;
                    a[j][k]=a[j][k]-a[i][k]*mult;
              }
            }
  }
  for(i=0;i<forder;i++)
  {
            deter=deter*a[i][i];
```

```
  }
  return(deter);
}


int chckdgnl(float array[][LIMIT],int forder)
{
  int i,j,k;
  float nonzero;
/*
            If a row or column is zero, then determinant value is zero
*/
  for(i=0;i<forder;i++)
  {
             if(array[i][i]==0)
             {
                    for(j=0;j<forder;j++)
                    {
                      if(array[i][j]!=0)
                      {
                             k=j;
                             break;
                      }
                      if(j==(forder)) //forder-1
                             return(0);
                    }
                    for(j=0;j<forder;j++)
                    {
                      array[j][i]=array[j][i]-array[j][k];
                    }
             }
  }
  return(1);
}
```

PROGRAM 12—Write a program to display all vowels in an i/p line of text by using OR operation in an 'if' loop.

SOLUTION

```
#include <stdio.h>
main()
{
 char line[80],c;
 int i;
```

```
 printf("\nEnter string:-");
 printf("\nString will be terminated if you press ENTER");
 printf("\nSTRING:-");
 scanf("%[^\n]",line);
 for (i=0;(i<=80 && line[i]!='\0');i++)
 {
  if (line[i]=='a'||line[i]=='A'||line[i]=='e'||line[i]=='E'||line[i]=='i'
                ||line[i]=='I'||line[i]=='o'||line[i]=='O'||line[i]=='u'||
line[i]=='U')
  {
            printf("%c",line[i]);
            printf("  ");
  }
 }

}
```

PROGRAM 13—Write a program to encrypt and decrypt a given string.

SOLUTION

```
#include <stdio.h>
void main()
{
 int n,i;
 char a[80];

 printf("\nEnter string:-  ");
 gets(a);
 printf("\n\n\tOPTIONS:- ");
 printf("\n(1) Encrypt the string.");
 printf("\n(2) Decrypt the string.");
 printf("\nCHOICE(1 or 2):-  ");
 scanf("%d",&n);

 switch(n)
 {
  case 1:
             for (i=0;(i<80&&a[i]!='\0');i++)
              a[i]=a[i]+2;/* to encrypt the string, add 2 to ASCII value of
each character of the string*/
             printf("\nIts encrypted form is:-  ");
             printf("\n\t%s",a);
             break;
```

```
  case 2:
             for (i=0;(i<80&&a[i]!='\0');i++)
              a[i]=a[i]-2;  /* subtract 2 from ASCII value of each character
of the string */
             printf("\nIts decrypted form is:-  ");
             printf("\n\t%s",a);
             break;

  default:
              printf("\nERROR.");
 }

}
```

PROGRAM 14—Write a program to find the existence and frequency of an i/p sub-string in an i/p main string.

SOLUTION

```
#include <stdio.h>

void insert(char str1[],char str2[]);
 main()
{
 char a[80],b[80];
//Read the Strings main string,Sub-String
 printf("\nEnter main string(i.e. str1):-\n");
 scanf(a);
 printf("\nEnter sub-string(i.e. str2):-\n");
 scanf(b);
// call insert  function  to print the sub-string
 insert(a,b);
 printf("");

}

 void insert(char str1[],char str2[])
 {
  char c[80],d[80];
  int i=0,j=0,k=0,count=0,l=0,k1=0;
  for (i=0;(i<80 && str1[i]!='\0');i++)
  {
            c[i]=str1[i];
            d[i]=str2[i];
```

```
  }

 l=strlen(d);
 i=0;
 while (c[i]!=EOF)
 {
  if (c[i]==d[j])
  {
            i++;
            j++;
            k1=1;
            if (j==l)
            {
             j=0;
             k=1;
             count=count+1;
            }
  }
  else
  {
            if (k1==1)
            {
             j=0;
             k1=0;
            }
            else
             i++;
  }
 }
// presence of substring depends on value of k .
// if k equals 1, the given substring is present
  if (k==1)
  {
            printf("\n\nThe  given  sub-string  is  present  in  the  main
string.");
            printf("\nIt is present %d times.",count);
  }
  else
  {
            if (k==0)
             printf("\n\nThe given sub-string is not present in the main
string.");
  }
 return;
}
```

PROGRAM 15—Write a program to implement Fibonacci series.

SOLUTION

```
#include <stdio.h>

main()
{
   int OldNum, NewNum, FibNum, MaxNum;

   printf("\nGenerate Fibonacci Numbers till what number? ");
   scanf("%d", &MaxNum);

   OldNum=0;
   NewNum=1;
   FibNum = OldNum + NewNum;

   printf("%d, %d, %d, ", OldNum, NewNum, FibNum);
// Till the FibNum is less than MaxNum, the previous value of FibNum is
added with the OldNum
   for(;;)
   {
      OldNum = NewNum;
      NewNum = FibNum;
      FibNum = OldNum + NewNum;
      if(FibNum > MaxNum)
      {
         printf("\n");
         exit(1);
      }
      printf("%d, ", FibNum);
   }
}
```

PROGRAM 16—Write a program to find prime numbers.

SOLUTION

```
#include <stdio.h>

main()
{
  int n,m,k,i,max;
  char c;
```

```
  max=0;
  k=2;
  n=1;
  printf("Assuming that 1 is prime\n");
  printf("You want prime numbers upto:-  ");
  scanf("%d",&max);
  printf("\n\n");
 if(max>=2) printf("2 ");
  for (i=1;i<=max;i++)
            {
             again: m=(n/k)*k;
             if (m!=n)
            k=k+1;
             else
            goto try1;
             if (k < n/2)
            goto again;
             else
            printf("%d",n);
            printf("  ");
try1: n=n+1;
            k=2;
             }
}
```

PROGRAM 17—Write a program to find the type of triangle by taking i/p values as coordinates of its vertices.

SOLUTION

```
#include <stdio.h>

#include <math.h>
#include<stdlib.h>
void main()
{
 float x1,y1,x2,y2,x3,y3;
 float a,b,c,m1,m2,m3;
 printf("Enter coordinates of vertex A(x & y respectively):-  ");
 scanf("%f%f",&x1,&y1);
 printf("\nEnter coordinates of vertex B(x & y respectively):-  ");
 scanf("%f%f",&x2,&y2);
 printf("\nEnter coordinates of vertex C(x & y respectively):-  ");
 scanf("%f%f",&x3,&y3);
```

```
 if ((x1==x2 && x2==x3)||(y1==y2 && y2==y3)) // If two co ordinates are
similar, it acnt form a triangle.
 {
  printf("\n\nThese coordinates can't represent a triangle.");
  printf("\nA,B & C are collinear & thus consitute a line.");
  printf("\n\n\n\n\n\t\t\tHAVE A NICE DAY! BYE.");
  exit(0);
 }
 else
 {//Find slope.
  m1=(y2-y1)/(x2-x1);
  m2=(y3-y2)/(x3-x2);
  m3=(y3-y1)/(x3-x1);
 }
 if (m1==m2||m2==m3||m3==m1)
 {
  printf("\n\nThese coordinates can't represent a triangle.");
  printf("\nA,B & C are colinear & thus consitute a line.");
  printf("\n\n\n\n\n\t\t\tHAVE A NICE DAY! BYE.");
  exit(0);

 }
// Calculate the length of each side of the triangle
 a = sqrt(pow((x2-x3),2) + pow((y2-y3),2));
 b = sqrt(pow((x3-x1),2) + pow((y3-y1),2));
 c = sqrt(pow((x2-x1),2) + pow((y2-y1),2));
 printf("\n\nLength of side AB is = %f",c);
 printf("\nLength of side BC is = %f",a);
 printf("\nLength of side CA is = %f",b);
 if (a==b==c)//If all lengths are equal, equilateral triangle
  printf("\n\nTriangle made by these vertices is an equilateral triangle.");
 else if (a==b||b==c||c==a)// If any 2 sides are equal.
 {
           if (a==b==c);
  else
           printf("\n\nTriangle made by these vertices is an isosceles
triangle.");
 }
 else if (a!=b && b!=c && c!=a)
           printf("\n\nTriangle made by these vertices is a scalene tri-
angle.");
 printf("\n\n\n\n\n\t\t\tHAVE A NICE DAY! BYE.");
}
```

PROGRAM 18— Write a program to find whether a given year is a leap year or not.

SOLUTION

```
#include <stdio.h>

void main()
{
 int year,leap;

 printf("enter the year:-  ");
 scanf("%d",&year);
 if ((year%100)==0)
  leap=(year/400)*400;
 else
  leap=(year/4)*4;
 if (leap==year)
  printf("\n%d is a leap year.",year);
 else
  printf("\n%d is not a leap year.",year);
// getch();
}
```

PROGRAM 19—Write a program to create a linked list.

SOLUTION

```
#include<stdio.h>
#include<malloc.h>
struct ll
{
            int num;
            struct ll *next;
};
typedef struct ll node;
void create(node **p,int n)
{
            if((*p)==NULL)
            {
                    (*p)=(node*)malloc(sizeof(node));
                    (*p)->next = NULL;
                    (*p)->num = n;
            }
```

```
            else
                    create(&(*p)->next,n);
}
void display(node *p)
{
            if(p!=NULL)
            {
                    printf("%d %p\n",p->num,p);
                    display(p->next);
            }
}
int main()
{
            int n;
            node *head = NULL;
            while(1)
            {
                    printf("Enter -1 to stop else input a positive number:");
                    scanf("%d",&n);
                    if(n<0) break;
                    create(&head,n);
            }
            display(head);
}
```

PROGRAM 20—Write a program to implement the maximum of four numbers.

SOLUTION

```
#include<stdio.h>
int max( int x , int y )
{
if(y!=0) // to check for y not being zero
{
if(x/y)
return x;
else return y;
}
else
return x;
}
int main()
{
int a,b,c,d;
//give some values to a,b,c,d
```

```
printf("Enter 4 numbers");
scanf("%d %d %d %d",&a,&b,&c,&d);
printf("%d",max(max(a,b),max(c,d)));
return 0;
}
```

PROGRAM 21—Write a program to multiply two matrices.

SOLUTION

```
#define MAXROWS 30
#define MAXCOLS 30
void readinput(int a[][MAXCOLS],int m,int n);
void computeproduct(int a[][MAXCOLS],int b[][MAXCOLS],int c[][MAXCOLS],int
m,int n,int p);
void writeoutput(int c[][MAXCOLS],int m,int p);
int z=0;
void main()
{

 /* char c;*/
 int nrows,ncols,mrows,mcols;
 int a[MAXROWS][MAXCOLS],b[MAXROWS][MAXCOLS],c[MAXROWS][MAXCOLS];
 // Read Input parameters for both the matrices
 printf("\n\nHow many rows in the first matrix?  ");
 scanf("%d",&nrows);
 printf("\n\nHow many cols in the first matrix?  ");
 scanf("%d",&ncols);
 printf("\n\nHow many rows in the second matrix?  ");
 scanf("%d",&mrows);
 printf("\n\nHow many cols in the second matrix?  ");
 scanf("%d",&mcols);
// the rows of second matrix and columns of first matrix should be same
 if (ncols != mrows)
 {
  printf("The product of these matrices is not defined.");

  exit(0);
 }
/*
           Reading input for matrix a
*/
 printf("\n\nFirst table:\n");
 readinput(a,nrows,ncols);
/*
```

```
            Reading input for matrix b
*/
 printf("\n\nSecond table:\n");
 readinput(b,mrows,mcols);
// Computing product and storing matrix C
 computeproduct(a,b,c,nrows,ncols,mcols);
 printf("\n\nProduct of the matrices is:\n\n");
// matrix C is displayed nrows , mcols
 writeoutput(c,nrows,mcols);
}

/*
            this function reads the input matrix a
*/
void readinput(int a[][MAXCOLS],int m,int n)
{
 int row,col;
 for (row=0;row<m;row++)
 {
  printf("\nEnter data for row no. %4d\n",row+1);
  for (col=0;col<n;col++)
   scanf("%d",&a[row][col]);
 }
 z=z+1;
 printf("\tTable %d\n",z);
 for (row=0;row<m;row++)
 {
  for (col=0;col<n;col++)
   printf("%d%c",a[row][col],' ');
  printf("\n");
 }
 return;
}
/*
            Matrix multiplication function for resultant matrix
*/

void computeproduct(int a[][MAXCOLS],int b[][MAXCOLS],int c[][MAXCOLS],int
m,int n,int p)
{
 int i,j,k,sum=0;
 for (i=0;i<m;i++)
 {
  for (j=0;j<p;j++)
  {
   for (k=0;k<n;k++)
```

```
            sum=sum + (a[i][k]*b[k][j]);
    c[i][j]=sum;
    sum=0;
   }
  }
  return;
}


void writeoutput(int c[][MAXCOLS],int m,int p)
{
 int row,col;
 for (row=0;row<m;row++)
 {
  for (col=0;col<p;col++)
   printf("%6d",c[row][col]);
  printf("\n");
 }
 return;
}
```

PROGRAM 22—Write a program to find pair of numbers in an array that have their difference as the given value.

SOLUTION

```
#include<stdio.h>
void lin_search(int x[], int y[], int n)
   {
     int i,j,count=0;
           printf("The pair of numbers that have their difference as the
given value are:");
       for(i=0;i<n;i++)
        for(j=0;j<n;j++)
{
        if(y[i]==x[j])         /* search each element of 2nd array with that
of every element in 1st array; if any match found print the corresponding
element in the 1st array and the element with index value given by the 2nd
array element for ex:if b[0] matches with a[2] print a[0] and a[2] */
         {
          printf("\n%d %d and the positions are %d and %d \n",x[i],x[j],i,j);
          count++; break;
         }
}
if(count==0)
```

```
            printf("%d pairs found",count);
   }
int main()
{
int a[25],b[25],n,m,i;
printf("Enter the number of integers to be in the array");
scanf("%d",&m);            /* get the number of integers in the array */
printf("Enter the numbers to add them in the array");
for(i=0;i<m;i++)
{
scanf("%d",&a[i]);        /*  get the array input */
}
printf("Enter the number");
scanf("%d",&n);
for(i=0;i<m;i++)
{
b[i]=a[i]+n;              /* create another array with inputs as addition
of original array input and given value    ex: a[0]=4; given value n=4;
b[0]=4+4=8 */
}
lin_search(a,b,m);
return 0;
}
```

PROGRAM 23—Write a program to search for a palindrome.

SOLUTION

```
#include <stdio.h>

#define EOL '\n'
#define TRUE 1
#define FALSE 0
void main()
{
 char letter[80];
 int tag,count,countback,flag,loop=TRUE;

 while (loop)
 {
  flag=TRUE;
  printf("\n\nPlease enter a word, phrase or sentence below(type END to exit
the program):\n");
  for (count=0;(letter[count]=getchar())!=EOL;++count)//assign the word to
letter array.
```

```
   ;
      if   ((toupper(letter[0])=='E')   &&   (toupper(letter[1])=='N')   &&
(toupper(letter[2])=='D')) //check if END is pressed.
   break;
  tag=count-1;
     for   ((count=0,countback=tag);count<=(tag/2);(++count,--countback))//
count gives the index from left to right, whereas countback gives the index
from right to left.
  {
    if (letter[count]!=letter[countback])//If both are not equal then flag
is set off.
   {
            flag=FALSE;
            break;
   }
  }
  for (count=0;count<=tag;++count)
   putchar(letter[count]);
  if (flag)
   printf("  is a palindrome.");
  else
   printf("  is not a palindrome.");
 }
}
```

PROGRAM 24—Write a program to find the pascal triangle.

SOLUTION

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
            int a[1000][1000]={0};
            int lines;
            int i,j;
            a[0][0]=1;
            a[1][0]=a[1][1]=1;
            printf("Enter total number of lines:");
            scanf("%d",&lines);// get input
            for(i=2;i<lines;i++)
            {
                    for(j=0;a[i-1][j]!=0;j++)
                    {
                            if(j==0) a[i][j]=1;
```

```
                        else
                        a[i][j] = a[i-1][j]+a[i-1][j-1]; // the previous
line's left and right side elements are being added
                }
                a[i][j]=1;
        }
        for(i=0;i<lines;i++)
        {
                for(j=0;a[i][j]!=0;j++)
                        printf("%d  ",a[i][j]);
                printf("\n");
        }
}
```

PROGRAM 25—Write a program to pick and display the largest element of an input matrix.

SOLUTION

```
#include <stdio.h>
#include <math.h>
#define MAXROWS 30
#define MAXCOLS 30
void largest(int a[][MAXCOLS],int nrows,int ncols);
void readinput(int a[][MAXCOLS],int m,int n);
void main()
{
 int nrows,ncols;
 int a[MAXROWS][MAXCOLS];
 printf("How many rows in the matrix?  ");
 scanf("%d",&nrows);
 printf("How many columns in the matrix?  ");
 scanf("%d",&ncols);
 printf("\n\nTable\n");
 readinput(a,nrows,ncols);
 largest(a,nrows,ncols);
}
 void readinput(int a[][MAXCOLS],int m,int n)
 {
  int row,col;
  for (row=0;row<m;++row)
  {
   printf("\nEnter data for row no. %2d\n",row+1);
   for (col=0;col<n;++col)
    scanf("%d",&a[row][col]);
  }
```

```
 printf("\tTABLE 1");
 for (row=0;row<m;++row)
 {
  printf("\n\t\t");
  for (col=0;col<n;++col)
   printf("%d%c",a[row][col],' ');
 }
   return;
}


void largest(int a[][MAXCOLS],int m,int n)
{
 int i,j,largest;
 largest = a[0][0];
 for (i=0;i<m;++i)
 {
  for (j=0;j<n;++j)
  {
   if (a[i][j]>largest)
    largest=a[i][j];
  }
 }
 printf("\nThe largest element of the matrix is %d",largest);
 return;
}
```

PROGRAM 26—Write a program to implement queue through array.

SOLUTION

```
#include <stdio.h>
#include<ctype.h>
# define MAXSIZE 200

int q[MAXSIZE];
int front, rear;
void main()
{
void add(int);
int del();
int will=1,i,num;
front =0;
rear = 0;
printf("\n\t\tProgram for queue demonstration through array\n\n\n");
```

```
while(will ==1)
{
printf("\n\t\tMAIN MENU: \n\t1.Add element to queue\n\t2.Delete element
from the queue\n");
scanf("%d",&will);

switch(will)
{
case 1:
            printf("\nEnter the data... ");
            scanf("%d",&num);
            add(num);
            break;
case 2: i=del();
            printf("\nValue returned from delete function is  %d ",i);
            break;
default: printf("Invalid Choice ... ");
}

printf(" \n\n\t\t\tDo you want to do more operations on Queue ( 1 for yes,
any other key to exit) ");
scanf("%d" , &will);
} //end of  outer while
}               //end of main

//Add elements in the queue.
void add(int a)
{
//rear is incremented whenever an element is added.
if(rear>MAXSIZE)
            {
            printf("\n\n\t\tQUEUE FULL\n\n");
            return;
            }
else
            {
            q[rear]=a;
            rear++;
            printf("\n Value of rear = %d and the value of front is
%d",rear,front);
            }
}
// Delete elements in the queue.
int del()
{
int a;
```

```
if(front == rear)
            {
            printf("\n\n\t\tQUEUE EMPTY\n\n");
            return(0);
            }
else
            {
            a=q[front];
            front++;//front is incremented whenever an element is deleted.
            }
            return(a);
}
```

PROGRAM 27—Write a program to implement towers of Hanoi.

SOLUTION

```
#include<stdio.h>
void hanoi(int n,char A,char B,char C)
{
            if(n==1)
                    printf("Move disc from %c to %c\n",A,C);
            else
            {
                    hanoi(n-1,A,C,B);
                    hanoi(1,A,B,C);
                    hanoi(n-1,B,A,C);
            }
}
main()
{
            int n;
            printf("Enter no of discs:");
            scanf("%d",&n);
            hanoi(n,'A','B','C');
}
```

PROGRAM 28—Write a program to remove redundant characters.

SOLUTION

```
#include<string.h>
#include <stdio.h>
void func(char a[])
```

```
{
            int i;char flag[256]={0};
            char b[1000]={0};
            int j=0;
/*
            set the bit of read character in flag array
            if the character is set in flag array already,it's not added to
resulting string
*/
            for(i=0;i<strlen(a);i++)
            {       if(flag[a[i]]==0) b[j++] = a[i];
                    flag[a[i]]=1;
            }
            strcpy(a,b);
}
int main()
{
            char str[1000];
            // reading input string str
            scanf("%s",str);
            // call function to remove redundant characters
            func(str);
            printf("%s\n",str);
}
```

PROGRAM 29—Write a program that will reverse a character string and then, write a program to reverse its input a line at a time.

SOLUTION

```
#include<stdio.h>
#define MAXLINE 1000

int getline(char line[], int maxline);
void reverse(char s[]);

main()
{
            char line[MAXLINE];

            while(scanf("%s",line)!=EOF)
                    {
                            reverse(line);
                            printf("%s",line);
                    }
```

```
}

void reverse(char s[])
{
            int i,j;
            char temp;
            i=0;
            while(s[i]!='\0')
                    ++i;
                    --i;
            if(s[i]=='\n')
                    --i;
            j=0;
            while(j<i)
                    {
                            temp=s[j];
                            s[j]=s[i];
                            s[i]=temp;
                            --i;
                            ++j;
                    }
            }
```

PROGRAM 30—Write a program to reverse the bits in an integer.

SOLUTION

```
#include<stdio.h>
int main()
{
unsigned int num =1;          // Reverse the bits in this number.
unsigned int temp = num;      // temp will have the reversed bits of num.

int i;
for (i = (sizeof(num)*8-1); i; i--)
{
 temp = temp | (num & 1); // num&1 will give the last bit of 'num' and it
is ORed with temp's value.
 temp <<= 1;  // left shift temp once.
 num  >>= 1;  // right shift num once.
}

temp = temp | (num & 1);
printf("Reversed bit value %d ",temp);
}
```

PROGRAM 31—Write a program to reverse a linked list.

SOLUTION

```
#include<stdio.h>
#include<malloc.h>
struct ll
{
            int num;
            struct ll *next;
};
typedef struct ll node;
node *reverseHead;
void create(node **p,int n)
{
            if((*p)==NULL)
            {
                    (*p)=(node*)malloc(sizeof(node));
                    (*p)->next = NULL;
                    (*p)->num = n;
            }
            else
                    create(&(*p)->next,n);
}
void display(node *p)
{
            if(p!=NULL)
            {
                    printf("%d %p\n",p->num,p);
                    display(p->next);
            }
}
void reverse(node *p)
{
            int a[1000];
            int count=0;
            while(p!=NULL)
                    a[count++]=p->num,p=p->next;
            count--;
            for(;count>=0;count--)
                    create(&reverseHead,a[count]);
}
int main()
{
```

```
            int n;
            node *head = NULL;
            while(1)
            {
                    printf("Enter -1 to stop else input a positive number:");
                    scanf("%d",&n);
                    if(n<0) break;
                    create(&head,n);
            }
            display(head);
            reverse(head);
            printf("-------------------------------------\n");
            display(reverseHead);
}
```

PROGRAM 32—Write a program to sort any no. of numeral i/p in ascending or descending order.

SOLUTION

```
#include<stdio.h>
void sort(void);
int c,a[20],l;
void main()
{

 printf("Enter no. of elements in the list:-  ");
 scanf ("%d",&l);
 printf("\nCHOICE:-");
 printf("\n(1) Sort in ascending order.");
 printf("\n(2) Sort in descending order.");
 printf("\nCHOICE:- ");
 scanf("%d",&c);
 if (c!=1 && c!=2)
 {
  printf("\nERROR");

  exit(0);
 }
 sort();

}


void sort(void)
```

```
{
 int n,i,j,temp=0,min,k;
 for (i=1;i<=l;i++)
 {
  printf("\nEnter no.:-  ");
  scanf("%d",&a[i]);
 }
 for (i=1;i<=(l-1);i++)
 {
  min=a[i];
  k=i;
  for (j=(i+1);j<=l;j++)
  {
   if (a[j]<min)
   {
            min=a[j];
            k=j;
   }
  }
  temp=a[k];
  a[k]=a[i];
  a[i]=temp;
 }

 switch(c)
 {
  case 1:
                    printf("\nElements in ascending order are:-");
                    for (i=1;i<=l;i++)
                     printf("\n%d",a[i]);
                    break;
  case 2:
                    printf("\nElements in descending order are:-");
                    for (i=l;i>=1;i--)
                     printf("\n%d",a[i]);
                    break;
  default:
                     printf("\nERROR");
 }
 return;
}
```

PROGRAM 33—Write a program to implement stack through array.

SOLUTION

```
#include <stdio.h>
#include<ctype.h>
# define MAXSIZE 200

int stack[MAXSIZE];
int top;    //index pointing to the top of stack
void main()
{
void push(int);
int pop();
int will=1,i,num;


while(will ==1)
{
printf("\n\t\tMAIN MENU: \n\t1.Add element to stack\n\t2.Delete element
from the stack\n");
scanf("%d",&will);

switch(will)
{
case 1:
            printf("\nEnter the data... ");
            scanf("%d",&num);
            push(num);
            break;
case 2: i=pop();
            printf("\nValue returned from pop function is %d",i);
            break;
default: printf("Invalid Choice. ");
}

printf(" \n\n\t\t\tDo you want to do more operations on Stack ( 1 for yes,
any other key to exit) ");
scanf("%d" , &will);
} //end of  outer while
}               //end of main

// Add elements in the stack.
void push(int y)
{
```

```
if(top>MAXSIZE)
       {
       printf("\n\n\t\tSTACK FULL\n\n");
       return;
       }
else
            {
            top++;//top is incremented whenever an element is added.
            stack[top]=y;
            }
}
// Delete elements in the stack in the Last In First Out(LIFO) fashion.
int pop()
{
int a;
if(top<=0)
            {
            printf("\n\n\t\tSTACK EMPTY\n\n\t\t");
            return 0;
            }
else
            {
            a=stack[top];
            top--;//top is decremented whenever an element is deleted.
            }
return(a);

}
```

PROGRAM 34—Write a program to find the sum of series.

SOLUTION

```
#include <stdio.h>
long int factorial(int n);
void main()
{
 int n,i;
 float s,r;
 char c;
// Read n
 printf("\n\nYou have this series:- 1/1! + 2/2! + 3/3! + 4/4! ...");
 printf("\nTo which term you want its sum?  ");
 scanf("%d",&n);
```

```
 s=0;
// compute factorial of i .. and divide i by factorial(i)
 for (i=1;i<=n;i++)
  {   s=s+((float)i/(float)factorial(i)); }
 printf("\nThe sum of %d terms is %f",n,s);
 fflush(stdin);
}
// recursive function computing factorial of n
long int factorial(int n)
 {
  if (n<=1)
            return(1);
  else
            n=n*factorial(n-1);
            return(n);
 }
```

PROGRAM 35—Write a program to perform temperature conversion.

SOLUTION

```
#include<stdio.h>
float celsius(float fahr);

            main()
            {
                    float fahr;
                    float c;
                    printf("Celsius :");
                    scanf("%f",&c);
                    fahr = (1.8*c) + 32.0;
                    printf("Fahrenheit : %f\n",fahr);
            }
```

PROGRAM 36—Write a program to find the transpose of a martix in sparse form.

SOLUTION

```
#include <stdio.h>
//#include <conio.h>
int a[100][100],b[100][100];

void main()
{
```

```
int i,m,n,p,q,col,t;
//clrscr();
printf("Enter the no. of rows");
scanf("%d", &a[0][0]);
printf("\nEnter the no. of cols");
scanf("%d", &a[0][1]);
printf("\nEnter the number of non zero terms");
scanf("%d", &a[0][2]);

for(i=1;i<=a[0][2];i++)
{
printf("\nEnter the value (that is non zero)");
scanf("%d",&a[i][2]);
printf("\nEnter the row  for %d  : ",a[i][2]);
scanf("%d",&a[i][0]);
printf("\nEnter the col for %d  : ",a[i][2]);
scanf("%d",&a[i][1]);
}

/* Printing for testing the sparse input */
printf("\n ****************************\n The martix you entered is \n
**********************\n\n Row  \t Col \t Value \t");
for ( i = 0;i <= a[0][2];i++)
{
printf("\n %d \t %d \t %d \t " , a[i][0],a[i][1],a[i][2]);
}

/* Calling function for evaluation of transpose */

m = a[0][0];
n = a[0][1];
t = a[0][2];

b[0][0] = n;
b[0][1] = m;
b[0][2] = t;

q=1;

for( col = 1; col <=n; col++)
{
            for(p = 1; p<=t;p++)
            {
                    if(a[p][1] == col)
                    {
```

```
                    b[q][0] = a[p][1];
                    b[q][1] = a[p][0];
                    b[q][2] = a[p][2];
                    q++;
                    }
            }
}                   //end of outer for loop

/* Printing the transposed matrix */

//getch();
printf("\nThe Transpose of the above matrix is ");
for ( i = 0;i <= a[0][2];i++)
{
printf("\n %d \t %d \t %d \t " , b[i][0],b[i][1],b[i][2]);
}
//getch();
}
```

PROGRAM 37—Write a program to implement tree.

SOLUTION

```
#include<stdio.h>
#include<malloc.h>
struct ll
{
            int num;
            struct ll *left,*right;
};
typedef struct ll node;
void create(node **p,int n)
{
            if((*p)==NULL)
            {
                    (*p)=(node*)malloc(sizeof(node));
                    (*p)->left=(*p)->right=NULL;
                    (*p)->num = n;
                    return;
            }
            if((*p)->num>n) create(&(*p)->left,n);
            else create(&(*p)->right,n);
}
void inorder(node *n)
{
```

```
            if(n!=NULL)
            {
                    inorder(n->left);
                    printf("%d\n",n->num);
                    inorder(n->right);
            }
}
main()
{
            int n;
            node *root = NULL;
            printf("Input positive numbers in tree:");
            while(1)
            {
                    scanf("%d",&n);
                    if(n<0) break;
                    create(&root,n);
            }
            inorder(root);
}
```

PROGRAM 38—Write a program to print the multiples of 7 without using the multiplication operator.

SOLUTION

```
#include<stdio.h>
int main()
{
  unsigned int num;
  unsigned int val;
  printf("Enter the number ");
  scanf("%d",&num);
  val= (num<<3)-num;// 7x can be written as 8x-x. 8x can be written as num<<3
as the binary value of 8 is 1000.
  printf("The multiplied value is %d", val);
}
```

PROGRAM 39—Write a program to check if 20th bit is on or off.

SOLUTION

```
#include<stdio.h>
int main()
```

```
{
  unsigned int num;
  printf("Enter the number ");
  scanf("%d",&num);
  if((num & 0x00001000)== 0x00001000) //0x00001000 is equivalent to 0x0000
0000   0000   0000   0001   0000   0000   0000. The 20th bit alone is AND
with 1 to check if it is set or not.
  {
           printf("The bit is on /n");
  }
  else
  {
           printf("The bit is off /n");
  }
}
```

PROGRAM 40—Write a program to add two polynomials.

SOLUTION

```
#include<stdio.h>
#include<malloc.h>
#include<stdlib.h>
struct link{
  int coeff;
  int pow;
  struct link *next;
};

struct link *poly1=NULL,*poly2=NULL,*poly=NULL;
//Create a polynomial.
void create(struct link *node)
{
    int ch;
   // get all the coefficients and powers of the polynomial equation
    do
    {
      printf("\n enter coeff:");
      scanf("%d",&node->coeff);
      printf("\n enter power:");
      scanf("%d",&node->pow);
      node->next=(struct link*)malloc(sizeof(struct link));
      node=node->next;
      node->next=NULL;
```

```
      printf("\n continue(yes(1)/no(0)):");
      scanf("%d",&ch);
     }
    while(ch==1);
}
// show() displays the equation.
void show(struct link *node)
{
        // display the polynomial equation
            while(node->next!=NULL)
        {
        printf("%dx^%d",node->coeff,node->pow);
        node=node->next;
        if(node->next!=NULL)
        printf("+");
        }
}
//polyadd() performs polynomial addition.
void polyadd(struct link *poly1,struct link *poly2,struct link *poly)
{
     // The powers of each polynomial is checked and the corresponding co-
efficients of the two polynomials are checked.
      while(poly1->next && poly2->next)
      {
       if(poly1->pow>poly2->pow)//When polynomial1's power is greater than
poly2's power, the power, coefficient of the poly1 is set to the resultant
polynomial.
        {
        poly->pow=poly1->pow;
        poly->coeff=poly1->coeff;
        poly1=poly1->next;
        }
       else if(poly1->pow<poly2->pow)//When polynomial2's power is greater
than poly1's power, the power, coefficient of the poly2 is set to the resul-
tant polynomial.
        {
        poly->pow=poly2->pow;
        poly->coeff=poly2->coeff;
        poly2=poly2->next;
        }
     else//When both the powers of poly1 and poly2 are same, the coefficients
of both polynomials are added and set to resultant polynomial's coefficient.
        {
        poly->pow=poly1->pow;
        poly->coeff=poly1->coeff+poly2->coeff;
        poly1=poly1->next;
```

```
        poly2=poly2->next;
        }
        poly->next=(struct link *)malloc(sizeof(struct link));
        poly=poly->next;
        poly->next=NULL;
      }
     while(poly1->next || poly2->next) // Check If still polynomial exists
      {
        if(poly1->next) //Set poly1's parameters to resultant polynomial
        {
        poly->pow=poly1->pow;
        poly->coeff=poly1->coeff;
        poly1=poly1->next;
        }
        if(poly2->next)//Set poly2's parameters to resultant polynomial
        {
        poly->pow=poly2->pow;
        poly->coeff=poly2->coeff;
        poly2=poly2->next;
        }
        poly->next=(struct link *)malloc(sizeof(struct link));
        poly=poly->next;
        poly->next=NULL;
       }
}
main()
{
       int ch;
       do{
        poly1=(struct link *)malloc(sizeof(struct link));
        poly2=(struct link *)malloc(sizeof(struct link));
        poly=(struct link *)malloc(sizeof(struct link));
        printf("\nenter 1st number:");
        create(poly1);
        printf("\nenter 2nd number:");
        create(poly2);
        printf("\n1st Number:");
        show(poly1);
        printf("\n2nd Number:");
        show(poly2);
        polyadd(poly1,poly2,poly);
        printf("\nAdded polynomial:");
        show(poly);
      }
}
```

PROGRAM 41—Write a program to implement binary search tree.

SOLUTION

```
#include <stdio.h>
#include <stdlib.h>

struct tnode {
 int data;
 struct tnode *left;
 struct tnode *right;
};

/* insert, swap, search value, search minimum and search maximum values */
struct tnode *tnode_insert(struct tnode *p, int value);
struct tnode *tnode_swap(struct tnode *p);
struct tnode *tnode_search(struct tnode *p, int key);
struct tnode *tnode_searchmin(struct tnode *p);
struct tnode *tnode_searchmax(struct tnode *p);

/* destroy, count tree nodes */
void tnode_destroy(struct tnode *p);
int tnode_count(struct tnode *p);

/* print binary tree inorder, preorder, postorder [recursive] */
void print_inorder(struct tnode *p);
void print_preorder(struct tnode *p);
void print_postorder(struct tnode *p);

int main(void) {
 int demo_nr[] = {1, 3, 4, 7, 2, 9, 9, 0, 5, 6, 8, 7, 1, 2, 4};
 struct tnode *root = NULL;
 struct tnode *searchval = NULL;
 int querry = 0;
 int i = 0;

 /* demo: insert some nr's into the binary tree */
 for(i = 0; i < 15; i++)
  root = tnode_insert(root, demo_nr[i]);

 printf("=-=-=\n");
 printf("Total number of tree nodes: %d\n", tnode_count(root));
 printf("inorder  : ");
 print_inorder(root);
 printf("\n");
```

```
printf("preorder : ");
print_preorder(root);
printf("\n");

printf("postorder: ");
print_postorder(root);
printf("\n");

printf("=-=-=\n");
printf("Enter integer, find: ");
scanf("%d", &querry);
searchval = tnode_search(root, query);
if(searchval == NULL)
 printf(" * %d Not! found in btree\n", query);
else
 printf(" * Found! %d in btree\n", searchval->data);

searchval = NULL;
printf("Searching for Minimum value\n");
searchval = tnode_searchmin(root);
if(searchval == NULL)
 printf(" * Minimum Not! found in btree ?\n");
else
 printf(" * Found! minimum value %d in btree\n", searchval->data);

searchval = NULL;
printf("Searching for Maximum value\n");
searchval = tnode_searchmax(root);
if(searchval == NULL)
 printf(" * Maximum Not! found in btree ?\n");
else
 printf(" * Found! Maximum value %d in btree\n", searchval->data);

printf("=-=-=\n");
printf("Exchanging all tree nodes: left <-> right\n");
root = tnode_swap(root);

printf("inorder  : ");
print_inorder(root);
printf("\n");

printf("preorder : ");
print_preorder(root);
printf("\n");
```

```
 printf("postorder: ");
 print_postorder(root);
 printf("\n");

 printf("=-=-=\n");
 printf("Destroying btree..!\n");
 tnode_destroy(root);

 return 0;
}

/* insert a tnode into the binary tree */
struct tnode *tnode_insert(struct tnode *p, int value) {
 struct tnode *tmp_one = NULL;
 struct tnode *tmp_two = NULL;

 if(p == NULL) {
  /* insert [new] tnode as root node */
  p = (struct tnode *)malloc(sizeof(struct tnode));
  p->data = value;
  p->left = p->right = NULL;
 } else {
  tmp_one = p;
  /* Traverse the tree to get a pointer to the specific tnode */
  /* The child of this tnode will be the [new] tnode */
  while(tmp_one != NULL) {
   tmp_two = tmp_one;
   if(tmp_one ->data > value)
    tmp_one = tmp_one->left;
   else
    tmp_one = tmp_one->right;
  }

  if(tmp_two->data > value) {
   /* insert [new] tnode as left child */
   tmp_two->left = (struct tnode *)malloc(sizeof(struct tnode));
   tmp_two = tmp_two->left;
   tmp_two->data = value;
   tmp_two->left = tmp_two->right = NULL;
  } else {
   /* insert [new] tnode as left child */
   tmp_two->right = (struct tnode *)malloc(sizeof(struct tnode));
   tmp_two = tmp_two->right;
   tmp_two->data = value;
   tmp_two->left = tmp_two->right = NULL;
  }
```

```
 }

 return(p);
}

/* print binary tree inorder */
void print_inorder(struct tnode *p) {
 if(p != NULL) {
  print_inorder(p->left);
  printf("%d ", p->data);
  print_inorder(p->right);
 }
}

/* print binary tree preorder */
void print_preorder(struct tnode *p) {
 if(p != NULL) {
  printf("%d ", p->data);
  print_preorder(p->left);
  print_preorder(p->right);
 }
}

/* print binary tree postorder */
void print_postorder(struct tnode *p) {
 if(p != NULL) {
  print_postorder(p->left);
  print_postorder(p->right);
  printf("%d ", p->data);
 }
}

/* returns the total number of tree nodes */
int tnode_count(struct tnode *p) {
 if(p == NULL)
  return 0;
 else {
  if(p->left == NULL && p->right == NULL)
   return 1;
  else
   return(1 + (tnode_count(p->left) + tnode_count(p->right)));
 }
}

/* exchange all left and right tnodes */
struct tnode *tnode_swap(struct tnode *p) {
```

```
 struct tnode *tmp_one = NULL;
 struct tnode *tmp_two = NULL;

 if(p != NULL) {
  tmp_one = tnode_swap(p->left);
  tmp_two = tnode_swap(p->right);
  p->right = tmp_one;
  p->left  = tmp_two;
 }

 return(p);
}

/* locate a value in the btree */
struct tnode *tnode_search(struct tnode *p, int key) {
 struct tnode *temp;
 temp = p;

 while(temp != NULL) {
  if(temp->data == key)
   return temp;
  else if(temp->data > key)
   temp = temp->left;
  else
   temp = temp->right;
 }

 return NULL;
}

/* locate a minimum value in the btree */
struct tnode *tnode_searchmin(struct tnode *p) {
 if(p == NULL)
  return NULL;
 else
  if(p->left == NULL)
   return p;
  else
   return tnode_searchmin(p->left);
}

/* locate a maximum value in the btree */
struct tnode *tnode_searchmax(struct tnode *p) {
 if(p != NULL)
  while(p->right != NULL)
```

```
   p = p->right;

 return p;
}

/* destroy the binary tree */
void tnode_destroy(struct tnode *p) {
 if(p != NULL) {
  tnode_destroy(p->left);
  tnode_destroy(p->right);

  free(p);
 }
}
```

PROGRAM 42—Write a program to implement circular queue through array.

SOLUTION

```
#include <stdio.h>
#include<ctype.h>
# define MAXSIZE 200

int cq[MAXSIZE];
int front,rear;

void main()
{
void add(int,int [],int,int,int);
int del(int [],int ,int ,int );
int will=1,i,num;
front = 1;
rear = 1;


printf("\n\t\tProgram for Circular Queue demonstrationthrough array\n\
n\n");
while(will ==1)
{
printf("\n\t\tMAIN MENU: \n\t1.Add element to Circular Queue\n\t2.Delete
element from the Circular Queue\n");
scanf("%d",&will);

switch(will)
{
```

```
case 1:
            printf("\nEnter the data... ");
            scanf("%d",&num);
            add(num,cq,MAXSIZE,front,rear);
            break;
case 2: i=del(cq,MAXSIZE,front,rear);
            printf("\nValue returned from delete function is  %d ",i);
            break;
default: printf("\n\nInvalid Choice . ");
}

printf(" \n\n\t\t\tDo you want to do more operations on Circular Queue ( 1
for yes, any other key to exit) ");
scanf("%d" , &will);
} //end of  outer while
}               //end of main
//Add elements in the circular queue.
void add(int item,int q[],int MAX,int front,int rear)
{
rear++;//rear is incremented whenever an element is added.
rear= (rear%MAX);
if(front ==rear)
            {
            printf("\n\n\t\tCIRCULAR QUEUE FULL\n\n");
            return;
            }
else
            {
            cq[rear]=item;
            printf("\nRear = %d    Front = %d ",rear,front);
            }
}
//delete elemets in the circular queue.
int del(int q[],int MAX,int front,int rear)
{
int a;
if(front == rear)
            {
            printf("\n\n\t\tCIRCULAR STACK EMPTY\n\n");
            return (0);
            }
else
            {
            front++;//front is incremented whenever a element is deleted.
            front = front%MAX;
```

```
            a=cq[front];
            return(a);
            printf("\nRear = %d    Front = %d ",rear,front);
            }
}
```

PROGRAM 43—Write a program that generically is legal according to both C and C++.

SOLUTION

However, if the file is stored with .cpp extension, then the program should behave differently. If the file is stored with .c extension, then again the program should behave differently.

```
Solution
#include<stdio.h>
#include<string.h>

main()
{
            char name[100];
// The name of the file will be stored in __FILE__ macro
            strcpy(name,__FILE__);
// Check the last character of the file name
            if(name[strlen(name)-1]=='c')
            {
                    // its a c file
                    printf("the program is a C file");
            }
            else
            {
                    // if its not c then its CPP file
                    printf("It is c++ file");
            }
}
```

PROGRAM 44—Write a program to find the number of bits set in a number.

SOLUTION

```
#include<stdio.h>
int main()
{
  unsigned int num;
  int ctr=0;
```

```
  printf("Enter the number ");
  scanf("%d",&num);
  for(;num!=0;num>>=1) //num>>x 'x' times right shifts 'num'.
  {
    if(num&1) // num&1 gives the value 1 if the last bit in 'num' is set.
    {
      ctr++;// ctr is the counter that counts the number of bits set.
    }
  }

  printf("\n Number of bits set in [%d] = [%d]\n", num, ctr);
  return(0);

}
```

PROGRAM 45— Write a program to perform password authentication using Hashing Separate Chaining method.

SOLUTION

```
/*Description of password authentication using Hashing Separate Chaining
method

* Users must give the passwords (strings) to populate the Hash Table.
* Hash Table is implemented using separate chaining to avoid collision.
* Each Password is encrypted using the encrypt() function.
* The encrypt() function adds the ASCII value of each character in the pass-
word, generating the key value which is an integer.
* Now the key is hashed using hash() function and stored in the hash table.
* The valid user is prompted for the password for which the same encryption
and key is hashed.
* The hash value is compared in the hash table.
* If a match is found, user is authenticated.
* Else error message is sent.

Separate Chaining Method:

* If two keys are hashed to the same value, collision occurs.
* To avoid this, Hash table is maintained with an array. For each index, a
pointer is maintained which points to a linked list containing nodes.
* These nodes contain key values which hash to the same value.
* Thus the hash table is maintained without collision.

*/
#include <stdlib.h>
```

```
#define MinTableSize (10)
  typedef int ElementType;
  typedef unsigned int Index;
  struct ListNode;
  typedef struct ListNode *Position;
  struct HashTbl;
  typedef struct HashTbl *HashTable;

   HashTable InitializeTable( int TableSize );     /* Declare the functions
here */
  void DestroyTable( HashTable H );
  Position Find( ElementType Key, HashTable H );
  void Insert( ElementType Key, HashTable H );
  ElementType Retrieve( Position P );

/* declare structure for node containing hashvalue (Element) and pointer to
next node */

struct ListNode
{
    ElementType Element;
    Position    Next;
};

typedef Position List;

/* List *TheList will be an array of lists, allocated later */
struct HashTbl
{
    ElementType TableSize;
    List *TheLists;
};

/* Hash function for ElementTypes */
Index
Hash( ElementType Key, ElementType TableSize )
{
    return Key % TableSize;             /* simple hash  take modulus of key
by the tablesize  so every hash fits into the table */
}


HashTable
InitializeTable( ElementType TableSize )
{
```

```
    HashTable H;
    ElementType i;

if( TableSize < MinTableSize )
    {
  printf( "Table size too small" );
  return NULL;
    }

    /* Allocate table */
H = malloc( sizeof( struct HashTbl ) );
if( H == NULL )
  printf( "Out of space!!!" );

H->TableSize =  TableSize ;

    /* Allocate array of lists */
H->TheLists = malloc( sizeof( List ) * H->TableSize );
     if( H->TheLists == NULL )
  printf( "Out of space!!!" );

    /* Allocate list headers */
     for( i = 0; i < H->TableSize; i++ )
    {
  H->TheLists[ i ] = malloc( sizeof( struct ListNode ) );
  if( H->TheLists[ i ] == NULL )
printf( "Out of space!!!" );
else
H->TheLists[ i ]->Next = NULL;
    }

return H;
}


Position
Find( ElementType Key, HashTable H )
{
    Position P;
    List L;

L = H->TheLists[ Hash( Key, H->TableSize ) ]; /* get the hash value for
given key; use it as index to point to the header of the list */
P = L->Next;
while( P != NULL && P->Element != Key ) /*now traverse the list until you
find the node with correct key value */
```

```
  P = P->Next;
return P;
}
void
Insert( ElementType Key, HashTable H )
{
    Position Pos, NewCell;
    List L;
Pos = Find( Key, H );
if( Pos == NULL )   /* Key is not found */
    {
  NewCell = malloc( sizeof( struct ListNode ) ); /*allocate new node for
the key */
  if( NewCell == NULL )
printf( "Out of space!!!" );
else
{
L = H->TheLists[ Hash( Key, H->TableSize ) ]; /*else find the hash value for
given key and use it as index to find the header of the list in the array
of lists */
NewCell->Next = L->Next;
NewCell->Element = Key;
     L->Next = NewCell;
}
    }
}
ElementType
Retrieve( Position P )
{   if(P==NULL)
    return -1;
    else
    return P->Element; /* if p is null, key is not found else, return the
corresponding element */
}
void
DestroyTable( HashTable H )
{
    ElementType i;

    for( i = 0; i < H->TableSize; i++ )
    {
Position P = H->TheLists[ i ];
Position Tmp;

while( P != NULL )
{
```

```
    Tmp = P->Next;      /* free each node in the list  */
    free( P );
    P = Tmp;
}
    }

    free( H->TheLists ); /* free the list header */
    free( H );            /* when memory allocated to each list is freed,
free the hashtable itself */
}
int encrypt(char *s)
{
int n=0;
while(*s)
n=(int)*s++;      /*simple encryption: for the string add the integer value
of each character and return it as key value */
return n;
}
int main()
{
int i=1,n;
char *s;
HashTable H;
H=InitializeTable(17);
s=(char *)malloc(sizeof(char)*100);
while(i)
{
printf("\nEnter the password to populate the hash table\n");
scanf("%s",s);               /* get the input */
printf("\n");
n=encrypt(s);                /* encrypt the string and get the key */
Insert(n,H);                 /* now insert the key */
printf("Enter 1 to Continue or 0 to Stop\n");
scanf("%d",&i);
printf("\n");
}
printf("\nPasswords are Hashed to their values\n");
printf("-------------------------------------------\n");
printf("\nEnter the password to authenticate\n");
scanf("%d",s);
printf("\n");
n=encrypt(s);              /* for the given password, encrypt and find the
key */
n=Retrieve(Find(n,H));     /* retrieve the key value using the hash value
for it */
```

```
if(n!=-1)                /* if it is found, then he is an authorized user */
printf("\nYour password is valid\n");
else
printf("\nUnAuthorized Entry\n");
DestroyTable(H);
return 0;
}
```

PROGRAM 46 —Write a program to print a square matrix helically.

SOLUTION

```
#include<stdio.h>
int main()
{
  unsigned int n,i,j,k,i1,j1;
  unsigned int v[100][100];
  printf("Enter the n value of square matrix ");
  scanf("%d",&n);
  printf("Enter the matrix value ");
  for(i=0;i<n;i++)
  {
            for(j=0;j<n;j++)
            {
                    scanf("%d",&v[i][j]);
            }
  }
//Print the matrix
  printf("Actual matrix \n");
  for(i=0;i<n;i++)
  {
            for(j=0;j<n;j++)
            {
                    printf("%d \t",v[i][j]);
            }
            printf("\n");

  }
  printf("the helical square matrix\n");
  for(j1=n-1, i1=0; i1 < j1; i1++, j1--)
  {
            for(k=i1;k<j1;k++)
                    printf("%d\t",v[i1][k]);//traversal from left to right
            for(k=i1;k<j1;k++)
                    printf("%d\t",v[k][j1]);//traversal from top to bottom
```

```
            for(k=j1;k>i1;k--)
                    printf("%d\t",v[j1][k]);//traversal from right to left
            for(k=j1;k>i1;k--)
                    printf("%d\t",v[k][i1]);//traversal from bottom to top
  }
  if((n%2)==1)
            printf("%d", v[((n-1)/2)][((n-1)/2)]);// print the middle ele-
ment.
return 1;
}
```

PROGRAM 47— Write a program to find out the larger of two numbers without using any comparison.

SOLUTION

```
#include<stdio.h>
#include<math.h>
main()
{
            int a,b;
            scanf("%d %d",&a,&b);
            // ADD sum and difference of 2 numbers
            // This cancels the effect of smallest number
            // (big+small) + (big-small)
            // we get a number 2*big .. hence divide by 2 to get the number
            printf("Largest :%d ",((a+b)+abs(a-b))/2);
}
```

PROGRAM 48— Write a program to find out the smaller of two numbers without using any comparison.

SOLUTION

```
#include<stdio.h>
#include<math.h>
main()
{
            int a,b;
            scanf("%d %d",&a,&b);
            // find difference between  sum and difference of 2 numbers
            // This cancels the effect of biggest number
            // (big+small) - (big-small)
```

```
            // we get a number 2*small .. hence divide by 2 to get the number
            printf("smallest :%d ",abs(((a+b)-abs(a-b))/2));
}
```

PROGRAM 49—Write a program to multiply two numbers without any arithmetic operators.

SOLUTION

```
#include<stdio.h>
int main()
{
int a,b;
char *p;
printf("Enter the two numbers to multiply\n");
scanf("%d %d",&a,&b);       /* get the two numbers to multiply */
p=(char *)a;  /* now the content of p is a */
while(--b)
{
p=&p[ a ];   /* here p[a] means to content of p add a */
}           /* continuing the above operation for b times a*b is computed */
printf("%d",p);
free(p);
}
```

PROGRAM 50 —Write a program to check if a number is a power of 2.

SOLUTION

```
#include<stdio.h>
int main()
{
  unsigned int n;
  printf("Enter the number ");
  scanf("%d",&n);
  if((n!=0) &&((n&(n-1))==0)) // when a number is a power of 2, it's pre-
vious number's bits will be complement of the number. That is, for example
take 8 == 1000 and 7== 0111. Both are complement of each other and hence
when we perform the AND operation between them, 0 will be the answer.
  {
            printf("The number %d is a power of 2 ",n);
  }
```

```
  else
            printf("The number %d is not a power of 2 ",n);
}
```

PROGRAM 51— Write a program to implement queens problem.

SOLUTION

```
#include<stdio.h>
#include<stdlib.h>
static int t[10]={-1};
void queens(int i);
int empty(int i);

void print_solution();

int main()
{
  queens(1);
  print_solution();
  return(0);
}

void queens(int i)
{
  for(t[i]=1;t[i]<=8;t[i]++)
  {
    if(empty(i))
    {
       if(i==8)
       {
          print_solution();
          /* If this exit is commented, it will show ALL possible combina-
tions */
          exit(0);
       }
       else
       {
          // Recurse! Go for next row.
          queens(i+1);
       }

    }
  }
} int empty(int i)
```

```
{
  int j;
  j=1;
//The position value of the present t[] is compared with the previously as-
signed row's position values.
  while(t[i]!=t[j] && abs(t[i]-t[j])!=(i-j) &&j<8)j++;

  return((i==j)?1:0);
}


//Print the solution.
void print_solution()
{
  int i;
  for(i=1;i<=8;i++)printf("\nt[%d] = [%d]",i,t[i]);
}
```

PROGRAM 52— Write a program to implement queue through structures and pointers.

SOLUTION

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node  *link;
}     ;
struct node *front, *rear;
int main()
{

int wish,will,a,num;
void add(int);
int del();
wish=1;
front=rear=NULL;

while(wish == 1)
            {
            printf("\nMain  Menu  \n1.Enter  data  in  queue  \n2.Delete  from
queue ");
            scanf("%d",&will);
            switch(will)
```

```
                    {
                    case 1:
                            printf("\nEnter the data");
                            scanf("%d",&num);
                            add(num);
                            //display();
                            break;
                    case 2:
                            a=del();
                            printf("\nValue returned from front of the queue
is %d ",a);
                            break;
                    default:
                            printf("\nInvalid choice ");
                    }
            printf("\nDo you want to continue, press 1 ");
            scanf("%d",&wish);
            }

}
// Function add is used to add elements in the queue.
void add(int y)
{
struct node *ptr;
ptr=(struct node*)malloc(sizeof(struct node));
ptr->data=y;
ptr->link=NULL;
if(front ==NULL)
            {
            front = rear= ptr;
            }
else
            {
            rear->link=ptr;
            rear=ptr;
            }
}

// Function del is used to delete the elements in FIFO(First In First Out)
fashion.
int  del()
{
int num;
if(front==NULL)
            {
            printf("\n            QUEUE EMPTY ");
```

```
            return(0);
            }
else
            {
            num=front->data;
            front = front->link;
            printf("\n Value returned by delete function is %d ",num);
            return(num);
            }
}
```

PROGRAM 53— Write a program to print semicolon *n* times without using semicolon anywhere in the program.

SOLUTION

```
#include<stdio.h>
#include<stdlib.h>
// get the number using the command line arguments
main(int argc,char *argv[])
{
            // The first command line argument is type casted into int to
read n
            if((argc=atoi(argv[1]))!=0)
            {
                    while(argc--)
                    {
                            // the ascii value of semicolon is used to print it
                            // printf returns the number of characters printed
                            if(printf("%c\n",59))
                            {
                            }
                    }
            }
}
```

PROGRAM 54— Write a program to implement stack through structures and pointers.

SOLUTION

```
#include<stdio.h>
#include<stdlib.h>
```

```
# include "malloc.h"
struct node
{
            int data;
            struct node *link;
}          ;
struct node  *top;

int main()
{
void push(int);
void display();
int pop();
int wish, num,will,a;
wish = 1;
top = NULL;
printf("Program for Stack as Linked List demo..");
while(wish == 1)
            {
            printf("Main Menu1.Enter data in stack 2.Delete from stack");
            scanf("%d",&will);
            switch(will)
                  {
                  case 1:
                        printf("Enter the data");
                        scanf("%d",&num);
                        push(num);
                        display();
                        break;
                  case 2:
                        a=pop();
                        printf("Value returned from top of the stack is
%d",a);
                        break;
                  default:
                        printf("Invalid choice");
                  }
            printf("Do you want to continue, press 1");
            scanf("%d",&wish);
            }
return 0;
}

// THIS FUNCTION ADDS AN ELEMENT IN THE STACK.
void push(int y)
{
```

```
struct node *x=(struct node *)malloc(sizeof(struct node));
printf(" Address of newly created node x is %d",x);
x->data = y;
x->link = top;
top = x;
}
void display()
{
int i =0;
struct node * temp;
temp = top;

            while(temp!=NULL)
            {
            printf("Item No. %d : Data %d Link %d",i++,temp->data,temp->link);
            temp=temp->link;
            }
}

//THIS FUNCTION REMOVES TOP NODE FROM THE STACK AND RETURNS ITS VALUE

int pop()
{
  int a;
  if(top==NULL)
  {printf("        STACK EMPTY..."); return 0;}
  else
  {
  a=top->data;
  printf("The value returned is %d ",a);
  free(top);
  top=top->link;            return (a);
  }
}
```

PROGRAM 55— Write a program to implement palindrome word testing program using stack.

SOLUTION

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
int i;
```

```
struct LLNode {                 // struct for linked list node
            char key;
   struct LLNode *next;
};

struct LLNode *initLLNode(char val);
char        empty (struct LLNode *head, struct LLNode *tail);
void        push  (char                    val, struct LLNode *head);
char        pop   (struct LLNode *curr);
int         is_palindrome (int length, char temp[80],struct LLNode *head,
struct LLNode *tail);
int main (void){
   int x;
   int length;
   int temp_count;
   int again;
   char inp[80];
   char temp[80];
   struct LLNode *head;
   struct LLNode *tail;
   do {
            head = initLLNode('\0');        // initialize head and tail to
NULL
            tail = initLLNode('\0');
            head->next = tail;              // head point to tail
      printf("\n*****************************************\n");
      printf("This is a program to test a palindrome word.\n\n");
            printf("Please enter a word: ");
            scanf("%s", inp);
            x= strlen(inp);
      //The for loop eliminated any spaces in the string
                  //and the new string is placed in array temp
                  temp_count = 0;
        for(i = 0;i < x;i++){
                  if(!isspace(inp[i])){
                  temp[temp_count]=inp[i];
            temp_count++;
            }
            }
            length = temp_count;
                  if(is_palindrome(length, temp, head, tail)){
                  printf("The string: <%s> is a palindrome.\n", inp);
            } else {
                  printf("The string: <%s> is not a palindrome.\n", inp);
            }
      printf("\n*****************************************\n");
```

```
            printf("Enter 1 to test another word (0 to quit): ");
      scanf("%d", &again);
   } while (again == 1); // repeat the loop for user convenient
    return 0;
}
/* initialize linked list node  */
struct LLNode *initLLNode (char val) {
            struct LLNode *temp;
   temp = (struct LLNode *) malloc (sizeof(struct LLNode));
   temp->key = val;
   temp->next = NULL;
   return temp;
}
/* test if the stack is empty */
char empty (struct LLNode *head, struct LLNode *tail){
            if(head->next == tail)
            return 1;
   else
            return 0;
}
/* push a value into stack */
void push (char val, struct LLNode *head){
   struct LLNode *temp;
   temp = initLLNode(val);
   temp->next = head->next;
   head->next = temp;
   return;
}

/* pop a value from stack */
char pop (struct LLNode *curr){
            struct LLNode *temp;
   char val;
   val = curr->next->next->key;
   temp = curr->next;
   curr->next = curr->next->next;
   free(temp);
   return val;
}
/* test if a word is palindrome */
int is_palindrome(int length, char temp[80],
                                                    struct    LLNode    *head,
struct LLNode *tail){
            int     is_pal = 0;
            char *revWord;
```

```
   // push input character to stack
        for(i = 0; i < length; i++){
            push(temp[i], tail);
   }
   printf("\n");
   // reserve memory for revWord
   revWord = (char *) calloc (length, sizeof(char));
   for(i = 0; i < length; i++){
            revWord[i] = pop(head);           // assign poped value to rev-
Word
      if(revWord[i] == temp[i]){
            is_pal++;                    // test if revWord equal to temp
      }
   }
   if(is_pal == length){
            return 1;
   } else {
            return 0;
   }
}
```

The McGraw·Hill Companies

Model Test Papers

13

MODEL TEST I

Time: 1 Hour
Max.: 30 Marks

Answer **all** the questions by choosing the correct option.

1. Which is the data type not supported by C?
   (a) char (b) boolean (c) float (d) long double
2. What is the minimum value of a signed int data type that is 8 bits in size?
   (a) –128 (b) –127 (c) –256 (d) –257
3. What is the value of x after execution of the following code?
   ```
   float x = 1/4.0 + 1/4
   ```
   (a) 0.25 (b) 0.50 (c) 0.75 (d) 0.00
4. Which one is not a valid floating constant?
   (a) 5E-7 (b) 10.5E 10 (c) 0.05e – 3 (d) 2.438F
5. The type cast operator is
   (a) cast() (b) type() (c) (type) (d) \
6. Identify the conversion specification used to print a character with the printf() function.
   (a) %c (b) %s (c) %char (d) %lc
7. How many times the following loop will be executed?
   ```
   main(){unsigned int i= 5; while (i- - >= 0);}
   ```
   (a) 6 (b) 1 (c) 0 (d) none
8. ```
   switch(i)
   {
   case 0: printf("RED-\n");
   case 1: printf("BLUE \n");
   case 2: printf("GREEN \n");
   default: printf("RBG \n");
   }
   ```
   ```

To execute all the printf() statements, what will be the value of i?

(a) 0 or 1 or 2 (b) any value other than 0, 1 or 2
(c) 0 (d) None of the above

9. What is the use of extern storage class?

(a) It is used to access a variable from another file.
(b) It is used to access a global variable.
(c) Both options a and b.
(d) Neither option a nor b.

10. Identify the valid expressions in the declaration **`int y[5] ={1, 2, 3, 4, 5};`**

1. y[2] 2. *(y + 2) 3. 2[y]

(a) option 1 (b) option 2 (c) options 1 and 2 (d) options 1, 2 and 3

11. Identify the wrong initialization.

(a) `char s[] = {'A','P','P','L','E','\0'};`
(b) `char s[] = {'A','P','P','L','E'};`
(c) `char s[] = ("APP" "LE" } ;`
(d) `char s[] = {'A', "PPLE"};`

12. What is the output of the following code?

```
main() {void display();display();}
void display(char *str) {printf("%s\n", str);}
```

(a) syntax error (b) execution error
(c) compile successfully and produce null (d) linking error

13. Which one will print the number of elements in an array, given **int a[10];**?

(a) `sizeof (a)/sizeof (int)` (b) `sizeof (a)`
(c) `sizeof (a[0])/sizeof (a)` (d) `sizeof (int)/sizeof (a)`

14. What is the correct statement regarding the second argument c in memset(s,c,n)?

(a) c is a char variable.(b) c is an int converted to an unsigned char.
(c) c is an int. (d) c is a char array.

15.
```
const int a = 0; static int b =1;
void fun() {int c = 2; static int d = 3;}
```

Which variables can be accessed from other files?

(a) variable a only (b) variables a and b (c) variables c and d (d) variable d only

16. What is the output of the following code?

```
{
  int x[5] = {0, 1, 2, 3, 4};
  int *p; p = x+3;
  printf("%d\n", p[-2]);
}
```

(a) 1 (b) 0 (c) undefined (d) invalid

17. What is the value of p after execution of the following code?

```
int *const p = 5;
```

If the address of p is 4044, p – – is

(a) 4 (b) 4042 (c) 4043 (d) invalid

18. The following declaration represents

```
int (*(*a[3])())[5];
```

(a) array of pointers to a function returning an array of pointers to int
(b) array of pointers to a function returning a pointer to an array of int
(c) array of pointers to an array of pointers to a function
(d) array of pointers to a function returning a pointer to a function

19. Assuming 16 bits for int, what is the value of p after execution of the following code? Assume p points to the memory location 2020 after successful allocation of memory.

```
int *p = malloc(sizeof (int));
p += 5;
```

(a) 2025 (b) 2030 (c) 2040 (d) invalid

20. Given the declaration `int y[2] [3] [2] ={ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11,12};` what is the value of *(*(*y+2))?

(a) 4 (b) 5 (c) 6 (d) Syntax error

21. What is the output of the following code?

```
int *p, *q, *r;
q=(int*)malloc(sizeof(int)); p = q - 1; r = q + 1;
*p = 10; *q = 20; *r = 30; p++; ++q;
printf("%d %d\n", *p, *q);
```

(a) 20 30 (b) 10 30 (c) 20 20 (d) 30 30

22. Given the declaration `int x [5] [3] [2]` ; the element x[h][i][j] may be accessed by

(a) *(*(x[h]+i)+j) (b) *(*(*(x+h)+i)+j) (c) both options a and b (d) none of the above

23. Given the declaration

```
int *pl, **p2, ***p3, v = 25;
pl = &v ;  p2 = &pl;  p3 = &p2;
```

What is the value of **p3?

(a) the value of v (b) the address of v (c) the address of p1 (d) the address of p2

24. What is the output of the following code?

```
char *language[2]; char **sptr;
language[0] = "ANSI C"; language[1] = "C++";
sptr = &language[1];
printf("%s\n", *sptr);
```

(a) C++ (b) ANSI C (c) undefined (d) invalid

25. Identify the invalid pointer arithmetic.

(a) Addition of float value to a pointer.
(b) Comparison of pointers that do not point to the elements of the same array.
(c) Subtracting an integer from a pointer.
(d) Assigning the value 0 to a pointer variable.

26. Identify the correct structure declaration.

(a) `typedef struct {member declarations;};`
(b) `struct tag {member declarations;}`

(c) typedef struct tag {member declarations;}NEWNAME;
(d) typedef struct {member declarations;} NEWNAME;

27. How to access the field regno given the following declaration?

```
typedef struct stud
                    {
                      int regno;
                      char name[30];
                    } STUDENT;
STUDENT *sp = malloc (sizeof(STUDENT));
```

(a) *sp.regno (b) sp->regno (c) (*sp).regno (d) options b and c

28. Stream oriented files are called as

(a) low level files (b) high level files (c) system oriented files (d) none of the above

29. What is the output of the following code?

```
#define DECIMAL(X)                3.# #X
main()
  {
    printf("%.2f\n", DECIMAL(14));
  }
```

(a) 3.00 (b) 3 (c) 3.14 (d) Error

30. What are the macros used in functions with variable arguments?

(a) va_start (b) va _arg (c) va _end (d) all the above

KEY TO MODEL TEST-1 QUESTIONS

1. b	11. d	21. a
2. a	12. c	22. c
3. a	13. a	23. b
4. b	14. b	24. a
5. c	15. a	25. a
6. a	16. a	26 .d
7. d	17. d	27. d
8. c	18. b	28. b
9. c	19. b	29. c
10. d	20. b	30. d

MODEL TEST II

Time: 2 Hours
Max.: 60 Marks

Answer **all** the questions by choosing the correct answer.

1. Which is the data type NOT supported by C?

(a) signed char (b) long double (c) long float (d) char

2. What is the minimum value of an unsigned int data type that is 16 bits in size?
   (a) 0 (b) –65536 (c) –65535 (d) –32768
3. Identify the invalid constant.
   (a) "I (b) " " (c) "\b" (d) '\c'
4. Manifest is defined as
   (a) `# define MAX 10` (b) `# define MAX 10;`
   (c) `# define MAX 10` (d) `# define MAX = 10`
5. What is the output of the following code?

```
int a = 4, b = 8; double c;
c = (double) (a/b);
printf("%f\n",c);
```

   (a) 0.500000 (b) 2.000000 (c) 1.000000 (d) 0.000000
6. What is the value of y in the following code?

```
#define MINUS(A) A-1
main()
    {
      int x = 3; y = 4 * MINUS(x * 4);
      printf("%d\n", y);
    }
```

   (a) 47 (b) 44 (c) 36 (d) error
7. Which of the following expressions yield the same value?
   (a) `10 & 1; 10 ^ 1` (b) `10 ^ 1; 10 | 1`
   (c) `10«2 ; 10^1` (d) `10»2 ; 10^1`
8. If both the operands of the operator / are float types the result is
   (a) a float value (b) an int value (c) undefined (d) boolean value
9. Identify the logical operator.
   (a) ! (b) != (c) == (d) ~
10. Identify the invalid compound assignment.
    (a) %= (b) >>= (c) ^ = (d) ~ =
11. When the digit 0 is entered from keyboard, what is the value returned by getchar() function?
    (a) 48 (b) 0 (c) '\0' (d) –1
12. Conversion specification commences with
    (a) # (b) % (c) ( ) (d) conversion character
13. What is the output of the following code?

```
int y= 0; for (;;) {if(y++ == 5) break;}
printf("%d\n",y);
```

    (a) 0 (b) 5 (c) 6 (d) 7
14. What will be the values of i and j after execution of the following code?

```
i=0; j = 0; for (j=1; j < 5; j++)  {i = i + 1;}
```

    (a) i = 4 j = 5 (b) i = 5 j = 5 (c) i = 4 j = 4 (d) i = 5 j = 4

15.
```
int x = 2, a = 2, b = 3, c = 8;
if (x==b) x = a;
<5; j++) {i = i +1;}
else if (x ==a) c = c + b;
     else c = c + a;
printf("c = %d\n",c);
```
What will be the output?

(a) c = 10 (b) c = 2 (c) c = 8 (d) c = 11

16. Which one will form an infinite loop given the following declaration?
```
int y = 10;
```
(a) `while (2) { if ((y == 10) ) break;...}`
(b) `while (!0) { if ((y!= 10)) break;...}`
(c) `while (!(y == 10)) {...}`
(d) `do {} while ((y>10));`

17. Which one includes all the properties of a storage class?

(a) linkage and scope (b) scope and longevity
(c) longevity and linkage (d) scope, longevity and linkage

18. Identify the valid scope of variables.

(a) external and static (b) global, file and block
(c) file and block (d) internal, external and static

19. What will be the values assigned to the array elements in the following declaration?
```
int a[5] = {1,2};
```
(a) a[0] = 1, a[1] = 2, a[2] = 0, a[3] = 0, a[4] = 0
(b) a[0] = 1, a[1] = 2, a[2], a[3] and a[4] are undefined.
(c) a[0] = 1, a[1] = 2, a[2] = 1, a[3] = 1, a[4] = 1
(d) Invalid initialization because the number of values given in the initialization is less than the size of array.

20. The number of elements in an array declared below is
```
int x[10][10][10];
```
(a) 110 (b) 30 (c) 100 (d) 1000

21. What is the output of the following code if the function is called with a = 229?
```
void print(int a)
  {
    if (a!= 0)
        {
          print(a/8);
          printf("%c", "abcdefg"[a%8]);
        }
    else printf("\n");
  }
```
(a) def (b) fed (c) cde (d) edc

22. What will be the output of the following code?

```
main()
  { call(4);}
void call(int a)
  { if (a < 9) call (++a);
     printf("%d", a);
  }
```

(a) 567899 (b) 987655 (c) 998765 (d) 987654

23. What is the output of the following code?

```
char sl[11] = "POWERFUL C";
char s2[11] = "FLEXIBLE C";
strcpy (sl,s2); printf ("%s\n", sl);
```

(a) FLEXIBLE (b) FLEXIBLE C
(c) POWERFUL (d) POWERFUL C FLEXIBLE C

24.
```
void *pl = "CONSTANT";
void *p2 = "CONTINUE";
```

Which of the following code will return zero?

(a) `memcmp (pl, p2, 3);` (b) `strncmp (pl, p2, 3);`
(c) `! memcmp (pl, p2, 3);` (d) `!strncmp (pl, p2, 3);`

25. Which one is not defined in string.h?

(a) strspn() (b) strerror() (c) memchr() (d) strtod()

26. Identify the correct argument for the function call **fflush()** in ANSI C.

(a) stdout (b) stdin (c) stderr (d) option a or c

27. What is the output of the following code?

```
putchar(5["ANSI C"]);
```

(a) C (b) blank (c) undefined (d) invalid

28.
```
main(int argc, char *argv[])
  {
    printf("%d %s\n", argc, argv[2]);
  }
```

If this program is executed using the following command after successful compilation, what is the output?

```
a. out line plane circle
```

(a) 4 plane (b) 3 circle (c) 4 circle (d) 3 line

29. What is to be replaced in ??? portion of the following code to get the square of 5?

```
main()
    {
      int (*ap[2])();
      ap[0] = square; ap[1] = cubic;
      printf ("square of %d is %d\n", 5,???);
    }
square(int x) {return x * x;}
cubic (int x) {return x * x * x;}
```

(a) (*ap[0]) (5) (b) (*ap) [0] (5)
(c) ap[0] (5) (d) *(*ap[0]) (5)

30. Identify the undefined pointer arithmetic.
(a) Multiplying two pointers
(b) Shifting pointers
(c) Comparison of pointers that do not point to the elements of the same array
(d) Adding a pointer and an integer

31. What is to be replaced in ??? portion of the code?

```
typedef struct {char name[20]; int empid} RECORD;
main()
    {
      RECORD *r;
      r = malloc(sizeof (RECORD));
      scanf ("%s%d", ???);
      printf ("%s%d\n", r->name, r->empid);
    }
```

(a) `r->name, &r->empid` (b) `r->name, r->empid`
(c) `&r->name, &r->empid` (d) `r->name, r.empid`

32. What is the effect of the following code?

```
union mixed (float f; int i;} mix = 5;
```

(a) f = 5.0 (b) i = 5 (c) f = 5i = 5 (d) error

33. Identify the correct statement(s) given the following code. Assume 16 bits as word size of the computer.

```
struct
     {
      int first_bit:1;
      unsigned: 14;
      int last_bit: 1;
     } bf;
  bf.first_bit = 1; bf.last_bit = 1;
```

(a) unnamed bit field width is 14. (b) first and last bits are assigned.
(c) other than first and last bits are padded. (d) all the above.

34. Identify the incorrect statement.
(a) bit fields do not have addresses.
(b) an array cannot have bit fields.
(b) bit fields can be read using scanf() function.
(c) bit fields cannot be accessed using pointer.

35. Identify the portable expression to obtain the most significant bit of an unsigned integer y.
(a) y & 0XFF000000 (b) y > > 24
(c) y > > (8*(sizeof(int) – 3)) (d) y > > (8*(sizeof(int) – 1))

36. Give the order of evaluation of the expression.

    a||!b&&c

    (a) !b, a||(!b) I (a||(!b)) && c
    (b) !b, (!b) &&c, a||(!b)) &&c)
    (c) b&&c, !(b&&c), a||(!(b&&c))
    (d) invalid expression

37. Which one of the following operators has highest precedence?

    >> ! = ^ ? : %

    (a) >
    (b) ?:
    (c) %
    (d) ^

38. What are the values of p and q after execution of the following statement and giving inputs as 356 47 3 ?

```
scanf("%2d %2d", &p, &q);
```

    (a) p=35q=47
    (b) p=35q=6
    (c) p = 35 q = 64
    (d) p = 356 q = 47

39. Which of the following code will interchange two integers without using a temporary variable?

    (a) (a ^ = b); (b ^ = a); (a ^ = b)
    (b) a = a + b; b = a – b; a = a – b;
    (c) a ^ = (b ^ = a)
    (d) options a and b

40. What is the output of the following code?

```
char s[] = "WELCOME";
printf("%d\n", strchr(s, 'e') - s);
```

    (a) prints the index of last occurrence of e in s
    (b) prints the index of first occurrence of e in s
    (c) prints the length of prefix up to first e in s
    (d) prints the length of prefix up to last e in s

41. What will be output of the following code?

```
char x[15], y[15], *p = y;
strcpy(x, "ILANGOVAN"):
strcpy(y, "KAMBAN"); p = x;
strcpy(x,"KALA"); *p = 'M';
```

    (a) p = MALA
    (b) p = MALAUTVAN
    (c) MAMBAN
    (d) p = MALAGOVAN

42. Identify the portable expression to obtain the MSB of an int y.

    (a) y || OXF0000000
    (b) y || OXF000
    (c) y > > CHAR_BIT *(sizeof(int) – 1) + 7
    (d) y>> CHAR _BIT *(sizeof(int) – 1) + 8

43. If ch is a char variable and ch assumes any alphabet the expression ch || 32 is equivalent to

    (a) changing upper case to lower case
    (b) lower case to upper case
    (c) tolower(ch)
    (d) toupper(ch)

44. What is the output of the following code?

```
int a[2][3] ={1,2,3,4,5,6};
int *ptr = a[1]; ptr -= 3; printf("%d\n", *ptr);
```

    (a) 1
    (b) syntax error
    (c) 2
    (d) undefined

45. What is the output of the following code?

```
int x[4][5][8];
printf("%d\n", *(x[2] + 3) +5);
```

    (a) the value of x[2][3][5]
    (b) the address of x[2][3][5]
    (c) syntax error
    (d) undefined

46. Identify the valid linkage of identifiers.

(a) internal and external linkage. (b) internal, external and file linkages.
(c) internal, external and no linkages. (d) block and file linkages.

47. Given the inputs **x y** which one will assign x to p and y to q?

```
char p,q;
```

(a) `scanf("%c%ls", &p, &q);` (b) `scanf("%*[]%c", &p, &q);`
(c) `scanf("%c%c", &p, &q);` (d) all the above

48. What is the value of **a** after execution of the expression **a = b– = c*=5** given b= 110 and c= 20?

(a) 450 (b) 10 (c) 110 (d) –10

49. What is the output of the following code if the input is given as August 15th 1947 followed by a line feed?

```
main()
    {
      int digit = 0; char ch;
      while ((ch = getchar(    )) != '\n')
          {
            switch(ch)
              {
                case '0':
                case '1':
                case '2':
                case '3':
                case '4':
                case '5':
                case '6':
                case '7':
                case '8':
                case '9': digit ++;
              }
      printf("%d\n", digit);
      } }
```

(a) 6 (b) 1 (c) runtime error (d) syntax error

50. Pointers are used to

(a) return multiple values (b) achieve call by reference
(c) both options a and b (d) neither option a nor b

51. What does the declaration mean?

```
float *(*a[5])()[3];
```

(a) array[5] of pointer to function returning an array[3] of float
(b) pointer to an array[5] of function returning an array[3] of float
(c) array[3] of pointer to function returning an array[5] of float
(d) pointer to an array[5] of function returning an array[3] of float

52. Which of the following is true?
    1. C allows static memory allocation.
    2. C allows dynamic memory allocation.

    (a) option 1 only (b) option 2 only (c) options 1 and 2 (d) neither option 1 nor 2

53. What is the output of the following code?

```
main()
    {
    char *t[5];
    strcpy(t[0], "BASIC");
    printf("%s\n", t[0]);
   }
```

    (a) BASIC (b) illegal code (c) B (d) compilation error

54. What is the output of the following code?

```
main()
   {
    int *pl, **p2, ***p3;
    pl = (int *) malloc(sizeof (int));
    p2 = (int **) malloc(sizeof (int));
    *pl = (int) p2;
    p3 = (int ***) malloc(sizeof (int));
    *p2 = (int ) p3; *p3 = (int**)10;
    printf("%u\n", *p3);
   }
```

    (a) 10 (b) the value pointed to by p2
    (c) illegal assignment (d) runtime error

55. Identify the illegal expressions given the following declarations.

```
int x ; register int i;
```

    (a) &x (b) (*&)x (c) &i (d) options b and c

56. What is the dimension of array x in the following declaration?

```
char x[][6] = {'d', 'e', 'i', 'o', 'u', '\0'};
```

    (a) 1 × 6 (b) 6 × 6
    (c) number of rows not defined and hence error
    (d) Dynamic array with 6 columns

57. What is the output of the following code?

```
main()
   {
    int buf [5] ;
    int *bptr = &buf[3];
    bptr[-1] = 2; bptr[-2] = 1;
    bptr[-3] = 0;
    printf("%d \n", buf[2]);
   }
```

(a) 1 (b) 2 (c) 0 (d) error

58. What will be the final values of i and j when the following code terminates?

```
main()
  {
    static int i, j= 5;
    for (i = 0; i< 3; i++)
      { printf("%d%d\n", i, j);
       if(j > 1) {j--; main( );}
      }
  }
```

(a) 02 (b) 11

(c) 21 (d) error, because main cannot be called recursively

59. Which one of the following is true?

1. Structure may contain union
2. Union may contain structure
3. Union may contain bitfield
4. Structure cannot mix ordinary field with bitfields

(a) options 1 and 3 (b) options 1 and 2

(c) options 1, 2 and 4 (d) options 1, 2 and 3

60. Which one of the following is true?

1. FILE is a data type
2. A block of information in an object of type FILE is recorded while reading or writing.
3. FILE is a storage region.

(a) option 1 only (b) options 1 and 2

(c) options 2 and 3 (d) options 1, 2 and 3

KEY TO MODEL TEST-II QUESTIONS

1. c	11. a	21. a	31. a	41. a	51. a
2. a	12. b	22. c	32. a	42. c	52. c
3. d	13. c	23. b	33. d	43. c	53. b
4. c	14. a	24. a	34. c	44. a	54. a
5. d	15. d	25. d	35. d	45. b	55. d
6. a	16. b	26. d	36. b	46. c	56. a
7. b	17. d	27. a	37. c	47. d	57. b
8. a	18. c	28. a	38. b	48. b	58. c
9. a	19. a	29. a	39. d	49. a	59. b
10. d	20. d	30. c	40. b	50. c	60. b

Crack the Tough Nuts

14

1. Write a program to print the following with the word **structured** underlined. C is a structured language.
2. Write a program to round off a floating point value.
3. Write a program to find the word size of your host machine.
4. Write a program to interchange two numbers without using temporary variables.
5. Write an efficient program to find the factorial values for the numbers up to 10.
6. Write the eqivalent code for the following by reducing the number of iterations.

```
for(i=0;i<100;i++)
 {
  fun (i);
 }
```

7. Write a program to convert a lower case letter to upper case letter using bitwise operator.
8. Write a program to find the GCD of 2 positive integers using recursion.
9. Imagine a small railway line with 15 railway stations. How many different kinds of tickets the railway has to print?
10. Generate the first 10 terms of the following sequence using a loop construct.

    –3, 4, –6, 10, –18, 34, ...
11. Generate the first 10 terms of the following sequence using a loop construct.

    1, –3, 6, –10, 15, –21, ...
12. Generate the following sequence of numbers by expressing each term of the sequence formed from identical digits.

    12, 23, 34, 45, 56, 67, 78, 89, 100
13. Write a program to generate the following figure using loop construct.

```
                    1
          2     5
          6     9    14
         15    18    23    30
   31    34    39    46    55
```

14. Write a program to generate the following figure using loop construct.

```
            1
        1   2   3
    1   2   3   4   5
1   2   3   4   5   6   7
    1   2   3   4   5
        1   2   3
            1
```

15. Write a program to find an integer ODD or EVEN without using any control construct.
16. Write a program to generate the first four perfect numbers.

    (**Hint:** A perfect number is a positive integer that equals the sum of its positive integer divisors, including unity but excluding the number itself. The number 6 is the first perfect number. (6=1+2+3).)
17. Write a program to generate queer numbers.

    (**Hint:** A queer number is perfect square whose remainder value when divided by thousand is divisible by 111. For example, 289444 (538*538) is a queer number.)
18. Write a program to generate Armstrong numbers.

    (**Hint:** An Armstrong number is a three digit number that is equal to the sum of the cubic values of the integer digits in it. For example, 153(1^3+ 5^3 + 3^3) is an Armstrong number.)
19. Write a program to generate Pythagorean triplets.

    (**Hint:** Pythagorean Triplets are the three numbers a,b,c so that a < b < c with $a^2 + b^2 = c^2$. (3,4,5) is one such triplet.)
20. Write a program to find all the possible combinations of a 5 letter word.
21. Write a program to check whether a given string is a palindrome or not using recursion.
22. Guess a three digit number x having identical digits. Give the quotient value y when x is divided by 37. Write a program to find x.
23. Write a program to read two positive integers x and y and find the last n digits of x+y.
24. Write a program to find Ramanujan's number that is the sum of the cubes of two positive integers in two different ways.
25. A merchant wants to weigh up to 40 kilograms using minimum number of weights. Write a program to find the minimum number of weights and weighing capacity of each weight.
26. Write a program to find the day of a week on which a given date falls.
27. A man wishes to cross a river. He has with him a goat, a cabbage and a wolf. The man's boat can hold only him and one of his possessions at any given time. Therefore, the man must leave two of his possessions on the shore at any given time, as he brings the third item across. He cannot leave the goat with the cabbage, because the former will eat the later; for similar reasons, he cannot leave the wolf with the goat.

    Write a program to solve this puzzle using recursive function.

SOLUTION FOR TOUGH NUTS

1.

```
main()
  {
    printf("C is a structured\b\b\b\b\b\b\b\b\b\b_ _ _ _ _ _ _ language.\n");
  }
```

The output on the screen will be:

C is a – – – – – – – language.

Only an underline without the word structured is displayed because only one character (latest one) can be displayed at a time on the screen in a particular position. Since underscore is written after the word structured as per the program, undersore is displayed. The correct output will be obtained if a **print out** is taken. The characters in the word **structured** and **underscores** will be printed on the same position and hence the effect of underline is obtained. Try to use hyphens instead of underscores. The effect is striking off the word. If XXXXXXXXXX is used in the place of underscores, the effect is crossing each character of the word preceding it.

2.

```
main
  {
    float y;
    int x;
    printf("Enter a float value:");
    scanf("%f",&y);
    printf("\nGiven float value: y = %f\n",y);
    x=y+0.5;
    printf("After round off: y = %d\n",x);
  }
```

Output:

Enter a float value: 5.7

Given float value: y = 5.700000

After round off: y = 6

By adding 0.5 to a float value, the float value gets next higher integer part. Assigning the float to an int results in truncation of the fractional part and the result is as required.

To print exactly one decimal place after the decimal point and round off to one tenth position, add .05 and use %. lf. Some compilers automatically round off the value for %.lf.

Rounding off an integer to its tenth position can be done by adding 5 to the integer. Then the resulting number is divided by 10. The quotient is then multiplied by 10 resulting in rounding off the given integer to its tenth position. The same concept can be applied for rounding off an integer to its hundredth position by adding 50 and then dividing the resulting number by 100 and then multiplying the quotient by 100.

3.

```
main()
    {
      int i; unsigned x= -0; /* x is having all bits set to 1*/
      for(i=1; (x=x»1)> 0;i++);
      printf("Word size in bytes: %d\n",i/8);
      printf("Word size : %d %d\n",sizeof(int), sizeof i);
    }
```

Output:

Word size in bytes: 4

Word size: 4 4

The word size of a host machine is considered as the size of int data type. Hence, the sizeof(int) is used to verify it. Observe that **sizeof i** does not require parentheses. If the data type is used as the operand for sizeof operator, parentheses are mandatory and for variable names, parentheses are optional. Shifting all bits as given also helps in finding word size.

4.

```
#define SWAP(x,y) {x ^= y; y ^=x; x ^= y;}
main()
    {
      int a,b;
      printf("Enter 2 numbers: ")
      scanf("%d%d",&a,&b);
      printf("\nBefore SWAP: a=%d,b=%d\n",a,b);
      SWAP(a,b);
      printf("After SWAP: a=%d,b=%d\n",a,b);
      swap(a,b);
    }
      swap(int a, int b)
       {
          printf("Before swap: a=%d,b=%d\n",a,b);
          a=a+b;
          b= a-b;
          a= a-b;
          printf("After swap: a=%d,b=%d\n",a,b);
        }
```

Output:

Enter 2 numbers: 25 40
Before SWAP: a=25, b=40
After SWAP: a=40, b=25
Before swap: a=40, b=25
After swap: a=25, b=40

The program uses two methods of swapping without using temporary variables.

5.

```
long int a[]={1,1,2,6,24,120,720,5040,40320,403200};
main()
    {
      long int fact();
      int n;
      printf("%ld\n",fact(n));
    }
long int fact(int n)
    {
      return a[n];
    }
```

The program does not use any calculation in the function. Instead the factorial values of 1 to 10 are stored in an array and the respective values are returned from the function. Such method of solving problems is known as **table look up** method. This concept may be helpful in many situations to write efficient programs.

6.

The given loop can be written using loop unrolling as given below.

```
for(i=0;i<100;i++)
  {
    fun(i); i++;
    fun(i); i++;
    fun(i); i++;
    fun(i); i++;
    fun(i); i++;
    fun(i); i++;
    fun(i); i++;
    fun(i); i++;
    fun(i); i++;
    fun(i); i++;
  }
```

If the function fun() does not use i the code may be as follows.

```
for(i=0;i<100;)
    {
      fun() ;
      fun() ;
      fun() ;
      fun() ;
      fun() ;
      fun() ;
      fun() ;
      fun() ;
      fun() ;
      fun() ;
      i+=10;
    }
```

The time taken for a loop is more if the number of iterations are more. Reducing the number of iterations (10 in this case) in a loop makes the program more efficient. The method of reducing the iterations in a loop is known as **loop unrolling**.

Also refer to Appendix C to observe the timings of basic operations in C to write an efficient program for faster execution.

7.

```
main()
  {
    int ch;
    for(ch='A';ch<='Z';ch++)
    putchar(ch^32) ; /* uppercase to lowercase*/
    printf("\n");
    for(ch='A';ch<='Z';ch++)
    putchar(ch|32) ; /* uppercase to lower case*/
    printf("\n");
    for(ch='a';ch<='z';ch++)
    putchar(ch^32);   /*lowercase to uppercase*/
    printf("\n");
    for(ch='a';ch<='z';ch++)
    putchar(ch|32);
}
```

Output:

```
abcdefghijklmnopqrstuvwxyz
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
```

Observe that the operator ^ can be used for both the conversions upper to lower and lower to upper whereas the operator | can be used only for uppercase to lowercase.

8.

```
main( )
    {
      int a,b;
      printf("Enter two positive integers.\n");
      scanf("%d%d",&a,&b);
      printf("GCD of %d and %d is %d\n",a,b,gcd(a,b));
    }
    gcd(int a,int b)
      {
        if(a)
          return( b ? gcd(b,a%b),: a);
        else
          return b;
}
```

Output:

Enter two positive integers.
24 36
GCD of 24 and 36 is 12

9.

At each station, passengers can get tickets to any of the other 14 stations. Hence the railway has to print 210 (15 × 14 = 210) number of different tickets. In general, for n stations, n*(n–1) different tickets are required.

10.

```
main()
    {
      int n= -3,m=4,i;
      for(i=l;i<=5;i++)
        {
          n *=i;
          printf("%d %d ",n,m);
          m = m*(i+l) + 2*i ;
        }
    }
```

Output:

–3 4 –6 10 –18 34 –72 142 –360 718

11.

```
main()
    {
       int i,sum=0;
       for(i=1; i<=10;i++)
         {
          sum += i;
          if( i%2 )
            printf("%d ",sum);
          else
            printf("%d ",-sum);
          }
        }
```

Output:

1 –3 6 –10 15 –21 28 –36 45 –55

12.

```
main()
    {
      int a[]={11,22,33,44,55,66,77,88,99};
      int i;
      for(i=1;i<10;i++)
```

```
        printf("%d+%d/%d\n",a[i-1],i,i);
      }
```

Output:

11+1/1
22+2/2
33+3/3
44+4/4
55+5/5
66+6/6
77+7/7
88+8/8
99+9/9

13.

```
main ()
   {
       int i,j,k=1;
       printf("%d\n",k);
       for(j=1;j<5;j++)
         {
          printf("%d\t",++k);
          for(i=0;i<j;i++)
            {
              k+=2*i+3;
              printf("%d\t",k);
            }
         printf("\n");
        }
  }
```

14.

```
main( )
   {
      int n,i=1,j,k;
      for(n=5;n>1;n--)  /*First nested for loop */
       {
       for(j=n;j>1;j--)
        printf("\t");
      for(k=1;k <=i;k++)
       printf("%d\t",k);
     i += 2;
      printf("\n");
     }
```

```
i -= 2;
    for(n=1;n<5;n++)  /*Second nested for loop */
     {
      for(j=0;j<=n;j++)
      printf("\t");
      i -= 2;
    for(k=1;k<=i;k++)
    printf("%d\t",k);
    printf("\n");
  }
}
```

15.

Refer to **8.72** in short answer type questions of Chapter 8.

16.

```
main()
  {
    int n,i,sum=0,count=0;
    for(n=4; ;n++)
     {
      sum=0;
      for(i=1;i<=n/2;i++)
      {
        if((n%i)==0)
        sum+=i;
      }
if (sum==n)
  {
      printf("%d\n",sum);
      count++;
      if(count==4)
      goto end;
    }
  }
end:
      ; /* Null Statement  */
  }
```

Output

```
6
28
496
8128
```

17.

```
main( )
   {
    int i;
    printf("The generated queer numbers :\n");
    for(i=11;i < 2000;i++)
    if(queer(i) != 0)
    printf("%d ",queer(i));
    printf("\n");
  }
queer(int x)
  {
    int y;
    if(((Y=(x*x) %1000) != 0) && ((y % 111) == 0))
    return x*x;
  else return 0;
}
```

Output:
The generated queer numbers:
1444 213444 289444 925444 1077444 2137444 2365444 3849444

18.

```
main( )
    {
        int n,a,b,c,temp;
        printf("The Armstrong numbers are\n");
        for(n=100;n<1000;n++)
          {
              a = n%10;
              b = (n%100)/10;
              c = n/100;
              temp = a*a*a+b*b*b+c*c*c;
              if(n == temp)
              printf("%5d",n);
          }
    }
```

Output:
The Armstrong numbers are
153 370 371

19.

```
main()
    {
      int i,j,k;
```

```
        for(i=1;i<=20;i++)
        for(j=i+l;j<=20;j++)
        for(k=j+l;k<=20;k++)
        if((i*i +j*j) == (k*k))
        printf("Square(%d)+Square(%d)=Square(%d)\n",i,j,k);
      }
```

Output:

Square(3) + Square(4) = Square(5)
Square(5) + Square(12) = Square(13)
Square(6) + Square(8) = Square(10)
Square(8) + Square(15) = Square(17)
Square(9) + Square(12) = Square(15)
Square(12) + Square(16) = Square(20)

20.

```
#define MAX 5
main( )
{
char t[MAX];  int i,j,k,l,m,n= 1;
printf("Enter the string:\n");  gets(t);
printf("Given string:\n");puts(t);
printf("\nWords formed from the combinations\n");
printf("of all characters from a five letter word:\n");
for(i=0;i<MAX;i++)
  for(j=0;j<MAX;j++)
    {
    if(j == i) continue;
    for(k=0;k<MAX;k++)
    {
      if(k == i || k == j) continue;
      for(1=0;1<MAX;1++)
      {
      if(l == i || 1==||  1 == k) continue;
      m = 10 - (i+j+k+l):
      putchar(t[i]); putchar(t[j]); putchar(t[k]);
      putchar(t[1]); putchar(t[m]); putchar(' ');
      if(n%10 == 0)
      putchar('\n'); n++;
      }
    }
   }
      printf("\nNo. of words formed : %d\n",n-1);
    }
```

Output:
Enter the string:
madam
Given string:
madam

Words formed from the combinations
of all characters from a five letter word:
madam madma maadm maamd mamda mamad mdaam mdama mdaam mdama mdmaa mdmaa
maadm maamd madam madma mamad mamda mmada mmaad mmdaa mmdaa mmaad mmada
amdam amdma amadm amamd ammda ammad admam admma adamm adamm admma admam
aamdm aammd aadmm aadmm aammd aamdm ammda ammad amdma amdam amamd amadm
dmaam dmama dmaam dmama dmmaa dmmaa damam damma daamm daamm damma damam
damam damma daamm daamm damma damam dmmaa dmmaa dmama dmaam dmama dmaam
amadm amamd amdam amdma ammad ammda aamdm aammd aadmm aadmm aammd aamdm
admam admma adamm adamm admma admam ammad ammda amamd amadm amdma amdam
mmada mmaad mmdaa mmdaa mmaad mmada mamda mamad madma madam maamd maadm
mdmaa mdmaa mdama mdaam mdama mdaam mamad mamda maamd maadm madma madam

No. of words formed: 120

21.

```
main()
  {
    char s[10],t[10];
    printf("Enter a string:");
    gets(s);
    strcpy(t,s);
    printf(" Entered string is ");
    puts(s);
    palin(s);
    printf(" \nReversed string is ");
    puts(s);
    if(!strcmp(s,t))
      printf(" The given string is a palindrome\n");
    else
      printf(" The given string is not a palindrome\n");
  }
palin(char *s)
  {
   reverse(s,0,strlen(s));
  }
    reverse(char *s, int i, int len)
    {
      int ch,n;
```

```
        n= len - (i+l);
        if (i<n)
         {
          ch = s[i];
          s[i] = s[n];
          s[n] = ch;
          reverse(s,++i,len);
       }
     }
```

Output:
Enter a string:madam
Entered string is madam
Reversed string is madam
The given string is a palindrome

22.

Any 3 digit number x having identical digits when divided by sum of the 3 identical digits y always yield 37. Hence, x/37 yields y. The individual digit is obtained by dividing y by 3. Now the guessed 3 digit number can be written. Now the reader can write a suitable program to find the guessed number.

Similarly for 2 digit numbers (identical digits), x/y is 5.5, 4 digit numbers (identical digits), x/y is 277.75, and so on.

23.

```
main()
  {
    int x,y,n,s,tp,number,i,digit;
    printf("Enter the values of x,y and n:");
    scanf("%d%d%d",&x,&y,&n);
    s= x+y;tp=1;
  for(i=0;i<n;i++)
  tp *=10;
  number=s%tp;
  putchar('\n');
  for(i=0;i<n;i++)
    {
     tp /=10;
     digit=number/tp;
     number %= tp;
     printf("%c",'0'+digit);
     if(i<n-1)
      printf(" ");
    else
      printf("\n");
    }
  }
```

Output:
Enter the values of x,y and n:23456 43566 4
7 0 2 2

24.

```
main()
{
int i,j,k,n,count;
int a[] ={1,8,27,64,125,216,343,512,729,1000,1331,1728,2197,2744,3375};
int b[] ={1,8,27,64,125,216,343,512,729,1000,1331,1728,2197,2744,3375};
int c[15] [15],d[15] [15];
    for(i=0;i<15;i++)
      for(j=0;j<15;j++)
        {
         c[i][j]= a[i]+b[j];
         d[i] [j]=c[i][j];
        }
for(k=0;k<15;k++)
for(n=0;n<15;n++)
  {
    count = 0;
    for(i=0;i<15;i++)
     for(j=0;j<15;j++)
     {
      if(c[i][j]== d[k][n])
      count++;
    }
    if(count >= 4)
    printf("\ncube(%d)+cube(%d)= %d\n",k+l,n+l,d[k][n]);
  }
 }
```

Output:
cube(1)+cube(12)= 1729
cube(9)+cube(10)= 1729
cube(10)+cube(9)= 1729
cube(12)+cube(1)= 1729

25.

Generate the sequence of numbers (weights) 3^0, 3^1, 3^2, 3^3.

This will result in the sequence 1, 3, 9, 27. Only these four weights are required to measure up to 40 kilograms. The sum of all these weights yield 40. The weights less than 40 can be obtained as given below. Now

the reader can write the program to generate the sequence.

Required weight	Combination of weights
1	1
2	3–1
3	3
4	3+1
5	9–3–1
6	9–3
7	(9+1)–3
8	9-1
9	9
10	9+1
11	(9+3)–l
...	...
40	1+3+9+27

The subtracted weights are placed in the right pan and the added weights are placed in the left pan.

26.

Refer to **7.36** in the short answer type questions of Chapter 7.

27.

```
char *p[]={"man","goat","cabbage","wolf"};
main()
  {
    int'man = l,goat=2,cab=3,wolf=4;
    int source=l,destination=2;
    trip(wolf,source,destination);
  }
trip(int n,int s, int d)
  {
  if(n==1)return;
else
  {
    trip(n-l,s,d);
    move(n,s,d);
    if (n==4)
    n--;
    trip(n-l,d,s);
  }
 }
move(int n, int s, int d)
 {
```

```
 static int other=0;
 if(n==2)
 {
  if(other==0)
   {
    printf("Move %s and %s from %d to %d\n",p[0],p[n-1],s,d);
    printf("Move %s from %d to %d\n",p[0],d,s);
    other++;
   }
else if(other==1)
   {
  printf("Move %s and %s from %d to %d\n",p[0],p[n-1],s,d); other++;
   }
else   if(other==2)
   {
    printf("Move %s from %d to %d\n",p[0],s,d);
    printf("Move %s and %s from %d to %d\n",p[0],p[n-1],s,d);
   }
 }
if (n>2)
  printf("Move %s and %s from %d to %d\n",p[0],p[n-1],s,d);
}
```

Output:

Move man and goat from 1 to 2
Move man from 2 to 1
Move man and cabbage from 1 to 2
Move man and goat from 2 to 1
Move man and wolf from 1 to 2
Move man from 2 to 1
Move man and goat from 2 to 1

Appendix A

THE ASCII CHARACTER SET

ASCII Value	Character	ASCII Value	Character	ASCII Value	Character	ASCII Value	Character
000	NUL	018	DC20	036	$	054	6
001	SOH	019	DC3	037	%	055	7
002	STX	020	DC4	038	&	056	8
003	ETX	021	NAK	039	‘	057	9
004	EOT	022	SYN	040	(	058	:
005	ENQ	023	ETB	041	)	059	;
006	ACK	024	CAN	042	*	060	<
007	BEL	025	EM	043	+	061	=
008	BS	026	SUB	044	,	062	>
009	HT	027	ESC	045	–	063	?
010	LF	028	FS	046	.	064	@
011	VT	029	GS	047	/	065	A
012	FF	030	RS	048	0	066	B
013	CR	031	US	049	1	067	C
014	SO	032	BLANK	050	2	068	D
015	SI	033	!	051	3	069	E
016	DLE	034	"	052	4	070	F
017	DCI	035	#	053	5	071	G

ASCII Value	Character	ASCII Value	Character	ASCII Value	Character	ASCII Value	Character
072	H	086	V	100	d	114	r
073	I	087	W	101	e	115	s
074	J	088	X	102	f	116	t
075	K	089	Y	103	g	117	u
076	L	090	Z	104	h	118	v
077	M	091	[	105	i	119	w
078	N	092	\	106	j	120	x
079	O	093	]	107	k	121	y
080	P	094	^	108	l	122	z
081	Q	095	_	109	m	123	{
082	R	096	‘	110	n	124	|
083	S	097	a	111	o	125	}
084	T	098	b	112	p	126	~
085	U	099	c	113	q	127	DEL

Appendix B

PRECEDENCE AND ASSOCIATIVITY OF OPERATORS

Precedence level	Operator	Operation	Associativity
1	( )	Function call	Left to Right
	[ ]	Array subscript	Left to Right
	.	Dot	Left to Right
	–>	Arrow	Left to Right
2	!	Logical NOT	Right to Left
	~	One's complement	Right to Left
	–	Unary minus (negation)	Right to Left
	+ +	Increment	Right to Left
	– –	Decrement	Right to Left
	&	Address of	Right to Left
	*	Indirection	Right to Left
	(data_type)	Cast operator	Right to Left
	sizeof	Size of	Right to Left
3	*	Multiplication	Left to Right
	/	Division	Left to Right
	%	Modulus	Left to Right
4	+	Addition	Left to Right
	–	Subtraction	Left to Right

Precedence level	Operator	Operation	Associativity
5	<<	Left shift	Left to Right
	>>	Right shift	Left to Right
6	<	Less than	Left to Right
	<=	Less than or equal to	Left to Right
	>	Greater than	Left to Right
	>=	Greater than or equal to	Left to Right
7	==	Equal to	Left to Right
	!=	Not equal to	Left to Right
8	&	Bitwise AND	Left to Right
9	^	Bitwise XOR	Left to Right
10	|	Bitwise OR	Left to Right
11	&&	Logical AND	Left to Right
12	||	Logical OR	Left to Right
13	?:	Conditional	Right to Left
14	= += -= *= /= %= >>= <<= &= ^=|=	Simple and Compound Assignment	Right to Left
15	,	Comma	Left to Right

Note: Lower the precedence level number, higher the priority of evaluation.

Appendix

C

TIMINGS OF BASIC C OPERATIONS IN OUR HOST MACHINE

Basic Operation	Time in usec
clocks_per_ sec	1000000
(loop overhead	0.459
empty	0.001
comments	0.000
#define	0.000
declaration	0.000
array[]	0.065
*pointer	0.067
int =	0.011
empty func()	0.097
bit shift	0.017
if-then-else	0.013
int + int	0.016
int – int	0.016
int ^ int	0.017
int * int	0.066
int / int	0.300
(int) float	0.098
float + float	0.042
float * float	0.044

Basic Operation	Time in usec
float / float	0.261
strcpy()	0.453
strcmp()	0.279
rand()	0.126
sqrt()	0.701
malloc/free	1.674
fopen/fclose	79.534
system()	12292.224

Note: The host machine used to give the timings as shown in the above table is Pentium MMX 133 MHz. with 96 MB RAM and 6 GB HDD.

Appendix D

ANSI C LIBRARY FUNCTIONS

D1. ctype.h:

Return data type : int
Argument declaration : `int c;`
Return value : Non-zero (true) when c satisfies the condition described, else 0 (false)

The functions used to check whether c is

1. an alphabet or digit	**isalnum(c)**
2. an alphabet	**isalpha(c)**
3. a control character	**iscntrl(c)**
4. a decimal digit	**isdigit(c)**
5. a printing character except space	**isgraph(c)**
6. a lower case letter	**islower(c)**
7. a printing character including space	**isprint(c)**
8. a printing character except space or alphabet or digit	**ispunct(c)**
9. a space, form feed, new line, carriage return, tab, vertical tab	**isspace(c)**
10. an upper case letter	**isupper(c)**
11. a hexadecimal digit	**isxdigit(c)**

D2. string.h

Argument declarations:

```
int c, cl; char *s, *t ; const char cs, ct ;
void *sl, *tl; const void *csl, *ctl;
size_t n;/* size_t is an unsigned int returned by
            sizeof operator. */
```

Remark: c is converted to char; cl is converted to an unsigned char.

Return Type	Function Call	Purpose / Return Value
char *	strcpy(s,ct)	Copies string ct to string s including `\0'; return s.
char *	strncpy(s,ct,n)	Copies at most n characters of string ct to s; return s.
char *	strcat(s,ct)	Concatenates string ct to end of string s; return s.
char *	strncat(s,ct,n)	Concatenates at most n characters of string ct to string s, terminate with `\0'; return s.
int	strcmp(cs,ct)	Compares string cs to string ct; return a negative value if cs<ct, 0 if cs == ct, or positive value if cs>ct.
int	strncmp(cs,ct,n)	Compares at most n characters of string cs to string ct; return a negative value if cs<ct, 0 if cs == ct, or positive value if cs>ct.
char *	strchr(cs,c)	Returns pointer to first occurrence of c in cs or NULL if not present.
char *	strrchr(cs,c)	Returns pointer to last occurrence of c in cs or NULL if not present.
size_t	strspn(cs,ct)	Returns length of prefix of cs consisting of characters in ct.
size _t	strcspn(cs,ct)	Returns length of prefix of cs consisting of characters not in ct.
char *	strpbrk(cs,ct)	Returns pointer to first occurrence in string cs of any character of string ct, or NULL if none are present.
char *	strstr(cs,ct)	Returns pointer to first occurrence of string ct in Cs, or NULL if not present.
size _t	strlen(cs)	Returns length of cs.
char *	strerror(n)	Returns pointer to implementation-defined string corresponding to error n.
char *	strtok(s,ct)	Searches s for tokens delimited by characters from ct.
void *	memcpy(sl,ctl,n)	Copies n characters from ctl to sl, and return sl.
void *	memmove(sl,ctl,n)	Copies n characters from ctl to sl, and return sl. It works even if the objects overlap.
int	memcmp(csl,ctl,n)	Compares the first n characters of csl with ctl; return as with strcmp.
void *	memchr(csl,cl,n)	Returns pointer to first occurrence of character cl in cs 1 or NULL if not present among the first n characters.
void *	memset(sl,cl,n)	Places character cl into first n characters of sl, and return sl.

D3. math.h:

Return data type: double
Argument declarations:
```
int n ; int *exp ; double x, y ;
double *ip;
```

Function Call	Return Value
sin(x)	Sine of x, x in radians.
cos(x)	Cosine of x, x in radians.
tan(x)	Tangent of x, x in radians.
asin(x)	Arcsine of x in range $[-\pi/2, \pi/2]$, $x \epsilon [-1,1]$
acos(x)	Arccosine of x in range [0,], $x \epsilon [-1,1]$.
atan(x)	Arctangent of x in range$[-\pi/2, \pi/2]$.
atan2(y,x)	Arctangent of (y/x) in range$\{-\pi, \pi]$.
sinh(x)	Hyperbolic sine of x.
cosh(x)	Hyperbolic cosine of x.
tanh(x)	Hyperbolic tangent of x.
exp(x)	Exponential function e^x.
log(x)	Natural logarithm ln(x), x>0.
Function Call	Return Value
log 10(x)	Base 10 logarithm log 10(x), x>0.
pow(x,y)	x^y . A domain error if x==0 and y<=0 , or if x<0 and y is not an integer.
sqrt(x)	$\sqrt{x}$, x>=0.
ceil(x)	Smallest integer not less than x, as a double.
floor(x)	Largest integer not greater than x, as a double.
fabs(x)	Absolute value of x, i.e., [x].
ldexp(x,n)	$x.2n^n$
frexp(x,exp)	It splits x into a normalized fraction in the interval (1/2,1), which is returned, and a power of 2 which is stored in *exp. If x is 0, both parts of the result are zero.
modf(x,ip)	It splits x into integral and fractional parts, each with the same sign as x. It sores the integral part in *ip, and returns the fractional part.
fmod(x,y)	Floating point remainder of x/y, with the same sign as x. If y is zero the result is implementation defined.

D4. stdlib.h:

The header file stdlib.h consists of the following four types of functions.

1. Number conversion.
2. Random number generation.
3. Storage allocation.
4. Environment related function.

D4.1 Number Conversion Functions:

Argument declarations: `const char *s ; char **endp ; int base;`

Return Type	Function Call	Purpose / Return Value
double	atof(s)	Converts s to double.
int	atoi(s)	Converts s to int.
long	atol(s)	Converts s to long.
double	strtod(s,endp)	Converts the prefix of s to double, ignoring leading white spaces.
long	strtol(s,endp,base)	Converts the prefix of s to long, ignoring leading white spaces; it sores a pointer to any unconverted suffix in *endp unless endp in NULL. If base is between 2 and 36, conversion is done assuming that the input is written in that base. If base is 0, the base is 8, 10 or 16. Leading 0 implies octal and leading 0x or OX implies hexadecimal. Alphabets in either case represent digit from 10 to base - 1. A leading 0x or 0X is permitted in base 16. If the answer overflows, LONG _MAX or LONG _MIN is returned, depending on the sign of the result, and errno is set to ERANGE.
unsigned long	strtoul (s,endp,base)	It is the same as strtol except that the result is unsigned long and the error value is ULONG_MAX.

D4.2 Random Number Generation Functions:

Argument declarations: `unsigned int seed;`

Return Type	Function Call	Purpose / Return Value
int	rand(void)	It returns a pseudo random integer in the range 0 to RAND_ MAX, which is at least 32767.
void	srand(seed)	It uses seed as the seed for a new sequence of pseudo random numbers. The initial seed is 1.

D4.3 Storage Allocation Functions:

Argument declarations: `size_t nobj, size ; void *p;`

Return Type	Function Call	Purpose / Return Value
void *	calloc(nobj, size)	It returns a pointer to space for an array of nobj objects, each of size size, or NULL if the request cannot be satisfied. The space is initialized to 0 bytes.
void *	malloc(size)	It returns a pointer to space for an object of size size, or NULL, if the request cannot be satisfied. The space is uninitialized.
void *	realloc(p, size)	It changes the size of the object pointed to by p to size. The condense will be unchanged up to the minimum of old and new sizes. If the new size is larger, the new space is uninitialized. realloc returns a pointer to the new space, or NULL, if the request cannot be satisfied, in which case *p is unchanged.
void	free(p)	It deallocates the space pointed to by p. It does nothing if p is NULL. p must be a pointer to space previously allocated by calloc, malloc, or realloc.

D4.4 Environment-related Functions:

Argument declarations:
```
int status, n, num, denom;
long n, num, denom;
const char *s; const char *name ;
const void *key ; const void ;
const base ; size_t n, size ;
int (*cmp) (const void *keyval, const void *datum);
int (*cmpl) (const void *, const void *) ;
void (*fcn) (void) ;
```

Return Type	Function Call	Purpose / Return Value
void	abort()	It causes the program to terminate up normally.
void	exit(status)	It causes normal program termination.
int	atexit(fcn)	It registers the function fcn to be called when the program terminates normally. It returns non-zero if the registration cannot be made.
int	system(s)	It passes the string s to the environment for execution. If s is NULL, system returns non-zero. If s is not NULL, the return value is implementation dependent.

Return Type	Function Call	Purpose / Return Value
char*	getenv(name)	It returns the environment string associated with name, or NULL if no string exist.
void *	bsearch(key, base, n, size, cmp 1)	It searches base[0] ... base[n – 1] for an item that matches *key. The function c and p must return negative if its first argument is less than its second, 0 if equal, and positive if greater. Items in the array base must be in ascending order. bsearch returns a pointer to a matching item, or NULL if none exists.
void	qsort(base, n, size,(cmp 1)	It sorts into ascending order an array base[0] ... base[n – 1] of object of size size. The comparison function cmp is as in bsearch.
int	abs(n)	It returns the absolute value of its int argument.
long	labs(n)	It returns the absolute value of its long argument.
div_ t	div(num, denom)	It computes the quotient and remainder of num/denom. The results are stored in the int members quot and rem of a structure of type div_t.
Idiv_t	ldiv(num, denom)	It computes the quotient and remainder of num/denom. The results are stored in the long members quot and rem of a structure of type ldiv _t,

D5. assert.h:

It is used to add diagnostics to programs.

Argument declarations: `int expression ;`

Return Type	Function Call	Purpose / Return Value
void	assert(expression)	If expression is zero the assert macro will print error message on stderr. It then calls abort to terminate execution.

The constants defined for the sizes of integral data types are given below.

D6. limits.h:

Constant name	Value	Explanation
CHAR _BIT	8	Bits in a char.
CHAR _MAX	UCHAR_MAX or SCHAR_ MAX	Maximum value of char.

Constant name	Value	Explanation
CHAR _MIN	0 or SCHAR_ MIN	Minimum value of char.
INT _MAX	+32767	Maximum value of int.
INT _MIN	–32767	Minimum value of int.
LONG_ MAX	+2147483647	Maximum value of long.
LONG_MIN	–2147483647	Minimum value of long.
SCHAR_MAX	+127	Maximum value of signed char.
SCHAR_ MIN	–127	Minimum value of signed char.
SHRT_MAX	+32767	Maximum value of short.
SHRT_MIN	–32767	Minimum value of short.
UCHAR_ MAX	255	Maximum value of unsigned char.
UINT _MAX	65535	Maximum value of unsigned int.
ULONG_MAX	4294967295	Maximum value of signed long.
USHRT _MAX	65535	Maximum value of unsigned short.

D7. floats.h:

Constant name	Value	Explanation
FLT _RADIX	2	Radix of exponent representation, e.g. 2, 16.
FLT _ROUNDS		Floating point rounding mode for addition.
FLT _DIG	6	Decimal digits of precision.
FLT_ EPSILON	lE-5	Smallest number x such that 1.0 + x ≠ 1.0.
FLT_ MANT_ DIG		Number of base FLT _RADIX digits in mantissa.
FLT_ MAX	1E+37	Maximum floating point number.
FLT_MAX_EXP		Maximum n such that FLT_RADIX^n –1 is representable.
FLT _MIN	1E-37	Minimum normalized floating point number.
FLT _MIN_ EXP		Minimum n such that 10^n is a normalized number.
DBL _DIG	10	Decimal digits of precision.
DBL _EPSILON	1 E –9	Smallest number x such that 1.0 + x ≠ 1.0.
DBL _MANT DIG		Number of base FLT_ RADIX digits in mantissa.
DBL _MAX	1E+37	Maximum double floating point number.
DBL_ MAX_EXP		Maximum n such that FLT _RADIX^n –1 is representable.
DBL_ MIN	I E-37	Minimum normalized, double floating point number.
DBL _MIN_ EXP		Minimum n such that 10^n is a normalized number.

D8. time.h:

Argument declarations:

```
time _t *tp, time1, time2;
const time _t*tp0; struct tm *tp1;
const struct tm *tp2;
const char*fmt ;
char *s; size_t smax;
```

Return Type	Function Call	Purpose / Return Value
clock_t	clock()	It returns the processor time used by the program.
Return Type	Function Call	Purpose / Return Value
time _t	time(tp)	It returns a current calendar time or –1 if the time is not available. If tp is not NULL, the return value is also assigned to *tp.
double	difftime(time 2, time 1)	It returns time2 – time1 expressed in seconds.
time _t	mktime(tp 1)	It converts the local time in the structure *tp 1 into calendar time in the same representation used by time. The components will have values in the ranges shown. mktime returns the calendar time or –1 if it cannot be represented.
char *	asctime(tp2)	It converts the time in the structure *tp2 into a string of the form Sun Jan 3 15: 14: 13 1988\n\0 ‘
char *	ctime(tp0)	It converts the calendar time *tp0 to local time. It is equivalent to asctime(localtime(tp0))
struct tm *	gmtime(tp0)	It converts the calendar time *tp0 into Coordinated Universal Time (UTC). It returns NULL if UTC is not available. The name gmtime has historical significance.
struct tm *	localtime(tp0)	It converts the calendar time *tp0 into local time.
size_c	strftime(s, max, fmt, tp2)	It formats date and time information from *tp2 into s according to fmt , which is analogous to a printf format. Ordinary characters including \0 are copied into s. No more than smax characters are placed into s. strftime returns the number of characters, excluding the \0, are 0 if more than smax characters were produced.

D9. stdio.h:

This header file consists of the following:

1. File operations.
2. Formatted input and output functions.
3. Character input and output functions.
4. Direct input and output functions.
5. File positioning functions.
6. Error functions.

D9.1. File Operations:

Argument declarations: `const char *filename ; const char *mode ;`

The mode may be

"r" open text file for reading.
"w" create text file for writing; discard previous contents, if any.
"a" append; open or create text file for writing at end of file.
"r+" open text file for update (i.e., reading and writing).
"w+" create text file for update; discard previous contents, if any.
"a+" append; open or create text file for update, writing at end.
"rb", "wb", "ab", "r+b", "w+b", "a+b", The letter b indicates a binary file.

```
FILE *stream;
const char *oldname, const char *newname;
char *buf; int mode; size_t size;
char s[L_tmpnam]
```

Return Type	Function Call	Purpose / Return Value
FILE *	fopen(filename, mode)	It opens the named file, and returns a stream, or NULL if the attempt fails.
FILE *	freopen(filename, mode, stream)	It opens the file with the specified mode and associates the stream with it. It returns stream, or NULL if an error occurs.
int	fflush(stream)	On an output stream, fflush causes any buffered but unwritten data to be written. On an input stream, the effect is undefined. It returns EOF for a write error, and zero otherwise. fflush(NULL) flushes all output streams.
int	fclose(stream)	It flushes any unwritten data for stream, discards any unread buffered input, frees any automatically allocated buffer, then closes the stream. It returns EOF if any errors occurred and zero otherwise.

Return Type	Function Call	Purpose / Return Value
int	remove(filename)	It removes the named file. It returns non-zero if the attempt fails.
int	rename(oldname, newname)	It changes the name of a file. It returns non-zero if the attempt fails.
FILE *	tmpfile()	It creates a temporary file of mode "wb+" that will be automatically removed when closed or when the program terminates normally. It returns a stream, or NULL if it could not create the file.
char *	tmpnam(s)	tmpnam(NULL) creates a stream that is not a name of a existing file, and returns a pointer to an internal static array. tmpnam(s) stores the stream in s as well as returning it as the function value. s must have room for at least L_tmpnam characters. tmpnam generates a different name each time it is called. At most TMP _MAX different names are possible during execution of the program. Observe that tmpnam creates a name not a file.
int	setvbuf(stream, buf, mode, size)	It controls buffering for the stream. It must be called before reading, writing, or any other operation. A mode of _IOFBF causes full buffer-ng, _IOLBF line buffering of text files, and _IONBF no buffering. If buf is not NULL, it will be used as a buffer; otherwise the buffer will be allocated. size determines the buffer size. setvbuf returns non- zero for any error.
void	setbuf(stream, buf)	If buf is NULL, buffering is turned off for the stream. Otherwise, setbuf is equivalent to (void) setvbuf(stream, buf_ IOFBF, BUFSIZ).

D9.2 Formatted Output:

The printf functions provide formatted output conversion.

Argument declarations:

```
FILE *stream ; const char *format ;
char *s ; va_list arg ;
```

va_ list is a predefined data type, which is a pointer to the argument list.

Return Type	Function Call	Purpose / Return Value
int	fprintf(stream, format, ...)	It converts and writes output to stream under the control of format. The return value is the number of characters written, or negative if an error occurred.
int	printf(format, ...)	It is equivalent to fprintf(stdout, ...).
int	sprintf(s,format, ...)	It is the same as printf except that the output is written into the string s, terminated with'\()'. s must be big enough to hold the result. The return count does not include the `\0'.
int	vprintf(format, arg)	It is equivalent to the printf function, except that the variable argument list is replaced by arg, which has been initialized by the va_ start macro and perhaps va _arg calls.
int	vfprintf(stream, format, arg)	- do -
int	vsprintf(s, format, arg)	- do -

Formatted Input:

The scanf function deal with formatted input conversion.

Argument declarations: `FILE *stream ; const char *format; char *s;`

Return Type	Function Call	Purpose / Return Value
int	fscanf(stream, format, ...)	It reads from stream under the control of format, and assigns converted values through subsequent argument, each of which must be a pointer. It returns when format is exhausted. fscanf returns EOF if end of file or an error occurs before any conversion; otherwise, it returns number of input items converted and assigned.
int	scanf(format, ...)	It is identical to fscanf(stdin, ...)
int	sscanf(s, format, ...)	It is equivalent to scanf(...) except that the input characters are taken from the string s.

D9.3 Character Input and Output Functions:

Argument declarations: `FILE *stream; char *s; int n, c;`

Return Type	Function Call	Purpose / Return Value
int	fgetc(stream)	It returns the next character of stream as an unsigned char(converted to an int), or EOF if end of file or error occurs.
char *	fgets(s, n, stream)	It reads at most the next n – 1 characters into the array s, stopping if a new line is encountered. The new line is included in the array, which is terminated by '\0'. fgets returns s, or NULL if end of file or error occurs.
int	fputc(c, stream)	It writes the character c (converted to an unsigned char) on stream. It returns the character written, or EOF for error.
int	fputs(s, stream)	It writes the stream s (which need not contain '\n') on stream. It returns non-negative, or EOF for an error.
int	getc(stream)	It is equivalent to fgetc except that if it is a macro, it may evaluate the stream more than once.
int	getchar()	It is equivalent to getc(stdin).
char *	gets(s)	It reads the next input line into the array s. It replaces the terminating new line with '\0'. It returns s, or NULL if end of file or error occurs.
int	putc(c, stream)	It is equivalent to fputc except that if it is a macro, it may evaluate stream more than once.
int	putchar(c)	It is equivalent to putc(c, stdout).
int	puts(s)	It writes a stream s and a new line to stdout. It returns EOF if an error occurs, non-negative otherwise.
int	ungetc(c, stream)	It pushes c (converted to a unsigned char) back on to stream, where it will be returned on the next read. Only one character of pushback per stream is guaranteed. EOF may not be pushed back. ungetc returns the character pushed back, or EOF for error.

D9.4 Direct Input and Output Character:

Argument declarations:

```
void *ptr; size_t size, nobj; FILE *stream;
const void *ptr;
```

Return Type	Function Call	Purpose / Return Value
size_t	fread(ptr, size, nobj, stream)	It reads from stream into the array ptr at most nobj objects of size size. It returns the number of objects read. This may be less than the number requested. feof and ferror must be used to determine status.
size_t	fwrite(ptr, size, nobj, stream)	It writes, from the array ptr, nobj objects of size size on stream. It returns the number of objects written, which is less than nobj on error.

D9.5. File Positioning Functions:

Argument declarations: `int origin ; long offset ; FILE *stream ;`
`fpos_t *ptr ; const fpos_t *ptr ;`

Return Type	Function Call	Purpose / Return Value
int	fseek(stream, offset, origin)	It sets the file position for stream. A subsequent read or write will access a data beginning at the new position. For a binary file, the position is set to offset characters from origin, which may be SEEK _SET (beginning), SEEK _CUR (current position), or SEEK_ END (end of file). For a text stream, offset must be zero, or a value returned by ftell. fseek returns non-zero on error.
long	ftell(stream)	It returns the current file position for stream, or –I L on error.
void	rewind(stream)	rewind(fp) is equivalent fseek (fp, OL, SEEK_SET); clearerr(fp).
int	fgetpos(stream, ptr)	It records the current position in stream in *ptr, for subsequent use by fsetpos. The type fpos _t is suitable for recording such values. fgetpos returns non-zero on error.
int	fsetpos(stream, ptr)	It positions stream at the position recorded by fgetpos in *ptr. It returns non-zero on error.

D9.6 Error Functions:

Argument declarations: `const char *s; FILE *stream;`

Return Type	Function Call	Purpose / Return Value
void	clearerr(stream)	It clears the end of file and error indicators for stream.
int	feof(stream)	It returns non-zero if end of file indicator for stream is set.
int	ferror(stream)	It returns non-zero if the error indicator for stream is set.
void	perror(s)	It prints s and an implementation defined error message corresponding to the integer in errno, as if by fprintf(stderr, "%s: %s\n", s, "error message").