DATA STRUCTURES USING C
1000 Problems and Solutions

About the Author

Sudipta Mukherjee was born in 1980. He is an Electronics and Communication Engineer and is presently working with Tata Consultancy Services (TCS) Limited as a software developer. Besides his routine consultancy job, Mr. Mukherjee had taught in the global training centre of TCS and was awarded "Best Faculty" for the year for teaching Good Programming Skills, C, UNIX and Java. From his childhood he had a great passion for computers and wished to do something new apart from general academics. In 2002, he got associated with Kasamba Inc. as an outsourcing programmer which has a large clientele around the world. He did his certification in C, C++, and UNIX from Ramakrishna Mission, with affiliation from Chicago University. In 2003, he wrote a solutions manual for the book *Object Oriented Programming with C++*, second edition by E. Balagurusamy. This was his debut effort as a young author. This material was much appreciated and accepted by Tata McGraw-Hill Publishing Company. Mr. Sudipta Mukherjee is a member of IEEE (Institute of Electrical and Electronics Engineers). Mr. Mukherjee's research interest includes Data Mining, Automatic Text Processing, Tool Development and Artificial Intelligence.

DATA STRUCTURES USING C
1000 Problems and Solutions

Sudipta Mukherjee
Software Developer
Tata Consultancy Services
India

Tata McGraw-Hill Publishing Company Limited
NEW DELHI

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

Tata McGraw-Hill

Published by the Tata McGraw-Hill Publishing Company Limited,
7 West Patel Nagar, New Delhi 110 008.

ISBN (13): 978-0-07-066765-5
ISBN (10): 0-07-066765-9

Managing Director: *Ajay Shukla*

General Manager: Publishing—SEM & Tech Ed: *Vibha Mahajan*
Asst. Sponsoring Editor: *Shalini Jha*
Editorial Executive: *Nilanjan Chakravarty*
Executive—Editorial Services: *Sohini Mukherjee*
Senior Production Executive: *Anjali Razdan*

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

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

Typeset at The Composers, 260, C.A. Apt., Paschim Vihar, New Delhi 110 063 and printed at
Adarsh Printers, C-50-51, Mohan Park, Naveen Shahdara, Delhi 110 032

Cover Printer: Rashtriya Printers

RALQCRDXRBCZR

Dedicated
To
My beloved parents, Dipali Mukherjee and Subrata Mukherjee
My uncle Tushar Mukherjee
and
My mentor Eknath Das

Preface

Data are the blood of any application that we develop. A data structure is like the viscera that pumps and carries that blood. In other words, data structure is the heart of any application we design. Selection of a data structure can drastically enhance the performance of the system.

The main motivation behind this book is to show the application of several data structures for different diverse problems. Matrix multiplication can be used to calculate the total revenues of a chain of departmental stores. In other data structure books we find the logic of matrix multiplication but their applications are not available. Moreover, data structure is taught as a stand-alone subject devoid of application of other fields of science. In this book the relationship between data structure and other fields of science like numerical methods, applied statistics, physics, etc., are shown with a variety of problems. While I studied data structure, there was no such book available which covered all the basic algorithms of different data structures and their applications. So I decided to write one for future students of data structures.

This book is mainly written for the students of BE, BSc (Comp Sc.), BSc (IT), BCA, and MCA but it can be used by students of any data structure course because it covers a variety of data structures and teaches its readers how to create their own data structures to serve a specific need. Students who are preparing for GATE examination on Computer Science will also find this book useful. Undergraduate students who know about C arrays and structures can skip chapters 1 and 2. Other than this exception, the chapters are arranged in an order in which they should be read.

"Before we start" discusses what a data structure is and why it is needed. This section gives the reader an overview of a data structure and its importance in programming different applications.

Chapter 1 discusses arrays, starting from declaring an array to how arrays can be used in console based animation programs.

Chapter 2 discusses structures that is the basic building block of all the data structures.

Chapter 3 discusses all kinds of linked lists. Linked list is the most basic pointer-based data structure and is the building block of other different data structures.

Chapter 4 discusses string handling. This chapter covers different string processing functions and their applications in solving some interesting problems like "UPC product code verification", "Credit card number verification", etc.

Chapter 5 discusses recursion, a powerful programming technique. This chapter shows how recursion can be useful in diverse programmatic situations starting from a very common example of generating Fibonacci numbers to solving nonlinear equations using recursive root finding methods and recursive creation of fractals like Koch Snowflake.

Chapter 6 discusses stack data structure. Besides describing the usual push, pop algorithms, in this chapter a new data structure, MTF list, has been introduced and modeled using stacks. A tray of stack is also modeled under the name of saguaro stacks. Stacks are used greatly in parsers. An XML parser is written to introduce the capacity which this simple data structure has to offer.

Chapter 7 discusses Queue data structure. In this chapter it has been shown how real-time queues can be simulated using queues. MTF is also modeled using queues.

Chapter 8 discusses Tree data structure. In this chapter different types of tree data structure and their methods have been discussed. Trees are very important data structures and find applications in lot of problems. A diverse set of problems have been identified and presented in order to showcase this immense capacity of tree data structures.

Chapter 9 discusses Graph data structure. Graph is probably the most complicated data structure and has application in almost all fields of science and technology. In this chapter the basic graph theory algorithms are implemented in order to get the reader interested in this subject. Graph theory is so massive that complete coverage of the algorithms available till date are out of scope of this book and can easily be accommodated as content of another title.

Chapter 10 discusses sorting algorithms. The chapter starts with a broader classification of algorithms and then implementation of the algorithms with their time and space complexities shown on tabular and graphical ways. There are good comparison graphs that show which algorithm is better. Apart from these, the chapter has a list of problems where sorting is the key to solve.

Chapter 11 discusses hashing techniques. Broad classification of hashing algorithms have been classified and implemented in full-length programs. Besides this, it has been shown how hashing can be used in computer security softwares.

Chapter 12 discusses ADT. This chapter shows how to create a new ADT, what are the different types of methods an ADT can have and how to distinguish them, etc. This chapter must be read before the chapters 13, 14 and 15.

Chapter 13 discusses Date data structure. This chapter shows how to create different functions that deal with dates. Knowledge of these will enable the reader to implement date-involved calculations in different applications.

Chapter 14 discusses Map data structure. A map is basically a hash table that is nothing but a collection of key-value pairs. This chapter shows how a simple map can be used in design of a phone book, a dictionary and a Random Ciphering Machine (RCM).

Chapter 15 discusses the Currency data structure. A Currency can also be represented as a double. But that doesn't convey the intention of the programmer. Thus the code becomes less readable and in turn, less maintainable. So in order to create a more programmer-friendly code, a separate data type for currency is needed. In this chapter it has been shown how different currencies can be modeled using C structures and several methods are also defined that operate on this data structure.

Chapter 16 discusses File Handling in C. This chapter mainly shows how to save and retrieve data from permanent storage files. Before reading this chapter, reading the chapter on string is mandatory.

Appendix A: Project Ideas. This section gives some exciting and innovative ideas to the reader to implement.

Appendix B: Bibliography, for further reading.

I am open to constructive criticism, comments and suggestion for the improvement of this book. Please let me know if you find anything worth reporting.

O'Fallon, MO

SUDIPTA MUKHERJEE

Acknowledgements

During the writing of this book, I have been helped by many people in innumerable ways, who provided constructive criticism, spontaneous encouragement and shared responsibilities. In this section I would like to take the privilege to thank them all. First, I want to thank **Mr. Subhabrata Chakrabarty**, Assistant General Manager (Eastern India), TMH, for encouraging me to work hard toward the completion of this book. This book would not have taken birth at all without the immense help and support that I got from my editors **Ms. Shalini Jha** and **Mr. Nilanjan Chakravarty.** During this project my family has been very supportive as always. I would like to take this opportunity to thank the following reviewers who took out some of their valuable time to read this script and offer valuable suggestions.

Dr. M.P. Sebastian	National Institute of Technology, Department of Computer Science and Engineering, Calicut
Mr. Amit Jain	Radha Govind Engineering College (UPTU), Anuyogipuram, Meerut.
Prof. Shashi Mogalla	Professor, Department of Computer Science and Engineering, College of Engineering, Andhra University, Visakhapatnam.
Mr. U. A. Deshpande	Department of Electronics and Computer Science, Vishveshwaraya National Institute of Technology, Nagpur.
Mr. S. D. Deshpande	Assistant Professor, Computer Science and Engineering Department, Shri Sant Gajanan Maharaj College of Engineering, Maharashtra
Mr. Sanjay Goswami	Lecturer, Department of Computer Applications, Narula Institute of Technology, Agarpara, Kolkata.

Words will fall short to explain how my parents **Mr Subrata Mukherjee** and **Mrs. Dipali Mukherjee** always kept my moral high. My special thanks go to **Mr. Tushar Mukherjee**, my youngest uncle, who bought me a desktop computer while I was in college and most of my passion today for programming is due to the time I spent on that system. Special thanks to my colleagues and seniors in TCS-ILP training centre at Bhubaneswar and Trivandrum, to **Suman Bhattacharya, Meera Sidhardhan, Jayanthi K.P., Sumit Bose** and **Ashish Ghosh** for their interest in this project and encouragement.

Last but not the least, I want to thank my friends **Anindya Ghosh**, **Vinit Sharma** and **Supratim Chatterjee** here in Mystic Cove, for encouraging me to pursue this project and their spontaneous feedback and homely concern. Thanks to you all!

O'Fallon, MO

SUDIPTA MUKHERJEE

Contents

Preface *xix*
Acknowledgements *xxi*

1 ARRAY—*EASY, CONTIGUOUS, ELEGANT!* **1**

Introduction *1*
1.1 How to Initialize an Array *1*
1.2 How to Traverse an 1D Array using Index *2*
1.3 How to Manipulate Elements of the Array *3*
1.4 How to Add Array Elements in a Specific Region *4*
1.5 How to Add Elements in the Odd and Even Places in the Array *4*
1.6 How to Perform Operations Involving External Variables *5*
1.7 How to Find Function Values *7*
1.8 How to Solve a Demographical Application, a Problem of Vital Statistics *7*
1.9 Where to Apply 3D Arrays *8*
1.10 How to Delete a Particular Item from an Array *9*
1.11 How to Delete an Item from a Particular Location *10*
1.12 How to Find the Maximum Number in an Array *11*
1.13 How to Find the Minimum Number in an Array *12*
1.14 How to Sort the Array Alphabetically *12*
1.15 How to Check If a String is a Palindrome or not *13*
1.16 How to Search for an Array Element *14*
1.17 How to Make the Array Elements Unique *14*
1.18 How to Find the Mean of the Array Elements *16*
1.19 How to Find Weighted Average of an Array of Numbers *16*
1.20 How to Find the Median of the Array Elements which are Already Sorted *17*
1.21 How to Find the Mode of the Array Elements *17*
1.22 How to Find the Range of the Array Elements *18*
1.23 How to Find Standard Deviation of an Array *18*
1.24 How to Find the Variance of the Array Elements *19*
1.25 How to Find an Interpolated Value using Newton's Forward Difference Interpolation *20*
1.26 How to Interpolate using Lagrange's Interpolation Formula *21*
1.27 How to Find a Regression Line on *X* or *Y* *23*
1.28 How to Find Simple Aggregation Index Number *24*
1.29 How to Find the Simple Average of a Price-Relative Index *25*
1.30 How to Find Laspeyre's Index Number *26*
1.31 How to Find Paasche's Index Number *26*
1.32 How to Find Bowley's Index Number *26*
1.33 How to Find Fisher's Index Number *27*
1.34 How to Find Marshall–Edward Index Number *27*
1.35 How to Represent a Matrix Using 2D Arrays *27*

1.36 How to Add Two 3 × 3 Matrices *27*
1.37 How to Subtract Two 3 × 3 Matrices *28*
1.38 How to Multiply Two Matrices *28*
1.39 How to Calculate Revenues using Matrix Multiplication *30*
1.40 Multiplication of Two 2 × 2 Matrices using Strassen's Algorithm which uses 7 Multiplications and 18 Additions *30*
1.41 How to Find the Hadamard Product of Two Matrices *31*
1.42 How to Find the Kronecker Product of Two Matrices *32*
1.43 How to Find the Transpose of a Matrix *34*
1.44 How to Find the Inverse of a Square Matrix *34*
1.45 How to Find the Upper Triangular of a Matrix *35*
1.46 How to Find a Strict Upper Triangular Matrix *36*
1.47 How to Find the Lower Triangular of a Matrix *37*
1.48 How to Find a Strict Lower Triangular Matrix *37*
1.49 How to Create a Toeplitz Matrix from a given Row and Column *38*
1.50 How to Find whether a Matrix is Symmetric or not *39*
1.51 Representing a Sparse Matrix as Arrays *39*
1.52 3D Array Applications How to use a 3D Array to Store and Manipulate the Literacy Details of 5 Cities around a Year *42*
1.53 How to Return More than One Value from a Function *43*
1.54 How to Clone String Tokenizer Class of Java *44*
1.55 Conversion of Binary to Decimal *44*
1.56 How to Design a Chart for Share Trading *45*
1.57 How to Find HHI Index *46*
1.58 How to Find GINI Coefficient Measurement for a City *47*
1.59 How to Find whether Three given Numbers are in AP, GP or HP *48*
1.60 Animation of Different Signaling Formats *49*
1.61 A Well-known Cryptographic Technique—Cipher Text *54*
1.62 Decoder Program for the above Encrypter *55*
1.63 How to Find the Histogram of a 256 Gray Scale Image *55*
1.64 How to Convert a Gray Scale Image to Binary Image/Negative Image *56*
Revision of Concepts *57*
Review Questions *58*
Programming Problems *59*

2 STRUCTURES—*THE BUILDING BLOCKS* **61**

Introduction *61*
2.1 Use of `typedef` *61*
2.2 Accessing the Structure Elements *62*
2.3 Some Built-in useful Structures in Turbo C (under DOS) *63*
2.4 How to Define a Structure that Represents a Point in 3D *63*
2.5 How to Find the Centroid of a Polygon using Point Structure *64*
2.6 How to Find the Distance between Two Points in 3D *64*
2.7 How to Find the Area of Any Regular Polygon *65*
2.8 How to Test Collinearity for Three Points *65*
2.9 How to Check IF a Triangle is Equilateral *66*
2.10 How to Check IF a Triangle is Isosceles *66*
2.11 How to Model a Triangle using Point Structure? *66*
2.12 How to Check IF a Triangle is Right Angled *67*
2.13 How to Find whether a Triangle is Equilateral or not *67*
2.14 How to Model a Tetrahedron using Triangles *68*

2.15 How to Model a Rectangle using `Struct` and `Enum` *68*
2.16 How to Model a Trapezium using Point *69*
2.17 How to Check whether a Trapezium is Equilateral or not *69*
2.18 How to Find whether a Point is within a Triangle or not *70*
2.19 How to Find whether a Point is within a Rectangle or not *71*
2.20 How to Find whether a Point is within a Circle or not *71*
2.21 How to Find whether Two Circles are Touching Internally or not *71*
2.22 How to Find whether Two Circles are Touching Externally or not *72*
2.23 How to Model a Straight Line in Slope Format *72*
2.24 How to Model a Straight Line in *XY* Intercept Format *72*
2.25 How to Convert an *XY* Intercept form Line to Slope Format Line *73*
2.26 How to Convert a Slope Line Format to *XY* Intercept Format *73*
2.27 How to Find whether Two Lines are Parallel or not *73*
2.28 How to Find the Point of Intersection of Two Straight Lines *73*
2.29 How to Find the Tangent on any Point on a Circle *73*
2.30 How to Model a Parabola using a Straight Line and Point *74*
2.31 How to Find the Tangent on any Point on a Parabola *74*
2.32 How to Find the Normal on any Point on a Parabola *74*
2.33 How to Model an Ellipse *74*
2.34 How to Find the Area of an Ellipse *75*
2.35 How to Find the Tangent at any Point of an Ellipse *75*
2.36 How to Find the Normal at any Point of an Ellipse *75*
2.37 How to Model a Prism using Structure *75*
2.38 How to Model a Circular Cylinder *76*
2.39 How to Find the Surface Area of a Cylinder *76*
2.40 How to Model a Cone *77*
2.41 How to Find the Area of a Cone *77*
2.42 How to Find the Volume of the Cylinder Defined by a Circle and Point *77*
2.43 How to Find the Area of the Prism *78*
2.44 How to Find out whether a Point is within an Ellipse or not *78*
2.45 How to Find out whether a Point is within a Hyperbola or not, Assume that the Major or Minor Axes are Given *78*
2.46 How to Model a Rhombus *79*
2.47 How to Find the Area of a Rhombus *79*
2.48 How to Model Vectors as Structure *79*
2.49 How to Write a Function to Add Vectors *79*
2.50 How to Find the Weighted Sum of Vectors *80*
2.51 How to Find IF the Weighted Sum of Vectors is an *Affine Summation* or not *80*
2.52 How to Write a Function to Find DOT Product of Two Vectors *81*
2.53 How to Write a Function to Find Cross Product of Two Vectors *81*
2.54 How to Write a Function for Scalar Multiplication of a Vector *82*
2.55 How to Find Dot Product of Three Vectors *82*
2.56 How to Find whether Three Vectors are Coplanar or not *82*
2.57 How to Find the Cross Product of Three Vectors *82*
2.58 How to Find the Scalar Product of Four Vectors *83*
2.59 How to Find the Vector Product of Four Vectors *83*
2.60 How to Model a Complex Number as a Structure *83*
2.61 How to do Conversion from Polar to Rectangular Form and vice versa *84*
2.62 How to Add Complex Numbers *84*
2.63 How to Subtract One Complex Number from another *85*

2.64 How to Multiply Two Complex Numbers *85*
2.65 Proving De Moivre's Theorem using Polar Complex Structure *86*
2.66 How to Write a Phonebook Simulation Program using Structure *86*
2.67 How to Model a Bank Account as a Combination of Structures *89*
2.68 How to Write a POS (Point of Sale) Simulation using Structure *91*
Revision of Concepts 102
Review Questions 103
Programming Problems 103

3 LINKED LIST—*SCATTERED YET LINKED!* **105**

Introduction *105*
3.1 Single Linked List *106*
3.2 Double Linked List *107*
3.3 Circular Linked List *108*
3.4 What do you Mean by Array of Linked Lists? *108*
3.5 Linked List in C and Predictors *109*
3.6 Linked List Function Philosophy of this Chapter *109*
3.7 How to Insert a Node at the end of a Single Linked List—The Node may be the First Node of the Linked List *111*
3.8 How to Insert a Node at the Front of the Single Linked List *111*
3.9 How to Find the Front Element of the Single Linked List *112*
3.10 How to Find the Back Element of the Single Linked List *112*
3.11 How to Traverse the Single Linked List *113*
3.12 How to Count the Number of Nodes in a Single Linked List *113*
3.13 How to Find the Frequency of an Item in a Single Linked List *113*
3.14 How to Search a Particular Item in the Single Linked List *114*
3.15 How to Find the Address of a Particular Node in a Single Linked List *114*
3.16 How to Insert Nodes at a Particular Location in a Single Linked List *115*
3.17 How to Insert Nodes before a Particular Node in a Single Linked List *116*
3.18 How to Display all the Contents of a Single Linked List *116*
3.19 How to Find the Maximum Element from a Single Linked List *116*
3.20 How to Find the Minimum Element from a Single Linked List *117*
3.21 How to Edit the Content of a Particular Node with a Given Value *117*
3.22 How to Write a Function to Merge Two Linked Lists *118*
3.23 How to Write a Function to Insert a List within Another List *119*
3.24 How to Swap the Head and Tail Node (i.e. The First and the Last Node) of the Single Linked List *120*
3.25 How to Swap the Contents of Any Two Other Nodes apart from Head and Tail *121*
3.26 How to Delete a Particular Node given by an Index Number *121*
3.27 How to Delete a Range of Elements from a List *122*
3.28 How to Delete Alternate Elements from a Single Linked List *122*
3.29 How to Make the List Entries Unique *123*
3.30 How to Delete the First Element of the List *123*
3.31 How to Delete the Last Element of the List *124*
3.32 Linked List and Predicates *124*
3.33 What are the Attributes and Methods of a Polynomial as a Data Structure? *125*
3.34 How to Represent a Polynomial using a Single Linked List *126*
3.35 Polynomial Tool Box *126*
3.36 How to Add a New Term to a Polynomial *127*
3.37 How to Add Two Polynomials and Return Their Sum *128*

3.38 How to Multiply Two Polynomials *128*
3.39 How to Find the Differentiation of a Polynomial *128*
3.40 How to Calculate the Integral of a Polynomial *128*
3.41 How to Evaluate the Value of the Polynomial at a Value *129*
3.42 How to Find the Definite Integral Value of a Function *129*
3.43 How to Display a Polynomial *129*
3.44 How to Find the Value of a Composite Function *129*
3.45 How to Add a New Term to a Polynomial *130*
3.46 How to Add Two Polynomials of Three Variables *131*
3.47 How to Multiply Two Polynomials of Three Variables *131*
3.48 How to Differentiate a Polynomial with Respect to *x* *132*
3.49 How to Differentiate a Polynomial with Respect to *y* *132*
3.50 How to Differentiate a Polynomial with Respect to *z* *132*
3.51 How to Integrate the Polynomial with Respect to *x* Assuming the Other Two Variables are Keeping Constant *132*
3.52 How to Integrate a Polynomial with Respect to *y* Assuming the Other Two Variables are Keeping Constant *133*
3.53 How to Integrate a Polynomial with Respect to *z* Assuming the Other Two Variables are Keeping Constant *133*
3.54 How to Integrate a Polynomial when All Three Variables are Varying *133*
3.55 How to Evaluate a Polynomial at Given Values of *x*, *y* and *z* *133*
3.56 How to Integrate a Polynomial within a Given Limit *134*
3.57 Some Applications of Polynomial Toolbox *134*
3.58 How to Find the Curl of a Function of Three Variables *134*
3.59 Huge Numbers: Application of Linked Lists *135*
3.60 How to Store Two Huge Integers as Single Linked List and then Add those Two Numbers and Display the Summation *136*
3.61 Digital Signal Processing *138*
3.62 How to Find the Length of a Signal *138*
3.63 How to Find the Index of a Given Amplitude in a Signal *139*
3.64 How to Add a New Value at the End of a Signal *139*
3.65 How to Add a New Value at the Front of a Signal *140*
3.66 How to Return the First Signal Node Pointer *140*
3.67 How to Return the Last Signal Node Pointer *141*
3.68 How to Insert a Node at a Particular Location of a Signal *141*
3.69 How to Display a Digital Signal *141*
3.70 How to Get the Amplitude of the First Signal Node *142*
3.71 How to Get the Amplitude of the Last Signal Node *142*
3.72 How to Find Frequency of a Particular Amplitude in a Given Signal *142*
3.73 How to Get the Address of a Signal Node Given the Index *142*
3.74 How to Get the Amplitude of a Signal at a Particular Point *143*
3.75 How to Check whether a Digital Signal is Even or Not *143*
3.76 How to Check whether a Digital Signal is Causal *143*
3.77 How to Check whether a Digital Signal is Anti-Causal *144*
3.78 How to Check whether a Digital Signal is Non-Causal *145*
3.79 How to Add at the End of a Single Circular Linked List *146*
3.80 Double Linked List *146*
3.81 How to Add a Number at the End of a Double Linked List *147*
3.82 How to Add a Number at the Front of a Double Linked List *147*
3.83 How to Go to the Next Node of a Double Linked List *147*

3.84 How to Go to the Previous Node of a Double Linked List *148*
3.85 How to Display the Double Linked List in Forward Direction *148*
3.86 How to Display the Double Linked List in Backward Direction *148*
3.87 How to Insert a Value at a Location in the Linked List *148*
3.88 How to Add a Number at the end of a Circular Doubly Linked List *149*
3.89 Linked List Applications in Biochemistry *149*
3.90 How to Hybridize Two Single DNA Strands to One DNA *151*
3.91 How to Melt One DNA to a Couple of Strands *152*
3.92 How to Emulate Linking of One DNA Strand to the Other *153*
3.93 How to Represent a Sparse Matrix using Jagged Arrays *153*
3.94 How to Add an Item in the Sparse Matrix *154*
3.95 How to Add a Jagged Row to the Jagged Representation of the Sparse Matrix *155*
3.96 How to Accept the Sparse Matrix Details from the User and Create the Jagged Array of Linked Lists to Represent the Matrix in a Space-efficient Way *156*
3.97 Representation of Handwritten Signatures using Jagged Arrays *157*
3.98 How to Model *Simple Content Management* System using Linked List *157*
3.99 How to Model *Workflow Engine* System (*Like* K2.NET) using Linked List *161*
3.100 A Comparison between Arrays and Linked Lists *161*
Revision of Concepts *163*
Review Questions *163*
Programming Problems *164*

4 STRINGS—*DATABASE TO DNA!* **165**

Introduction *165*
4.1 Some Key Facts about Strings in C *165*
4.2 C-Style String *166*
4.3 How to Initialize at the Time of Declaration *166*
4.4 How to Initialize Strings using user-Defined Values *166*
4.5 How to Initialize One String with Another String *167*
4.6 How to Initialize a String using Character Values *167*
4.7 How to Initialize a String using ASCII Values *167*
4.8 Introduction to some Built-in Turbo C String Library Functions *168*
4.9 Designing Utility Tools using these Two Functions *172*
4.10 A Tool for Changing Case of Few Chosen Abbreviations in a File (Using strupr()) *173*
4.11 How to Reverse a String *174*
4.12 How to Set Characters of a String with Another Character *175*
4.13 How to Find the First Occurrence of a Character of a Substring within Another String *176*
4.14 How to Find the Location from where Two Strings Start to Differ *177*
4.15 How to Create the Duplicate of a String in a Memory-Efficient Manner *178*
4.16 How to Tokenize a given String *178*
4.17 What do you mean by Prefix of a String? *179*
4.18 What do you mean by Suffix of a String? *180*
4.19 What do you mean by Subsequence of a String? *181*
4.20 How to Check whether a String is a valid ISBN or not *194*
4.21 How to Check Validity of a Social Insurance Number (SIN) Code *195*
4.22 How to Check whether a given Credit Card Number is Valid or not *195*
4.23 How to Change the Case of a Sentence to Sentence Case *200*
4.24 How to Toggle the Case of the Letters of a Sentence *200*
4.25 How to Find out the Soundex Code for a given Word *205*
Review Questions *207*
Programming Problems *207*

5 RECURSION—*TIME AND AGAIN!* **208**

Introduction *208*
5.1 Different types of Recursion *208*
5.2 Pitfalls of Recursion.. *209*
5.3 Fibonacci Numbers and Golden Ratio *209*
5.4 Random Number Generation using Recursion *212*
5.5 How to Generate Pseudo Random Numbers(PRNs) using Von Numann's Middle Squaring Method *213*
5.6 How to Generate the Ackermann's Function *216*
5.7 What is Inverse Ackermann's Function? *216*
5.8 How to Generate TAK Function for given Variables *216*
5.9 Solving Non-Linear Equations using Recursion *221*
5.10 Pattern Generation using Recursion *229*
5.11 How to Write a Recursive Function to Generate the Numbers of the Pascal Triangle *232*
5.12 What is the Relationship between Pascal Triangle Numbers and Fibonacci Numbers? *233*
5.13 How to Write a Recursive Function to Generate the Numbers of the Bell Triangle. To Accept Two Numbers, One for Row and the Other for Column *233*
5.14 Application of Bell Numbers *234*
5.15 How to Write a Recursive Function to Generate the Numbers of Bernoulli Triangle *234*
5.16 How to Write a Recursive Function to Generate the Numbers of Catalan's Triangle *235*
5.17 What is the Recursive Relation that Generates a Catalan Number? *236*
5.18 Solving Euler's Polygon Division using a Catalan Number *236*
5.19 DYCK Path and Catalan Number *236*
5.20 Ballot Problem and Catalan Number *237*
5.21 How to Write a Recursive Function to Generate the Numbers of Losanitsch's Triangle *237*
5.22 How to Write a Recursive Function to Generate the Numbers of the Leibnitz Harmonic Triangle *238*
5.23 L-System, Recursion and more Fractals *242*
5.24 Fractal Generation using Recursion *243*
5.25 Koch Curve *243*
5.26 Koch Snowflake *243*
5.27 Recursion in Natural Scene Generation *244*
Revision of Concepts *245*
Review Questions *245*
Programming Problems *246*

6 STACK—*ONE UPON ANOTHER* **247**

Introduction *247*
6.1 Model a Stack as a Struct *247*
6.2 How to Initialize the Stack Modeled above *248*
6.3 How to Pop The MRA Element from the above Stack *249*
6.4 How to Display the Stacktop Element *249*
6.5 How to Swap the Top Two Elements *250*
6.6 Putting it All Together using Arrays *250*
6.7 Model a Stack using a Linked List *251*
6.8 Pushing an Element *252*
6.9 How to Pop an Element from the Stack *252*
6.10 How to Peep at the Stack Top *253*
6.11 How to Swap the Top Two Elements *253*
6.12 How to Write a Parenthesis Matcher using Stack *258*
6.13 Switchbox Routing Problem *258*

6.14 Saguaro Stack *265*
6.15 How to Write the Algorithm to use a Saguaro Stack to Check a Wrongly Entered URL *267*
6.16 What is an MTFL? *271*
6.17 How to Model an MTFL using Two Stacks which are themselves Modeled by a Linked List *271*
6.18 How to Find the Most Sought Item in a Departmental Store using an MTF List *276*
6.19 What is Backtracking? *277*
6.20 How to Develop a Backtracking Algorithm to Find a Path in a Maze using Stack *277*
Revision of Concepts *282*
Review Questions *282*
Programming Problems *283*

7 QUEUE—*WAITING OR PRIVILEGED?* **284**

Introduction *284*
7.1 How to Model a Linear Queue using an Array *284*
7.2 How to Initialize the Linear Queue Defined above *285*
7.3 How to Append an Element in the Queue *285*
7.4 How to Delete from a Queue *286*
7.5 How to Search an Element in the Queue *288*
7.6 How to Display the Elements in a Queue *289*
7.7 Model a Queue using a Linked List *289*
7.8 How to Append an Item to a Linked Queue *290*
7.9 How to Delete the Front Item of a Queue *291*
7.10 How to Search an Element in a Queue *291*
7.11 How to Display the Elements of a Queue *292*
7.12 Model a Linear Queue using two Stacks *294*
7.13 How to Model a Stack using two Queues *295*
7.14 Model a Circular Queue using Structure *297*
7.15 Model a Priority Queue using an Array *298*
7.16 Model a Priority Queue using a Single Linked List *302*
7.17 An Application of Priority Queue—*Scheduling Appointments* *309*
7.18 How to Model a Deque (Double-Ended Queue) using a Linked List *316*
7.19 How to Model a Move to Front List (MTFL) using a Queue *317*
7.20 How to Simulate the Queue in Front of the Cash Counter *318*
Revision of Concepts *322*
Review Questions *323*
Programming Problems *324*

8 TREES—*EXPLORER TO GENETICS!* **325**

Introduction *325*
8.1 What are the Different Ways Trees are Represented? *327*
8.2 What is a Strictly Binary Tree? *327*
8.3 What is an almost Complete Binary Tree? *327*
8.4 What is a Complete Binary Tree? (*Also known* as Perfect Binary Tree) *327*
8.5 What is a Weak Binary Tree? *328*
8.6 What is a Strong Binary Tree? *328*
8.7 How to Model a Binary Tree using an Array *329*
8.8 How to Find whether a Node is a Right Child of its Parent *333*
8.9 How to Find the Address of the Sibling of a Node *335*
8.10 How to Find the Address of the Uncle of a Node *335*
8.11 How to Traverse the Tree "In-order" *335*

8.12 What is a Binary Search Tree? *337*
8.13 How to Search a Value in a BST *340*
8.14 What is the Right Rotation on a BST? *341*
8.15 Some Areas of Application of BST *342*
8.16 What is an Expression Tree? *343*
8.17 What is a Decision Tree? *347*
8.18 Binary Search Tree and Games *348*
8.19 How to Handle Multiple Subjects in such an Adaptive Test *351*
8.20 How to Convert a Multiway Tree to a Binary Tree *353*
8.21 What is the Balance Factor of a BST? *354*
8.22 How to Balance a Binary Search Tree *355*
8.23 What is a Splay Tree? *356*
8.24 What is a Heap? *358*
8.25 What are the Different Approaches to Create a Heap? *358*
8.26 How to Implement a Binary Heap using Array *359*
8.27 What is an AVL Tree? *359*
8.28 How to Insert an Element to an AVL Tree *359*
8.29 What is a BSP Tree? *360*
8.30 What is a Quad Tree? A 2D Variation of BSP (*Also known as* Q-tree)? *360*
8.31 How to Get the North-East Uncle of a Quad Tree Node *362*
8.32 How are Images Represented using a Quad Tree? *364*
8.33 How to Convert a Quad Tree to a Binary Tree *365*
8.34 Superimposing Multiple Binary Images using a Binary Tree *365*
8.35 Handwriting Recognition using Quad Tree *367*
8.36 How to Compress Images using a Quad Tree *368*
8.37 What is an Octree? *370*
8.38 What is a Trie? *371*
8.39 How to Model a Trie using Linked List *371*
8.40 How to Add a Key to a Trie *372*
8.41 How to Search a Key in a Trie *372*
8.42 How to Find whether a Key in a Trie can be Deleted or Not *375*
8.43 How to use a Trie for Spell Checking *376*
Review Questions *377*
Programming Problems *377*

9 GRAPHS—*MATHEMATICS TO WAN* ***378***

Introduction *378*
9.1 What are the Different Ways Graphs are Represented? *378*
9.2 How to Add an Edge in a Graph Modeled by Adjacency Matrix *379*
9.3 How to Remove an Edge in a Graph Modeled by Adjacency Matrix *379*
9.4 What is a Path Matrix? *379*
9.5 How to Find whether a Graph is a Tree or not *379*
9.6 What is Minimum Spanning Tree of a Graph? *380*
9.7 Prim's Algorithm *380*
9.8 Kruskal's Algorithm to find the MST *381*
9.9 Reverse Delete Algorithm to find the MST *382*
9.10 What is a Directed Acyclic Graph or DAG? *384*
9.11 What is Topological Sorting of a DAG? *384*
9.12 How to Find whether a Graph is Planer or not *392*

9.13 Breadth First Search (BFS) *396*
9.14 Depth First Search (DFS) *398*
Revision of Concepts 398
Review Questions 399
Programming Problems 399

10 SORTING—*MICRO, MACRO, MAMMOTH* ***400***

Introduction *400*
10.1 Functions in this Chapter *400*
10.2 Sorting Algorithms Classifications *400*
10.3 Exchange Sort Algorithms *400*
10.4 What is the Time Complexity of Bubble Sort? *403*
10.5 What is Odd-Even Transposition Sort? *404*
10.6 What is the Time Complexity of Bidirectional Bubble Sort? *406*
10.7 What is the Time Complexity of Comb Sort? *409*
10.8 Insertion Sort Algorithms *410*
10.9 What is the Time Complexity of Insertion Sort? *410*
10.10 Comparison with the Ideal *O*(*N*^2) Curve *411*
10.11 What is the Time Complexity of Binary Insertion Sort? *413*
10.12 Problems with Insertion Sort: Shifting *413*
10.13 Explain the Library Sort Algorithm (*Also known as* Gapped Insertion Sort) *414*
10.14 What is the Time Complexity of Shell Sort? *415*
10.15 Selection Sort Algorithms *416*
10.16 What is the Time Complexity of Selection Sort? *417*
10.17 What is Bingo Sort? *418*
10.18 Hybrid Sort Algorithms *418*
10.19 What is J-Sort? *418*
10.20 Divide-N-Conquer Sorting Algorithms *419*
10.21 How to Write a Function to Demonstrate Quick Sort *419*
10.22 What is the Time Complexity of Quick Sort? *419*
10.23 How to Select the Pivot in Quick Sort *420*
10.24 What is the Time Complexity of Merge Sort? *421*
10.25 What is the Time Complexity of Stooge Sort? *422*
10.26 Distribution Sorting Algorithms *422*
10.27 Bucket Sort *422*
10.28 Performance Comparisons of the Sorting Algorithms with *O* (*n*^2) Time Complexity *423*
10.29 Performance Comparisons of the Sorting Algorithms with *O* (*n* log *n*) Time Complexity *424*
10.30 Bogo Sort and Friends *424*
10.31 Tree Sort *425*
10.32 Lexicographic Sort *425*
10.33 Radix Sort *426*
10.34 Address Calculation Sort using Hashing *426*
10.35 Application of Sorting *428*
10.36 What is Clustering? *428*
10.37 Business Clustering *428*
10.38 Finding the Shortest Path *428*
10.39 Finding the Most Wanted DVD in the City *428*
10.40 Finding the Greatest Online Scorer in the Online Pool Competition *428*
10.41 Finding the Largest Shape when their Dimensions are Given *429*
Revision of Concepts 429
Review Questions 430
Programming Problems 430

11 HASHING—*ACCIDENT OR CHOICE?* **431**

Introduction *431*
11.1 Concept of Collision and its Resolution *431*
11.2 Some Key Facts and Jargons about Hashing *433*
11.3 How to Demonstrate the Separate Chaining Method for Hashing Elements in a Hash Table *435*
11.4 What is Coalesced Hashing? *437*
11.5 What are the Variations of Coalesced Hashing (*Linked Hashing*)? *441*
11.6 What is a Hash Chain and what is its Utilization for OTP? *441*
11.7 How is a Hash Tree used to Check the Data Integrity of a Media Downloaded from a Peer-to-Peer (P2P) Network *443*
Review Questions *444*
Programming Problems *444*

12 ADT—*DELIVERED INBUILT PLUMBING!* **445**

12.1 The Black-Box Concept *445*
12.2 ADT *445*
12.3 ADT Design in C *446*
12.4 Designing your own ADT *446*
Review Questions *448*
Programming Problems *448*

13 DATE—*TODAY WAS TOMORROW!!* **449**

Introduction *449*
13.1 How to Find the Day of Week (Sun, Mon, etc) *453*
13.2 How to Find the Date of the Next N^{th} Sunday, from a given Date *456*
13.3 How to Find the Date of the Previous N^{th} Sunday, from a given Date *458*
13.4 Wrapper Functions: Increases the Readability of Your Code *459*
13.5 Interaction with the System Built in `Date` Structure *464*
13.6 Interaction with the Real World *465*
Review Questions *467*
Programming Problems *468*

14 MAP—*PHONEBOOK, DICTIONARY, CRYPTOGRAPHY* **469**

Introduction *469*
14.1 How to Represent a Map *469*
14.2 How to Define a Predicator over a Map and use it from a Client *474*
14.3 How to know who is who's Friend from the Buddy List *474*
14.4 How to Design a Random Cipher Encoder using a Map *475*
14.5 Application of Map of Maps *476*
14.6 Multilanguage Word Map *477*
14.7 Key Interlinked Map (KIM) *477*
Review Questions *478*
Programming Problems *478*

15 CURRENCY—*NO PRIMITIVE PLEASE!* **479**

Introduction *479*
15.1 A Practical Application: Getting the Lowest Bid Amount *484*

15.2 How to Convert USD to GBP and vice versa *485*
15.3 How to Convert USD to GBP and vice versa Datewise *486*
Review Questions *487*
Programming Problems *487*

16 FILE HANDLING—*SEED, SAVE, SHARE* **488**

Introduction *488*
16.1 What is File? *488*
16.2 What does the Function `rewind`() Do? *492*
16.3 How to Simulate UNIX `cat` Command *496*
16.4 How to Simulate UNIX `grep` Command with Exact Match *497*
16.5 How to Simulate UNIX `Grep` Command for Switch–*V* *498*
16.6 How to Print those Lines of a File that Contain a Word that Sounds like a given Word *503*
16.7 How to Replace a Character in a File with Another Character *503*
16.8 How to Replace a Word in a File with Another Word *504*
16.9 How to Compare Two Text Files Line by Line *505*
16.10 How to Print Same Lines of Two Files *506*
16.11 How to Copy a File from a Source to a Destination *507*
16.12 File Handling in Console-Based Games *524*
16.13 Function Definitions *532*
Review Questions *537*
Programming Problems *538*

Appendix A: Project Ideas! ***539***

Appendix B: Bibliography ***543***

Index ***545***

Before We Start

Whom is the Book for? ***Beginners*** who have just picked up C and want to try their hands on some data structures can use this book. This book could be used as a supplement to any undergraduate data structure course.

Intermediate programmers who have a grip over C and common data structures will come to know about other data structures like 'tries', 'Hash Maps' and how to design them using age-old C building blocks like arrays, structures, etc.

Expert programmers can use this book as a handy data structure reference. So this is a book for everyone who deals with data structures!!

Happy Structuring!
Best of Luck ☺

Organization of this Book Programming examples are the prime focus of this book. It attempts to teach its reader how to apply well-known data structures to solve problems in diverse areas/fields. There is an algorithm for approximate string matching that finds application in cellphones, DNA matching and statistical analysis for medicine grouping. Here, we will learn the algorithm as a topic with more inputs for its application to solve real-world problems, rather than beating around the theory of the algorithm.

Compilers and OS I have used Microsoft Visual Studio 6.0, Microsoft Visual Studio Express Edition and Turbo C(for some programs) to code the applications. But these codes can be compiled in Turbo C 3.5 or higher under DOS. Whenever any part of the code is compiler specific, that is clearly mentioned. My PC runs on *Microsoft Windows XP Professional.* You can use any Windows OS.

What is a Data Structure? A Data Structure is nothing but a container for data. As water is to a container, data are to data structures. In simple terms, it is a structure that could hold your data. Thus, the name *data structure* is justified.

Why do we need One? For simple applications like arithmetic manipulations, we can do away with user-defined data structures, but for rather serious applications where we need to store a lot of information like age, name, address, etc., we need to have things like 'Data Structures'. Sometimes it may happen that any of the inbuilt data structures (although C has only one 'array') does not solve a particular need. Then we shall have to write our customized data structures using the available primitive data structures of the language.

1

Array
Easy, Contiguous, Elegant!

INTRODUCTION

An array is the most basic data structure that C offers. Let's say we have to find the average marks obtained in a subject by students of a class or, assume that we shall have to find the histogram of a gray-level image. In these two cases we have a couple of choices. We can either store all the values in different variables and use them, or we can create an array that will hold all these values in contiguous memory locations. In C, arrays are declared as follows:

<data type> Array_Name [Number of elements in the array]

For example,

`int array[20]` is an array of 20 integers.

1.1 HOW TO INITIALIZE AN ARRAY

There are many ways to initialize an array.

Initialization: While Declaring the Array

1. `int codes[10]={1,2,3,4,5,6,7,8,9,10};`
2. `int codes[10]={0};//Only in C++`
3. `int codes[10]={1,2};`
4. `int codes[10];`

Here, **codes** is an 'integer array of ten elements', which has 10 elements *initialized* with values ranging from 1 to 10. In the second statement, all the values of the **codes** array are initialized to 0. In the third statement, the first two of the **codes** array are set to 1 and 2, and the rest 8 values are left blank. The last statement doesn't initialize anything. If you don't initialize the array then it will be full of junk/garbage items.

Initialization Using a Loop

A loop can be used to initialize when the array declared is very long or when the array elements have a logic associated with the array index. For example, suppose we want to plot the ramp function which is given by $f(x) = x$. In this case, we will use a loop to initialize the array. The code snippet shows how to initialize an array for the ramp function.

```
int myRamp[100];
int i;
for(i=0;i<100;i++)
{
      myRamp[i]=i;//This is same as f(x) = x
}
```

Initialization: With Values from Another Array

Sometimes, one array needs to be filled with values from other arrays with/without manipulations. In Cipher text applications we need to slide the characters. This is done mainly for copying one array to the other. An example is shown below.

```
char myAlphabets[]={'a','b','c','d'};
char yourAlphabets[4];
int I;
for(I=0;I< strlen(myAlphabets);I++)
{
      yourAlphabets[I]=myAlphabets[I];
}
```

Initialization: With Specific Values

Sometimes we need to arbitrarily initialize the specific values of an array. Say, we want to initialize the 10^{th} element of an array, and then we will write a `[9] = 24`. As the array index starts from zero, the 10^{th} element is a[9]. But one should be careful while initializing the specific values so that it doesn't go beyond the bound of the array because C doesn't check for array index overflow error.

1.2 HOW TO TRAVERSE AN 1D ARRAY USING INDEX

This is a trivial problem. In data structures, we need to traverse an array very often. We could transverse an array using any logic. For this purpose, we could use a simple loop. We can access the array elements by array index or by a pointer of the same type as that of the elements of the array. In case we want to access the elements of the array by a pointer, we need to initialize the pointer with the base address of the array, i.e. the address of the first element. Here it is shown how to access the elements using an index.

```
//In C Using Index
int my_array_name[4]={10,20,30,40};
int counter=0;
for(counter=0;counter<4;counter++)
      printf("%d\n",my_array_name[counter]);
```

An array name is nothing but a pointer to the same array.

How to Traverse a One-dimensional Array Using Pointer

```
//Using Pointer
int my_array_name[4]={10,20,30,40};
int counter=0;
for(counter=0;counter<4;counter++)
      printf("%d\n",*(my_array_name+counter));
```

How to Traverse a Two-dimensional Array Using Index

```
int my_2d_array[10][10];
//Assume that the array my_2d_array is already pre-filled.
int i,j;
//Traversing the array
for(i=0;i<10;i++)//Walk Down-wise the rows.
     for(j=0;j<10;j++)//Walk across the columns.
      printf("%d", a[i][j]);
printf("\n");
```

How to Traverse a 2D Array Using Pointer

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

1.3 HOW TO MANIPULATE ELEMENTS OF THE ARRAY

In data structure, you might have to take the summation of the elements of the array or do some kind of a mathematical operation on them. To make things complicated, you may need to operate functions over array indices and then use the elements of those indices which satisfy a predefined condition, as arguments of another function. Here is a code that prints the square of the even numbers from one to ten.

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

int isEven(int m)
{
      return m%2==0?1:0;
}

int main()
{
       int a[10];
       int i;
       for(i=0;i<9;i++)
          a[i]=i+1;
       for(i=0;i<9;i++)
       {
          //Checking whether the index is even or not
          if(isEven(i+1))
          //Can perform any mathematical operation like
          //Add
          //Multiplication
               printf("%f\n",pow((float)a[i],2));
        }
       getch();
       return 0;
}
```

You may also be required to operate functions over a particular section of the array. For example, if you want to find whether a number is divisible by 11 or not, then you need to find the difference between the sum of digits in even and odd places. Some examples of manipulation of array elements are listed below.

1. Add the array elements.
2. Multiply the array elements.
3. Add elements in the even places.
4. Add elements in the odd places.
5. Find $f(x)$ of where x is the set of array elements.
6. Add a number to each element of the array.
7. Subtract a number from each element of the array.
8. Multiply a number to each element of the array.
9. Divide all array elements by a number.
10. Add a number to specific array elements.
11. Subtract a number from specific array elements.
12. Multiply a number to specific array elements.
13. Divide specific array elements by a number.
14. Find all even array elements.
15. Find all odd array elements.
16. Find the square of the array elements, and so on…

1.4 HOW TO ADD ARRAY ELEMENTS IN A SPECIFIC REGION

This function will add all the elements of the array and will return the sum to the calling method. This function will take the array name and start and finish indices as arguments. Here is the C code.

```
int Add(int array[],int start,int finish)
{
      int i=start;
      int sum = 0;
      for(;i<=finish;i++)
            sum+=array[i];
      return sum;
}
```

This method will add all the array elements whose locations fall between the start and finish index inclusive of the two. Say, there is an array like

```
int a[]={1,2,3,4,5,66,7,8,9,10,11,12};
```

And we want to add 1 to 5; then we will call the above method as `Add(a,0,6)`.
This function can be used to find out the summation of all the array elements if the start index is 0 and the last index is the number of array elements—1. That means for the above array, if we want to find the summation of the entire array elements then we should write

```
Add(a,0,11);
```

1.5 HOW TO ADD ELEMENTS IN THE ODD AND EVEN PLACES IN THE ARRAY

To add elements within a specific region we can pass the start and finish index as shown in the C code above. We *can make this function even more intelligent*. This intelligent function can be used to sum those integers in the even place of the array or the odd place elements, or can be made to return the sum of all the elements in the specified range. *A flag will be passed as the third argument*. If the argument is 0 then the method will return sum of all the elements in the specified range including the start and finish

index. If the argument is 1, then it will return the sum of only the even place numbers. If the argument is 2, then this method would return the sum of all integers in the odd place within the specified range by start and finish. Here is the code in C.

```
double ArrayAdd(double array[],int start,int finish,int flag)
 {
      int i=start;
      double sum = 0;
      if(flag==0)//Add all the elements from start to finish
      {
            for(;i<=finish;i++)
                  sum+=array[i];
      }
      if(flag==1)//Adding only the even place numbers
      {
            if(start%2==0)//Where to really start from?
                  i=start;
            else
                  i=start+1;
            for(;i<=finish;i+=2)
                  sum+=array[i];
      }

      if(flag==2)//Adding only the odd place numbers
      {
            if(start%2!=0)//Where to really start from?
                  i=start;
            else
                  i=start+1;
            for(;i<=finish;i+=2)
                  sum+=array[i];
      }
      return sum;
}
```

Have you noticed that we can use this function as the building block of our program? *Can you write a program that will find whether a given number is divisible by 11 or not without using the module(%) operator? Try it! Use this addition method. [Clue: The solution will be recursive in nature].*

1.6 HOW TO PERFORM OPERATIONS INVOLVING EXTERNAL VARIABLES

Sometimes, you will have to multiply the array elements by an external variable or constant. *One very clear example of this is magnifying the value of a vector in all three coordinates*. A function can be written in C/C++ that will allow doing any mathematical operation involving array elements and an external constant or a variable. Here is the code in C.

```
void ExtOp(int array[],const float MyCon,int start,int finish)
 {
    int i=start;
    for(;i<finish;i++)
      {
            //Write your Code here
            array[i]*= MyCon;
            printf("%d ",array[i]);
      }
}
```

How to Multiply the Elements of an Array

```
int mult=1;
void multiplyelements(int array[],int size)
{
          for(i=0;i<size;i++)
                mult*=array[size];
}
```

How to Add only the Even Elements in the Array

```
int evenadd(int array[],int size)
{
         int i=0;
         int Sum=0;
         for( ; i<size;i++)
          {
                  if(array[i]%2==0)//If the element is even or not
                           Sum+=array[i];
           }
         return Sum;
}
```

How to Add only the Odd Elements in the Array

```
int oddadd(int array[],int size)
{
          int i=0;
          int Sum=0;
         for( ; i<size;i++)
         {
                  if(array[i]%2!=0)
                     Sum+=array[i];
         }
         return Sum;
}
```

How to Add an Element to Every Element of the Array

```
void addanumber(int array[],int size,int number)
{
     int i;
     for(i=0;i<size;i++)
     {
         array[i]+=number;
     }
}
```

How to Subtract an Element from Every Element of the Array

```
void subanumber(int array[],int size,int number)
{
     int i;
     for(i=0;i<size;i++)
     {
         array[i]-=number;
     }
}
```

How to Multiply an Element to Every Element of the Array

```
void mulnumber(int array[],int size,int number)
{
     int i;
     for(i=0;i<size;i++)
     {
         array[i]*=number;
     }
}
```

This type of multiplication in case of vectors are known as **amplitude modification**.

How to Divide an Element from Every Element of the Array

```
void DivideByANumber(int array[],int size,int number)
{
     int i;
     for(i=0;i<size;i++)
     {
         array[i]/=number;
     }
}
```

How to Square Each Element of the Array

```
void SquareArrayElements(int array[],int size,int number)
{
     int i;
     for(i=0;i<size;i++)
     {
         array[i]=pow(array[i],2);
     }
}
```

1.7 HOW TO FIND FUNCTION VALUES

Functions are the most-used mathematical operations. They are most popular as they help us to understand the relationship between one variable with another from a scientific point of view. Thus, functions have widespread applications in science, commerce, sociology and biotechnology.

1.8 HOW TO SOLVE A DEMOGRAPHICAL APPLICATION, A PROBLEM OF VITAL STATISTICS

A demographical problem tells that, in a city, every year 10% of the population gets married and 3% of the couples married in the previous year plan to have babies. Now we are given a list of the population in the city for the last 10 years. We shall have to find how many people got married in the tenth year and how many of them planned to have their babies in the next year, i.e., the 11^{th} year. We can easily write a method that will give us how many people get married each year and then we can get to know how many of them are actually planning to have their babies. Here is the C code to solve this problem.

```
long HowManyGotMarriedEachYear (long Population)
{
    long PeopleWhoGotMarried;
    PeopleWhoGotMarried=Population*0.1;
    return PeopleWhoGotMarried;
}
```

We have created an array for holding the population of the city. And then with each year's population, we shall call this method that will return the count for people who got married that year. Then we can store the value in the same array. Here is the call to this method.

```
long pop[]={1111,2222,3333,4444};
long mar[4];
int i =0;
for(;i<4;i++)
    printf("%l\n",HowManyGotMarriedEachYear(pop[i]));
```

This gives the output.

111
222
333
444

as expected of 10% of the total population. To find the number of couples who want their babies on the 5^{th} year we will have to find 3% of 10% population on the 4^{th} year, that is, 444. So the answer is

```
printf("Children on 5th year is
%l",HowManyGotMarriedEachYear(pop[3])*0.03);
```

There can be innumerable other applications.

Can you extend this to find out how many toddlers (0–4 year kids) are there on the 11^{th} year, assuming no kid died in the past 4 years.

1.9 WHERE TO APPLY 3D ARRAYS

You may wonder why and when should we use a 3D array. There can be many situations where we may need to use a 3D or multi-dimensional array. Always remember, that for a given problem with *n degrees of freedom*, we need an *n-dimensional array* to solve the problem.

For example, if you want to keep track of the locus of a moving particle in 3D using Cartesian coordinates, then the coordinates in the three axes need to be stored. We can find at any particular point of time where the particle was by using these array elements.

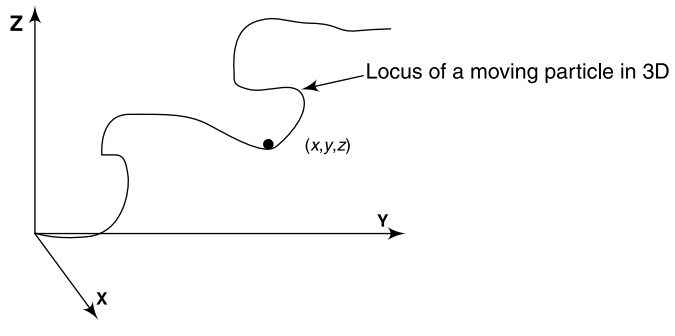


Fig. 1.1

Let's use an array of double coordinates[100][100][100] that will hold the coordinate values in *x*, *y*, and *z* . These types of storages can help us solve many problems of particle dynamics, e.g. modeling Brownian motion, etc. This problem can be more easily solved with structures. Please refer the chapter on structure for the point structure and its usage in geometrical applications.

1.10 HOW TO DELETE A PARTICULAR ITEM FROM AN ARRAY

Deletion in an array can be done in two ways. They are

1. Deletion by item
2. Deletion by location

Suppose there is an array, which contains the following integers 1,2,3,4,5. And we want to delete 3. Before deletion, the situation is like this.

Elements	1	2	3	4	5
Index	0	1	2	3	4

After deletion the memory will be like this.

Elements	1	2	4	5	0
Index	0	1	2	3	4

Fig. 1.2

So to delete 3, we have to do the following:

1. Find the location of the item to be deleted.
2. For all elements which come to the right of the *element to be deleted*, the index decreases by one. For those which come before the *element to be deleted*, the index would remain the same.
3. The *element to be deleted* is overwritten by the immediate next element.
4. This process continues till the end of the array.

Notice the above diagram. The indices of the elements after the deleted item have decreased by unity.

```
void delete_item(int a[],int size,int x)
{
      int i;
      int flag=0;
      for(i=0;i<size;i++)
            if(a[i]==x)//Searching the number to delete.
            {
                  flag=1;//The number searched is found
                  break;
            }
      if(flag==1)
      //Shifting rest-all elements to the left by unity
            for(int k=i;k<size-1;k++)
                  a[k]=a[k+1];
      else
            printf("The value is not found!");
}

int main()
{
      int a[10]={1,2,3,4,5,6,7,8,9,10};
      display(a,10);
      getch();
      delete_item(a,10,3);
      display(a,9);
      getch();
      return 0;
}
```

1.11 HOW TO DELETE AN ITEM FROM A PARTICULAR LOCATION

Sometimes, we may want to delete a particular item from a particular location. In this case, the item in that location is not important. It is only the location in the array that is important. It is like saying "delete the 4th item of the array from the beginning" or "delete every 2nd element from the end". In these cases we need not search for the element in the array. All we have to do is to shift the position of the rest items to the left accordingly. Here is an example. Say there is a character array like

```
char myCodes[]={'X','Y','W','Z','D','C','B','A'};
```

We want to delete the 4th element of the array. For that we need to write a function that will take the array and looks for the specified location and deletes the element from the identified location. It will return true if the deletion is successful, else it will return "false".

```
int delete_by_location(char myCodes[], int delete_location)
{
      int flag = 0;
      if(delete_location>=0 && delete_location<size)
      {
            for(int i=delete_location;i<strlen(myCodes)-1;i++)
                  array[i]=array[i+1];
            flag = 1;//If deletion is successful
      }
      return flag;
}
```

There can be more complex situations when we want to delete elements from an array following a **particular pattern** or **depending on some condition**. Here, we will discuss only deletion that follows a particular pattern. Consider, there is a situation where we have to delete every 2nd element of the array from the start. That means if initially the array looks like

Elements	1	**2**	3	**4**	5	**6**	7
Index	0	1	2	3	4	5	6

then after deletion it will look like

Elements	1		3		5		7
Index	0		1		2		3

Fig. 1.3

Let's have a close look at the index of the array before and after deletion. Let's call the array before deletion **old_array** and after deletion let the name be **new_array**. Then from the above two figures, we can clearly conclude that

new_array[0] = old_array[0];
new_array[1] = old_array[2];
new_array[2] = old_array[4];
new_array[3] = old_array[6];

In general, we can say that new_array[i] = old_array[2*i] = old_array[i + i/1]

That means if we want to delete every 4th element from an array then the new array elements will be given by `new_array[counter]=old_array[counter + counter /3]`. After deletion the number

of elements of the array will be given by Initial Length – Initial Length/(Index of the first item to be deleted + 1). So if initially the array holds 20 elements and we delete every 4th element of the array then after deletion the number of elements of the array will be given by

New Length = Initial Length – Initial Length/(Index of the first item to be deleted + 1)
New Length = 20 – 20/(3+1) = 20 – 5 = 15.

Here is the C Code to delete every 4th element from an array.

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

int main()
{
      int a[]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};
     //Deleting every 4th element of the array
     for(int i=0;i<20;i++)
        a[i] = a[i + i/3];
     //Displaying the array after deleting the elements
     for(int i=0;i<20-20/(3+1);i++)
            printf("%d ",a[i]);
     printf("\n");
     getch();
return 0;
}
```

Here is the ***generalized code*** to delete any element of the array.

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

int main()
{
     int whichnumber;
     int a[]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};
     printf("Enter which number you want to delete :");
     scanf("%d",&whichnumber);
     for(int i=0;i<20;i++)
        a[i] = a[i + i/(whichnumber-1)];
     //Displaying the array after deleting the elements
     for(int i=0;i<20-20/whichnumber;i++)
            printf("%d ",a[i]);
     printf("\n");
     getch();
return 0;
}
```

1.12 HOW TO FIND THE MAXIMUM NUMBER IN AN ARRAY

To find the maximum value of the array is a simple task. What is needed is just a comparison of two items. If one is more than the other then the index for the maximum of the two values is stored in a temporary variable. At the end, the value is returned. *The number of loops it takes to find the maximum number from an array is = number of items in the array*,

Here is the C code for finding the maximum number from an integer array.

```
int FindMaximumNumber(int array[],int size)
{
      int max=array[0];
      for(i=1;i<size;i++)
      {
            if(array[i]>max)
                  max = array[i];
      }
       return max;
}
```

1.13 HOW TO FIND THE MINIMUM NUMBER IN AN ARRAY

Like the maximum number of the array, we can find the minimum number of the array. In this case the logic in the conditional operator will be just the reverse. Here is the C code.

```
int FindMinimumNumber(int array[],int size)
{
       int min = array[0];
       for(int i=0;i<size;i++)
      {
            if(array[i]<=min)
              min = array[i];
      }
       return min;
}
```

1.14 HOW TO SORT THE ARRAY ALPHABETICALLY

This is a very common task in programming. To sort alphabetically we can use the library method `strcmp()` and if we choose to ignore case then we can use the function `strcmpi()` from the string libraries. Here, a string array that holds a few names, is taken and then is sorted alphabetically in C.

```
void AlphaSort(char *array[],int size)
{
       char *temp;
       for(int j=0;j<size;j++)
       for(int i=0;i<j;i++)
       {
            if(strcmp(array[i],array[i+1])>0)
           {
                  strcpy(temp,array[i]);
                  strcpy(array[i],array[i+1]);
                  strcpy(array[i+1],temp);
           }
        }
       //Displaying the sorted array
       for(int i=0;i<size;i++)
            printf("%s\n",array[i]);

}
```

This method is capable of sorting names and alphanumeric codes alphabetically. Try this method with the following arrays:

```
char *names[]={"Zamal","Fakir","Amal","Kalam"};
char *codes[]={"A235","Z324","B325"};
```

call the method like AlphaSort(names, 4);

1.15 HOW TO CHECK IF A STRING IS A PALINDROME OR NOT

A *palindrome* is a symmetrical string of characters that will read the same whichever direction you read it. *There is a misconception that there can be palindrome words only. This is not true. There can be palindrome sentences also.* One example is "Was it a cat I saw". This sentence is taken from *Alice in Wonderland* .To find whether a string is a palindrome or not, people normally reverse the string and then check it with the original string. We can do this using a library function in C. It is much better to use a library function whenever we can do so instead of coding by hand. It makes the code faster and more readable.

Here is the C code.

```
int IsItAPalindrome(char *whatever)
{
     //To increase the readability of the code enum is used
      enum{NOT_PALINDROME,PALINDROME};
      if(strcmpi(whatever,strrev(whatever))==0)
            return PALINDROME;
      else
        return NOT_PALINDROME;
}
```

Here we have used two library functions.

`strcmpi()`: Compares two constant strings ignoring their case

`strrev()`: reverses a given string

The above program will work only for single words. Palindromes like

"Madam I'm adam" or "was it a cat I saw" will not be checked by the above program.

To find whether these strings are palindromes or not, we shall have to extract the special characters from the string first. Here is a code that performs that.

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

int main()
{
      char s[20]="Madam I'm Adam";
      char cs[20]="";
      char *rcs="";
      int k=0;
      int i=0;
      int flag=0;
      for(i=0;i<strlen(s);i++)
      {
           //Anything other than characters is not considered.
            if((s[i]>='a' && s[i]<='z') ||
               (s[i]>='A' && s[i]<='Z'))
            {
```

```
            cs[k]=s[i];
            k++;
         }

}
if(strcmp(cs,strrev(cs))==0)
     printf("It's a Palindrome\n");
else
     printf("It's not a Palindrome\n");
return 0;
}
```

1.16 HOW TO SEARCH FOR AN ARRAY ELEMENT

This is one of the most trivial activities of the array. But this is needed very often. There are many ways to find an array element. But here the most powerful method has been shown. That is a bit time consuming but it never fails. This method of searching is called **linear searching**. This method takes four arguments. They are

1. Array name
2. Start index, from where to search
3. Finish index, where to finish searching
4. What to find

Here is the C code of the method to find an element in an integer array.

```
int Find(int array[],int start,int finish,int whattofind)
{
      enum{NOT_FOUND,FOUND};
      int search_result = NOT_FOUND;
      int i;
      for(i=start;i<finish;i++)
      //Checking if this is the one we are searching for
            if(array[i]==whattofind)
            {
                  search_result = FOUND;//We got it!
                  break;
            }
       return search_result;
}
```

As you can see this method uses enum variables to store the status of the search. This increases the readability of the code. Always try to use such variables for status. The added advantage of enum is that there is no need to explicitly initialize the variables. They are automatically initialized starting from zero.

Break is used to come out of the loop as soon as the searched element is found. Otherwise we need to run through the entire loop, which will be expensive computationally.

1.17 HOW TO MAKE THE ARRAY ELEMENTS UNIQUE

If you are familiar with databases then you may know that there is a command in SQL called DISTINCT that is used for selecting distinct (i.e., unique) field values from the tables.

There can be many other situations like this where you need to find the unique values of the array. Suppose you want to do a statistical analysis of the marks obtained by the students in a class. There will be many students whose marks will be the same. So you need to make the marks unique before you find their frequencies (i.e., how many students score a particular mark). The method below takes the array name and its size as arguments and makes the array elements unique.

Now we are going to use this `Find( )` method to build another utility that will display only distinct elements of an array. Here is the C code that will make an integer array unique.

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

int Find(int array[],int start,int finish,int whattofind)
{
      enum{NOT_FOUND,FOUND};
      int search_result = NOT_FOUND;
      int i;
      for(i=start;i<finish;i++)
            if(array[i]==whattofind)
            {
                  search_result = FOUND;
                  break;
            }
      return search_result;
}

void Distinct(int a[],int size)
{
      int flag=0;
      printf("%d\n",a[0]);
      for(int i=1;i<size;i++)
      {
            if(!Find(a,0,i-1,a[i]))//Sliding the window
                  printf("%d\n",a[i]);
      }
}

int main()
{
      int a[]={1,2,3,4,2,3,4,5,6,7,8,9,4,8,9};
      Distinct(a,15);
      return 0;
}
```

Here the algorithm for making array elements unique is demonstrated. Let's assume that the array to be made unique is as in the above example.

```
int a[]={1,2,3,4,2,3,4,5,6,7,8,9,4,8,9};
```

The strategy is, starting from the 2nd index (i.e., 1) the loop will rotate till the end of the array. And while the loop counter is in (i-1) th place then the `Find()` method will be called to search whether a[i] exists between 0 and i-1 or not. If a[i] exists, then clearly it is a repetition and thus is not shown in the console. Otherwise, the value a[i] will be shown.

Say, for example, in the array a[ ] above, a[1] = 2 and a[4] = 2 also. So while the loop counter is at 4 it will check all the values of the array from 0 to 4-1(and that is 3). So the method finds that a[1] = a[4] thus clearly 2 is repeated in the array and will not be shown again as it has already been shown once. *In case we need to return the new distinct array, try to figure out what modification is needed*.

1.18 HOW TO FIND THE MEAN OF THE ARRAY ELEMENTS

Mean is the average of the array elements. Here is the C code for finding the average of an integer array.
double Mean(int array[ ],int size)

```
double Mean(int array[],int size)
{
      double avg=0;
      int i=0;
      for(;i<size;i++)
            avg+=(double)array[i]/(double)(size);
      return avg;
}
```

Do you realize that this is nothing but 'Operations with External Constants' as discussed above?

1.19 HOW TO FIND WEIGHTED AVERAGE OF AN ARRAY OF NUMBERS

In the above case, equal weightage has been given to each array element. But in some situations we will have to calculate the weighted average of a set of numbers where each one of them will have a different weightage than the other. A common example is grading in exam papers. An important subject will be given more weightage than others. The formula for weighted average is

$$\text{Weighted average} = \frac{\text{(Summation of product of weightages and number associated with that weightage)}}{\text{Summation of weightages}}$$

Here is a C code that finds the weighted average of a student in 4 subjects, each of which has predefined weightages.

```
#include <stdio.h>
double WMean(double marks[],double weights[],int size)
{
      double MarksWeightageProductSum=0;
      double WeightageSum=0;
      int i=0;
 for(i=0;i<size;i++)
 {
      MarksWeightageProductSum+=marks[i]*weights[i];
      WeightageSum+=weights[i];
  }
   return MarksWeightageProductSum/WeightageSum;
}
```

```
int main()
{
      double w[]={1,2,3,4};
      double m[]={100,78,89,78};
      printf("%f\n",WMean(m,w,4));
      getch();
      return 0;
}
```

1.20 HOW TO FIND THE MEDIAN OF THE ARRAY ELEMENTS WHICH ARE ALREADY SORTED

```
#include <stdio.h>

float median(float array[],int size)
{
     int n=0;
     n=size/2;
     if(size%2!=0)
        return array[n];
     else
        return (array[n]+array[n+1])/2;
}

int main()
{
       float array[9]={1,2,3,4,5,6,7,8,9};
       printf("%f\n",median(array,9));
       return 0;
}
```

1.21 HOW TO FIND THE MODE OF THE ARRAY ELEMENTS

The mode of a set of elements is the element that occurs for the maximum number of times.

So it is really easy to find the mode of an array of elements. As we need to find the frequency of each element in the array, a separate method is written. The method below returns the frequency of a number in an array of elements.

```
int Count(int a[],int size,int x)
{
      int i,frequency=0;
      for(i=0;i<size;i++)
      {
            if(a[i]==x)
                  frequency++;
      }
      return frequency;
}
```

Now this method will be used to find the mode of an array of elements. Here is the C code.

```
int Mode(int a[],int size)
{
      int temp=0;
      int i;
      temp = a[0];
      for(i=0;i<size;i++)
      {
            if(Count(a,size,a[i])<Count(a,size,a[i+1]))
                  temp = a[i+1];
      }
      return temp;
}
```

So here, the frequency of each element is checked and then the element that has the maximum frequency is returned, which is the mode of the array.

1.22 HOW TO FIND THE RANGE OF THE ARRAY ELEMENTS

To find the range of a set of *n* numbers, the smallest number is subtracted from the largest number. This measures how widely the numbers are dispersed. For example, the range of

4, 3, 8,12,23,37 is 37 – 3 = 34

Here is the code that finds the range of an integer array.

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

int FindMaximumNumber(int array[],int size)
{
      //See code above
}
int FindMinimumNumber(int array[],int size)
{
      //see code above
}
int main()
{
      int a[]={123,121,245,365,2,155};
      printf("Range is %d\n",
            FindMaximumNumber(a,6)- FindMinimumNumber(a,6));
      getch();
}
```

1.23 HOW TO FIND STANDARD DEVIATION OF AN ARRAY

Another way to measure the dispersion of a set of numbers is **standard deviation** which measures the distance between the arithmetic mean and the set of numbers. To calculate the standard deviation of a set of numbers, first the average is found and then the average is subtracted from each element and then their difference is squared and then the squared differences are summed up. Standard deviation is the square root of the average of the squared differences.

Here is the code to find the standard deviation, for which the method to find mean (defined above) will be used.

```
//This program finds the standard deviation of a set of numbers
#include <stdio.h>
#include <math.h>
#include <conio.h>

//This method returns the Average of the numbers
double Mean(double array[],int size)
{
      int i=0;
      double sum=0;
      for(;i<size;i++)
            sum+=array[i];
      return sum/size;
}
//This method returns the addition of the numbers
double Add(double array[],int start,int finish)
{
      int i=start;
      double sum = 0;
      for(;i<=finish;i++)
            sum+=array[i];
      return sum;
}

int main()
{
      int i=0;
      double a[]={1,2,3,4};
      double b[5];
      double MeanOfNumbers = Mean(a,4);
      for(i=0;i<4;i++)
      {
            double x = a[i]-MeanOfNumbers;
            b[i]=pow(x,2);
      }
      printf("Standard Deviation is =  %f\n",sqrt(Add(b,0,3)/3));
      getch();
      return 0;
}
```

1.24 HOW TO FIND THE VARIANCE OF THE ARRAY ELEMENTS

The **variance** of a collection of items is the square of the standard deviation.

```
printf("Variance of the array is =  %f\n",Add(b,0,3)/3);
```

Compare the bold line entry in the above two lines of code. Here, we are passing the calculated standard deviation to the outer `sqrt()` method.

1.25 HOW TO FIND AN INTERPOLATED VALUE USING NEWTON'S FORWARD DIFFERENCE INTERPOLATION

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

int fact(int n)
{

    if(n==1)
          return 1;
    else
          return fact(n-1)*n;
}

float calculatex(float height,int x)
{
       float nr=1;
       int k;
       for(k=0;k<=x;k++)
           nr*=(float)(height-k);
       if(x==0)
           return nr;
       else
           return nr/(float)fact(x+1);
}

int diff(int y[],int m)
{
         return y[m+1]-y[m];
}

int main()
{
                            int x[20];
                            int y[20];
                            int i=0;
                            int j=0;
                            int k=0;
                            int noe=0;
                            static int count=0;
                            int dx=0;
                            static float sy = 0;
                            static float height = 0.0;
                            printf("No of observations :");
                            scanf("%d",&noe);
                            printf("Determination Point :");
                            scanf("%d",&dx);
                            for(i=0;i<noe;i++)
                            {
```

```
printf("Enter independent variable :");
 scanf("%d",&x[i]);
printf("Enter the dependent variable :");
 scanf("%d",&y[i]);
}
sy+=y[0];
 height = (float)(dx - x[0])/(float)noe;
for(i=0;i<noe-1;i++)
                        {
                           for(j=0,k=0;k<noe-i-1;j++,k++)
                           {
                          y[k]=diff(y,j);
                            }
                           sy+=calculatex(height,i)*y[0];
                        }
                        printf("Value at %d is %f\n",dx,sy);
                        return 0;
                }
```

1.26 HOW TO INTERPOLATE USING LAGRANGE'S INTERPOLATION FORMULA

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

int calculateNr(int x[],int size,int dx,int n)
{
         int result=1;
         int i;

         for(i=0;i<size;i++)
         {
                  if(n!=i)
                  {
                           result*=(dx-x[i]);
                  }

                else
                           result*=1;

                }
                return result;
}
```

```
int calculateDr(int x[],int size,int k)
{
             int result=1;
             int i;

             for(i=0;i<size;i++)
```

```
{
      if(i!=k)
      {

      }

       else
                result*=1;
             }
      return result;
}

int main()
{
                              int x[20];
                              int y[20];
                              int i=0;
                              int j=0;
                              int k=0;
                              int noe=0;
                              static int count=0;
                              int dx=0;
                              static float sy = 0;
                              static float height = 0.0;
                              printf("No of observations :");
                              scanf("%d",&noe);
                              printf("Determination Point :");
                              scanf("%d",&dx);
                              for(i=0;i<noe;i++)
                              {
                              printf("Enter independent variable :");
                               scanf("%d",&x[i]);
                              printf("Enter the dependent variable :");
                               scanf("%d",&y[i]);
                              }

                              for(i=0;i<noe;i++)
                              {

                              sy+=y[i]*((float)calculateNr(x,noe,dx,i)/
                                    (float)calculateDr(x,noe,i));

                              }
                              printf("%f\n",sy);
}
```

1.27 HOW TO FIND A REGRESSION LINE ON *X* OR *Y*

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

float x[100];
float y[100];

int main()
{
                                int NoElements;
                                float Sum_X=0;
                                float Sum_Y=0;
                                float Sum_XY=0;
                                float Sum_sqrx=0;
                                float Sum_sqry=0;
                                int i=0;
                                float a;
                                float b;
                                printf("How many elements :");
                                scanf("%d",&NoElements);
                                printf("Enter the values for independent
                                variable :");

                                for(i=0;i<NoElements;i++)
                                {
                                 scanf("%f",&x[i]);
                                    Sum_X +=x[i];
                                Sum_sqrx+=(float)pow(x[i],2);
                                }

                                printf("Enter the values for dependent
                                variable :");
                                for(i=0;i<NoElements;i++)
                                {
                                 scanf("%f",&y[i]);
                                    Sum_Y+=y[i];
                                Sum_sqry+=(float)pow(y[i],2);
                                }

                                for(i=0;i<NoElements;i++)
                                 Sum_XY=x[i]*y[i];

                                b = (NoElements*Sum_XY - Sum_X*Sum_Y)
                                     /((NoElements - 1)*Sum_sqry);
                                a = (Sum_XY - b*Sum_sqry)/Sum_Y;

                                printf("%f %f\n",a,b);

                                return 0;

}
```

1.28 HOW TO FIND SIMPLE AGGREGATION INDEX NUMBER

This is a statistic which assigns a single number to several individual statistics in order to quantify trends. The best-known index in the United States is the consumer price index, which gives a sort of 'average' value for inflation based on price changes for a group of selected products. The Dow Jones and NASDAQ indexes for the New York and American Stock Exchanges, respectively, are also index numbers.

Let p_n be the price per unit in period n, q_n be the quantity produced in period n, and $v_n \equiv p_n q_n$ be the value of the n units. Let q_a be the estimated relative importance of a product. There are several types of indices defined, among them are those listed in the following table.

Index	*Abbr.*	*Formula*
Bowley index	P_B	$\frac{1}{2}(P_L + P_P)$
Fisher index	P_F	$\sqrt{P_L P_P}$
Geometric mean index	P_G	$\left[\Pi\left(\frac{p_n}{p_D}\right)^{v_D}\right]^{1/\Sigma v_D}$
Harmonic mean index	P_H	$\frac{\sum p_n q_n}{\sum \frac{p_D^2 q_n}{p_n}}$
Laspeyres' index	P_L	$\frac{\sum p_n q_n}{\sum p_D q_n}$
Marshall–Edgeworth index	P_{ME}	$\frac{\sum p_n (q_D + q_n)}{\sum (v_D + v_n)}$
Mitchell index	P_M	$\frac{\sum p_n q_n}{\sum p_D q_n}$
Paasche's index	P_P	$\frac{\sum p_n q_n}{\sum p_D q_n}$
Walsh index	P_W	$\frac{\sum \sqrt{q_D q_n}\, p_n}{\sum \sqrt{q_D q_n}\, p_n}$

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

int previous[20];
int next[20];
int i;
```

```
int noElements;
int Sum_N=0;
int Sum_D=0;

int main()
{
       printf("Enter the number of entries :");
       scanf("%d",&noElements);
       for(i=0;i<noElements;i++)
       {
             printf("Enter previous value :");
             scanf("%d",&previous[i]);
             Sum_P+=previous[i];
             printf("Enter next value :");
             scanf("%d",&next[i]);
             Sum_N+=next[i];
       }


       printf("Simple Aggregation Index number is
              :%f\n",(float)Sum_N/(float)Sum_P);

    return 0;

}
```

1.29 HOW TO FIND THE SIMPLE AVERAGE OF A PRICE-RELATIVE INDEX

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

int previous[20];
int next[20];
int i;
int noElements;
int Sum_N=0;
int Sum_D=0;
int Sum=0;

int main()
{
       printf("Enter the number of entries :");
       scanf("%d",&noElements);
       for(i=0;i<noElements;i++)
       {
             printf("Enter previous value :");
             scanf("%d",&previous[i]);
             printf("Enter next value :");
             scanf("%d",&next[i]);
             Sum+=(float)next[i]/(float)previous[i];
       }
```

```
    printf("Simple Average of Price Relative Index is
             :%f\n",Sum/noElements);

    return 0;

}
```

1.30 HOW TO FIND LASPEYRE'S INDEX NUMBER

```
float Laspeyres()
{
      for(i=0;i<noElements;i++)
      {
            printf("Enter previous value :");
            scanf("%d",&previous[i]);
            printf("Enter next value :");
            scanf("%d",&next[i]);
            printf("Enter the previous quantity :");
            scanf("%d",&pquantity[i]);
            printf("Enter the next quantity :");
            scanf("%d",&nquantity[i]);
            Sum_N+=next[i]*pquantity[i];
            Sum_D+=previous[i]*pquantity[i];
       }
        return (float)Sum_N/(float)Sum_D;

}
```

1.31 HOW TO FIND PAASCHE'S INDEX NUMBER

```
float Paasche()
{
      printf("Enter previous value :");
      scanf("%d",&previous[i]);
      printf("Enter next value :");
      scanf("%d",&next[i]);
      printf("Enter the previous quantity :");
      scanf("%d",&pquantity[i]);
      printf("Enter the next quantity :");
      scanf("%d",&nquantity[i]);
      Sum_N+=next[i]*pquantity[i];
      Sum_D+=previous[i]*nquantity[i];
      return (float)Sum_N/(float)Sum_D;
}
```

1.32 HOW TO FIND BOWLEY'S INDEX NUMBER

Bowley's index number is the average of Laspeyre's and Paasche's index numbers.

```
float Bowley()
{
       return (Laspeyres()+Paasche())*0.5;
}
```

1.33 HOW TO FIND FISHER'S INDEX NUMBER

Fisher's index number is nothing but the geometric mean of Laspeyre's and Paasche's index numbers.

```
float Fisher()
{
        return sqrt(Laspeyres()*Paasche());
}
```

1.34 HOW TO FIND MARSHALL–EDWARD INDEX NUMBER

```
float MarshallEdward()
{
      printf("Enter previous value :");
      scanf("%d",&previous[i]);
      printf("Enter next value :");
      scanf("%d",&next[i]);
      printf("Enter the previous quantity :");
      scanf("%d",&pquantity[i]);
      printf("Enter the next quantity :");
      scanf("%d",&nquantity[i]);
      Sum_N+=(pquantity[i]+nquantity[i])*next[i];
      Sum_D+=(pquantity[i]+nquantity[i])*previous[i];
      return (float)Sum_N/(float)Sum_D;
}
```

1.35 HOW TO REPRESENT A MATRIX USING 2D ARRAYS

We know that a matrix has rows and columns. So we can map them as a 2D array programmatically. This is shown below. Say, for example, we have a matrix like

1	3	5
4	5	1
1	4	6

Then this matrix can be written in the form of a 2D integer array like

```
int AMatrix [3][3] = {1,3,5,4,5,1,1,4,6};
```

To increase the readability of the code, i.e., to make the array look more like a 3×3 matrix, we can write it like

```
int AMatrix[3][3]={
                    1,3,5,
                    4,5,1,
                    1,4,6
                  };
```

In all the problems below, a static array has been used. For any practical purposes where you don't know previously what will be the count, linked lists are a better choice than arrays. Arrays are, of course, easier to access and process than linked lists on the other hand, and are chosen when we have large volume of the same type of data.

1.36 HOW TO ADD TWO 3×3 MATRICES

Programmatically, as you can see, adding two matrices is nothing but adding two 2-dimensional array element by element. Here is the code to add two matrices.

```
void Add(int mat1[3][3], int mat2[3][3])
{
       int result[3][3];
       for(int i= 0;i<3;i++)
       {
            for(int j=0;j<3;j++)
                  result[i][j]=mat1[i][j]+mat2[i][j];
       }
      //Displaying the Sum here
       for(int i= 0;i<3;i++)
       {
            for(int j=0;j<3;j++)
                  printf("%d ",result[i][j]);
            printf("\n");
       }
}
```

This method of addition assumes that there are no errors in the dimensions for adding successfully. And this method can only add two 3 by 3 matrices. To make this method generic, we can declare the arrays in the global section of the program. See the code below.

```
void Add(int rows, int cols)
{
       for(int i= 0;i<rows;i++)
       {
            for(int j=0;j<cols;j++)
                  result[i][j]=mat1[i][j]+mat2[i][j];
       }
}
```

In this case all the three arrays, mat1, mat2 and result, are 2D arrays and they are declared global. Addition of matrices is very straightforward. Here, for a 2D array we need two loops as we have 2 dimensions. If we have a 3D array then we can add it by using 3 loops, and so on.

Can you think of a situation where we will need a 3D array or even more dimensions?

1.37 HOW TO SUBTRACT TWO 3 × 3 MATRICES

Subtraction is the same as that of addition. Only the sign needs to be changed!

1.38 HOW TO MULTIPLY TWO MATRICES

Multiplication of two matrices is only possible if and only if the number of rows of one is same as that of the number of columns of the other. Here is the C code to multiply two 2D arrays or matrices.

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


int A[10][10];
int B[10][10];
int C[10][10];

int ar=0,ac=0,br=0,bc=0,cr=0,cc=0;
```

```
int i=0,j=0,k=0;
int main()
{
      printf("Enter Details about the first matrix :\n");
      printf("How many rows :");
      scanf("%d",&ar);
      printf("How many cols :");
      scanf("%d",&ac);
for(i=0;i<ar;i++)
{
      for(j=0;j<ac;j++)
      {
            printf("Enter element for row %d col %d :",i+1,j+1);
            scanf("%d",&A[i][j]);
      }
}

printf("Enter Details about the second matrix :\n");
printf("How many rows :");
scanf("%d",&br);
printf("How many cols :");
scanf("%d",&bc);
for(i=0;i<br;i++)
{
      for(j=0;j<bc;j++)
      {
            printf("Enter element for row %d col %d :",i+1,j+1);
            scanf("%d",&B[i][j]);
      }
}

if(ac!=br)
      printf("Dimensions don't match.\n");
else
{
      cr = ar;
      cc = bc;
      //loop control
      for( i = 0; i < ar; i++)
            for( j = 0; j < bc; j++)
                  for( k = 0; k < ac; k++)
                              C[i][j]+=A[i][k]*B[k][j];

}
for(i=0;i<cr;i++)
{
      for(j=0;j<cc;j++)
            printf("%d ",C[i][j]);
      printf("\n");
}

      getch();
      return 0;
}
```

1.39 HOW TO CALCULATE REVENUES USING MATRIX MULTIPLICATION

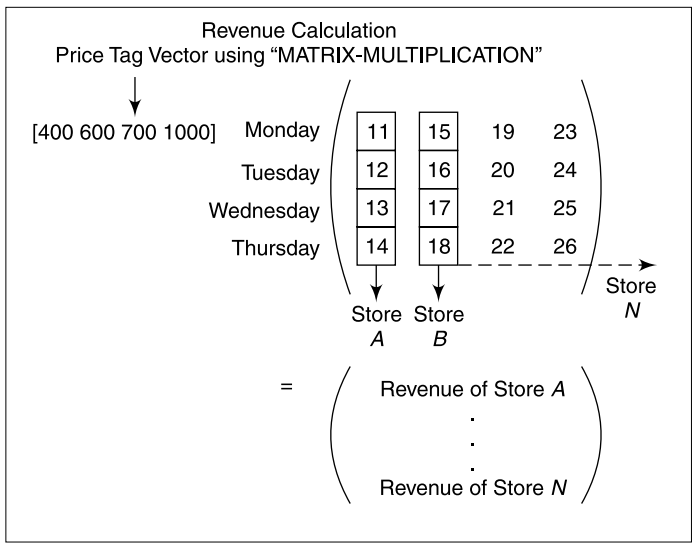


Fig. 1.4

1.40 MULTIPLICATION OF TWO 2 × 2 MATRICES USING STRASSEN'S ALGORITHM WHICH USES 7 MULTIPLICATIONS AND 18 ADDITIONS

Strassen's algorithm for matrix multiplication is based on a recursive divide and conquer scheme. Given n by n matrices A and B we wish to calculate $C = AB$. To see how this algorithm works, we first divide the matrices as follows:

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

It is assumed that each block is square. This can be achieved by padding the matrices. Strassen showed how C can be computed using only 7 block multiplications and 18 block additions:

$$P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$P_2 = (A_{21} + A_{22}) * B_{11}$$
$$P_3 = A_{11} * (B_{12} - B_{22})$$
$$P_4 = A_{22} * (B_{21} - B_{11})$$
$$P_5 = (A_{11} + A_{12}) * B_{22}$$
$$P_6 = (A_{21} - A_{11}) * (B_{11} + B_{12})$$
$$P_7 = (A_{12} - A_{22}) * (B_{21} + B_{22})$$
$$C_{11} = P_1 + P_4 - P_5 + P_7$$
$$C_{12} = P_3 + P_5$$
$$C_{21} = P_2 + P_4$$
$$C_{22} = P_1 + P_3 - P_2 + P_6$$

Here is the C function to calculate the product.

```
void StrassenMult ()
{
      int P1=(A[0][0]+A[0][1])*(B[0][1]+B[1][0]);
      int P2=(A[1][0]+A[1][1])*B[0][0];
      int P3=(B[0][1]-B[1][1])*A[0][0];
      int P4=(B[1][0]-B[0][0])*A[1][1];
      int P5=(A[0][0]-A[0][1])*B[1][1];
      int P6=(A[1][0]-A[1][1])*(B[0][0]+B[0][1]);
      int P7=(A[0][1]-A[1][1])*(B[1][0]+B[1][1]);
      int C11=P1+P4-P5+P7;
      int C12=P3+P5;
      int C21=P2+P5;
      int C22=P1+P3-P2+P6;
      printf("%d %d\n%d %d\n",C11,C12,C21,C22);

}
```

This algorithm is not popular because of the following reasons.

- This algorithm is not numerically stable.
- This algorithm demands more space.

Coppersmith and Winograd's algorithm is the best known matrix multiplication algorithm which uses arithmetic progression.

1.41 HOW TO FIND THE HADAMARD PRODUCT OF TWO MATRICES

If two matrices are of same dimensions, then apart from the normal matrix multiplication, there is another product called Hadamard product or Entrywise product. The Hadamard product of two matrices of order $m \times n$ is denoted by

$A \cdot B$ which is also an $m \times n$ matrix. Every element in the product matrix is nothing but the product of that element in the component matrices. That means $(A \cdot B)[i][j] = A[i][j] * B[i][j]$

Given below is an example.

Hadamard Product

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11} \cdot b_{11} & a_{12} \cdot b_{12} & a_{13} \cdot b_{13} \\ a_{21} \cdot b_{21} & a_{22} \cdot b_{22} & a_{23} \cdot b_{23} \\ a_{31} \cdot b_{31} & a_{32} \cdot b_{32} & a_{33} \cdot b_{33} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 4 & 2 \\ 2 & 0 & 0 \\ 3 & 5 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 \\ 3 & 1 & 0 \\ 2 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 1 \times 1 & 4 \times 2 & 2 \times 4 \\ 2 \times 3 & 0 \times 1 & 0 \times 0 \\ 3 \times 2 & 5 \times 0 & 2 \times 3 \end{bmatrix} = \begin{bmatrix} 1 & 8 & 8 \\ 6 & 0 & 0 \\ 6 & 0 & 6 \end{bmatrix}$$

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

int A[10][10];
int B[10][10];
int rows=0,cols=0;
int Hadamard[10][10];
```

```
void scanmatrices()
{
      int i,j;
      for(i=0;i<rows;i++)
            for(j=0;j<cols;j++)
            {
                  printf("A[%d][%d] = ",i+1,j+1);
                  scanf("%d",&A[i][j]);
                  printf("B[%d][%d] = ",i+1,j+1);
                  scanf("%d",&B[i][j]);
            }
}
```

```
void displayHadamard()
{
      int i,j;
      for(i=0;i<rows;i++)
      {
            for(j=0;j<cols;j++)
                  printf("%d",A[i][j]*B[i][j]);
            printf("\n");
      }
}

int main()
{
      printf("Enter number of rows :");
      scanf("%d",&rows);
      printf("Enter number of columns :");
      scanf("%d",&cols);
      scanmatrices();
      displayHadamard();
      return 0;
}
```

1.42 HOW TO FIND THE KRONECKER PRODUCT OF TWO MATRICES

This product also known as Tensor product or Direct Matrix Product and has heavy applications in System Theory.

The Kronecker product of two matrices is a block matrix which is calculated using the following rule as shown in the figure below.

If A is an $m \times m$ matrix and B is an $n \times n$ matrix then their Kronecker product will yield a matrix of $mn \times mn$. This product is also known as Tensor Product or "Direct Product".

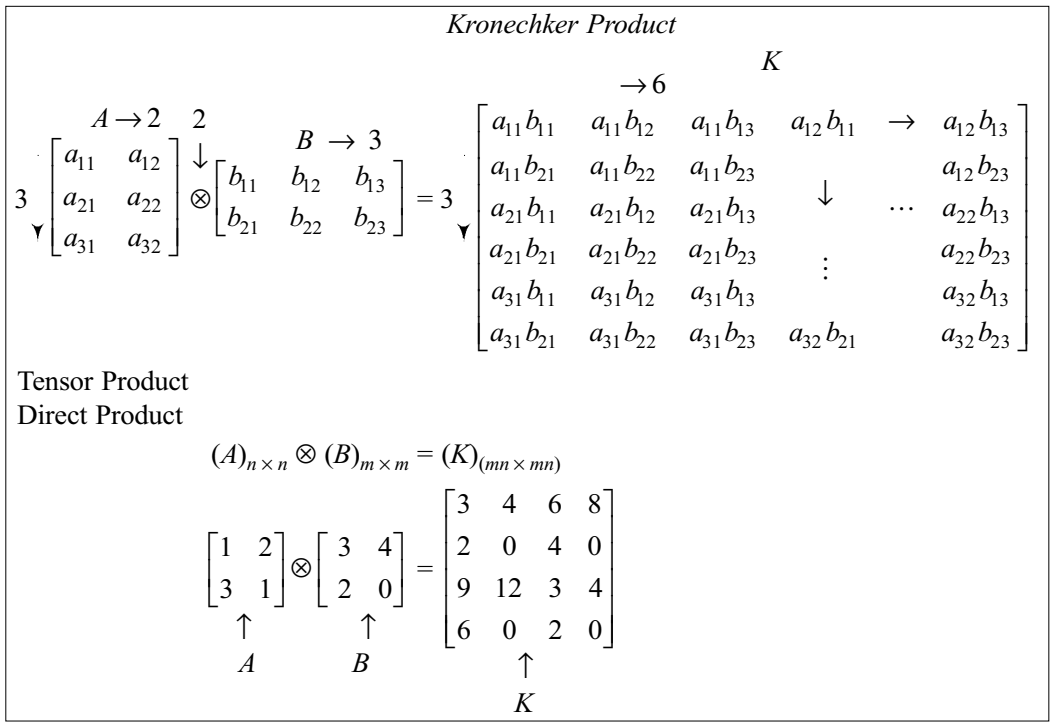


Fig. 1.5

//Arrays declared above are used here also.

```
void KroneckerProduct()
{
      //As because all the variables are available
      //globally then no need to declare them again
      int i,j,k,l,alpha,beta;
      int kmr = ar*br;
      int kmc = ac*bc;

      for(i=0;i<ar;i++)
            for(j=0;j<ac;j++)
                  for(k=0;k<br;k++)
                        for(l=0;l<bc;l++)
                        {
                              alpha=br*(i-1)+k+br;
                              beta =bc*(j-1)+l+bc;
                              C[alpha][beta]=A[i][j]*B[k][l];
                        }
      //Displaying the Kronecker Product of two matrices.
       for(i=0;i<kmr;i++)
       {
             for(j=0;j<kmc;j++)
                   printf("%d",C[i][j]);
             printf("\n");
       }
}
```

Although we discussed Hadamard and Kronecker products in line with matrix multiplication, don't have the misconception that they are the same as matrix multiplication.

1.43 HOW TO FIND THE TRANSPOSE OF A MATRIX

The transpose of a matrix is a matrix formed, from the original where the rows of the original matrix are columns of the transpose matrix. Say, for example, we have a 3 by 3 matrix like

1 2 3
6 7 8
4 5 9

Then its transpose will be

1 6 4
2 7 5
3 8 9

Let's examine this phenomenon closely from a programming point of view. Let's call the original array as *M* and the transpose matrix as *MT*. Then *M*[1][0] = 6 and in the transpose of this matrix we find that *MT*[0][1] = 6. So we can conclude that programmatically, transposing a matrix is just swapping the index variables. Technically speaking,

```
MT[Row][Column] = M[Column][Row];
```

Here is a C code that finds the transpose of a matrix.

```
void show_transpose(float mat[][10],int row,int col)
{
     int i,j;
      for(i=0;i<row;i++)
      {
            for(j=0;j<col;j++)
                  printf("%f\t",mat[j][i]);
      printf("\n");
      }
}
```

This method displays the transpose of a matrix.

1.44 HOW TO FIND THE INVERSE OF A SQUARE MATRIX

This is one of the tough problems programmatically to be done on a matrix. There are many algorithms to find the inverse of a matrix. Here, Gaussian Elimination Technique is used. The other methods include LU Decomposition, Matrix Equation, etc. Here is the C code to find the inverse of a matrix and display it also.

```
void Inverse(float mat[][10],int r,int c)
{
      float b[10][10],ratio,a[10][10];
int i,j, k;
if(r==c)
{
clrscr();
for (i=0;i<r;i++)
{
      for(j=0;j<c;j++)
```

```
        {
         //Copying values of mat into a
              a[i][j]=mat[i][j];
              b[i][j]=0;
        }
        b[i][i]=1;
}
for(k=0;k<r;k++)
      {
            for(i=0;i<c;i++)
            {
                  if(i==k)
                        continue;
                  else
                  {
                  ratio=a[i][k]/a[k][k];
                  for(j=0;j<c;j++)
                  {
                        a[i][j]-=ratio*a[k][j];
                        b[i][j]-=ratio*b[k][j];
                  }
                  }
            }
      }
      for(i=0;i<r;i++)
      {
            for(j=0;j<c;j++)
                  b[i][j]/=a[i][i];
      }
      for(i=0;i<r;i++)
      {
            for(j=0;j<c;j++)
                  printf("%f\t" ,b[i][j]);
            printf("\n");
      }
}
```

1.45 HOW TO FIND THE UPPER TRIANGULAR OF A MATRIX

Triangular matrices make many matrix-related calculations easy. There can be two types of triangular matrices—upper triangular matrix and lower triangular matrix. In an upper triangular matrix, the elements above the main diagonal remain as they are and the elements below the diagonal are zero. An upper triangular matrix can be formed from a matrix very easily. Let's have a close look at the matrix.

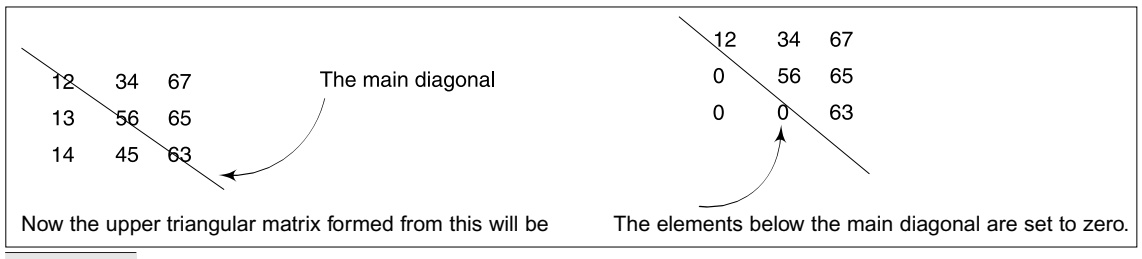


Fig. 1.6

Let's name the 2D array that is representing the upper diagonal of this matrix, MyMatrix. So programmatically, in MyMatrix

MyMatrix [0][0] = 12
MyMatrix [0][1] = 34
MyMatrix [0][2] = 67
MyMatrix [1][0] = 0
MyMatrix [1][1] = 56
MyMatrix [1][2] = 65
MyMatrix [2][0] = 0
MyMatrix [2][1] = 0
MyMatrix [2][2] = 63

Have a close look at the bold lines above. Can you see that columns are less than rows? So the strategy to find the upper triangular matrix from a given matrix is to set all those elements whose rows are greater than their columns to zero. Here it is achieved programmatically in C.

```
void triu(int matrix[][3],int rows,int cols)
{
      int i=0;
      int j=0;
      for(;i<rows;i++)
      {
            for(j=0;j<cols;j++)
            {
                  if(j<i)
                        printf("0 ");

                  else
                        printf("%d ",matrix[i][j]);
                  }
            printf("\n");
    }

}
```

1.46 HOW TO FIND A STRICT UPPER TRIANGULAR MATRIX

A strict upper triangular matrix is an upper triangular matrix in which the main diagonal elements are also zero. So just changing an operator in the above code can do this. Here is the code to find the strict upper triangular matrix.

```
void strict_triu(int matrix[][3],int rows, int cols)
{
      int i=0;
      int j=0;
      for(;i<rows;i++)
      {
            for(j=0;j<cols;j++)
            {
            //For including the Diagonal Elements
```

```
//the  operator has been changed to
//<= because for diagonal elements the row index
//and the column index are same.
      if(j<=i)
            printf("0");
      else
            printf("%d",matrix[i][j]);
            }
            printf("\n");
    }

}
```

1.47 HOW TO FIND THE LOWER TRIANGULAR OF A MATRIX

Finding the lower triangular matrix is similar to that of finding the upper triangular matrix. Except in this case the elements above the main diagonal will be zero. Here is the code to find the lower triangular matrix.

```
void tril(int matrix[][3],int rows,int cols)
{
      int i=0;
      int j=0;
      for(;i<rows;i++)
      {
            for(j=0;j<cols;j++)
            {
                  if(j>i)
                        printf("0");
                  else
                        printf("%d",matrix[i][j]);

            }
            printf("\n");
    }

}
```

1.48 HOW TO FIND A STRICT LOWER TRIANGULAR MATRIX

Like in the case of upper triangular matrix, here also the diagonal elements along with the elements above it are made zero. Here is the code to make a matrix a strict lower triangular matrix.

```
void strict_tril(int matrix[][3],int rows,int cols)
{
int i=0;
int j=0;
for(;i<rows;i++)
{
      for(j=0;j<cols;j++)
      {
      //For including the Diagonal Elements
```

```
            //the  operator has been changed to
            //>= because for diagonal elements the row index
            //and the column index are same.
                  if(j>=i)
                        printf("0");
                  else
                        printf("%d",matrix[i][j]);
                  }
                  printf("\n");
            }

}
```

1.49 HOW TO CREATE A TOEPLITZ MATRIX FROM A GIVEN ROW AND COLUMN

If the elements of a matrix are all constants and show a particular symmetry where all elements on a diagonal parallel to the main one of the matrix are same, then that matrix is known as a **Toeplitz matrix** after its discoverer Otto Toeplitz. Here is an example.

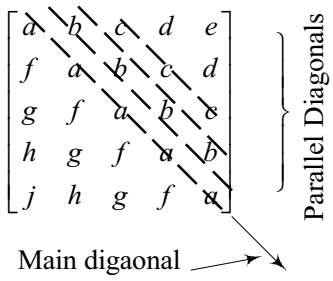


Fig. 1.7

Given a row and a column vector, a Toeplitz matrix can be constructed by the following rule.

```
C[i][j]  = cols[i-j+1] where i-j>=0
C[i][j]  = rows[j-i+1] where j-i>0
```

This code creates and prints the Toeplitz matrix which is created from a given row and column vector.

```
void  Toeplitz(int rows[10],int cols[10],int r,int c)
{

   for(i=0;i<r;i++)
     for(j=0;j<c;j++)
     {
        if(i-j>=0)
            T[i][j]=cols[i-j+1];
        if(j-i>0)
            T[i][j]=rows[j-i+1];
     }
   for(i=0;i<r;i++)
   {
```

```
    {
        for(j=0;j<c;j++)
        printf("%d ",T[i][j]);
        printf("\n");
     }

}
```

1.50 HOW TO FIND WHETHER A MATRIX IS SYMMETRIC OR NOT

A matrix is said to be symmetrical if the element at (i, j) location is same as that of at (j, i) location. This code below finds out whether the given matrix is symmetric or not.

```
enum {NO,YES};

int IsSymmetricMatrix(int Matrix[][10],int rows,int cols)
{
      int status=YES;
      for(i=0;i<rows;i++)
      for(j=0;j<cols;j++)
      {
          if(Matrix[i][j]!=Matrix[j][i])
          {
                 status = NO;
                 break;
          }
       }
return status;
}
```

1.51 REPRESENTING A SPARSE MATRIX AS ARRAYS

A **sparse matrix** is a matrix of whose 1/3 elements are only non-zero and rest are all zero. So to represent this type of matrix if we use the traditional array then there will be a problem regarding space. There are many ways to store the sparse matrix. In almost every way there are three entries. One for rows, one for columns and the other for elements. Normally, structures are used to represent sparse matrix in C. Here is one such matrix.

```
      typedef struct element
      {
            int rowpos;
            int colpos;
            float val;
      }element;
      typedef struct spmat
      {
            int noe;
            int no_row;
            int no_col;
            element data[25];
    }spmat;
```

As you can see, typedef is used. So there is no need to name the same structure keyword again and again.

Here is the code to accept a sparse matrix that is stored in the following structures.

```
void InputSparse(spmat sp)
{
            int i,j,k,flag=0;
            clrscr();
            printf("How many rows :");
            scanf("%d",&sp.no_row);
            printf("How many columns :");
            scanf("%d",&sp.no_col);
            printf("How many nonzero elements :");
            scanf("%d",&sp.noe);
            for(i=0;i<sp.noe;i++)
            {
                  printf("Enter the row position :");
                  scanf("%d",&sp.data[i].rowpos);
                  printf("Enter the column position :");
                  scanf("%d",&sp.data[i].colpos);
                  printf("Enter the value in position [%d,%d] :",
                  sp.data[i].rowpos,sp.data[i].colpos);
                  scanf("%f",&sp.data[i].val);
            }
}
```

How to Add Two Sparse Matrices

```
/*---------------------
Adds two sparse matrix
----------------------*/
void add_spmat(spmat s1,spmat s2)
{
      int i=0,j=0,z,k,n;
      clrscr();
      if((s1.no_row!=s2.no_row)||(s1.no_col!=s2.no_col))
      {
            gotoxy(22,22);
            cprintf(" ERROR: PARAMETER MISMATCH...can't add");
            gotoxy(22,23);
            MES;
      }
      addresult.noe=s1.noe+s2.noe;
      for(k=0;k<addresult.noe;k++)
      {
            if(i>=s1.noe)
            {
                  n=0;
                  break;
            }
            if(j>=s2.noe)
            {
                  n=1;
```

```
      break;
}
z=check_add_spmat(s1.data[i].rowpos,s1.data[i].colpos,
s2.data[j].rowpos,s2.data[j].colpos);
if(z==0)
{
      addresult.data[k].val=s1.data[i].val+s2.data[i].val;
      addresult.data[k].rowpos=s1.data[i].rowpos;
      addresult.data[k].colpos=s1.data[i].colpos;
      i++;
      j++;
}
if(z==1 || z==3)
{
      addresult.data[k].val=s2.data[j].val;
      addresult.data[k].rowpos=s2.data[j].rowpos;
      addresult.data[k].colpos=s2.data[j].colpos;
      j++;
}
if(z==2 || z==4)
{
      addresult.data[k].val=s1.data[i].val;
      addresult.data[k].rowpos=s1.data[i].rowpos;
      addresult.data[k].colpos=s1.data[i].colpos;
      i++;
    }
}
if(n==0)
{
      for(j;j<s2.noe;j++)
      {
            addresult.data[k].val=s2.data[j].val;
            addresult.data[k].rowpos=s2.data[j].rowpos;
            addresult.data[j].colpos=s2.data[j].colpos;
            k++;
      }
}
if(n==1)
{
      for(i;i<s1.noe;i++)
      {
            addresult.data[k].val=s1.data[i].val;
            addresult.data[k].rowpos=s1.data[i].rowpos;
            addresult.data[k].colpos=s1.data[i].colpos;
            k++;
      }
}
      addresult.noe=k;
      addresult.no_row=s1.no_row;
      addresult.no_col=s1.no_col;
}
```

Here you may have noticed that a method called `check_add_spmat` () is called to find which sparse matrix dimension to use. Here is the definition for this method.

```
/*----------------------------------------
used to check which sparse has to be used
and whence to add the two sparse matrices.
----------------------------------------*/
int check_add_spmat(int a,int b,int c,int d)
{
       int result;
      if((a==0) && (b==d))
            result=0;
      if(a>c)
            result=1;
      if(a<c)
            result=2;
      if((a==c) && (b>d))
            result=3;
      if((a==c) && (b<d))
            result=4;
             return result;
}
```

1.52 3D ARRAY APPLICATIONS HOW TO USE A 3D ARRAY TO STORE AND MANIPULATE THE LITERACY DETAILS OF 5 CITIES AROUND A YEAR

One real-life example may be a demographic application where we want to find the literacy percentage in 10 cities and in 10 years. So we will store this information in a 3D array. First, we will store the years and then we will store the cities and then we will store the number of literate people in a city in that particular year. After we have this info, we can answer any query related to the literacy rates, like which city has the highest number of literate people in any year, or which city showed maximum increase in literacy percentage in the years and such other queries.

Here is the C code to achieve what we discussed above.

```
#include <stdio.h>

int main()
{
      int year=0;
      int city=0;
      int month=0;
      int WhichYear=0;
      int    Literacy[5][5][12];

      for(year=0;year<5;year++)
      {
            for(city=0;city<5;city++)
                  for(month=0;month<12;month++)
                  {
                printf("How many literate people were there in
                    %d in city_%d in month %d",year,city,month);
                 scanf("%d",&Literacy[year][city][month]);
                }
}
printf("Which Year you want to know about ?");
scanf("%d",&WhichYear);
for(city=0;city<5;city++)
      for(month=0;month<12;month++)
```

```
                printf("There were %d literate people in
                   %d in city_%d in month %d",
            Literacy[WhichYear][city][month],year,city,month);


    return 0;
}
```

Have you noticed how easily we can calculate the literacy rate of a city in any month of the given year? So whenever we have more than one unique dimension we should go for arrays. By unique dimension I mean independent variables. Like if you know the interdependency of two variables beforehand, there is no point to store them in two different arrays. In some books you will find that. But that is not a good practice.

In some chemical reactions, there can be 70 to 80 parameters that you have to keep track of. And to make things worse they are absolutely independent. So shall we use an array of dimension 70 then? Well that is one option but of course not the best. In that case you should use an array of structures. We will take up this in the chapter on structures.

1.53 HOW TO RETURN MORE THAN ONE VALUE FROM A FUNCTION

In C normally we can return only one value from a function. But there may be cases when we need to send more than one values from a function, we can use arrays very well. Here is an example. This program takes an integer array and returns the sum and their product of the elements to the calling function. Here is the C code. This is a trivial situation. But the technique is given below.

```
#include <stdio.h>
#include <conio.h>
int*  Pool(int array[],int size)
{
      int *x;
      int i=0;
      int a[2]={0,1};

      for(i=0;i<size;i++)
      {
            a[0]+=array[i];//Putting the Summation of array values
            a[1]*=array[i];//Putting the Product of array values
      }
     //Assigning the base address of the array to the
     //Integer Pointer
      x=&a[0];
     //Returning the whole array
      return x;
}

int main()
{
      int a[]={1,2,3,4};
     int *c;
     c = Pool(a,4);
      printf("Sum = %d\nProduct = %d\n",c[0],c[1]);
      getch();
      return 0;
}
```

So you have learnt how to return multiple values from a C function using arrays and pointer. There can be many situations where you can find this technique useful.

The next application is a sample use of this concept. The code is not fully written for this application. A part is left for the reader to code.

1.54 HOW TO CLONE STRING TOKENIZER CLASS OF JAVA

In Java there is a very useful class that allows separating all parts of a string with a specific delimiter. Say, for example, telephone number. You can enter the telephone number as **CountryCode-CityCode-AreaCode-Number**. Now let's imagine you are writing an application where you need to validate these codes. For that you need each of these codes extracted from the entire phone number. We can write a method which will take the phone number and the delimiter as input and will return an array of codes. The first of the array returned will be Country Code; the second one will be City Code, and so on.

```
char** ExtractCodesFromPhoneNumber(char *PhoneNumber,char delm)
{

     //Logic to Extract Codes. This part is left for the reader
      char *codes[]={"91","033","2671","2431"};
      char **p;
      p=&codes[0];
      return p;
}

int main()
{
      char phonenumber[17]="91-033-2671-2431";
      char **p=ExtractCodesFromPhoneNumber(phonenumber,'-');
      printf("%s\n%s\n%s\n%s\n",*p,*(p+1),*(p+2),*(p+3));
      getch();
       return 0;
}
```

`ExtractCodesFromPhoneNumber()` method returns a pointer to a 2D character array. p is a pointer to the array of codes obtained from the passed PhoneNumber. Now the values are returned from the method to `main()`.

There can be many other uses of this utility. Say, for example, you are given the job to find out how many people in Kolkata have an email account with yahoo. Then what can you do? You can email a text document writing your email to your friends asking them to forward it to as many people as they know.

Each recipient will have to write their email in that file. After that the last recipient will send it back to you. Now you read that notepad file into a 2D character array and then use this utility with the delimiter as '@'. After that check how many tokens contain yahoo and you will get the count. Try it.

We will implement this method in the chapter on strings.

1.55 CONVERSION OF BINARY TO DECIMAL

Suppose you have a very big binary number like 10000111011001100111000101101l0 and you need to convert it to the equivalent decimal number. This may be difficult if you think of doing it the normal way. Let's store this number in an integer array and manipulate each bit so as to get the final answer.

Here is how we should approach this problem.

```
#include <stdio.h>
#include <math.h>//Used for pow()
#include <conio.h>

int BinaryToDecimal(int Binary[],int size)
{
      static int Sum=0;
      int i=0;
      for(i=0;i<size;i++)
            Sum+=pow(2,size-i-1)*Binary[i];
      return Sum;
}
int main()
{
      int Binary[]={1,0,1,0,1,0,1,1,0,0,1,1};
      printf("%d\n",BinaryToDecimal(Binary,12));
      getch();
      return 0;
}
```

1.56 HOW TO DESIGN A CHART FOR SHARE TRADING

Suppose, for example, you want to buy some shares. But before you want to buy the shares you want to be sure for yourself. So you decide to keep an eye on the values of the shares from different companies in different sectors for a few months before you actually invest your money in the share trading. So for this you need to maintain a chart which will give you the information instantly. A chart can be viewed as a multidimensional array from a programmer's point of view. Here is a C code to do this.

```
// Chart.cpp : Defines the entry point for the console application.
//

#include <stdio.h>

#define COMPANIES 7
#define SECTORS 5
#define MONTHS 12
#define SHARES 2

int main()
{
      char *companynames[]=
{"Microsoft","TCS","Infosys","Reliance","intel","Dell","IBM"};
      char *sectors[]=
{"Software","Learning","Development","Others"};
      char *months[]=
{"JAN","FEB","MAR","APR","MAY","JUN","JUL","AUG",
                        "SEP","OCT","NOV","DEC"};
int shares[COMPANIES][SECTORS][MONTHS][SHARES];
int co,se,mo,sh;
for(co=0;co<COMPANIES;co++)
{
      for(se=0;se<SECTORS;se++)
      {
            for(mo=0;mo<MONTHS;mo++)
            {
```

```
                        for(sh=0;sh<SHARES;sh++)
                        {
                              printf("Enter Share %d value
for\n",sh+1);
                              printf("%s in %s in %s sector :",
companynames[co],months[mo],sectors[se]);
                         scanf("%d",&shares[co][se][mo][sh]);
                        }
                  }
            }
            return 0;
}

//This part of the code reads from console the details of the
//shares for different companies and stores them in the 4D array.
//So you can access the details anytime
```

Suppose you want to know the share value for Reliance in the Learning Sector in October. Then it can be found easily, because the computer stores it in an easily accessible 4D array. Look at the Literacy Rate example above to find out how to do this. As an exercise you can add up the year with this array to keep a yearly track if you want. You can plug File Handling to save the details in a file permanently. Try it!

Companies	January				February				March				April …. >>			
Microsoft	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4

Fig. 1.8

The chart will look something like this where S1 and S2 denotes the price of shares in 1st sector, 2nd sector, and so on. You can add more parameters. This will be a lot easier when we finish structures.

1.57 HOW TO FIND HHI INDEX

HHI stands for Herfindahl–Hirschman Index. This is a commonly accepted measure of market concentration. It is calculated by squaring the market share of each firm competing in a market, and then summing the resulting numbers. The HHI number can range from close to zero to 10,000. The HHI is expressed as

HHI = s1^2 + s2^2 + s3^2 + ... + sn^2 (where sn is the market share of the ith firm).

The closer a market is to being a monopoly, the higher the market's concentration (and the lower its competition). If, for example, there were only one firm in an industry, that firm would have 100% market share, and the HHI would equal 10,000 (100^2), indicating a monopoly. Or, if there were thousands of firms competing, each would have nearly 0% market share, and the HHI would be close to zero, indicating nearly perfect competition.

We can use an array to check if a business sector is close to monopoly or not. And moreover we can tell which business company in that sector runs the monopoly service.

```
#include <conio.h>

long FindHHI(double x[],int size)
{
      long sum = 0;
      int i = 0;
      for(i=0;i<size;i++)
```

```
      {
            sum+=pow(x[i],2);
            printf("%d\n",sum);
      }
      return sum;

}
int main()
{
      double MarketShares[]={222,334,343,566,233};
      //Printing the HHI
      printf("HHI is %d",FindHHI(MarketShares,5));
      getch();
      return 0;
}
```

The US Department of Justice considers a market with a result of less than 1,000 to be a competitive marketplace; a result of 1,000–1,800 to be a moderately concentrated marketplace; and a result of 1,800 or greater to be a highly concentrated marketplace. As a general rule, mergers that increase the HHI by more than 100 points in concentrated markets raise antitrust concerns.

1.58 HOW TO FIND GINI COEFFICIENT MEASUREMENT FOR A CITY

To measure the inequality among the people of a city from one another, a coefficient is used named "Gini Coefficient". The Gini coefficient, invented by the Italian statistician Corado Gini, is a number between zero and one that measures the degree of inequality in the distribution of income in a given society. The coefficient would register zero (0.0 = minimum inequality) for a society in which each member received exactly the same income and it would register a coefficient of one (1.0 = maximum inequality) if one member got all the income and the rest got nothing...

The Gini coefficient (or Gini ratio) is a summary statistic of the Lorenz Curve and a measure of inequality in a population. The Gini coefficient is most easily calculated from unordered size data as the 'relative mean difference', i.e. the mean of the difference between every possible pair of individuals, divided by the mean size μ (mu),

$$G = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} \left| x_i - x_j \right|}{2n^2 \mu}$$

We can use arrays to calculate the Gini coefficient of a city. Here, *mu* denotes the mean income and *n* is the total population on which the survey is done.

```
double Gini(double array[],int size)
{
      int i=0;
      int j=0;
      double sum_1=0;
      double sum_2=0;
      double result=0;
      for(i=0;i<size;i++)
      {
            for(j=0;j<size;j++)
                  sum_1+=array[i]-array[j];//For the first level sum
        sum_2+=sum_1;//For the second level summation
      }
      result =abs(sum_2/(2*pow(size,2)*Mean(array,size)));
      return result;
}
```

The value returned will be anything between zero and one. If all the elements in the income array are made equal then Gini coefficient will be zero and that means wealth is equally distributed among all. Notice that `Mean()` method discussed earlier is used here. Try to reuse as many building blocks as possible. *To do that you need to write the methods in such a way that you can fit them with diverse requirements.*

1.59 HOW TO FIND WHETHER THREE GIVEN NUMBERS ARE IN AP, GP OR HP

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

enum status{NO,YES};

int isAP(float n[])
{
      int isitanAP;
      if(n[1]-n[0]==n[2]-n[1])
            isitanAP = YES;
      else
            isitanAP = NO;
      return isitanAP;
}


int isHP(float n[])
{
      int isitaHP;
      int cn[3];
      int i;
      for(i=0;i<3;i++)
            cn[i]=1/n[i];
      if(cn[1]-cn[0]==cn[2]-cn[1])
            isitaHP = YES;
      else
            isitaHP = NO;
      return isitaHP;
}

int isGP(float n[])
{
      int isitaGP;
      float r1 = n[1]/n[2];
      float r2 = n[0]/n[1];
      printf("%f %f",r1,r2);
      if(r1 == r2)
            isitaGP = YES;
      else
            isitaGP = NO;
      return isitaGP;
}
```

```
int main()
{
      float numbers[3];
      int i;
      for(i=0;i<3;i++)
      {
            printf("Enter number %d :",i+1);
            scanf("%f",&numbers[i]);
      }
      if(isAP(numbers))
      {
            printf("AP");
            printf("Common Difference :%f",numbers[1]-numbers[0]);
      }
      if(isHP(numbers))
            printf("HP");
      if(isGP(numbers))
      {
            printf("GP");
            printf("Common Ratio :%f",(numbers[0]/numbers[1]));
      }
      return 0;
}
```

1.60 ANIMATION OF DIFFERENT SIGNALING FORMATS

Example 1.1 ***Write a program to plot different signaling formats. Accept a bit stream (stream of 1 and 0) and then write the program to plot the following formats.***

Unipolar Non Return To Zero
Unipolar Return To Zero
Polar_Non Return To Zero
Polar Return To Zero
Manchester
Bipolar Non Return To Zero
Bipolar Return To Zero

/*Program plots signaling formats for a given bit stream*/

THIS PROGRAM WILL WORK ONLY IN TURBO C , UNDER DOS

Just copy and paste this code and run. This program has some characters that are non-printable without a compiler.

Solution

```
#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <conio.h>

void unipolar_nrz(char *,int);
void unipolar_rz(char *,int);
void polar_nrz(char *,int);
void polar_rz(char *,int);
void manchester(char *,int);
```

```
void bipolar_nrz(char *,int);
void bipolar_rz(char *,int);

int accept_bitstream(char *);

int main()
{
      char *bitstream;
      int size=0;
      size=accept_bitstream(bitstream);
      _setcursortype(_NOCURSOR);
      unipolar_nrz(bitstream,size);
      getch();
      unipolar_rz(bitstream,size);
      getch();
      polar_nrz(bitstream,size);
      getch();
      polar_rz(bitstream,size);
      getch();
      manchester(bitstream,size);
      getch();
      bipolar_nrz(bitstream,size);
      getch();
      bipolar_rz(bitstream,size);
      getch();
      return 0;
}

int accept_bitstream(char *bitstream)
{
      clrscr();
      printf("Enter a bitstream :");
      fflush(stdin);
      gets(bitstream);
      return strlen(bitstream);
}

void unipolar_rz(char bitstream[],int n)
{
      clrscr();
      gotoxy(1,2);
      printf("Unipolar Return to zero format for bitstream \n %s "
      ,bitstream);
      for(int i=0,k=0;k<n;i++,k++)
      {
            gotoxy(4+2*i,8);
            printf("%c",bitstream[k]);
if(bitstream[k]=='1')
{
      gotoxy(4+2*i,10);
      printf("%c",'ÄÄ');
}
else
{
```

```
                gotoxy(4+2*i,11);
                printf("ÄÄ");
            }
            delay(100);
      }

}
void unipolar_nrz(char bitstream[],int n)
{
      clrscr();
      gotoxy(1,2);
      printf("Unipolar Non return to zero format for bitstream \n%s"
      ,bitstream);
      for(int i=0,k=0;k<n;i++,k++)
      {
            gotoxy(4+2*i,8);
            printf("%c",bitstream[k]);
            if(bitstream[k]=='1')
            {
                  gotoxy(4+2*i,10);
                  printf("ÄÄ");
            }
            else
            {
                  gotoxy(4+2*i,11);
                  printf("ÄÄ");
            }
                        delay(100);
      }
}
void polar_nrz(char bitstream[],int n)
{
      clrscr();
      gotoxy(1,2);
      printf("Polar  Non return to zero format for bitstream \n %s ",
      bitstream);
      for(int i=0,k=0;k<n;i++,k++)
      {
            gotoxy(4+2*i,8);
            printf("%c",bitstream[k]);
            if(bitstream[k]=='1')
            {

                  gotoxy(4+2*i,10);
                  printf("ÄÄ");
            }
            else
            {
                  gotoxy(4+2*i,12);
                  printf("ÄÄ");
            }
                        delay(100);
      }
```

```
}
void polar_rz(char bitstream[],int n)
{
      clrscr();
      gotoxy(1,2);
      printf("Polar Return to zero format for bitstream \n %s ",
      bitstream);
      for(int i=0,k=0;k<n;i++,k++)
      {
            gotoxy(4+2*i,8);
            printf("%c",bitstream[k]);
            if(bitstream[k]=='1')
            {
                  gotoxy(4+2*i,10);
                  printf("Ä");
            }
            else
            {
                  gotoxy(4+2*i,12);
                  printf("Ä");
            }
                        delay(100);
      }
}
void manchester(char bitstream[],int n)
{
      clrscr();
      gotoxy(1,2);
      printf("Manchester format for bitstream \n %s "
      ,bitstream);
      for(int i=0,k=0;k<n;i++,k++)
      {
            gotoxy(4+2*i,8);
            printf("%c",bitstream[k]);
            if(bitstream[k]=='1')
            {
                  gotoxy(4+2*i,10);
                  printf("Ä");
                  gotoxy(5+2*i,12);
                  printf("Ä");
            }
            else
            {
                  gotoxy(4+2*i,12);
                  printf("Ä");
                  gotoxy(5+2*i,10);
                  printf("Ä");
            }
                        delay(100);
      }
}

/*Plots Bipolar Non return to zero format*/
void bipolar_nrz(char bitstream[],int n)
{
```

```
clrscr();
printf("Bipolar Non Return to Zero format for bitstream \n %s "
,bitstream);
int flag=0;
for(int k=0;k<n;k++)
{

      gotoxy(4+2*k,8);
      printf("%c",bitstream[k]);
      if(bitstream[k]=='1'&&flag==0)
      {
            gotoxy(4+2*k,10);
            printf("ÄÄ");
            flag=1;
            continue;
      }
      if(bitstream[k]=='0')
      {
            gotoxy(4+2*k,11);
            printf("ÄÄ");
      }
      if(bitstream[k]=='1'&&flag==1)
      {
            gotoxy(4+2*k,12);
            printf("ÄÄ");
            flag=0;
      }
                  delay(100);
}
}


void bipolar_rz(char bitstream[],int n)
{
      clrscr();
      printf("Bipolar Return to Zero format for bitstream \n %s "
      ,bitstream);
      int flag=0;
      for(int k=0;k<n;k++)
      {

            gotoxy(4+2*k,8);
            printf("%c",bitstream[k]);
            if(bitstream[k]=='1'&&flag==0)
            {
                  gotoxy(4+2*k,10);
                  printf("Ä");
                  flag=1;
                  continue;
            }
            if(bitstream[k]=='0')
            {
                  gotoxy(4+2*k,11);
                  printf("ÄÄ");
            }
```

```
        if(bitstream[k]=='1'&&flag==1)
        {
              gotoxy(4+2*k,12);
              printf("Ä");
              flag=0;
        }
        delay(100);
  }
```

Some screenshots of the above program output:

And so on..

Fig. 1.9

1.61 A WELL-KNOWN CRYPTOGRAPHIC TECHNIQUE—CIPHER TEXT

Cipher text is a popular encryption technique. What we do in cipher text is that we either move each character to the left or right. One example is "Hello". If it is encrypted with +1 Cipher then it will be "Ifmmp". This is normal Cipher text. We can have a customized cipher also. There we will have a predefined character for each. Like I = K, L = A, and so on. This selection of letters to represent other letters or this mapping can be absolutely random. Write a program to encrypt a given text to the plane +1 cipher text.

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>

char* cipher(char s[])
{
      char temp[200];
      int i;
      for(i=0;i<strlen(s);i++)
      {
            if(isaplha(s[i]))
            //Moving one letter towards right
                s[i]=toascii(s[i])+1;
      }

      return s;
}
```

```
int isaplha(char c)
{
      if((c>='a' && c<='z') || (c>='A' && c<='Z'))
            return 1;
      else
            return 0;
}
      int main()
      {
            char text[200]="I am an artist!! See my paintings!!";
            printf("%s",cipher(text));
            getch();
            return 0;
      }
```

1.62 DECODER PROGRAM FOR THE ABOVE ENCRYPTER

```
char* decipher(char s[])
{
      int i;
      for(i=0;i<strlen(s);i++)
      {
            if(isaplha(s[i]))
                  s[i]=toascii(s[i])-1;
      }

      return s;
}
```

See the bold line above. If you compare you can see, we increased the ASCII by unity when we encrypted the text. So while decrypting we decreased the ASCII of each character of the encrypted text by unity. This is called a linear cipher text. This is the simplest of the linear ciphers.

1.63 HOW TO FIND THE HISTOGRAM OF A 256 GRAY SCALE IMAGE

As you may know that in a 256 gray scale image, black is 0 and pure white is 255. The rest are within these two maxima. In the histogram of the given image, we find the frequency of all the pixel values present in the image.

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

int whichshade(int pixelvalue)
{
      int i=0;
      for(i=0;i<256;i++)
            if(pixelvalue==i)
                  break;
      return i;
}

//This method displays the Histogram in a Tabular Format.
//You can feed the same values to graphics software
```

```
//to see the plot. I have used Excel to plot the
void displaypixelstats(int pixelcounts[])
{
      int i=0;
      for(i=0;i<256;i++)
            printf("Gray Scale Shade : %d   Count :
%d\n",i,pixelcounts[i]);
}
int main()
{
      char file[20];
      int pixelvalue;
      //Container of Pixel Count
      int pixelcounts[255];
      FILE *fp;
      int i;
      //Initializing the container
      for(i=0;i<256;i++)
            pixelcounts[i]=0;
      printf("Enter the gray scale image file name :");
      fflush(stdin);
      scanf("%s",file);
      fp = fopen(file,"r");
      while(!feof(fp))
      {
            fscanf(fp,"%d",&pixelvalue);
            pixelcounts[whichshade(pixelvalue)]++;
      }
      fclose(fp);
      displaypixelstats(pixelcounts);
      getch();
      return 0;
}
```

1.64 HOW TO CONVERT A GRAY SCALE IMAGE TO BINARY IMAGE/ NEGATIVE IMAGE

Here, we ask the user to enter a particular threshold value and the binary image filename.

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

int main()
{
      int threshold=0;
      FILE *fpi;
      FILE *fpo;
      char iimage[20];
      char oimage[20];
      int pixelvalue=0;

      printf("Enter the input gray scale image name :");
      fflush(stdin);
      scanf("%s",iimage);

      printf("Enter the output binary image name :");
```

```
      fflush(stdin);
      scanf("%s",oimage);


      printf("Enter the Threshold Value :");
      scanf("%d",&threshold);

      fpi = fopen(iimage,"r");
      fpo = fopen(oimage,"w");
      while(!feof(fpi))
      {
            fscanf(fpi,"%d ",&pixelvalue);
            if(pixelvalue<threshold)
                  fprintf(fpo,"%d",0);
            else
                  fprintf(fpo,"%d",1);
      }
      printf("Converted to Binary!\n");
      fclose(fpi);
      fclose(fpo);
      return 0;
}
```

REVISION OF CONCEPTS

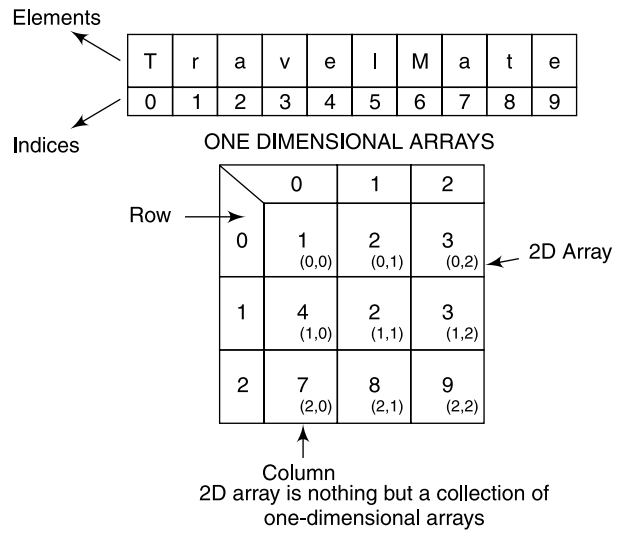


Fig. 1.10

Some Key Facts about Arrays

1. Arrays can hold only one type of elements. That means an integer array can only hold integers but can't hold a float value.

2. You can make an array for any data type, even pointers.
3. Array elements are stored in contiguous memory locations.
4. An array index start from zero.
5. Base address of the array is a pointer to the array itself.
6. Each element in an array can be accessed individually using either an index or pointer.
7. Array elements can be accessed by pointers faster.
8. Pointer access is faster than index access.
9. Arrays can be passed to function as arguments.
10. Whenever we just say array it refers to a one-dimensional array.
11. There is nothing called 2D array, it is an array of single-dimensional arrays.
12. In case of a 2D array the number of rows are not mandatory at the time of declaring the array.
13. 2D arrays are used to represent matrices.
14. Whenever the maximum number of elements to be processed is known at the beginning of the program, the best choice is an array, or an array-based data structure because of fastest access.

REVIEW QUESTIONS

1. What is wrong in the following declaration `int a[3][];`
2. Is the declaration `double a[][10];` valid ?
3. What are the constraints or limitations of an array?
4. What is the difference between a[5] and 5[a];
5. Given the following declaration: `int x[10][20];` Explain what each of the following represents:
   a. `x` b. `x + i` c. `*(x + i)`
   d. `*(x + i) + j` e. `*(*(x + i) + j)` f. `x[0]`
   g. `x[i]` h. `x[i] + j` i. `*(x[i] + j)`
6. What will this fetch *a(a + strlen(a) – 1);
7. What is the difference between strcpy() and strcat()
8. Can we achieve the same goal using any of the two ?
9. Is the following for loop valid? If no, then why ? If yes, then what does it do ?
   ```
   int counter=0;
   for(;array[counter]!='\0';counter++);
   ```
   Assume array is a character array.
10. Can we take an integer array as input using scanf()?
11. What do we need to print a character array in the console?
12. When do we use puts( ) over printf( )?
13. Which of the two printf() and puts() is more versatile and why?
14. Can we resize an array in C ?
15. Assume that you have a 4-dimensional array a[10][10][10][10]. What is the syntax to get the last element printed using index?
16. Use pointers for the above questions.
17. There is an integer array *a* and a float array *b*. What type of warning you will get when you try the following *(a+2) = *(b+1);
18. There is an integer array int a[]={1,2,3,4,5,6}; will the following line of code printf("%d",*(a–1)); compile ? If not why?

PROGRAMMING PROBLEMS

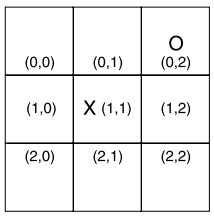
1. Write a C program to shuffle the numbers in an integer array, like, if the original contents are 1,2,3 then return an array that holds the shuffled array contents So the output will look like

{1, 2, 3}
{1, 3, 2}
{2, 1, 3}
{2, 3, 1}
{3, 1, 2}
{3, 2, 1}

2. Write a C program to convert a decimal number to BCD. See the link http://www.danbbs.dk/~erikoest/bcd.htm for more details regarding BCD.
3. Change the above program to find PACKED BCD using arrays.
4. Accept any 4 non-repetitive letters from users and generate all the possible combinations.
5. Create an array of 26 characters and then scan letters till the user presses Esc key. If the number of characters entered before Esc is hit is greater than 26 then clear the array and start filling it again from the beginning.
6. Write a program that converts a string to an integer using 2 arrays. For example, if we enter 4567 as string then it will convert it to an integer. C has a built in function for this called atoi( ) [*Array to integer*] what is asked here is to write your own definition for atoi().
7. Write a function rank() that returns the rank of a given matrix.
8. How do we find the determinant of a given matrix of dimension 3?
9. Write a function that returns the eigenvalues of a given matrix.
10. Write a program to find whether a matrix is a Monge matrix or not.
11. A Hankel matrix is a matrix where all the elements below the first antidiagonal of the matrix are zero. For example,

```
1 2 3 4
2 3 4 0
3 4 0 0
4 0 0 0
```

Is a Hankel matrix formed from the integer array [1, 2, 3, 4]. Write a program that will take an integer array as input and display the Hankel matrix.
12. Write a function eye() that accepts an integer and creates an identity matrix of the given size.
13. Imagine that you are recruited by the IT team in the weather office. You have to write a program that can help them to keep track of temperature and humidity in 10 metro cities round the year. How will you design it?
14. Is this a valid C array declaration int array[sizeof(double)];
15. Write a program to create a game interface for Tic-Tac-Toe where two users can play. The computer will only play the role of telling who has chances of winning and ultimately will decide when to finish the game and who is the winner. [**Hint:** From the image we can see that (0,0), (1,1) , (2,2) is a winning combination, and so on.]



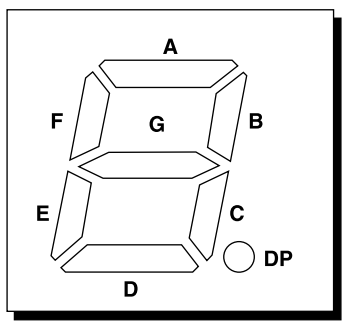
16. Perform a statistical analysis to find out which metro city has the highest degree of change as per temperature is concerned. Add one more dimension to make room for year. Now enter the past 10 year's data for all 10 metros. Can you tell what will be the temperature next October in metro no. 10.

17. What will be the statement to access the 4^{th} element in a 2×3 array?
18. What will be the syntax to access *mn*th element in an $m \times n$ array?
37. Write a program to check whether two eye matrices sum to the same diagonal or not?
38. What is the meaning of a[ ][10]?
39. Which of the following declaration is not correct?
(a) `int boxsizes[20]`
(b) `Integer boxSizes[20]`
(c) `LongInteger bigBoxSizes[20]`
(d) `All are valid declaration`
40. What is bad about the following code snippet?

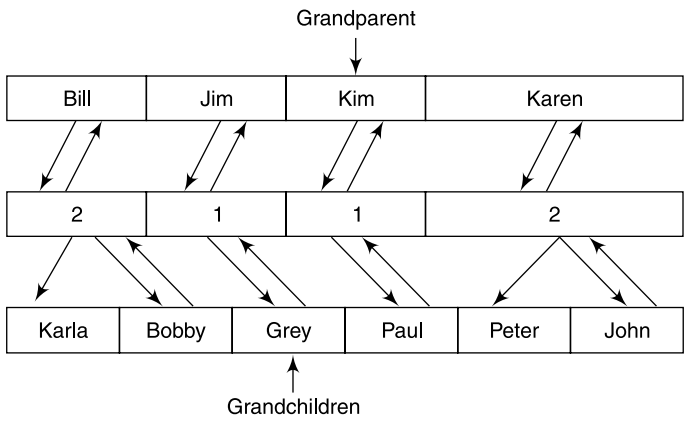
```
float d[4]={1,2,3,4};
printf("%f",d[d[d[1]]+2]);
```

41. What will be printed for this code snippet? printf("%d",*d);
42. Suppose we have an array of 2 dimensions like a[20][30]. If the starting address is 65000 then what will be the location of *n*th and *k*th element on the array?
43. What will be the content of the following array arloop?

```
float arloop[20];
for(loop=0;loop<20;loop++)
        for(foo=0;foo<28;foo++);
             arloop[loop] = foo;
```



44. A seven segment LED display can be modeled using an array of bits (1 and 0). Write a program that will accept a bit stream of 7 bits and will tell what the digit is, e.g. 0 maps to A, B, C, D, E, F. (See figure)
45. Relationships can be modeled using arrays. Let there be three arrays. The first two are of character pointers and the last one is of integers. The first one, say for example stores the names of the grandparents' second one the name of the grandchildren and the third one will store the number of grandchildren each grandparent has. Write a program that will allow the users to enter a grandparent's name and will return the names of the grandchildren. The user might enter a grandchild's name too and in that case the program should tell the grandparent's name. Let's assume that there is no name conflict. How will you prevent a name conflict ?



46. Write a function to find out the similarity index of two arrays. Similarity index of two arrays is the percentage of locations where they have the same value.
47. Write a function to assign Boolean values to a two-dimensional array that serves as the input to an array of 7 segment displays. Once the array is set, print the output that will be shown on the display.
48. Write a program to allow 4 players to play the board game Ludo.

2

Structures
The Building Blocks

INTRODUCTION

In the last chapter we have learnt about arrays, the most basic data structure offered by the C language. An array has a couple of potential disadvantages. Primarily, arrays can store only one type of variable. An integer array can only store integers, and so on. On the other hand, array elements are stored in contiguous memory locations. But to model real-life entities, we need some data container that can store data of multiple types. Structures step in there. In this chapter we will learn about structures, and how to define them. After we get acquainted with this beautiful building block of data structure, we will discuss about their different and diverse usages.

In C, the struct keyword is used to define a structure. Structures are used to define more complex data structures like linked lists, trees, graphs, etc.

A simple structure

```
struct Student
{
      char name[20];
      char course[10];
      int age;
};
```

`typedef` is a keyword that makes the defining of structures simple.

2.1 USE OF `typedef`

`typedef` is a keyword in C. It allows us to define the new data types. In more specific words, `typedef` allows programmers to give a name to their data types. `typedef` is typically used with structures. If we use `typedef` with a structure while defining the structure then the structure name becomes just a built-in data type like int or float, and we can create new structure-type objects using that name only. Every time we declare a type of that structure, the struct keyword need not be written. So using `typedef` saves

a lot of typing and makes the code much more readable and conceptual. Judicial use of `typedef` and `enum` enable us to do *literal coding* (where the code is self-explanatory, least amount of documentation is needed). Here is an example.

```
typedef struct MotorCar
{
      char *model_number;
      long kilometers;
      int year_of_mfg;
      int month_of_mfg;
      char* manufacturer;
      char *owner;
      double engine_efficiency;
}MotorCar;
```

After we have defined the above structure, we can use `MotorCar` just like another built-in primitive data type as

```
MotorCar MyOldMaruti;
MotorCar MyNewSantro;
MotorCar *PointerToOldCar;
```

As you can see, every time we don't have to use the `struct` keyword and the code becomes easy to understand and maintain.

2.2 ACCESSING THE STRUCTURE ELEMENTS

The structure elements or the attributes of a structure can be accessed using any of the two operators .(Dot) or -> (Arrow). When we have a pointer to the structure, the arrow operator is used. Otherwise the elements are accessed by Dot(.) operator. Say, we want to initialize the attributes of MyOldMaruti which is a type of MotorCar class. Here is how we can do this.

```
MyOldMaruti.model_number = "MARUTIZENDLXI";
MyOldMaruti.kilometers = 45234;
MyOldMaruti.year_of_mfg = 1999;
MyOldMaruti.month_of_mfg = 10;
MyOldMaruti.manufacturer = "MARUTI UDYOG LTD";
MyOldMaruti.owner = "Jacob";
MyOldMaruti.engine_efficiency = 89.56;
```

If we want to initialize the structure elements using a pointer, then we have to use the arrow operator. Here is how we can do that.

```
MotorCar *PointerToOldCar;
PointerToOldCar->model_number = "MARUTIZENDLXI";
PointerToOldCar->kilometers = 45234;
PointerToOldCar->year_of_mfg = 1999;
PointerToOldCar->month_of_mfg = 10;
PointerToOldCar->manufacturer = "MARUTI UDYOG LTD";
PointerToOldCar->owner = "Alexandar";
PointerToOldCar->engine_efficiency = 89.56;
```

A structure is one of the most useful data structures that is used to create more complex data structures.

2.3 SOME BUILT-IN USEFUL STRUCTURES IN TURBO C (UNDER DOS)

Date This structure is used to get/set the system date. We need to create an object to this structure in order to read the system date and set the same. Here is the structure defined.

```
struct date {
  int da_year;     /* current year */
  char da_day;     /* day of the month */
  char da_mon;     /* month (1 = Jan) */
};
```

To read the system date, we use the getdate function. Here is the C code for that.

```
int main(void)
{
  struct date d;

  getdate(&d);
  printf("The current year is: %d\n", d.da_year);
  printf("The current day is: %d\n", d.da_day);
  printf("The current month is: %d\n", d.da_mon);
  return 0;
}
```

Time

```
struct time {
  unsigned char  ti_min;   /* minutes */
  unsigned char  ti_hour;  /* hours */
  unsigned char  ti_hund;  /* hundredths of seconds */
  unsigned char  ti_sec;   /* seconds */
};
```

Like we used getdate to read the system date, we have to use gettime to read the system time.

File This structure is used for reading from a file and writing to a file. Apart from that, using this structure we can get information about a particular file in the system. Here is the File structure.

```
typedef struct FILE{
  short          level;
  unsigned       flags;
  char           fd;
  unsigned char  hold;
  short          bsize;
  unsigned char *buffer, *curp;
  unsigned       istemp;
  short          token;
} FILE;
```

2.4 HOW TO DEFINE A STRUCTURE THAT REPRESENTS A POINT IN 3D

To describe a point which is moving in three dimensions, we need three coordinates for the three axes. We can store these three co-ordinates as elements of a structure instead of storing them separately. Thus structure helps us to model real-world entities better. The point structure will look like

```
typedef struct Point
{
     double x_coordinate;
     double y_coordinate;
     double z_coordinate;
}Point;
```

After defining the above structure, we can create new variables of the type point as

```
Point   themovingpoint;
```

Now this point structure will be used to solve a few geometrical problems. We will pass point-type structure variables to different methods and there within those methods we will modify the values of the variables.

2.5 HOW TO FIND THE CENTROID OF A POLYGON USING POINT STRUCTURE

All we have to do is to divide the summation of each coordinate for each point and then divide the number by the number of vertices. Here is the code that accepts an array of point-type structures and then returns a point-type structure that represents the centroid of the polygon.

```
Point getcentroid(Point P[],double size)
{
      Point temp;
      int i;
      static double x=0;
      static double y=0;
      static double z=0;
      for(i=0;i<size;i++)
      {
            x+=P[i].x_coordinate;
            y+=P[i].y_coordinate;
            z+=P[i].z_coordinate;
      }
      temp.x_coordinate=x/size;
      temp.y_coordinate=y/size;
      temp.z_coordinate=z/size;

      return temp;
}
```

2.6 HOW TO FIND THE DISTANCE BETWEEN TWO POINTS IN 3D

In geometry finding the distance between the two points is basic for many geometrical problems. A method can be written that accepts two point-type structures and returns the distance between them as a double value. Here is the code for the method.

```
double distance(Point P1,Point P2)
{
      return sqrt(
              pow(P1.x_coordinate-P2.x_coordinate,2)+
              pow(P1.y_coordinate-P2.y_coordinate,2)+
              pow(P1.z_coordinate-P2.z_coordinate,2));
}
```

This method can be called to find the distance between two points. Using structures, we have made the code look much more conceptual. If we don't use structure, then to find the distance between two points in 3D we need to pass 6 arguments instead of two and that makes the code unreadable.

2.7 HOW TO FIND THE AREA OF ANY REGULAR POLYGON

Area of a regular polygon is given by the formula

$$A = \left(\frac{na^2}{4}\right)\cot\left(\frac{\pi}{n}\right)$$

where *a* is the arm length of the regular polygon and *n* is the number of arms.

```
double AreaOfAnyRegularPolygon
(Point AnyVertex, Point AdjacentVertex,int arms)
{

   double armlength = distance(AnyVertex,AdjacentVertex);
   return ((arms*pow(armlength,2))/4)*(1/tan(180/arms));

}
```

2.8 HOW TO TEST COLLINEARITY FOR THREE POINTS

We can check whether the three points in a plane are collinear or not by two ways. We can check whether the slope for the line joining 1st and 2nd point and the slope for the line joining the 2nd and 3rd point are equal or not. If the slopes are equal, then we can conclude that the points are collinear.

Otherwise, we can find the area formed by the three points that represent the vertices of the triangle. If the triangle area is zero then we can conclude that the three points are collinear. Here is the code to test collinearity using slopes. As the point structure is defined to model a point in 3D, then there will be three types of slopes defined namely slope in XY plane, slope in YZ plane and slope in ZX plane. There are three methods that calculate the slope in these three planes. So if we have to check whether the three points are collinear or not in XY plane we will use the method that calculates the slope in XY plane, and so on, for other planes.

Here is the code:

```
double slopeXY(Point P1,Point P2)
{
      return (P2.y_coordinate-P1.y_coordinate)/(P2.x_coordinate-
P1.x_coordinate);
}

int iscolinearXY(Point P1,Point P2,Point P3)
{
      int flag=0;
      if(slopeXY(P1,P2) == slopeXY(P2,P3))
            flag = 1;
      else
            flag = 0;
      return flag;
}
```

2.9 HOW TO CHECK IF A TRIANGLE IS EQUILATERAL

Now we can use the distance method to find out whether a triangle is equilateral or not. We will pass the three point type objects to the method and then calculate the distance between the points. If these distance values are same then the triangle is equilateral, otherwise the triangle is not equilateral. Here is the code for the function.

```
int is Equilateral(Point P1,Point P2,Point P3)
{
      int equal = 0;
      double d1 = distance(P1,P2);
      double d2 = distance(P2,P3);
      double d3 = distance(P3,P1);

      if(d1==d2 && d2==d3 && d3==d1)
            equal = 1;//the triangle is equilateral
      else
            equal = 0;//the triangle is not equilateral
      return equal;
}
```

This method can be used as a building block to other methods as `distance()` method is used here as a building block. *You must have noticed the main motivation for using structure to store the point details will give the code much more conceptual look.*

2.10 HOW TO CHECK IF A TRIANGLE IS ISOSCELES

Just by changing the operator in the above code we can find whether a triangle is isosceles or not. Here is the method that checks whether three given points will make an isosceles triangle or not.

```
int isisosceles(Point P1,Point P2,Point P3)
{
      int flag;
      double d1 = distance(P1,P2);
      double d2 = distance(P2,P3);
      double d3 = distance(P3,P1);
      //Checking whether any two arm lengths are
      //equal or not..
      if(d1==d2 || d2==d3 || d3==d1)
            flag = 1;
      else
            flag = 0;
      return flag;
}
```

2.11 HOW TO MODEL A TRIANGLE USING POINT STRUCTURE?

So long we have used point type structures separately to check whether a triangle is isosceles or not, and so on. But to move one level up, we can model a triangle with `Point` type structures and other variables. Here is the structure that will be representing a triangle.

```
enum  Type{ISOSCALES=0,EQUILATERAL,RIGHTANGLED,SCALENE};
typedef struct Triangle
{
      Point P1;//Vertex 1
      Point P2;//Vertex 2
      Point P3;//Vertex 3
      Point Centroid;
      Point Orthocentre;
      Point Incentre;
      Point Circumcentre;
      double area;
      double perimeter;
      enum Type  type;
}Triangle;
```

After this structure is defined we can pass one triangle type object instead of three points. Thus, we can extend the methods is`Equilateral()`, `Isisosceles()`, etc, to look even more conceptual. Moreover, we can pass an array of triangles to these methods and return multiple values using pointers (See chapter on arrays, More Applications Section) describing the type of each triangle passed.

2.12 HOW TO CHECK IF A TRIANGLE IS RIGHT ANGLED

To check whether a triangle is right angled or not we can check it using the slopes or by Pythagoras's Formula. Here is the code of the method that takes a triangle structure as object input and will return 1 or 0 depending on whether the triangle is right-angled or not. Here we have assumed that the triangle is lying in the XY plane. For any other plane you just need to change the method.

```
int isrightangled(Triangle T)
{

      if(slopeXY(T.P1,T.P2)*slopeXY(T.P1,T.P3)==-1 ||
         slopeXY(T.P1,T.P2)*slopeXY(T.P2,T.P3)==-1)
            T.type = 1;//It is a right angled one
      else
            T.type = 0;//It is not
      return T.type;
}
```

This approach to find whether a triangle is right angled or not is very easy, and computationally less expensive than implementing Pythagoras' theorem.

2.13 HOW TO FIND WHETHER A TRIANGLE IS EQUILATERAL OR NOT

Here is the code to check whether a triangle is equilateral or not using this triangle structure.

```
int istriangleequilateral(Triangle T)
{

      double d1 = distance(T.P1,T.P2);
      double d2 = distance(T.P2,T.P3);
      double d3 = distance(T.P3,T.P1);

      if(d1==d2 || d2==d3 || d3==d1)
            T.type = EQUILATERAL;
      else
            T.type = !EQUILATERAL;
      return T.type;
}
```

This is far more readable and more of a conceptual level than the previous code. Notice carefully how enum variable type is used to increase the readability. As point structure is used to model this triangle structure, similarly this triangle structure can be used to model a tetrahedron. Conceptually a tetrahedron is nothing but the combination of four equilateral triangles including the base triangle. We can think a tetrahedron as special triangular pyramid whose all the sides are of equal area. Now we will model triangular pyramid using triangle structure and then write methods to process the new structure.

2.14 HOW TO MODEL A TETRAHEDRON USING TRIANGLES

Let us first model a triangular pyramid using triangle structure. A triangular pyramid can be thought of as a combination of four triangles including the base.

```
enum TypeOfTriangularPyramid{NOTTETRAHEDRON,TETRAHEDRON};

typedef struct TriangularPyramid
{
      Triangle Side1;
      Triangle Side2;
      Triangle Side3;
      Triangle Base;
     Point Centroid;//Centroid of the Pyramid
      enum TypeOfTriangularPyramid type;
}TriangularPyramid;
```

Any triangular pyramid can be of two types. Either it is a tetrahedron or it is not. To make the code more readable enum variable '`TypeOfTriangularPyramid`' is used.

Now a method is written that accepts a `TriangularPyramid` structure as input, and checks whether the pyramid is a tetrahedron or not. Here is the code.

```
int isTetrahedron(TriangularPyramid tp)
{

      if(istriangleequilateral(tp.Base)==EQUILATERAL
        && istriangleequilateral(tp.Side1) == EQUILATERAL
        && istriangleequilateral(tp.Side2) == EQUILATERAL
        && istriangleequilateral(tp.Side3) == EQUILATERAL)


            tp.type = TETRAHEDRON;
      else
            tp.type = NOTTETRAHEDRON;

      return tp.type;
}
```

This method returns 1 when the triangular pyramid is a tetrahedron, otherwise it returns 0. Thus it gives more conceptual look to the code. As we know that tetrahedron is used extensively in 3D modeling, we can use this structure to model more complex 3D solids.

2.15 HOW TO MODEL A RECTANGLE USING `struct` AND `enum`

As we have modeled the triangle using point structures, we can model a rectangle using four Point type structures and one enum variable that will determine whether the rectangle is a square or not. Here is the structure that will model the structure rectangle.

```
enum TypeOfRectangle{RECTANGLE,SQUARE};

typedef struct Rectangle
{
      Point P[4];
      double area;
      double perimeter;
      enum TypeOfRectangle type;
}Rectangle;
```

Notice, that the rectangle structure holds the area and perimeter separately even though we have length and breadth defined in the structure. These variables are kept for the reverse calculations. If sometimes we are given the area and perimeter then we can find the other values. Here in this structure, unlike triangle, an array of point structure is kept instead of four different point variables. This approach makes the code, much more compact.

2.16 HOW TO MODEL A TRAPEZIUM USING POINT

To model a Trapezium we need four points. Here is a structure that represents a Trapezium

```
enum TypeOfTrapezium{ISOSCALES_TRAPEZIUM,NOT_ISOSCALES};
typedef struct Trapezium
{
     //When the user knows the co-ordinates of the points
     Point P1;
     Point P2;
     Point P3;
     Point P4;
     //When somebody dont know the co-ordinates of the vertices.
     double height;
     double length;
     double area;
     double perimeter;

     enum TypeOfTrapezium type;

}Trapezium;
```

A trapezium can be either iso-scales or not. So to model that we have used an enum variable. TypeOfTrapezium in order to find what type of trapezium is this.

2.17 HOW TO CHECK WHETHER A TRAPEZIUM IS EQUILATERAL OR NOT

```
int isIsoscalesTrapezium(Trapezium T)
{
      /*
   P1.......P2
   /         \
P4/...........\P3

*/

double b=distance(T.P1,T.P4);
```

```
      double c=distance(T.P2,T.P3);
      if(b==c)
            T.type = ISOSCALES_TRAPEZIUM;
      else
            T.type = NOT_ISOSCALES;
      return T.type;
      return 0;
}
```

2.18 HOW TO FIND WHETHER A POINT IS WITHIN A TRIANGLE OR NOT

```
enum Where{INSIDE=-1,ON,OUTSIDE};
int isPointWithinTriangle(Triangle T,Point P)
{
    Point MaxH;
    Point MaxV;
    int where=INSIDE;
   if(abs(T.P1.x_coordinate)>abs(T.P2.x_coordinate)
     && abs(T.P1.x_coordinate)>abs(T.P3.x_coordinate))
      MaxH = T.P1;
   if(abs(T.P2.x_coordinate)>abs(T.P1.x_coordinate)
     && abs(T.P2.x_coordinate)>abs(T.P3.x_coordinate))
      MaxH = T.P2;
   if(abs(T.P3.x_coordinate)>abs(T.P2.x_coordinate)
    && abs(T.P3.x_coordinate)>abs(T.P1.x_coordinate))
     MaxH = T.P3;

   if(abs(T.P1.y_coordinate)>abs(T.P2.y_coordinate)
     && abs(T.P1.y_coordinate)>abs(T.P3.y_coordinate))
      MaxV = T.P1;
   if(abs(T.P2.y_coordinate)>abs(T.P1.y_coordinate)
     && abs(T.P2.y_coordinate)>abs(T.P3.y_coordinate))
      MaxV = T.P2;
    if(abs(T.P3.y_coordinate)>abs(T.P2.y_coordinate)
    && abs(T.P3.y_coordinate)>abs(T.P1.y_coordinate))
     MaxV = T.P3;

  if(P.x_coordinate>MaxH.x_coordinate ||
    P.y_coordinate >MaxV.y_coordinate)
       where = OUTSIDE;

  if(iscolinearXY(T.P1,P,T.P2)
  || iscolinearXY(T.P1,P,T.P3)
  || iscolinearXY(T.P2,P,T.P3))
       where = ON;
       return where;
}
```

2.19 HOW TO FIND WHETHER A POINT IS WITHIN A RECTANGLE OR NOT

```
int isWithinRectangle(Rectangle R,Point P)
{
         /*
               D.............C
                :             :
                : .P          :
               A:.............B
          */
          int where;
          if((P.x_coordinate>R.P1.x_coordinate)
          && (P.x_coordinate > R.P4.x_coordinate)
          && (P.x_coordinate<R.P3.x_coordinate)
          && (P.x_coordinate<R.P2.x_coordinate))

                           where = INSIDE;

          if(P.x_coordinate == R.P1.x_coordinate
             || P.x_coordinate == R.P2.x_coordinate
             || P.x_coordinate == R.P3.x_coordinate
             || P.x_coordinate == R.P4.x_coordinate)

                              where = ON;
                    else
                              where = OUTSIDE;

                  return where;
}
```

2.20 HOW TO FIND WHETHER A POINT IS WITHIN A CIRCLE OR NOT

```
//
int isWithinCircle(Circle c1,Point P)
{
       int where;

       if(distance(c1.centre,P)<c1.radius)
          where=INSIDE;
       if(distance(c1.centre,P)==c1.radius)
          where=ON;
       else
          where=OUTSIDE;
       return where;
}
```

2.21 HOW TO FIND WHETHER TWO CIRCLES ARE TOUCHING INTERNALLY OR NOT

When two circles touch internally then the distance between their centers is less than the summation of the radii of the circles.

```
enum Status{NO,YES};
int isTouchingInternally(Circle c1,Circle c2)
{
  if(distance(c1.centre,c2.centre)<c1.radius+c2.radius)
         return YES;
  else
         return NO;
}
```

2.22 HOW TO FIND WHETHER TWO CIRCLES ARE TOUCHING EXTERNALLY OR NOT

When two circles touch externally then the distance between the centers of the circle is same as the Summation of the radii.

```
enum Status{NO,YES};
int isTouchingExternally(Circle c1,Circle c2)
{
    if(distance(c1.centre,c2.centre)==c1.radius+c2.radius)
       return YES;
    else
       return NO;
}
```

Here is the structure that will be used to represent a straight line in this format.

2.23 HOW TO MODEL A STRAIGHT LINE IN SLOPE FORMAT

```
typedef struct StraightLine
{
      //When you supply the m and c
      double m;
      double c;
      //When you give the Points on the line
      Point P1;
      Point P2;
} StraightLine;
```

As you may have noticed, that to model a straight line $y = 2x - 3.5$ value of $m = 2$ and $c = 3.5$ Instead of m and c there are two points. These two points can be any two points on the line. If the user doesn't provide m and c, then we can calculate the slope m and constant c.

We can create above line like

```
StraightLine aline;
aline.m = 1;
aline.c = 0;
```

aline represents $y = x$.

2.24 HOW TO MODEL A STRAIGHT LINE IN *XY* INTERCEPT FORMAT

```
typedef struct StraightLineXYIntercept
{
       double a;
       double b;

       Point P1;
```

```
      Point P2;
}StraightLineXYIntercept;
```

2.25 HOW TO CONVERT AN XY INTERCEPT FORM LINE TO SLOPE FORMAT LINE

```
StraightLine convertFromStraightLineXYIntercept
           (StraightLineXYIntercept sxy)
{
           StraightLine sl;
           sl.m = -(sxy.a/sxy.b);
           sl.c = -(sxy.a*sxy.b);
           return sl;
}
```

2.26 HOW TO CONVERT A SLOPE LINE FORMAT TO XY INTERCEPT FORMAT

```
StraightLineXYIntercept convertFromStraightLine(StraightLine sl)
{
   StraightLineXYIntercept sxyl;
   sxyl.a = -sl.c/sl.m;
   sxyl.b = c;//Y intercept
   return sxy l;
}
```

2.27 HOW TO FIND WHETHER TWO LINES ARE PARALLEL OR NOT

Two Straight lines are parallel to each other if and only if their slopes are equal.

```
enum {NOT_PARALLEL,PARALLEL};//An anonymous enum variable
int isParallel(StraightLine s1,StraightLine s2)
{
      if(s1.m == s2.m)
         return PARALLEL;
      else
         return NOT_PARALLEL;
}
```

2.28 HOW TO FIND THE POINT OF INTERSECTION OF TWO STRAIGHT LINES

```
Point FindIntersctionPoint(StraightLine s1,StraightLine s2)
{
     Point PointOfIntersection;
     PointOfIntersection.x_coordinate = (s2.c-s1.c)/(s2.m-s1.m);
     PointOfIntersection.y_coordinate =
       s1.m*PointOfIntersection.x_coordinate+s1.c;
     return PointOfIntersection;
}
```

2.29 HOW TO FIND THE TANGENT ON ANY POINT ON A CIRCLE

```
 StraightLineXYIntercept TangentOnCircle(Circle cir,Point P)
{
```

```
    //xx'+yy'  = a^2
  StraightLineXYIntercept Tangent;
  Tangent.a = pow(cir.radius,2)/P.x_coordinate;
  Tangent.b = pow(cir.radius,2)/P.y_coordinate;
  return Tangent;
}
```

2.30 HOW TO MODEL A PARABOLA USING A STRAIGHT LINE AND POINT

```
enum Direction{POSITIVE_X,POSITIVE_Y,NEGATIVE_X,NEGATIVE_Y};
typedef struct Parabola
{
         Point Focus;
         Point Vertex;
         StraightLineXYIntercept axis;
         StraightLineXYIntercept directrix;
         enum Direction dir;
}Parabola;
```

2.31 HOW TO FIND THE TANGENT ON ANY POINT ON A PARABOLA

```
StraightLine TangentOnParabola(Parabola P,Point op)
{
       StraightLine Tangent;
       Tangent.m =
(double)2*distance(P.Vertex,P.Focus)/(double)op.y_coordinate;
       Tangent.c = 2*distance(P.Vertex,P.Focus)*op.x_coordinate;
        return Tangent;
}
```

2.32 HOW TO FIND THE NORMAL ON ANY POINT ON A PARABOLA

```
StraightLine NormalOnParabola(Parabola P,Point pop)
{
         StraightLine Normal;
         Normal.m = -pop.y_coordinate/(2*distance(P.Vertex,P.Focus));
         Normal.c=(pop.x_coordinate*pop.y_coordinate)/(2*distance(P.Ver
         tex,P.Focus)) + pop.y_coordinate;
         return Normal;
}
```

2.33 HOW TO MODEL AN ELLIPSE

```
typedef struct Ellipse
{
       double Major;
       double Minor;
       Point Focus;
       Point RightEnd;
       Point LeftEnd;
       Point Up;
       Point Down;
} Ellipse;
```

In this structure defining ellipse, we can replace the point focus by two straight lines describing the major and the minor axis of the ellipse. In that case their point of intersection is nothing but the focus of the ellipse.

So in that case the ellipse will look like

```
typedef struct Ellipse
{
        double Major;
        double Minor;
        StraightLineXYIntercept MajorAxis;
        StraightLineXYIntercept MinorAxis;
}Ellipse;
```

2.34 HOW TO FIND THE AREA OF AN ELLIPSE

To find the area of an ellipse we need the length of major and minor axes of the ellipse. That is, independent of Focus.

Here is a *C* code that finds the area of an ellipse:

```
double getEllipseArea(Ellipse el)
{
       return M_PI*el.Major*el.Minor;
}
```

2.35 HOW TO FIND THE TANGENT AT ANY POINT OF AN ELLIPSE

```
StraightLineXYIntercept TangentOnEllipse(Ellipse elps,Point P)
{
    StraightLineXYIntercept sl;
    //Assuming an Ellipse whose major axis is X-Axis
     sl.a = pow(elps.Major,2)/P.x_coordinate;
     sl.b = pow(elps.Minor,2)/P.y_coordinate;
     return sl;
}
```

2.36 HOW TO FIND THE NORMAL AT ANY POINT OF AN ELLIPSE

Here, we will not directly use the normal equation. First let's find the tangent at that point. Normal at that point is nothing but a line which is perpendicular to the tangent and passes through that point.

```
StraightLine NormalOnEllipse(Ellipse elps, Point P)
{
         StraightLineXYIntercept TangentAtP = TangentOnEllipse(elps,P);
         StraightLine NormalAtP =
         convertFromStraightLineXYIntercept(TangentAtP);
         NormalAtP.m = -1/NormalAtP.m;
         NormalAtP.c = P.y_coordinate-NormalAtP.m*P.x_coordinate;
         return NormalAtP;
}
```

2.37 HOW TO MODEL A PRISM USING STRUCTURE

A prism is a geometrical shape that has a polygonal base and rectangular sides. The base of the prism will be a regular polygon. Say for example, the base of a prism is an octagon, then the model for the prism using rectangle and point structures in the compact form (using arrays of structures) will look like

```
typedef struct Prism
{
      Point BaseVertices[8];//8 Equidistant Points on a Plane
      double armlength;//Distance between any two
      double height;//Height of the Prism
      Rectangle AnyOneSide;//Any side of the Prism.
}Prism;
```

In this structure armlength is nothing but the distance between any two vertices of the regular octagon. In a way, this armlength actually becomes the breadth of the rectangle and height of the prism becomes the length for the rectangle. As in the previous case, here also the value of the armlength may not seem mandatory but to reverse the calculation as discussed above, these variables are needed.

Thus, it shows that how the very first structure point is used to create more complex 3-dimensional type of solids. These concepts can be enhanced further. After we discuss few other applications of structures, at the end of this chapter point structure is shown again to model few other 2D and 3D geometrical shapes.

2.38 HOW TO MODEL A CIRCULAR CYLINDER

A circular-cylinder is nothing but a combination of a circle and a point.

```
typedef struct Cylinder
{
      //Coordinate Geometry Point Of View
      Circle Base;
      Point Top;

      //Mensuration Point of View
      double height;
      double radius;
      double volume;
      double circulararea;

}Cylinder;
```

The height of the circular cylinder is the distance between the center of the base circle and the top point. Do you realize that volume and total surface area of the cylinder is nothing but a function of radius of the base circle and the height.

2.39 HOW TO FIND THE SURFACE AREA OF A CYLINDER

Total surface area of a cylinder is the area of two bases and the circular area. We can enum variables to write an intelligent function to find the area of portions.

Here is the code. The height of the cylinder is nothing but the distance between the center of the base and the top of the circle.

```
enum {BOTTOMORTOP, BOTH, SURFACE, ALL};

double getArea(Cylinder cin,enum whichpart part)
{
   double area=0;
   if(part==ALL)//All the surfaces
        area =
```

```
2*M_PI*pow(cin.Base.radius,2)+2*M_PI*cin.Base.radius*distance(cin.Base.
centre,cin.Top);
   if(part==BOTTOMORTOP)
        area = M_PI*pow(cin.Base.radius,2);
   if(part==BOTH)//Both the surface only , not curved surface
        area = 2*M_PI*pow(cin.Base.radius,2);
   if(part==SURFACE)//Only the curved surface
        area =
2*M_PI*cin.Base.radius*distance(cin.Base.centre,cin.Top);
        return area;
}
```

2.40 HOW TO MODEL A CONE

```
typedef struct Cone
{
      Circle Base;
      Point Top;
      Point AnyPointOnPerimeter;
      double slantheight;
      double height;
}Cone;
```

2.41 HOW TO FIND THE AREA OF A CONE

```
enum Where{INSIDE=-1,ON,OUTSIDE};
double getConeArea(Cone mycone,enum whichpart part)
{
      double area=0;
      mycone.slantheight =
distance(mycone.AnyPointOnPerimeter,mycone.Top);
      mycone.height = distance(mycone.Base.centre,mycone.Top);

      if(part==TOPORBOTTOM)
            area = M_PI*pow(mycone.Base.radius,2);
      if(part==SURFACE)
            area = M_PI*mycone.Base.radius*mycone.slantheight;
      if(part==ALL)
            area = M_PI*mycone.Base.radius
             *(mycone.Base.radius + mycone.slantheight);
      return area;
}
```

2.42 HOW TO FIND THE VOLUME OF THE CYLINDER DEFINED BY A CIRCLE AND POINT

```
double getVolume(Cylinder cin)
{
      return
      M_PI*pow(cin.Base.radius,2)*distance(cin.Base.center,cin.Top);
}
```

2.43 HOW TO FIND THE AREA OF THE PRISM

Just like a cylinder we can write a function that will accept a flag and depending on the flag value it will give us the area of a particular part of the prism.

```
enum whichpart{BOTTOMORTOP, BOTH, SURFACE, ALL};
double getPrismArea(Prism P,int arms,enum whichpart part)
{
      P.armlength  = distance(P.BaseVertices[0],P.BaseVertices[1]);
    P.AnyOneSide.breadth = P.armlength;
      P.AnyOneSide.length  = P.height;
      if(part == TOPORBOTTOM)
            area =
AreaOfAnyRegularPolygon(P.BaseVertices[0],P.BaseVertices[1],arms);
      if(part == BOTH)
            area =
2*AreaOfAnyRegularPolygon(P.BaseVertices[0],P.BaseVertices[1],arms);
      if(part == SURFACE)
            area = arms*P.AnyOneSide.length*P.AnyOneSide.breadth;
      if(part == ALL)
            area =
2*AreaOfAnyRegularPolygon(P.BaseVertices[0],P.BaseVertices[1],arms)+
                  arms*P.AnyOneSide.length*P.AnyOneSide.breadth;
      return area;
}
```

2.44 HOW TO FIND OUT WHETHER A POINT IS WITHIN AN ELLIPSE OR NOT

```
enum Where{INSIDE=-1,ON,OUTSIDE};
int isPointWithinEllipse(Ellipse elps,Point P)
{
   //Lets assume that the Point is on the ellipse
   int where=ON;
   int value=pow(P.x_coordinate,2)/pow(elps.Major,2)
                +pow(P.y_coordinate,2)/pow(elps.Minor,2)-1;

   if(value<0)
      where = INSIDE;
   if(value>0)
      where = OUTSIDE;
   return where;
}
```

2.45 HOW TO FIND OUT WHETHER A POINT IS WITHIN A HYPERBOLA OR NOT, ASSUME THAT THE MAJOR OR MINOR AXES ARE GIVEN

```
enum Where{INSIDE=-1,ON,OUTSIDE};

int isPointWithinHyperbola(Hyperbola hyper,Point P)
{
   //Lets assume that the Point is on the circle
   int where=ON;
```

```
int value=pow(P.x_coordinate,2)/pow(hyper.Major,2)
                  -pow(P.y_coordinate,2)/pow(hyper.Minor,2)-1;

if(value<0)
      where = INSIDE;
if(value>0)
      where = OUTSIDE;
return where;
}
```

2.46 HOW TO MODEL A RHOMBUS

```
typedef struct Rhombus
{
      Point P1;
      Point P2;
      Point P3;
      Point P4;


}Rhombus;
```

2.47 HOW TO FIND THE AREA OF A RHOMBUS

```
double getRhombusArea(Rhombus Rom)
{
     double base = distance(Rom.P1,Rom.P2);
     Point R = NormalProjection(Rom.P4,Rom.P1,Rom.P2);
     double height = distance(Rom.P4,R);
     return base*height;
}
```

2.48 HOW TO MODEL VECTORS AS STRUCTURE

The structure is a very handy tool to store triplets. Modeling vectors using a structure like point, will give the liberty to use vector as a built in data type. Here is the structure that will represent vectors.

```
typedef struct vector
{
      double xmagnitude;//magnitude along unit vector i
      double ymagnitude;//magnitude along unit vector j
      double zmagnitude;//magnitude along unit vector k
}vector;
```

This will represent a vector. And we can write methods that accept vector as parameters and can return a vector from a method. Some vector algebra logic will be implemented now using this vector structure.

2.49 HOW TO WRITE A FUNCTION TO ADD VECTORS

A method is written using the vector structure that accepts an array of vector and returns a vector. Vector addition is nothing but addition in each direction. A temporary vector is created and its variables are initialized with the directed summation of vector magnitudes. After that this temporary vector is returned as the sum of the passed vectors. Here is the code for the method.

```
vector addvectors(vector vectors[],int size)
{
      vector temp;
      static double x;
      static double y;
      static double z;
      int i;
      for(i=0;i<size;i++)
       {
             x+=vectors[i].xmagnitude;//Adding the x components
             y+=vectors[i].ymagnitude;//Adding the y components
             z+=vectors[i].zmagnitude;//Adding the z components
       }
     //Assigning the magnitude of the temporary vector
     //using the summations above.
      temp.xmagnitude = x;
      temp.ymagnitude = y;
      temp.zmagnitude = z;

      return temp;//Returning the addition vector result
}
```

2.50 HOW TO FIND THE WEIGHTED SUM OF VECTORS

Sometimes it is needed to find the weighted sum of the vectors. This may happen heavily in application of vectors where we represent some quantities by vector. Here is a code to find the weighted sum..

```
vector wightedAverage(vector vecs[],int size,double weight[])
{
   vector wsum;
   int i=0;
   double ws;
   for(i=0;i<size;i++)
   {
      vecs[i]=scalarmult(vecs[i],weight[i]);
      ws+=weight[i];
   }
   wsum=addvectors(vecs,size);
   return scalarmult(wsum,1/ws);

}
```

2.51 HOW TO FIND IF THE WEIGHTED SUM OF VECTORS IS AN *AFFINE SUMMATION* OR NOT

If the summation of weights is unity then the vector summation is known as *affine summation*. Here is a code to find whether the sum of a set of vectors is affine or not.

```
vector weightedAverage(vector vecs[],int size,double weight[])
{
      vector wsum;
      int i=0;
      double ws;
      for(i=0;i<size;i++)
```

```
    {
          vecs[i]=scalarmult(vecs[i],weight[i]);
          ws+=weight[i];
    }
    if(ws==1)//Summation of the weights is unity
    printf("Affine Sum\n");
else
    printf("Not affine Sum\n");
      wsum=addvectors(vecs,size);
      return scalarmult(wsum,1/ws);

}
```

2.52 HOW TO WRITE A FUNCTION TO FIND DOT PRODUCT OF TWO VECTORS

The dot product of two vectors is nothing but a scalar value obtained by adding the product of the two vectors in each direction. For example if the two vectors are like

$$V1 = a1\ i + a2\ j + a3\ k$$
$$V2 = b1\ i + b2\ j + b3\ k$$

Then their dot product will be $a1b1 + a2b2 + a3b3$. A method is written that accepts two vectors and returns a scalar value as their dot product. Here is the code.

```
double dotproduct(vector v1,vector v2)
{
      return v1.xmagnitude*v2.xmagnitude +
             v1.ymagnitude*v2.ymagnitude +
             v1.zmagnitude*v2.zmagnitude;
}
```

2.53 HOW TO WRITE A FUNCTION TO FIND CROSS PRODUCT OF TWO VECTORS

The cross product of two vectors is another vector which is perpendicular to the plane in which the first two belong. The direction (whether upward or downward) of the *vector right hand rule*. If two vectors are A and B given by

$$A = Ax\ i + Ay\ j + Az\ k \text{ and}$$
$$B = Bx\ i + By\ j + Bz\ k \text{ then their cross product } C \text{ is given by}$$
$$C = AxB = (AxBz - AzBy)\ i + (AzBx - AxBz)\ k + (AxBy - AyBx)\ k$$

Here is the code to find the cross product of two vectors. This method accepts two vectors and returns the cross product vector.

```
vector crossproduct(vector v1,vector v2)
{
      vector temp;
      temp.xmagnitude = v1.ymagnitude*v2.zmagnitude -
                        v1.zmagnitude*v2.ymagnitude;
      temp.ymagnitude = v1.zmagnitude*v2.xmagnitude -
                        v1.xmagnitude*v2.zmagnitude;
      temp.zmagnitude = v1.xmagnitude*v2.ymagnitude -
                        v1.ymagnitude*v2.xmagnitude;
      return temp;
}
```

2.54 HOW TO WRITE A FUNCTION FOR SCALAR MULTIPLICATION OF A VECTOR

Suppose we want to magnify the vector by a certain value or we want to diminish the value of each component of the vector. A method can be written that accepts two parameters, one vector and the other one is a scalar that magnifies or diminishes the value of each co-efficient of the vector. The method will return the magnified or the diminished vector. Here is the code for that.

```
vector scalarmult(vector myvector,double scalar)
{
      myvector.xmagnitude*=scalar;
      myvector.ymagnitude*=scalar;
      myvector.zmagnitude*=scalar;
      return myvector;
}
```

When scalar is a fraction (0 < scalar < 1) then the returned vector will be diminished. On the other hand when the scalar value is more than 1 then the returned vector will be a magnified one.

2.55 HOW TO FIND DOT PRODUCT OF THREE VECTORS

```
double scalartrippleproduct(vector a,vector b,vector c)
{
   //a.(bXc)
   vector temp;
   temp = crossproduct(b,c);
   return dotproduct(a,temp);
}
```

Notice that this above function internally calls the crossproduct() fnction to find the cross product of the two vectors and then find the dot product with the first one.

2.56 HOW TO FIND WHETHER THREE VECTORS ARE COPLANAR OR NOT

The scalar triple product gives us the volume of a box whose dimensions are given by the three vectors. So if the volume is zero then we can conclude that the vectors are co-planar. Here is a function that accepts three vectors and return 1 if they are co-planar else they will return 0.

```
enum {NO,YES};
int isCoplanar(vector a,vector b,vector c)
{
      if(scalartrippleproduct(a,b,c)==0)
            return YES;
      else
            return NO;
}
```

2.57 HOW TO FIND THE CROSS PRODUCT OF THREE VECTORS

```
double scalartrippleproduct(vector a,vector b,vector c)
{
      //a.(bXc)
      vector temp;
      temp = crossproduct(b,c);
      return dotproduct(a,temp);
}
```

Notice that how we have used the previous method '`crossproduct()`' while calculating the cross product for three vectors.

2.58 HOW TO FIND THE SCALAR PRODUCT OF FOUR VECTORS

```
double scalarquadrapleproduct(vector a,vector b,vector c,vector d)
{
   vector m;
   vector n;
   m = crossproduct(a,b);
   n = crossproduct(c,d);
   return dotproduct(m,n);
}
```

2.59 HOW TO FIND THE VECTOR PRODUCT OF FOUR VECTORS

```
vector vectorquadrapleproduct(vector a,vector b,vector c,vector d)
{
   vector m;
   vector n;
   m = crossproduct(a,b);
   n = crossproduct(c,d);
   return crossproduct(m,n);
}
```

2.60 HOW TO MODEL A COMPLEX NUMBER AS A STRUCTURE

Like vectors, complex numbers can also be modeled using structures. In math.h two complex structures are already defined. These are available in Turbo C ++ 3.5 compiler under DOS. Here are the two built in structures

```
struct complex
{
      double x,y;
}//used by cabs

struct _complexl
{
      long double x,y;
}//Used by cabsl
```

But to represent complex numbers using C in MS VC++ environment we define the following structure.

```
enum TypeOfComplexNumber{HasRealPart,PurelyImaginary};

typedef struct ComplexNumber
{
      double real;
      double imag;
      enum TypeOfComplexNumber type;
}ComplexNumber;
```

We can also store a complex number in Polar format with r and theta. Here is a structure that model a complex number in Polar format as

```
typedef struct PolarComplex
{
      double r;
      double theta;
}PolarComplex;
```

We can use these two structures, `ComplexNumber` and `PolarComplex` to write some conversion routines. Both of these structures will be useful because sometimes the magnitude *r* and the argument theta are only available instead of explicit real and imaginary values.

2.61 HOW TO DO CONVERSION FROM POLAR TO RECTANGULAR FORM AND VICE VERSA

There will be many occasions when we will need to convert a polar complex number to rectangular form or vice versa. After we have the above two structures defined, we can write two functions that do the conversions from one form to the other. Here is the code.

```
//From rectangular form to polar form
PolarComplex ComplexNumber2PolarComplex(ComplexNumber c)
{
      PolarComplex temp;
      temp.r = sqrt(pow(c.real,2)+pow(c.imag,2));
      temp.theta = atan(c.imag/c.real);
      return temp;
}
//From polar form to rectangular form
ComplexNumber PolarComplex2ComplexNumber(PolarComplex pc)
{

      ComplexNumber temp;
      temp.real = pc.r*cos(pc.theta);
      temp.imag = pc.r*sin(pc.theta);
      return temp;

}
```

We will use these structures for different algorithms involving the complex numbers. The simple ones include the complex number algebra. Here are those methods one by one.

2.62 HOW TO ADD COMPLEX NUMBERS

The addition of two complex numbers is the simplest operation possible in the complex number algebra. Here is a method that takes an array of complex numbers and returns the resultant sum of them.

```
ComplexNumber addcomplexnumbers(ComplexNumber c[],int size)
{
      int i;
      ComplexNumber sum;
      //Assigning inital values to avoid garbage in
      sum.real = 0;
      sum.imag = 0;

      for(i=0;i<size;i++)
      {
```

```
        //Adding the real parts
        sum.real += c[i].real;
        //Adding the imaginary parts
        sum.imag += c[i].imag;
}
//Checking which type of complex number it is
if(sum.real==0)
        sum.type = PurelyImaginary;
    return sum;
}
```

Here we have added an array of complex numbers and returning their sum.

2.63 HOW TO SUBTRACT ONE COMPLEX NUMBER FROM ANOTHER

Subtraction of one complex number from another is simple. We will use complex number structure to demonstrate the subtraction. To display the subtraction result in a polar format, just convert the same using the conversion function already mentioned above. Here is a method that accepts two complex number structures as parameters and return their subtraction, the second one is subtracted from the first irrespective of the absolute value.

```
enum TypeOfComplexNumber{HasRealPart,PurelyImaginary};
ComplexNumber subtract(ComplexNumber c1,ComplexNumber c2)
{
    ComplexNumber temp;
    temp.real = c1.real - c2.real;
    temp.imag = c1.imag - c2.imag;
    if(temp.real==0)
        temp.type = PurelyImaginary;
    else
        temp.type = HasRealPart;
    return temp;
}
```

2.64 HOW TO MULTIPLY TWO COMPLEX NUMBERS

The multiplication of two complex numbers is simple. We just have to pass two complex number as arguments and the method will return the multiplied complex number. Here is the code.

```
ComplexNumber multiply(ComplexNumber c1,ComplexNumber c2)
{
    //(a + ib)*(c+id) = (ac-bd)+(bc+ad)i
    ComplexNumber c;
    c.real = c1.real*c2.real-c1.imag*c2.imag;
    c.imag = c1.imag*c2.real+c1.real*c2.imag;
    return c;
}
```

We can use this method recursively to find the multiplication of more than two complex numbers. In that case, we will pass an array of complex numbers to the method.

2.65 PROVING DE MOIVRE'S THEOREM USING POLAR COMPLEX STRUCTURE

To calculate the value of a complex number when raised by a scalar we use *De Moivre's theorem*. This theorem is based on the *polar form of complex numbers*. We can use the above defined polar complex structure to write a method that will find De Moivre's theorem, here is the code.

```
PolarComplex demoiver(PolarComplex c,int n)
{
      c.theta*=n;
      return c;
}
```

2.66 HOW TO WRITE A PHONEBOOK SIMULATION PROGRAM USING STRUCTURE

We can create a phonebook using structure and File. Here the records will not be stored permanently in a file instead they will be stored in an array of structure and then processed in runtime. That's why it is just a simulation. The phonebook will be able to store name, address, phone-number and email, etc. of friends. The structure that will store these variables for each friend is

```
typedef struct Friend
{
      char *name;
      char *address;
      char *phone;
      char *email;
      date birthday;//Only valid for Turbo C 3.5 Under DOS.
      char gender;
}Friend;
```

So after this structure is defined we can use friend as a built in data type like int or float. The phonebook will have the following operations:

- Add a new friend
- Modify an existing friend record
- Search a friend

As you can see we have used a structure Date[*As we are using MS Visual Studio 6.0 so we can't use the date structure that is built in Turbo C 3.5 and onwards under DOS*] to store date of birth of our friends. This is one example how we can use structure within another structure to model any real world entity like our friends in this case. We will take up several other examples later in this chapter.

A big array of friend type structures will be created and a global counter will be kept that will be increased by unity every time a new entry is added. Here are the global variables that are used by all the methods.

```
Friend Pals[10];//Array of friends
int count;//Total Number of friends
//loc is the variable for storing the index of the sought friend
int loc=-1;
```

Now here is the code for adding a new friend

```
void addapal()
{
      printf("Enter name :");
      fflush(stdin);
      gets(Pals[count].name);
      printf("Enter Address :");
      fflush(stdin);
      gets(Pals[count].address);
      printf("Enter Phone :");
      fflush(stdin);
      gets(Pals[count].phone);
      printf("Enter email");
      gets(Pals[count].email);
      printf("Enter birthdate :");
      scanf("%d%d%d",&Pals[count].bday.day,&Pals[count].bday.month,&Pal
s[count].bday.year);
      printf("M/F :");
      fflush(stdin);
      scanf("%c",&Pals[count].gender);
      printf("New Friend Successfully added !");
      count++;//One friend is Just Added, so count is increased by 1
}
```

This bold line shows how to access one structure which is within another structure. Bday is a date structure and Pals[count].Bday represents the date of birth of the current friend whose details are being added. So in the Pals[count].bday.day is the day of the date of birth of the current friend, and so on.

The method to search is simple. The global variable loc is initialized with the value –1. If the search is successful and the friend is found then value of loc will be overwritten with the index of the array where the sought friend is found. Otherwise the initial value –1 will remain. Thus we can check whether a search is successful or not using the value of the global variable loc. The search algorithm is linear. It starts from the first entry and the loop rotates until the array is finished. If the sought friend is found it breaks from the loop and the value of the breaking index is sent back to the calling method (*From where this search method is called, maybe main() may be some other method like modify() where it is necessary to find the friend before further processing*).

This search is an exact string search. That means if there is an entry where the name is "John Abraham" and another entry "Johnathon Swift"and the search string for name is "John" then none of these will match and the search will be unsuccessful. This will happen because we have used `strcmpi()` that compares two strings ignoring case and returns 0 if they are exactly same. But if we want both these names to match when the supplied name to search is "John" then `strncmpi()` method should be used instead of `strcmpi ()`.

```
int search()
{
char querystring[50];
int choice;
int d,m,y;
int i=-1;//Assuming that the Friend will not be present
printf("1.Search by name");
printf("2.Search by phone");
printf("3.Search by email");
```

```
printf("4.Search by birthdate");
printf("Enter your choice [1-4] :");
scanf("%d",&choice);
switch(choice)
{
case 1:printf("Enter name to search :");
         fflush(stdin);
         gets(querystring);
         for(i=0;i<count;i++)
               if(strcmpi(Pals[i].name,querystring)==0)
                     break;
         break;
   case 2:printf("Enter phone to search :");
         fflush(stdin);
         gets(querystring);
         for(i=0;i<count;i++)
               if(strcmpi(Pals[i].phone,querystring)==0)
                     break;
         break;
   case 3:printf("Enter email to search :");
         fflush(stdin);
         gets(querystring);
         for(i=0;i<count;i++)
               if(strcmpi(Pals[i].email,querystring)==0)
                     break;
         break;
   case 4:printf("Enter birthdate to search :");
         scanf("%d%d%d",&d,&m,&y);
         for(i=0;i<count;i++)
               if(Pals[i].bday.day == d && Pals[i].bday.month ==
m  && Pals[i].bday.year == y)
                     break;
         break;
   }
   return i;//returning the location of the friend sought

}
```

To display the records in the phonebook two methods have been written. They are self-explanatory. Here are the codes.

```
void displayarecord(int loc)
{
        printf("Name :%s\n",Pals[loc].name);
        printf("Address :%s\n",Pals[loc].address);
        printf("Phone :%s\n",Pals[loc].phone);
        printf("Email :%s\n",Pals[loc].email);
        printf("Birthdate :%d-%d-
%d\n",Pals[loc].bday.day,Pals[loc].bday.month,Pals[loc].bday.year);
        printf("Sex :%c\n",Pals[loc].gender);

}
void display()
```

```
{
     int i=0;
     printf("Count %d\n",count);
     if(loc==-1)//All phonebook entries are to be traversed
     {
          for(i=0;i<count;i++)
          {
           displayarecord(i);
          }
     }
     else//Only the sought entry
     {
          displayarecord(loc);
     }
}
```

The last operation of the phonebook is to modify the details of a friend. To modify a friend's details first of all we need to find whether the friend's details at all exist in the phonebook or not. So we will have to call search() method from modify() method.

If the friend is not found then a proper message will be displayed otherwise the system will ask what detail of the sought friend is to be modified. After user enters the new value for the chosen field the program will overwrite the contents of that field for that particular friend array entry. *Try to write this modify code by yourself [Clue: You may find it useful to break search() function first and then proceed]*

2.67 HOW TO MODEL A BANK ACCOUNT AS A COMBINATION OF STRUCTURES

A bank account represents a real world entity. We can model it using struct. Here are the structures that will be used to model a bank account.

```
enum TypeOfAccount{SAVINGS,CURRENT,CREDIT};
enum TypeOfIncome{SALARIED,BUSINESS};
enum TypeOfCreditCard{SILVER,GOLD,MANHATTAN};

typedef struct Date
{
     int day;
     int month;
     int year;
}Date;
typedef struct Car
{
      char modelnumber[20];
      long kilometers;
      int yearofmfg;
      int monthofmfg;
      char manufacturer[40];
      char owner[40];
      double engineefficiency;
       int hasevermetaccident;
}Car;
typedef struct Home
```

```
{
     char address[100];
     char country;
     long pincode;
     int residing;
     int abandoned;
     int rented;
     int own;
}Home;
typedef struct CreditCard
{
     char creditcardnumber[16];
     int csvnumber;
     Date validupto;
     long creditlimit;
     enum TypeOfCreditCard type;
}CreditCard;
typedef struct Properties
{
     Car cars[4];
     Home homes[3];
     CreditCard ccs[10];
     int noofcreditcardsinuse;
     double avgearningpermonth;
}Properties;
typedef struct BankAccount
{
     enum TypeOfAccount whichaccount;
     char accountholdername[40];
     char accountnumber[10];
     char address[100];
     Date openingdate;
     int age;
     int sex;
     double initialbalance;
     Properties props;//What are the properties of accountholder
     enum TypeOfIncome type;
}BankAccount;
```

Account or a credit card account For holding the type an enum variable is used. All property details of the account holder are kept because income tax calculation will be based on these. We can now use these structures to write a small banking application with the basic banking operations like deposit funds, withdraw funds, balance enquiry, interest calculation, etc.

Say a bank has the following rules.

1. *Initial balance can't be less than 2500 for Saving account.*
2. *Interest rate for savings account is 7%.*
3. *If someone has more than one home then current account balance can't be less than 3500.*
4. *If someone's car has ever met with an accident then whenever he deposits some cash in savings account 10% is cut and is given to the insurance company as a life insurance coverage.*
5. *Minimum amount withdraw from savings account is 500.*

6. *If any of the car's engine efficiency is less than 89% then each time he uses any of his credit cards the accountholder will be charged 4% of the amount as an environmental protection insurance.*

Write a program that uses the above structures and for 10 clients. Use an array of BankAccount Structure.

2.68 HOW TO WRITE A POS (POINT OF SALE) SIMULATION USING STRUCTURE

Items in a departmental store can be modeled as a structure. Thus we can write software using this structure to simulate the process of a departmental store as we did in the phonebook program above.

Here is the structure that defines one item in a departmental store.

```
typedef struct Item
{
     char itemname[20];
     int itemcode;
     float price;
     int quantityinstock;
     int ishighdemand;
     int soldtoday;
}Item;
```

Here is the entire code. There is room for improvement as per validations are concerned. This code is just for conveying the fact how structures can be used to solve our real-world problems.

```
Item itemsinstore[100];
int count=0;
int solditemcode;
//int loc=-1;

//Will show this message every time the stock is out for an item
void oos()
{
      printf("Sorry this item is out of stock!\n");
}
//For adding a new item in the store.
void addanitem()
{
      printf("Item Name :");
      fflush(stdin);
      gets(itemsinstore[count].itemname);
      printf("Item Code :");
      scanf("%d",&itemsinstore[count].itemcode);
      printf("Item Price :");
      scanf("%f",&itemsinstore[count].price);
      printf("Quantity in stock :");
      scanf("%d",&itemsinstore[count].quantityinstock);
      printf("The new item added successfully");
      count++;
}
//One sample of one item is soldout.
void soldoneitem(int whichitem,int howmanypiece)
{
```

```
      if(itemsinstore[whichitem].quantityinstock<howmanypiece)
            oos();
      else
            itemsinstore[whichitem].quantityinstock-=howmanypiece;
}
//Displaying the bill on console.
void preparebill(char *name,int whichitems[],int howmany[],int size)
{
      int i;
      float total=0;
      printf("\n-------------------------------------\n");
      printf("Bill for %s\n",name);
      printf("-------------------------------------\n");
      printf("Item                  Quantity       Price \n");
      printf("-------------------------------------\n");
     for(i=0;i<size;i++)
     {
          printf("%d      %s         %d =
%f\n",itemsinstore[whichitems[i]].itemcode,

itemsinstore[whichitems[i]].itemname,

howmany[i],itemsinstore[whichitems[i]].price*howmany[i]);
          total+=itemsinstore[whichitems[i]].price*howmany[i];
     }
     printf("-------------------------------------\n");
     printf("Total Bill Amount : %.2f + 1 percent VAT Tax =
%.2f\n",total,total*1.01);
     printf("Thanks for shopping with us! Come Again\n");


}

void updatestock(int whichitem,int byhowmany)
{
     itemsinstore[whichitem].quantityinstock+=byhowmany;
     printf("Stock is updated successfully");
}

void updatepriceofanitem(int whichitem,float updatedprice)
{
     itemsinstore[whichitem].price = updatedprice;
     printf("Price has been changed successfully");
}

int searchanitem(int itemcode)
{
     int i;
     int loc=-1;
     printf("Please wait while we serch :\n");
     for(i=0;i<count;i++)
```

```
{
      if(itemsinstore[i].itemcode==itemcode)
      {
            loc=i;
            break;
      }
}
return loc;
}
void displaypricelist()
{
     int i;
     printf("-----------------------------------------\n");
     printf("                            TODAYS PRICE LIST \n");
     printf("-----------------------------------------------\n");
     printf("Item Name                  | Item Code   |
| Stock  |          Item Price\n");
     printf("-------------------------------------\n");
     for(i=0;i<count;i++)
     {
      printf("%8s       %d         %d           %.2f       \n",
                itemsinstore[i].itemname,
                itemsinstore[i].itemcode,
              itemsinstore[i].quantityinstock,
              itemsinstore[i].price);
     }
}

void showmenu()
{
     ////////clrscr();
     printf("\n1.Add a new item\n");
     printf("2.Update Price\n");
     printf("3.Update stock\n");
     printf("4.Show Pricelist\n");
     printf("5.Sell an item\n");
     printf("6.Exit\n");
     printf("Choice [1-6]:");
}
int main(void)
{
     int i;
     int loc;
     int choice;
     int code;
     int whichitem;
     float price;
     int more;
     char name[50];
     int size;
     int howmany[100];
     int whichitems[100];
```

```
do
{
      loc = -1;
      showmenu();
      scanf("%d",&choice);
      switch(choice)
      {
                  case 1:addanitem();
                  break;
            case 2:printf("Enter the code of the item you want to
update the price :");
                  scanf("%d",&code);
                  whichitem=searchanitem(code);
                  if(whichitem==-1)
                        printf("No Such item is there in the
store!\n");
                  else
                  {
                        printf("Enter the new price ");
                        scanf("%f",&price);
                        updatepriceofanitem(whichitem,price);
                        printf("Price is Updated
Successfully!");
                  }
                  break;

            case 3:printf("Enter the code of the item you want to
update the stock:");
                  scanf("%d",&code);
                  whichitem=searchanitem(code);
                  if(whichitem==-1)
                        printf("No Such item is there in the
store!\n");
                  else
                  {
                        printf("How many more :");
                        scanf("%d",&more);
                        updatestock(whichitem,more);
                        printf("Stock is updated!\n");
                  }
                  break;
            case 4:displaypricelist();
                  break;
            case 5:printf("Name Please :");
fflush(stdin);
gets(name);
printf("How many items :");
scanf("%d",&size);
for(i=0;i<size;i++)
{

      printf("Enter code :");
```

```
                        scanf("%d",&solditemcode);
                        loc=searchanitem(solditemcode);

                        if(loc==-1)
                        {
                                printf("Sorry! Wrong or Invalid ItemCode
Entered! Billing Process Terminated.\nHit a Key to Start Again!\n");
                                getch();
                                showmenu();
                                break;
                        }
                        else
                        {
                                printf("Quantity :");
                                scanf("%d",&howmany[i]);
                                whichitems[i]=searchanitem(solditemcode);
                                soldoneitem(whichitems[i],howmany[i]);

                        }

                    }
                       preparebill(name,whichitems,howmany,size);
                       break;
            case 6:exit(0);
                    break;
            }
      }while(1);

      return 0;

}
```

Example 2.1 ***A perpendicular is drawn from an outside point on a line given by two points. Write a program that accepts three points and then return the Point where the Perpendicular meets the Straight line. Here is a diagram.***

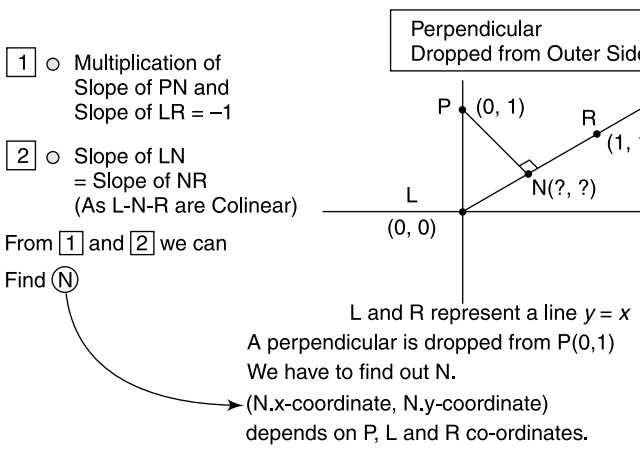


Fig. 2.1

Solution

```
Point NormalProjection(Point outside,Point left, Point right)
{
      Point temp;
      double l = right.y_coordinate-left.y_coordinate;
      double m = right.x_coordinate-left.x_coordinate;
      double p = left.y_coordinate*right.x_coordinate -
      left.x_coordinate*right.y_coordinate;
      double q = m*outside.x_coordinate+l*outside.y_coordinate;
      temp.x_coordinate = (m*q-l*p)/(pow(l,2)+pow(m,2));
      temp.y_coordinate = (p+l*temp.x_coordinate)/m;
      return temp;
}
```

Example 2.2 ***Suppose a plane mirror is modeled as a straight line. An object is placed before it. So an image is generated at the back of the mirror which will be as far from the mirror as the object. Write a function to find out the location of the image. The function will take three points as arguments. The first one will represent the object, the next two will represent any two points on the plane mirror. Here is a picture of the situation described.***

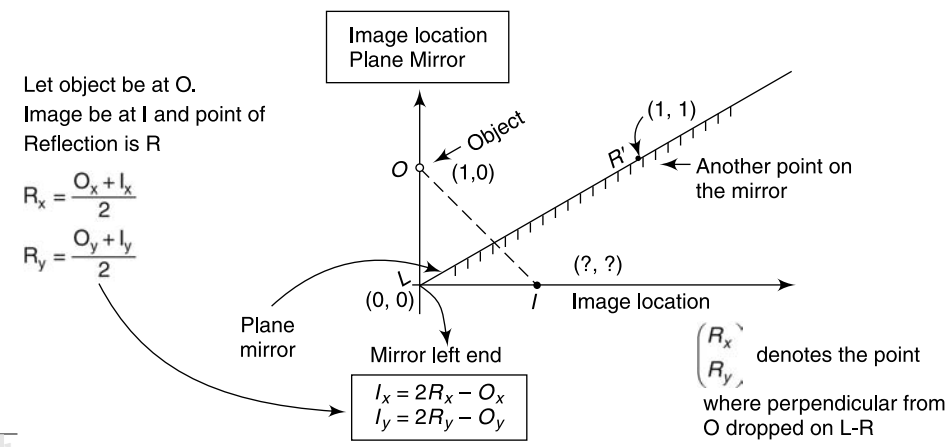


Fig. 2.2

Solution

```
Point imageLocation(Point Object,Point mirrorleft,Point mirrorright)
{
        Point Image;
        Point PointOfReflection;
        Image.x_coordinate=0;
        Image.y_coordinate=0;
      PointOfReflection =
NormalProjection(Object,mirrorleft,mirrorright);
   Image.x_coordinate = 2*PointOfReflection.x_coordinate -
                        Object.x_coordinate;
   Image.y_coordinate = 2*PointOfReflection.y_coordinate -
                        Object.y_coordinate;

        return Image;

}
```

Example 2.3 ***Write a function to find the conjugate of a complex number. Write another function to find the modulus of a complex number. Then using these two methods write one method to find the division of complex numbers.***

Solution

Before we write the method to calculate the division of two complex numbers we have to write a method that finds the conjugate complex of a given complex number. Basically the divisions of complex numbers involve the multiplication of complex numbers. Here is the method to find the conjugate complex of a given complex number.

```
ComplexNumber conjugate(ComplexNumber c)
{
      ComplexNumber conjugate_of_c;
      conjugate_of_c.real = c.real;
      conjugate_of_c.imag = -c.imag;
      return conjugate_of_c;
}
```

The division of two complex number follows the following steps.

- Multiply the numerator and denominator by the conjugate complex of the denominator.
- Separate the real and imaginary part

Multiplying a complex number with its conjugate gives the absolute value of the complex number. Let's write a method like cabs(), that calculates the absolute value of the supplied complex number. Here is the method.

```
double modulus(ComplexNumber c)
{
      return sqrt(pow(c.real,2)+pow(c.imag,2));
}
```

After we have the above two methods defined we can find division of two complex numbers very easily using these functions. Here is the code of the method that returns a complex number as the division result of the two complex numbers.

```
ComplexNumber divide(ComplexNumber numerator,ComplexNumber denominator)
{
      ComplexNumber temp;
      temp.real =
multiply(numerator,conjugate(denominator)).real/modulus(denominator);
      temp.imag =
multiply(numerator,conjugate(denominator)).imag/modulus(denominator);
      return temp;
}
```

If you notice carefully you can see how we have used the above defined methods conjugate and multiply to calculate the division of two complex numbers.

Example 2.4 ***Define a structure called Student that has the following attributes Name (character array of length 50), Age(integer 2 digit), Sex(one character either M or F). Standard.(one integer value from 1 to 12). Create an array of 10 Students.Interactively fill the details of the students. Use a loop to display the records. Write a method to search a particular student using Name. Write another method to modify the details.***

Solution

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


typedef struct student
{
      char name[40];
      int age;
      char sex;
      int standard;
}student;

student schoolstudents[10];

int main()
{
      int i=0;
      char nametosearch[40];
      printf("Enter Details about the students :");
      for(;i<10;i++)
      {
            printf("Name :");
            fflush(stdin);
            gets(schoolstudents[i].name);
            printf("Age :");
            scanf("%d",&schoolstudents[i].age);
            printf("Sex :");
            fflush(stdin);
            scanf("%c",&schoolstudents[i].sex);
            printf("Standard [1-12]:");
            scanf("%d",&schoolstudents[i].standard);
      }
      printf("Details Successfully Entered.\n");
      //Displaying the records
      for(i=0;i<10;i++)
            printf("Name :%s\nAge : %d\n Sex :%c\n Standard :%d\n\n",
                  schoolstudents[i].name,schoolstudents[i].age,
                    schoolstudents[i].sex,schoolstudents[i].standard);

      //Searching the record
      printf("Enter the name of the student to search :");
      fflush(stdin);
      gets(nametosearch);
      for(i=0;i<10;i++)
            if(strcmpi(nametosearch,schoolstudents[i].name)==0)
                  printf("Name :%s\nAge : %d\n Sex :%c\n
                  Standard :%d\n\n",
schoolstudents[i].name,schoolstudents[i].age,
                    schoolstudents[i].sex,schoolstudents[i].standard);
```

```
      return 0;
}
```

Write the modification part yourself. It involves searching of the student from the list.

Example 2.5 ***Define a structure called Medicine. The structure will have the following attributes. "Name", "Vendor", "Manufacture Date", "Expiry Date", "Dosages". Use the Date structure that is defined in the chapter. Using this structure simulate the working of a Medical Shop. Write a method that automatically generates a message whenever any particular medicine is low in stock.***

Solution

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>

enum {NOTFOUND,FOUND};

#define MEDICINES 2

typedef struct Date
{
      int day;
      int month;
      int year;
}Date;
typedef struct Medicine
{
      char name[20];
      char vendor[10];
      char adultdosage[40];
      char childdosage[40];
      Date mfgdate;
      Date expdate;
      float price;
      int thresholdquantity;
      int quantity;
}Medicine;

Medicine meds[MEDICINES];

void buydrug(int i)
{
      printf("Name of the drug :");
      fflush(stdin);
      scanf("%s",meds[i].name);
      printf("Vendor of the drug :");
      fflush(stdin);
      scanf("%s",meds[i].vendor);
      printf("Enter threshold quantity :");
      scanf("%d",&meds[i].thresholdquantity);
      printf("Enter mfg date :");
      scanf("%d%d%d",&meds[i].mfgdate.day,&meds[i].mfgdate.month,&med
```

```
s[i].mfgdate.year);
      printf("Enter exp date :");
      scanf("%d%d%d",&meds[i].expdate.day,&meds[i].expdate.month,&med
s[i].expdate.year);
      printf("Price :");
      scanf("%f",&meds[i].price);
      printf("Quantity :");
      scanf("%d",&meds[i].quantity);
      printf("Enter child dosage details :");
      fflush(stdin);
      scanf("%s",meds[i].childdosage);
      printf("Enter adult dosage details :");
]
      fflush(stdin);
      scanf("%s",meds[i].adultdosage);
      printf("Medicine Details Successfully Registered.\n");
      printf("Total Payment to be made to the Supplier :
%.2f\n",meds[i].price*meds[i].quantity);
}
void updatedrugstock()
{
      char drugname[20];
      int dquantity=0;
      int flag=NOTFOUND;
      int i;
      printf("Enter the name of the drug :");
      fflush(stdin);
      scanf("%s",drugname);
      for(i=0;i<MEDICINES;i++)
      {
            if(strcmpi(drugname,meds[i].name)==0)
            {
                  flag = FOUND;
                  break;
            }
      }
      if(flag==NOTFOUND)
            printf("The drug is not found\n");
      if(flag==FOUND)
      {
            printf("Enter quantity :");
            scanf("%d",&dquantity);
            meds[i].quantity+=dquantity;
      }
      printf("Stock successfully updated \n");
}
int checkifavailable()
{

      char drugtofind[20]="";
      int flag=NOTFOUND;
```

```
int i;
printf("Enter the name of the drug \n");
fflush(stdin);
scanf("%s",drugtofind);

for(i=0;i<MEDICINES;i++)
      if(strcmpi(drugtofind,meds[i].name)==0)
      {
            flag = FOUND;
            break;
      }
      if(flag==FOUND)
            return i;
      else
            return -1;
}

void selldrug()
{
      int howmany=0;
      int i = checkifavailable();
      if(i)
      {
            printf("Total available quantity :",meds[i].quantity);
            printf("How many you want :");
            scanf("%d",&howmany);
            if(howmany>meds[i].quantity)
                  printf("Sorry We can give only %d",meds[i].quantity);
            else
            {
                  meds[i].quantity-=howmany;
                  printf("Dosage Details\n");
                  printf("Child Dosage : %s \n Adult
Dosage%s\n",meds[i].childdosage,
                                          meds[i].adultdosage);
                  printf("Bill Amount :%f\n",meds[i].price*howmany);
            }
      }
      else
            printf("Sorry the drug you are searching is not in our
stock\n");

}
int main()
{
int choice=0;
int i=0;
do
{
            printf("1.Buy a new drug\n");
            printf("2.Update stock\n");
            printf("3.Sell drug\n");
```

```
                printf("4.Check Availability of a drug\n");
                printf("5.Quit\n");

                scanf("%d",&choice);
                switch(choice)

                {
                      case 1:i++;
                             if(MEDICINES)
                                buydrug(i);
                             else
                                printf("Can't buy\n");
                             break;
                      case 2:updatedrugstock();
                             break;
                      case 3:selldrug();
                                break;
                      case 4:if(checkifavailable())
                                   printf("We have the drug\n");
                             else
                                   printf("Sorry! we don't have the
medicine\n");
                             break;
                      case 5:exit(0);
                             break;
                }
      }while(1);
}
```

REVISION OF CONCEPTS

Some Key Facts about Structures

1. Structures are used to create user defined data types.
2. A structure can hold different data types within it, unlike arrays.
3. A structure can have another structure within it.
4. A structure can be passed to the function as arguments.
5. A structure can be returned from methods.
6. In C++ structure can hold functions also.
7. The structures can be inherited in C++.
8. By default all the members of a structure are public in C++.
9. It serves as the building block of many complicated data structures like linked list, stack, queue, tree, etc.
10. It is used for reading/writing images in C.
11. It is used to model real world entities. So they found application in inventory control systems. For a departmental store, starting from the item to be sold to the salesman who sells that can be modeled using structures.
12. It is also used in file compression algorithms like Mp3 file format etc.

REVIEW QUESTIONS

1. Suppose we have defined DNA as a structure. How will you create an array of DNA structures for 100 people?
2. Say there is an attribute called Guanine in our DNA structure defined. What will be the syntax to compare if Guanine content for first and last person matches or not. Assume that Guanine is a character array and we have in total 100 people's DNA sample information captured in the array of DNA structures.
3. How will you declare an array of structure which contains 5 structures of 3 different types of arrays? Be creative with the names!
4.
```
typedef struct Point
      {
              double x;
              double y;
              double z;
      }Point;
```
What will be the output of the following statement
```
printf("%d",sizeaof(Point));
```
5.
```
struct Number
{
  int x;
  float y;
};
Number wrongnumber;
```
Can you tell what is wrong in the above code ?
6. Write a structure definition to model a polynomial of 3 variables x,y,z along with coefficients and constant terms. Do you think one structure will be sufficient? If not then what other structures you want to write and what will they contain?
7. How will you read system date? Change the system date?
8. Which structure will you use for customizing your program for different locals?
9.
```
typedef struct Point
{
  Point p;
  Point *pp;
}Point;
```
What is wrong with this declaration?
10. Point P [100];//Assume we are using the point structure defined in the Chapter
P[0] = P[1]; what will this statement do ?

PROGRAMMING PROBLEMS

1. Define a structure called "Polygon" which can be of maximum 50 arms using the Point structures. Decrease the arm length in a regular fashion. Write a method to find the locus of the centroid. Is the locus of the centroid a known function to you? If not, can you fit this curve with the points you got for the moving centroid?

2. What will happen in the above case if the mirror rotates by theta degree.
3. How to find the point of intersection between a circle and a parabola?
4. How to find the point of intersection between a straight line and a circle?
5. How to find the exponentiation of the complex numbers?
6. In a room there are 100 particles for example. The particles can be modeled as moving points ignoring their dimensions. Any particle's *x* co-ordinate is changing as sin(*i*), *y* coordinate is changing as cos(*i*) and *z* coordinate is changing as tan(*i*) where *i* is the number of the particle. Using the point structure, write a program to plot the paths.
7. A plane mirror creates an image which is at same distance as that of the object from mirror. It means if any object is at a distance *d* from a plane mirror, its image will also be *d* distance at the back of the mirror. Using point structure write a method image location() that accepts a point structure and returns a point which represent the image location. Assume that the mirror is the x axis.
8. Extend the above method. Instead of assuming that mirror is x axis, accept two more points. Now the line joining these two points is the mirror.
9. Assume that x axis is the plane mirror and it can be rotated about origin. Accept two arguments. A Point and another scalar representing the degree of rotation. Then change the image location() method properly to find the image location.
10. Write a method calculateany polygonarea() to calculate the area of any regular closed polygon. The function should display proper message if the polygon is not regular. To represent the Polygon an array of Points representing the vertices of the Polygon will be sent to the method. If everything is ok, the function will return the area of the polygon.
11. Three points are given. Write a function to find out whether they are co-planer or not.
12. Write the same method for a triangle.
13. Write a function to calculate the exponentiation of one of the complex number by another. The method will accept two complex numbers as input and then return a complex number where one is raised to the power to another. Like if parameters passed to the function are *a+ib* and *c+id* then the function will return a complex number which is given by (*a+ib*)^(*c+id*) where ^ denotes exponentiation.
14. Forces are represented by vectors. Suppose there are *n* numbers of forces on a static body. Write a function to accept maximum 10 different forces on a body and then try to find out whether the body will be in equilibrium or not. If the body doesn't stay in equilibrium when all these forces are acted on simultaneously chances are it will move in a particular direction with some acceleration. Try to find that direction vector and acceleration if any. Use the vector structure that is already defined.
15. A body is in equilibrium due to the simultaneous application of three forces. Two of the forces are given. Write a method to find out the third. [*Clue: Use Lami's Theorem*]
16. Write a function that will take a starting Point and the number of arms and arm length as input. Then the function will give us the co-ordinates of other vertices. Do you realize that there can be more than one possible answer to this question ? If no starting point is given use origin (0,0) as the starting point. [Clue: *The question is not asking you to find all possible answers but only one of them. And it takes two consecutive vertices of any polygon to draw a straight line*]
17. Write a function to represent any polygon based pyramid using point structure.
18. Write a function to find the total area of such a pyramid.
19. Write a function to find the surface area of such a pyramid.
20. Write a function to find the volume of such a pyramid.
21. How to find that a given straight line is a tangent to a parabola at a particular point or not.
22. Write a function to find whether a sphere can reside in a pyramid or not.
23. Give an example of structure within structure?
24. Can this be viewed as containership? Explain your answer.

3

Linked List

Scattered Yet Linked!

INTRODUCTION

So far we have discussed array and structure. The main disadvantage of an array is that the memory locations needed to store an array should be contiguous. Sometimes that might make an array usage prohibitive. Then we need to use linked lists. The linked list is the most simple pointer based data structure that allows the user to store the variables in the diverse locations. The linked lists are the building block of any kind of pointer based data structure. In this chapter we will discuss about different type of linked lists and then show how different variations of linked lists can be used to solve problems from the polynomial mathematics to DNA reaction simulation.

Different Types of Linked Lists

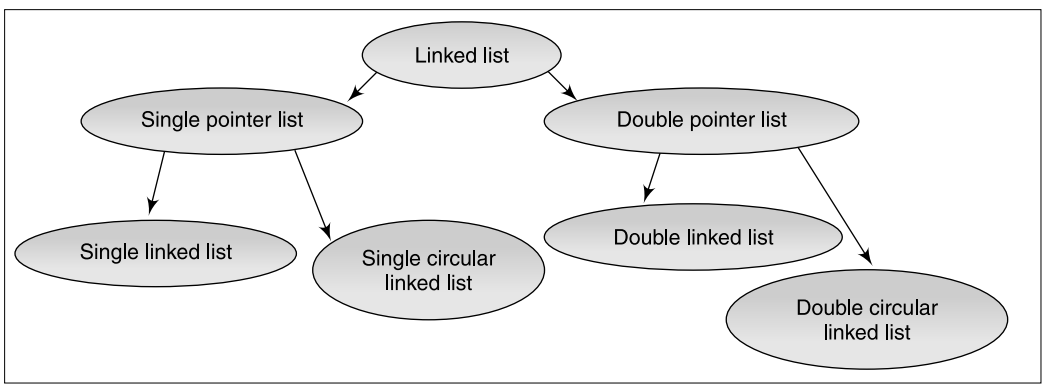


Fig. 3.1 *Classification of linked lists depending on number of pointers needed to implement*

Argument There can be an argument that traversing backward in a single linked list is possible, but should we do it? Our answer would be 'No'. We shouldn't kill the readability of the program to show

some fancy pointer acrobatics to the so called 'naïve' audience. We should remember that in a multi developer environment readability of the code is very important and we can't kill that just to show our Not-So-Legible pointer poetry. The above classification of linked list is done keeping the above lines in mind. So the thumb rule is when we need One-Way-Traffic, single linked list is the choice. On the other hand, Two-Way-Traffic will be best served by a double linked list. If we need to come back to the starting point right after we visit the last node in the list, we should use the circular versions of single or double linked lists.

3.1 SINGLE LINKED LIST

Each node of a single linked list is represented by a structure that holds the data and a pointer to the same structure. The structure below represents a single linked list of the doubles.

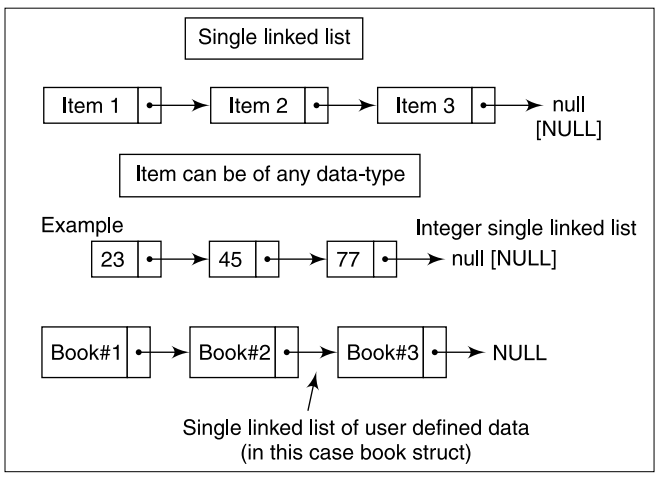


Fig. 3.2

```
typedef struct node
{
      double info;
      struct node *next;
}node;
```

This structure denotes a single linked list of double values. To design a single linked list of a user defined data we need to define the data first and then create a linked list of it. Here is an example.

```
//An user defined data
typedef struct Book
{
      char *title;
      char *author;
      char *publisher;
      int month_of_publication;
      int year_of_publication;
      int pages;
      int price;
      int edition;
```

```
}Book;
typedef struct A_Node_Of_Book_List
{
      Book book_of_this_node;
      struct A_Node_Of_Book_List *nextbookaddress;
}A_Node_Of_Book_List;
```

3.2 DOUBLE LINKED LIST

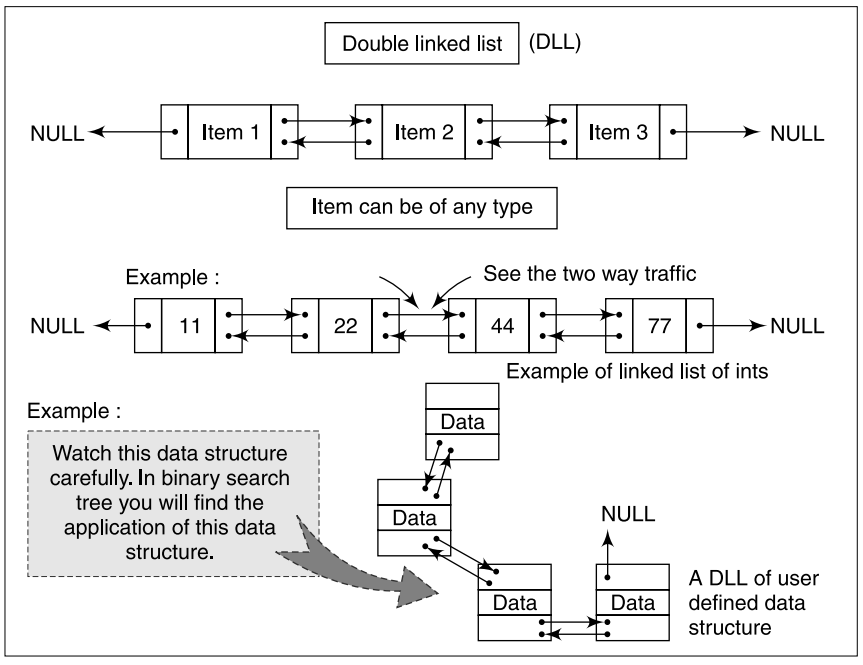


Fig. 3.3

Here is the structure that represents a node of the double linked list.

```
typedef struct node
{
      int info;
      struct node *next;
      struct node *prev;
}node;
```

We will use the same user defined *book structure* above to show how to create a double linked list of book structures. The changes are written in *bold*.

```
typedef struct A_Node_Of_Book_List
{
      Book book_of_this_node;
      Struct A_Node_Of_Book_List *nextbookaddress;//Where is Next Book ?
      Struct A_Node_Of_Book_List *prevbookaddress;//Where is Prev Book ?
}A_Node_Of_Book_List
```

3.3 CIRCULAR LINKED LIST

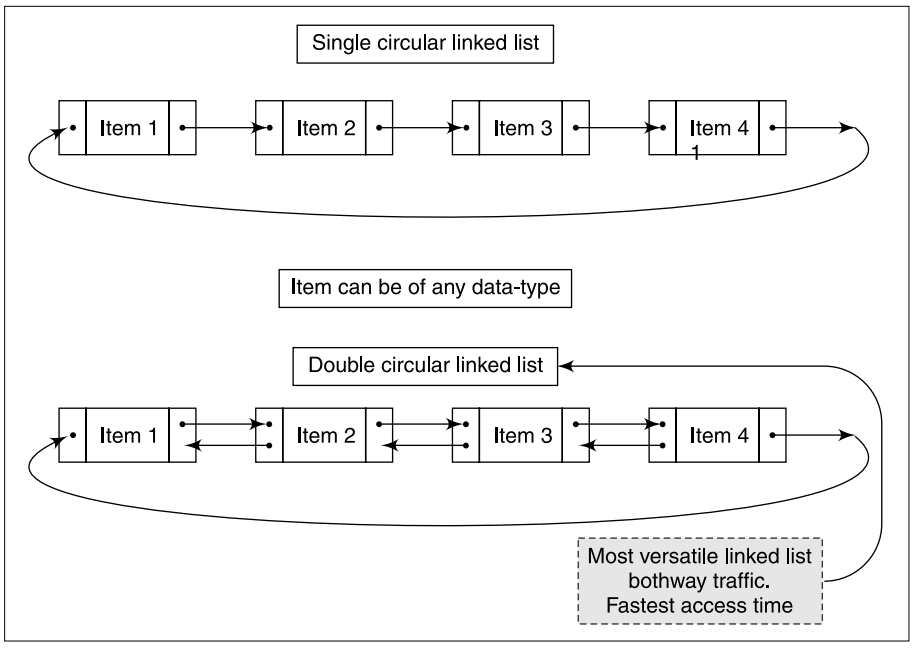


Fig. 3.4

Here is the structure to represent a node of the circular linked list.

```
typedef struct node
{
      int info;
      struct node *next;
}node;
```

The *circular linked list* structure representation looks no different from its single linked list sister. The only difference is that in the circular linked list, the last node's next pointer points to the first node.

3.4 WHAT DO YOU MEAN BY ARRAY OF LINKED LISTS?

Surprised to see such a question? An array of the linked lists is nothing but the array of head pointers of the few linked lists. If you remember, in C, for a 2D array all rows need to have same number of columns. But using an array of linked list we can create such a 2D array that will have variable number of columns in each row. In some languages, for example, C# this type of arrays are in-built and are known as '*Jagged Arrays*'.

Jagged arrays can be used

- to store sparse matrix in a very space efficient manner

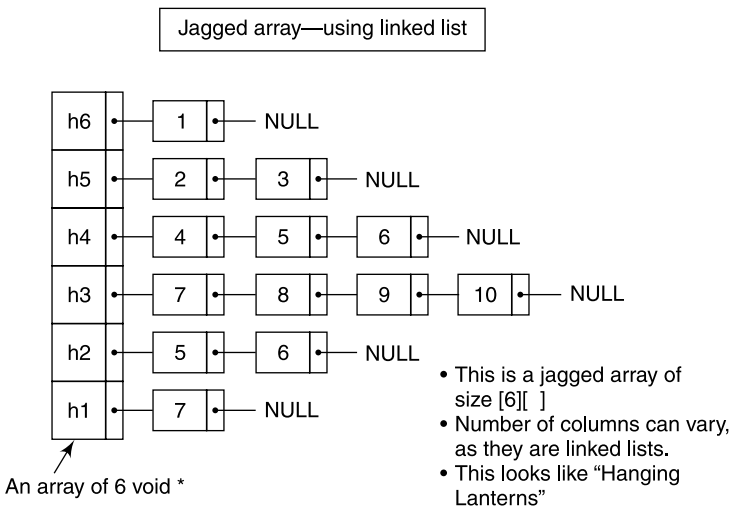


Fig. 3.5

- to use sparse matrix for solving the sparse linear systems in different areas
- to represent a dictionary of words where each word will have their synonyms and antonyms
- to store the location of occurrences of a character in a string

3.5 LINKED LIST IN C AND PREDICTORS

A predictor is a function that returns a Boolean value depending on some condition. Let me explain this. Suppose we want to find all those elements in an integer linked list whose square is less than 100. Then we need to write a function that checks whether the square of the argument passed is less than 100 or not. That function is called a *predictor*.

Predictor function LOGIC can be anything as you can see. So to implement this LOGIC we use function pointer in C. The function pointer passed to the linked list function will remain same. We will just change the function that pointer actually points to. Here is a typical prototype of a linked list function that accepts a function pointer to match a particular set of elements.

```
void display_All(list *, predictor function pointer)
```

3.6 LINKED LIST FUNCTION PHILOSOPHY OF THIS CHAPTER

No matter how much we go ahead with linked list, the plain old C array is always a better choice as far as simplicity and element accessing efficiency of the code is concerned. This fact is validated at the end of the chapter with graphs. The functions developed and discussed in this chapter are designed such that they allow you to manipulate linked list as if it is a close cousin of a C-array. The function names are almost similar to the methods of generic sequence container classes(vector<> and list<>) in STL. One more thing is worth noticing. *Readability of a code is much more important at the cost of slightly low efficiency, in a multi developer environment. The functions in this chapter and the chapters ahead, especially Trees and Graphs follow this strategy.*

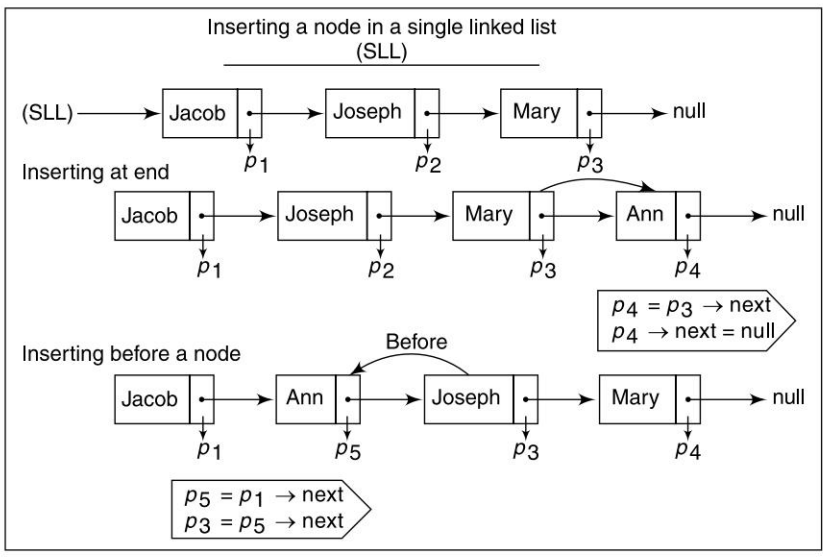


Fig. 3.6

Here is the flow chart.

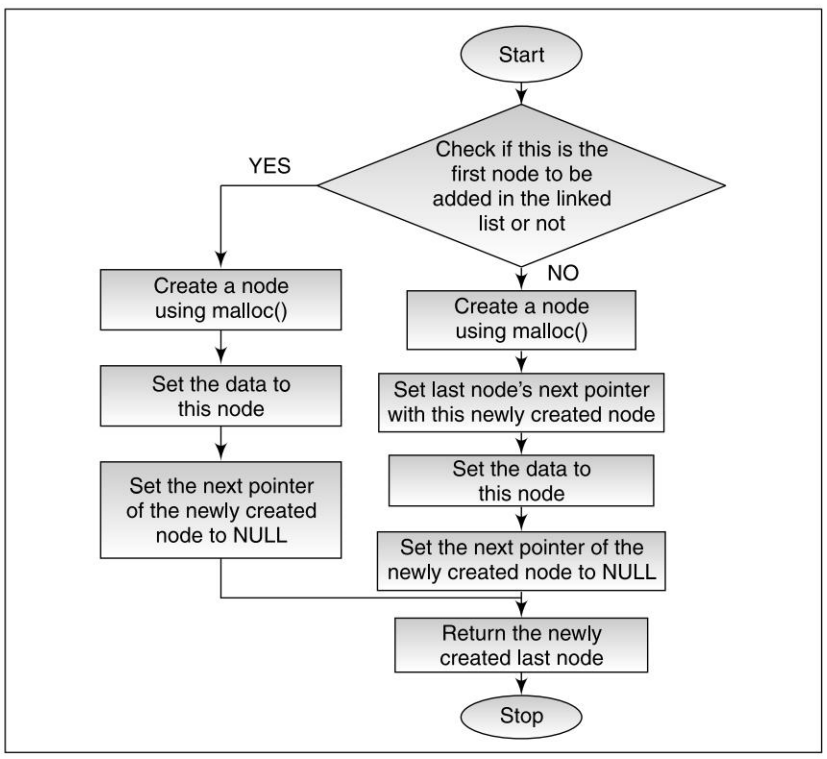


Fig. 3.7

3.7 HOW TO INSERT A NODE AT THE END OF A SINGLE LINKED LIST—THE NODE MAY BE THE FIRST NODE OF THE LINKED LIST

Here is the code to append a node at the end of a single linked list. The first thing that has to be done is to allocate memory for the new node. Then we have to check if that is the first node or not. These functions are given names similar to their cousins in STL library for C++.

```
node* push_back(node *last,int info)
{
      if(last==NULL)//If this is the first node
      {
            //Create Memory Space for this new node
            last = (node *)malloc(sizeof(node));
            last->data = info;
            //Terminate the node properly using NULL
            last->next = NULL;
            //return the back of the list
            return last;
      }
      //If it is not the first node
      else
     {
            //Creation of new node
            node *p = (node *)malloc(sizeof(node));
            if(p)//Check if memory is available for this node
            {
                  //The last node from now on will be p
                  last->next = p;
                  p->data = info;
                  //The list will be terminated by NULL
                  p->next = NULL;
            }
            return p;//Returning the end of the linked list
      }
}
```

3.8 HOW TO INSERT A NODE AT THE FRONT OF THE SINGLE LINKED LIST

Here we are maintaining an integer linked list. We can create a linked list of any data type.

```
node* push_front(node *h,int info)
{
      //Allocating memory for the new node
      node *p = (node *)malloc(sizeof(node));
      //Assigning the next pointer to the previous header of the list.
      p->next = h;
      //Assigning the data
      p->data = info;
      //Returning the newly created list header.
      return p;
}
```

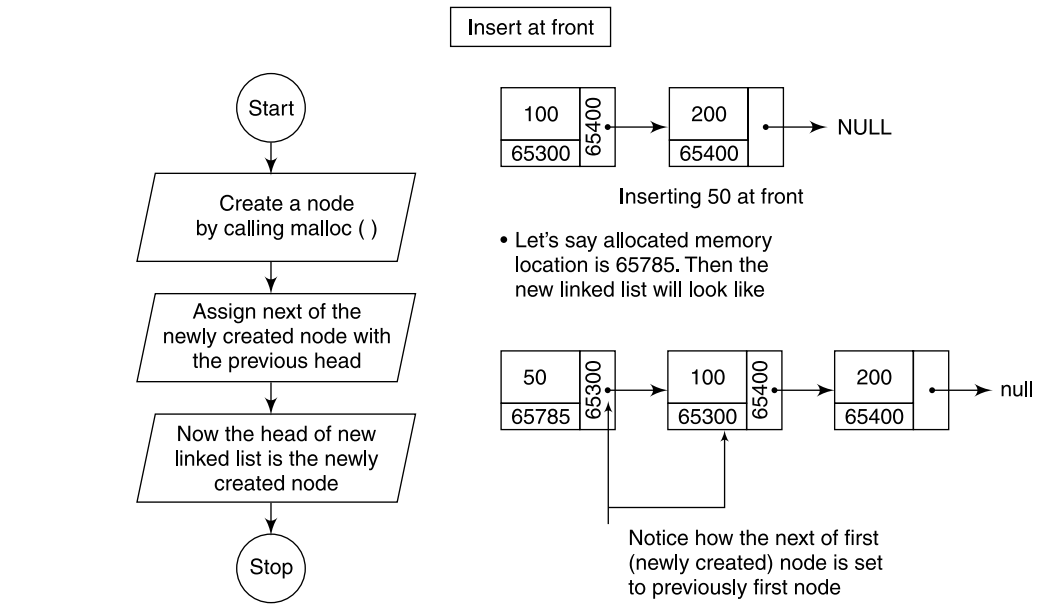


Fig. 3.8

3.9 HOW TO FIND THE FRONT ELEMENT OF THE SINGLE LINKED LIST

This is a wrapper function. This increases the readability to the code.

```
//returns the front element of the list
int front_element(node *h)
{
      //where h points to the list header.
      return first(h)->data;
}
```

Here we have passed the header node and here is the definition of the function first

```
node* first(node *h)
{
      return h;
}
```

Please notice that the function first is a *wrapper function*. This function is used to give increase readability of the program.

3.10 HOW TO FIND THE BACK ELEMENT OF THE SINGLE LINKED LIST

```
//returns the front element of the list
//This is a wrapper function that allows us to write more conceptual
//code.
int back_element(node *h)
{
      return last(h)->data;
}
```

Here we have passed the header node and here is the definition of the function last

```
//This function returns the address of the last node of the list.
//This is a wrapper function.
node* last(node *h)
{
      node *p = h;
      for(;p->next!=NULL;p=p->next);
      return p;
}
```

3.11 HOW TO TRAVERSE THE SINGLE LINKED LIST

```
//Display the list of numbers
void display(node *h)
{
      node *p = h;
      for(;p!=NULL;p=p->next)
          printf("Value = %d Address %u Next Address %u\n",
          p->data,p,p->next);
}
```

3.12 HOW TO COUNT THE NUMBER OF NODES IN A SINGLE LINKED LIST

```
int count(node *h)
{
      int numberofnodes=0;
      node *p = h;
      if(p==NULL)
            return 0;
      else
      {
            for(;p!=NULL;p=p->next)
                  numberofnodes++;
            return numberofnodes;
      }
}
```

Note: *Have you noticed that counting the nodes as in this above function is a function of O(n) time complexity. As C allows global variable, we can keep a global variable that will keep the track of nodes as we add or delete them. Then the counting problem can be solved at O(1) time but this process involves some amount of extra work in the other processes like insert, delete or add, which is acceptable, because incrementing a variable by unity is very trivial and is of O(1) complexity.*

3.13 HOW TO FIND THE FREQUENCY OF AN ITEM IN A SINGLE LINKED LIST

```
//This function returns the frequency of a particular element
int frequency(node *h,int value)
{
      int freq=0;
```

```
      node *p = h;
      for(;p!=NULL;p=p->next)
            if(p->data==value)
                  freq++;
      return freq;
}
```

3.14 HOW TO SEARCH A PARTICULAR ITEM IN THE SINGLE LINKED LIST

This function returns the location index of the variable *s* as in an array.

```
int searchindex(node *h,int s)
{
      int search_status = NOTFOUND;
      int c=0;
      node *p = h;
      for(;p!=NULL;p=p->next)
      {
            c++;
            if(p->data==s)
            {
                  search_status = FOUND;
                  break;
            }
      }
      if(search_status == FOUND)
            return c;
      else
     //The item is not found so we are returning an impossible index -1
            return -1;
}
```

3.15 HOW TO FIND THE ADDRESS OF A PARTICULAR NODE IN A SINGLE LINKED LIST

```
//This function returns the address of the
//variable at a particular location
node* get_address(node *h,int index)
{
      node *p = h;
      int c = 0;
      for(;p!=NULL;p=p->next)
      {
            c++;
            if(c==index)
                  break;
      }
      return p;
}
```

3.16 HOW TO INSERT NODES AT A PARTICULAR LOCATION IN A SINGLE LINKED LIST

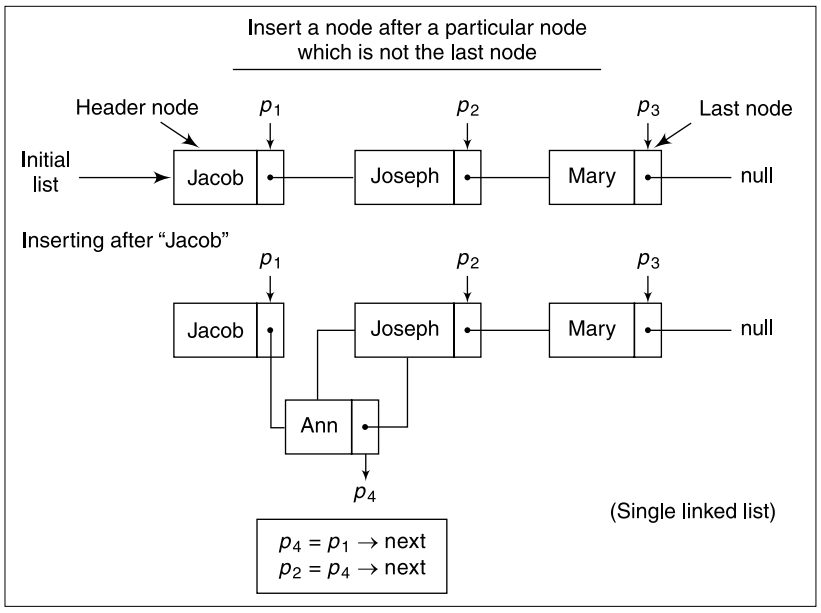


Fig. 3.9

```
//inserting a node at a particular location
node* insert(node *h,int location,int info)
{
      int c=0;
      node *p=h;
      node *r=p;
      node *q=(node *)malloc(sizeof(node *));
      if(location<count(h))
      {
            //Finding out the location where it matches
            //the given location
            for(;p!=NULL;p=p->next)
            {
                  c++;
                  if(c==location)
                        break;
            }
            q->next = p->next;
            p->next = q;
            q->data = info;
      }
      return r;
}
```

This function can be used to insert an element before or after a particular element in the list. We will discuss that now.

3.17 HOW TO INSERT NODES BEFORE A PARTICULAR NODE IN A SINGLE LINKED LIST

This case is nothing but a special case of inserting at a particular location.

To insert an element before a particular element, we need to subtract 1 from the location of the item before which we want to insert the element.

```
puts("Enter the value of the before which you want to\
       insert this node :");
scanf("%d",&sv);
//Finding the location of the integer before which
//we have to insert the number
loc = searchindex(c,sv);
if(loc!=-1)//Checking if the element at all exists
{
      puts("Enter the value for this new node :");
      scanf("%d",&value);
      //Insert before that element.
      //Please note that this insert before will not
      //work if you want to insert a node before the
      //first node (Header Node) of the list.
      //In that case you shall have to use push_front()
      //function.
      c = insert(c,loc-1,value);
      puts("Inserted before the specified node");
}
```

3.18 HOW TO DISPLAY ALL THE CONTENTS OF A SINGLE LINKED LIST

The next pointer of the last node contains NULL. That means until the next value of the node pointer is NULL the list exists. So to display the contents of the list the C code is

```
void display(node *h)
{
      node *p = h;
     //Let's travel towards the end,
     //till we encounter a null value
      for(;p!=NULL;p=p->next)
            printf("Value = %d Address %u Next Address %u\n",
                 p->data,p,p->next);
}
```

3.19 HOW TO FIND THE MAXIMUM ELEMENT FROM A SINGLE LINKED LIST

```
int findmax(node *h)
{
      node *p;
```

```
int max=0;
p = h;
max = p->data;
while(p!=NULL)
{

      //Is the current data value more than the previous
      if(p->data>max)
      {
            max=p->data;
      }
      //Let's move to the next memory block.
      p = p->next;
}
return max;
}
```

3.20 HOW TO FIND THE MINIMUM ELEMENT FROM A SINGLE LINKED LIST

```
int findmin(node *h)
{
      node *p;
      int min=0;
      p = h;
      min = p->data;
      while(p!=NULL)//Rotate till we come to the end of the list.
      {
            if(p->data<=min)
            {
                  min=p->data;
            }
            p = p->next;
      }
      return min;
}
```

//Please note that we are using a Single Linked List of integers to show the typical operations possible.
*//All these operations are logically (**NOT syntactically**) Valid for all other user defined data types.*

3.21 HOW TO EDIT THE CONTENT OF A PARTICULAR NODE WITH A GIVEN VALUE

```
node* replace(node *h,int location,int info)
{
      int c=0;
      node *p=h;
      node *r=p;
      node *q=(node *)malloc(sizeof(node *));
      if(location<count(h))
```

```
{
      //Finding the location of the target node
      for(;p!=NULL;p=p->next)
      {
            c++;//increase the count
            if(c==location-1)//Are we on the target node yet ?
                  break;
      }
            q = p->next;
            q->next = p->next->next;
            q->data = info;
      }
      return r;
}
```

3.22 HOW TO WRITE A FUNCTION TO MERGE TWO LINKED LISTS

```
//This function merges two-linked list as per user requirement
/*
 merge (l1,l2,s1,f1,s2,f2)
 this will merge two single linked lists.
 Suppose the lists are l1= 12,13,14,15 and l2 = 16,17,18,19

 If we want the merged list to be 12 13 18 19 then we should
 call merge like

 merge(l1,l2,1,2,3,4)

 if we want the list to be an end to end merge where the resultant
 list will be
 12 13 14 15 16 17 18 19

 then we should call merge as

 merge(l1,l2,1,count(l1),1,count(l2))

 If we want the merge list to be like

 12 13 14 15 17

 Notice here only one element from the second list is being merged
 with the first list. In this
 case we should call merge like

  merge(l1,l2,1,count(l1),2,2);

*/

node* merge(node *list_1,node *list_2
,int start_1,int finish_1,int start_2,int finish_2)
```

```
{
      int i;
      node *p = NULL,*q;
      //Putting the first value in the
      //merged linked list
      p = push_back(p,get_value(list_1,start_1));
      q = p;
      for(i=start_1+1;i<=finish_1;i++)
            p = push_back(p,get_value(list_1,i));
      for(i=start_2;i<=finish_2;i++)
            p = push_back(p,get_value(list_2,i));
      return q;
}
```

3.23 HOW TO WRITE A FUNCTION TO INSERT A LIST WITHIN ANOTHER LIST

Let me explain the question. Suppose there are two linked lists whose contents are like

11 12 13 14 24 34 44 54

and 15 16 17 18. Now we want to insert this second list within the first list starting from a customized position and up to a defined number of terms.

```
node* insertOneinAnother(node *list_1,node *list_2
                         ,int start_1,int howmany)
{
      int i=0;
      int x=0;
      for(i=1;i<=howmany;i++)
      {
            //Getting the i^th value from the second list.
            x=get_value(list_2,i);
            //Note that how insert() is used
            list_1 = insert(list_1,start_1+i-1,x);
      }
      return list_1;
}
```

Here is how to call this function:

```
      clrscr();
      //creating the first list
      a = push_back(a,11);
      c = a;//This assignment means c will point to the head of the list
      a = push_back(a,12);
      a = push_back(a,13);
      a = push_back(a,14);
      a = push_back(a,24);
      a = push_back(a,34);
      a = push_back(a,44);
      a = push_back(a,54);
      //creating the second list
      d = push_back(d,15);
```

```
e = d; //This assignment means e will point to the head of the list
d = push_back(d,16);
d = push_back(d,17);
d = push_back(d,18);

//inserting first 3 elements of the second list starting
//after the third element in the first list
c = insertOneinAnother(c,e,3,3);
display(c);
getch();
```

This above code will display

```
Value = 11 Address 2280 Next Address 2288
Value = 12 Address 2288 Next Address 2296
Value = 13 Address 2296 Next Address 2376
Value = 15 Address 2376 Next Address 2384
Value = 16 Address 2384 Next Address 2392
Value = 17 Address 2392 Next Address 2304
Value = 14 Address 2304 Next Address 2312
Value = 24 Address 2312 Next Address 2320
Value = 34 Address 2320 Next Address 2328
Value = 44 Address 2328 Next Address 2336
Value = 54 Address 2336 Next Address 0
```

The inserted entries are bolded and shaded.

If you want to insert the entire second list in the first starting after the third element in the first list then the code above will change like

```
//Notice how count() is used.
c = insertOneinAnother(c,e,3,count(e));
```

Always we should try to write functions that can be reused as much as possible.

3.24 HOW TO SWAP THE HEAD AND TAIL NODE (I.E. THE FIRST AND THE LAST NODE) OF THE SINGLE LINKED LIST

```
//This function swaps the head and the tail
//or the front and the back element of the list
node* swap_head_tail(node *h)
{
      node *p = h;
      //Holding the front element data in a temporary variable
      int temp = p->data;
      //Now putting the back element value in the front element.
      p->data = back_element(h);
      //Lets go back to back element. As the function back_element()
      //is read only, we can't use this for setting the values.

      //Notice the ; we just rotate here till the end.
      for(;p->next!=NULL;p=p->next);
      p->data = temp;
      return p;
}
```

3.25 HOW TO SWAP THE CONTENTS OF ANY TWO OTHER NODES APART FROM HEAD AND TAIL

```
//This function swaps the contents of any two nodes in the list
//apart from head and tail. For swapping head and tail info we need to
//use swap_head_tail().

node* swap(node *h,int loc a,int loc b)
{
   int temp;
   node *p=h;
   //Finding the value at the first location
   temp = get_value(h,loc_a);

      //replacing the location a value with loc_b value.
      p=replace(p,loc_a,get_value(h,loc_b));
      //replacing location b value with the value previously present at
      //location a
     p=replace(p,loc_b,temp);
     return p;
}
```

3.26 HOW TO DELETE A PARTICULAR NODE GIVEN BY AN INDEX NUMBER

```
//This function deletes a particular node from a list.
//This function will be used later in more complex deletion scenarios.
node* delete_at(node *h,int location)
{
      int c=0;
      node *p=h;
      node *r=p;
      node *x;
      //Traverse till the end of the list in the worst case
      if(location<count(h))
      {
            //loop till we find the location
            for(;p!=NULL;p=p->next)
            {
                  //lets go to the next node,
                  c++;
                  //See the beauty of the above expression.
                  //We are using linked list
                 //still this expression gives a look and feel
                 //of the plain old handy C array

//Human Readable Location:
//if you want to delete the fifth element
//it is actually the fourth in the list
//so human readable location is 5
//but actual location is 4
```

```
            //Are we on the target node ( The node to be deleted)
            if(c==location-1)
                  break;
      }
      //Identify which memory location
      //to free
      x = p->next;
            //The next node of the deleted node will
            //now be the next node.
            p->next = p->next->next;
            //free that location
            free(x);
      }
      return r;
}
```

3.27 HOW TO DELETE A RANGE OF ELEMENTS FROM A LIST

```
//deletes a particular range of elements
node* delete_range (node *h,int start,int finish)
{
      node *p=h;
      int c=0;
      int k=0;
      for(c=start,k=0;c<=finish;c++,k++)
      //Lets see how the following statement work
      //At first when c = 0 and k = 0 then c-k = 0
      //So the first element of the list is deleted.
      //Next time again the first element is deleted and
      //so on. Thus we end up deleting a range of numbers
      //specified by start and finish.
            delete_at(h,c-k);
      return p;
}
```

Notice carefully how this function uses **`delete_at()`** *internally.*
***Try Yourself*:** *Try to draw the list for each loop status. That will help you visualize the concept better.*

3.28 HOW TO DELETE ALTERNATE ELEMENTS FROM A SINGLE LINKED LIST

Suppose you have the first 100 natural numbers stored in a single linked list. Now we want to keep only the odd elements in the list. So the even elements need to be deleted. In these types of situations, this function will prove to be handy.

```
//deletes alternate elements from the list
node* delete_alternate(node *h, int start,int finish)
{
      node *p=h;
      int c=0;
      for(c=start;c<=finish;c++)
```

```
      //carefully notice the difference in the call to
      //delete_at() function.
            delete_at(h,c);
      return p;
}
```

3.29 HOW TO MAKE THE LIST ENTRIES UNIQUE

```
//ORACLE function distinct is used to display
//the distinct entries in a database table.
//we will use the same function name here

node* distinct(node *list)
{
      node *a=NULL;
      int i=0;
      int j=0;
      for(i=1;i<count(list);i++)
      {
            //For each element check whether it
            //occurred in any other place in the list or not

            for(j=1;j<count(list);j++)
            {
                  //When i==j the element is not a copy,
                  //rather it is the original element.
                  //For all other cases if the values match then
                  //that element is nothing but a copy of the sought
                  //original element. So we shall have to delete that
                  //entry.
                  if(i!=j && get_value(list,i)==get_value(list,j))
                  {
                        list = delete_at(list,j);
                  }

            }
      }
      return list;//The list now contains no duplicate entry
}
```

3.30 HOW TO DELETE THE FIRST ELEMENT OF THE LIST

```
//Deletes the first element
node* pop_front(node *h)
{
      node *x = h;//identifying the first node that has to be freed
      node *p = h->next;
      free(x);//freeing the memory space.
      return p;
}
```

3.31 HOW TO DELETE THE LAST ELEMENT OF THE LIST

```
//Deletes the last node
node* pop_back(node *h)
{
      node *p = h;
      node *r = p;
      for(;;)
      {
            //Look out if we are on the last but one node or not
            if(p->next->next==NULL)
            {
                  //If yes then free the last node
                  free(p->next->next);
                  p->next = NULL;
                  //come out of the loop
                  break;
            }
            //else continue to travel
            p=p->next;
      }
      return r;
}
```

3.32 LINKED LIST AND PREDICATES

Think of your smart mobile phone that allows you to send you SMS to different list of friends at a single shot. It basically keeps a list of friends and depending on your response it creates a predicate function that determines whom to send the SMS and whom not to.
Here is such a predicate.

```
bool canSend(int ID)
{
      enum {NO,YES};
      //Lets assume that get_value returns a PhoneBook Entry Object
      //that has a gender property which is a char (M/F)
      if(get_value(PhoneBook,ID).gender = = 'M')
            return YES;
      else
            return NO;
}
```

The above function takes an integer ID that depicts the ID of the current phone book entry. If the user chooses to send the SMS only to men, then the phone's list processor internally calls something like the above predicate method.

Say an insurance company chooses to give special discount to their customers, who are over 30, married and earn less than 20,000 per month. In a typical scenario, all these details will be stored in a linked list of customer structures. We need to write a predicate function that will decide for ourselves whether the insurance company (our client for the time being) can give the discount or not.

```
bool canGiveDiscount(int ID)
{
      enum {NO,YES};
      Customer CurrentCustomer = get_value(Customers,ID);
      if(CurrentCustomer.isMarried == YES && CurrentCustomer.age >=30)
            return YES;
      else
            return NO;
}
```

Now let's see how to call these predicates:

```
for(j=1;j<count(Customers);j++)
{
      if(canGiveDiscount(j) = = YES)
      {
            //Process Discount Logic here
      }
      else
      {
            //All other unfortunate customers
            //who are out of this scheme!
            //data will be processed here
      }
}
```

3.33 WHAT ARE THE ATTRIBUTES AND METHODS OF A POLYNOMIAL AS A DATA STRUCTURE?

Polynomial as a data structure
Attributes : Coefficient
Power

If the polynomial is of one variable then there will be only power for the single variable.

In case the polynomial is *monic* (A polynomial is monic if all the coefficient is one) , then the coefficient is one. Any constant towards the tail of the polynomial is nothing but a term with power zero.

Operations possible on polynomials are as follows.

Addition of two polynomials ,

Time Complexity : O(m+n), where m, n are the length of the polynomials

Multiplication of two polynomials

*Time Complexity : O(m*n), where m, n are the length of the polynomials*

Checking if two polynomials are same or not

Time Complexity : O(m) where m is the length of the smaller polynomial.

All the above-mentioned operations are performed on the polynomials represented with the linked list. The next few sections of this chapter deal with this topic at length. After we learn about the representation of single variable polynomial we will move to deal with the three variable polynomials and then we will explore their applications to solve some vector calculus problems.

3.34 HOW TO REPRESENT A POLYNOMIAL USING A SINGLE LINKED LIST

Here are the structures that are used to represent a polynomial.

```
typedef struct polynode
{
   float coef;
   int   power;
} polynode;
```

A polynomial can be thought of a chain or a list of polynodes. The struct polynomial below depicts a polynomial.

```
typedef struct polynomial
{
   polynode info;
   struct polynomial *next;
} polynomial;
```

3.35 POLYNOMIAL TOOL BOX

Example 3.1 ***Write a program that performs the following things:***

1. ***Accepts a polynomial from the user***
2. ***Prints the polynomials***
3. ***Adds two polynomials***
4. ***Multiplies two polynomials***
5. ***Evaluates a polynomial for a specific value***
6. ***Differentiates a polynomial***
7. ***Integrates a polynomial without limits (Indefinite integration, i.e. in this case it will return another polynomial)***
8. ***Integrates a polynomial with limits***
9. ***Finds the composite function denoted by fog where one polynomial represents f(x) and the other represents g(x)***

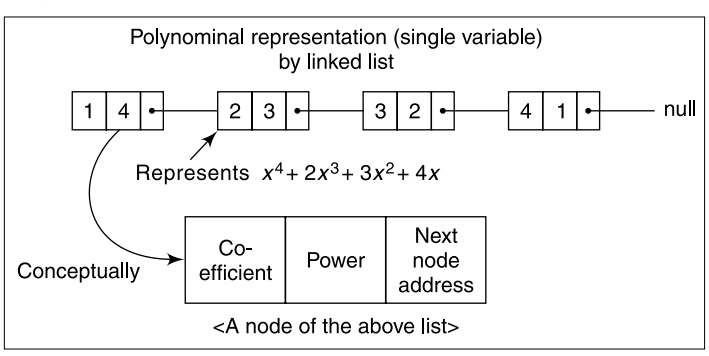


Fig. 3.10

Solution

```
typedef struct PolyNode
{
   double coeff;
   double power;
   struct PolyNode *next;
}PolyNode;
```

3.36 HOW TO ADD A NEW TERM TO A POLYNOMIAL

Polynomial AddNewTerm(Polynomial poly, double c,double p)

```
Polynomial AddNewTerm(Polynomial poly, double c,double p)
{
      int flag = 0;
      if(poly==NULL)
      {
            poly = (Polynomial) malloc(sizeof(PolyNode));
            poly->coeff = c;
            poly->power = p;
            poly->next = NULL;
            return poly;
      }
      else
      {
            Polynomial temp = poly;
            for(;temp!=NULL;temp=temp->next)
            {
                  if(temp->power == p)
                  {
                        temp->coeff+=c;

                        flag = 1;
                        return poly;
                  }
            }
            if(flag==0)
            {
                  Polynomial t =
(Polynomial)malloc(sizeof(PolyNode));
                  t->coeff = c;
                  t->power = p;
                  t->next = poly;
                  return t;
            }
      }
}
```

3.37 HOW TO ADD TWO POLYNOMIALS AND RETURN THEIR SUM

```
Polynomial AddPoly(Polynomial first,Polynomial second)
{
      Polynomial sum = NULL;
      Polynomial tf = first;
      Polynomial ts = second;
      for(;tf!=NULL;tf=tf->next)
            sum = AddNewTerm(sum,tf->coeff , tf->power);
      for(;ts!=NULL;ts=ts->next)
            sum = AddNewTerm(sum,ts->coeff , ts->power);
      return sum;
}
```

3.38 HOW TO MULTIPLY TWO POLYNOMIALS

```
Polynomial MultiplyPoly(Polynomial first,Polynomial second)
{
  Polynomial f=first,s=second,result=NULL;
  for(;f!=NULL;f=f->next)
    for(;s!=NULL;s=s->next)
      result = AddNewTerm(result,f->coeff * s->coeff, f->power + s->power);
  return result;
}
```

3.39 HOW TO FIND THE DIFFERENTIATION OF A POLYNOMIAL

```
Polynomial DiffPoly(Polynomial poly)
{
      Polynomial diffpoly = NULL;
      Polynomial t = poly;
      for(;t!=NULL;t=t->next)
            diffpoly = AddNewTerm(diffpoly,t->coeff*t->power,t->power-1);
      return diffpoly;
}
```

3.40 HOW TO CALCULATE THE INTEGRAL OF A POLYNOMIAL

```
Polynomial IntPoly(Polynomial poly)
{
      Polynomial intpoly = NULL;
      Polynomial t = poly;
      for(;t!=NULL;t=t->next)
            intpoly = AddNewTerm(intpoly,t->coeff/(t->power + 1),t->power+1);
      return intpoly;
}
```

3.41 HOW TO EVALUATE THE VALUE OF THE POLYNOMIAL AT A VALUE

```
double EvaluatePoly(Polynomial poly,double value)
{
      Polynomial t = poly;
      double sum = 0;
      for(;t!=NULL;t=t->next)
            sum+=t->coeff*pow(value,t->power);
      return sum;
}
```

3.42 HOW TO FIND THE DEFINITE INTEGRAL VALUE OF A FUNCTION

```
double DefIntPoly(Polynomial poly, double LowerBound, double
UpperBound)
{
      Polynomial IntegratePolynomial = IntPoly(poly);
      return EvaluatePoly(IntegratePolynomial,UpperBound)
           - EvaluatePoly(IntegratePolynomial,LowerBound);
}
```

3.43 HOW TO DISPLAY A POLYNOMIAL

```
void display(Polynomial p)
{
      Polynomial t = p;
      for(;t!=NULL;t=t->next)
            if(t->coeff!=0)
                  printf("%.0f*(x)^%.0f + ",t->coeff,t->power);
}
```

3.44 HOW TO FIND THE VALUE OF A COMPOSITE FUNCTION

```
double CompositeFun(Polynomial f, Polynomial g,double value)
{
      return EvaluatePoly(f,EvaluatePoly(g,value));
}
```

Example 3.2 ***Write a program that performs the following:***

1. ***Accept a polynomial of three variables (x,y,z) from the user***
2. ***Prints such a polynomial***
3. ***Adds two such polynomials***
4. ***Multiply two such polynomials***
5. ***Evaluate such a polynomial for a specific value of x, y and z***
6. ***Differentiate such a polynomial with respect to x, y and z***
7. ***Integrate a polynomial without limits (Indefinite Integration, in this case it will return another polynomial)***

8. There are three polynomials f(x,y,z), g(x,y,z), h(x,y,z) and s(x,y,z). then write a function to calculate f(g(x,y,z),h(x,y,z),s(x,y,z)). This is known as the composite function.

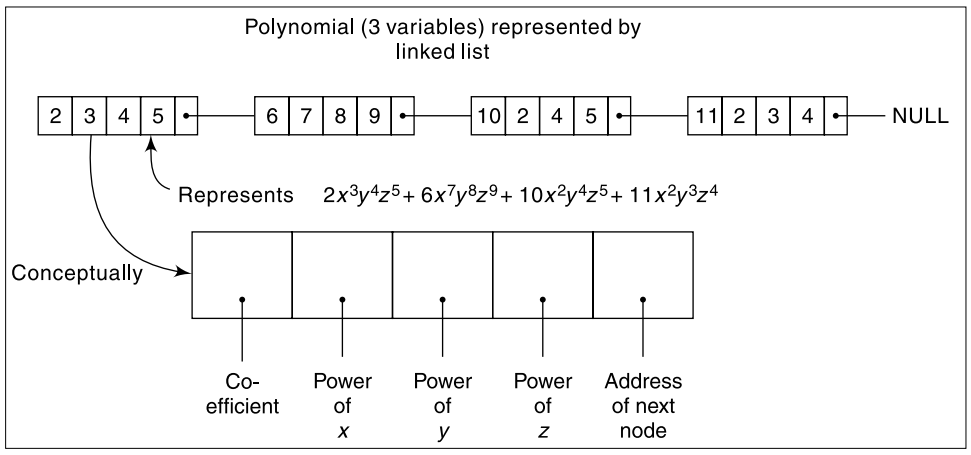


Fig. 3.11

```
//This structure represents
//a particular node of a three variable
//polynomial.
```

Solution

```
typedef struct PolyNode
{
      double coeff;
      double power_x;
      double power_y;
      double power_z;
      struct PolyNode *next;
}PolyNode;

typedef PolyNode* Polynomial;
```

3.45 HOW TO ADD A NEW TERM TO A POLYNOMIAL

```
Polynomial AddNewTerm(Polynomial poly, double c,double px,double py,double pz)
{
      int flag = 0;
      if(poly==NULL)
      {
            poly = (Polynomial) malloc(sizeof(PolyNode));
            poly->coeff = c;
            poly->power_x = px;
```

```
            poly->power_y = py;
            poly->power_z = pz;
            poly->next = NULL;
            return poly;
      }
      else
      {
            Polynomial temp = poly;
            for(;temp!=NULL;temp=temp->next)
            {
                  if(temp->power_x == px
                  && temp->power_y == py
                  && temp->power_z == pz)
                  {
                        temp->coeff+=c;

                        flag = 1;
                        return poly;
                  }
            }
            if(flag==0)
            {
                  Polynomial t = (Polynomial)malloc(sizeof(PolyNode));
                  t->coeff = c;
                  t->power_x = px;
                  t->power_y = py;
                  t->power_z = pz;
                  t->next = poly;
                  return t;
            }
      }
}
```

3.46 HOW TO ADD TWO POLYNOMIALS OF THREE VARIABLES

```
Polynomial AddPoly(Polynomial first,Polynomial second)
{
      Polynomial sum = NULL;
      Polynomial tf = first;
      Polynomial ts = second;

      for(;tf!=NULL;tf=tf->next)
            sum = AddNewTerm(sum,tf->coeff , tf->power_x,tf->power_y ,tf->power_z );
      for(;ts!=NULL;ts=ts->next)
            sum = AddNewTerm(sum,ts->coeff , ts->power_x,ts->power_y ,ts->power_z );
      return sum;
}
```

3.47 HOW TO MULTIPLY TWO POLYNOMIALS OF THREE VARIABLES

```
Polynomial MultiplyPoly(Polynomial first,Polynomial second)
{
```

```
      Polynomial f=first,s=second,result=NULL;
      for(;f!=NULL;f=f->next)
            for(;s!=NULL;s=s->next)
                  result = AddNewTerm(result,f->coeff * s->coeff
                  ,f->power_x + s->power_x,f->power_y + s->power_y
                  ,f->power_z + s->power_z);
      return result;
}
```

3.48 HOW TO DIFFERENTIATE A POLYNOMIAL WITH RESPECT TO *X*

```
Polynomial ddx(Polynomial poly)
{
      Polynomial diffpoly = NULL;
      Polynomial t = poly;
      for(;t!=NULL;t=t->next)
            diffpoly = AddNewTerm(diffpoly,t->coeff*t->power_x
                  ,t->power_x-1,t->power_y,t->power_z);
      return diffpoly;
}
```

3.49 HOW TO DIFFERENTIATE A POLYNOMIAL WITH RESPECT TO *Y*

```
Polynomial ddy(Polynomial poly)
{
      Polynomial diffpoly = NULL;
      Polynomial t = poly;
      for(;t!=NULL;t=t->next)
            diffpoly = AddNewTerm(diffpoly,t->coeff*t->power_y
                  ,t->power_x,t->power_y-1,t->power_z);
      return diffpoly;
}
```

3.50 HOW TO DIFFERENTIATE A POLYNOMIAL WITH RESPECT TO *Z*

```
Polynomial ddz(Polynomial poly)
{
      Polynomial diffpoly = NULL;
      Polynomial t = poly;
      for(;t!=NULL;t=t->next)
            diffpoly = AddNewTerm(diffpoly,t->coeff*t->power_z
                  ,t->power_x,t->power_y,t->power_z-1);
      return diffpoly;
}
```

3.51 HOW TO INTEGRATE THE POLYNOMIAL WITH RESPECT TO *X* ASSUMING THE OTHER TWO VARIABLES ARE KEEPING CONSTANT

```
Polynomial Intx(Polynomial poly)
{
      Polynomial intpoly = NULL;
      Polynomial t = poly;
      for(;t!=NULL;t=t->next)
```

```
            intpoly = AddNewTerm(intpoly,t->coeff/(t->power_x + 1),
t->power_x+1,t->power_y,t->power_z);
      return intpoly;
}
```

3.52 HOW TO INTEGRATE A POLYNOMIAL WITH RESPECT TO *Y* ASSUMING THE OTHER TWO VARIABLES ARE KEEPING CONSTANT

```
Polynomial Inty(Polynomial poly)
{
      Polynomial intpoly = NULL;
      Polynomial t = poly;
      for(;t!=NULL;t=t->next)

            intpoly = AddNewTerm(intpoly,t->coeff/(t->power_y + 1)
             ,t->power_x,t->power_y+1,t->power_z);
      return intpoly;
}
```

3.53 HOW TO INTEGRATE A POLYNOMIAL WITH RESPECT TO *Z* ASSUMING THE OTHER TWO VARIABLES ARE KEEPING CONSTANT

```
Polynomial Intz(Polynomial poly)
{
      Polynomial intpoly = NULL;
      Polynomial t = poly;
      for(;t!=NULL;t=t->next)
            intpoly = AddNewTerm(intpoly,t->coeff/(t->power_z + 1)
                        ,t->power_x,t->power_y,t->power_z+1);
      return intpoly;
}
```

3.54 HOW TO INTEGRATE A POLYNOMIAL WHEN ALL THREE VARIABLES ARE VARYING

```
Polynomial IntPolyxyz(Polynomial poly)
{
      return Intz(Inty(Intx(poly)));
}
```

3.55 HOW TO EVALUATE A POLYNOMIAL AT GIVEN VALUES OF *X*, *Y* AND *Z*

```
double EvaluatePolyxyz(Polynomial poly,double value_x,double value_y,double value_z)
{
      Polynomial t = poly;
      double sum = 0;
      for(;t!=NULL;t=t->next)
            sum+=t->coeff*(pow(value_x,t->power_x) + pow(value_y,t->power_y)
                  + pow(value_z,t->power_z));
      return sum;
}
```

3.56 HOW TO INTEGRATE A POLYNOMIAL WITHIN A GIVEN LIMIT

```
double DefIntPoly(Polynomial poly, double LowerBound_x, double UpperBound_x
                          ,double LowerBound_y, double UpperBound_y
                          ,double LowerBound_z, double UpperBound_z)
{
      Polynomial IntegratePolynomial = IntPolyxyz(poly);
return EvaluatePolyxyz(IntegratePolynomial,UpperBound_x,UpperBound_y,UpperBound_z)
-EvaluatePolyxyz(IntegratePolynomial,LowerBound_x,LowerBound_y,LowerBound_z);
}
```

3.57 SOME APPLICATIONS OF POLYNOMIAL TOOLBOX

How to Calculate the Divergence of a Function in Three Variables

Let $F(F1,F2,F3)$ be a vector field, where $F1$, $F2$ and $F3$ are functions of x, y and z continuously differentiable with respect to x, y and z. Then the divergence of F is defined by

$$\text{div } \mathbf{F} = \nabla \cdot \mathbf{F} = \frac{\partial F_1}{\partial x} + \frac{\partial F_2}{\partial y} + \frac{\partial F_3}{\partial z}$$

The name divergence is well chosen, because it is a measure of how much the vector v spreads out from the point in question. As in the above figure, the vector has a large (positive) divergence.

If the arrows point inwards, then the vector will have a very large (negative) divergence, where $F1$, $F2$ and $F3$ are nothing but functions of x, y and z. So the divergence is nothing but the summation of the partially differentiated polynomials. We have already written functions to find partial differentiation of a polynomial and function to add two polynomials. Combining these two functions we can easily find the divergence of a given vector field.

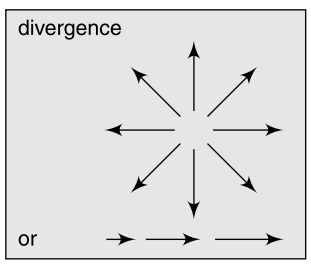


Fig. 3.12

```
//v(x)
void show_divergence(poly vx,poly vy,poly vz)
{
      poly r;
      r = addpoly(ddx(vx),ddy(vy));
      r = addpoly(r,ddz(vz));
      printpoly(r);
}
```

3.58 HOW TO FIND THE CURL OF A FUNCTION OF THREE VARIABLES

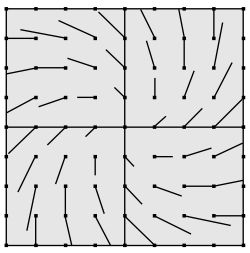


Fig. 3.13

With the functions declared above we can write some highly readable and efficient routines for some electromagnetic functions.

```
//This function displays the Curl in human readable format
void show_curl(poly vx,poly vy,poly vz)
{
      poly r,s,t,w,y,z;
      printf("x cap(");
      r=ddy(vz);
      s=ddz(vy);
      s->info.coef=-s->info.coef;
      t=AddPoly(s,r);
      display(t);
      printf(")");

      printf("+ y_cap(");
      r=ddz(vx);
      s=diffpoly_x(vz);
      r->info.coef=-r->info.coef;
      t=AddPoly(s,r);
      display(t);
      printf(")");

      printf("+ z_cap(");
      r=ddx(vy);
      s=ddy(vx);
      s->info.coef=-s->info.coef;
      t=AddPoly(s,r);
      display(t);
      printf(")");
}
```

3.59 HUGE NUMBERS: APPLICATION OF LINKED LISTS

Write a function that allows the user to create a single linked list of integers with variable number of integers. Use va_arg.

```
//This function creates a single linked list of integers
//passed as arguments. The three dots (…) are known as
//ellipsis and used to tell the compiler that this function
//is capable of accepting variable number of arguments
//(Though of the same type).
//The function scans the integers and put them in a linked list till it
//encounters a -1.

node* createlist(char *msg,...)
{
      node *p=NULL,*q;
      va_list ap;
      int arg;
      int c=0;
      va_start(ap,msg);
```

```
while((arg=va_arg(ap,int))!=-1)
{
      p = push_front(p,arg);
      if(c==0)
        q = p;
      c++;

} nerd
      puts(msg);
      return p;
}
```

Here it is shown how to call this function

```
node *c=NULL;
c = createlist("Huge 1",1,2,3,4,5,6,7,8,9,2,3,2,3,2,4,3,4,3,4,-1);
display(c);
```

for the above code c will now represent a linked list with content

```
1,2,3,4,5,6,7,8,9,2,3,2,3,2,4,3,4,3,4
```

In some cases we need to do arithmetical operations with huge integers. By huge, we mean that the number of digits of the integer is so large that it can't be stored in any built-in C data type. In these scenarios, linked list can prove to be useful.

3.60 HOW TO STORE TWO HUGE INTEGERS AS SINGLE LINKED LIST AND THEN ADD THOSE TWO NUMBERS AND DISPLAY THE SUMMATION

```
//This function adds two HUGE
//positive integers
//like
//huge 1 = 1234567892323243434
//huge 2 = 3434324234243242342

node* addhuge(node *one,node *two)
{
      node *sum=NULL,*s;
      node *ss=NULL,*sss;
      int i=0;
      int z=0;
      int x=0;
      int c=0;
      int y;
      for(i=1;i<count(one);i++)

      {
      sum = push_back(sum,get_value(one,i) + get_value(two,i));
      if(c==0)
            s = sum;
      c++;
}
for(i=1;i<=count(s);i++)
```

```
{
	if(i==1)
	{
		x = get_value(s,1);
		if(x>=10)
			ss = push_front(ss,x-10);
		else
			ss = push_front(ss,x);
		sss = ss;
	}
	else
	{
		if(get_value(s,i-1) >= 10)
		{
			y = get_value(s,i) + 1;
			if(y>=10)
				ss = push_front(ss,y-10);
			else
				ss = push_front(ss,y);
		}
		else
		{
			y = get_value(s,i);
			if(y>=10)
				ss = push_front(ss,y-10);
			else
				ss = push_front(ss,y);

		}

	}

	z = back_element(one) + back_element(two);
	if(back_element(s)>=10)
		z+=1;
	ss = push_front(ss,z);

	return ss;
}
```

To add two huge numbers and to show their sum, here is how to call the above functions..

```
c = createlist("Huge 1",1,2,3,4,5,6,7,8,9,2,3,2,3,2,4,3,4,3,4,-1);
display(c);//Showing the first Huge Number

d = createlist("Huge 2",3,4,3,4,3,2,4,2,3,4,2,4,3,2,4,2,3,4,2,-1);
display(d);//Showing the second Huge Number

puts("Sum");
display(addhuge(c,d));//Showing the sum of the two huge numbers.
```

Try yourself: *Try to find out the multiplication of two such huge numbers using linked list. There are few good algorithms. Among them* ***Karatsuba Algorithm*** *is quite popular among computer scientists. Try to implement that Algorithm using linked lists.*

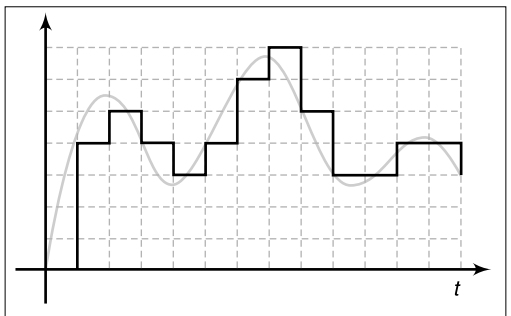
3.61 DIGITAL SIGNAL PROCESSING

How to Model a Digital Signal using Linked List

It takes two attributes to describe a digital signal, where the signal starts on the timeline (i.e. whether the signal is causal, anti-causal or non-causal) and what is the amplitude at any particular point of time.

So a structure can be defined to represent a signal in discrete timeline or to represent a digital signal.

```
typedef struct Signal
{
      int time;
      int amplitude;
      struct Signal *next;
}Signal;
```



The red straight lines in the above figure represent a digital signal

Fig. 3.14

Now we can create a linked list of this structure to represent a digital signal. Basically the linked list code for holding the integer variables need to be modified slightly to represent a signal. Here is the code of the building block functions.

Here only those functions are listed whose source code is slightly changed in order to create a digital signal. Carefully notice that the functions that only modify the content of a node are changed.

3.62 HOW TO FIND THE LENGTH OF A SIGNAL

The function name has been changed to length() because length of a signal makes more sense than count() of a signal.

```
//All digital signals that we handle are finite length signals.
int length(Signal *h)
{
      int count=0;
      Signal *p = h;
      if(p==NULL)
            return 0;
      else
      {
            for(;p!=NULL;p=p->next)
                  count++;
            return count;
      }
}
```

3.63 HOW TO FIND THE INDEX OF A GIVEN AMPLITUDE IN A SIGNAL

```
int searchindex(Signal *h,int amp)
{
      int search_status = NOTFOUND;
      int c=0;
      Signal *p = h;
      for(;p!=NULL;p=p->next)
      {
            c++;
            if(p->amplitude==amp)
            {
                  search_status = FOUND;
                  break;
            }
      }
      if(search_status == FOUND)
            return c;
      else
            return -1;
}
```

3.64 HOW TO ADD A NEW VALUE AT THE END OF A SIGNAL

```
//Notice: The same push_back() function that has been
//used in Integer linked list described above in this chapter
//The changes are highlighted.
Signal* push_back(Signal *last,int t,int amp)
{
      if(last==NULL)
      {
            last = (Signal *)malloc(sizeof(Signal));
            last->time = t;
            last->amplitude = amp;
            last->next = NULL;
            return last;
      }
      else
      {
            Signal *p = (Signal *)malloc(sizeof(Signal));
            if(p)
            {
                  last->next = p;
                  p->time  = t;
                  p->amplitude = amp;
                  p->next = NULL;
            }
            return p;
      }
}
```

3.65 HOW TO ADD A NEW VALUE AT THE FRONT OF A SIGNAL

```
Signal* push_front(Signal *h,int t,int amp)
{
      Signal *p = (Signal *)malloc(sizeof(Signal));
      p->next = h;
      p->time = t;
      p->amplitude = amp;
      return p;
}
```

Digital signal processing finds application in many diverse areas. While working on a stored digital signal, we may need to change particular amplitude (amplitude in a particular location) with another value. Here is the code.

```
Signal* replace(Signal *h,int location,int t,int amp)
{
      int c=0;
      Signal *p=h;
      Signal *r=p;
      Signal *q=(Signal *)malloc(sizeof(Signal *));
      if(location<length(h))
      {
            //
            for(;p!=NULL;p=p->next)

            {
                  c++;
                  if(c==location-1)
                        break;
            }

            q = p->next;
            q->next = p->next->next;
            q->time = t;
            q->amplitude = amp;

      }

      return r;

}
```

3.66 HOW TO RETURN THE FIRST SIGNAL NODE POINTER

```
Signal* first(Signal *h)
{
      return h;
}
```

3.67 HOW TO RETURN THE LAST SIGNAL NODE POINTER

```
Signal* last(Signal *h)
{
      Signal *p = h;
      for(;p->next!=NULL;p=p->next);
      return p;
}
```

3.68 HOW TO INSERT A NODE AT A PARTICULAR LOCATION OF A SIGNAL

```
Signal* insert(Signal *h,int location,int t,int amp)
{
      int c=0;
      Signal *p=h;
      Signal *r=p;
      Signal *q=(Signal *)malloc(sizeof(Signal *));
      if(location<length(h))
      {
            //
            for(;p!=NULL;p=p->next)
            {
                  c++;
                  if(c==location)
                        break;
            }
            q->next = p->next;
            p->next = q;
            q->time = t;
            q->amplitude = amp;

      }
      return r;
}
```

3.69 HOW TO DISPLAY A DIGITAL SIGNAL

```
//Display the digital signal
//For simplicity it is assumed that the time line for this signal
//is entered serially. By that we mean suppose a signal starts at -2 and
//finishes at 4, then it is assumed that the entries started from time
//t = -2.
//The display pattern is
void display(Signal *h)
{
      Signal *p = h;
      for(;p!=NULL;p=p->next)
            printf("x[%d] = %d\n",p->time,p->amplitude);
}
```

3.70 HOW TO GET THE AMPLITUDE OF THE FIRST SIGNAL NODE

```
//returns the first amplitude of the signal
int front_element(Signal *h)
{
      return first(h)->amplitude;
}
```

3.71 HOW TO GET THE AMPLITUDE OF THE LAST SIGNAL NODE

```
//returns the last amplitude of the signal
int back_element(Signal *h)
{
      return last(h)->amplitude;
}
```

3.72 HOW TO FIND FREQUENCY OF A PARTICULAR AMPLITUDE IN A GIVEN SIGNAL

```
int frequency(Signal *h,int amp)
{
      int freq=0;
      Signal *p = h;
      for(;p!=NULL;p=p->next)
            if(p->amplitude==amp)
                  freq++;
      return freq;
}
```

3.73 HOW TO GET THE ADDRESS OF A SIGNAL NODE GIVEN THE INDEX

```
//This function returns the address of the
//signal node at a particular location
Signal* get_address(Signal *h,int index)
{
      Signal *p=h;
      int c=0;
      for(;p!=NULL;p=p->next)
      {
            c++;
            if(c==index)
                  break;
      }
      return p;
}
```

3.74 HOW TO GET THE AMPLITUDE OF A SIGNAL AT A PARTICULAR POINT

```
//This returns the value at a particular address
//It is always nice to have an array like indexing
//facility.
int get_value(Signal *h,int index)
{
      return get_address(h,index)->amplitude;
}
```

3.75 HOW TO CHECK WHETHER A DIGITAL SIGNAL IS EVEN OR NOT

A *digital signal* is said to be even if it is symmetric about the *y* axis. That means, the right half of the signal is a mirror image of the left half with respect to origin. Here is the picture of an even digital signal.

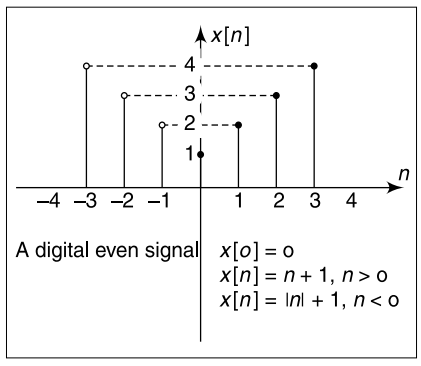


Fig. 3.15

```
int isEven(Signal *x)
{
      int i=0,j=0;
      int even=YES;
      int len = length(x);
      int mid = ceil(len/2);
      for(i=len,j=1;i>=mid;i--,j++)
      {
            if(get_value(x,i)!=get_value(x,j))
            {
                  even  = NO;
                  break;
            }
      }
      return even;
}
```

3.76 HOW TO CHECK WHETHER A DIGITAL SIGNAL IS CAUSAL

A signal is said to *causal signal* if and only if the signal has non zero values for time $t > 0$.
That means before $t = 0$, all the amplitudes are zero for a causal signal. Here is a causal signal shown in the picture.

```
enum {NO,YES};
int isCausal(Signal *x)
{
int causal = YES;
Signal *p=x;
for(;p!=NULL;p=p->next)
{
      //Any value at the negative discrete time zone?
      if(p->time<0)
```

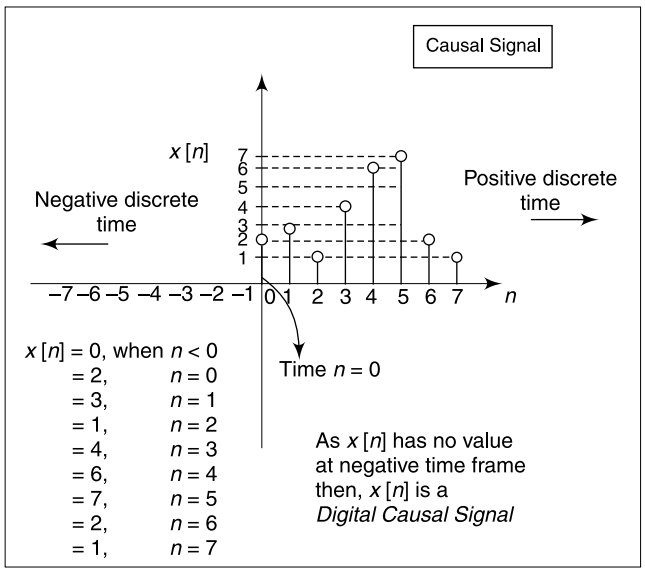


Fig. 3.16

```
            {
                causal = NO;
                break;
            }
        }
        return causal;
}
```

3.77 HOW TO CHECK WHETHER A DIGITAL SIGNAL IS ANTI-CAUSAL

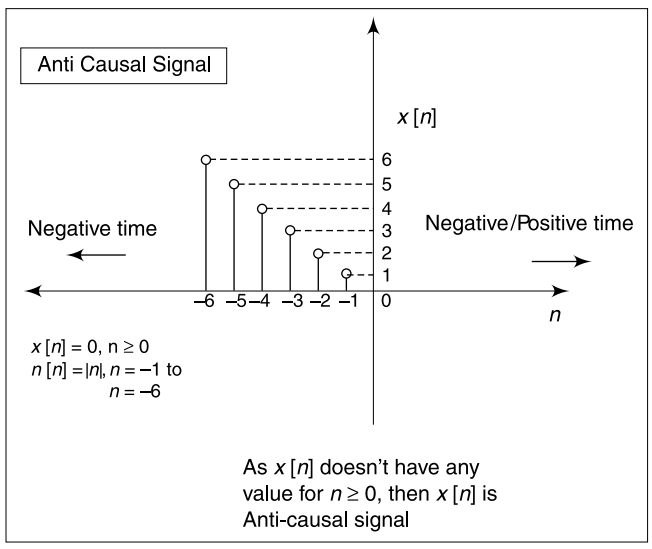


Fig. 3.17

A signal is said to be anticausal if the signal has non-zero amplitudes only in negative time. So for time $t = 0$ or more all the amplitudes of an anti-causal signal are zero. Here is an *anti-causal signal* picture.

```
int isAntiCausal(Signal *x)
{
      int anticausal = YES;
      Signal *p=x;
      for(;p!=NULL;p=p->next)
      {
            //Any amplitude at the positive discreet time zone is
            //greater than zero or not
            if(p->time>=0)
            {
                  anticausal = NO;
                  break;
            }
      }
      return anticausal;
}
```

3.78 HOW TO CHECK WHETHER A DIGITAL SIGNAL IS NON-CAUSAL

Here is a non-causal signal:

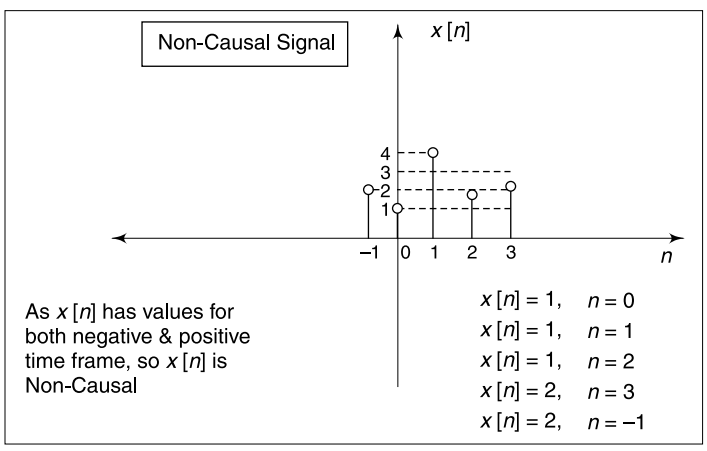


Fig. 3.18

A signal is said to be non-causal if the signal has values in both negative and positive time zone.

So from the definitions we can say that a non-causal signal is a signal which is neither causal nor anti-causal. Using the above two functions we can write a single line function that will tell us whether a signal us non-causal or not. Here is the code:

```
int isNonCausal(Signal *x)
{
      return !isCausal(x) &&  !isAntiCausal(x);
}
```

See how this function `isNonCausal()` is written as the combination of `isCausal()` and `isAntiCausal()`

3.79 HOW TO ADD AT THE END OF A SINGLE CIRCULAR LINKED LIST

To maintain a circular single linked list. We need to maintain a global pointer of type node that will be modified every time we add a node at the front. Let's call this *firstnode*. 26326

```
node* push_back(node *last,int info)
{
      //If this is the first node of the circular linked list.
      if(last==NULL)
      {
            last = (node *)malloc(sizeof(node));
            last->data = info;
            last->next = NULL;
            return last;
      }
      else
      {
            node *p = (node *)malloc(sizeof(node));
            if(p)
            {
                  last->next = p;
                  p->data = info;
                  //The next pointer of the last node
                  //now points to the first node.

                  //First Node is nothing but a pointer
                  //of type node* that is modified every time

                  p->next = FirstNode;
            }
            return p;
      }
}
```

Apart from the above change, everything else will remain same as that of Single Linked List. So those functions are not duplicated here.

3.80 DOUBLE LINKED LIST

How to Write a Structure to Model a Double Linked List of Integers

```
typedef struct node
{
      int data;
      struct node *next;
      struct node *prev;
}node;
```

3.81 HOW TO ADD A NUMBER AT THE END OF A DOUBLE LINKED LIST

```
node* push_back(node *last,int info)
{
      //If this is the first node of the double linked list.
      if(last==NULL)
      {
            last = (node *)malloc(sizeof(node));
            last->data = info;
            last->next = NULL;
            last->prev = NULL;
            return last;
      }
      else
      {
            node *p = (node *)malloc(sizeof(node));
            if(p)
            {
                  last->next = p;
                  p->data = info;
                  p->next = NULL;
                  //The previous last node now becomes the
                  //last but one node of the list
                  p->prev = last;
            }
            return p;
      }
}
```

3.82 HOW TO ADD A NUMBER AT THE FRONT OF A DOUBLE LINKED LIST

```
node* push_front(node *h,int info)
{
      node *p = (node *)malloc(sizeof(node));
      p->prev = NULL;
      p->next = h;
      p->data = info;
      return p;
}
```

3.83 HOW TO GO TO THE NEXT NODE OF A DOUBLE LINKED LIST

```
node* Next(node *ANode)
{
      return ANode->next;
}
```

3.84 HOW TO GO TO THE PREVIOUS NODE OF A DOUBLE LINKED LIST

```
node* Prev(node *ANode)
{
      return ANode->prev;
}
```

For the above two methods please note that the return value will be null in case the given node don't have a previous or next node.

3.85 HOW TO DISPLAY THE DOUBLE LINKED LIST IN FORWARD DIRECTION

```
//Display the list of numbers
void display_forward(node *head)
{
      node *p = head;
      for(;p!=NULL;p=p->next)
            printf("Value = %d Address %u Next Address %u\n",
                  p->data,p,p->next);
}
```

3.86 HOW TO DISPLAY THE DOUBLE LINKED LIST IN BACKWARD DIRECTION

```
void display_backward(node *tail)
{
      node *p = tail;
      for(;p!=NULL;p=p->prev)
            printf("Value = %d Address %u Next Address %u\n",
                  p->data,p,p->prev);
}
```

3.87 HOW TO INSERT A VALUE AT A LOCATION IN THE LINKED LIST

```
node* insert_at(node *head,int index,int value)
{
      node *n = head;
      int i = 0;
      for(;head!=NULL;head=head->next)
      {
            i++;
            if(i==index)
            {
                  node *p = (node *)malloc(sizeof(node));
                  p->next = head->next;
                  head->next->prev = p;
                  p->prev = head;
```

```
                p->data = value;
                head->next  = p;
                break;
            }
        }
        return n;
}
```

Rest all functions will be same as those of Single Linked List. Moreover, if you notice carefully, you will discover that push_back() and push_front() functions, serve as the building block of other insertions functions like insert(), insertOneinAnother(), etc.

3.88 HOW TO ADD A NUMBER AT THE END OF A CIRCULAR DOUBLY LINKED LIST

```
node* push_back(node *last,int info)
{
      //If this is the first node of the double linked list.
      if(last==NULL)
      {
            last = (node *)malloc(sizeof(node));
            last->data = info;
            last->next = NULL;
            last->prev = NULL;
            return last;
      }
      else
      {
            node *p = (node *)malloc(sizeof(node));
            if(p)
            {
                  last->next = p;
                  p->data = info;
                  p->next = FirstNode;
                  //The next pointer of the last node
                  //now points to the first node.
                  p->prev = last;
            }
            return p;
      }
}
```

Please note that the Firstnode is nothing but a node pointer that is updated as and when we add a node in front of the list.

3.89 LINKED LIST APPLICATIONS IN BIOCHEMISTRY

Representations of the DNA Strand using Linked List

DNA Deoxyribonucleic Acid

- Blueprint of life (has the instructions for making an organism)
- Established by *James Watson* and *Francis Crick*

- Codes for your genes
- Shape of a double helix
- Made of repeating subunits called *nucleotides*

Gene Gene a segment of DNA that codes for a protein, which in turn codes for a trait (skin tone, eye color..etc), a gene is a stretch of DNA.

Nucleotide Nucleotide consists of a sugar, phosphate and a base.
Nucleotides (also called *Bases*)
Adenine, *Thymine*, *Guanine*, *Cytosine* or *A*, *T*, *G*, *C*
Nucleotides pair in a specific way—called the *Base-Pair Rule*

Adenine Pairs to Thymine

Guainine Pairs to Cytosine

Memory Helper—think 'A T Granite City'—which is where you live
*The rungs of the ladder can occur in any order (as long as the base-pair rule is followed)

How the Code Works
For instance, a stretch of DNA could be AATGACCAT—which would code for a different gene than a stretch that read: GGGCCATAG.

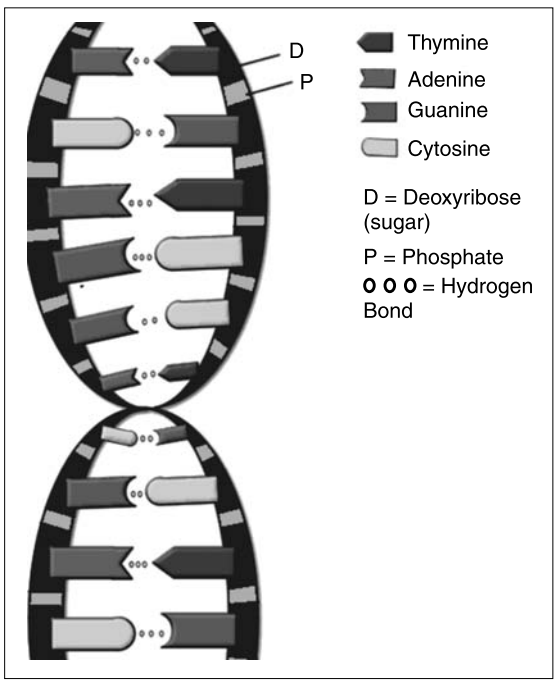


Fig. 3.19

Those 4 bases have endless combinations just like the letters of the alphabet can combine to make different words.

A DNA Strand (Single or Double) can be represented with a double linked list as shown below.

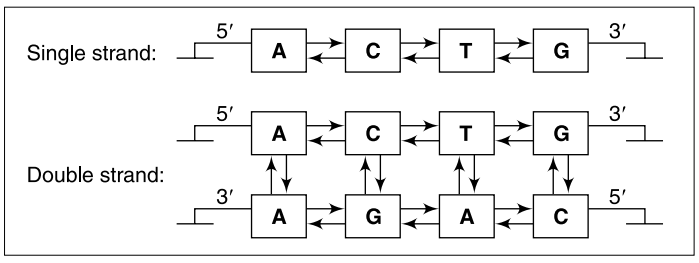


Fig. 3.20

We can create a Single Strand of a DNA using a double linked list. We can read a list of nucleotides from a notepad file and then create a strand as a linked list. The code snippet below shows how we can emulate a DNA using couple of Double Linked List.

```
enum {NO,YES};
enum {NOTFOUND,FOUND};

typedef struct node
{
```

```
      char nucleotide;
      struct node *prev;
      struct node *next;
}node;

node *head=NULL;

typedef struct DNA
{
      node *strand_1;
      node *strand_2;
}DNA;
```

The primitive bio-operations that can be performed on DNA molecules are:

- Shortening
- Cutting
- Hybridization
- Melting
- Linking or Ligation, and
- Multiplication.

Of these, the molecules that undergo hybridization, melting and ligation in parallel, thereby enhance the power of bio-computing.
All these primitive DNA operations can be easily emulated by using the DNA Structure.

3.90 HOW TO HYBRIDIZE TWO SINGLE DNA STRANDS TO ONE DNA

Before discussing the hybridization and other bio operations performed by the DNA, one needs to understand WC_Compatibility. Adenine only pairs with Thyamine, and Guanine only bonds with Cytosine. So Adenine or *A* is compatible to Thyamine or *T* on the other hand Guanine or *G* is compatible with Cytosine or C. This phenomenon is termed as *WC_Compatibility* in honour to the scientists *Watson* and *Creek* who discovered the DNA helix model.

Hybridization

Two single strands, which are WC-complement of each other, are joined laterally through *H*-bond to form a double strand. Our virtual model implements this reaction as a *C* function **hybridize (S1*, S2*)** which takes two DNA strands, i.e., two linked lists, as input and checks the WC-complementarity for each nucleotide. If all the nucleotide pairs satisfy complimentarily, it joins them by making the element of one DNA structure. Notice the code of the function below.

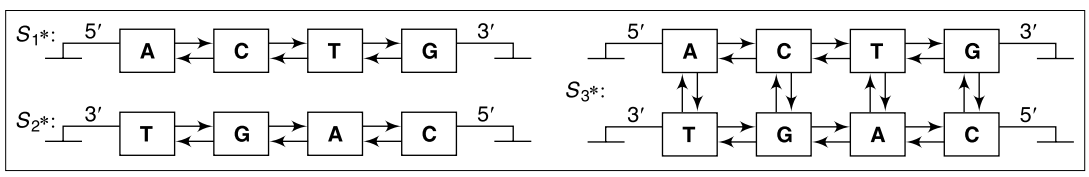


Fig. 3.21

The input couple of strands are A-C-T-G and T-G-A-C
And these two makes up the component strands for the DNA.

```
DNA Hybridize(node *S1,node *S2)
{
      int WC_Compatible  = NO;
      DNA tempDNA;
      node *temp1 = S1;
      node *temp2 = S2;
      for(;temp1!=NULL;temp1->next,temp2->next)
      {
            //Checking for WC_Compatibility
            if(((S1->nucleotide=='A' && S2->nucleotide =='T')
              ||(S2->nucleotide=='A' && S1->nucleotide =='T'))
              && (S1->nucleotide=='G' && S2->nucleotide =='C')
              ||(S2->nucleotide=='C' && S1->nucleotide =='G'))

                  WC_Compatible = YES;
            else
                  WC_Compatible = NO;
                  break;
      }
      if(WC_Compatible==YES)
      {
            tempDNA.strand_1 = S1;
            tempDNA.strand_2 = S2;
            return tempDNA;
      }

      else
      {
            tempDNA.strand_1 = NULL;
            tempDNA.strand_2 = NULL;
            return tempDNA;
      }
}
```

3.91 HOW TO MELT ONE DNA TO A COUPLE OF STRANDS

Melting

The weak H-bonds between complementary bases are broken by heating a double strand, yielding two single strands. This reaction is emulated by the *C* function **melt (S*)** which takes a double strand, i.e. a pointer to a laterally linked list pair, as input, and removes lateral links (the H-Bonds) between the corresponding nodes of the two lists.

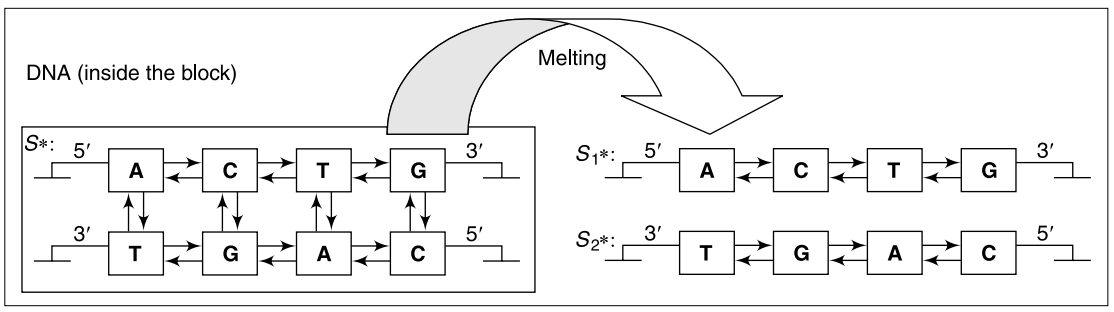


Fig. 3.22

Here is the function that returns the DNA strands after it melts.

```
node** Melt(DNA d)
{
      node* temp[2];
      temp[0] = d.strand_1;
      temp[1] = d.strand_2;
      return temp;
}
```

3.92 HOW TO EMULATE LINKING OF ONE DNA STRAND TO THE OTHER

This reaction involves head to head joining of single strands by phosphodiester bonds in the presence of a certain class of enzymes called *Ligases.* The emulation is done by **ligate (S1*, S2*)** which takes two single strands as input and joins them end to end as shown in the figure below.

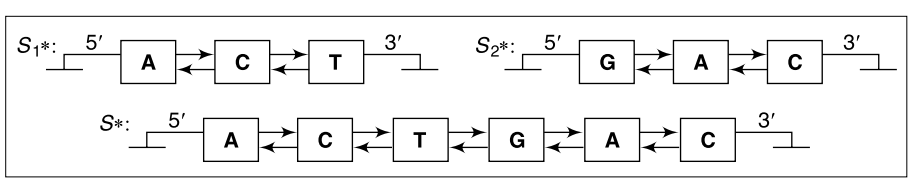


Fig. 3.23

Here is the function that ligates two single DNA strands.

```
node* Ligate(node *S1,node *S2)
{
      return merge(S1,S2,1,count(S1),1,count(S2));
}
```

See how the merge function is being used in the *ligate code*.

3.93 HOW TO REPRESENT A SPARSE MATRIX USING JAGGED ARRAYS

A sparse matrix can be space efficiently using Jagged Arrays. A Matrix is said to be Sparse if the total number of non-zero elements in the matrix is not more than $1/3^{rd}$ of the total number of elements. In the chapter on Arrays, we have seen how Sparse Matrices can be represented using 2D numeric arrays. But that has got a disadvantage with it. Lot of memory space is wasted due to this storage technique.

The jagged array can represent the sparse matrices in a space efficient manner. Here is a picture of the concept.

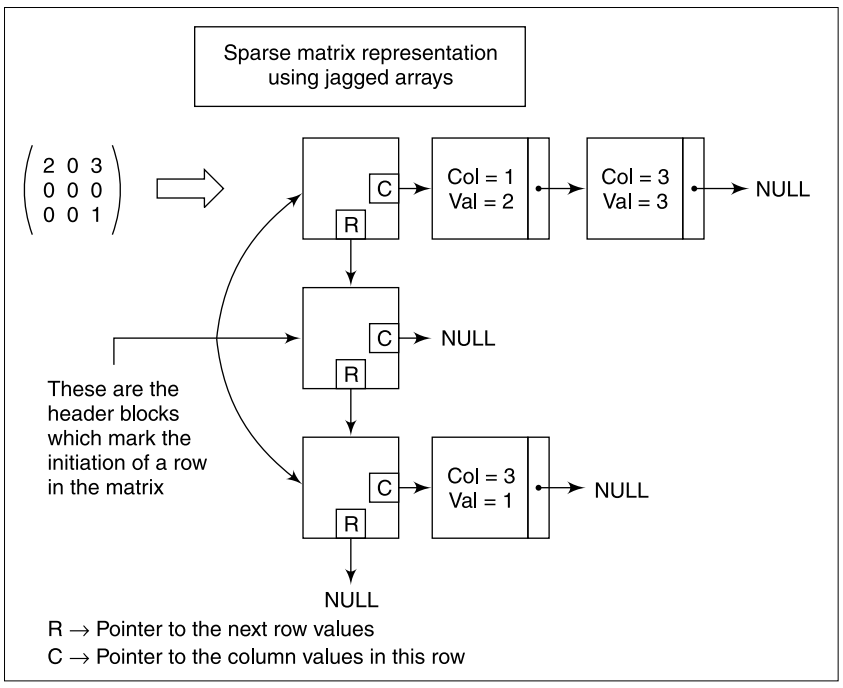


Fig. 3.24

Here are two structures that are designed to represent the above representation of sparse matrix using a jagged array of linked lists.

```
typedef struct Node
{
      int value;
      int col;
      Node *Next;//Horizontal Next
}Node;

typedef struct JaggedArray
{
      Node *Data;//Pointer to the column of values
      struct JaggedArray *Next;//Vertical Next
}JaggedArray;
```

3.94 HOW TO ADD AN ITEM IN THE SPARSE MATRIX

```
//The following function adds an item represented by column and value
//at the end of a list. This linked list represents a particular row of
//the sparse matrix.
```

```
Node* push_back_node(Node *last,int col,int val)
{
      if(last==NULL)//If this is the first Node
      {
            //Create Memory Space for this new Node
            last = (Node *)malloc(sizeof(Node));
            last->col = col;
            last->value = val;
            //Terminate the Node properly using NULL
            last->next = NULL;
            //return the back of the list
            return last;
      }
      //If it is not the first Node
      else
      {
            //Creation of new Node
            Node *p = (Node *)malloc(sizeof(Node));
            if(p)//Check if memory is available for this Node
            {
                  //The last Node from now on will be p
                  last->next = p;
                  p->col = col;
                  p->value = val;
                  //The list will be terminated by NULL
                  p->next = NULL;
            }
            return p;//Returning the end for the linked list
      }
}
```

3.95 HOW TO ADD A JAGGED ROW TO THE JAGGED REPRESENTATION OF THE SPARSE MATRIX

```
JaggedArray* push_back_jaggedrow(JaggedArray *last,Node *info)
{
      if(last==NULL)//If this is the first node
      {
     //Create Memory Space for this new node
            last = (JaggedArray *)malloc(sizeof(JaggedArray));
            last->data = info;
            //Terminate the node properly using NULL
            last->next = NULL;
            //return the back of the list
            return last;
      }
      //If it is not the first node
      else
      {
            //Creation of new node
            JaggedArray *p = (JaggedArray *)malloc(sizeof(JaggedArray));
            if(p)//Check if memory is available for this node.
```

```
{
      //The last node from now on will be p
      last->next = p;
      p->data = info;
      //The list will be terminated by NULL
      p->next = NULL;
}
return p;//Returning the end for the linked list
      }
}
```

3.96 HOW TO ACCEPT THE SPARSE MATRIX DETAILS FROM THE USER AND CREATE THE JAGGED ARRAY OF LINKED LISTS TO REPRESENT THE MATRIX IN A SPACE-EFFICIENT WAY

0	0	0	1	0
0	0	0	0	0
0	2	0	0	0
0	0	0	0	0
0	0	0	3	5

In the program below we will create the above sparse matrix using the structures jagged array.

```
int main()
{
      //Storing the following sparse matrix in the Jagged Array of linked list.
      //0 0 0 1 0
      //0 0 0 0 0
      //0 2 0 0 0
      //0 0 0 0 0
      //0 0 0 3 5
      Node *n = NULL,*cn = NULL;
      JaggedArray *head = NULL,*chead = NULL;
      n = push_back_node(n,4,1);
      head = push_back_jaggedrow(head,n);
      chead = head;
      n = NULL;
      head = push_back_jaggedrow(head,n);
      n = push_back_node(n,2,2);
      head = push_back_jaggedrow(head,n);
      n = NULL;
      head = push_back_jaggedrow(head,n);
      n = push_back_node(n,4,3);
      cn = n;
      n = push_back_node(n,5,5);
      head = push_back_jaggedrow(head,cn);

      for(;chead!=NULL;chead = chead->next)
            for(;chead->data!=NULL;chead->data=chead->data->next)
                  printf("Col = %d Val = %d\n",chead->data->col,chead-
>data->value);

      getch();
      return 0;
}
```

Here is a screenshot of the sample run of the program.

```
Co1 = 4 Ua1 = 1
Co1 = 2 Ua1 = 2
Co1 = 4 Ua1 = 3
Co1 = 5 Ua1 = 5
```

So we have retained all the intelligent information to reconstruct the sparse matrix using less memory than that of array representation.

3.97 REPRESENTATION OF HANDWRITTEN SIGNATURES USING JAGGED ARRAYS

Signature on white paper with black ink is nothing but a sparse matrix of black dots (*specifically Pixels*). The above image shows the signature of *Neil Armstrong*. This is an image of dimension 311 * 350 (*The image above is scaled down due to proper view*), that means in total there are 108850 pixels of which only 633 elements are black dots and rest all are white. So the intelligent information of the above image is residing in only 633/108850 part of the total size of the image. That means the image is highly sparse and can be represented as a *sparse matrix*. Now we have already shown that a sparse matrix can be efficiently stored as a jagged array, so the signatures can also be stored as jagged arrays.

Fig. 3.25

3.98 HOW TO MODEL *SIMPLE CONTENT MANAGEMENT* SYSTEM USING LINKED LIST

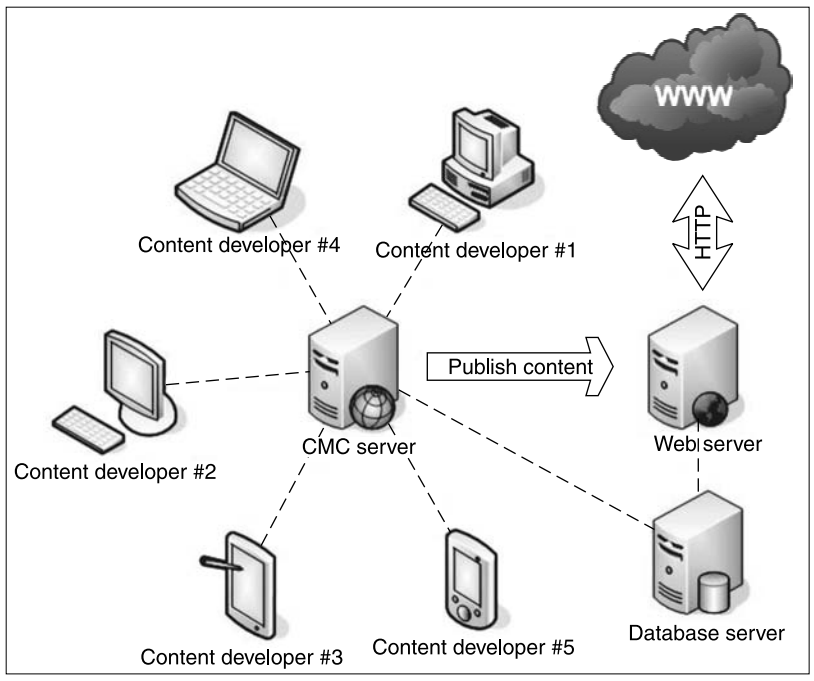


Fig. 3.26

In a content management system there are three types of people involved. The readers, authors/contributors and the moderators. The readers will only have the read access to the documents on the other hand the authors/moderators and the contributors can edit the documents. These are some of the possible operations on the documents in the content management software.

- Add a new document to a group
- Move a document from one group to the other
- Change the publication status of a document
- Add a reader/user
- Add a user to the Moderator Group
- Remove a user from the Moderator Group
- Add a user to the contributor group
- Remove a user from the contributor group

We can create a linked list of the document objects that will hold pointers to their contributors and users. There will be a separate list of users for each group. The sketch given describes the CMS system.

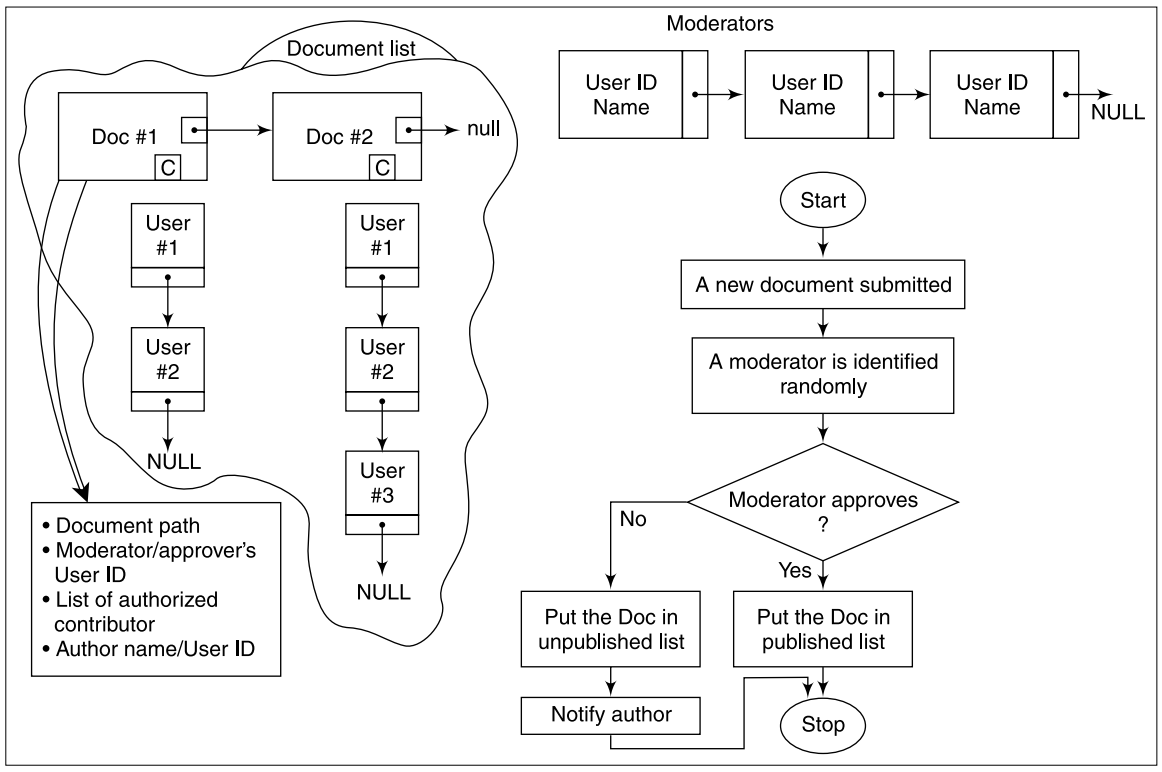


Fig. 3.27

Here is a structure that represents users of the CMS:

```
enum TypeOfUser{USER,MODERATOR};

typedef struct User
{
```

```
      int UserType;//What can this user do?
      char *UserID;//User's ID
      char *Name;
      int NumberOfPosts;//How many posts have this user done?
}User;
```

Here is a structure that represents documents of the CMS:

```
typedef struct Document
{
      int DocumentID;//What is the access ID?
      int PublishedStatus;//What is the publication status?
      User  Author;//Who wrote it?
      User  Moderator;//Who approved it?
      struct User *Contributors;//Who can modify it?
}Document;
```

The entities of this program are users and the documents. Their details will be stored in the delimited files as shown.

The following function is written to load all the existing users in the file. Don't feel worried if you don't understand the function well. This function uses a function called **`Split().`** You can jump to the chapter on string to read about it.

```
User* LoadUsers()
{
      char line[81];
      int count = 0;
      String *toks = NULL;
      User *us = NULL,*users = NULL;
      User u;
      FILE *fp = fopen("C:\\users.txt","r");
      while(!feof(fp))
      {
            fgets(line,81,fp);
            toks = split(line,"-");
            if(toks->next!=NULL)
            {
                  strcpy(u.UserID,toks->s);
                  toks=toks->next;
                  strcpy(u.Password,toks->s);
                  toks=toks->next;
                  strcpy(u.Name,toks->s);
                  toks = toks->next;
                  u.UserType = atoi(toks->s);
                  us = push_back_User(us,u);
                  count++;
                  if(count==1)
                        users = us;
            }
      }
```

```
	fclose(fp);

	return users;
}
```

The program on CMS should be able to perform at least the following..

```
int menu()
{
	int choice  = 0;
	puts("1.Add a new user");
	puts("2.Post a new article");
	puts("3.Publish a document");
	puts("4.Review a document");
	puts("5.Make an User the moderator");
	puts("6.List all published document");
	puts("7.List all unpublished document");
	puts("8.List all published document from an author");
	puts("9.Exit System");
	scanf("%d",&choice);
	return choice;
}
```

Here are two functions that puts one document at the end of document list.

```
Document* push_back_Document(Document *last,Document d)
{
	if(last==NULL)
	{
		last = (Document *)malloc(sizeof(Document));
		last->next = NULL;
		last->prev = NULL;
		last->Author = d.Author ;
		last->Contributors = d.Contributors;
		last->DocumentID = d.DocumentID;
		strcpy(last->DocumentURL , d.DocumentURL);
		last->Moderator = d.Moderator;
		return last;
	}
	else
	{
		Document *p = (Document *)malloc(sizeof(Document));
		last->next = p;
		p->prev = last;
		p->next = NULL;
		p->Author = d.Author ;
		p->Contributors = d.Contributors;
		p->DocumentID = d.DocumentID;
		strcpy(p->DocumentURL , d.DocumentURL);
		p->Moderator = d.Moderator;
		return p;

	}
}
```

When a document will be added, by default the publication status will be unpublished.

***Try yourself**: Finish this simulation using these structures and functions and the flow chart above and use your creativity.*

3.99 HOW TO MODEL *WORKFLOW ENGINE* SYSTEM (*LIKE* K2.NET*)* USING LINKED LIST

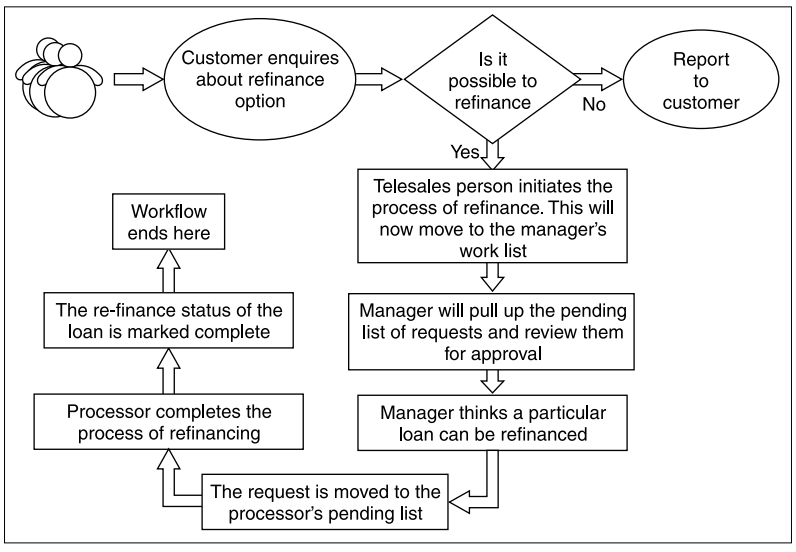


Fig. 3.28

Here is a k2.net workflow image.

Workflow engines are systems that allow its users to create a flow of work for a particular activity. Suppose in a bank a customer applies for the home loan, then the telesales stuff of the bank will gather the information from the customer and will initiate the process. Once initiated the refinance loan request will move to the pending list of the manager. Once the manager thinks everything is ok, then he marks the refinance request as approved and then the request will be moved to the processor's pending list. Once the processor does all the required operations, then the request is marked as *complete*.

Basically this workflow grows around a list of requests.

Try to create a workflow engine using linked lists.

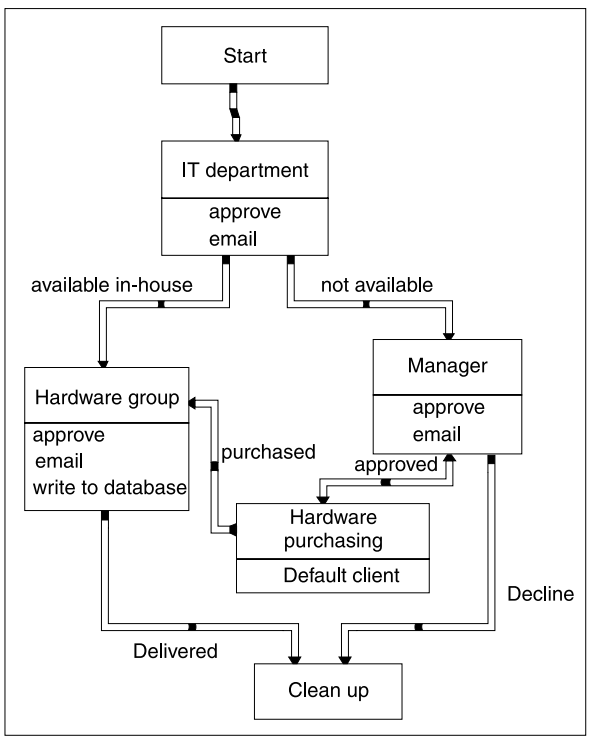


Fig. 3.29

3.100 A COMPARISON BETWEEN ARRAYS AND LINKED LISTS

When someone asks 'What is the better data structure? array or linked list?' the reverse question should be 'What you want to do most with your data structure?'

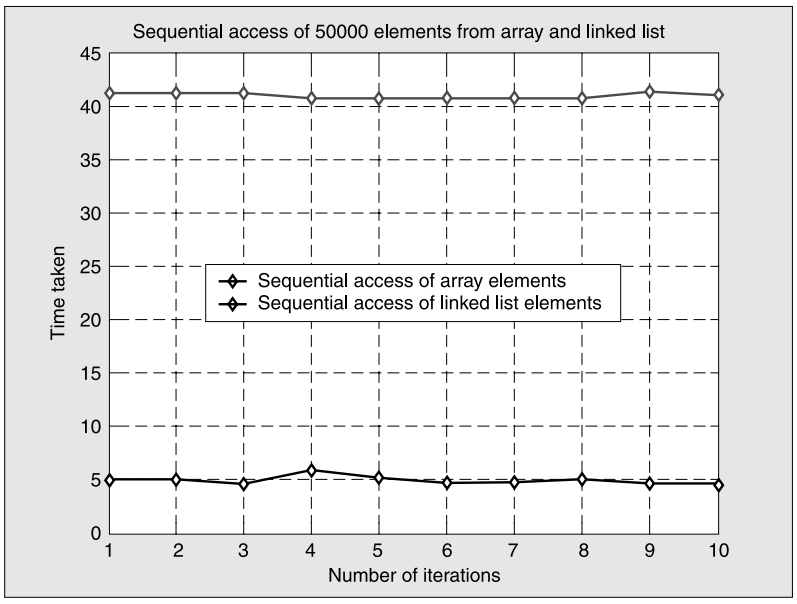
Select an array If

- The maximum number of elements are known at the start of the program.
- The elements will be accessed in a random fashion mostly.
- There could be occasions to access the elements sequentially.
- There would be ideally no insertion or deletion inside the length of the array.
- All the additions and deletions will be at the end of the array.

Select a linked list if

- Random insertion/deletion of the elements occur most of the times
- Accessing array elements (Either Sequential or Random) will be very rare

From the above graphs we can conclude that for a linked list the sequential access is almost 8 times slower than array and for random access array access is close to 5 times faster than linked list access. The following curves are generated using data from 10 sample runs.



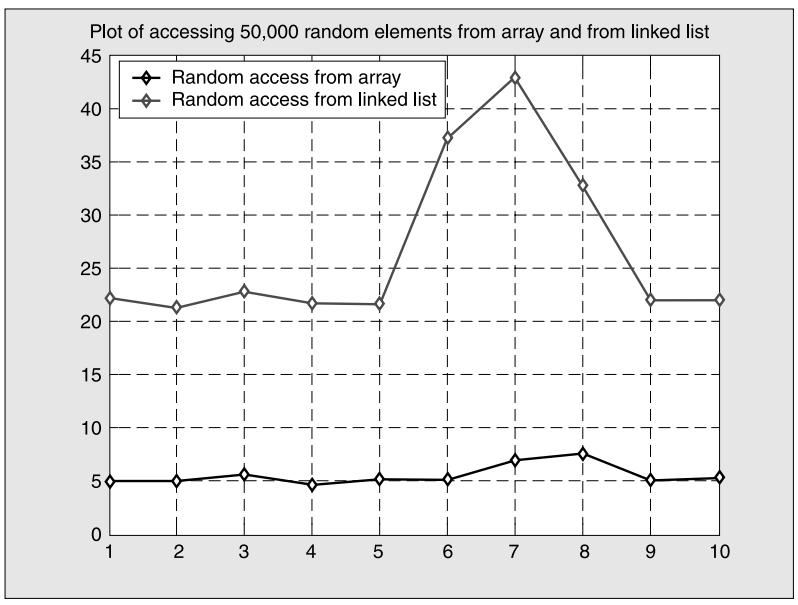


Fig. 3.30

Follows the same scale and axis as the previous one

REVISION OF CONCEPTS

1. Linked list was first discovered by *Allen Newell, Cliff Shaw and Herbert Simon* in 1955 at RAND corporation for their Information Processing Language (IPL) which was used for many early Artificial Intelligence (**AI**) programs.
2. Linked list is the simplest pointer based data structure and it serves as the base for several other pointer based complex data structures like trees, graphs, etc.
3. It is better than array as far as memory management is considered, because memory allocations need not be contiguous. But accessing elements from a linked list is slower than that from an array because in the later memory is contiguous.
4. Linked list access is slower than that of the array access. Specifically in such occasions where the elements of the list need to be processed in a random fashion and we know previously how many elements need to be processed array is a better choice over linked list. When we know that we will use the data in the list as we go down or as we go up then linked list is the best data structure available.
5. A node of a single linked list holds one pointer to the next node.
6. Normally *A* node of a double linked list holds two pointers. One to the next node and the other to the previous node. You can design a double linked list that holds only a single pointer and then using some pointer arithmetic you can find the address of the previous element. But this approach kills the conceptual look and feel of your code and should be strictly avoided until there is a special need.
7. A node in circular linked list holds one pointer to the next node. But here the next pointer of the last node points to the first node. Thus the list is circular.
8. Unlike array, linked list can grow and diminish in run time. So, whenever the nature of a list is dynamic, linked list is a better choice over array.
9. Whenever there is a list of items to be processed at run time then the linked list is the best data structure to store the data.
10. The data field in the linked list can be anything, a primitive data type or a user defined data. In most of the cases, it will be the user defined data (*Efficiently represented as C structure*).
11. The data field in the linked list can be another linked list. This allows linked list to become a very useful data structure. The functional programming language LISP uses the linked list data structure to build its compiler and linked list is the inherent data structure in this language.

REVIEW QUESTIONS

1. What is wrong about the following statement node* r = push_back(r,11); given that push_back accepts an object of type node to be pushed at the end of the list.
2. Write a single line statement to find the maximum elements of a merged linked list.
3. Write statements to add the 11,14,10,13,12 in an ascending order to a linked list. Just write the client code that calls functions described in this chapter.
4. Write statements to insert two repetitions of the sequence 45,90 after 13 in the sequence 11,22,13,34,56.
5. Write a statement to delete the n^{th} element of the above linked list.

PROGRAMMING PROBLEMS

1. Write a function to delete all occurrences of an item from a linked list if a condition is true. The condition should be passed to the function as a predicate.
2. Write a function to copy the items of one linked list to another if the condition is satisfied. The condition should be passed to the function as a predicate.
3. Write a program to calculate the mean of a linked list of double variables.
4. Write a program to calculate the median of a linked list of double variables.
5. Write a program to calculate the mode of a linked list of double variables.
6. Write a program to calculate the standard deviation of a linked list of double variables.
7. Write a program to calculate the variance of a linked list of double variables.
8. Write a function that accepts a jagged linked list of linked lists of doubles and returns the mean, median, mode, standard deviation, variance of these lists.
9. Write a program that accepts a linked list of point structure (for details on point structure refer Chapter on Structure) and determine whether these points can be vertices of a polygon or not.
10. Write a function to swap alternate nodes of a linked list.
11. Write a function to copy the contents of a linked list to another if the condition is satisfied.
12. Write a function push_back_if() that pushes an element at the end of the linked list if that element satisfies a particular condition. Basically this is a wrapper method that will check the validity of the element before addition and then will use push_back.
13. Write a function get_locations_if() that returns the location of the elements on the linked list, if the condition is satisfied.
14. Write a function find_first_if() will return the first element that matches the condition given.
15. Write a function find_last_if() will return the last element that matches the condition given.
16. Write a function find_all_where() that will return all the elements of the list that matches the criterion supplied by the predicate.
17. Write a function collect(int power) that will accept two polynomials and then will collect the terms of both the polynomials whose power is supplied as the function parameter.
18. Write a function arethesefactors() that will accept a linked list of polynomials and a single polynomial. If the product of the elements of the linked list of polynomials is same as the stand alone supplied polynomial then those polynomials are factors of the solitary polynomial and the function will return 1 else it will return 0.
19. Write a function samepoly() that checks whether two argument polynomials are same or not.
20. Write a function monicpoly () that will return 1 if the polynomial supplied as argument is monic. A polynomial is said to be monic if the coefficient of the highest degree term of the polynomial is one.
21. Write a function subtractpoly() that will return the result of the subtraction between two polynomials.
22. Two digital signals are given. Write a function for the resonance signal.

4

Strings
Database to DNA!

INTRODUCTION

In this chapter we will learn how to represent a string in C. In computer science, we deal regularly with strings everywhere, starting from details of customer in a service industry to DNA strands in the biochemistry lab. C doesn't offer string as a basic built in data structure. We use char array or char pointer to simulate string behavior. This type of representation is commonly known as *C-Style String Representation*. Once we learn basic string initialization, then we will learn about the C-library functions which will be the building blocks for some user defined functions that we build towards the later of this chapter. Moreover linked lists are also used to represent strings and store them. Most contemporary object oriented programming languages, like Java and C#, support splitting a given string by delimiters. We will create such a function in this chapter. This function will take a string and a delimiter to chop it and will return a linked list of chopped strings. Later in the book, especially in the chapter on file handling we will use this function for reading delimited notepad files and process them.

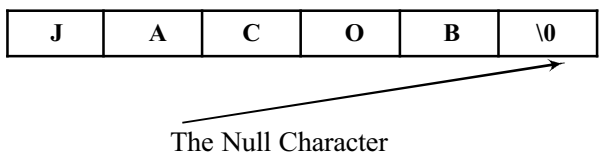
4.1 SOME KEY FACTS ABOUT STRINGS IN C

- C- Strings are nothing but NULL ('\0') terminated character array.
- char *string; is a pointer to a C-style string.
- char string[20] is a C-Style string, that can hold 20 characters
- %s is used to scan string from console using scanf()..
- There is no need to give '&' sign while scanning a string from the console, because in C, strings are nothing but null terminated character array and the array name is the pointer to itself.
- For using built-in C-Style string manipulation functions you need to include `string.h` in the program.
- Strings are used to represent a variety of data, starting from name of a person in a database table to a DNA strand of an individual.
- Fast String matching algorithms like Boyer Moore and KMP are used for matching DNA strands, protein patterns, cancer cell repetition pattern, etc.

- String processing also finds application to find "Plagiarism". Mostly "Rabin-Karp" algorithm is used to detect it.
- String comparison algorithms/methods are used in different areas like "Spell Checking", "DNA matching", "Adaptive Text Prediction using T9 dictionary for mobile phones", etc.

4.2 C-STYLE STRING

A *C*-Style String is a null('\0') terminated character array. Here is a picture of a *C*-Style String



If the character array named "Name" stores this *C*-Style String then

```
Name[0] = 'J';
Name[1] = 'A';
Name[2] = 'C';
Name[3] = 'O';
Name[4] = 'B';
Name[5] = '\0';
```

These *C*-Style strings can be initialized in so many ways as discussed later.

String Initialization

C-Style Strings can be initialized in many ways. From now on *C*-Style strings will be referred by just "string".

4.3 HOW TO INITIALIZE AT THE TIME OF DECLARATION

While declaring the string, we can initialize it. Here are few examples.

```
//Not Calculative. Codes is the pointer to the string
//We can store any length string.
char *codes="wxy8#455j%32";


//Calculative Initialization
//The name below has exactly 15 characters
//and thus it has been stored in
//a character array of 16 pockets. The
//last pocket of the array is for storing the null character.
char name[16]="Jacob Alexander";
```

4.4 HOW TO INITIALIZE STRINGS USING USER-DEFINED VALUES

Most of the cases you will be interested to initialize the string using user defined/supplied values. We can use scanf( ) or gets( ) method for scanning the, user string inputs, but there is a difference. In case there is a blank space in the input then if we try to scan it using scanf( ) it will scan only till that part when the scanf( ) encountered a space. On the other hand if we use gets( ) we can scan strings with white spaces in between them. Here is one example:

```
char *name;
printf("Enter your name :");
fflush(stdin);
gets(name);
```

If you enter 'Brian Kernighan' while asked for the name, the name array will store it like

name

B	R	i	a	n	<space>	K	e	r	n	i	n	g	h	a	n	\0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

<space> denotes a single space character

The integers denote the locations of the characters in the name. *Later it will be shown how to extract a part of a given string. Then we can use that function to extract the first, middle and the last name.*

4.5 HOW TO INITIALIZE ONE STRING WITH ANOTHER STRING

One string can be initialized with another string *or part of it (substring)*. Here is an example.

```
char *s="STEPANOV";
char *copy_of_a=a;
```

Here copy_of_a is a string and contents the value of *a*. This is done when the string *a* will be manipulated and at the end we again need to use the initial value of the string *a*. In such cases value of *a* will be copied in some other character arrays.

4.6 HOW TO INITIALIZE A STRING USING CHARACTER VALUES

We can use characters separately to fill *a* string. Here is one example.

```
char a[5]={'A','B','C','D','E'};
```

This above declaration is same as

```
//Easy and looks Professional.
char *a="ABCDE";
```

4.7 HOW TO INITIALIZE A STRING USING ASCII VALUES

In some real life applications, mostly in cryptography, we get ASCII values of characters instead of the characters. Fortunately, C allows us to assign ASCII Values to assign a C String. The ASCII Value for '\0' (Null character that marks the end of a C-Style String) is zero. So the end value should be 0. Here is an example.

```
char a[6]={65,66,67,68,69,0};
puts(a);
```

The output of this snippet will be
ABCDE

Initializing a string with ASCII values is the only possible solution when we need **to put some unavailable (in the keyboard) characters in the array**. For example think of a situation where we are programming for a biological institute and we need to print human genome symbols for both genders Here are pictures of the symbols:

♂ The Male Genome Symbol

♀ The Female Genome Symbol

The following program prints these patterns.

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

int main()
{
      char s[2]={12,11};
      printf("The Female Symbol :%c\n",s[0]);
      printf("The Male Symbol   :%c\n",s[1]);
      return 0;
}
```

Here is the output of the program,

The Female Symbol: ♀

The Male Symbol: ♂

This scheme for initializing strings are highly used in cryptography and user interface design for console based C programs. (You will find these applications in departmental stores).

4.8 INTRODUCTION TO SOME BUILT-IN TURBO C STRING LIBRARY FUNCTIONS

The functions are grouped and discussed according to the *jobs they perform*.

Finding the Length of the String

Functions used:

strlen()

To find the length of the string we can use library method strlen(). The following code snippet explains how to use strlen().

```
char *name="Sir Issac Newton";
printf("Name is of length :%d",strlen(name));
```

S	i	r		I	s	s	a	c		N	e	w	t	o	n	\0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

strlen() returns length of the string including the null character. So the length of the string returned is 16.

Concatenating Two C-Style Strings

Functions used:

- `strcat()`
- `strncat()`

strcat()

To concatenate two strings entirely (One is concatenate at the end of the other) we can use strcat () as shown below.

```
char *firstname = "John";
//Note the carefully left blank space before the family name.
//If this blank space is not left then, given name and family name will
//be close and not readable

char *familyname="  Gottamann";
printf("Full Name :%s",strcat(firstname,familyname));
```

The output of this program will be John Gottamann

strncat()

This function is used when we need to concatenate up to some predefined number of characters of one string at the end of the other. Here is an example.

```
char *firstname = "John";
char *familyname="  Gottamann";
printf("Full Name :%s",stnrcat(firstname,familyname,5));
```

This code will output John Got.

Comparing Two Strings

Functions used:

- strcmp()
- strcmpi() [A macro]
- stricmp() [Same as strcmpi() but it is a function]
- strncmp()
- strncmpi()

strcmp()

Two strings can be compared using the built-in library function strcmp(). This method compares two strings considering case. That means if we compare the strings "World" and "wOrLd" then its comparison will tell us that the strings are not equal. In case the two passed arguments are equal then this function returns zero. Otherwise this method will return a non zero value. Here is an example.

```
char *a="world";
char *b="worm";
if(strcmp(a,b)==0)
      printf("Two Strings are same ");
else
      printf("They are unequal");
```

The output of this program snippet will be "They are unequal" as the two strings are unequal.

Besides using this built-in function you can write your own string comparison method. The algorithm is to check letter by letter and then proceed until you encounter the null character or a mismatch.

There are two other methods that are used to compare strings. They are

strcmpi()

This functions compares two strings ignoring their cases. This function also returns 0 in case the two string values are same. Two Strings "World" and "wOrLd" are same to strcmpi(). So the output of the code snippet

```
char *s="World";
char *t="wOrLd";
if(strcmpi(s,t)==0)
      printf("The two strings are same");
else
      printf("The strings are different");
```

strncmp()

It accepts three arguments, two strings and one integer. It checks two strings up to that n^{th} character. In case they are same, then zero is returned. Otherwise, non zero value is returned. This function becomes particularly handy, when we search for names in address-book programs. Suppose you want to find out that how many names are there in the address-book that starts with "Ja", then we can do something like

```
//Start searching
//Till the end of the address book
//pick the name of the person in a string called "Name" then
if(strncmp(Name,"Ja",strlen("Ja"))==0)
 //Show the name
```

`strlen("Ja")` = 2. So the above piece of code will return success for all those entries in the address book entries for which, the first 2 letters of the name are "Ja", like "Jacob", "Jack", "Jasmin", "Jana", etc.
But it will return false for "Jonathon", because here only the first character matches.

strncmpi()

Same as strcmp(). The only difference is that this function is case-insenstive. This function compares two strings up-to n^{th} character ignoring their case. So in the above example (of address-book) the best suited method is strncmpi() because that will match names starting with "Ja", "jA", "JA" and "ja"…
This function also returns 0 on success and a non-zero value on failure.

Copying a String to Another

Functions used:

- `stpcpy()`
- `strcpy()`
- `strncpy()`

stpcpy()

This function copies all the characters of one string to other empty string till it reaches the null character. So this function is appropriate when you are trying to add some more characters at the logical end of the string being copied.

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

int main()
{
      char *a="FRANKFURT";
      char *b="";
      stpcpy(b,a);
      puts(b);
      getch();
      return 0;
}
```

strcpy()

This function copies one string to another blank string. This function is mainly used for assigning the new string variables.

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

int main()
{
```

```
char *a="FRANKFURT";
char *b="Hamburg";
//We will swap the contents of the strings using strcpy
char *temp="";
clrscr();
printf("Before Swapping a = %s and b = %s\n",a,b);
strcpy(temp,a);
strcpy(a,"");
strcpy(a,b);
strcpy(b,"");
strcpy(b,temp);
printf("After Swapping a = %s and b = %s\n",a,b);
getch();
return 0;
}
```

The output of the program will be

```
Before Swapping a = FRANKFURT  and b = Hamburg
After Swapping a = Hamburg and b = FRANKFURT
```

strncpy()

This function is similar to that of strcpy(), but it will copy up to the n^{th} character of the source string.

Suppose we have the complete name for a person and we want to extract the first name. To do that, we first need to find out the index of the first white space in the string. Then we need to copy part of the string from starting to the index one less than the index of the first white space.

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

int main()
{
      char *name="Jacob Alexander";
      char *fname="";
      int i;
      clrscr();
      //Finding the location of the first whitespace.
      for(i=0;name[i]!=' ';i++);
      printf("%d ",i);
      strncpy(fname,name,i);
      //Putting the null character manually.
      fname[strlen(fname)]='\0';
      puts(fname);
      getch();
      return 0;
}
```

The output of this code will be
Jacob.

Changing Case of a String to Lower or to UPPER

Functions used

- `strlwr()`
- `strupr()`

strlwr()

This function returns a string which is a lower cased version of the argument string.

strupr()

This function returns a string which is a upper cased version of the argument string.

4.9 DESIGNING UTILITY TOOLS USING THESE TWO FUNCTIONS

Correcting Wrong Case Program in C/C++ (Using strlwr( ))

When PASCAL programmer migrate to C/C++, he often goes wrong syntactically, because in PASCAL almost everything is written in CAPITAL letters while in C/C++ family, languages are written in small letters. But there are some alimony also. In C/C++ the built-in constants (like M_PI in <math.h>) are all written in capital letters.

Assuming that there is no built-in constants used in a particular C/C++ Program, we can write an application that can correct a wrong case C Program to correct case. Here is a typical input file.

//test.C

```
#INCLUDE <STDIO.H>
#INCLUDE <stdlib.h>
#include <CONIO.H>

int MAIN ()
{
      puts ("I am here");
      RETURN 0;
}
```

The program accepts the filename and then displays the corrected version of the program.

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

int main()
{

      char *line;
      FILE *fp;
      clrscr();
      //Open the input "wrong case" program file
      fp = fopen("D:\\test.C","r");
      if(fp)
      {
            while(!feof(fp))
            {
                  fgets(line,81,fp);
                  puts(strlwr(line));
            }
      }
      else
            perror("Error");
      getch();
      return 0;
}
```

Here is the output of the program.

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

int main ()
{
      puts ("i am here");
      return 0;
}
```

Try to extend this application that will take care of the following situations:

1. User defined constants like #define USD2INR 44.5 should not be changed to lower case.
2. User defined function name like draw_rectangle() should not be changed to lower case.
3. Inbuilt constant names (like M_PI_2) and macro names (like EOF) should not be changed to lower case.

4.10 A TOOL FOR CHANGING CASE OF FEW CHOSEN ABBREVIATIONS IN A FILE (USING STRUPR())

Abbreviations are normally written in CAPITAL letters, like WHO, UNICEF, etc. It is really painful to capitalize these words while editing a file. A small C program can be written that can do these changes with little or no effort. Here is the strategy. The program will first open the file where these abbreviations need to be changed and then read that file word – by – word. If any word exactly matches to any of these abbreviations supplied to the code then the program will change those words to uppercase and display it. We can redirect the corrected output to another text file.

Here is the program source.

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

int main()
{

      char *line;
      FILE *fp;
      clrscr();
      fp = fopen("D:\\who.txt","r");
      if(fp)
      {
            while(!feof(fp))
            {
                  fscanf(fp,"%s",line);
                  if(strcmpi(line,"who")==0 ||
                  strcmpi(line,"unicef")==0)
                        printf("%s ",strupr(line));
                  else
                        printf("%s ",line);
            }
      }
```

```
      else
            perror("Error");
      getch();
      return 0;
}
```

Who.txt contains

WHO, the World Health Organization, had identified 7th June 2006 as Global Polio Eradication Day. Who said that whosoever may tell you that these 2 drops are useless, you shouldn't listen to them. unicef and who requested the state and central government to take necessary measures to assure that all the kids up to the age of 5, get the polio vaccine as per who standard. unicef along with local ngos are trying their best to ensure that no child is left out from this gigantic who - unicef effort. I will take my children to the nearest polio booth. As a who doctor, I ask you to do so my friend! let's fight polio along with who, unicef and ngos.

and the output of the program will be

WHO the World Health Organization had identified 7th June 2006 as Global Polio Eradication Day. **WHO** said that whosoever may tell you that these 2 drops are useless, you shouldn't listen to them. **UNICEF** and **WHO** requested the state and central government to take necessary measures to assure that all the kids up to the age of 5, get the polio vaccine as per **WHO** standard. **UNICEF** along with local NGOs are trying their best to ensure that no child is left out from this gigantic **WHO - UNICEF** effort. I will take my children to the nearest polio booth. As a **WHO** doctor, I ask you to do so my friend! Let's fight polio along with **WHO, UNICEF** and NGOs.

See the changes are made bold

Try extending this program with the following changes.

1. Try to make the changes (i.e. capitalization) interactively. For example there may be a sentence like who doesn't know who? It actually means who doesn't know WHO ? Whenever an abbreviation from the pool of abbreviations is found, display the full line and then ask user whether to change that abbreviation or not. If yes then make the case change, else skip it and continue search for next matching abbreviation.
2. Create a pool of abbreviations
3. Allow users to add new abbreviations.

4.11 HOW TO REVERSE A STRING

Functions

- `strrev( )`

strrev ( )

This function returns a string which is reverse of the input string. In string processing reversing a string is very trivial and important operation.

```
#include <stdio.h>
#include <conio.h>
int main()
{
      char *s=”ABACAS”;
      printf(“Original String : %s\n”,s);
      printf(“Reverse String : %s\n”,strrev(s));
      return 0;
}
```

This will display
Original String: ABACAS
Reverse String: SACABA

See the Palindrome Example in the Array Chapter to know how to use this method.

4.12 HOW TO SET CHARACTERS OF A STRING WITH ANOTHER CHARACTER

Functions Used

- `strset ()`
- `strnset()`

strset()

This function is used to set all the characters of a string to another user given character. Here is a sample code to explain how the function behaves.

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

int main()
{
      char *s=”abcdefghijklmnop”;
      char symbol = ‘x’;
      printf(“The original string is : %s\n”,s);
      printf(“After setting with x :%s\n”,strset(s,symbol));
      return 0;
}
```

The output of the program will be
The original string is: abcdefghijklmnop
After setting with *x*: xxxxxxxxxxxxxxx

strnset()

Sometime you may be interested to set only a particular number of characters from the start of the string. In those situations, you have to use strnset( ). This function sets the first *n* characters of a string to another character as given by the user.

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

int main()
{
      char *s=”abcdefghijklmnop”;
      char symbol = ‘x’;
      printf(“The original string is  : %s\n”,s);
      printf(“After setting with x up to first 4 characters
                        :%s\n”,strnset(s,symbol,4));
      return 0;
}
```

The output of the program will be
The original string is: abcdefghijklmnop
After setting with *x* up to first 4 characters: xxxxefghijklmnop

These functions can be used in computer secrecy projects. For example, you may have noticed that when you buy something over internet using your credit card, the digits of the credit card are set to some different characters. Sometimes all the characters are set and sometime some predefined range of characters are set. These sorts of operations can be achieved using these two functions.

4.13 HOW TO FIND THE FIRST OCCURRENCE OF A CHARACTER OF A SUBSTRING WITHIN ANOTHER STRING

Functions used:

- strchr()
- strstr()

strchr()

This function finds for the first occurrence of the sought character. Finding the occurrence of a character in a string is very common and trivial operation in the string algorithms. Instead of writing a loop by yourself you can use this function.

strchr(string_to_search, char_to_search) returns a pointer to the location where the sought character is found for the first time. So by subtracting the base pointer to the string from the returned pointer we can get the index of the character's first occurrence.

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

int main()
{
      char *s="Life is beautiful";
      printf("The first occurrence of  e is at %d\n",
            strchr(s,'e')-s);
      return 0;
}
```

The output of the program will be
The first occurrence of e is at 3

strstr()

This function is used for finding out the first occurrence of a substring/word from a given string. This is syntactically the same as strchr( ). This function returns the substring that starts from the given word. For example if the given string is

char *s = "Life is beautiful";

And the search pattern/word is "is" then if we call strstr() as strstr(s,"is") then it will return
is beautiful

Using similar pointer arithmetic as in strchr() we can find the index location of the first occurrence of the sought word. Here is the code

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

int main()
```

```
{
      char *s=”Life is beautiful”;
      printf("The first occurrence of  \"is\" is at
                  %d\n",strstr(s,"is")-s);
      return 0;
}
```

The output of this code is

The first occurrence of “is” is at 5

4.14 HOW TO FIND THE LOCATION FROM WHERE TWO STRINGS START TO DIFFER

Functions used:

- `strspn()`
- `strcspn()`

strspn()

This function is used to find the location of a string where it started to differ from another given string. Here is a code snippet to demonstrate the function’s behaviour.

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

int main()
{
      char *s="12345678";
      char *t="1234ffsf";
      printf("The strings intersect at %d\n",strspn(s,t));
      return 0;
}
```

This program prints

The strings intersect at 4

The point where from the strings start differing is sometime referred as their intersection point.

strcspn()

This is just the complimentary function of strspn(). This function finds that position in a string till when it doesn’t match with another string. Here is the code

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

int main()
{
      char *s="12345678";
      char *t="ffsf5678";
      printf("The strings intersect at %d\n",strcspn(s,t));
      return 0;
}
```

Here is the output of the program:

The strings intersect at 4

4.15 HOW TO CREATE THE DUPLICATE OF A STRING IN A MEMORY-EFFICIENT MANNER

Function used: `strdup( )`

strdup()

This function creates a duplicate string of the passed argument. Very often we need to keep copies of a string before we process it further. At the end of processing we can use this duplicate copy for some operations. Here is a code to demonstrate its use.

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

int main()
{
      char *s=”Able was I as I saw Elba”; //A Palindromic String!
      char *dup_s = strdup(s);
      printf(“Duplicate String : %s\n”,dup_s);
      free(dup_s);
      return 0;
}
```

This will print
Duplicate String: Able was I as I saw Elba

4.16 HOW TO TOKENIZE A GIVEN STRING

Function used:

- `strtok ()`

strtok()

This function is great for tokenizing needs. Do you remember that in Array chapter we tried to design some clone to stringtokenizer of Java. The same type of application can be performed by this neat and portable function in C. Suppose we have a string like

12,400

And we want to extract part of the string that is leading “,” and the trailing part. Here is the code:

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

int main()
{
      char *s=”12,400”;
            printf(“Part before , is %s\n“,strtok(s,”,”));
      printf(“Part after , is %s\n”,strtok(NULL,”,”));
      return 0;
}
```

This will print
Part before , is 12
Part after , is 400

So you may have noticed that to get the trailing part (Whichever part of the string is after the delimiter) we need to call strtok(NULL,"delimiter"); after calling strtok(stringname, "delimiter") ;

This function can be very handy to extract part of a code or phone number. Typically any land line number has three parts. They are country code, city code and the phone number. We can write a small program from which we will accept a phone number from the user and then extract these three sections from that number. Here is the code and the sample output.

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

int main()
{
      char s[16];
      puts("Enter Number as [country_code-City_code-Phone Number ,
                  Eg 123-456-12345678] :" );
      gets(s);
      char *p;
      p=strtok(s,"-");
      printf("Country Code : %s\n",p);
      p = strtok(NULL,"-");
      printf("City Code : %s\n",p);
      printf("Phone Number : %s\n",strtok(NULL,"-"));
      return 0;
}
```

A Sample Run of this code
Enter Number as [country_code-City_code-Phone Number , Eg 123-456-12345678]:
91-011-25882746
Country Code: 91
City Code: 011
Phone Number: 25882746

Next in this chapter there is a function Split() that uses strtok() internally to tokenize a given line with a given delimiter. If you want, you can jump there to understand how that works.

4.17 WHAT DO YOU MEAN BY PREFIX OF A STRING?

A string is said to be prefix of another string if the second one starts with the first. For example "Wonder" is a prefix of the string "Wonderful" because "Wonderful" starts with "Wonder".

Example 4.1 ***Write a program to find whether a string is a prefix of another string or not.***

Solution

```
int isPrefix(char *Text,char *Pattern)
{
int Pref = 1;
int i = 0;
int j = 0;
if(Text[0]==Pattern[0])
{
      for(i=0,j=0;i<strlen(Pattern);i++,j++)
```

```
{
      for(i=0,j=0;i<strlen(Pattern);i++,j++)
      {
            if(Text[j]!=Pattern[i])
            {
                  Pref = 0;
                  break;
            }
      }
      else
            Pref = 0;
      return Pref;

}
```

4.18 WHAT DO YOU MEAN BY SUFFIX OF A STRING?

A string is said to be suffix of another string if the second one ends with the first. For example "long" is a Suffix of the string "Lifelong" because "Lifelong" ends with "long". Later in this chapter you will find two functions **startswith()** and **endswith()** that can be re-written as wrappers of these two functions **isPrefix()** and **isSuffix()**. *Try that yourself!*

Example 4.2 ***Write a program to find whether a string is a suffix of another string or not.***

Solution

```
int isSuffix(char *Text,char *Pattern)
{
      int i = 0;

      int j = 0;
      int Suff = 1;
      if(Text[strlen(Text)-1]==Pattern[strlen(Pattern)-1])
      {
            for(i=strlen(Text)-2,j=strlen(Pattern)-2;j>=0;i--,j--)
            {
                  if(Text[i]!=Pattern[j])
                  {
                        Suff = 0;
                        break;
                  }

            }
      }
      else
            Suff = 0;
      return Suff;
}
```

4.19 WHAT DO YOU MEAN BY SUBSEQUENCE OF A STRING?

Sometimes people misunderstand subsequence as a substring, which is not correct. There is a subtle difference between these two. A string is called a subsequence of the another string, if the characters of the first occur in the second from left to right, but not necessarily in contiguous locations unlike substrings. For example "**Wine**" is a subsequence of the string "**W**orld **i**s **n**ot **e**nough".

Example 4.3 ***Write a program to find whether a string is subsequence of another string or not. Assume that the alphabets present in both the strings are unique.***

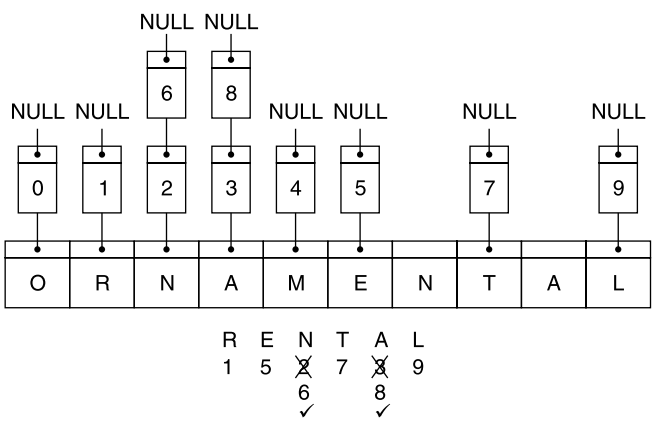
Solution This version works only when *a* and *b* do not have any duplicate characters.

```
int isSubSequence(char *a,char *b)
{
      //Wonderful
      //oder
      int iss = 1;
      int i = 0, j = 0 , k = 0;
      int arr[30];
      for(i=0;i<strlen(b);i++)
      {
            for(j=0;j<strlen(a);j++)
            {
                  if(b[i]==a[j])
                  {
                        arr[k] = j;
                        k++;
                  }
            }
      }
      for(i=0;i<k-1;i++)
      {
            if(arr[i+1]<arr[i])
            {
                  iss = 0;
                  break;
            }
      }
      return iss;

      //O(MN)
}
```

This above function finds whether b is a subsequence of **a** or not, provided **a** and **b** do **NOT** have any duplicate characters. The time complexity of the above code is O(MN + K) where M is the length of the text, N is the length of the Pattern **b.**

Try Yourself*: Try to modify this above program to allow duplicate in pattern* ***a*** *and* ***b****:*

*For example the above code will say that "****order****" is a subsequence for "w****o****n****der****ful" but it will fail to identify that "****rental****" is a subsequence of "****or****nam****ental****" because* ***'a'*** *occurs once before* ***'e'*** *and the above program does not store location information of characters of the word.*



Maintain a linked list of the positions for each character. Write a structure to store each character and its location list. Now once you preprocess the main word (example "Ornamental" in the figure) then you will be ready for checking whether another string (Like rental) is a subsequence of it or not. The picture explains the rest for you. Try it!

How to Check whether a Word starts with a Prefix or not

```
int startsWith(char *string,char *pattern)
{
  int start = 1;
  int i=0;
  for(;i<strlen(pattern);i++)
  {
        if(string[i]!=pattern[i])
        {
              start=0;
              break;
        }
  }
  return start;

}
```

We can also write the above method using strncmp as

```
  int startsWith(char *string,char *pattern)
  {
        return strncmp(string,pattern,strlen(pattern));
}
```

How to Check whether a Word Ends with a given Suffix or not.

```
//This function returns the reverse of a string.
char* rstring(char *s)
{
      char rs[30];
      int i;
      int j;
```

```
            for(i=strlen(s)-1,j=0;i>=0;i--,j++)
                  rs[j]=s[i];
            rs[j]='\0';
            return rs;
      }
      //This function finds out whether the given string with the given
pattern
      int endsWith(char *string,char *pattern)
      {
            char rs[20],rp[20];
            strcpy(rs,rstring(string));
            strcpy(rp,rstring(pattern));
            return startsWith(rs,rp);//Notice how startsWith() is re-
            used.
      }
```

Example 4.4 ***Write a function to check whether a string matches the following basic asterisk (*) wild character matches or not. Namely,***
<pattern>*
***<pattern>**
<pattern1>*<pattern2>
<pattern>

Solution Here is a simple function that uses the startsWith and endsWith function defined above in order to match the four wildcard patterns mentioned. It doesn't work for a generalized combination of those wildcard characters.

```
int wildCharMatch(char *string,char *pattern)
{
      char t[20];
      char *p = strchr(pattern,'*');
      int starindex =  p - pattern;
      if(pattern[0]=='*')
      {
            if(pattern[strlen(pattern)-1]=='*')
            {
                  strcpy(t,substring(pattern,1,strlen(pattern)-2));
                  return strstr(string,t)!=NULL?1:0;
            }
            else
            {
                  strcpy(t,substring(pattern,1,strlen(pattern)-1));
                  return endsWith(string,t);
            }
      }

if(pattern[strlen(pattern)-1]=='*')
{
      strcpy(t,substring(pattern,0,strlen(pattern)-2));
      return startsWith(string,t);
}
```

```
if(starindex!=0 || starindex!=strlen(pattern)-1)
{
      strcpy(t,substring(pattern,0,starindex-1));
      if(startsWith(string,t)==1)
      {

strcpy(t,substring(pattern,starindex+1,strlen(pattern)-1));
            return endsWith(string,t);
       }
 }

 else
       return 0;
}
```

How to accept a String of Words Delimited with Space and Return a Linked List of Those Words as char*

```
typedef struct String
{
      char s[20];
      struct String *next;
}String;

 String* push_back_String(String *last,char *s)
 {
      if(last==NULL)
      {
            last = (String *)malloc(sizeof(String));
            strcpy(last->s,s);
            last->next = NULL;
            return last;
      }
      else
      {
            String *p = (String *)malloc(sizeof(String));
            last->next = p;
            strcpy(p->s,s);
            p->next = NULL;
            return p;
     }
 }
```

```
//This function splits the given string by the provided delimeter.
String* split(char *string,char *del)
{
String* h=NULL;
String* ch=NULL;
char *p;
p = strtok(string,del);
h = push_back_String(h,p);
ch = h;
while((p=strtok(NULL,del))!=NULL)
```

```
                h = push_back_String(h,p);
            return ch;//Returning the head of the list
      }
```

How to Count the Total Number of Words in a Sentence

```
int countWords(String *words)
{
            int totalwords = 0;
            String *cwords=words;
            for(;cwords!=NULL;cwords=cwords->next)
                  totalwords++;
            return totalwords;
}
```

How to Replace a Word by another Word from a Phrase

```
char* replace(char phrase[81],char *word,char *newword)
{
      int count = 0;
      int i,j=0;
      String *words = NULL;
      words = split(phrase," ");
      String *cwords = NULL, *modphrase  = NULL;
      char newphrase[81];
      for(;words!=NULL;words=words->next)
      {
            if(strcmp(words->s,word)==0)
            {
                  cwords = push_back_String(cwords,newword);
                  count++;
                  if(count==1)
                        modphrase = cwords;
                  cwords = push_back_String(cwords," ");
            }
            else
            {
                  cwords = push_back_String(cwords,words->s);
                  count++;
                  if(count==1)
                        modphrase = cwords;
                  cwords = push_back_String(cwords," ");

            }

      }

      for(;modphrase!=NULL;modphrase=modphrase->next)
            for(i=0;i<strlen(modphrase->s);i++,j++)
                  newphrase[j]=modphrase->s[i];

      newphrase[j]='\0';

      return newphrase;

}
```

When called with the following client code

```
char phrase[]={"A hundred fathoms a hundred fathoms away from home."};
strcpy(t,replace(phrase,"fathoms","miles"));
puts(t);
```

it prints the following string.

```
A hundred miles a hundred miles away from home.
```

How to Delete all Occurrences of a given Word from a Sentence

```
char* DeleteWord(char *sentence,char *word)
{
      int i=0,j=0;
      char cs[200];
      char modifiedsentence[200];
      int count = 0;
      String *cwords = NULL;
      String *modifiedline = NULL;
      strcpy(cs,sentence);

      String *words = split(cs," ");
      for(;words!=NULL;words=words->next)
      {
            if(strcmpi(words->s,word)!=0)
            {
                  cwords = push_back_String(cwords,words->s);
                  count++;
                  if(count == 1)
                        modifiedline = cwords;
                  cwords = push_back_String(cwords," ");

            }

      }
      for(;modifiedline!=NULL;modifiedline=modifiedline->next)
            for(i=0;i<strlen(modifiedline->s);i++,j++)
                  modifiedsentence[j] = modifiedline->s[i];

      modifiedsentence[j] = '\0';

      return modifiedsentence;
}
```

Here is a client code snippet that demonstrates the usage of the above function.

```
strcpy(t,"Anger is never without reason but seldom with a good one");
strcpy(t,DeleteWord(t,"without"));
strcpy(t,DeleteWord(t,"reason"));
strcpy(t,DeleteWord(t,"but"));
strcpy(t,DeleteWord(t,"seldom"));
strcpy(t,DeleteWord(t,"with"));
strcpy(t,DeleteWord(t,"a"));
strcpy(t,DeleteWord(t,"one"));
puts(t);
```

This outputs

```
Anger is never good
```

How to Display Text in a Word Wrap Mode

```
void wordWrap(char *longsentence,int wordsperline)
{
      int count = 0;
      String *words = NULL;
      words = split(longsentence," ");
      for(;words!=NULL;words=words->next)
      {
            printf("%s ",words->s);
            count++;
            if(count==wordsperline)
            {
                  count=0;
                  printf("\n");
            }
      }
}
```

When called by the following client code

```
int main()
{
      char *t;
      strcpy(t,"Few people are capable of expressing with equanimity
      opinions which differ from the prejudices of their social
      environment. Most people are even incapable of forming such
      opinions Albert Einstein");
      wordWrap(t,3);//Wraps the text with maximum three words per line

      getch();
      return 0;
}
```

it generates the following output:

```
Few people are
capable of expressing
with equanimity opinions
which differ from
the prejudices of
their social environment.
Most people are
even incapable of
forming such opinions
Albert Einstein
```

How to Demonstrate the Random Cipher Encryption of a Text

```
char* encrypt(char *pwd)
{
      char *epwd;
      int i,j;
//Add more symbols if you want
      char a[]={'a','b','c','d','e','f','g','h','i','j','k'
,'l','m','n','o','p','q','r','s','t','u','v','w','x','y','z',
'1','2','3','4','5','6','7','8','9','0'};

char b[]={'0','9','8','7','6','5','4','3','2',
'1','z','y','x','w','v','u','t','s','r','q','p','o','n','m','l',
           'k','j','i','h','g','f','e','d','c','b','a'};
```

```
for(i=0;i<strlen(pwd);)
{
      for(j=0;j<strlen(a);j++)
      {
            if(pwd[i]==a[j])
            {
                  epwd[i]=b[j];
                  i++;
            }
      }
      epwd[i]='\0';
      return epwd;

}
```

How to Decrypt a Text Encrypted with the Above Function

```
char* decrypt(char *pwd)
{
      char *dpwd;
      int i,j;
      char a[]={'a','b','c','d'
                ,'e','f','g','h'
                ,'i','j','k','l'
            ,'m','n','o','p','q'
            ,'r','s','t','u','v'
            ,'w','x','y','z','1'
            ,'2','3','4','5','6'
            ,'7','8','9','0'};

      char b[]={'0','9','8','7'
               ,'6','5','4','3','2'
               ,'1','z','y','x','w'
               ,'v','u','t','s','r'
               ,'q','p','o','n','m'
               ,'l','k','j','i','h'
               ,'g','f','e','d','c'
               ,'b','a'};

      for(i=0;i<strlen(pwd);)
      {
            for(j=0;j<strlen(a);j++)
            {
                  if(pwd[i]==b[j])
                  {
                        dpwd[i]=a[j];
                        i++;
                  }
            }
      }
      dpwd[i]='\0';
      return dpwd;
}
               ,'l','k','j','i','h'
               ,'g','f','e','d','c'
               ,'b','a'};

      for(i=0;i<strlen(pwd);)
```

```
{
      for(j=0;j<strlen(a);j++)
      {
            if(pwd[i]==b[j])
            {
                  dpwd[i]=a[j];
                  i++;
            }
      }
      }
      dpwd[i]='\0';
      return dpwd;
}
```

How to Represent a String as a Linked List of Characters

```
typedef struct character
{
      char c;
      struct character *next;
      struct character *prev;
}character;

typedef character* string;
```

How to Create a New String Using the Above Linked List Representation of the Strings

```
string push_back(character *last,char c)
{
      if(last==NULL)
      {
            last = (character *)malloc(sizeof(character));
            last->c = c;
            last->next = NULL;
            last->prev = NULL;
            return last;
      }
      else
      {
            character *p = (character *)malloc(sizeof(character));
            last->next = p;
            p->prev = last;
            p->c = c;
            return p;
      }
}
string createNew(char *ns)
{
      character *cs = NULL;
      character *s = NULL;
      int i = 0;
      for(i=0;i<strlen(ns);)
      {
            s = push_back(s,ns[i]);
            i++;
            if(i==1)
                  cs = s;
      }
```

```
      s=push_back(s,'\0');
      return cs;
}
```

How to Display such a String Represented by a Linked List

```
void displayString(string s)
{
      for(;s!=NULL;s=s->next)
      {
            if(s->c=='\0')
                  break;
            else
                  putc(s->c,stdout);
      }
}
```

How to Extract Substring of a String Starting from an Index and Ending at Another Index

```
char* substring(char *string,int start,int end)
{
      char *temp;
      int i,j;
      for(i=start,j=0;i<=end;i++,j++)
            temp[j] = string[i];
      temp[j] = '\0';
      return temp;

}
```

How to Trim a Specified Number of Characters from the Left of a Given String

```
char* trimleft(char *word,int index)
{
      char *tl;
      int last = strlen(word);
      strcpy(tl,substring(word,index,last));
      return tl;
}
```

This function will trim the specified number of characters from the left of the given word. Here is how to call the function.

`trimleft(",Wonderful",1)` will return `"Wonderful"`

How to Trim a Specified Number of Characters from the Right of a Given String

```
char* trimright(char *word,int index)
{
      char *t;
      int x = strlen(word);
      strcpy(t,substring(word,0,x-index-1));
      return t;
}
```

The function will trim the specified number of characters from the right of the given word. Here is how to call the function

`trimright("Lovely",2)` will return `"Love"`

How to Pad *n* Number of Specified Characters to the Left of a String

```
char* padleft(char *word,char c,int n)
{
      char *t;
```

```
      int i;
      for(i=0;i<n;i++)
            t[i]=c;
      t[i]='\0';
      strcat(t,word);
      return t;
}
```

The function will pad n number of characters c at the left of the word. Here is an example run.
`padleft("100.00",'$',1)` will return $100.00

How to Pad *n* Number of Specified Characters to the Right of a String

```
//In this function we have not used strcat()
char* padright(char *word,char c,int n)
{
      char *t;
      int i,j;
      for(i=0;i<strlen(word);i++)
            t[i]=word[i];
      for(j=i;j<n+i;j++)
            t[j]=c;
      t[j]='\0';
      return t;
}
```

This function will pad the given word at the left with the given number of characters at the right side. Like if the function is called like `padright("chapter0",'?',2)` will return "chapter0??"
We can use these building block functions to create the following functions.

How to Trim White Spaces from Both Sides of a Word

```
//It is assumed that the word passed as a parameter to this function has
//whitespaces on both sides.
char* trim(char *word)
{
      char *templ;
      char *tempr;
      int i = 0;
      int index = 0;
      for(;i<strlen(word);i++)
      {
       if(word[i]!=' ')
       {
             index = i;
             break;
       }
    }
      strcpy(templ,trimleft(word,index));//Left whitespaces trimmed
      for(i=0;i<strlen(templ);i++)
      {
            if(templ[i]==' ')
                  break;
      }
      strcpy(tempr,trimright(templ,i));//Right whitespaces trimmed
      return tempr;

}
```

How to Pad a String with a Specified Character and an Alignment Choice

```
enum {ALLIGN_LEFT = -1, ALLIGN_CENTER, ALLIGN_RIGHT};
```

```
char *pad(char *word,int totalwidth,char c,int allignment)
{
	//-1 left 0 center 1 right
	char temp[100];
	int len = strlen(word);
	if(allignment==ALLIGN_LEFT)
	{
		//If we have to make the string appear left alligned,
		//we have to pad characters to its right hand side
		strcpy(temp,padright(word,c,totalwidth-len));
		return temp;
	}
	if(allignment==ALLIGN_RIGHT)
	{
		//If we have to make the string appear right alligned,
		//we have to pad characters to its left hand side.
		strcpy(temp,padleft(word,c,totalwidth-len));
		return temp;
	}
	if(allignment==ALLIGN_CENTER)
	{
		//If we have to make the string appear center alligned
		//we shall have to pad same number of characters to its both ends.
		strcpy(temp,padleft(word,c,(totalwidth-len)/2));
		strcpy(temp,padright(temp,c,(totalwidth-len)/2));
		//Warning! We might be less than the total number of characters,
		//So lets add them at the end.
		if(strlen(temp)<totalwidth)
			strcpy(temp,padright(temp,c,totalwidth-strlen(temp)));

		return temp;
	}
}
```

Here is a sample client code snippet that demonstrates the above function **pad():**

```
strcpy(t,pad("Love",20,'.',ALLIGN_LEFT));
puts(t);
printf("Length = %d\n",strlen(t));
strcpy(t,pad("Love",20,'.',ALLIGN_CENTER));
puts(t);
printf("Length = %d\n",strlen(t));
strcpy(t,pad("Love",20,'.',ALLIGN_RIGHT));
puts(t);
printf("Length = %d\n",strlen(t));
```

And here is the output of this code..

```
Love................
Length = 20
........Love........
Length = 20
................Love
Length = 20
```

How to Remove all the Blank Spaces in a Phrase

```
char* sweepwspace(char *phrase)
{
	int i = 0,j=0;
	char temp[100];

	for(;i<strlen(phrase);i++)
```

```
{
      if(phrase[i]!=' ')
      {
            temp[j]=phrase[i];
            j++;
      }
}
temp[j]='\0';
return temp;
}
```

How to Extract all the *k*-grams of a String for a given *k*

```
String* kgrams(char *line,int k)
{
      int i = 0;
      int count = 0;
      char temp[50];
      //deleting all the white spaces
      strcpy(line,sweepwspaces(line));
      String *words = NULL,*ktokens = NULL;
      int total  = strlen(line)-k-1;
      for(;i<=total;i+=k)
      {
            strcpy(temp,substring(line,i,i+k-1));
            words = push_back_String(words,temp);
            count++;
            if(count == 1)
                  ktokens = words;
      }
      return ktokens;

}
```

k-grams are used for generating the digital fingerprint of intelligent materials like a document. See the end of chapter on file. There is an algorithm called Winnowing which uses *k*-grams to detect plagiarism. Different nearest neighbor identification algorithms, like "Near Duplicate Document" finding in a web search uses *k* grams.

How to check whether a String is a Valid UPC (Universal Product Code) Code or not

Each product being sold in the market has a different product code. This code is known as *Universal Product Code* or UPC in short. This concept was first introduced for making grocery store checkout

time faster. The first digit in the UPC tells about the type of the product. The last digit (6 in this case) is the check digit. This digit tells the scanner whether the code had been read properly or not.

The algorithm to check whether UPC is valid or not is popularly known as *Module* 10 *algorithm* or as *Luhn Algorith* according to its discoverer Hans Peter Luhn.

Step 1: The even place digits are added

Step 2: The number at odd places are multiplied by 3 and then the sum of the digits of all generated numbers are found.

Step 3: If the total of the numbers obtained in step 1 and step 2 is divided by 10.
If the remainder is 0 that means the UPC code is valid. Otherwise it means that the UPC code is invalid.

```
int digsum(int number)
{
      int sum = 0;
      while(number!=0)
      {
            sum+=number%10;
            number/=10;
      }
      return sum;
}


int isValidUPC(char *UPC)
{
      String *kgs = kgrams(UPC,1);
      int count = 1;
      int EvenSum = 0;
      int OddSum = 0;
      for(;kgs!=NULL;kgs=kgs->next)
      {
            if(count%2!=0)
                  OddSum+=digsum(atoi(kgs->s));
            else
                  EvenSum+=atoi(kgs->s);
            count++;
      }

      return (EvenSum + 3 * OddSum)%10==0;

}
```

4.20 HOW TO CHECK WHETHER A STRING IS A VALID ISBN OR NOT

The algorithm to check whether an ISBN number is valid or not is just a dialect of 10 module algorithm described as follows.

```
int isValidISBN(char *ISBN)
{
      String *kgs = NULL;
      int weight = 1;
      int sum = 0;
      if(strlen(ISBN)!=10)
            return 0;
      else
      {
            kgs = kgrams(ISBN,1);
            for(;kgs->next!=NULL; kgs = kgs->next ,weight++)
                        sum+=atoi(kgs->s)*weight;
            if(sum%11==atoi(kgs->s))
                  return 1;
            else
                  return 0;
      }

}
```

4.21 HOW TO CHECK VALIDITY OF A SOCIAL INSURANCE NUMBER (SIN) CODE

In Canada, every person has a social insurance number which is unique and used for identification. This number can be validated with a variant of Check Digit or Module 10 algorithm. This algorithm is implemented in the function below.


```
int isValidSIN(char *SIN)
{
      String *kgs = kgrams(SIN,1);
      int count=1;

      int EvenSum = 0;
      int OddSum = 0;
      for(;kgs!=NULL;kgs=kgs->next)
      {
            if(count%2==0)
                  //Find digit sum of numbers we get by multiplying the
digits at
                  //even locations.
                  EvenSum+=digsum(atoi(kgs->s)*2);
            else

                  OddSum+=atoi(kgs->s);
            count++;
      }
      return (EvenSum + OddSum )%10==0;
}
```

4.22 HOW TO CHECK WHETHER A GIVEN CREDIT CARD NUMBER IS VALID OR NOT

To validate a credit card number three things need to be tested. First of all we need to find out whether

the card has a proper length for its type or not. For example, if the card is of type "VISA" then the length of the card needs to be validated. Once that is done, then it is checked whether the card number starts with a predefined globally accepted prefix for VISA type cards or not. Once all these criteria are passed then the number is validated using check digit or Luhn or Module 10 algorithm.

The following two functions implement the algorithm.


```
int IsValidCheckDigit(char *ccn)
{
      String *kgs = kgrams(ccn,1);
      int count = 1;
      int EvenSum = 0;
      int OddSum = 0;
      for(;kgs!=NULL;kgs=kgs->next)
      {
            if(count%2==0)
                  EvenSum+=digsum(2*atoi(kgs->s));
            else
                  OddSum+=digsum(atoi(kgs->s));
            count++;
      }
      return (EvenSum + OddSum)%10==0;
}

int isValidCreditCard(char *type, char *ccn)
{

      if(strcmpi(type,"American Express")==0)
      {
            if(strlen(ccn)==15)
            {
                  if(startsWith(ccn,"34")==1 || startsWith(ccn,"37")==1)
                  {
                        if(IsValidCheckDigit(ccn)==1)
                              return 1;
                        else
                              return 0;
                  }
                  else
                        return 0;
            }
            else
                  return 0;
      }
      if(strcmpi(type,"Carte Blanche")==0 || strcmpi(type,"Diners Club")==0 )
      {
            if(strlen(ccn)==14)
            {
                  if(startsWith(ccn,"300")==1
                  || startsWith(ccn,"301")==1
                  || startsWith(ccn,"302")==1
                  || startsWith(ccn,"303")==1
                  || startsWith(ccn,"304")==1
                  || startsWith(ccn,"305")==1
                  || startsWith(ccn,"36")==1
                  || startsWith(ccn,"38")==1)
                  {
                        if(IsValidCheckDigit(ccn)==1)
                              return 1;
```

```
                else
                        return 0;
        }
        else
                return 0;
}
else
        return 0;
        }

        if(strcmpi(type,"Discover")==0 )
        {
                if(strlen(ccn)==16)
                {
                        if(startsWith(ccn,"6011")==1)
                        {
                                if(IsValidCheckDigit(ccn)==1)
                                        return 1;
                                else
                                        return 0;
                        }
                        else
                                return 0;
                }
                else
                        return 0;
        }

        if(strcmpi(type,"Enroute")==0 )
        {
                if(strlen(ccn)==15)
                {
                        if(startsWith(ccn,"2014")==1 || startsWith(ccn,"2149")==1)
                        {
                                if(IsValidCheckDigit(ccn)==1)
                                        return 1;
                                else
                                        return 0;
                        }
                        else
                                return 0;
                }
                else
                        return 0;
        }
if(strcmpi(type,"JCB")==0 )
        {
                if(strlen(ccn)==15 || strlen(ccn)==16)
                {
                        if(startsWith(ccn,"3")==1
|| startsWith(ccn,"2131")==1 || startsWith(ccn,"1800")==1)
{
        if(IsValidCheckDigit(ccn)==1)
                return 1;
        else
                return 0;
}
```

```
                else
                        return 0;
        }
        else
                return 0;
    }
  if(strcmpi(type,"Maestro")==0 )
    {
        if(strlen(ccn)==16)
        {
                if(startsWith(ccn,"6")==1 || startsWith(ccn,"5020")==1)
                {
                        if(IsValidCheckDigit(ccn)==1)
                                return 1;
                        else
                                return 0;
                }
                else
                        return 0;
        }
        else
                return 0;
    }

  if(strcmpi(type,"Mastercard")==0 )
    {
        if(strlen(ccn)==16)
        {
                if(startsWith(ccn,"51")==1
                || startsWith(ccn,"52")==1
                || startsWith(ccn,"53")==1
                || startsWith(ccn,"54")==1
                || startsWith(ccn,"55")==1)
                {
                        if(IsValidCheckDigit(ccn)==1)
                                return 1;
                        else
                                return 0;
                }
                else
                        return 0;
        }
        else
                return 0;
    }
  if(strcmpi(type,"Solo")==0 )
    {
        if(strlen(ccn)==16 || strlen(ccn)==18 || strlen(ccn)==19)
        {
if(startsWith(ccn,"6334")==1 || startsWith(ccn,"6767")==1)
{
      if(IsValidCheckDigit(ccn)==1)
            return 1;
      else
            return 0;
}
else
```

```
                        return 0;
                }
                else
                        return 0;
        }

    if(strcmpi(type,"Switch")==0 )
        {
                if(strlen(ccn)==16 || strlen(ccn)==18 || strlen(ccn)==19)
                        {
                                if(startsWith(ccn,"4903")==1 ||
    startsWith(ccn,"4905")==1
                                        || startsWith(ccn,"4911")==1
                                        || startsWith(ccn,"4936")==1
                                        || startsWith(ccn,"564182")==1
                                        || startsWith(ccn,"633110")==1
                                        || startsWith(ccn,"6333")==1
                                        || startsWith(ccn,"6759")==1)
                                {
                                        if(IsValidCheckDigit(ccn)==1)
                                                return 1;
                                        else
                                                return 0;
                                }
                                else
                                        return 0;
                        }
                        else
                                return 0;
                }

                 if(strcmpi(type,"Visa")==0 )
                {
                        if(strlen(ccn)==13 || strlen(ccn)==16)
                        {
                                if(startsWith(ccn,"4")==1)
                                {
                                        if(IsValidCheckDigit(ccn)==1)
                                                return 1;
                                        else
                                                return 0;
                                }
                                else
                                        return 0;
                        }
                        else
                                return 0;
                }

                if(strcmpi(type,"Visa Electron")==0 )
                {
                        if(strlen(ccn)==16)
                        {
                                 if(startsWith(ccn,"417500")==1
                                ||startsWith(ccn,"4917")==1 || startsWith(ccn,"4913")==1)
                                 {
                                        if(IsValidCheckDigit(ccn)==1)
                                                return 1;
                                        else
```

```
                        return 0;
                }
                else
                        return 0;
        }
        else
                return 0;
}
```

4.23 HOW TO CHANGE THE CASE OF A SENTENCE TO SENTENCE CASE

Sometimes we miss the capitalization after fullstop while composing a letter or a doc. We can correct our text by passing them to the function below. This function makes sure that every character that follows a dot ('.') is in upper case.

```
char* SentenceCase(char *sentence)
{
      int i,index=0;
      for(i=0;i<strlen(sentence);i++)
      {
            if(sentence[i]=='.')
                  sentence[i+1]=toupper(sentence[i+1]);
      }
      return sentence;
}
```

When called with the following client code snippet

```
char *t;
strcpy(t,"I am here.let's folk again!");
strcpy(t,SentenceCase(t));
puts(t)
```

the following output is generated.

I am here.Let's folk again!

Try Yourself: Try to change the above program so that it can handle random capitalization. For example, a call to **`SentenceCase()`** like

```
strcpy(t,"I am here.lET's FoLk agAin!");
strcpy(t,SentenceCase(t));
puts(t)
```

should output

I am here. Let's folk again.

4.24 HOW TO TOGGLE THE CASE OF THE LETTERS OF A SENTENCE

This function toggles the case of each letter in the sentence or the phrase passed.

```
char* ToggleCase(char *sentence)
{
      int i,index=0;
      char *toggledsentence;
      for(i=0;i<strlen(sentence);i++)
      {
            if(sentence[i]>='a' && sentence[i]<='z')
            {
```

```
            toggledsentence[i] = toupper(sentence[i]);
            continue;
      }
      if(sentence[i]>='A' && sentence[i]<='Z')
      {
            toggledsentence[i] = tolower(sentence[i]);
            continue;
      }
            else
            {
                  toggledsentence[i] = sentence[i];
                  continue;
            }
      }
      toggledsentence[i]='\0';
      return toggledsentence;

}
```

When called with the following client code snippet

```
char *t;
strcpy(t,"I am here.let's fOlk agAin!");
strcpy(t, ToggleCase(t));
puts(t)
```

the following output is generated.

```
i AM HERE.LET'S FoLK AGaIN!
```

How to Calculate the Frequency of a given Word in a Sentence

```
int WordFrequency(char *sentence,char *word)
{
      int freq = 0;
      char *csen;
      strcpy(csen,sentence);
      String *words = split(csen," ");
      for(;words!=NULL;words=words->next)
            if(strcmp(words->s,word)==0)
                  freq++;
      return freq;
}
```

To know how this function can be called, see the next question.

How to Display the Word Histogram of a given Sentence

```
void WordHistogram(char *sentence)
{
      char *longsentence;
      char *mostusedword;
      strcpy(longsentence,sentence);
      String *words = split(sentence," ");
      for(;words!=NULL;words=words->next)
            printf("%s %d\n",words->s,WordFrequency(longsentence,
words->s));
}
```

Here is a client code snippet that demonstrates how to use the above function.

```
strcpy(t,"Anger is never without a reason but it is seldom with a good
one");
WordHistogram(t);
```

The following output is generated for this client call.

```
Anger 1
is 2
never 1
without 1
a 2
reason 1
but 1
it 1
is 2
seldom 1
with 1
a 2
good 1
one 1
```

Try Yourself*: As you can see this program shows same word multiple times if it occurs multiple times in the sentence. Create an user defined Word-Frequency List to remove this duplication.*

How to Find the Most Used Word in a Sentence/Phrase/String

```
char* MostUsedWord(char *sentence)
{
      char longsentence[100];
      int maxf;
      char mostusedword[30];
      strcpy(longsentence,sentence);
      String *words = split(sentence," ");
      maxf = WordFrequency(longsentence,words->s);
      for(;words!=NULL;words=words->next)
            if(maxf<WordFrequency(longsentence,words->s))
                  strcpy(mostusedword,words->s);
      return mostusedword;
}
```

When called with the following client code snippet,

```
char *t,*s;
strcpy(t,"If you miss the train I'm on you will know that I am gone");
strcpy(s,MostUsedWord(t));
puts("The most used word is ");
puts(s);
```

the following output is generated.

```
The most used word is
you
```

How to Find whether a Word/Phrase is an Anagram of Another One

An anagram is a different combination of same characters that occur in a word or phrase. While comparing whether two words/phrases are anagrams of one another or not we check for the frequencies of all the characters in two strings. If they match, then the two strings are anagram of one another. Punctuations and special characters are discarded. For example ***"Great Taste!"*** and ***"Gear at Test!"*** are the anagrams of one another. While generating anagrams all the punctuations are discarded.

```
typedef struct CharacterWithFrequency
{
      char c;
      int freq;
      struct CharacterWithFrequency *next;
}CharacterWithFrequency;

int isAnagram(char *phrase,char *diffcomb)
{
      int flag = 0;
      CharacterWithFrequency *s=NULL,*s1=NULL;
      CharacterWithFrequency *t=NULL,*t1=NULL;
      CharacterWithFrequency cf;
      CharacterWithFrequency *temp=NULL;

      int anagram = 1;
      int i=0,j=0;
      int count=0;
      if(strlen(phrase)!=strlen(diffcomb))
            anagram = 0;
      else
      {
            //Creating the histogram of characters for the first string
            for(;i<strlen(phrase);i++)
            {
                  cf.freq = 0;
                  cf.c = phrase[i];
                  for(j=0;j<strlen(phrase);j++)
                  {
                        if(phrase[i]==phrase[j])
                        {
                              cf.freq++;
                        }
                  }
                  if(!Contains(s1,cf.c))
                  {
                        s = push_back(s,cf);
                  }
                  count++;
                  if(count==1)
                        s1 = s;

            }
            count = 0;
            i=0;
            //Creating the histogram of characters of the second string
            for(;i<strlen(diffcomb);i++)
            {
                  cf.freq = 0;
cf.c = diffcomb[i];
for(j=0;j<strlen(diffcomb);j++)
{
      if(diffcomb[i]==diffcomb[j])
      {
            cf.freq++;
      }

}
```

```
            if(!Contains(t1,cf.c))
            {
                  t = push_back(t,cf);
            }
            count++;
            if(count==1)
                  t1 = t;
      }
      }
      //Voila! Now we have both the histogram of strings. So we are ready to run
      //through these histograms to see if they match or not.
      for(;s1!=NULL;s1=s1->next)
      {
         temp = t1;
         //Two words which are anagram of one another
         //s=doctorwho
         //d-1
         //o-3
         //c-1
         //t-1
         //r-1
         //w-1
         //h-1
         //t=torchwood
         //t-1
         //o-3
         //r-1
         //c-1
         //h-1
         //w-1
         //d-1
         for(;temp!=NULL;temp=temp->next)
         {
                if(s1->c==temp->c)
                {
                      flag=1;
                      //Well, we have the same character in both the strings
                      //but their frequencies don't match. So they are not anagram
                      //of one another
                      if(s1->freq!=temp->freq)
                      {
                            anagram = 0;
                            break;
                      }
                }

         }

         //there is some character in one of the string
         //that does not occur at all in another
                if(flag==0)
                {
                      anagram = 0;
                      break;

                }
            }
            return anagram;
      }
```

4.25 HOW TO FIND OUT THE SOUNDEX CODE FOR A GIVEN WORD

A soundex code is the phonetic code of the word. If the soundex codes of two words are same, that means they are pronounced same way. This soundex algorithm was developed for indexing surnames in a census.

Here is the soundex algorithm:

- Delete all occurrences of a,e,i,o,u,w,h and y from the word
- The first character of the soundex code will be the same as that of the word's first character.
- If two consecutive characters are same, then the first one will be considered and the rest all will be ignored.
- If the character encountered in the word is either b,f,p or v put 1 in the corresponding soundex code of the word.
- If the character encountered in the word is either c,k,x,q,g,z,j, or s put 2 in the corresponding soundex code of the word.
- If the character encountered in the word is either d or t put 3 in the corresponding soundex code of the word.
- If the character encountered in the word is l put 4 in the corresponding soundex code of the word.
- If the character encountered in the word is either m or n put code 5 in the corresponding soundex code of the word.
- If the character encountered in the word is r put 6 in the corresponding soundex code of the word.
- Find the substring of the Soundex code till the 4^{th} character.
- If there are fewer than 4 characters in the soundex code, including the initial letter, then pad the right of the soundex code with zeros.

```
//The following function returns the soundex code for a given word.
char* SoundexCode(char *word)
{
      int i,j;
      char temp[30];
      temp[0]=word[0];
      for(i=1,j=1;i<strlen(word);i++)
      {
            if(word[i]==word[i-1])
                  i++;//This is to avoid repetition of characters
            if(word[i]!='a' || word[i]!='e' || word[i]!='h' ||
word[i]!='i'
            || word[i]!='o' || word[i]!='u' || word[i]!='w' ||
word[i]!='y')
            {
                  if(word[i]=='b' || word[i]=='f' || word[i]== 'p' ||
word[i]== 'v')
                  {
                        temp[j]='1';
                        j++;
                  }
if(word[i]=='c' || word[i]=='g' || word[i]=='j'
|| word[i]=='k' || word[i]=='q' || word[i]=='s'
|| word[i]=='x' || word[i]=='z')
{
      temp[j]='2';
      j++;
}
if(word[i]=='d' || word[i]=='t')
```

```
{
      temp[j]='3';
      j++;
}
if(word[i]=='l')
{
      temp[j]='4';
      j++;
}
                  if(word[i]=='m' || word[i]=='n')
                  {
                        temp[j]='5';
                        j++;
                  }

                  if(word[i]=='r')
                  {
                        temp[j]='6';
                        j++;
                  }

            }
      }
      temp[j]='\0';
      strcpy(temp,substring(temp,0,3));
      return temp;
}
```

How to Check whether two Words are Pronounced Same or not Using Their Soundex Codes

```
int isSameSoundex(char *word,char *suspectedhomonym)
{
      if(strcmp(word,suspectedhomonym)==0)
            return 1;
      else
            return 0;
}
```

Here is a client code that tests two functions defined above.

```
int main()
{
      //The soundex code will never be four characters.
      //Anything beyond 4 characters are discarded.
      char t[4],s[4];
      strcpy(t,SoundexCode("build"));
      puts(t);
      strcpy(s,SoundexCode("billed"));
      puts(s);
      printf("%d",isSameSoundex(t,s));
      getch();
      return 0;
}
```

The output of the above client code is as follows.

```
b43
b43
1
```

So **"build"** and **"billed"** both have the same soundex code b430 (Blank means zero) and that's because they are pronounced the same way. If you want to test the above algorithm for different strings, visit Alan Cooper's most extensive list of homonyms *http://www.cooper.com/alan/homonym_list.html* The above algorithm works fine for almost all these strings listed here. The algorithm doesn't work for those homonyms that start with different letters and pronounced the same way somehow because it retains the first letter of the word.

REVIEW QUESTIONS

1. What can be said about x from this statement x = split("L1-333-353","-");
2. What will be the output for trimleft() when applied like trimleft("12345.23",2);
3. How can trimright() be applied to get the whole part of a number with a decimal digit?
4. How can trimleft() be applied to get the decimal part of a number?
5. How can pad() be used to print the output in a fixed width pattern?

PROGRAMMING PROBLEMS

1. Write a function to check whether a string matches the single wildcard pattern or not.
2. Write a program to demonstrate Naïve string search algorithm. This is also known as Brute Force Algorithm.
3. Write a program to demonstrate Karp-Robin Algorithm.
4. Write a program to demonstrate Boyer Moore string search algorithm.
5. Write a program to demonstrate Boyer Moore Horspool string search algorithm.
6. What do you mean by string distance?
7. Write a program to calculate Hamming distance of two strings.
8. Write a program to calculate the levensthian distance of two strings.
9. What do you mean by similarity of two strings?
10. Write a function ExtractDigits() that extracts the digits from a string.
11. Write a function isValidIdentifier() that accepts a string as a variable name and returns 1, if the name can be a valid C Identifier else it returns 0.
12. Write a function isCamelCase() that returns 1, if the string passed as argument is a valid C identifier written in Camel Case.
13. Write a function isPascalCase() that returns 1, if the string passed as argument is a valid C identifier written in Pascal Case.
14. Write a function isAComment() that return 1, if the string passed is a valid C++ single line comment or C Style single line comment. Otherwise the function should return 0.
15. Write a function GetTense() that returns the tense of the sentence that is passed as the argument. For example, the call like GetTense("I shall have been learning PHP next year by this time") will return "Future Perfect Continuous" and a call like GetTense("I shall have finished my dinner by then") will return Future Perfect.
16. Write a function Metaphone() to find the metaphone of a string.

5

Recursion
Time and Again!

INTRODUCTION

Recursion as the name suggests means re occurrence of the same method. Something that is occurring over and over again is recurrence. In this chapter we will learn about different types of recursion and then we will see how this technique is applied to solve different types of problems starting from solving quadratic equations to creation of recursive fractals like Serpinski triangle, Monge Sponge, etc.

5.1 DIFFERENT TYPES OF RECURSION

Binary Recursion A recursive function which calls itself twice during the course of its execution.

Linear Recursion Recursion where only one call is made to the function from within the function (thus if we were to draw out the recursive calls, we would see a straight, or linear, path).

Exponential Recursion Recursion where more than one call is made to the function from within itself. This leads to exponential growth in the number of recursive calls.

Circularity In terms a recursion, circularity refers to a recursive function being called with the same arguments as a previous call, leading to an endless cycle of recursion.

Mutual Recursion A set of functions which call themselves recursively indirectly by calling each other. For example, one might have a set of two functions, is_even() and is_odd(), one defined in terms of the other.

Nested Recursion A recursive function where the argument passed to the function is the function itself. *Ackermann's function* is defined as a Nested Recursion. Q Number generation program is a good example of Nested Recursion.

Recursive Definition A definition defined in terms of itself, either directly (explicitly using itself) or indirectly (using a function which then calls itself either directly or indirectly).

Tail Recursion A recursive procedure where the recursive call is the last action to be taken by the function. Tail recursive functions are generally easy to transform into iterative functions.

5.2 PITFALLS OF RECURSION..

Recursion brings a mixed blessing. It can make simple work of otherwise complex and enormously iterative tasks. It can also be a nightmare that brings the system to its knees by hogging huge amounts of memory.

The down side of recursion starts to sink in when you realize that the functioning of the code is not always immediately apparent on inspection, so another programmer who is unfamiliar with your code might find it difficult to work with.

Debugging recursive routines can be maddeningly difficult. It can take great patience and skill to have success fixing broken recursive code.

Recursive programs tend to have very 'tight' code, so making modifications to an existing design can prove difficult, even impossible. In fact, it's often easier to rewrite a recursive function than it is to patch in changes.

Lastly, perhaps most importantly, recursion can be a resource hog, using up precious memory resources rather quickly. For systems that have scant memory resources (Such as DOS in real mode), the programmer may elect to avoid using recursion unless there is no other choice.

Example 5.1 ***Write a program using recursion that finds the factorial of a number.***

Solution Let's start with a simple example. The following program calculates the factorial of a number.

```
int fact(int n)
{
        if(n==0 || n==1)
      return 1;
        else
      return fact(n-1)*n;
}
```

Example 5.2 ***Write a program using recursion that finds the binomial coefficient.***

Solution The value of Binomial Coefficient is given explicitly by nCk = n!/(n–k)! k!

```
int nCk(int n,int k)
{
      return fact(n)/(fact(n-k)*fact(k));
}
```

This function can be used in several other problems where binomial coefficient is a building block. Notice how the recursive function `fact()` is being used in the function `nCk()`.

5.3 FIBONACCI NUMBERS AND GOLDEN RATIO

Example 5.3 ***Write a program to find the nth Fibonacci Number.***

Solution Fibonacci numbers are not a number series, rather a rule that says add the two seeds provided and you will get the next number and carry on these process with two latest numbers. This series was first created by the Italian mathematician *Leonardo di Pisa*, or *Pisano*, known also under the name Fibonacci in 1202. It is a deceptively simple series, but its ramifications and applications are nearly limitless.

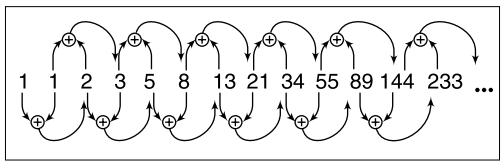


Fig. 5.1

In words: you start with **0** and **1**, and then produce the next Fibonacci number (F_n) by adding the two previous Fibonacci numbers:

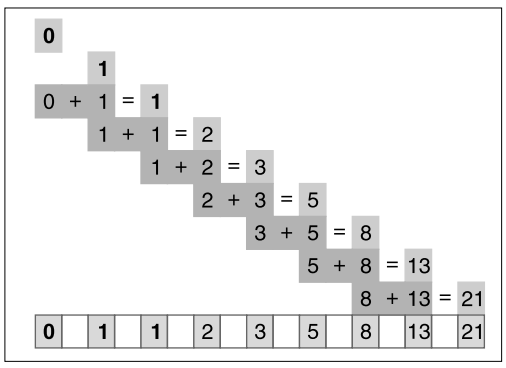


Fig. 5.2

Fibonacci numbers have some very interesting properties. Here are some:

Lucas' Theorem

$$F_m \text{ gcd } F_n = F_{(m \text{ gcd } n)}$$

[gcd = greatest common divisor]

Cassini's Formula

$$F_{n+1} \cdot F_{n-1} - (F_n)^2 = (-1)^n$$

A variant

$$F_{n-2} \cdot F_{n+1} - F_{n-1} \cdot F_n = (-1)^{n-1}$$

Simson's Relation

$$F_{n+1} \cdot F_{n-1} + (-1)^{n-1} = (F_n)^2$$

Shifting Property

$$F_{m+n} = F_m \cdot F_{n+1} + F_{m-1} \cdot F_n$$

```
#include <stdio.h>

int fibo(int num)
{
   if(num==0)
        return 0;
   if(num==1||num==2)
        return 1;
   else
        return fibo(num-1)+fibo(num-2);
}
```

Example 5.4 ***Write a program that calculates the numbers of a Fibonacci type series. Write a function that will accept three values. The first two are the seeds (Like for Fibonacci numbers the first two are 0 and 1) and the number of terms up to which the loop rotates.***

Solution

```
#include <stdio.h>
int fibogen(int num,int first,int second)
{
   if(num==1)
        return first;
   if(num==2)
        return second;
   else
        return fibogen(num-1,first,second)+
               fibogen(num-2,first,second);
}
```

If this function is called with initial values as 2, 1 respectively in place of first and second then we get a series known as *Lucas Series*. First few numbers of Lucas Series are

2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123 .

The *relation between Lucas Numbers and Fibonacci Numbers is*

$$\mathbf{L_n = F_{n-1} + F_{n+1}}$$

Example 5.5 ***Write a program to find the greatest common divisor of two numbers, using Euclid's algorithm.***

Solution Almost 2500 years ago, *Euclid* discovered an algorithm to find the greatest common factor or the test common divisor of two numbers.

```
int gcf(int x,int y)
{
    if(y ==0)

       return x;
    else
       return gcf(y,x%y);
}
```

Notice that this is a ***Tail Recursive Algorithm***.

Example 5.6 ***Write a program to find the GCD using Joseph Stein's Algorithm. This method is also known as binary method.***

Solution Euclid's algorithm involves division and thus it is computationally expensive. A quite different GCD *algorithm* specially suited for the binary number was first proposed by *Joseph Stein* is given by

If u and v are both even then

GCD(u,v) = 2* GCD(u/2,v/2)

If u is even and v is odd then

GCD(u,v) = GCD(u/2,v)

If u and v are both odd then u-v is even so,

GCD(u,v) = GCD(u-v,v)

Here is the code in C

```
int jsgcf(int u,int v)
{
        if(u%2==0 && v%2==0)
                return jsgcf(u/2,v/2);
        if(u%2==0 && v%2!=0)
                return jsgcf(u/2,v);
        else
                return jsgcf(abs(u-v),v);

}
```

5.4 RANDOM NUMBER GENERATION USING RECURSION

Example 5.7 ***Write a program to create a random number sequence using recursion.***

Solution *D.H.Lehmer* came up with a scheme in 1949 to generate a pseudo random number sequence. This is by far the most popular random number generator even today. The algorithm uses four magic integers. They are

M, the modulus 0<m

a, the multiplier 0<=a<m

c, the increment 0<=c<m

Xo the starting value 0<=Xo<m

The desired sequence of random numbers (Xn) is obtained by

Xn+1 = (aXn + c) mod m, n>=0

This method is known as *Linear Congruential Method*. There are many varieties of this method.

Here is the C code.

```
void linrand(int a,int m,int c,int x,int n)
{
        if(n!=0)
        {
                printf("%d\n",(a*x+c)%m);
                n--;
                x = (a*x+c)%m;
                linrand(a,m,c,x,n);
        }
        else
                return;
}
```

Notice that the function linrand() is also *tail recursive.*

Example 5.8 ***Write a program to generate random numbers using quadratic congruent method.***

Solution To do a genuine improvement of linear congruent sequence we have to make it a quadratic in nature. An interesting quadratic method was proposed by R.R.Coveyou. In this random number generator the value of m is a power of 2. Here we have used e as the power of 2.

Here is the code to generate Coveyou sequence…

```
void coveyou(int a,int m,int c,int x,int n)
{
        const int eto2 = (int)pow(2,M_E);
```

```
if(n!=0)
{
        printf("%d\n",(x*x+x)%eto2);
        n--;
        x = (x*x+x)%eto2;
        coveyou(a,m,c,x,n);
}
else
        return;
}
```

5.5 HOW TO GENERATE PSEUDO RANDOM NUMBERS(PRNs) USING VON NUMANN'S MIDDLE SQUARING METHOD

Von Numann used one way to create random numbers. The technique is known as *Middle Extraction Technique*. First a number is taken as the seed. Then that number is squared and the middle part of the number is extracted and is used for the next seed. For example, if the starting number is 1234, then the next seed will be middle digits of the square of 1234 which is 5227 because

$1234^2 = 01522756$

If the square of the initial seed or any seed in the process has odd number of digits then zeros are padded to the left so that the next seed can be extracted.

This method is simple but has couple of drawbacks.

1. After some iterations the PRNs start to repeat themselves and is dependent on the initial seed. It may be possible for different seeds we get different sequences where some numbers are frequently occurring.
2. If all the digits in the middle suddenly become zero then the method fails to execute anymore.

Here is a C code that generates PRNs using this middle extract method:

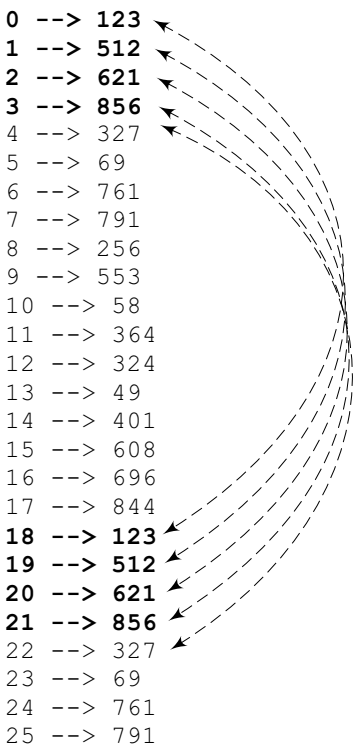
```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

int extractmiddle(char *n)
{
      //12321 -> 232
      int i,j=0;
      char s[4];
      for(i=1;;i++,j++)
      {
            s[j]=n[i];
            if(j==3)
                  break;
      }
      s[j]='\0';
      return atoi(s);

}
int von(int n)
{
```

```
      unsigned long s = n*n;
      char x[20];
      ultoa(s,x,10);
      return extractmiddle(x);
}
```

Here is a client main() function to call the above program.

```
int main()
{


 int i=0;
 int seed = 123;
 for(;i<40;i++)
 {
       printf("%d --> %d\n",i,seed);
       seed = von(seed);
 }

 getch();
 return 0;
}
```

Here is the output of the above program.



```
0 --> 123
1 --> 512
2 --> 621
3 --> 856
4 --> 327
5 --> 69
6 --> 761
7 --> 791
8 --> 256
9 --> 553
10 --> 58
11 --> 364
12 --> 324
13 --> 49
14 --> 401
15 --> 608
16 --> 696
17 --> 844
18 --> 123
19 --> 512
20 --> 621
21 --> 856
22 --> 327
23 --> 69
24 --> 761
25 --> 791
```

```
26 --> 256
27 --> 553
28 --> 58
29 --> 364
30 --> 324
31 --> 49
32 --> 401
33 --> 608
34 --> 696
35 --> 844
36 --> 123
37 --> 512
38 --> 621
39 --> 856
```

The seed is chosen such that the repetition characteristics of PRNs generated through this scheme can be well proved. If you notice the above curves and highlighted entries, you will find that each of the PRN is repeating itself at every 18^{th} iterations. Or in other words 1^{st} and 18^{th} iteration values are same, 2^{nd} and 20^{th} iterated values are same. (Look at the above output).

Ideally the seed should not have any zero amongst its digits, because that will kill the sequence even faster. Here is an example output where the seed was 108:

```
0 --> 108
1 --> 166
2 --> 755
3 --> 700
4 --> 900
5 --> 100 (Border Line PNR)
6 --> 0
7 --> 0
8 --> 0
9 --> 0
```

See anything after 5^{th} iteration is zero.

Even using nonzero digits in the initial seed doesn't guarantee a good sequence. See the following example output generated with seed 222.

```
0 --> 222
1 --> 928
2 --> 611
3 --> 733
4 --> 372
5 --> 383
6 --> 466
7 --> 171
8 --> 924
9 --> 537
10 --> 883
11 --> 796
12 --> 336
13 --> 128
14 --> 638
15 --> 70
```

```
16 --> 900
17 --> 100
18 --> 0
19 --> 0
```

5.6 HOW TO GENERATE THE ACKERMANN'S FUNCTION

Ackermann's function is a function of two parameters whose value grows very fast.

Formal Definition:

- A(0, j) = j+1 for $j \geq 0$
- A(i, 0) = A(i-1, 1) for $i > 0$
- A(i, j) = A(i-1, A(i, j-1)) for i, $j > 0$

This function grows very fast. In compiler design this function is used to check how a compiler can handle recursion problems. Here is the code to generate Ackermann's Number.

```
int ack(int m,int n)

{

       if(m==0 && n!=0)

        return n+1;

       if(n==0 && m!=0)

        return ack(m-1,n);

       if(m!=0 && n!=0)

        return ack(m-1,ack(m,n-1));

}// This is also a tail recursive function.
```

Have you noticed the bold line above in the code? This type of calling is known as *Nested Recursion* because one of the arguments in this call actually calls the ack() function again.

5.7 WHAT IS INVERSE ACKERMANN'S FUNCTION?

$\alpha(m,n) = \min\{i \geq 1: A(i, \lfloor m/n \rfloor) > \log_2 n\}$ where A(i,j) is *Ackermann's function*.

The line above defines the *Inverse Ackermann's function*. Unlike Ackermann's function this function grows very slowly. This function is also known as *alpha function*.

The half brackets in the above expression denote the ceiling of the value m/n. Have you noticed that the inverse Ackermann's Function is defined in terms of Ackermann's function?

Can you use Ackermann's function defined above and generate the reverse Ackermann's Function?

5.8 HOW TO GENERATE TAK FUNCTION FOR GIVEN VARIABLES

TAK function was discovered by I. Takeuchi in 1978. This definition of the function is as follows.

TAK(x,y,z) = z , unless y > x

Else

TAK(x,y,z) = TAK(TAK(x-1,y,z), TAK(y-1,z,x), TAK(z-1,x,y))

```
int TAK(int x,int y,int z)
{
      if (x<=y)
            return z;
      else
            return TAK(TAK(x-1,y,z), TAK(y-1,z,x), TAK(z-1,x,y));
}
```

Both Ackermann function and TAK Functions are used as a benchmark for language with optimization for recursion.

Example 5.9 ***Write a program to find the solution for Tower of Hanoi.***

Solution In this function *n* is the number of disks. The other three variables denote the number of the shafts that will be used as from, to and the temporary shaft number.

```
void hanoi(int n,int from,int to,int temp)
{
      if(n==1)
            printf("Move disc from %d to %d\n",from,to);
      else
      {
            hanoi(n-1,from,temp,to);
            printf("Move disc from %d to %d\n",from,to);
            hanoi(n-1,temp,to,from);
      }
}
```

Example 5.10 ***Write a program to achieve the following. When the user inputs any number, show that in words. For example, if the user enters 99458 then the program will display Ninety-Nine Thousand Four Hundred and Fifty-Eight.***

Solution

```
#include <stdio.h>

int countdigits(int number)
{
     int digits=0;
     while(number!=0)
     {
          number/=10;
          digits++;
     }
     return digits;
}

void Number2Words(number)
{
     char *one2nine[9]=
```

```
                        {
                         "ONE",
                         "TWO",
                         "THREE",
                         "FOUR",
                         "FIVE",
                         "SIX",
                         "SEVEN",
                         "EIGHT",
                         "NINE"
                        };
        char *tens[10]=
                        {
                         "TEN",
                         "TWENTY",
                         "THIRTY",
                         "FOURTY",
                         "FIFTY",
                         "SIXTY",
                         "SEVENTY",
                         "EIGHTY",
                         "NINETY"
                        };
        char *teens[9]=
                        {
                         "ELEVEN",
                         "TWELVE",
                         "THIRTEEN",
                         "FOURTEEN",
                         "FIFTEEN",
                         "SIXTEEN",
                         "SEVENTEEN",
                         "EIGHTEEN",
                         "NINETEEN"
                        };
        char *modifiers[]={"HUNDRED","THOUSAND"};

        int copy=0;
        int digits;
        int dig[5];
        int i=0;
        int isTeens=0;
        int isZero=0;
        int noZeros=0;
        int isAlreadySet=0;

        copy = number;
        digits = countdigits(number);
        if(number==0 && isAlreadySet==0)
        {
```

```
     printf("ZERO");
     isAlreadySet=1;
}
if(digits==1 && isAlreadySet==0)
{
     printf("%s ",one2nine[number-1]);
     isAlreadySet=1;
}

     if(digits==2    &&    (number>10    &&    number    <20)    &&
isAlreadySet==0)
{
     printf("%s",teens[number%10-1]);
     isAlreadySet=1;
}
if(digits==2 && (number%10==0) && isAlreadySet==0)
{
     printf("%s",teens[number%10-1]);
     isAlreadySet=1;
}
if(digits==2 && (number%10==0) && isAlreadySet==0)
{
     printf("%s",tens[number/10-1]);
     isAlreadySet=1;
}

while(number!=0)
{
     dig[i]=number%10;
     if(dig[i]==0 && digits>2 && isAlreadySet==0)
     {
          noZeros++;
          isZero=1;
     }
     number/=10;
     i++;
}
if(digits==2 && isZero==0 && copy>=21

  && copy<=99 && isAlreadySet==0)
{
     printf("%s %s",tens[dig[1]-1],
          one2nine[dig[0]-1]);
          isAlreadySet=1;
}
if(digits==3 && isAlreadySet==0)
{
     if(dig[1]==1)
          isTeens=1;
     if(digits==3
      && isTeens==0
```

```
          && isZero==0
          && isAlreadySet==0)
         {
              printf("%s HUNDRED AND %s %s ",
              one2nine[dig[2]-1],
              tens[dig[1]-1],one2nine[dig[0]-1]);
              isAlreadySet=1;
         }
              if(digits==3
                && isTeens==1
                && isZero==0
                && isAlreadySet==0)
              {
                   printf("%s HUNDRED AND %s ",
                   one2nine[dig[2]-1], teens[dig[0]%10-1]);
                   isAlreadySet=1;
              }
              if(digits==3
              && noZeros==2
              && isAlreadySet==0)
              {
                   if(dig[0]==0 && dig[1]==0)
                   {
                        printf("%s HUNDRED ",one2nine[dig[2]-1]);
                        isAlreadySet=1;
                   }
              }
              if(digits==3 && noZeros==1 && isAlreadySet==0)
              {
                   if(dig[1]==0)//102
                        printf("%s HUNDRED AND %s ",
                                 one2nine[dig[2]-1],
                                 one2nine[dig[0]-1]);
                   if(dig[0]==0)//210
                        printf("%s HUNDRED AND %s
                                ",one2nine[dig[2]-1],
                                 tens[dig[1]-1]);
                   isAlreadySet=1;
              }
         }
}

int main()
{
int number=0;
do
{
     int number=0;
     printf("Enter a number :");
     scanf("%d",&number);
     if(number>100000)
```

```
{
     Number2Words(number/100000);
     printf("LAKH ");
     number=number%100000;
}
if(number>1000)
{
     Number2Words(number/1000);
     printf("THOUSAND ");
     number=number%1000;
}
if(number>0)
{
     Number2Words(number);
}

    }while(1);
}
```

Have you noticed that the program only has code to handle up to 3 digit number but it recursively calls itself so that it can work properly to convert the bigger integers.

5.9 SOLVING NON-LINEAR EQUATIONS USING RECURSION

Example 5.11 ***Write a program to find the nth root of a number.***

Solution There are n roots of the equation $y = x^n$. Now the principal nth root of $y = f(x) = A$ is a positive real number x such that

$x^n = A$.

There is an algorithm living till the time of Babylonians to get the principal root of a number. It is given by the following recursive relation.

$$x_{k+1} = \frac{1}{n}\left[(n-1)x_k + \frac{A}{x_k^{n-1}}\right]$$

In order to find the square root of a number $n = 2$ and then the above formula reduces to

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{A}{x_k}\right)$$

Here is the C code:

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

double root(int A,double Xo,int n,int iter)
{
      double r;
      r = ((n-1)*Xo+A/pow(Xo,n-1))/n;
      iter--;
      if(iter==0)
```

```
            return r;
      else
            r = root(A,r,n,iter);
      return r;
}

int main()
{
      printf("Root is :%f",root(18,4,2,20));
      return 0;
}
```

Example 5.12 ***Write a Program to find the root of a function using Newton–Raphson Method. This method is recursive in nature.***

Solution When the derivation of $f(x)$ is a simple expression and easily found, the real roots of $f(x) = 0$ can be computed rapidly by a process called *Newton-Raphson method* as shown in the above figure:

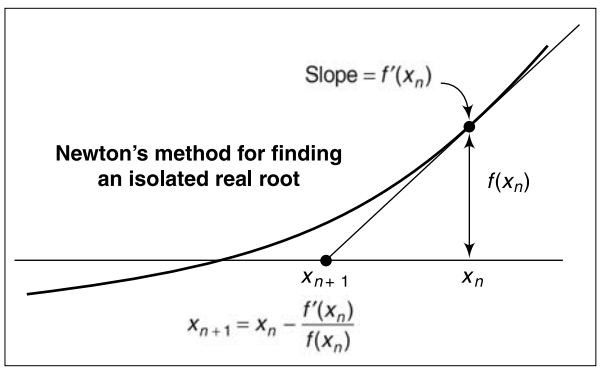


Fig. 5.3

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

double fun(double x)
{
     //Define your Function here
      return pow(x,3)-x-4;
}

double dfun(double x)
{
     //Define the derivative of the function here
      return 3*pow(x,2)-1;
}

double NewtonRaphson(double Xo,int iter)
{
      Xo=Xo-fun(Xo)/dfun(Xo);
      printf("%f\n",Xo);
      iter--;
      if(iter==0)
            return Xo;
      else
          //Tail recursive call
            Xo = NewtonRaphson(Xo,iter);
      return Xo;

}
int main()
```

```
{
      printf("Root is  : %f",NewtonRaphson(2,10));
      return 0;
}
```

Example 5.13 ***Write a program to find the root of a function using bisection method.***

Solution The method of bisection to find the roots of non-linear equations is old and trivial and in a way inefficient. If for two values of x, a and b the two values of y, $y(a)$ and $y(b)$ have different signs, then, there exists a root of $f(x)$ between a and b. As the name suggests, method of bisection, bisects this span and the point of bisection replaces either a or b depending on the sign of $f(x)$ at point of bisection. For example if $f(a)$ is positive and $f(b)$ is negative then there is a root between a and b. According to bisection method the close approximation of the root is $x = (a+b)/2$. Now if $f(x)$ is positive then a will be replaced by else b will be replaced by x. If $f(x) = 0$, then x is a perfect root of $f(x)$.

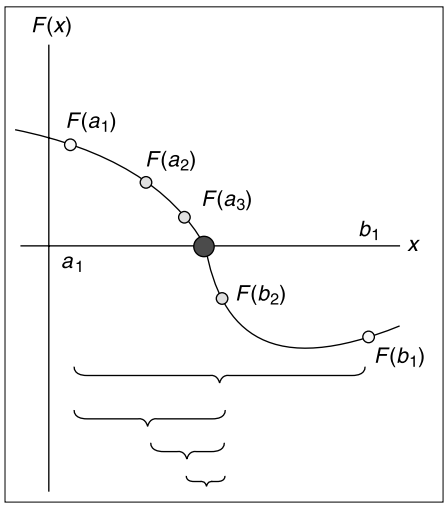


Fig. 5.4

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

double fun(double x)
{
     //Define your Function here
      return pow(x,3)-x-4;
}

double bisection(double a,double b, int iter)
{
      double fa=fun(a);
      double fb=fun(b);
      double c = 0.5*(a+b);//Calculating the approximate root
      if(fun(c)*fa<0)
            b=c;
      if(fun(c)*fb<0)
            a=c;
      iter--;
      if(iter==0)
            return c;
      else
            c = bisection(a,b,iter);
      return c;
}

int main()
{
      printf("Bisection :%f\n",bisection(1,2,10));
      return 0;
}
```

Example 5.14 ***Write a program to find the root using Regula–Falsi method.***

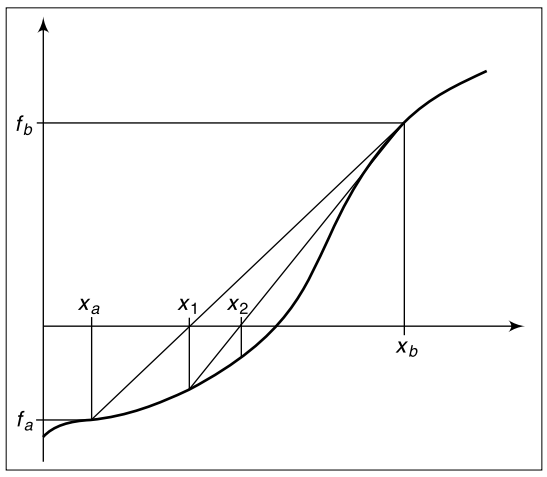


Fig. 5.5

Solution The oldest method for computing the real roots of a numerical equation is the method of false position, or regula falsi. In this method we find two numbers *xa* and *xb* (*as shown in the figure above*) between which the root lies. These numbers should be as close as possible since the root lies between *xa* and *xb* the graph for $y = f(x)$ must cross the x axis between $x = xa$ and $x = xb$ and $y(a)$ and $y(b)$ must have opposite signs.

Now since the portion of a smooth curve is practically straight for a short distance, it is ok to assume that changes in $f(x)$ is same as changes in x for a short interval. The method of false position is based on these principles cause it assumes as per the above figure, that the graph $f(x)$ is a straight line between $(a, f(a))$ and $(b, f(b)$ and it crosses the x axis at $x1$. But we can clearly see that $x1$ is not the solution. As per the above figure $x1$ is still left to the root and closer to it than *xa*. So now *xa* will be replaced by $x1$ and the same process will follow further in a recursive manner.

//This code finds

```
double RegulaFalsi(double x0,double x1,int iter)
{
      double x2=x0-(fun(x0)*(x1-x0))/(fun(x1)-fun(x0));
      iter--;
      //We might find a perfect root or
      //we might end up our desired number of iterations.
      //Either way we come out of the loop.
      if(iter==0 || fun(x2)==0)
            return x2;
      else
      {
            if(fun(x2)>0)
                  x1=x2;
            if(fun(x2)<0)
                  x0=x2;
            x2=RegulaFalsi(x0,x1,iter);//Tail Recursion
      }
      return x2;
}
```

Here is a graph that shows the performance summary for these tree methods.

From this graph we can predict that almost after 5 to 6 iterations all these methods start to give the same result. But rate of convergence towards the correct result is highest for Newton Raphson method provided the starting point is supplied properly. From the above figure we can conclude that performances of these methods are in the order:

Bisection Method < Regula–Falsi Method < Newton–Raphson Method

Example 5.15 ***Write a program to find the value of a function using Inverse Quadratic Interpolation.***

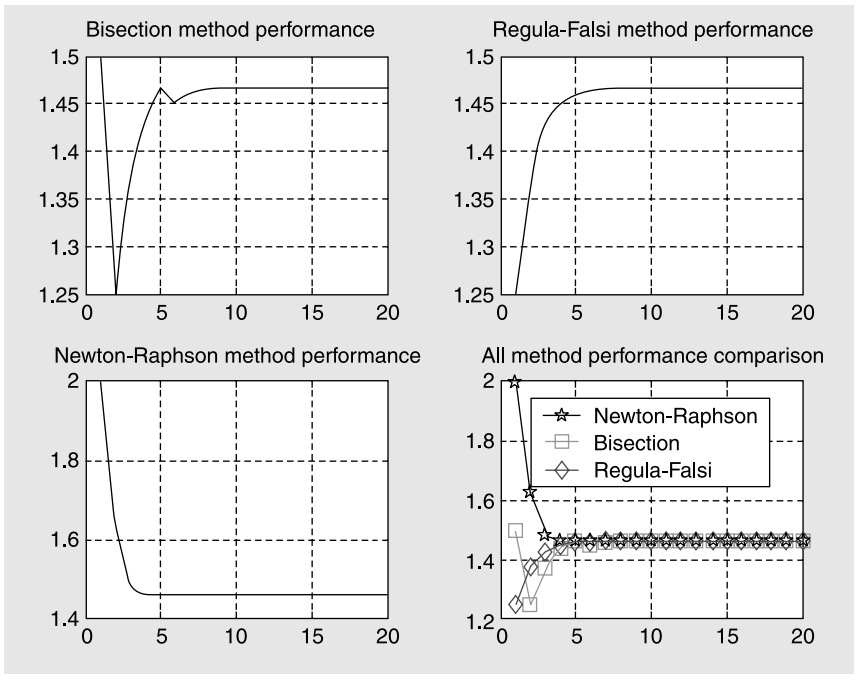


Fig. 5.6

Solution This method is not used individually but is used in hybrid numerical in *Dekker's and Brent's method*. The concept here, is first the inverse quadratic interpolation found at a point starting from three initial guesses *xn*,*xn*-1 and *xn*-2 using Lagranges distribution. Let that point be *f*^(–1) (*y*) then the solution for $y = f(x) = 0$ is given by the following equation.

$$x_{n+1} = \frac{f_{n-1} f_n}{(f_{n-2} - f_{n-1})(f_{n-2} - f_n)} x_{n-2} + \frac{f_{n-2} f_n}{(f_{n-1} - f_{n-2})(f_{n-1} - f_n)} x_{n-1}$$

$$+ \frac{f_{n-2} f_{n-1}}{(f_n - f_{n-2})(f_n - f_{n-1})} x_n$$

Here's the function that calculates the value of X_{n+1}

```
double inversequad(double a,double b,double c)
{
      double first=0,second=0,third=0,next=0;
      double ab=
       first=(fun(b)*fun(c)*a)/(((fun(a)-fun(b))*((fun(a)-fun(c)));
       second=(fun(a)*fun(c)*b)/(((fun(b)-fun(a))*((fun(b)-fun(c)));
       third=(fun(a)*fun(b)*c)/(((fun(c)-fun(a))*((fun(c)-fun(b));
```

```
       next = first+second+third;
      return next;
}
```

Note that fun() is the function that evaluates the function. You can write your own functions.

Example 5.16 ***Write a program to find the root of a function using Secant Method.***

Solution Newton's formula has a basic disadvantage. It involves differentiation of the function being sought. A *secant* or a *chord of a curve* is defined as a straight line that intersects the curve at least twice. This method calculates the root of a nonlinear equation using the secant. See in the above figure, the secant through $x0$ and $x1$ meets the x axis at $x2$. This will be the new approximation of the root of the curve. This process continues until we reach the desired level of accuracy. The recursive relation in the figure above gives the $n+1^{th}$ approximation of the root.

As you can probably understand from the figure above that if $x0$ and $x1$ are actually close to the actual root, the rate of convergence will be very high. Secant method is the base for *Muller's method* which tries to approximate the curve by a parabola instead of a straight line. Thus Muller method is more accurate and its convergence rate is comparable to that of Newton Raphson's method which is really fascinating.

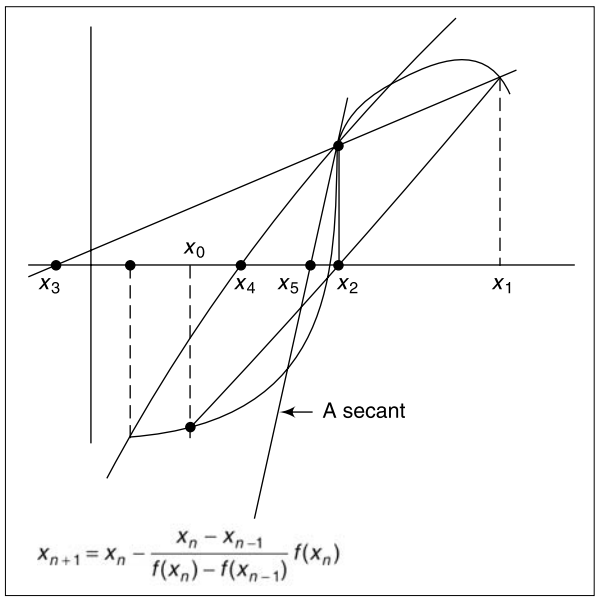


Fig. 5.7

Here is the C function that solves the secant method.

```
double SecantMethod(double xn_1, double xn,  int m)
{

      double d;
      d = (xn - xn_1)* fun(xn) / (fun(xn) - fun(xn_1)) ;
      m--;
      if(m==0)
            return xn;
      else
      {
            xn_1 = xn;
            xn = xn - d;
            d = SecantMethod (xn_1,xn,m);
        }
     return xn;
}
```

This method is highly dependent on the initial inputs.

Example 5.17 ***Write a program to find the root of an equation using Muller's Method.***

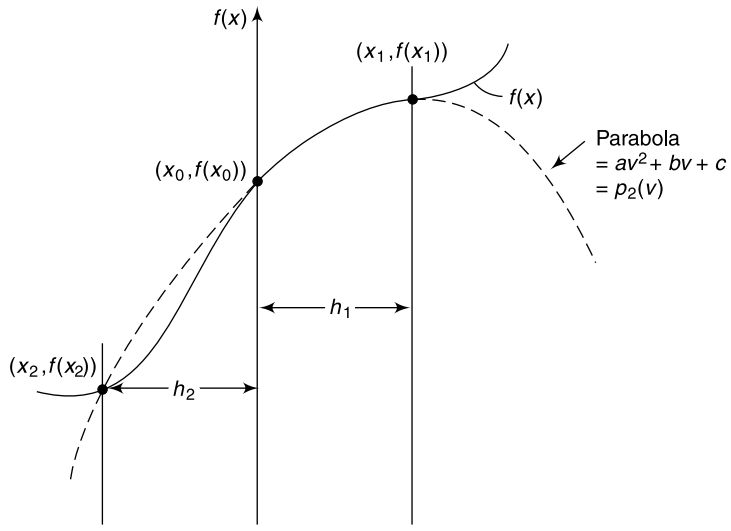


Fig. 5.8

Solution Muller's method is a close cousin of secant method where at each iteration the curve is tried to be mapped using a parabola that passes through the three points. x_k, x_{k-1} and x_{k-2}. Using Netwon's divided difference formula the curve can be approximated as the following equation below.

$$y = f(x_k) + (x - x_k) f[x_k, x_{k-1}] + (x - x_k)(x - x_{k-1}) f[x_k, x_{k-1}, x_{k-2}]$$

where $f[x_k, x_{k-1}]$ and $f[x_k, x_{k-1}, x_{k-2}]$ denote divided differences. This can be rewritten as

$$y = f(x_k) + w(x - x_k) f[x_k, x_{k-1}, x_{k-2}](x - x_k)^2$$

where

$$w = f[x_k, x_{k-1}] + f[x_k, x_{k-2}] - f[x_{k-1}, x_{k-2}]$$

The next value in the iteration process is given by the root of the quadratic equation $y = 0$. This yields the recurrence relation

$$x_{k+1} = x_k - \frac{2 f(x_k)}{\omega \pm \sqrt{\omega^2 - 4 f(x_k) f[x_k, x_{k-1}, x_{k-2}]}}$$

Note than *xk* + 1 should be one step close to the sought root than *xk*. So the denominator value for the right half should be maximum. The sign + or – is chosen depending on this logic.

Here is the C code. Here we are trying to solve the same equation as above.

```
double divdiff2(double x1,double x2)
{
     return (fun(x1)-fun(x2))/(x1-x2);
}

double divdiff3(double x0,double x1,double x2)
{
     return (divdiff2(x2,x1)-divdiff2(x1,x0))/(x2-x0);
}

double omega(double x0,double x1,double x2)
{
     return
fun(divdiff2(x2,x0))+fun(divdiff2(x2,x1))+fun(divdiff2(x1,x0));
}

double max(double a,double b)
{
     return a>b?a:b;
}

double Muller(double x0,double x1,double x2,int iter)
{
     double z=pow(omega(x0,x1,x2),2)-
               4*fun(x2)*fun(divdiff3(x2,x1,x0));
     double x_3=x2-(2*fun(x2))/(max(omega(x0,x1,x2)
               +sqrt(z),omega(x0,x1,x2)-sqrt(z)));
     iter--;
     if(iter==0)
          return x_3;
     else
     {
          x0=x1;
          x1=x2;
          x2=x_3;
          x_3=Muller(x0,x1,x2,iter);
     }
     return x_3;


}

int main()
{
     printf("Muller :%f\n",Muller(0,1,2,10));
     return 0;
}
```

Graph comparing Newton–Raphson and Muller's method.

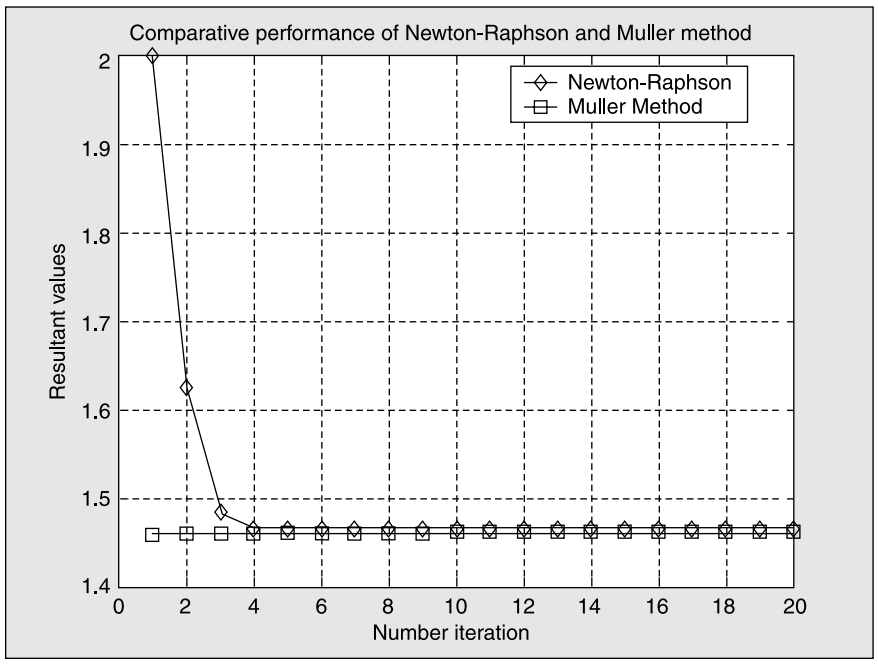


Fig. 5.9

The most fascinating fact about Muller's method is that it achieves almost the same level of convergence as that of Newton-Raphson method without the need of differentiation. So from a computation point of view, Muller's method is the best of all these methods discussed above. Muller method acquires up to 2 decimal place accuracy fairly quickly (maximum by 5 iterations). In the above figure we see that values from Newton-Raphson method are slightly more but the difference between values we get from Newton-Raphson and Muller's method become steady after 4 iterations. If we notice carefully we will see that Newton-Raphson method also took that time to stabilize. So we can conclude that Muller's method accuracy and stability is comparable with Newton-Raphson's method and as an advantage it doesn't require the knowledge of derivative of the function whose roots are being sought.

5.10 PATTERN GENERATION USING RECURSION

Example 5.18 ***Write a program to check whether a Number is happy or not.***

Solution Suppose there is a number n. If n is squared then the digits of this square number is squared separately and added. This sum of the square of the digits becomes the new number (new n). The same process is repeated again until we reach any of these numbers 1,4,16,20,37,42,58,89 or 145. If the sum is 1 then we conclude that the initial number n is happy else it is not. A number which is not happy is known as unhappy. A happy number which is also a prime number is known as happy prime like 79.

Here is the C code that checks whether a number is Happy or not.

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

int count=0;

int squareextractsum(int n)
{
      int s=0;
      int sum=0;
      if(count==1)
            s=n*n;
      else
            s = n;

      while(s!=0)
      {
            sum+=(int)pow(double(s%10),2);
            s/=10;
      }
      return sum;
}

int isHappy(int n)
{
      if(n==1)
            return 1;
     if(n==4 || n==16 || n==20
       ||n==37 || n==42 || n==58
       ||n==89 || n==145)
      return 0;

      else
      {
            n=squareextractsum(n);
            return isHappy(n);//Tail Recursive call
      }
}


int main()
{
      printf("%d \n",isHappy(230));
      return 0;
}
```

This will output 1 as 230 is a happy number.

***Try Yourself*:** Try to write a program that prints all happy numbers in a given range.

Example 5.19 ***Write a program to generate a section number pattern.***
Say there are 2 big sections in a chapter and under each big section there are 5 small sections. Write a program for printing the pattern

1.1
1.2
:
:
:
2.5

Solution Here is the C code which recursively generates the pattern:

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

int big=0,small=0;
int i=1,j=1;
int count=0;
FILE *fp;

void generateSectionNumberFile(char *file,int bg,int s)
{
      /*big = 4, small = 5
      1.1
      1.2
      1.3
      1.4
      1.5

      2.1
      2.2
      2.3
      2.4
      2.5

      and so on
      */
      fprintf(fp,"%d.%d\n",i,j);
      j++;
      if(j<=small)
            generateSectionNumberFile(file,i,j);
      if(j>small && i<big)
      {
            j=1;
            i++;
            generateSectionNumberFile(file,i,j);
      }
      if(i>big && j>small)
            fclose(fp);

}

int main()
{
```

```
    char file[20];
    printf("Number of Big sections :");
    scanf("%d",&big);
    printf("Number of sections under each Big sections :");
    scanf("%d",&small);
    printf("Enter the file name :");
    fflush(stdin);
    scanf("%s",file);
    fp = fopen(file,"a");
    generateSectionNumberFile(file,1,1);
    return 0;
}
```

As you can see, this function can be extended to any level of hierarchy. For example this method can be enhanced to generate a pattern like

1.1
  1.1.1
  1.1.2
1.2
  1.2.1
  1.2.2

The output of the above program when run with 2 and 4 as the inputs for Big and small will generate a file with pattern

1.1
1.2
1.3
1.4
2.1
2.2
2.3
2.4

5.11 HOW TO WRITE A RECURSIVE FUNCTION TO GENERATE THE NUMBERS OF THE PASCAL TRIANGLE

It will accept two numbers, one for row and the other for column and then it will display the pascal number at that position of the Pascal Triangle

The Pascal Triangle also known as Yanghui Triangle in China is

```
1
1    1
1    2    1
1    3    3    1
1    4    6    4    1
1    5    10   10   5    1
1    6    15   20   15   6    1
```

The construction of Pascal Triangle is governed by the following rule:
Each subsequent row is obtained by adding the two entries diagonally as given above.

```
int compute_pascal(int row, int position)
{
      //For matching the first column entries
      if(position == 1)
      {
            return 1;
      }
      //for matching the last column entries
      else if(position == row)
      {
            return 1;
      }
      //In between any row and any column
      else
      {
            return compute_pascal(row-1, position) +
          compute_pascal(row-1, position-1);
      }
}
```

5.12 WHAT IS THE RELATIONSHIP BETWEEN PASCAL TRIANGLE NUMBERS AND FIBONACCI NUMBERS?

The "shallow diagonals" of Pascal's triangle sum to Fibonacci numbers, i.e.

$1 = 1$
$1 = 1$
$2 = 1 + 1$
$3 = 2 + 1$
$5 = 1 + 3 + 1$
$8 = 3 + 4 + 1$

and, in general,

$$\sum_{k=0}^{[n/2]} \binom{n-k}{k} = F_{n+1}$$

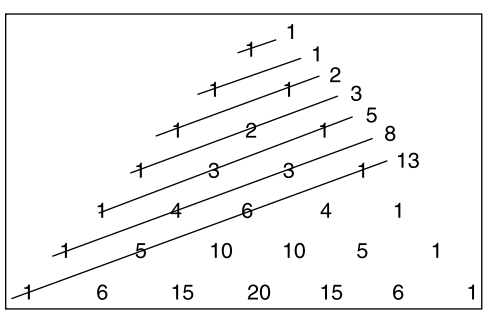


Fig. 5.10

5.13 HOW TO WRITE A RECURSIVE FUNCTION TO GENERATE THE NUMBERS OF THE BELL TRIANGLE. TO ACCEPT TWO NUMBERS, ONE FOR ROW AND THE OTHER FOR COLUMN

The Bell Triangle is the triangle formed by Bell Numbers. This triangle looks like

1				
1	2			
2	3	5		
5	7	10	15	
15	20	27	37	52

This triangle is also known as *Aitken's Array or Pierce Triangle*

This triangle is obtained by beginning the first row with the number one, and beginning subsequent rows with last number of the previous row. Rows are filled out by adding the number in the preceding column to the number above it

```
int compute_bell(int row,int position)
{
      if(row==1)
            return 1;
```

```
if(row == 2 && position ==1)
      return 1;

else
{
  if(position == 1)
            return compute_bell(row-1,row-1);
  else
           return compute_bell(row,position-1)+
                    compute_bell(row-1,position-1);
       }
}
```

5.14 APPLICATION OF BELL NUMBERS

Bell Numbers, or Bell's Numbers, are the sequence {1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147....}. The numbers count the ways that *N* distinguishable objects can be grouped into sets if no set can be empty. For example the letters ABC can be grouped into sets so that:

(1) A, B, and C are in three separate sets;
(2) A and B are together and C is separate;
(3) A and C are together and B is separate;
(4) B and C are together and A is separate;
(5) or A, B, and C are all together in a single set.

Thus when $N = 3$, there are five partitions, so the third Bell number is 5.

The Bell numbers are also the coefficients of the Maclauren expansion

$$e^{e^x} = e\left(1 + \frac{1x}{1!} + \frac{2x^2}{2!} + \frac{5x^3}{3!} + \ldots\right)$$

5.15 HOW TO WRITE A RECURSIVE FUNCTION TO GENERATE THE NUMBERS OF BERNOULLI TRIANGLE

It will accept two numbers, one for row and the other for column and then it will display the pascal number at that position of the Bernoulli Triangle.

Bernoulli triangle is a triangle containing the Bernoulli numbers. The rule to get a Bernoulli number is to sum the number above it with the number in the previous row and previous column. For example if you see the number 7 on the 4th row 3rd column in the triangle, you will notice that it is nothing but the sum of 4 and 3. 4 is right above it and 3 is in the previous column and previous row.

1					
1	2				
1	3	4			
1	4	7	8		
1	5	11	15	16	
1	6	16	26	31	32

Here is the code that generates the *Bernoulli Triangle Pattern*. First a function is written that generates the Bernoulli number at any given location and then it uses a loop to print those numbers in the pattern.

```
int compute_bernoulli(int row,int position)
{
      if(row==1)
            return 1;
      if(position==1)
            return 1;
      if(row==2 && position ==2)
            return 2;
```

```
      else
            return
   compute_bernoulli(row-1,position-1)
   +compute_bernoulli(row-1,position);
}

void display_bernoulli(int rows)
{
      for(i=1;i<=rows;i++)
      {
            for(j=1;j<=i;j++)
                  printf("%d ",compute_bernoulli(i,j));
            printf("\n");
      }
}
```

5.16 HOW TO WRITE A RECURSIVE FUNCTION TO GENERATE THE NUMBERS OF CATALAN'S TRIANGLE

It will accept two numbers; one for row and the other for column and then it will display the pascal number at that position of the Catalan's Triangle.

Catlan's Triangle is a triangle where each element is the sum of one above and one to the left. The last element of each row of Catalan's Triangle is the summation of the previous row elements. Here is the code that generates the *Catalan's Triangle Pattern*.

1						
1	1					
1	2	2				
1	3	5	5			
1	4	9	14	14		
1	5	14	28	42	42	
1	6	20	48	90	132	132

First 7 rows of Catalan's Triangle are

```
void display_catalan(int rows)
{
      for(i=1;i<=rows;i++)
      {
            for(j=1;j<=i;j++)
                  printf("%d ",compute_catalan(i,j));
            printf("\n");
      }
}

int compute_catalan(int row,int position)
{
      if(row==1)
            return 1;
      if(position==1)
            return 1;
      if(row==2 && position==2)
            return 1;
      else
      {
            if(row==position)//Last element
                  return catalan_sum(row-1);
            else
                  return compute_catalan(row-
1,position)+compute_catalan(row,position-1);
      }
}
```

```
int catalan_sum(int r)
{
      int i=1;
      int sum=0;
      for(i=1;i<=r;i++)
            sum+=compute_catalan(r,i);
      return sum;
}
```

5.17 WHAT IS THE RECURSIVE RELATION THAT GENERATES A CATALAN NUMBER?

Catalan Number follows a recursive relation among themselves as follows:

$$C_{n+1} = (4n + 2)C_n / (n + 2) \text{ and C0} = 1 \text{ is the initial value.}$$

where C_{n+1} is the n+1th Catalan Number.

Catalan number is also closely related with binomial coefficient as

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} \quad \text{for } n \geq 0$$

Here is the C Program that will generate n^{th} Catalan Number:

```
int Catalan(int n)
{
      return nCk(2*n,n)/(n+1);
}
```

5.18 SOLVING EULER'S POLYGON DIVISION USING A CATALAN NUMBER

Catalan Number was generated from a problem known as 'Euler's Polygon division problem', that is of finding in how many ways E_n a plane convex polygon of n sides can be divided into triangles by diagonals. Euler first proposed it to Christian Goldbach in 1751, and the solution is the Catalan number $E_n = C_{n-2}$.

This above solution comes from a recurrence relation known as 'Segner's recurrence relation':

$$En = E2.En - 1 + E3.En - 2 + \ldots + En - 1.E2$$

5.19 DYCK PATH AND CATALAN NUMBER

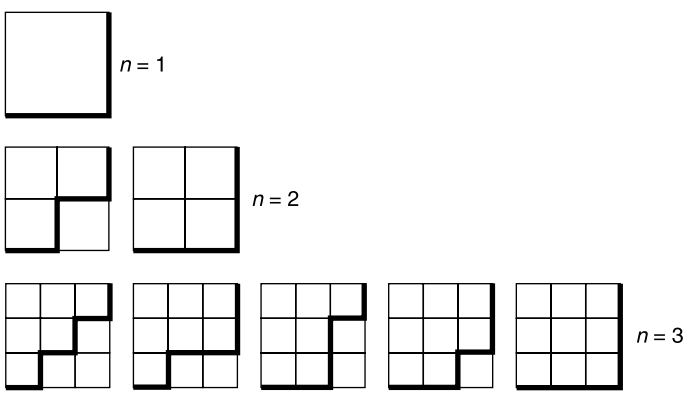


Fig. 5.11

A *Dyck Path* is a staircase walk from (0, 0) to (n, n) that lies strictly below the reverse diagonal which is inclined at 45 degree to the horizon. The number of Dyck paths of order n is given by n^{th} Catalan Number. See the above image to understand the concept better.

The first box in the above image corresponds to a staircase where there is only one stair. So the number of Dyck path is 1 for $n = 2$ it is the second catalan number 2, and for 3 it is the third Catalan's Number = 5, and so on.

5.20 BALLOT PROBLEM AND CATALAN NUMBER

Suppose A and B are candidates for office and there are 2n voters, n voting for A and n for B. In how many ways can the ballots be counted so that B is never ahead of A? The solution is a Catalan number C_n.

Please note using the function Catalan() defined above we can solve these type of problems. Catalan numbers find their usage in several combinatorial problems.

5.21 HOW TO WRITE A RECURSIVE FUNCTION TO GENERATE THE NUMBERS OF LOSANITSCH'S TRIANGLE

It will accept two numbers; one for row and the other for column and then it will display the pascal number at that position of the Losanitsch's Triangle.

Losanitsch's triangle is a number triangle for which each term is the sum of the two numbers immediately above it, except that, numbering the rows by $n = 0, 1, 2, \ldots$ and the entries in each row by $k = 0, 1, 2, \ldots n$.

Here is a code that generates Losanitsch's Triangle:

```
int compute_Losanitsch(int row,int position)
{
     if(row==1)
           return 1;
     if(position==1)
           return 1;
     if(row==position)
           return 1;
     if(row==3 && position==2)
           return 1;
     if(row ==4 && (position==2 || position == 3))
           return 2;
     if(row==5 && (position==2 || position==4))
           return 2;
     if(row==5 && position==3)
           return 4;
     else
     {
           if(row%2==0 && (position==2 || position==row-1))
                 return                          compute_Losanitsch(row-
1,position)+compute_Losanitsch(row-1,position-1);
           if(row%2!=0 && (position==2 || position==row-1))
                 return compute_Losanitsch(row-1,position);
           else
                 return                          compute_Losanitsch(row-
1,position)+compute_Losanitsch(row-1,position-1);

     }

}
```

```
void display_losanitsch(int rows)
{
     for(i=1;i<=rows;i++)
     {
           for(j=1;j<=i;j++)
                 printf("%d ",compute_Losanitsch(i,j));
           printf("\n");
     }
}
```

Here is how the output looks:

1										
1	1									
1	1	1								
1	2	2	1							
1	2	4	2	1						
1	3	6	6	3	1					
1	3	9	12	9	1	1				
1	4	12	21	21	10	2	1			
1	4	16	33	42	31	12	1	1		
1	5	20	49	75	73	43	13	2	1	
1	5	25	69	124	148	116	56	15	1	1
1	6	30	94	193	272	264	172	71	16	2	1

Losanitsch's triangle is given such a name after Serbian chemist *Sima Lozanic* who researched it in his investigation into the symmetries exhibited by rows of paraffins.

5.22 HOW TO WRITE A RECURSIVE FUNCTION TO GENERATE THE NUMBERS OF THE LEIBNITZ HARMONIC TRIANGLE

It will accept two numbers, one for row and the other for column and then it will display the Pascal number at that position of the Leibnitz Harmonic Triangle
This triangle below is *Leibnitz Harmonic Triangle*.

$\frac{1}{1}$				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{3}$	$\frac{1}{6}$	$\frac{1}{3}$		
$\frac{1}{4}$	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{1}{4}$	
$\frac{1}{5}$	$\frac{1}{20}$	$\frac{1}{30}$	$\frac{1}{20}$	$\frac{1}{5}$

In this triangle each fraction is the sum of numbers below it and the initial and final entries in the n^{th} row are given by $1/n$.

The terms are given by the recurrences

$$a_{n,1} = \frac{1}{n}$$

$$a_{n,k} = a_{n-1,k-1} - a_{n,k-1}$$

Here is the code to generate numbers of this triangle.

```
double compute_Leibnitz(int row,int position)
{
      if(position==1)
            return (double)1/(double)row;
      if(row==2 && (position==1 || position==2))
            return 0.5;
      else
      {
            return compute_Leibnitz(row-1,position-1)
                  -compute_Leibnitz(row,position-1);
      }
}
```

This function calculates Leibnitz Harmonic Number at row and position.

Example 5.20 ***Write a program to find the Entringer numbers using recursion.***

Solution

$$E(n,k) = E(n,k-1) + E(n-1,n-k)$$

This relation gives the number of

Where, E(0,0) = 1 and E(n,0) = 0

Here is the C code to generate the Entringer numbers.

```
int compute_Entringer(int n,int k)
{
      if(n==0 && k==0)
            return 1;
      if(k==0)
            return 0;
      else
       return compute_Entringer(n,k-1)
             +compute_Entringer(n-1,n-k);
}
```

Example 5.21 ***Write a program to display the numbers of Hofstadter Conway $10,000 Sequence.***

Solution *Hofstadter Conway* discovered a sequence which is recursive in nature and is given by the following equation:

A(n) = A(A(n-1)) + A(n-A(n-2))

Where A(1) = A(2) = 1

So the first few values are 1,1,2,2,3,4,4,4,5,6 etc

Here is the C code that recursively generates the numbers of Hofstadter Conway Sequence..

```
int compute_HConway(int n)
{
      if(n==1 || n==2)
            return 1;
      else
            //Tail Recursive Call
            return compute_HConway(compute_HConway(n-1))
                  +compute_HConway(n-compute_HConway(n-1));
}

void display_HConway(int n)
{
```

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

Write functions to generate and display the numbers of the Mallows Sequences

```
int compute_Mallows(int n)
{
      if(n==1 || n==2)
            return 1;
      else
            return compute_Mallows(compute_Mallows(n-2))
                  +compute_Mallows(n-compute_Mallows(n-2));
}

void display_Mallows(int n)
{
      for(i=1;i<=n;i++)
            printf("%d \n",compute_Mallows(i));
}
```

Example 5.22 ***Write a program to generate Hofstadter's Q Sequence. These numbers are also known as Q- Numbers.***

Solution This is a recursive sequence and generated by the recurrence equation

`Q(n) = Q(n-Q(n-1)) + Q(n-Q(n-2))`

with Q(1) = Q(2) = 1 . The first few values are 1, 1, 2, 3, 3, 4, 5, 5, 6, 6, .

Here is the C code to generate and display the Q-number

```
int computeQ(int n)
{
      if(n==1 || n==2)
            return 1;
      else
      {
            return computeQ(n-computeQ(n-1))
                   +computeQ(n-computeQ(n-2));
      }
}

void display_Q(int n)
{
      for(i=1;i<=n;i++)
            printf("%d \n",computeQ(i));
}
```

Example 5.23 ***Write a program to generate Serpinski Fractal Pattern.***
Use the fact that Serpinski Triangle Pattern can be generated by using Pascal's Triangle. In place of each even number in Pascal's triangle put 1 and in place of each odd number put 0 to generate the Serpinski Triangle Pattern. Here is the code and output.

Solution

```
void display_serpinski(int rows)
{
      int n=0;
      int m=0;
```

```
for(i=1;i<=rows;i++)
{
      for(m=0;m<(51/2)-i;m++)
            printf(" ");

      for(j=1;j<=i;j++)
      {
            n = compute_pascal(i,j);
            if(n%2==0)
            {
                  printf("%c ",'1');
            }
            else
            {
                  printf("%c ",'0');
            }
      }
      printf("\n");
   }
}
```

Note that we have used compute_pascal( ) function described above.
This program generates a pattern like

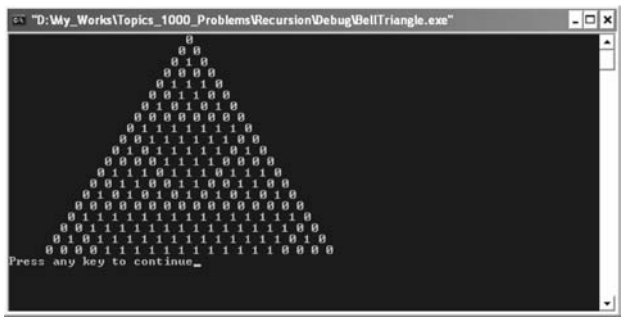


Fig. 5.12

An image of the fractal Serpinski Triangle looks like

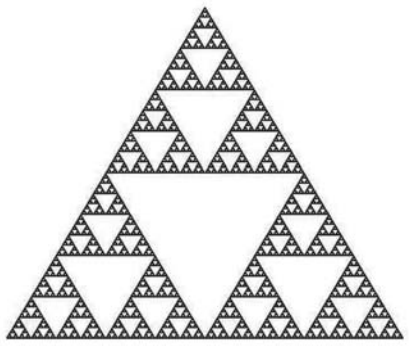


Fig. 5.13

Can you find similarity between the generated pattern and the fractal image?

5.23 L-SYSTEM, RECURSION AND MORE FRACTALS

L-System is a system developed by *Aristid Lindenmayer* in 1968 to describe the growth of self similar plants like fern etc. L-System is a formal grammar and recursive in nature. Thus, nowadays L-System is used to generate many recursive fractals like Koch curve, Koch snowflakes, etc.

A self similar structure can be modeled using an L-System. Every system that is being designed using an L-System will be needed to have the following.

Alphabet Which is the building block of the L-System representation of the fractal

Rules Which makes the fractal to grow or diminish

Constants Something that remains constant throughout the process

Initial Conditions What are the initial conditions of the process?

Special Conditions (this is optional) Is there any special condition that can modify a rule?

Lindmeyer originally tried to model the growth of an algae using the L-System as follows:

Alphabets: A B
Constants: none
Initial Condition: A
Rules: (A → AB), (B → A)
Special Condition: None

which produces:

n = 0: A
n = 1: AB
n = 2: ABA
n = 3: ABAAB
n = 4: ABAABABA

A → AB means A will be replaced by AB and B → A means B will be replaced with A.

If you notice carefully you will find that this algee growth model discussed using L-System can be easily modeled using recursion. All we have to do is to pass the iterated string to the system.

```
char* algee(char start[10])
{
      char temp[10];

      for(int i=0,j=0;i<strlen(start);i++)
      {
            if(start[i]=='A')
            {
            temp[j]='A';
            j++;
            temp[j] = 'B';
            j++;
      }
      if(start[i]=='B')
      {
            temp[j] = 'A';
```

```
                j++;
            }
      }
      if(strcmpi(start,"ABAABABA")==0)
            //The function is Tail-Recursive
            return algee(start);
 }
int main()
{
      puts(algee("A"));
      getch();
      return 0;
}
```

Many recursive fractals or some of their variations can be generated using this L-System.

5.24 FRACTAL GENERATION USING RECURSION

Fractals have structural self-similarity on multiple scales. That means a piece of a fractal will often look like the whole part. Fractals are used in computational geometry for rendering realistic looking scenery for computer games. Next to this section, an algorithm is shown how to generate terrains using fractal algorithm.

5.25 KOCH CURVE

Generation of Koch Curve

- A Straight line is transformed
  - The line is split into three equal sizes
  - Middle part is removed
  - Middle is replaced with two lines of the same length (1/3 of original line)
- For each straight line in the transformed line, repeat the process.
- At some point the changes are no longer visible (if we keep our viewing scale constant)

The length of the curve at depth *d* is (4/3)^*d* as shown in the figure.

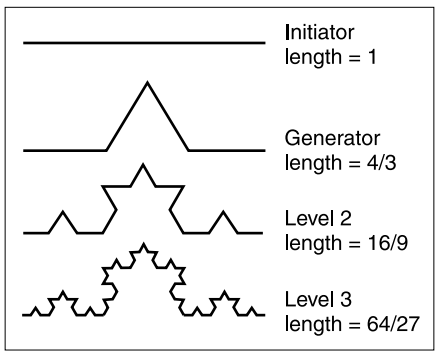


Fig. 5.14

5.26 KOCH SNOWFLAKE

Koch Snowflake is a fractal that starts from an equilateral triangle. Then each arm of the equilateral triangle is subdivided into three sections as shown in the second image above. This process continues as long as there is no perceptible change occurs between two consecutive structures, as long as we want. The more we break the lines, the structure will have more details.

This is a recursive process to construct such a structure. As the structure looks like a snowflake, that's why it is called *Koch Snowflake* in honor of the mathematician *Helge Von Koch* who first described this pattern in 1904.

Inspired by the beauty of these fractal structures they are now used in ornaments. Fractal ornaments are getting popularity these days.

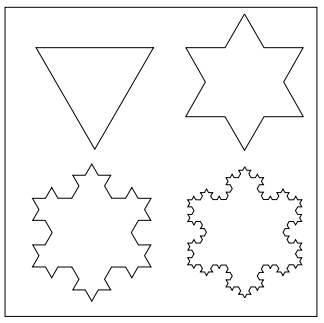


Fig. 5.15

5.27 RECURSION IN NATURAL SCENE GENERATION

Terrain Generation using Recursion

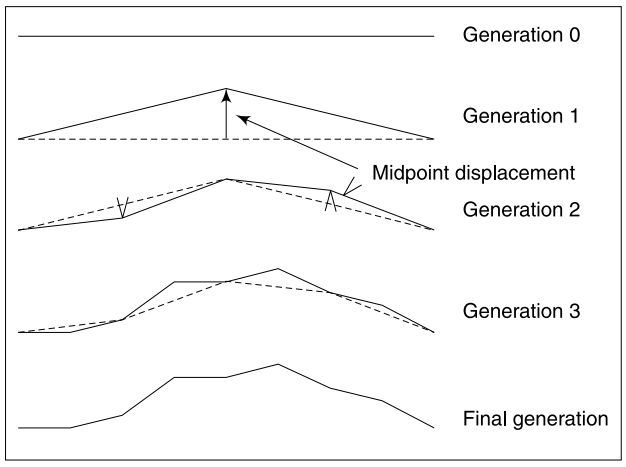


Fig. 5.16

One dimensional mid-point displacement is a great algorithm for drawing a ridgeline, as mountains might appear on a distant horizon. Here is how it works.

Start with a straight line as shown in the figure above

Repeat for a sufficiently large number of times

```
{
   Repeat over each line segment in the scene
   {
         Find the mid point of the line segment
         Displace the midpoint in Y by a random amount
         Reduce the range for random numbers,
   }
}
```

How much we want to reduce the range for random numbers depends on how rough we want the mountain to look like. The more we reduce it in each pass through the loop, the smoother the resulting ridgeline will be.

Note two things about the above algorithm. First, it is recursive in nature; and secondly, it can create fairly complex image with very low level details in few simple steps.

The realization that a small, simple set of instructions can create a complex image has lead to research in a new field known as *fractal image compression*. The idea is to store the simple, recursive instructions for creating the image rather than storing the image itself. This works great for images which are truly fractal in nature, since the instructions take up much

less space than the image itself. *Chaos and Fractals, New Frontiers of Science* [3] has a chapter and an appendix devoted to this topic and is a great read for any fractal nut in general.

Without much effort, you can read the output of this function into a paint program and come up with something like this:

REVISION OF CONCEPTS

Some Key Facts about Recursion

- Recursion means re-occurring of the same function or method.
- Any function can call any function.
- Some people think that main() can't be called from any other function. That is a misconception. Any function can be called from any function. As because main() is a function so it can be called from any function.
- When a function calls another function and the called function in turn calls the calling function, which is called Mutual Recursion. For example, say there is a function called Menu() that is being called from the main() function. And from Menu() again, main () is being called. Then we can say that main() and Menu() are mutually recursive.
- When a function calls itself then that phenomenon is called as Self Recursion. And when we say recursion we mean Self Recursion unless otherwise mentioned.
- Recursion can be used to replace loops. That's why it is sometimes called as virtual loop.
- Whenever recursion is used the exit, criterion should be there.
- Recursion is a good technique to generate number sequences.
- Recursion is also used to generate some recursive Fractal Pattern like Sierpinski triangle, which can be used to model chaotic objects like clouds, etc.
- An unusual use of recursion in a definition comes from Professor *Seymour Papert, Professor of Media Technology, at MIT* (The Massachusetts Institute of Technology) and the inventor of the graphical programming language LOGO. Here are Papert's instructions on how to make a circle. First, take a step forward, then turn a little to the right, then make a circle. His description is a very unusual one because it describes a circle as a *process* rather than as a static geometric shape. His description is recursive because making a circle is defined in terms of making a circle. Recursion is often used to define functions.
- Recursion is used to generate numbers sequences.
- Recursion is used to generate batrachions (A special class of curves defined only for integer sequence) Example includes Q – number sequence, mallows sequence.

REVIEW QUESTIONS

1. What is n^{th} Ackermann's number? Where n is the product of two consecutive primes starting with 1,2.
2. What is Hilbert Curve. What are the steps to create one recursively.

3. What is the L-System representation of Koch Snowflake?
4. What is the L-System representation of Serpinsky Triangle?
5. What is the L-System representation of Monge Sponge?
6. What is the L-System representation of Koch Curve?

PROGRAMMING PROBLEMS

1. Write a program to check whether a number is prime or not.
2. Write a program to generate prime numbers within a given range.
3. Write a program to generate al the combinations of a word.
4. Write a general algorithm to re-write a recursive algorithm in a non-recursive way.
5. Write a Program to generate Delannoy numbers given by the recurrence relation $D(a, b) = D(a - 1, b) + D(a, b - 1) + D(a - 1, b - 1)$ with initial value $D(0, 0) = 1$.
6. Write a program to generate random numbers using primitive polynomials.
7. Write a program to generate random numbers using the famous blum-blum-shub recursive algorithm.
8. Write a program to solve a non-linear equation using Brent's method.
9. Rewrite the function to solve the root of a non-linear equation using Newton Raphson method. Pass the tolerance level instead of the number of iteration.
10. Rewrite the function to solve the root of a non-linear equation using Bisection method. Pass the tolerance level instead of the number of iteration.
11. Rewrite the function to solve the root of a non-linear equation using Regula-Falsi method. Pass the tolerance level instead of the number of iteration.
12. Rewrite the function to solve the root of a non-linear equation using Secant method. Pass the tolerance level instead of the number of iteration.
13. Rewrite the function to solve the root of a non-linear equation using Muller's method. Pass the tolerance level instead of the number of iteration.
14. Write a program to print all the anagrams of a given string.
15. Write a program to find out whether there is a path between two locations in a maze or not.
16. Write a program to find whether there is a path between two nodes in a maze or not.
17. Write a program to find the root of a non-linear equation using Brent's method.
18. Write a program to find the highest Fibonacci number within the range INT_MAX.
19. Compare a recursive relation with Ackerman's function.
20. Compare a recursive relation with Tak function.
21. Demonstrate how LOGO is recursive, specifically speaking tail recursive in nature.
22. Show how Koch Curve can be generated using recursion.
23. Show how Serpinsky's triangle can be generated using recursion.

6

Stack
One upon Another

INTRODUCTION

Some real-life scenarios can be modeled easily with a first in, last out list. In these types of situations a stack will be the ideal data structure. A stack can be designed either by using arrays or by using linked list. The basic operations possible on a stack are popularly known as *push* and *pop*. All these operations are described in this chapter. Stack is a simple data structure but it has tremendous usages in the software industry. Starting from a simple postfix calculator to a complicated XML reader, stacks find their way. All these diverse applications of this simple data structure have been discussed at length in this chapter.

6.1 MODEL A STACK AS A STRUCT

Stack organizes elements one above another. So to model a stack in C we need two variables. We can use arrays to hold the values of the stack. The other variable will hold the number of elements currently in the stack. So, we can use a structure to represent stack. Given below is such a structure that represents an integer stack that can hold 10 integers.

```
#define MAX 10

typedef struct MyStack
{
      int data[MAX];
      int count;
}MyStack;
```

Here data [MAX] is an integer array that can hold 10 integers in the stack. *Count* is the variable that holds the current number of elements in the stack. *Typedef* has been used so that from now on MyStack can be used as a built-in data type.

6.2 HOW TO INITIALIZE THE STACK MODELED ABOVE

Here we use arrays internally to store the Stack. Since array index starts from zero, to indicate an empty stack, the count variable of the stack is set to –1. After push or pop operation, the count value is modified, and the stack is reinitialized before any further operation. A global variable is kept and that is incremented by unity while pushing, and decreased by unity while popping elements to and from the stack.

Here the global variable is named as *sc*. Here is the code to initialize the stack.

```
int SC = -1;
void initMyStack(MyStack *st)
{
      st->count=sc;
}
```

By pushing we mean adding an element to the top of the stack. Until count reaches the maximum number that the stack can hold, the elements added will be appended at the top of the stack. Pointer to structure will be used to push an element. Here is the code for the method `push()`. Assume that we are operating on the stack defined above. Here is the figure that explains the push and pop operation associated with a stack.

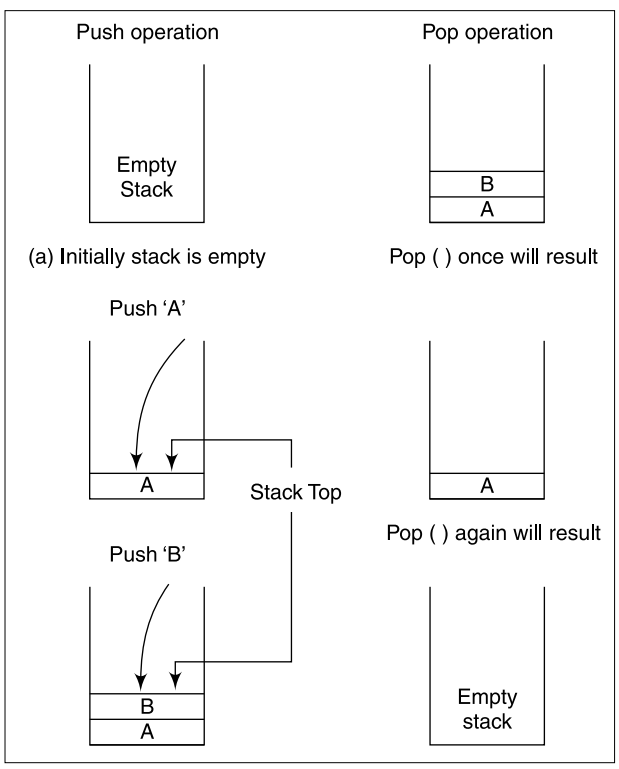


Fig. 6.1

```
int push(MyStack *st)
{
      if(st->count==MAX-1)
```

```
{
      printf("MyStack is full!\n");
      return 0;
}
else
{
      printf("Enter the number :");
      scanf("%d",&st->data[st->count]);
            sc++;
            printf("Successfully pushed");
            return 1;
      }
}
```

Since the counting of the elements for the stack starts from –1 (Empty Stack), the total number of elements in the stack when the stack is full is given by MAX – 1. This method returns 1 in case the pushing of elements is successful, otherwise it returns 0. In case, the stack is not full, the element entered is put at the last of the array in the location `st->count`.

Here the method itself asks for the element to be added at the top of the stack. But we can pass the element to the method.

6.3 HOW TO POP THE MRA ELEMENT FROM THE ABOVE STACK

Deleting the last element from the stack is popularly known as *popping* of *Most Recently Added*(MRA) element. If the stack is empty and we want to pop the MRA element, then the system will display the message "Stack is empty, Underflow Error". Otherwise it will pop the last element. from the Stack. Here is the code to pop the last element from the stack defined above.

```
int pop(MyStack *st)
{

      if(st->count<0)
            printf("MyStack is Empty!\nUnderflow Error");
      else
      {
            printf("Last number is popped\n");
            sc--;
      }
      return st->data[st->count-1];
}
```

This method returns the popped element to the calling method.

6.4 HOW TO DISPLAY THE STACKTOP ELEMENT

The element at the stack top is of interest. To display the element at the top of the stack, we have to display the last element of the array. Here is the code to display the stack top using the stack defined above.

```
void display_MyStack_top(MyStack *st)
{
      if(st->count<0)
            printf("MyStack Empty\n");
```

```
      else
            printf("Top = %d\n",st->data[(st->count)-1]);
}
```

Now it checks whether the Stack is empty or not. If the Stack is not empty, then it displays the last element of the array.

6.5 HOW TO SWAP THE TOP TWO ELEMENTS

Sometimes you may need to swap the top two elements of the stack. Here is the code to swap the last two elements from our stack.

```
void swap(MyStack *st)
{
      if(st->count<1)
            printf("Swapping can't be done\n");
      else
      {
            int temp=st->data[st->count-1];
            st->data[st->count-1] = st->data[st->count-2];
            st->data[st->count-2] = temp;
            printf("Swapped Successfully!\n");
      }
}
```

If there is only one element, then the count is 0, and swapping is not possible.

6.6 PUTTING IT ALL TOGETHER USING ARRAYS

Here is the source code putting all these methods together.

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

#define MAX 10

typedef struct MyStack
{
      int data[MAX];
      int count;

}MyStack;

MyStack ms;
MyStack *s = &ms;
int sc=-1;
void initMyStack(MyStack *);
int push(MyStack *);
void display_MyStack_top(MyStack *);
void display_MyStack(MyStack *);
int pop(MyStack *);
void swap(MyStack *);
```

```
//Copy and paste each method's code here that are listed above

void initMyStack(MyStack *st)
{
      st->count=sc;
}

//Now call these methods from within main as below
int main()
{
      int choice;
      do
      {
            initMyStack(s);
            printf("\n1.Push\n");
            printf("2.Pop\n");
            printf("3.MyStack Top\n");
            printf("4.Is Empty\n");
            printf("5.Swap\n");
            printf("6.Exit\n");
            printf("Choice [1-6] :");
            scanf("%d",&choice);
            switch(choice)
            {
            case 1:push(s);break;
            case 2:pop(s);break;
            case 3:display_MyStack_top(s);break;
      case 4:s->count<0?printf("Stack is empty"):printf("Stack is not
empty");break;
            case 5:swap(s);break;
            case 6:exit(0);
            }
      }while(1);
      return 0;

}
```

6.7 MODEL A STACK USING A LINKED LIST

The main limitation of using the array to build stack is that we have to fix the size of the array in design time. If we use linked list to design our stack, then the size of the stack can increase or decrease in the design time. So in real time applications whenever we need to use stack, we will model that using a linked list. Here we have modeled an integer stack using a linked list.

```
typedef struct link
{
       int info;
       link *next;
}link;
```

This structure above represents a particular node of the linked implementation of the stack.

6.8 PUSHING AN ELEMENT

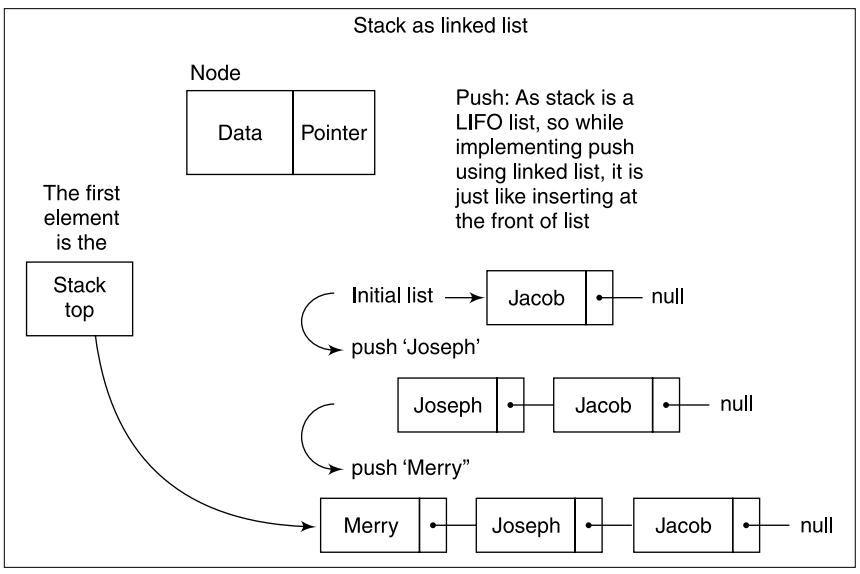


Fig. 6.2

When we represent the stack using a linked list, pushing an element onto the top of the stack involves two activities. First of all, we have to create enough space using `malloc()` method to insert one more node in the list. Then the "next address" of the new node is assigned with the previous last node. This is done because stack is a LIFO list, so MRA element will be at the top. Here is the code of the function that pushes an element on to the stack top.

```
Stack * push(Stack *rec)
{
      Stack *new_rec;
      printf("Enter the value to push :");
      new_rec = (Stack *)malloc(sizeof(Stack));
      scanf("%d",&new_rec->info);
      new_rec->next=rec;
      rec = new_rec;//The old is now new
      return rec;
}
```

This method returns the pointer to the last node. malloc() returns a void * . So it needs to be type casted to the Stack * type.

6.9 HOW TO POP AN ELEMENT FROM THE STACK

Popping from a stack that is modeled using linked list requires the memory segment to be freed manually. The method `free()` is used for deleting the MRA element. Here is the code for popping the top element from a stack designed with a linked list.

```
Stack * pop(Stack *rec)
{
      Stack *temp;
```

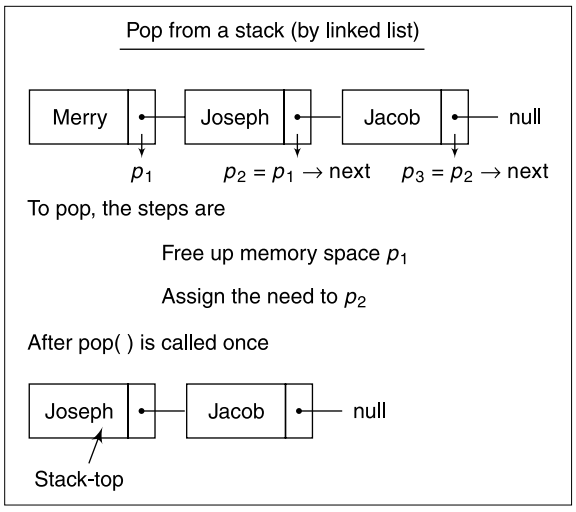


Fig. 6.3

```
      if(rec==NULL)
            printf("Stack is Empty");
      else
      {
            temp=rec->next;
            free(rec);//Freeing up memory space
            rec = temp;
      }
      return rec;
}
```

From this method also the pointer to the top element is returned.

6.10 HOW TO PEEP AT THE STACK TOP

Peeping at the top element of the stack requires the least amount of code. We just need to check whether the stack is empty or not. If it is not empty, then the MRA element is shown. Here is the method.

```
void peep(Stack *rec)
{
      if(rec==NULL)
            printf("Stack is empty.");
      else
            printf("Stack Top Element = %d",rec->info);

}
```

6.11 HOW TO SWAP THE TOP TWO ELEMENTS

Swapping the top two elements involve swapping the values of the nodes. Here is the code for swapping the top two elements of the stack.

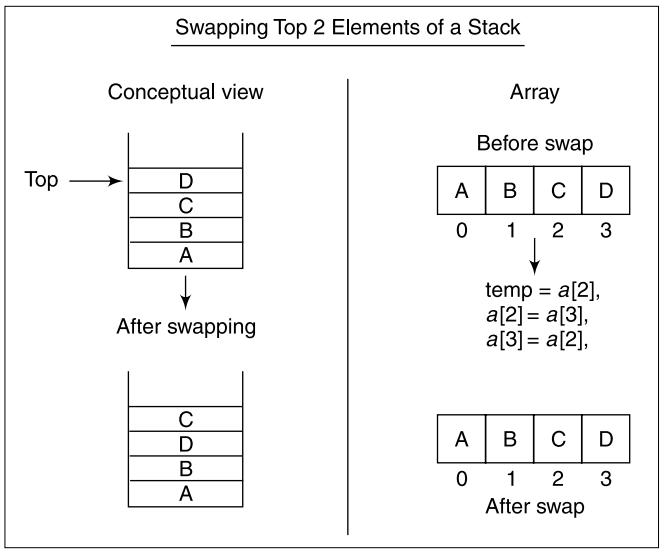


Fig. 6.4

```
Stack * swap(Stack *rec)
{
      int temp;
      if(rec->next==NULL||rec==NULL)
            printf("Swapping can't be done. Too few elements");

      else
      {
          //Swapping the values of the top 2 nodes
            temp = (rec->next)->info;
            (rec->next)->info = rec->info;
            rec->info = temp;
      }
      return rec;//returning the pointer to the top element

}
```

Example 6.1 ***Write a program using stack to evaluate a postfix expression. Assume that the postfix expression can only take +,-,*,/ and all the parameters are from 0 to 9. The program will work as follows. A postfix expression will be entered from the console by the user and then the program will calculate the value of the expression and displays that on console. But the program doesn't check whether the supplied postfix expression is valid or not. Model the stack using linked list.***

Algebraic expressions such as (A+B)/(C–D) have an inherent tree-like structure. For example, the figure below is a representation of the expression in equation (A+B)/(C–D). This kind of tree is called an expression tree.

If we travel the tree in-order we get infix notation. If we travel the tree in post-order then we get the postfix expression. Verify that the expression tree values and the postfix expression match below.

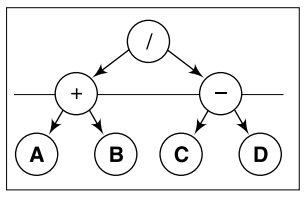


Fig. 6.5

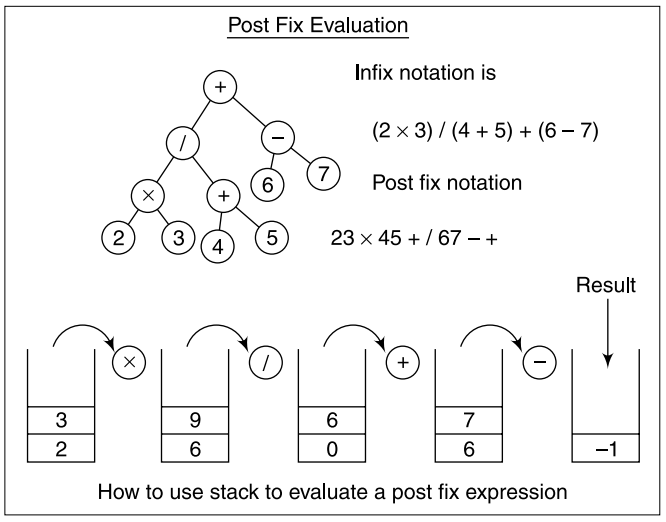


Fig. 6.6

Solution

```
//This shows how to use a Stack to evaluate the given postfix
//expression.

#include <stdio.h>
#include <conio.h>

#include <malloc.h>
#include <ctype.h>


typedef struct Stack
{
    int info;
    struct Stack *next;
}Stack;

Stack *start;

Stack* push(Stack *,int);
Stack* pop(Stack *);
int peep(Stack *);
```

```
int peep(Stack *rec)
{
    if(rec==NULL)
             printf("Stack is empty.");
    else
             return rec->info;
}
```

```
Stack * push(Stack *rec,int c)
{
          Stack *new_rec;
          new_rec = (Stack *)malloc(sizeof(Stack));
          new_rec->info=c;
          new_rec->next=rec;
          rec = new_rec;
          return rec;
}


Stack * pop(Stack *rec)
{
         Stack *temp;
          if(rec==NULL)
                printf("Stack is Empty");
          else
          {
                    temp=rec->next;
                    free(rec);
                    rec = temp;
             }
                 return rec;

}
//This method checks whether the character read is an Operator or
//not. These are the basic 4 mathematical operators. You can write
int isop(char o)
{
    if(o=='+'||o=='-'||o=='*'||o=='/')
       return 1;
    else
       return 0;
}

int main()
{
         char *postfix;
         int first=0;
         int second=0;
         int i=0;
         int result=0;
         int temp=0;
         clrscr();
 printf("Please enter a Postfix Expression :");
 scanf("%s",postfix);
 for(i=0;i<strlen(postfix);i++)
{
          if(!isop(postfix[i]))
         {
                  printf("Enter value of %c",postfix[i]);
```

```
                scanf("%d",&temp);
                start = push(start,temp);
         }
          else
        {
                second = peep(start);
                start = pop(start);
                first = peep(start);
                 start = pop(start);

                if(postfix[i]=='+')
                 {

                             result=0;
                             result+=(first+second);
                             start = push(start,result);
                  }

                 if(postfix[i]=='-')
                {
                             result=0;
                             result+=(first-second);
                             start = push(start,result);
                 }

                 if(postfix[i]=='*')
                  {
                           result=0;
                           result+=(first*second);
                           start = push(start,result);
                  }

                 if(postfix[i]=='/')
                  {
                            result=0;
                            result+=(first/second);
                            start = push(start,result);
                  }
                  //Add your own custom defined operators
                  //Like ^ for exponentiation and so on

             }
}
   printf("Result is = %d",peep(start));
    getch();
       return 0;
}
```

A sample run of the program.

```
Please enter a Postfix Expression :ab+cd-/
Enter value of a 80
Enter value of b 20
```

Enter value of c 5
Enter value of d 1
Result is = 25

6.12 HOW TO WRITE A PARENTHESIS MATCHER USING STACK

Compilers make use of stack structures. This example shows a simple illustration of the way a stack can be used to check the syntax of a program. It uses the stack class you have created. In the example, a stack is used to check that the braces {and} used to mark out code

- The source file is read character by character.
- Each time a { is read an object (any object, it doesn't matter what) is pushed onto the stack.
- Each time a } is read an object is popped from the stack.
- If the stack is empty when a } is read then there must be a missing { .
- If the stack is not empty at the end of the file then there must be a missing } .

Here is the Pseudo code for this. Assume that we are using the predefined MyStack structure.

```
printf("Enter the File Name :");
 fflush(stdin);
 gets(file);
 printf("%s",file);
 FILE *fp = fopen("C:\\tezt.c","r");
 while(!feof(fp))
 {
       c = fgetc(fp);//reading character by character
       printf("%c",c);
       if(c=='{')
             push(s);
       if(c=='}')
       {
             if(isEmpty(s))

                   printf("Missing {\n");
                   pop(s);
             }
       }
 }
 if(!isEmpty(s))//We have reached the end of file
       printf("} Missing\n");
```

6.13 SWITCHBOX ROUTING PROBLEM

Example 6.2 ***Write a program that accepts the pin numbers of a switchbox or IC and then states whether the pin numbers are routable or not. A switchbox is routable by that we mean that there will be no cross connection if we connect the given pin numbers. (See Fig. 6.7.)***

Solution As you can see in the figure below, the internal connection inside a IC/PCB/switchbox is similar. Suppose we have to connect pin number 1 to pin number 4 and like that, but no overlapping is allowed because that will cause short-circuit.

A stack can be used to find out whether a switchbox is routable or not as shown in the following figure.

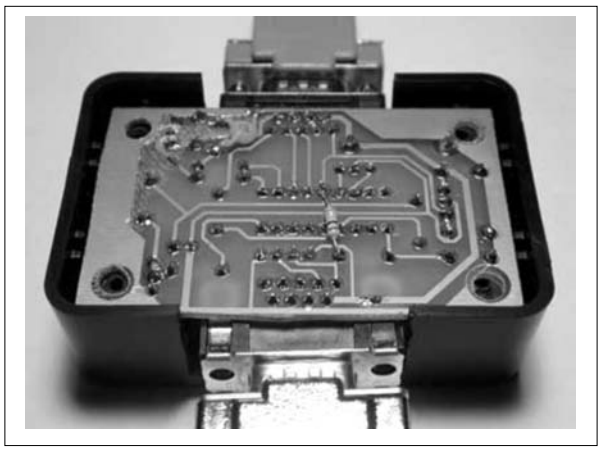


Fig. 6.7

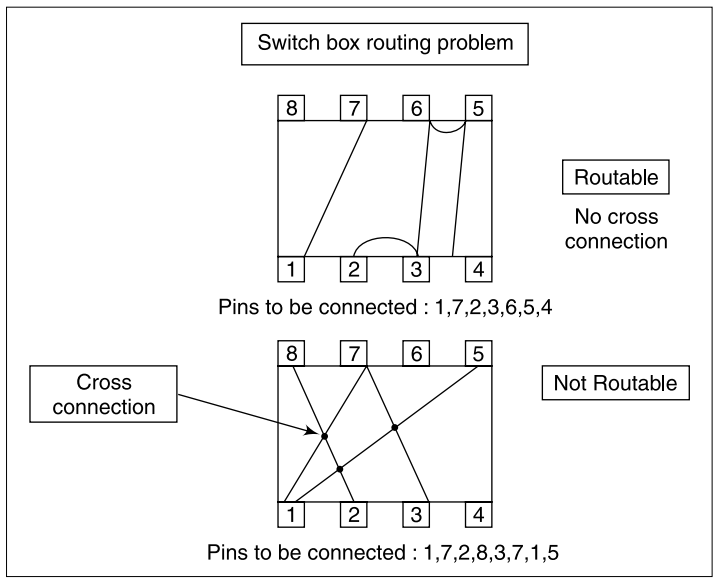


Fig. 6.8

Here is the code in C to solve the problem using stack.

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <ctype.h>
#include <stdlib.h>


typedef struct Stack
{
      int info;
      struct Stack *next;
}Stack;
```

```
Stack *start;


void display(Stack *);
Stack* push(Stack *);
Stack* pop(Stack *);
int peep(Stack *);
Stack* swap(Stack *);
```

```
Stack * swap(Stack *rec)
{
      int temp;
      if(rec->next==NULL||rec==NULL)
            printf("Swapping can't be done. Too few elements");

      else
      {
            temp = (rec->next)->info;
            (rec->next)->info  = rec->info;
            rec->info = temp;
      }
      return rec;

}
void display(Stack *rec)
{
      if(rec==NULL)
            printf("Stack is empty.");
      else
      {
            while(rec!=NULL)
            {
                  printf("Info = %d Address : %u Next Address :
                        %u\n",rec->info,rec,rec->next);
                  rec = rec->next;
            }
      }
}

int peep(Stack *rec)
{
      if(rec==NULL)
            printf("Stack is empty.");
      else
            return rec->info;
}

int isempty(Stack *rec)
{
      if(rec==NULL)
            return 1;
```

```
      else
            return 0;
}


Stack * push(Stack *rec,int info)
{
      Stack *new rec;
      new_rec = (Stack *)malloc(sizeof(Stack));
      //scanf("%d",&new_rec->info);
      new_rec->info = info;
      new_rec->next=rec;
      rec = new_rec;
      return rec;
}
Stack * pop(Stack *rec)
{
      Stack *temp;
      if(rec==NULL)
            printf("Stack is Empty");
      else
      {
            temp=rec->next;
            free(rec);
            rec = temp;
      }
      return rec;

}


int main()
{
      int i;
      int pins;
      int net[]={1,2,3,4,5,6,7,4};
      printf("How many pins :");
      scanf("%d",&pins);
      for(i=0;i<pins;i++)
      {
            if(isempty(start))
            {
                  start = push(start,i);
            }
else
{
      if(net[i]==net[peep(start)])
      {
            printf("%d",peep(start));
            start = pop(start);
      }
```

```
                else
                {
                        start = push(start,i);
                }
        }


}
        if(isempty(start))
                printf("Switchbox is routable");
        else
                printf("Switchbox is not routable");
        return 0;
}
```

Example 6.3 ***Write a program to find out whether a given string is a palindrome or not using a stack. Ignore white space and other characters. Check your program with the following inputs . "Madam I am Adam" and "was it a cat I saw" and "Level". The program should be case insensitive.***

Solution Here is the code that uses stacks to find whether the string entered is a palindrome or not. The program skips all the white spaces. The stack is modeled using a linked list, because the string can be of any length. Three stacks are used to perform this operation

Here is the illustration. People speak about so many palindromes. I have used one of my favorites.

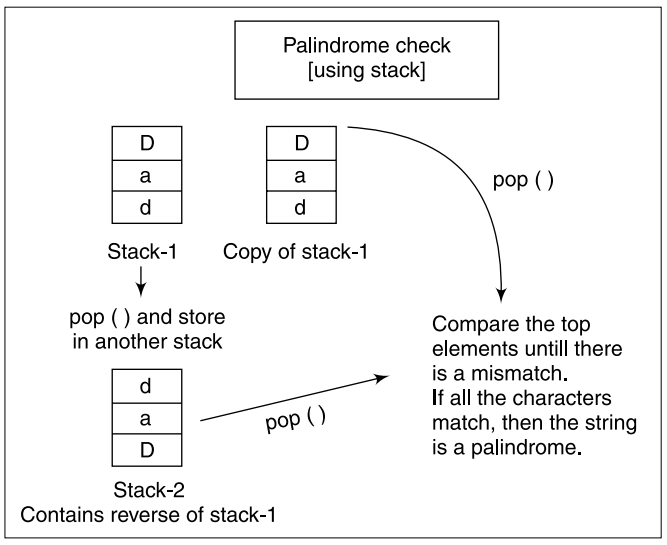


Fig. 6.9

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <ctype.h>
#include <stdlib.h>
```

```
typedef struct Stack
{
      char info;
      struct Stack *next;
}Stack;

Stack *start;
Stack *rstart;
Stack *cstart;

Stack* push(Stack *);
Stack* pop(Stack *);
char peep(Stack *);

int isaplha(char c)
{
     //Checking whether 'c' is a alphabet or not.
     //White space is also not allowed. ASCII of space is 32
     //by writing c!=32 I mean that ASCII of c will not be 32.
      if(((c>='a' && c<='z') || (c>='A' && c<='Z')) && (c!=32))
            return 1;
      else
            return 0;
}

//This function will display the Stack top element.
char peep(Stack *rec)
{
      if(rec==NULL)
            printf("Stack is empty.");
      else
            return rec->info;
}

//This function will check whether the Stack is empty or not
int isempty(Stack *rec)
{
      if(rec==NULL)
            return 1;
      else
            return 0;
}

Stack * push(Stack *rec,char info)
{
      Stack *new_rec;
      new_rec = (Stack *)malloc(sizeof(Stack));
      new_rec->info = info;
      new_rec->next=rec;
      rec = new_rec;
```

```
      return rec;
}

Stack * pop(Stack *rec)
{
      Stack *temp;
      if(rec==NULL)
            printf("Stack is Empty");
      else
      {
            temp=rec->next;
            free(rec);
            rec = temp;
      }

      return rec;

}

int main()
{
      char c;
      int flag=0;
      printf("Enter the characters :");
      do
      {
            c = getche();
            if(isalpha(c))
            {
                  start = push(start,c);
                  cstart = push(cstart,c);
            }


      }while(c!=13);//Untill Enter is pressed

      while(!isempty(start))
      {
            rstart = push(rstart,peep(start));
            start = pop(start);
      }

      while(!isempty(rstart))
      {
            if(toupper(peep(rstart))==toupper(peep(cstart)))
            flag = 1;
else
{
      flag=0;
      break;
}
rstart = pop(rstart);
```

```
            cstart = pop(cstart);
      }
      if(flag==1)
            printf("The string is a palindrome");
      else
            printf("The string is not a palindrome");
      return 0;
}
```

Try this program with the following inputs:
Mom
LiriL
Brother
Was it a cat i saw!
Madam I m Adam
A Man, A Plan, A Canal, Panama!

6.14 SAGUARO STACK

The saguaro stack or cactus stack is a kind of stack where there can be more than one stack at the top of another stack. In a saguaro stack, one stack can't be popped unless the stack at its top is empty. This stack looks like the saguaro cactus. Thus it got its name.

A saguaro cactus

This type of stack can be very useful. For example say the URL entered in internet explorer contains some illegal characters, using saguaro stack we can extract the invalid characters. Branches of this URL stack are www , the company name and the domain(.com, .edu., .org etc..) See the figure below for more explanations.

A saguaro stack can be thought as the combination of stacks. Here is the structure that is used to represent the stack.

```
typedef struct MyStack
{
      int data[MAX];
      int count;
}MyStack;

typedef struct SaguaroStack
{
      MyStack Stacks[MAX];
      int count;
}SaguaroStack;
```

A Saguaro Stack can't be empty until all its branches are empty.

How to Push an Item in a Saguaro Stack

```
int push(SaguaroStack *st,int whichStack,int whattopush)
{
      if(st->Stacks[whichStack].count==MAX)
      {
            printf("The specified Stack is full!\n");
            return 0;
      }
      else
      {

            st->Stacks[whichStack].data[st->Stacks[whichStack].count] =
whattopush;
            st->Stacks[whichStack].count++;
            sc++;
            printf("Successfully pushed");
            return 1;
      }
}
```

How to Pop an Item from a Saguaro Stack

```
int pop(SaguaroStack *st,int whichStack)
{

      if(st->Stacks[whichStack].count<0)
            printf("Specified Stack is Empty!\nUnderflow Error");
      else
      {
            printf("Last number from Stack %d is
popped\n",whichStack);
            sc--;
      }
      return st->Stacks[whichStack].data[st-
>Stacks[whichStack].count-1];
}
```

6.15 HOW TO WRITE THE ALGORITHM TO USE A SAGUARO STACK TO CHECK A WRONGLY ENTERED URL

This algorithm will work for URLs like www.yahoo.com and will not work for URL like www.google.co.in

1. Accept the URL from the user.
2. Add the characters of the URL one by one in a stack called **URL**.
3. Pop this URL stack 4 times and add the values to another stack known as *domain* Stack.
4. Now pop the URL stack till it is empty and put the values in another stack called *temp*.
5. Now pop the Temp stack 4 times and put the values in another stack known as **WWW**.
6. If the last element of the WWW stack is anything other than "**.**" (Dot) then the URL is not valid.
7. If any of the element of WWW stack apart from the last element is not *w*, then the URL is not valid.
8. If the domain stack's last element is anything other than "." Then the url is not valid.
9. Whatever is left in the temp stack is the company name. This can only contain alphabets numeric digits (0–9) and '_'(Under score) or – (Hiphen) and no other characters. In case it contains any other characters then the URL is not valid.

Here is a figure to illustrates the above algorithm.

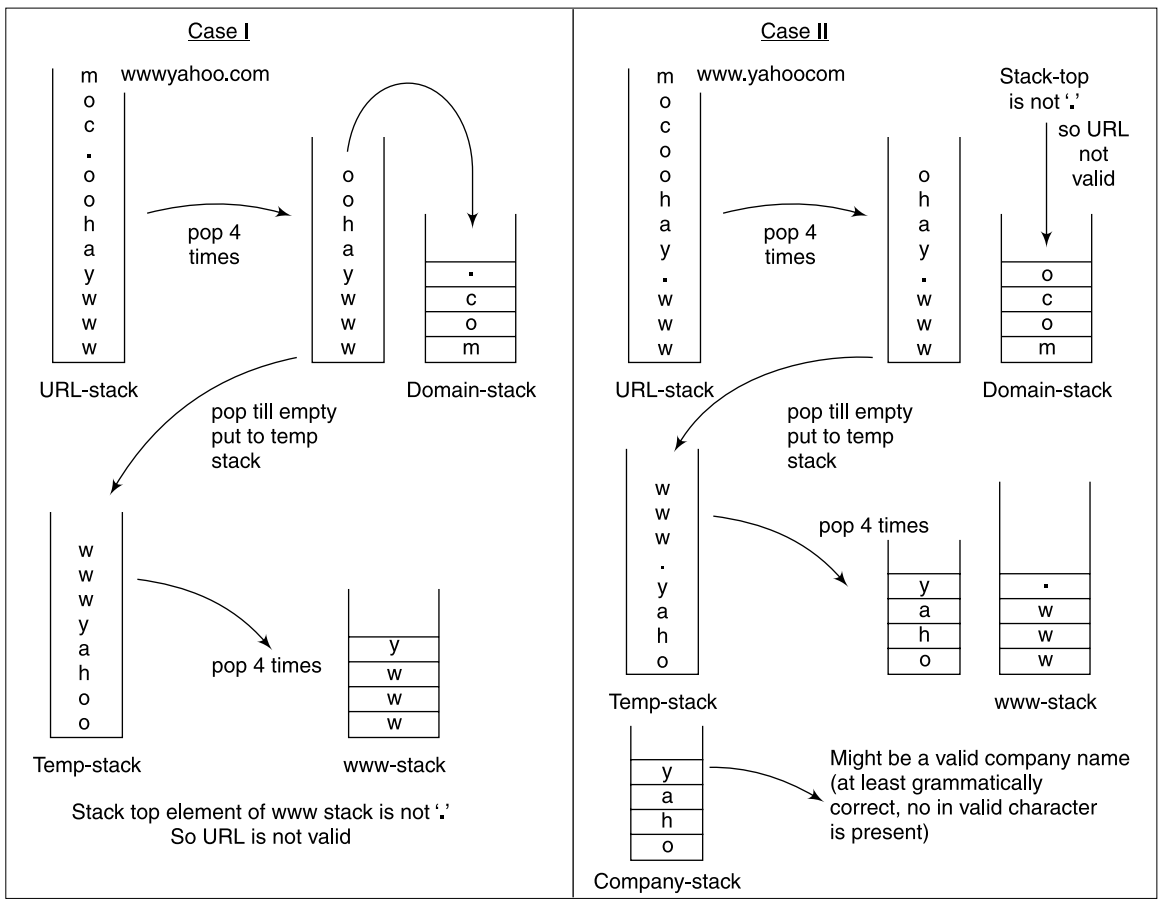


Fig. 6.10

Try Yourself: *Try to implement the above algorithm using linked list versions of stack described above in this chapter. We have to use linked list because the URL name can be of any length.*

Example 6.4 ***Write an XML Reader using Stack. The program should be able to understand what part of the input XML string is a tag and what is the content for that particular string.***

Solution Here is an example of an XML File.

```
<CATALOG>
<CD>
<TITLE>Empire Burlesque</TITLE>
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
</CD>
<CD>
<TITLE>Hide your heart</TITLE>
<ARTIST>Bonnie Tyler</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>CBS Records</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1988</YEAR>
</CD>
</CATALOG>
```

The algorithm to find out the tags and contents from the given XML string is as follows.

1. Read the string till end –1 and start pushing the characters read to a stack if the character is not '/'. Let's call it the xml stack.
2. If at any point while traversing the string you it is found that the next character is '/' then start popping from the xml stack until '>' is got and push the characters read to another stack known as *content* stack.
3. While pushing the elements onto the contents stack from xml stack, if '>' is encountered then come out of the previous loop and pop the content stack.
4. Push '>' to the tag Stack.
5. Pop the stack xml until you get '<' and push them onto the tag Stack.
6. Pop the contents of the tag stack until it's empty and show character by character.
7. Pop the contents of the contents stack until it's empty and show character by character.

See Fig. 6.11 for further details.

Here is the C program. So you know that the program uses 3 stacks. One is known as the XML stack that holds the total XML. The other two holds the tag and the contents within the tag respectively.

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

typedef struct Stack
{
      char info;
      struct Stack *next;
}Stack;
```

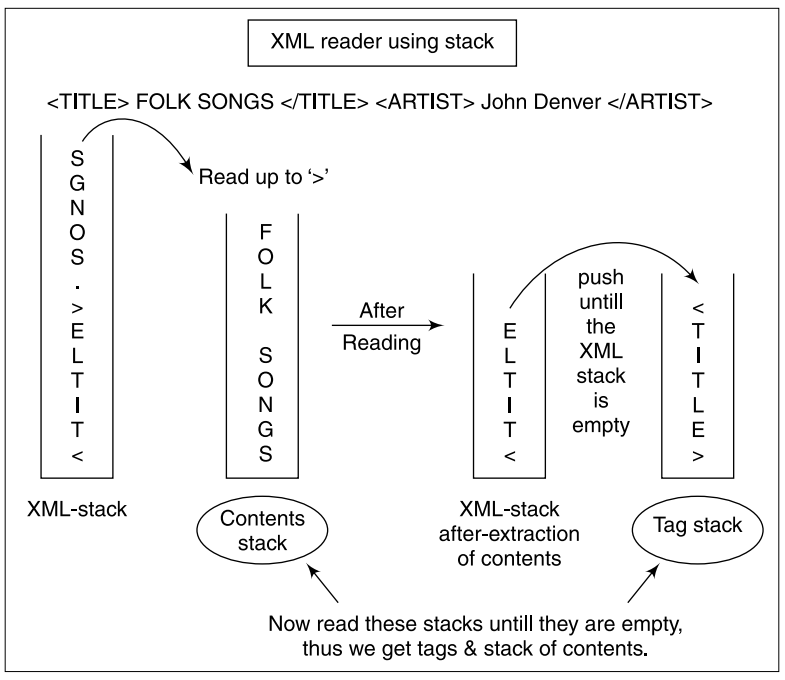


Fig. 6.11

```
Stack *start;
Stack *tag;
Stack *content;


Stack* push(Stack *,char);
Stack* pop(Stack *);
char peep(Stack *);


char peep(Stack *rec)
{
      if(rec==NULL)
            printf("Stack is empty.");
      else
            return rec->info;

}
```

```
Stack * push(Stack *rec,char c)
{
      Stack *new_rec;
      //printf("Enter the value to push :");
      new_rec = (Stack *)malloc(sizeof(Stack));
      new_rec->info=c;
```

```
      new_rec->next=rec;
      rec = new_rec;
      return rec;
}

Stack * pop(Stack *rec)
{
      Stack *temp;
      if(rec==NULL)
            printf("Stack is Empty");
      else
      {
            temp=rec->next;
            free(rec);
            rec = temp;
      }
      return rec;

}


int main()
{
      int i=0;
      char c;
      char string[200]="<TITLE>Empire Burlesque</TITLE> <ARTIST>Bob\
Dylan</ARTIST> <COUNTRY>USA</COUNTRY>";

      for(i=0;string[i]!='\0';i++)
      {

            if(string[i+1]!='/')
            {
               start = push(start,string[i]);
            }
          else
            {
                  do
                  {
                        c=peep(start);
                        content = push(content,c);
                        start = pop(start);
                  }while(c!='>');
                  content = pop(content);
tag = push(tag,'>');
do
{
      c=peep(start);
      tag = push(tag,c);
      start = pop(start);
}while(c!='<');
```

```
printf("\nTag is :\n");
//Displaying the Tag Stack
while(tag!=NULL)
{
      c = peep(tag);
      tag = pop(tag);
      printf("%c",c);
}
                  printf("\nContents is :\n");
                  //Displaying the Contents Tag
                  while(content!=NULL)
                  {
                        c = peep(content);
                        content = pop(content);
                        printf("%c",c);
                  }
            }
      }
      return 0;
}
```

The output of the above program is
Tag is :
<TITLE>
Contents is :
Empire Burlesque
Tag is :
<ARTIST>
Contents is :
Bob Dylan
Tag is:
<COUNTRY>
Contents is:
USA
Try Yourself: *Go one step further and try to find out what are the children nodes of a given node.*

6.16 WHAT IS AN MTFL?

MTFL : *Move to front list* is a special kind of list where the sought elements crawl to the front of the list. Have you ever noticed that the last dialed number from your cell comes at the top of the dialed numbers. So next time you want to dial the same number then you don't have to browse through other numbers. Similarly in word processors like MS Word, you have noticed that the last used font is loaded at the top of the list so that you don't have to search a long list.

Thus we understand that in this list *search* operation is not a passive operation like other lists. Here, when we search an item the item comes at the front of the list.

6.17 HOW TO MODEL AN MTFL USING TWO STACKS WHICH ARE THEMSELVES MODELED BY A LINKED LIST

The strategy to design an MTF list by two stacks involve the following steps.

1. Pop from the stack containing the list items until we encounter the item to be sought.

2. Pop the stack once.
4. Push these items onto the stack-top of another stack, known as *temp stack*.
5. When you get the item to be sought then hold it in some other variable.
6. Pop the temporary stack until it is empty and then push the elements back in the stack containing the items.
7. After you put all the elements of the temporary stack, put the sought item back at the stack top.

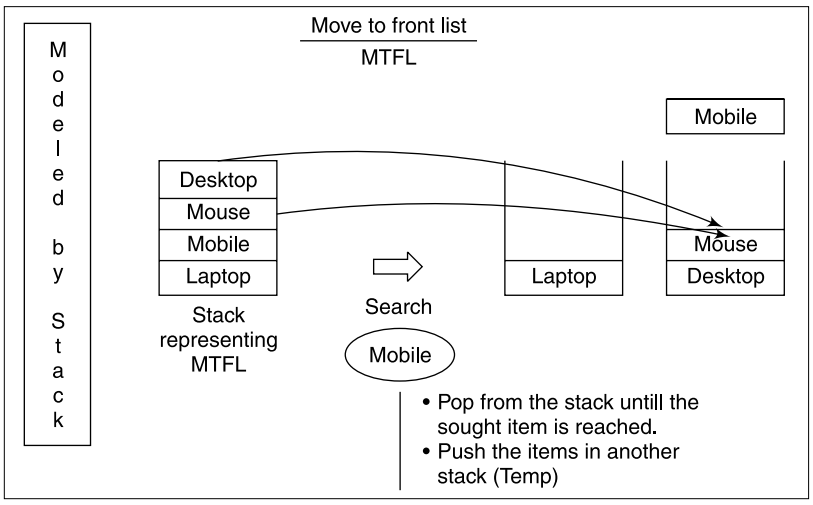


Fig. 6.12

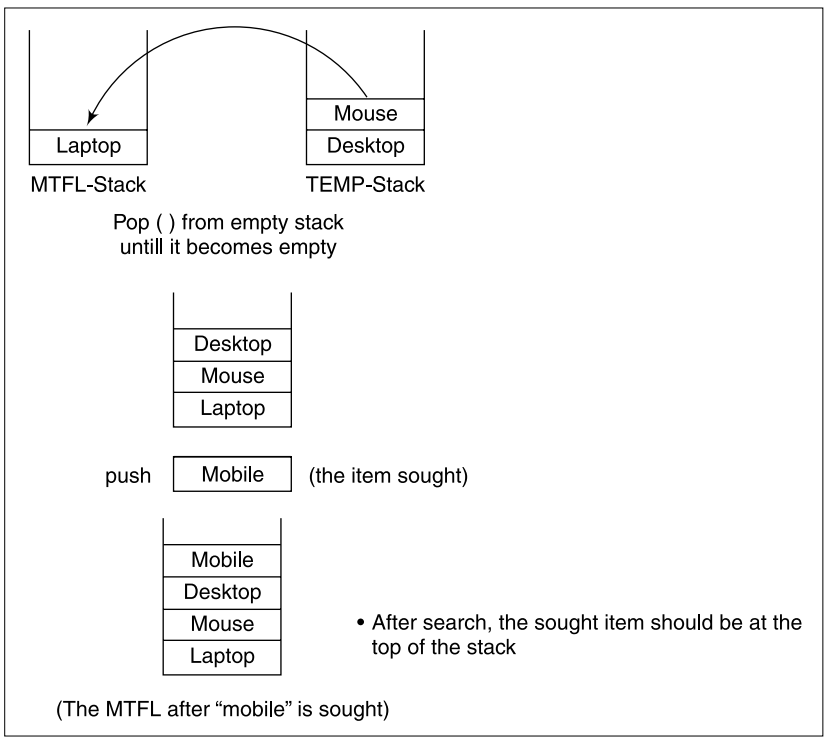


Fig. 6.13

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <ctype.h>
#include <stdlib.h>

typedef struct Item
{
 int info;
 int count;
}Item;

typedef struct Stack
{
 Item it;
 struct Stack *next;
}Stack;

Stack *start;
Stack  *temp;

void display(Stack *);
Stack* push(Stack *);
Stack* pop(Stack *);
void peep(Stack *);
Stack* swap(Stack *);

Stack * swap(Stack *rec)
{
 int temp;
 if(rec->next==NULL||rec==NULL)
    printf("Swapping can't be done. Too few elements");

 else
 {
    temp = (rec->next)->it.info;
    (rec->next)->it.info = rec->it.info;
    rec->it.info = temp;
 }
 return rec;
}
/*//////////////////////////////////////////
The Active Search Operation for MTF List
This function not only searches the List,
But also modifies the list. It pulls the sought
Item to the top of the list.
//////////////////////////////////////////*/
Stack* Search(Stack *rec,int info)
{

     int searchinfo;
     while(rec->it.info!=info)
```

```
{
      temp=push(temp,rec->it.info);
      rec = rec->next;
}
searchinfo = rec->it.info;
rec = pop(rec);

while(temp!=NULL)
     {
           rec = push(rec,temp->it.info);
           temp = temp->next;
     }
     rec = push(rec,searchinfo);
  return rec;
}

//////////////////////////////////////////
void display(Stack *rec)
{
      if(rec==NULL)
            printf("Stack is empty.");
      else
      {
            while(rec!=NULL)
            {
                  printf("Info = %d Address : %u Next Address : %u\n",
rec->it.info,rec,rec->next);
                  rec = rec->next;
            }
      }
}

void peep(Stack *rec)
{
      if(rec==NULL)
            printf("Stack is empty.");
      else
            printf("Stack Top Element = %d",rec->it.info);

}

Stack * push(Stack *rec,int info)
{
      Stack *new_rec;
      new_rec = (Stack *)malloc(sizeof(Stack));
      new_rec->it.info=info;
      new_rec->next=rec;
      rec = new_rec;
      return rec;
}
```

```
Stack * pop(Stack *rec)
{
      Stack *temp;
      if(rec==NULL)
            printf("Stack is Empty");
      else
      {
            temp=rec->next;
            free(rec);
            rec = temp;
       }
      return rec;

}

int main()
{
 do
 {
    int choice;
    int info;
    printf("\n1.Push\n");
    printf("2.Pop\n");
    printf("3.MyStack Top\n");
    printf("4.Is Empty\n");
    printf("5.Search\n");
    printf("6.Exit\n");
    printf("Choice [1-6] :");
    scanf("%d",&choice);
    switch(choice)
    {
    case 1:  printf("Enter the info to push :");
              scanf("%d",&info);
             start = push(start,info);break;
    case 2:start = pop(start);break;
    case 3:display(start);break;
    case 4:peep(start);break;
    case 5: printf("Enter the info to search :");
              scanf("%d",&info);
             start=Search(start,info);break;
    case 6:exit(0);
    }
 }while(1);

 return 0;
}
```

Have you noticed the main difference between this list and other types of lists? In this data structure, search operation is not a passive operation like all other types of lists. Here, whenever we search something, then the sought item comes to the front of the list. Thus, it drastically reduces the searching time for the next time.

Many applications use MTF list. For example, in mobile phones the most recently dialed number comes at the top of the dialed number list. So, next time if you want to call the same number you don't have to browse through your phonebook sequentially, because the number will be sent at the top of the dialed number list.

In departmental store, this MTF list is used to find what the most sought item is in the store. It is also used to find out what else is being sought along with the most sought item.

6.18 HOW TO FIND THE MOST SOUGHT ITEM IN A DEPARTMENTAL STORE USING AN MTF LIST

To find out the most sought item in a departmental store, we need to use the count variable of the item structure. Each time an item is sought we need to increase the item count by unity. Then we can write a function that can be described as "See through MTF" which will scan count of all the elements. So the element which has the highest count is the most sought item till then.

Here is the function code to find the most sought item in the store.

Item MostSoughtItem(Stack *rec)

```
 Item MostSoughtItem(Stack *rec)
{
      int maxcount=rec->it.count;
      Item mostsought;
      Stack *crec;
      crec = rec;
      while(rec!=NULL)
      {
            if(rec->it.count>maxcount)
                  maxcount=rec->it.count;
            rec=rec->next;
      }
```

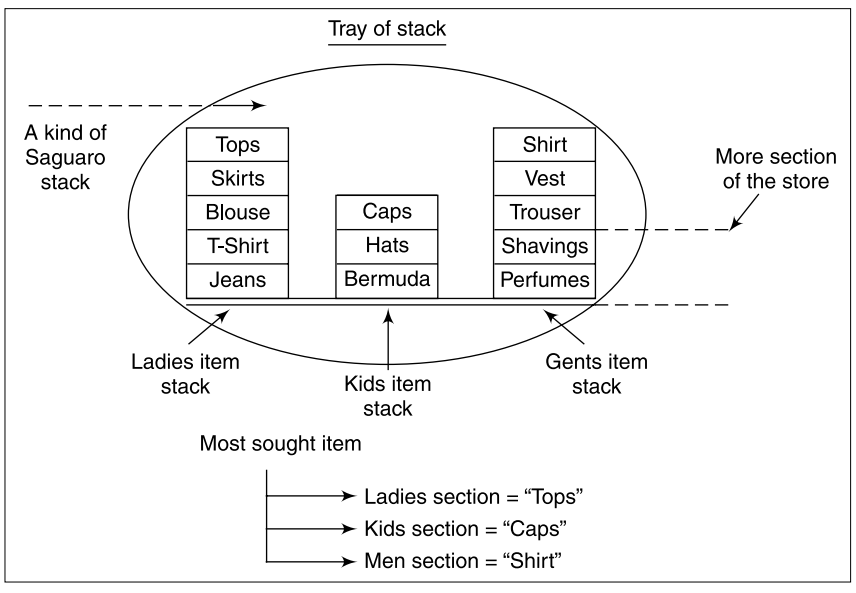


Fig. 6.14

```
      mostsought=findit(crec,maxcount);
      return mostsought;
}


Item findit(Stack *rec,int max)
{
      while(rec!=NULL)
      {
            if(rec->it.count==max)
                  break;
      }
      return rec->it;
}
```

If you notice carefully, you will notice that the most sought item will be available at the top of the stack always. So the `peep()` function only will suffice to find the most sought item in a store.

Now, to make things complicated, we may need to find out the most sought item from different sections [ladies, gents, kids, home, etc] of a store. Then we can use a tray of stacks (May be considered as a different version of a cactus stack), where each stack represents the sought item list for each section. The fig. above is a pictorial representation of the phenomenon.

6.19 WHAT IS BACKTRACKING?

Backtracking is a strategy for finding solutions to constraint satisfaction problems. The term *backtrack* was coined by American mathematician *D. H. Lehmer* in 1950s. Constraint satisfaction problems are problems with a complete solution, where the order of elements does not matter.

The problems consist of a set of variables each of which must be assigned a value, subject to the particular constraints of the problem. Backtracking attempts to try all the combinations in order to obtain a solution. Its strength is that many implementations avoid trying many partial combinations, thus speeding up the running-time.

6.20 HOW TO DEVELOP A BACKTRACKING ALGORITHM TO FIND A PATH IN A MAZE USING STACK

Backtracking is a very useful concept in programming to solve problems with partial solutions. Here is the pictorial description how to find path from a maze using stack.

1. Model a cell using the row and column, and an integer value to keep the track whether the cell is already visited or not.
2. Create a maze of these cells.
3. Move in any direction that is possible [If a cell is not a dead end or if the cell is already visited then the cell is not go able] and push that onto a stack and change the value of the integer is visited to 1. That is set by default false.
4. Push until you reach a dead end.
5. If you reach a dead end pop one item at a time from the stack and put that in a variable.
6. After popping see the stack top cell.
7. If there are more cells where you can go apart from the cell at the stack-top, stop popping.
8. Start from step 3 untill you reach the destination.

There are many problems that can be solved using backtracking.

There can be many such problems which can be solved with backtracking using stack. Like 8 Queens Problem, Bishop Problem, etc.

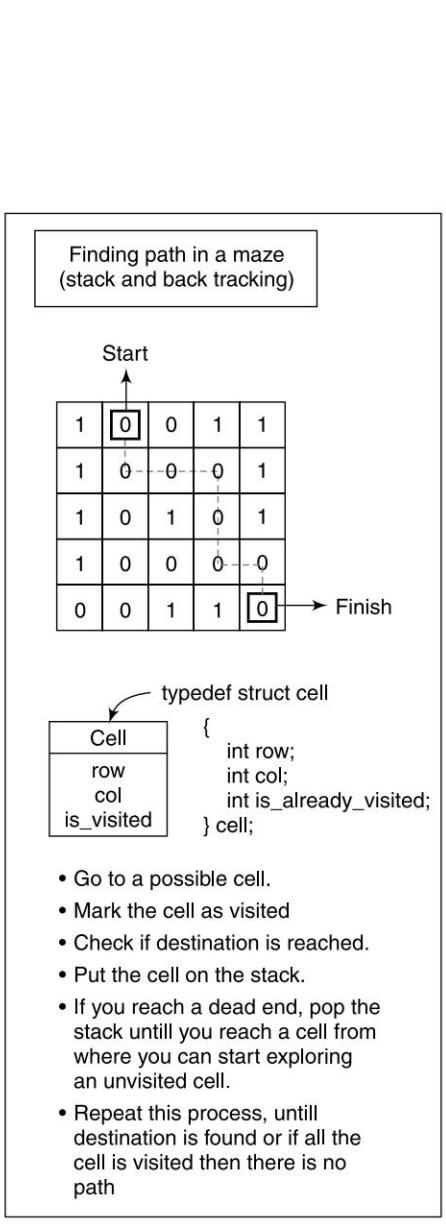


Fig. 6.15

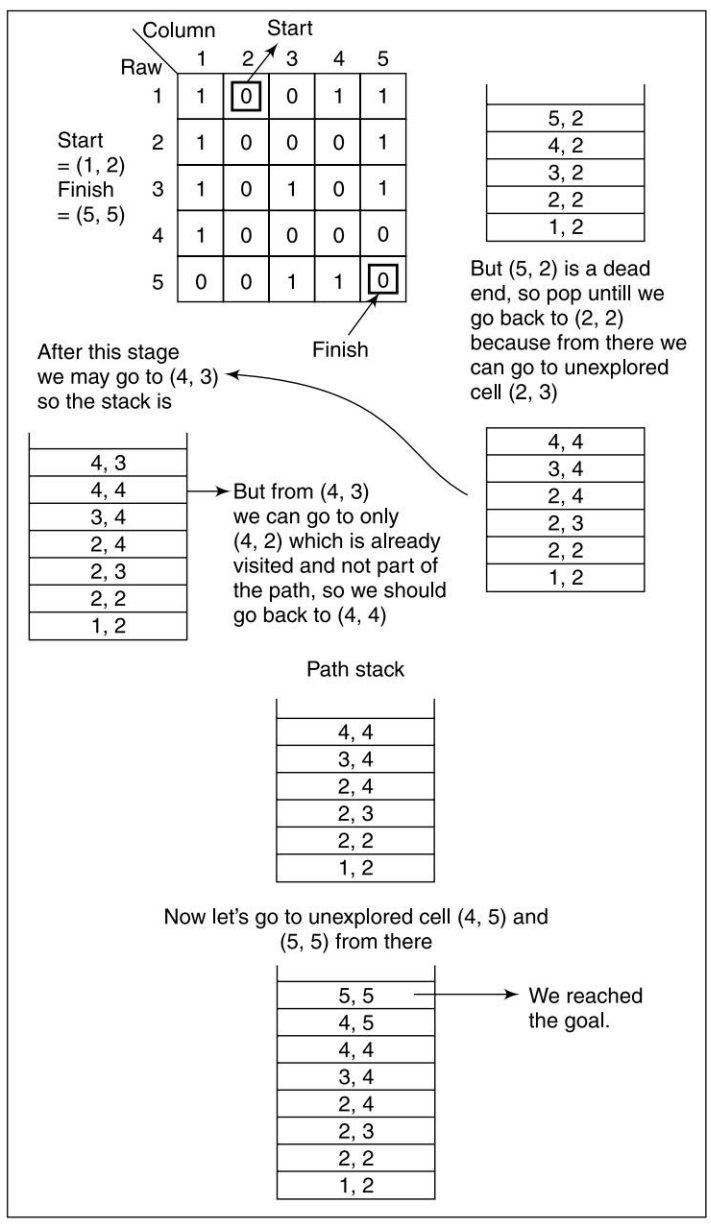


Fig. 6.16

Example 6.5 ***Write a program that takes as input a file name that contains directions to a place. Each line of the file should contain text of the form***

```
<direction-to-travel> <route – name>
```

For example

"north on RT 511"

Use a stack to produce instructions on how to come back from that place.

Solution Here is the C code.

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <ctype.h>
#include <string.h>

typedef struct Stack
{
      char info[30];
      struct Stack *next;
}Stack;

Stack *start;



void display(Stack *);
Stack* push(Stack *,char d[]);
Stack* pop(Stack *);
char* peep(Stack *);
char* reversedirection(char *);


void display(Stack *rec)
{
      if(rec==NULL)
            printf("Stack is empty.");
      else
      {
            while(rec!=NULL)
            {
                  printf("Info = %s \n",rec->info);
                  rec = rec->next;
            }
      }
}

char * peep(Stack *rec)
{
      if(rec==NULL)
            return "Stack is Empty";
      else
            return rec->info;
}

Stack * push(Stack *rec,char d[])
{
      Stack *new_rec;
      new_rec = (Stack *)malloc(sizeof(Stack));
      strcpy(new_rec->info, d);
      new_rec->next=rec;
      rec = new_rec;
```

```
      return rec;
}

Stack * pop(Stack *rec)
{
      Stack *temp;
      if(rec==NULL)
            printf("Stack is Empty");
      else
      {
            temp=rec->next;
            free(rec);
            rec = temp;
      }
      return rec;

}



char* reversedirection(char direction[])
{

      if(direction[0]=='E')
      {
            direction[0]='W';
            direction[1]='E';
            return direction;
      }

      if(direction[0]=='W')
      {
            direction[0]='E';
            direction[1]='A';
            return direction;
      }

      if(direction[0]=='N')
      {

            direction[0]='S';
            direction[2]='U';
            return direction;

      }

      if(direction[0]=='S')
      {

            direction[0]='N';
            direction[2]='R';
            return direction;
      }

}
```

```
int main()
{

      char file[20];
      char route[30];

      FILE *fp;
      printf("Enter the Route Direction File :");
      fflush(stdin);
      scanf("%s",file);
      fp = fopen(file,"r");
      printf("On the way there \n");
      while(!feof(fp))
      {
            fgets(route,strlen(route),fp);
            printf("%s \n",route);
            start = push(start,route);
      }

      fclose(fp);
      printf("Return Trip Guide \n");
      while(start!=NULL)
      {
            printf("%s \n",reversedirection(peep(start)));
            start = pop(start);
      }
      printf("End\n");
      return 0;

}
```

Here is a sample run of the program..
Enter the Route Direction File :goingthere.txt

On the way there
NORTH ON SALISBURY RD
WEST ON ROBIOUS RD
NORTH ON RT 511
WEST ON RT 150

Return Trip Guide
EAST ON RT 150
SOUTH ON RT 511
EAST ON ROBIOUS RD
SOUTH ON SALISBURY RD
End

Contents of going there file is
NORTH ON SALISBURY RD
WEST ON ROBIOUS RD
NORTH ON RT 511
WEST ON RT 150

REVISION OF CONCEPTS

Some Key Facts about Stack and Terminology

1. LIFO : Last In First Out
2. MRA : Most Recently Added
3. Push : Entering one element at the Stack top
4. Pop : Deleting the MRA element
5. Swap: Swapping the top two elements of the Stack.
6. Peep: Look at the stack – top element.
7. Stack is a LIFO List. The MRA element is taken out first.
8. First element entered initially can only be extracted from a stack at the end.
9. Stack is heavily used in compiler designing and parsers. Compilers use stack internally to keep track of the function calls.
10. Any recursive method (like the iterative root finding algorithms of polynomials) can be written as a non-recursive algorithm using stacks.
11. Whenever you need to design a parser, the best suited data structure will be stack. See the XML Reader code below for further clarification.
12. Stack is used to model other data structures like queues etc.
13. Recursive algorithms can be made non-recursive by using stacks.
14. To model any "Cascaded System" stack is the best data structure. By "Cascaded" I mean that output of one system is the input to the other system. So stack can be highly used to model cascaded LTI (linear Time Invariant) systems in electronics.
15. To design the functional languages like LISP, Haskell etc, stack is used.
16. Browsers use stack to store different type of protocols. Whenever a new URL is entered in the address bar, the browser match the protocol of the URL with any of the stack entries.
17. Stacks are used in games programming/word processing software to do the undo move feature.
18. Oracle uses Stacks to keep track of the transactions. So whenever we wish we can do a rollback. This rollback is nothing but popping of the stack that holds the transactions.
19. UNIX/Linux kind of OS, uses stack to keep track of the commands issued. The same things happen when we use DosKey command in DOS.

REVIEW QUESTIONS

1. Where all you can use stack in real life?
2. Write pseudo code for basic operations on a stack that is modeled by two queues?
3. Can we have a stack of structures? Give some example where it might be needed.
4. Can we have a stack of function pointers? Where do you think this is used ?
5. There is an integer stack *S* then what are the following commands doing ? Assume the methods perform as they are defined in the chapter. push(S,22) ; push(S,S.count); push(S,sin(20)-pop()); pop() ; display_Stack_top() + 5;
6. Create a stack of point structures.
7. Create an array of this stack elements.
8. How will you access the co-ordinates of a point inside such a structure?
9. Can you model a queue using two stacks ?
10. How a saguaro stack can be used in compilers ?

11. Convert the following Infix expression to postfix a+b*(c-d^e+g*(k-l)). Show contents for both stack and the answer in each and every step.

12. Say there is a stack called *MS*. And all the basic operations are defined. Predict what will be the output of the pseudo code ? All methods are same as they are described in the chapter.

```
PUSH(MS, 23)
PUSH(MS, 34)
PUSH(MS, 56)
POP( )
PUSH(MS,isEmpty( ));
DISPLAY_STACK_TOP();
```

PROGRAMMING PROBLEMS

1. There is a garage. This has only one entrance. So the car that is parked first can't go out until all the other cars that are parked after it are moved out. Model such a garage. Write a simulation program. Use linked implementation of the stack.

2. Model a stack using two linear queue which are modeled using linked list.

3. Model a queue using two stacks which are modeled using linked list.

4. Write a program that will read an HTML file and will report for missing tags. Assume that there is no orphan tag[*Clue: An orphan tag is a tag that doesn't require the opening tag. Example </br>*]

5. One grocer needs to automate his shop. The basic problem is output of each of his calculation is the input to the next level. He will use 4 commands "ADD", SUBTRACT,DIVIDE,MULTIPLY along with 2 arguments. He will write the commands in a notepad file like
(a) ADD 45 99.56 (b) SUBTRACT 34 (c) DIVIDE 2 (d) ADD 6
(e) MULTIPLY 45
So as you can see addition of 45 and 99.56 in the first statement will be the first parameter for the next command "SUBTRACT". So 34 will be subtracted from (45+99.56)=144.56 and so on. Write a program using stack to calculate these type of file inputs.

6. Use the structure from the phonebook example in Structure chapter. And create an MTFList of these structures. Instead of simple array use this MTFList to code that phonebook program.

7. Use stack to solve N queens problem on a chess board.

8. Write a function add an element to a saguaro MTF.

9. Write a function to search for an element in a saguaro MTF.

10. Write a program to convert an infix notation to postfix using stack.

11. How can the variables be swapped at the top of the stack

12. Write a program using stack to represent a cascaded system.

13. Write a program using stack to represent a one way garage system.

14. Write a program to demonstrate how "undo feature" can be enabled using stack. Re-write the program on POS (Point of Sale Software in the Structure chapter) to hold the transactions on a stack. If a customer says that he doesn't want to buy an item then we can revert back.

15. Write a program for simple calculations. The latest command should be visible retrievable with the cursor keys. For example if the program accepts the commands add, sub etc then the stack will hold these commands as they are applied and once required it can pull up the last command.

16. Modify the program on 8. Use an MTF list to put the most used command at the top of the stack.

7

Queue
Waiting or Privileged?

INTRODUCTION

In the last chapter, we discussed about stack data structure which is a LIFO list. In this chapter we will discuss about a list which resembles the real life FIFO(First In First Out) queues. In real life queue, the first request is served first and the last request is served last. This type of list is known as FIFO list or normally termed as *queue*. Stack is also a queue which is LIFO queue. In this chapter we will discuss about the different operations possible on a queue and different diverse applications of this simple data structure. A queue can be modeled using either an array or a linked list.

7.1 HOW TO MODEL A LINEAR QUEUE USING AN ARRAY

A linear queue can be implemented in several ways. First of all we will model it using a simple array. A queue has a front and a rear. We have modeled a linear queue of character arrays. There is a structure to keep all the necessary information about the queue. Here is the structure that we will be using for other methods that operate on this queue. Later in this chapter when we will model the queue with the linked list and will call it *linked linear queue*.

```
typedef struct MyQueue
{
      int front;
      int rear;
      int count;
      char MyList[MAX][MAX];
}MyQueue;
```

Front represents the starting index where the array starts. Rear is the rear end of the queue and count is the number of entries in the queue at any moment of time. `MyList` is the array that holds the character arrays.

7.2 HOW TO INITIALIZE THE LINEAR QUEUE DEFINED ABOVE

Before we start to describe the basic operations of a queue which is modeled using array the queue we should know how the queue is initialized. Here is the code to initialize the linear queue that is modeled above.

```
int f=0,r=0,c=0;
void initQ(MyQueue *qu)
{
      qu->front = f;
      qu->rear = r;
      qu->count = c;
}
```

7.3 HOW TO APPEND AN ELEMENT IN THE QUEUE

As this is an array based representation so before we go for appending something at the end of the queue we should check for the availability of space. There is one method, **isFull()**, which checks whether the queue is full or not. Here is the code of that method.

```
int isFull(MyQueue *qu)
{
      if(qu->count>=MAX)
            return 1;//The queue is full
      else
            return 0;//There is space for more entries.
}
```

Here we have used pointer to the structure so that the access time is less. Now here is the code to append a new character array at the end of the queue. This is known as *enqueue*. The Queue defined above maintains a list of nicknames.

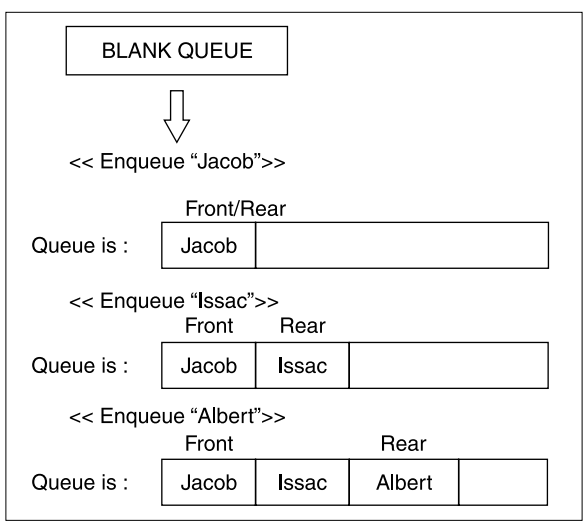


Fig. 7.1

```
void appendQ(MyQueue *qu)
{
     char name[10];
     if(isFull(qu)==1)//There is no space for a new entry.
          printf("Overflow Error!");
     else
     {
          printf("Enter nickname of the person joining the queue :");
          fflush(stdin);
          gets(name);
          strcpy(qu->MyList[qu->rear],name);
          r++;//One new element, rear shifts one unit to the back
          c++;//One new element, so count increase by unity
          qu->rear = r;
          qu->count = c;
          printf("\nSuccessfully Appended\n");
     }

}
```

The picture below describes the situation even better.

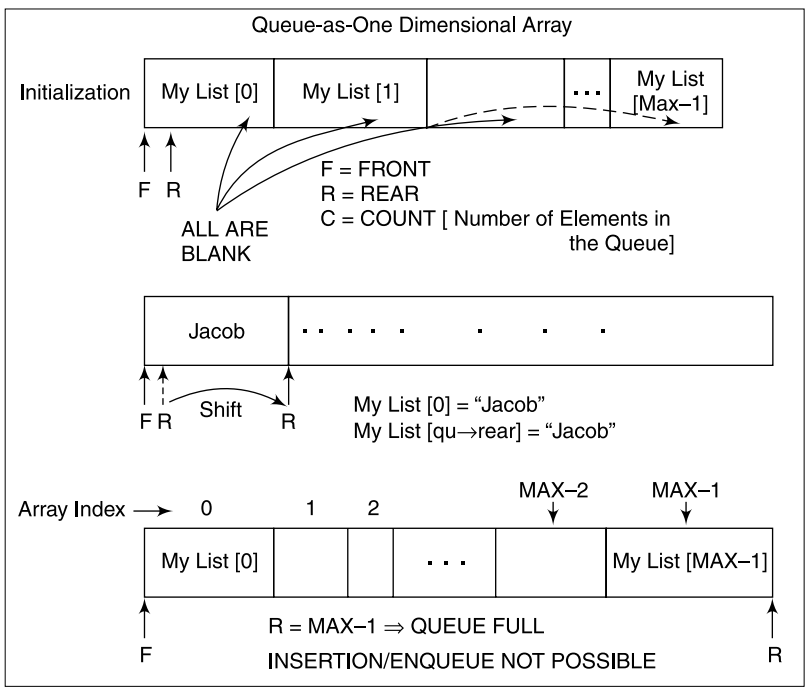


Fig. 7.2

7.4 HOW TO DELETE FROM A QUEUE

Deletion is only allowed at the front of a queue. Unlike stack it is a data structure where the first entry is deleted first. That's why it is a FIFO list. The deleting operation is commonly known as *dequeue*. The deletion operation done is a LOGICAL deletion. It is not a physical deletion. To understand this concept see one figure of the code mentioned as follows.

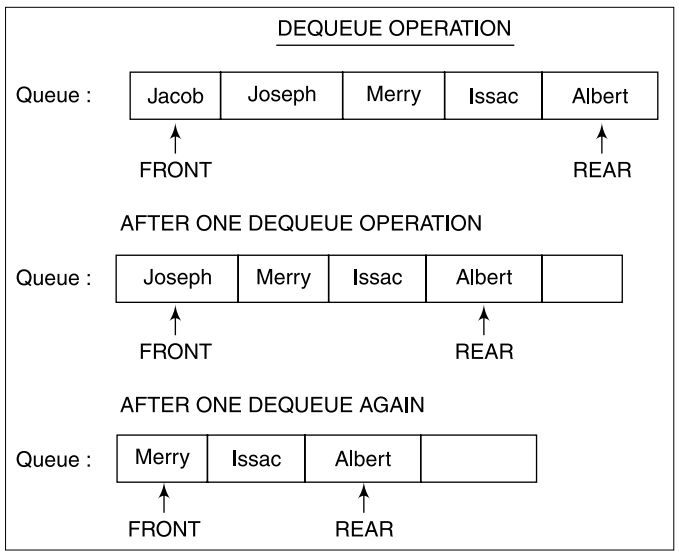


Fig. 7.3

This diagram gives a conceptual view.
Here is the programmatic dissection of the operation.

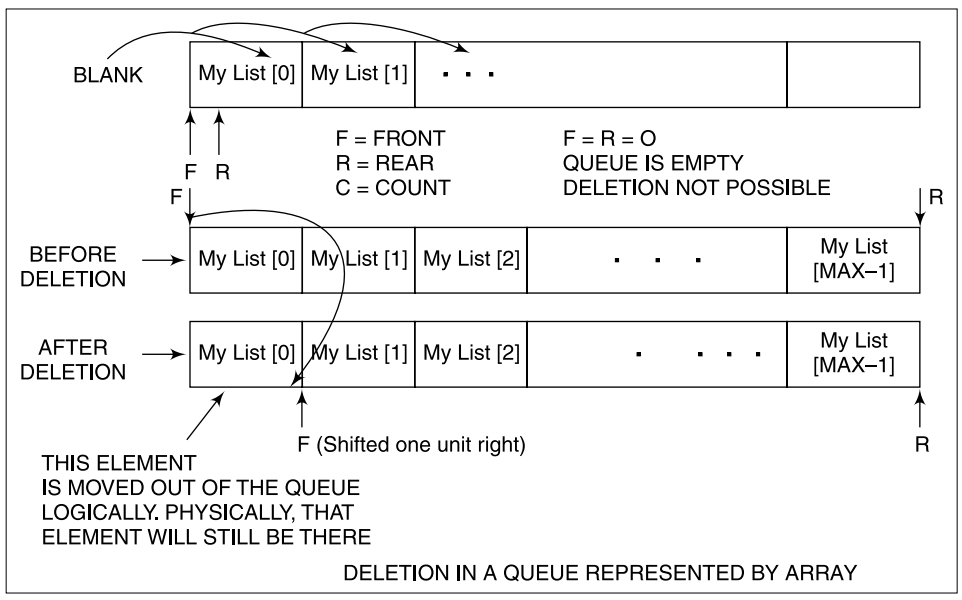


Fig. 7.4

```
void deleteQ(MyQueue *qu)
{
      if(isEmpty(qu)==1)//There is no element in the queue to dequeue
            printf("\nUnderflow error");
      else
      {
            f++;//Next element is now the new front of the queue
```

```
            c--;//There is one less element
            qu->front = f;
            qu->count = c;
            printf("\nFirst person at the front of the queue
                    is serviced. Next please");
      }
}
```

Before deleting the element from the queue we need to check whether there is at all any element or not. In case there is no element in the array, the error occurred is known as *underflow error*. The method that checks whether the queue is empty or not is shown as follows.

```
int isEmpty(MyQueue *qu)
{
      if(qu->count==0)
            return 1;
      else
            return 0;
}
```

7.5 HOW TO SEARCH AN ELEMENT IN THE QUEUE

Apart from enqueue and dequeue operations, there will be certain conditions where you will need to search a particular element. Imagine a situation some people are standing in a queue for meeting a doctor. Suddenly somebody feels very sick and doctor wants to meet him as an urgent case. Then the person needs to be sought in the queue.

The searching algorithm here is a linear search. Starting from the front you should move till the rear end of the queue and check each and every element with the sought element. If the sought element is matched then break from the loop and don't continue searching any longer. Here is the code:

```
#define MAX 10
enum{NOTFOUND,FOUND};
int search_status = NOTFOUND;
int r=0,f=0,c=0;

void searchQ(MyQueue *qu)
{
     int k;
     char name[10];
     int occur=1;
     if(isEmpty(qu)==1)
          printf("Nobody is there in the queue");
     else
     {
          printf("Whom do you want to search :");
          fflush(stdin);
gets(name);
 //rotating till the read end
for(k=qu->front;k<qu->rear;k++)
     if(strcmpi(qu->MyList[k],name)==0)//checking
     {
          printf("Occurrence %d of %s is at position %d\n",
               occur,name,k-qu->front+1);
```

```
                occur++;
                search_status=FOUND;
            }
            if(search_status==NOTFOUND)
                printf("Nobody is there with this name in the
queue!");
    }
}
```

7.6 HOW TO DISPLAY THE ELEMENTS IN A QUEUE

To view what all elements are there we need to traverse the list. Here is the code to traverse the above queue.

```
void displayQ(MyQueue *qu)
{
     int k;
     if(isEmpty(qu)==1)
          printf("\nSorry there is no one in the queue");
     else
     {
          for(k=qu->front; k<qu->rear; k++)
               printf("Name :<%s>     Position is Queue :%d\n",
                      qu->MyList[k],k-qu->front+1);
     }
}
```

7.7 MODEL A QUEUE USING A LINKED LIST

The main disadvantage to model a linear queue using array is that it has a maximum number of possible entries. You can't insert *N* number of values [Where *N* is changing dynamically] in it. So we had a function in the previous implementation called **`isFull()`** that was used to check whether the queue is

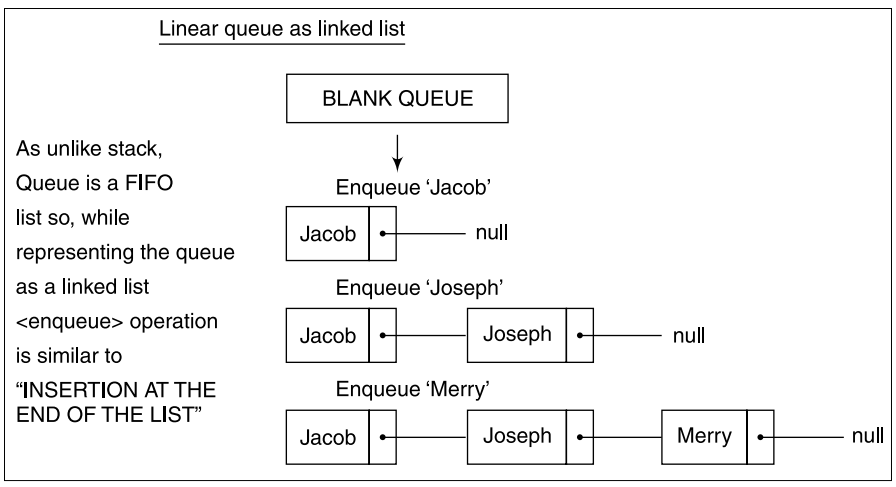


Fig. 7.5

full or not. Now the queue is modeled using a single linked list. So there is no need to check whether the queue is full or not every time we go for a new insertion. Here, a linear queue of integer variables is modeled. Here is the structure that represents each element of the queue.

```
typedef struct node
{
      int data;
      struct node *next;
}node;
```

Illustration of a queue implemented using a linked list is as follows.

7.8 HOW TO APPEND AN ITEM TO A LINKED QUEUE

Appending at the end of this linked linear queue is very simple. Here is the code for enqueue.

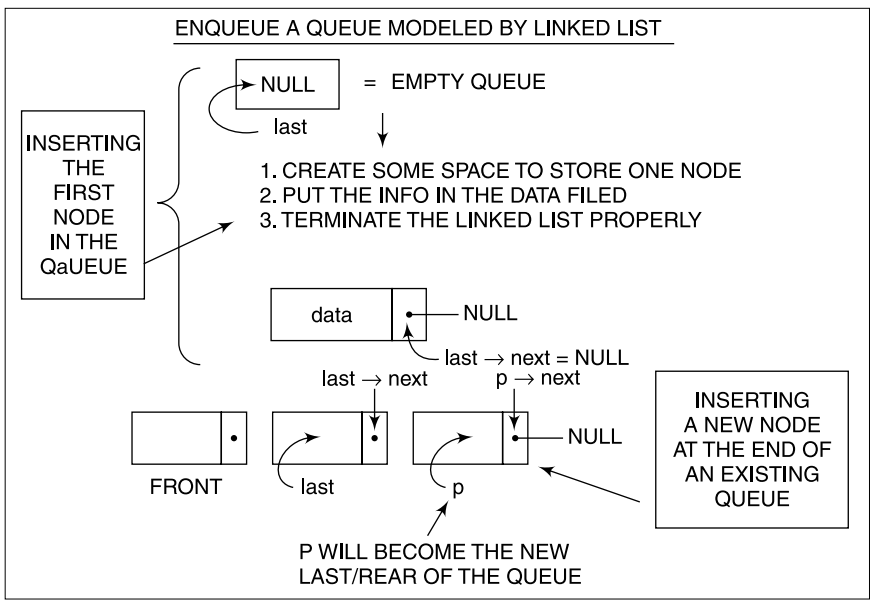


Fig. 7.6

```
node* enqueue(node *last,int info)
{
      if(last==NULL)//If this is the first node for the queue
      {
            last = (node *)malloc(sizeof(node));
            last->data = info;
            last->next = NULL;
            return last;
```

```
}
else
{
      node *p = (node *)malloc(sizeof(node));
      if(p)
      {
            last->next = p;
            p->data = info;
            p->next = NULL;
      }
      return p;
}
}
```

How to find the Number of Elements in a Queue

```
int count(node *h)//h denotes a pointer to the front of the queue
{
      int numberofnodes=0;
      node *p = h;
      if(p==NULL)//The Queue doesn't exist at all!
            return 0;
      else
      {
            for(;p!=NULL;p=p->next)
                  numberofnodes++;
            return numberofnodes;
      }
}
```

In many data structure books, you will find that they wrote two separate functions to find the size (number of nodes) in the queue and to find out whether the queue is empty or not. That approach increases the readability of the program. Here we are writing one function that will check whether the list is empty or not.

```
int isEmpty(node *h)
{
      return count(h)==0;
}
```

As you can see **`isEmpty( )`** behaves just like a synonym to count(). These types of functions are given a special name *Wrapper Function*. They are very common in computer science. More about wrapper functions will be discussed in the chapter on date.

7.9 HOW TO DELETE THE FRONT ITEM OF A QUEUE

Before deleting it is necessary to check whether the queue is empty or not. If there is no element then front will be NULL. Here is the code for deleting the first entry from the queue.

```
node* dequeue(node *h)
{
      node *x = h;//identifying the first node that has to be freed
      node *p = h->next;
      free(x);//freeing the memory space.
      return p;//returns a pointer to the front of the queue
}
```

7.10 HOW TO SEARCH AN ELEMENT IN A QUEUE

Searching an element requires traversing the linked list from front to the rear. If the sought element matches with any of the element of the linked list then we will break from the loop otherwise the loop will continue. Here is the code for search.

```
enum {NOTFOUND,FOUND};
int searchindex(node *h,int s)
{
      int search_status = NOTFOUND;
      int c=0;
      node *p = h;
      for(;p!=NULL;p=p->next)
      {
            c++;
            if(p->data==s)
            {
                  search_status = FOUND;
                  break;
            }
      }
      if(search_status == FOUND)
            return c;
      else
            return -1;
}
```

7.11 HOW TO DISPLAY THE ELEMENTS OF A QUEUE

Here is the code for displaying the elements of the linked linear queue. Before we try to display the contents of the linear queue, we should check whether the list is empty or not.

```
void display(node *h)
{
      node *p = h;
      for(;p!=NULL;p=p->next)
            printf("Value = %d Address %u Next
                  Address %u\n",p->data,p,p->next);

}
```

How to Find the Front Element of a Queue

//This function returns the pointer to the front of the queue.

```
node* first(node *h)
{
      return h;
}//Have you noticed that this function is a “wrapper method”

//This function returns the front element in the queue
int front_element(node *h)
{
      return first(h)->data;
}
```

How to Find the Back Element of a Queue

//This function returns the pointer to the back of the queue.

```
node* last(node *h)
{
      node *p = h;
      for(;p->next!=NULL;p=p->next);
      return p;
}

//This function returns the back element in the queue
int back_element(node *h)
{
      return last(h)->data;
}
```

Here is the client code that uses these functions.

```
int showmenu()
{
     int choice = 0;
     puts("1.Enqueue ");
     puts("2.Dequeue");
     puts("3.Display Queue");
     puts("4.Search Queue");
     puts("5.Show the first element");
     puts("6.Show the last element");
     puts("7.Exit");
     puts("Your choice [1-7]:");
     scanf("%d",&choice);
     return choice;
}


int main()
{
      int val=0;
      node *a=NULL,*c=NULL;
      do
      {
            switch(showmenu())
            {
            case 1: puts("Enter an integer :");
                      scanf("%d",&val);
                      if(count(c)==0)
{
      a=enqueue(a,val);
      c = a;
}
else
{
      a=enqueue(a,val);
}
```

```
                break;
        case 2:if(count(c)!=0)
                c=dequeue(c);
             else
                puts("Queue is Empty!");
             break;
        case 3:display(c);
                break;
                case 4:puts("Enter value to search :");
                     scanf("%d",&val);
                     if(searchindex(c,val)!=-1)
                          printf("The value is found at
                       %d\n",searchindex(c,val));
                     else
                          puts("The value is not found in the queue");
                     break;
                case 5:if(count(c)!=0)
                          printf("Front Element :%d\n",front_element(c));
                     else
                          puts("The Queue is Empty");
                     break;
                case 6:if(count(c)!=0)
                          printf("Back Element :%d\n",back_element(c));
                     else
                          puts("The Queue is Empty");
                     break;
                case 7:exit(0);
                     break;

              }
       }while(1);
     return 0;
}
```

***Try Yourself*:** *Try to remove count() and use* `isEmpty()` *whenever needed.*

7.12 MODEL A LINEAR QUEUE USING TWO STACKS

A Linear Queue can be modeled using two back to back queues. Here is a picture to explain the problem

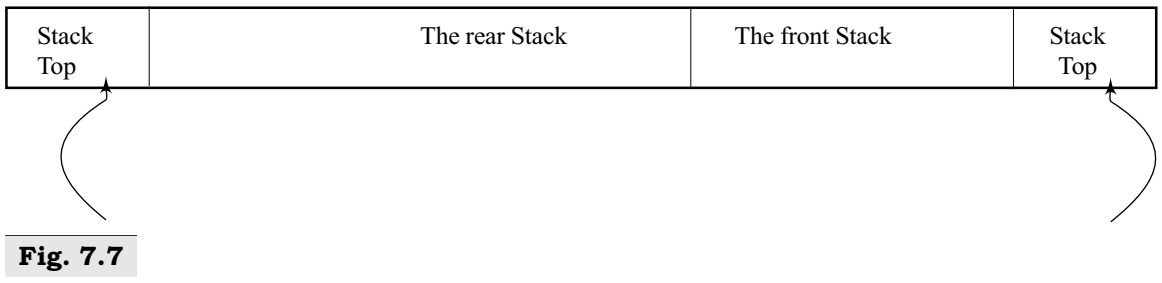


Fig. 7.7

Push operation to this stack is similar to enqueue operation to the queue.

Pop operation from this stack is similar to the dequeue operation from the queue. If we model the stacks using arrays. Then they should use the same array of elements. We can make a global array of integers as element array if we want to model a queue of integers.

Here are the structures that represent such a queue.

```
#define MAX 10

int data[MAX];

typedef struct MyStack
{
      int count;
}MyStack;

typedef struct Queue
{
      MyStack frontstack;
      MyStack rearstack;
}Queue;
```

To understand this concept better, think of a plate holder. At one end you insert the plate and you can recollect it from the front or from the back, where you kept it. Deletion is normally not allowed at the back or rear of a linear queue. Think of a real life situation that you are in a long queue and after some time you decide to leave, you can leave from any place of the queue. In that case, if you were not the last person waiting in the queue, each person who was after you would have been proceeded by one place.

The code in the chapter Stack can be used. Just make the integer array data as global. Remove the declaration from within the MyStack as done above.

To initialize this queue

```
Queue q;
MyStack *ptf=&q.frontstack;//Assigning address of front stack
MyStack *ptr=&q.rearstack;//Assigning address of the rear stack
```

To enqueue this queue is synonymous to push an element at the rear stack. Hence the code to enqueue an element to this queue is

```
push(ptr);
```

To dequeue this queue is synonymous, to pop the stack top element from the front stack. Hence the code to dequeue is

```
pop(pfr);
```

To display the elements of the queue, we just have to traverse the array from start to the end.

7.13 HOW TO MODEL A STACK USING TWO QUEUES

A stack allows insertion and deletion only at the top. On the other hand, a queue allows insertion at the rear end and deletion at the front. So a stack can be modeled using two queues who are directionally opposite. Here is an illustration for better understanding.

The structures needed to model such a stack are listed below.

```
char MyList[MAX][MAX];//The Global Data Source

typedef struct Mystack
{
      int front;
      int rear;
      int count;
}Mystack;
```

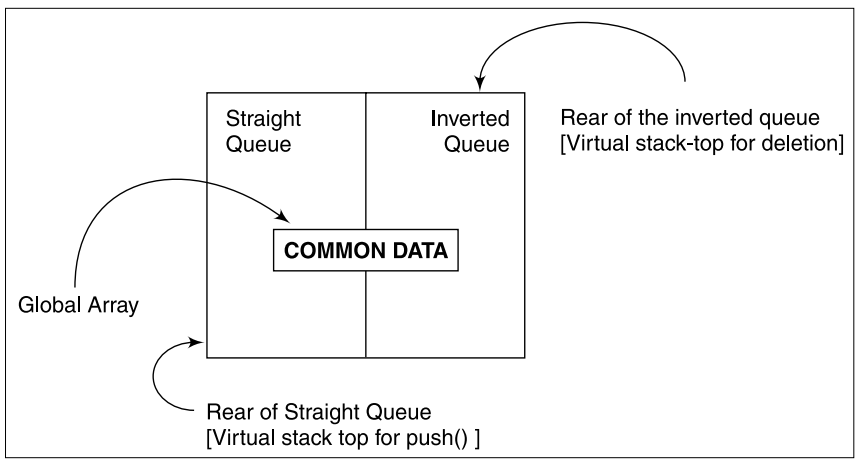


Fig. 7.8

```
typedef struct Stack
{
      Mystack straight;//The straight queue
      Mystack reverse;//The reverse queue
}Stack;
```

All the methods that we used for the Queue operations using an array will be used here to perform basic operations on the component queues. Only the variable MyList should be made global and the function display will change a little. The changed display method is listed.

```
void displayQ(Mystack *qu)
{
      int k;
      if(isEmpty(qu)==1)
            printf("\nSorry there is no one in the stack");
      else
      {
            for(k=qu->rear-1;k>=qu->front;k--)
                  printf("Name :<%s>
                  Position is stack :%d\n",MyList[k],k-qu->front+1);
      }

}
```

To initialize such a stack write

```
Stack s;
Mystack *sqp = &s.straight;
Mystack *rqp = &s.reverse;
```

To push() an element at the top of the stack.

```
appendQ(sqp);
```

To pop the element at the top of the stack.

```
deleteQ(rqp);
```

7.14 MODEL A CIRCULAR QUEUE USING STRUCTURE

A circular queue is a queue where the last element points to the first element. This is not much used in real life because this hardly resembles any real life scenario. So I will not elaborate this concept much.

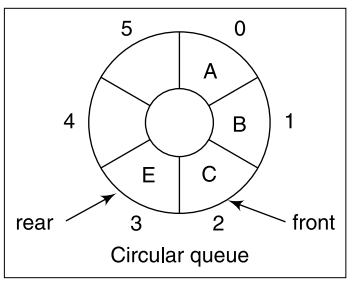


Fig. 7.9

Here is the structure that is used to model a circular queue.

```
typedef struct CirQueue
{
      int count;
      int front;
      int rear;
      char entry[MAXQ][MAXQ];
}CQ;
```

Here the data to be stored in the queue are character arrays.

How to Initialize the Circular Queue

```
void initializeq(CQ *icq)
{
      icq->count = c;
      icq->front = f;
      icq->rear = r;
}
```

How to Add a String to the above Definition of the Circular Queue

```
void addq(char *item,CQ *cqp)
{
      if(cpq->count==MAXQ)
            puts("Queue is full\n");
      else
      {
            c++;
            cpq->count=c;
            r = (r+1)%MAXQ;
            cpq->rear = r;
            strcpy(cpy->entry[cpq->rear],item);
            printf("Successfully Appended ");
      }
}
```

How to Delete the First String from the Circular Queue

```
void delq(CQ *cpq)
{
      if(cpq->count==0)
                  puts("Queue is empty");
      else
      {
            c--;
            f = (f+1)%MAXQ;
            cpq->count = c;
            cpq->rear = f;
            puts("Successfully Deleted the front item");
      }
}
```

How to Search a Particular Item in the Circular Queue

```
enum {NOTFOUND,FOUND};
int search_status = NOTFOUND;

void searchq(char *sc,CQ qu)
{
      if(qu->count<0)
            puts("Queue is empty");
      else
      {
            int occur=1;
            for(int k=qu->front;k<=qu->rear;k++)
                  if(strcmpi(Occurance %d of %s is at position %d\n",
                        occur,sc,k-qu->front+1);
                  occur++;
                  search_status=FOUND;

      }
            if(search_status==NOTFOUND)
                  puts("Searched string is not found");
}
```

How to Display the Circular Queue

```
void displayq(CQ *cpq)
{
      if(cpq->count==0)
            puts("Queue is empty");
      else
      {
            for(int k=cpq->front;k<=cpq->rear;k++)
                  printf("<%s> Location in queue :%d\n",
cpq->entry[k],k-cpq->front+1);
      }
}
```

7.15 MODEL A PRIORITY QUEUE USING AN ARRAY

Priority Queue is a data structure that is a close cousin of queue data structure. Unlike queue where the elements are added at the end, in priority queue the elements are added at the specified location according to their priority. The priority can be anything. Figure below shows that there can be two types of priority queues available. One is called a MAX-PQ where the elements are added in the decreasing order [The first element in the queue has the maximum priority] of their priority, and the other one is a MIN – PQ [The first element will have the minimum priority] where the elements are added according to increasing priority.

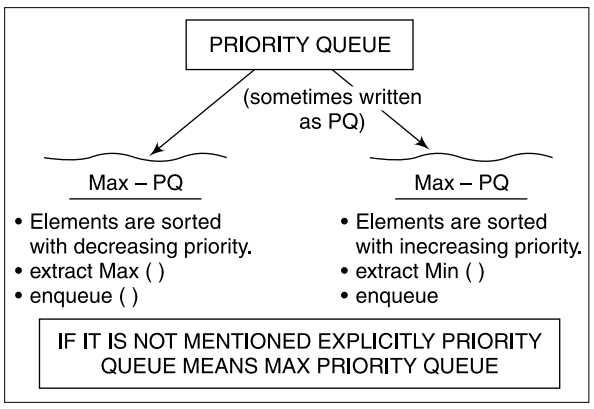


Fig. 7.10

A priority queue is a kind of queue where the elements are added according to the decreasing priority. The priority can be anything. A typical example for the priority queue is the printer queue. In some printer queues the job with least amount of time needed will be the *highest priority element*. On the other hand in some queues it may happen that each element is given a priority. The element with the highest priority is processed first and the lowest at the end. The numbers in the above figure denote the priorities, not the actual values.

While we insert values in the priority queue that is modeled using an array, we have to insert values in a sorted array. Here is the code in C to simulate a priority queue.

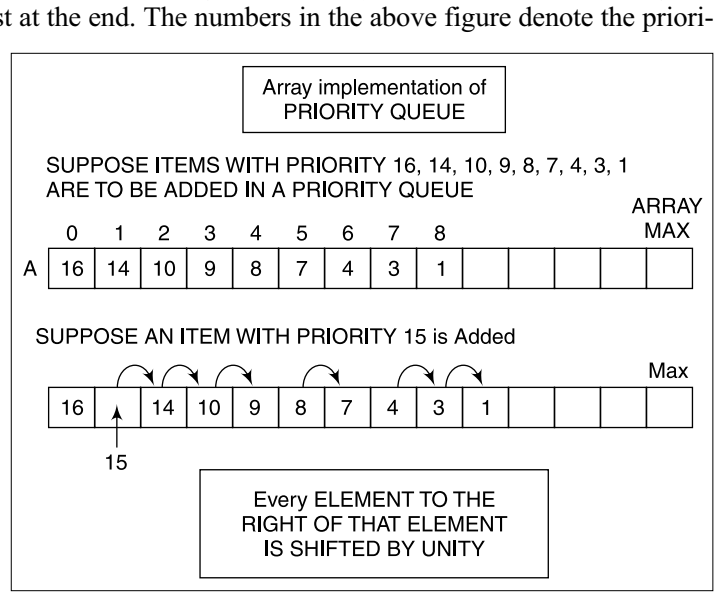


Fig. 7.11

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

#define MAX 10

enum{NOTFOUND,FOUND};
int search_status = NOTFOUND;
int r=0,f=0,c=0;

typedef struct MyQueue
{
      int front;
      int rear;
      int count;
      char MyList[MAX][MAX];
      int priority[MAX];//Priority associated with this element.
}MyQueue;


MyQueue *q;

void initQ(MyQueue *);
void appendQ(MyQueue *,char *,int);
void deleteQ(MyQueue *);
void searchQ(MyQueue *);
void displayQ(MyQueue *);
int isEmpty(MyQueue *);
int isFull(MyQueue *);

//Initializing the queue
void initQ(MyQueue *qu)
{
      qu->front = f;
      qu->rear = r;
      qu->count = c;
}
void prioritysort(MyQueue *qu)
{
      int k=0;
      char temp[MAX];
```

```
int tp=0;
int j;
for(j=0;j<qu->rear;j++)
{

      for(k=qu->front;k<qu->rear-1;k++)
      {
            if(qu->priority[k]<qu->priority[k+1])
            {
                  strcpy(temp,qu->MyList[k]);
                  strcpy(qu->MyList[k],qu->MyList[k+1]);
                  strcpy(qu->MyList[k+1],temp);

                  tp = qu->priority[k];
                  qu->priority[k] = qu->priority[k+1];
                  qu->priority[k+1]= tp;
            }
      }
}


}


void appendQ(MyQueue *qu,char *name,int priority)
{
      if(isFull(qu)==1)
            printf("Overflow Error!");
      else
      {
            strcpy(qu->MyList[qu->rear],name);
            qu->priority[qu->rear] = priority;
            r++;
            c++;
            qu->rear = r;
            qu->count = c;
            printf("\nSuccessfully Appended\n");
      }
      //Sort the elemnents according to descending priority
      prioritysort(qu);
}

void deleteQ(MyQueue *qu)
{
      if(isEmpty(qu)==1)
            printf("\nUnderflow error");
      else
      {
            f++;
            c--;
            qu->front = f;
            qu->count = c;
            printf("\nFirst person at the front
                  of the queue is serviced. Next please");
      }
}
```

```
int isEmpty(MyQueue *qu)
{
     if(qu->count==0)
          return 1;
     else
          return 0;
}

int isFull(MyQueue *qu)
{
     if(qu->count>=MAX)
          return 1;
     else
          return 0;
}

void displayQ(MyQueue *qu)
{
     int k;
     if(isEmpty(qu)==1)
          printf("\nSorry there is no one in the queue");
     else
     {
          for(k=qu->front;k<qu->rear;k++)
               printf("Name :<%s>     Position is Queue :%d  Priority :
%d\n",
               qu->MyList[k],k-qu->front+1,qu->priority[k]);
     }

}

void searchQ(MyQueue *qu)
{
     int k;
     char name[10];
     int occur=1;
     if(isEmpty(qu)==1)
          printf("Nobody is there in the queue");
     else
     {
          printf("Whom do you want to search :");
          fflush(stdin);
          gets(name);
          for(k=qu->front;k<qu->rear;k++)
               if(strcmpi(qu->MyList[k],name)==0)
               {
                    printf("Occurrence %d of %s is at position %d\n",
                         occur,name,k-qu->front+1);
                    occur++;
                    search_status=FOUND;
               }
               if(search_status==NOTFOUND)
                    printf("Nobody is there with this name
                              in the queue!");
     }
}
```

```
void main()
{

     int choice;
     char name[20];
     int pri=0;
     MyQueue mq;
     q=&mq;
     initQ(q);
     do
     {
          printf("1.Append\n");
          printf("2.Service\n");
          printf("3.Display\n");
          printf("4.Search\n");
          printf("5.Exit\n");
          printf("[1-5] : choice ");
          scanf("%d",&choice);
          switch(choice)
          {
          case 1:
               printf("Enter nickname of the person joining the
               queue :");
                    fflush(stdin);
                    gets(name);
                    printf("Enter priority :");
                    scanf("%d",&pri);
               appendQ(q,name,pri);break;
          case 2:deleteQ(q);break;
          case 3:displayQ(q);break;
          case 4:searchQ(q);break;
          case 5:exit(0);break;
          }
     }while(1);

}
```

7.16 MODEL A PRIORITY QUEUE USING A SINGLE LINKED LIST

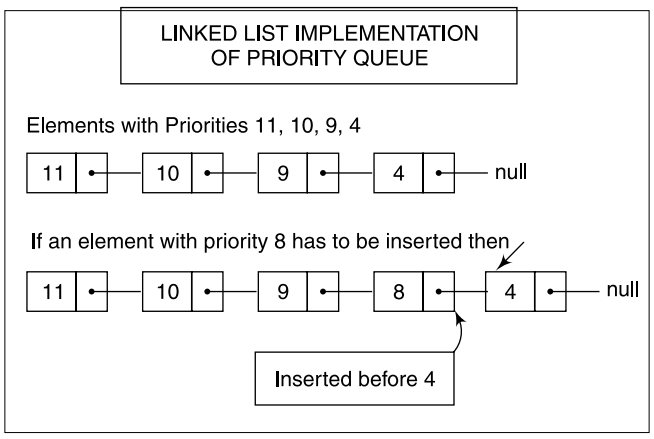


Fig. 7.12

Here is the complete code that creates and maintains a priority queue using a linked list.

//This program Implements a MAX-priority queue. A max priority queue is a priority queue //where the elements are stored in decreasing priority order.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
enum {NOTFOUND,FOUND};

//This structure denotes a particular element in the PQ.
typedef struct pq
{
      int key;
      int prio;
}pq;



//This structure denotes a node of the linked list that is used to
//represent the PQ.

typedef struct QueueNode
{
      pq data;
      struct QueueNode *next;
}QueueNode;

int count(QueueNode *h)
{
      int numberofQueueNodes=0;
      QueueNode *p = h;
      if(p==NULL)
            return 0;
      else
      {
            for(;p!=NULL;p=p->next)
                  numberofQueueNodes++;
            return numberofQueueNodes;
      }

}

QueueNode* push_back(QueueNode *last,pq info)
{
      if(last==NULL)
      {
            last = (QueueNode *)malloc(sizeof(QueueNode));
            last->data = info;
            last->next = NULL;
            return last;
      }
      else
```

```
{
      QueueNode *p = (QueueNode *)malloc(sizeof(QueueNode));
      if(p)
      {
            last->next = p;
            p->data = info;
            p->next = NULL;
            return p;
      }
}

QueueNode* push_front(QueueNode *h,pq info)
{
      QueueNode *p = (QueueNode *)malloc(sizeof(QueueNode));
      p->next = h;
      p->data = info;
      return p;
}



QueueNode* first(QueueNode *h)
{
      return h;
}
QueueNode* last(QueueNode *h)
{
      QueueNode *p = h;
      for(;p->next!=NULL;p=p->next);
      return p;
}
//inserting a QueueNode at a particular location
QueueNode* insert(QueueNode *h,int location,pq info)
{
      int c=0;
      QueueNode *p=h;
      QueueNode *r=p;
      QueueNode *q=(QueueNode *)malloc(sizeof(QueueNode *));
      if(location<count(h))
      {
            //Finding the address where to insert
            for(;p!=NULL;p=p->next)
            {
                  c++;
                  if(c==location)
                        break;
            }
            q->next = p->next;
            p->next = q;
            q->data = info;

      }
      return r;
}
```

```
//This function is same as dequeue function above.
QueueNode* pop_front(QueueNode *h)
{
      //identifying the first QueueNode that has to be freed
      QueueNode *x = h;
      QueueNode *p = h->next;
      free(x);//freeing the memory space.
      return p;
}

void display(QueueNode *h)
{
      QueueNode *p = h;
      for(;p!=NULL;p=p->next)
            printf("Value = %d Priority = %d\n",
      p->data.key,p->data.prio);

}

//This function find the maximum priority in the Priority Queue
int findmax(QueueNode *h)
{
      QueueNode *p;
      int max=0;
      p = h;
      max = p->data.prio;
      while(p!=NULL)
      {
            if(p->data.prio>max)
            {
                  max=p->data.prio;
            }
            p = p->next;
      }
      return max;
}
//This function find the minimum priority in the Priority Queue
//can be used to design a min priority queue
int findmin(QueueNode *h)
{
      QueueNode *p;
      int min=0;
      p = h;
      min = p->data.prio;
      while(p!=NULL)
      {
            if(p->data.prio<=min)
            {
                  min=p->data.prio;
            }
            p = p->next;
      }
      return min;
}
```

```
int showmenu()
{
      int choice=0;
      puts("1.Enqueue new element");
      puts("2.Display PQ");
      puts("3.Dequeue PQ");
      puts("4.Exit");
      puts("Your choice [1-4] :");
      scanf("%d",&choice);
      return choice;
}

int wheretokeep(QueueNode *h,pq newitem)
{
      QueueNode *p = h;
      int c=0;
      for(;p!=NULL;p=p->next)
      {
            c++;
            if(p->data.prio<newitem.prio)
                  break;
      }
      return c;
}


int main()
{
      QueueNode *a=NULL,*c;
      pq item;
      int key=0,prio=0;


      do
      {
            switch(showmenu())
            {
            case 1:puts("Enter new element and the priority :");
                  fflush(stdin);
                  scanf("%d %d",&key,&prio);
                  item.key = key;
                  item.prio = prio;
                  //The Bolded part below performs the insertion

                  //When the list is Empty, the node will be added at
                  //the end of the queue.
                  if(count(a)==0)
{
                        a = push_back(a, item);
                        c = a;
}
```

```
else //If the Queue is Not Empty
{
     //If the priority of the element to be
     //added falls between the minimum and maximum
     //priority of the queue
     if(item.prio>findmin(c) &&
          item.prio<findmax(c))//within the priority range
     {
               //wheretokeep(c,item) returns the index
               //of that item present in the list, whose
               //priority is just less than the priority
               //of the item to be added. So
               // wheretokeep(c,item)-1 denotes the location
               //identified to store the current element
               //being added
               c = insert(c,wheretokeep(c,item)-1,item);
               puts("The Queue is ");
     }

          //If the priority of the item being added falls
          //outside the present range of the priority queue
          else
          {
          //We are modeling a Max-Priority Queue. That means
          //that the element with highest priority will be
          //added at the front of the Queue and the element
          //with lowest priority is added at the end of the
          //queue. Findmax()
          //returns the maximum priority present in the list
          //Similarly findmin() returns the minimum priority
          //in the queue.
           if(item.prio>findmax(a))
             c = push_front(c,item);
           if(item.prio<=findmin(a))
             a = push_back(a,item);
          }
     }
     break;
case 2:  if(count(c)!=0)
              display(c);
          else
              puts("The Queue is Empty");
          break;
case 3:  c=pop_front(c);//dequeue
          break;
case 4:exit(0);
          break;
          }

    }while(1);
    return 0;
}
```

Here is a sample run of the program.

1. Enqueue new element

```
2. Display PQ
3. Dequeue PQ
4. Exit
Your choice [1-4] :
1
Enter new element and the priority :
11 10
1. Enqueue new element
2. Display PQ
3. Dequeue PQ
4. Exit
Your choice [1-4] :
1
Enter new element and the priority :
12 9
1. Enqueue new element
2. Display PQ
3. Dequeue PQ
4. Exit
Your choice [1-4] :
1
Enter new element and the priority :
14 6
1. Enqueue new element
2. Display PQ
3. Dequeue PQ
4. Exit
Your choice [1-4] :
2
Value = 11 Priority = 10
Value = 12 Priority = 9
Value = 14 Priority = 6
1. Enqueue new element
2. Display PQ
3. Dequeue PQ
4. Exit
Your choice [1-4] :
1
Enter new element and the priority :
20 8
The Queue is
1. Enqueue new element
2. Display PQ
3. Dequeue PQ
4. Exit
```

```
Your choice [1-4] :
3
1. Enqueue new element
2. Display PQ
3. Dequeue PQ
4. Exit
Your choice [1-4] :
2
Value = 12 Priority = 9
Value = 20 Priority = 8
Value = 14 Priority = 6
1. Enqueue new element
2. Display PQ
3. Dequeue PQ
4. Exit
Your choice [1-4] :
1
Enter new element and the priority :
20 10
1. Enqueue new element
2. Display PQ
3. Dequeue PQ
4. Exit
Your choice [1-4] :
2
Value = 20 Priority = 10
Value = 12 Priority = 9
Value = 20 Priority = 8
Value = 14 Priority = 6
```

Try Yourself: *Try to create a Min PQ (A queue where elements are stored in increasing order of their priority. The least priority element is stored at the front and the highest priority element is stored at the end of the PQ). Hint: Use findmin( ) function instead of findmax(), or you can just change some operators and it will be fine.*

7.17 AN APPLICATION OF PRIORITY QUEUE—*SCHEDULING APPOINTMENTS*

The priority queue described above is called an *explicit PQ* where we can assign some values as the priority of each object to be put in the queue. Some examples of this type of queue are a queue of print jobs for a shared printer, a queue of jobs where execution time is the deciding factor, etc.

In this above case, the priority can be the volume of the work. The program may decide to put the minimum volume of work at the front, or the maximum volume of work at the front.

In all these cases, we don't give priority explicitly. Rather the program uses some predictors to find out the location of the newly added object in the queue.

Here is an example of such an implicit PQ. This queue accepts few dates and appends them in increasing order (i.e. the earliest date comes first, the latest comes last).

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>

enum {NOTFOUND,FOUND};

typedef struct DateOfAppointment
{
   int day;
   int month;
   int year;
}DateOfAppointment ;


typedef struct Appointment
{
   DateOfAppointment key;

}Appointment;

typedef struct AppointmentNode
{
   Appointment data;
   struct AppointmentNode *next;
}AppointmentNode;

AppointmentNode *head=NULL;

int isPast(DateOfAppointment d1,DateOfAppointment d2)
{
   //returns 1 if d2 is in past that d1
   if(d1.year>d2.year || (d1.year==d2.year && d1.month>d2.month)
    ||(d1.year==d2.year && d1.month==d2.month && d1.day>d2.day))
      return 0;
   else
      return 1;
}

//this function returns 1 if both the dates are same
int isPresent(DateOfAppointment d1,DateOfAppointment d2)
{
   if(d1.day == d2.day && d1.month == d2.month && d1.year == d2.year)
      return 1;
   else
      return 0;
}

//this function returns 1 if d1 is at future of
int isFuture(DateOfAppointment d1,DateOfAppointment d2)
{
   if(!isPresent(d1,d2) && !isPast(d1,d2))
      return 1;
```

```
   else
      return 0;
}

int count(AppointmentNode *h)
{
   int numberofAppointmentNodes=0;
   AppointmentNode *p = h;
   if(p==NULL)
      return 0;
   else
   {
      for(;p!=NULL;p=p->next)
         numberofAppointmentNodes++;
      return numberofAppointmentNodes;
   }

}

AppointmentNode* push_back(AppointmentNode *last,Appointment info)
{
   if(last==NULL)
   {
      last = (AppointmentNode *)malloc(sizeof(AppointmentNode));
      last->data = info;
      last->next = NULL;
      return last;
   }
   else
   {
      AppointmentNode *p = (AppointmentNode
*)malloc(sizeof(AppointmentNode));
      if(p)
      {
         last->next = p;
         p->data = info;
         p->next = NULL;
      }
      return p;
   }
}

AppointmentNode* push_front(AppointmentNode *h,Appointment info)
{
   AppointmentNode *p = (AppointmentNode
*)malloc(sizeof(AppointmentNode));
   p->next = h;
   p->data = info;
   return p;
}

AppointmentNode* first(AppointmentNode *h)
{
   return h;
}
```

```
AppointmentNode* last(AppointmentNode *h)
{
   AppointmentNode *p = h;
   for(;p->next!=NULL;p=p->next);
   return p;
}

//inserting a AppointmentNode at a particular location
AppointmentNode* insert(AppointmentNode *h,int location,Appointment
info)
{
   int c=0;
   AppointmentNode *p=h;
   AppointmentNode *r=p;
   AppointmentNode *q=(AppointmentNode *)malloc(sizeof(AppointmentNode
*));
   if(location<count(h))
   {
      for(;p!=NULL;p=p->next)
      {
         c++;
         if(c==location)
            break;
      }
      q->next = p->next;
      p->next = q;
      q->data = info;

   }
   return r;
}

AppointmentNode* pop_front(AppointmentNode *h)
{
   AppointmentNode *x = h;//identifying the first AppointmentNode that
has to be freed
   AppointmentNode *p = h->next;
   free(x);//freeing the memory space.
   return p;
}

char* toMonth(int month)
{

   switch(month)
   {
   case 1:return "Jan";
      break;

   case 2:return "Feb";
      break;

   case 3:return "Mar";
      break;
```

```
case 4:return "Apr";
   break;

case 5:return "May";
   break;

case 6:return "Jun";
   break;
   case 7:return "Jul";
      break;

   case 8:return "Aug";
      break;

   case 9:return "Sep";
      break;

   case 10:return "Oct";
      break;

   case 11:return "Nov";
      break;

   case 12:return "Dec";
      break;
   }
}

void display(AppointmentNode *h)
{
   AppointmentNode *p = h;
   int serial = 1;
   puts("Appointment Schedule >>");
   for(;p!=NULL;p=p->next)
   {

      printf("Appointment #%d :  On %s - %d - %d\n",
         serial,toMonth(p->data.key.month),
         p->data.key.day,
         p->data.key.year);
               serial++;
   }

}

int showmenu()
{
   int choice=0;
   puts("1.Enter new Date Of Appointment");
   puts("2.Display Appointment List");
   puts("3.Delete the first Appointment(Which you have already
attended)");
```

```
   puts("4.Exit");
   puts("Your choice [1-4] :");
   scanf("%d",&choice);
   return choice;
}

int whereretokeep(AppointmentNode *h,Appointment newappointment)
{
   AppointmentNode *p = h;
   int c=0;
   for(;p!=NULL;p=p->next)
   {
      c++;
      if(isPresent(newappointment.key,p->data.key))
      {
         c = 0;
         break;
      }
      if(isPast(newappointment.key,p->data.key))
         break;
   }
   return c;
}


int main()
{
   AppointmentNode *a=NULL,*c;
   Appointment item;
   DateOfAppointment temp;
   int index = 0;

   int key=0,prio=0;
   do
   {
      switch(showmenu())
      {
      case 1:puts("Enter Date Of Appointment [dd - mm - yyyy ] :");
            fflush(stdin);
            scanf("%d %d %d",&temp.day,&temp.month,&temp.year);
            item.key = temp;
            //Assuming that the date entered is valid
            if(count(a)==0)
            {
               a = push_back(a,item);
               c = a;
}
else
{
   if(isPresent(item.key,first(c)->data.key)
   ||isPresent(item.key,last(c)->data.key))
```

```
{
     puts("ERROR : AN APPOINTMENT
                ALREADY EXISTS ON THAT DATE");
      break;
 }
if(isPast(item.key,first(c)->data.key))
{
    c = push_front(c,item);
                   break;
                }

                if(isFuture(item.key,last(c)->data.key))
                {
                    c = push_back(c,item);
                   break;
                }
                else
                {
                   index = wheretokeep(c,item);
                   if(index!=0)
                     c = insert(c,wheretokeep(c,item)-1,item);
                   else
                     puts("ERROR : AN APPOINTMENT
                                   ALREADY EXISTS ON THAT DATE");
                   break;
                }
              }
              break;
      case 2: display(c);
               break;
      case 3:c=pop_front(c);
              break;
      case 4:exit(0);
              break;
      }

   }while(1);
   return 0;
}
```

Here is the output of the program.

```
1. Enter new Date Of Appointment
2. Display Appointment List
3. Delete the first Appointment(Which you have already attended)
4. Exit
Your choice [1-4] :
1
Enter Date Of Appointment [dd - mm - yyyy ] :
11 1 2009
1. Enter new Date Of Appointment
2. Display Appointment List
```

```
3. Delete the first Appointment(Which you have already attended)
4. Exit
Your choice [1-4] :
1
Enter Date Of Appointment [dd - mm - yyyy ] :
31 12 2008
1. Enter new Date Of Appointment
2. Display Appointment List
3. Delete the first Appointment(Which you have already attended)
4. Exit
Your choice [1-4] :
2
Appointment Schedule >>
Appointment #1 : On Dec - 31 - 2008
Appointment #2 : On Jan - 11 - 2009
1. Enter new Date Of Appointment
2. Display Appointment List
3. Delete the first Appointment(Which you have already attended)
4. Exit
Your choice [1-4] :
1
Enter Date Of Appointment [dd - mm - yyyy ] :
5 1 2009
1. Enter new Date Of Appointment
2. Display Appointment List
3. Delete the first Appointment(Which you have already attended)
4. Exit
Your choice [1-4] :
2
Appointment Schedule >>
Appointment #1 : On Dec - 31 - 2008
Appointment #2 : On Jan - 5 - 2009
Appointment #3 : On Jan - 11 - 2009
1. Enter new Date Of Appointment
2. Display Appointment List
3. Delete the first Appointment(Which you have already attended)
4. Exit
Your choice [1-4]:
```

Try Yourself: *This program doesn't check whether the entered date is a valid date or not and whether it is in past or not. Try to protect accepting such inputs.*

Notice : How the **`wheretokeep()`** *function is changed here in this code.*

7.18 HOW TO MODEL A DEQUE (DOUBLE-ENDED QUEUE) USING A LINKED LIST

All the operations that are discussed above for the queue are also applicable for deque. The only additional operation that is possible on this ADT is the insertion at the front. Here is the code to push an element at the front.

```
QueueNode* push_front(QueueNode *h,pq info)
{
      QueueNode *p = (QueueNode *)malloc(sizeof(QueueNode));
      p->next = h;
      p->data = info;
      return p;
}
```

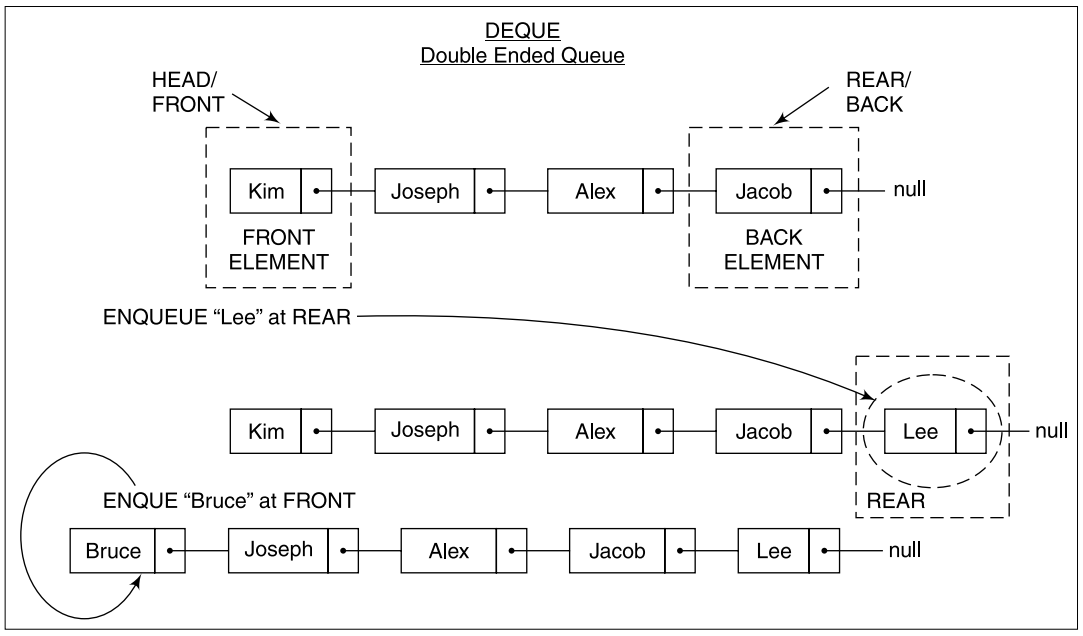


Fig. 7.13

7.19 HOW TO MODEL A MOVE TO FRONT LIST (MTFL) USING A QUEUE

We have discussed MTFL at length in the Stack chapter. Using linear queue we can design the MTFL structure easily. All the other functions defined above for queue will be applicable for MTFL also, but only the search routine will be different. It involves two steps.

1. To find out the index of the variable.
2. Delete it from its current location.
3. To put the element in front of the queue.

Thus this search, unlike search on other data structures is an example of active function. Here is the code that describes what has to be done when the item with value 20 is sought in the list [*Let's assume that we are performing the search on an integer queue*.]

```
//Here c denote the head/front of the queue
int loc = searchindex(c,20);
if(loc!=-1)
{
      //delete the item from its original location in the list
      c = delete_at(c,loc);
      //put it at the front of the list
      c = push_front(c, 20);
}
```

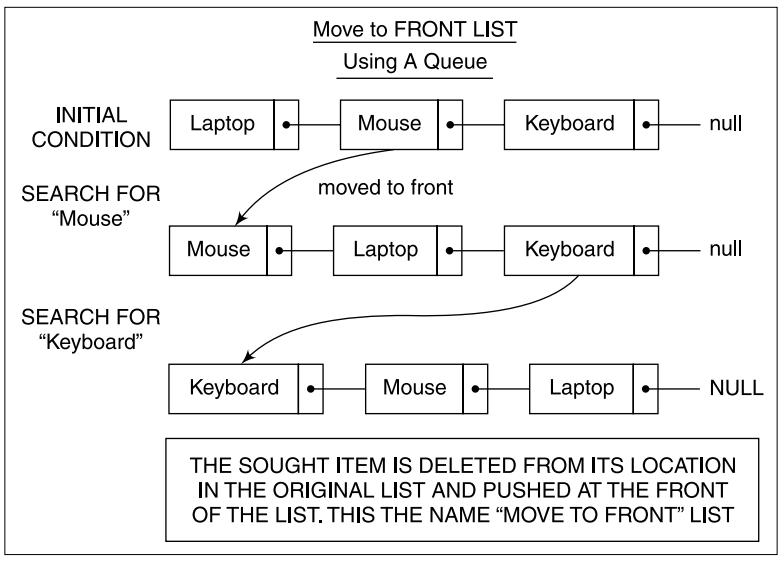


Fig. 7.14

7.20 HOW TO SIMULATE THE QUEUE IN FRONT OF THE CASH COUNTER

In real life, wherever there is a server and more than one clients/customers/objects/requests to be served, then a queue is created in front of the server. In case of a bank, the server may be the bank officer (cashier in this case) and the clients who want to deposit/withdraw money are elements of the queue.

See the figure below:

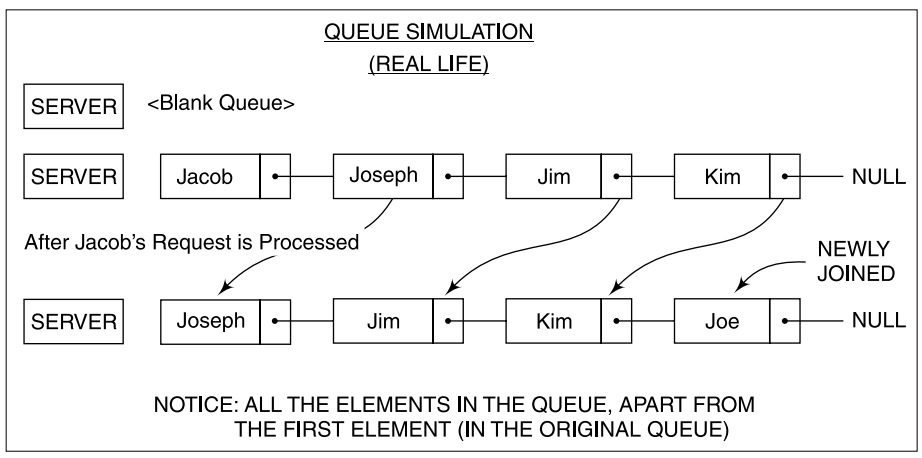


Fig. 7.15

As shown in the figure, the first person to be served is "Jacob", next is "Joseph" and so on. When Jacob's request is served by the server (Cashier) then Jacob moves out of the list and "Joseph" becomes the first in the list. Notice that customers join the queue at the end.

As the queue grows and diminish in run time, so the real life queue simulations are always done using queues which are modeled using linked lists.

Here is a code that simulates the queue in a bank. and stores the simulation result in a file queuesim.txt

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

//Global Variables that affects the queue simulation
#define MAX_QUEUE_LENGTH 25
#define SERVER_NUMBERS 6



enum
RequestType{DEPOSIT,WITHDRAW,BALANCEENQUIRY,DRAFTMAKING,BILLPAYMENT,DIS
CUSSISSUES};
enum TypeOfQueue{FIFO,PRIO,RANDOM};

FILE *fp;

typedef struct customer
{
      char name[20];
      int position;//position in quee
      float waittime;
      enum RequestType request;
    int whichservertogo;
}customer;

typedef struct server
{
      char name[20];
      float servicetime;
      int servicerate;
      enum RequestType dealswith;
    int lengthofawaitingqueue;
}server;


server cashiers[10];
//For 10 Servers, each server will serve a maximum of 25 customers
customer Customers[MAX_QUEUE_LENGTH][SERVER_NUMBERS];
server Servers[SERVER_NUMBERS];

int WhereWillNewCustomerGo(int req)
{
int serverno=0;
int i=0;
for(i=0;i<SERVER_NUMBERS;i++)
{


      if(cashiers[i].dealswith == req)
```

```
            {
                  serverno = i;
                  break;
            }
      }
      return serverno;
}

char* request2string(int x)
{
      switch(x)
      {

      case 0:return "DEPOSIT";break;
      case 1:return "WITHDRAW";break;
      case 2:return "BALANCEENQUIRY";break;
      case 3:return "DRAFTMAKING";break;
      case 4:return "BILLPAYMENT";break;
      case 5:return "DISCUSSISSUES";break;
      default:return "DEPOSIT";break;
      }
}

void CreateCustomer(int howmanyjoin)
{

      int i=0;
      int lastinthequeue=0;


      for(i=0;i<rand()%howmanyjoin+1;i++)
      {
            customer newcustomer;

            newcustomer.request = rand()%6;
      newcustomer.whichservertogo =
WhereWillNewCustomerGo(newcustomer.request);

      cashiers[newcustomer.whichservertogo].lengthofawaitingqueue++;
            newcustomer.position = lastinthequeue;
            fprintf(fp,
   "New Customer Request is %s and joined at the end of Queue #%d\n",

      request2string(newcustomer.request),newcustomer.whichservertogo);
      }
}

void ProcessCustomer()
{
      int i;
      //for(i=0;i<3000;i++);
      for(i=0;i<SERVER_NUMBERS;i++)
      {
            if(cashiers[i].lengthofawaitingqueue>=1)
                  cashiers[i].lengthofawaitingqueue--;
      }
}
```

```
void ShowCustomerQueues()
{
     int i;
     for(i=0;i<SERVER_NUMBERS;i++)
     {
          fprintf(fp,"cashier[%d] Deals With %s Queue Length [%d] \n",
             i,request2string(cashiers[i].dealswith),
             cashiers[i].lengthofawaitingqueue);
     }
     fprintf(fp,"-----------------------------------------\n");


}


int main()
{
     int i;
     int c=0;
     int howmanyjoin=0;

     printf("How many customers join the queue :");
     scanf("%d",&howmanyjoin);
     fp = fopen("D:\\queuesim.txt","a");
     //Server initializations
     srand((unsigned)time(NULL));
     for(i=0;i<SERVER_NUMBERS;i++)
     {

            cashiers[i].dealswith = i;
            puts(request2string(i));
               cashiers[i].lengthofawaitingqueue = 0;
      }
     srand((unsigned)time(NULL));
     do
     {
          CreateCustomer(howmanyjoin);
          ProcessCustomer();
          ShowCustomerQueues();
          c++;
     }while(c!=20);
     fclose(fp);
     return 0;
}
```

Here is a sample partial output stored in the output file:

```
New Customer Request is DISCUSSISSUES and joined at the end of Queue #5
New Customer Request is DISCUSSISSUES and joined at the end of Queue #5
New Customer Request is DRAFTMAKING and joined at the end of Queue #3
New Customer Request is BALANCEENQUIRY and joined at the end of Queue #2
New Customer Request is BILLPAYMENT and joined at the end of Queue #4
New Customer Request is BALANCEENQUIRY and joined at the end of Queue #2
cashier[0] Deals With DEPOSIT Queue Length [0]
```

```
cashier[1] Deals With WITHDRAW Queue Length [0]
cashier[2] Deals With BALANCEENQUIRY Queue Length [1]
cashier[3] Deals With DRAFTMAKING Queue Length [0]
cashier[4] Deals With BILLPAYMENT Queue Length [0]
cashier[5] Deals With DISCUSSISSUES Queue Length [1]
-----------------------------------------------------------
New Customer Request is DEPOSIT and joined at the end of Queue #0
New Customer Request is DISCUSSISSUES and joined at the end of Queue #5
New Customer Request is DEPOSIT and joined at the end of Queue #0
New Customer Request is WITHDRAW and joined at the end of Queue #1
cashier[0] Deals With DEPOSIT Queue Length [2]
cashier[1] Deals With WITHDRAW Queue Length [0]
cashier[2] Deals With BALANCEENQUIRY Queue Length [0]
cashier[3] Deals With DRAFTMAKING Queue Length [0]
cashier[4] Deals With BILLPAYMENT Queue Length [0]
cashier[5] Deals With DISCUSSISSUES Queue Length [1]
-----------------------------------------------------------
New Customer Request is DISCUSSISSUES and joined at the end of Queue #5
New Customer Request is DISCUSSISSUES and joined at the end of Queue #5
New Customer Request is DEPOSIT and joined at the end of Queue #0
cashier[0] Deals With DEPOSIT Queue Length [2]
cashier[1] Deals With WITHDRAW Queue Length [0]
cashier[2] Deals With BALANCEENQUIRY Queue Length [0]
cashier[3] Deals With DRAFTMAKING Queue Length [0]
cashier[4] Deals With BILLPAYMENT Queue Length [0]
cashier[5] Deals With DISCUSSISSUES Queue Length [2]
```

REVISION OF CONCEPTS

Some Key Facts about Queues and Terminologies

1. FIFO : First In First Out
2. LILO : Last In Last Out
3. FCFS : First Come First Serve
4. FCLS : First Come Last Serve
5. LCFS : Last Come First Serve
6. LCLS : Last Come Last Serve
7. RSO : Random Service Ordering
8. By queue we normally mean a FIFO/LILO/FCFS list that may not be always true. Queue elements can be processed by any of the above technique and then the name of the queue will be prefixed with that acronym, for example if the last element is processed first in a queue [which is nothing

but a stack] then we can call it a LCFS/LIFO Queue. *From now on whenever we write queue, we mean the normal queue,* [i.e the FCFS/FIFO queue]

9. Deletion of items is only allowed in front of a queue.
10. Elements can only be added at the end of the queue.
11. Addition of elements is called Enqueue.
12. Queues can be implemented using either array or linked list. If we use array to create a queue the pre-condition for the enqueue operation is to check whether the array is full or not. When we define the queue using linked list, then this pre-operation is not needed.
13. Deletion of element at the front is called Dequeue.
14. No matter which data structure we use array or linked list to define a queue, before dequeue operation, it should be checked, whether the queue is empty or not. If the queue is empty the dequeue operation will fail.
15. Priority queue is a queue where elements are processed in order of their priority.
16. A Max priority queue is one where the element with highest priority is deleted/serviced first.
17. A Min priority queue is one where the element with lowest priority is deleted/serviced first.
18. A Priority Queue can be easily implemented using a heap data structure.
19. Priority Queues are used for *Discrete Event Simulations*.
20. A dequeue is a double ended queue, where deletions and insertion are allowed at both end.
21. Queues are used in operating systems, for controlling access to shared system resources such as bandwidth, routers, printers, files, communication lines, disks and tapes.
22. Queues are also used for simulation of real-world situations. For instance, a new bank may want to know how many tellers to install. The goal is to service each customer within a reasonable wait time, but not have too many tellers for the number of customers. To find out a good number of tellers, they can run a computer simulation of typical customer transactions using queues to represent the waiting customers.

REVIEW QUESTIONS

1. Apart from enqueue and dequeue operations can you think of other type of branching operations in a queue.
2. Let there be a queue of names. Write the code for retrieving the first name and the last name.
3. An integer queue is there and the operations enqueue(q,item) and dequeue() operations are defined on it. Can you say what will be the contents of the queue after the following commands

   Enqueue(q,3) ; enqueue(q,5),enqueue(q,isEmpty()), dequeue()

4. typedef struct MyQueue
   {
   MyQueue *next;
   int value;
   }MyQueue; What is wrong in this declaration.
5. Define a structure called student, and then create a queue of 10 such students.

PROGRAMMING PROBLEMS

1. Write a program using queue to check whether a string is a palindrome or not. Use the same test strings as given in stack chapter's exercise.
2. Simulate a queue of persons in front of a ticket counter. Assume the following things. Any number of people may join at the end of the queue. So it will be better if you use a linked list implementation. Time taken by the counter boy is one minute per person. Ask the user for how many minutes the simulation should run. After each minute print how many people are there currently in the queue and what is the average and worst wait time in the queue. Use pseudorandom numbers to populate the queue.
3. Imagine yourself waiting for a movie ticket at the end of a long queue. Suddenly some people in front of you got irritated by waiting and they decided to leave. So you get a chance to go ahead. Write a function which simulates such a situation. Which type of queue implementation will you prefer if you have to model this? Linked list or array implementation? Justify your answers.
4. Write a program that accepts a queue of names and returns the sorted names alphabetically.
5. Write a complete MTF implementation using Queues. You can take the help of the code written in the chapter.
6. A certain protocol (SPP) sends packets to the end workstations. Assume that the source speed is 10 times more than that of the processing speed of the workstations. That means at a particular point of time each workstation gets 10 times more packet than they can process. Thus, these excess packets are overhead to the system. Assume that one source of SPP packets are being served by 5 workstations of similar capacity and processing time. Simulate such a situation. The objective of this is to find out how many more workstations will be needed to assure that no packet wait at all. This can be particularly useful for a cyber café.

8

Trees

Explorer to Genetics!

INTRODUCTION

Some Key Facts about Trees and Jargons

A Binary Tree

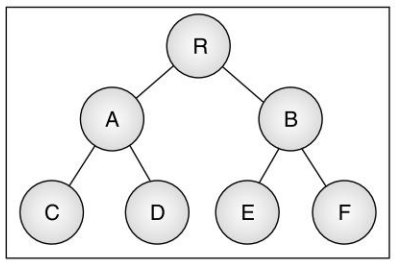


Fig. 8.1

- ***Tree:*** A data structure that matches the shape of a tree.
- ***Node:*** Where the data in a tree structure are kept is called a node.
- ***Root:*** The node from where two subsections (For Binary Tree)/M-Subsections (for M-are tree) are connected. This is the node from where the tree starts. Surprisingly this node is drawn at the top. That means the tree data structure has been drawn like a tree upside down.
- ***Level:*** The level of a node. Starting from root which has a level 0
- ***Height:*** The length of the longest path from root to a leaf. Therefore, the leaves are all at height zero. Height and level are antonyms as you can probably see from their definitions.
- ***Degree:*** The highest level in a tree. Degree and height are the same.
- ***Parent:*** A node's predecessor from which it is generated. It is one level up in the tree.
- ***Ancestor:*** Synonym of parent.
- ***Child:*** A node which resides in a level below it's parents.
- ***Left Sub Tree:*** Binary tree is a recursive structure. The left of root is also a binary tree and is known as left sub-tree. This is valid for any node that has got a left child.
- ***Right Sub Tree:*** Binary tree is a recursive structure. The right of root is also a binary tree and is known as right sub-tree. This is valid for any node that has got a left child.
- ***Left Child:*** The child to the left of the node seen from the viewer.

- ***Right Child**:* The child to the right of the node seen from the viewer.
- ***Inorder Predecessor:*** While traversing the tree inorder, the node that comes before this one.
- ***Inorder Successor:*** While traversing the tree inorder, the node that comes after this one.
- ***Preorder Predecessor:*** While traversing the tree pre-order, the node that comes before this one.
- ***Preorder Successor:*** While traversing the tree pre-order, the node that comes after this one.
- ***Postorder Predecessor:*** While traversing the tree post-order, the node that comes before this one.
- ***Postorder Successor:*** While traversing the tree post-order, the node that comes after this one.
- ***Sisters***: Nodes in the same level are known as sisters.
- ***Brothers***: Synonymous to Sisters.
- ***Siblings***: Synonymous to Brothers/Sisters.
- ***Cousin***: Nodes who are in the same level but who has different parents.
- ***Uncle***: In at least a level 2 complete binary tree, the cousin of parent is known as uncle to the child nodes.
- ***Descendent***: Synonym of Child.
- ***Grandchild***: Child node of Child node.
- ***Grandparent***: Ancestor of parent.
- ***An External Node***: A node that has no children.
- ***An Internal Node:*** A node that has 2 children.
- ***Leaf***: A node that has no child. An external node.
- ***Binary***: A tree in which each node at the maximum can only have two children. From these children, again other two nodes from each of these nodes can be possible.
- ***M-ary***: A tree where each node can have a maximum of *M* nodes. Binary tree is a special case where $M = 2$
- In a binary tree of *L* levels the maximum number of nodes are given by

$$N = \sum_{k=0}^{L} 2\text{^}k$$

- In a binary tree of height *H* the maximum number of nodes are given by

$$N = \sum_{k=0}^{H-1} 2\text{^}k$$

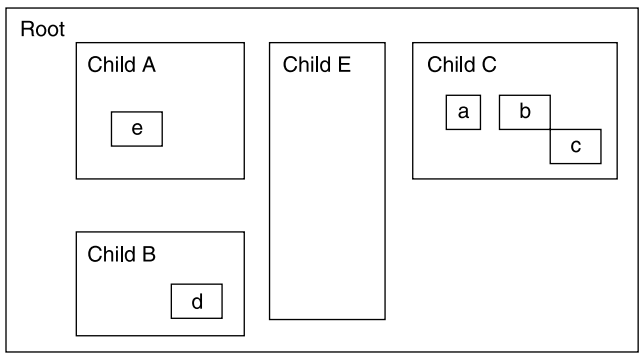
- For a binary tree of height *h* and nodes *n* $h <= n <= 2\text{^}h - 1$

In the above picture a binary tree is shown. From the above picture we can conclude the following statements:

- R is the root of the tree.
- A is the left child of the tree.
- B is the right child of the tree.
- C is the left child of the node A.
- D is the right child of the node A.
- E is the left child of the node B.
- F is the right child of the node B.
- A and B are sisters/brothers/siblings.
- C and D are sisters/brothers/siblings.
- E and F are sisters/brothers/siblings.
- Level of R is 0.

- Level of A and B is 1.
- Level of C and D is 2.
- The leaves are A, C and D because they don't have any child.
- Height of the tree is 2.
- Degree of the tree is 2.
- A is C and D's grandparent.
- C and D are A's grandchildren.

8.1 WHAT ARE THE DIFFERENT WAYS TREES ARE REPRESENTED?



Representing trees with box/Venn Diagrams

A
B
C
D
E
F
G
K
L
M
H
I
J

Representing Tree using Indentation

8.2 WHAT IS A STRICTLY BINARY TREE?

A binary tree is called a strictly binary tree if every non-leaf node in a binary tree has non-empty left and right sub-trees.

8.3 WHAT IS AN ALMOST COMPLETE BINARY TREE?

A binary tree of depth *d* is an ***almost complete binary tree*** if

- Any node at level less than $d - 1$ has two daughters
- For any node ***nd*** in the tree with a right descendent at level, ***nd*** must have a left daughter and every left descendent of ***nd*** is either a leaf at level d or has two daughters.

Here is an almost complete binary tree.

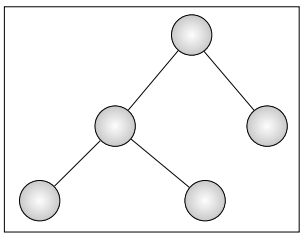


Fig. 8.2

8.4 WHAT IS A COMPLETE BINARY TREE? (*ALSO KNOWN AS* PERFECT BINARY TREE)

A *binary tree* in which every level, except possibly the deepest, is completely filled. At depth *n*, the *height* of the tree, all *nodes* must be as far left as possible.

In other words, A complete binary tree of depth d is a strictly binary tree all of whose leaves are at level d.

Here is a complete binary tree. The second picture is deliberately used to eradicate the misconception that binary trees need always be drawn like the first one. You should understand that its not, how it looks, rather how conceptually oriented the tree is.

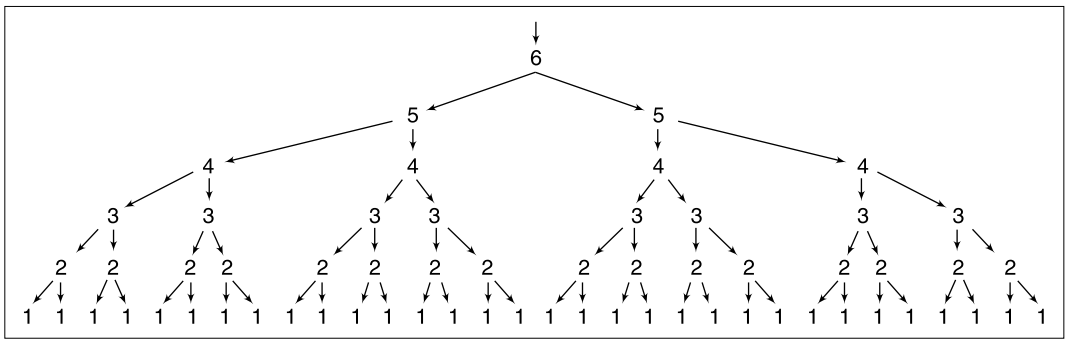


Fig. 8.3 *A typically drawn complete binary tree*

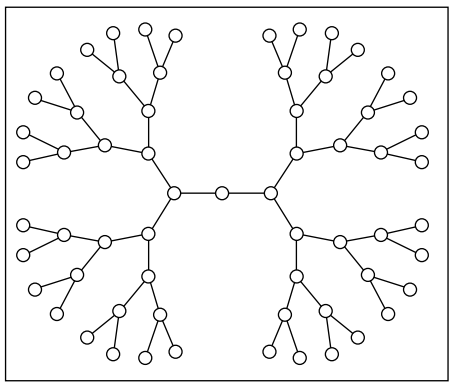


Fig. 8.4 *A Not- So-Typically (Yet Conceptually Correct) Drawn Complete Binary Tree*

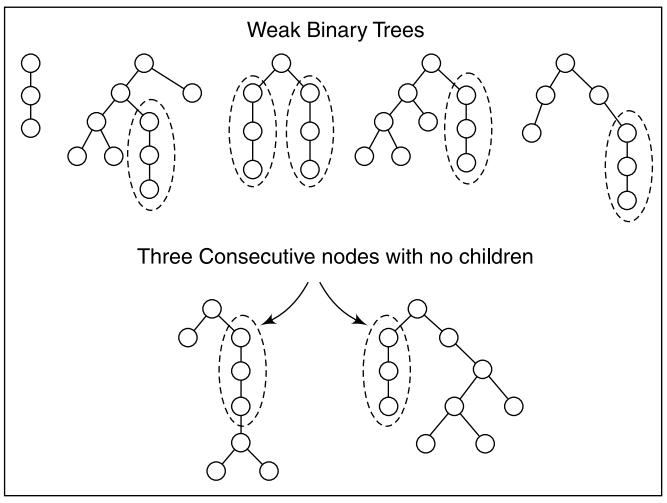


Fig. 8.5

Another way to define a complete binary tree geometrically is to state that a complete binary tree is a binary tree for which if a circle is drawn with a radius of any path length, it will meet all the leaves of the tree.

8.5 WHAT IS A WEAK BINARY TREE?

A weak binary tree is a binary tree that has got at least three or more nodes either directly beneath root (Without any child) or in any of the sub trees.

Here are a few Weak Binary Trees

8.6 WHAT IS A STRONG BINARY TREE?

A tree can be either rooted or not rooted. In case of a not rooted the root is the only node of the tree and

does not have any child. A Binary tree where the root is either childless or has a couple of them, then the tree is known as a strong binary tree. In the figure below, there are some strong binary trees.

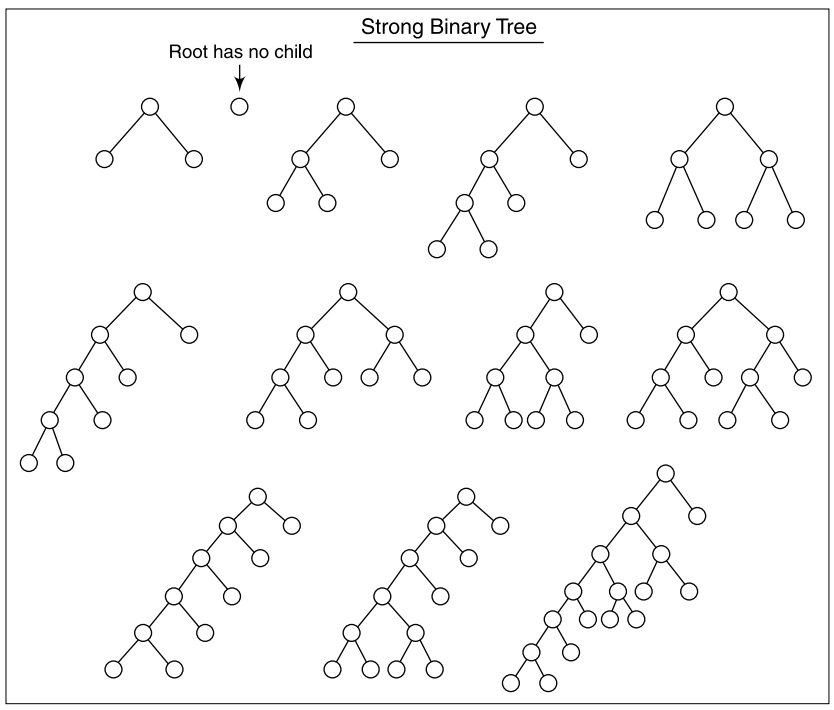


Fig. 8.6

Some Useful Relations

- In a complete binary tree of degree n there are 2^j nodes in each level where j runs from 0 to the degree of the tree.
- Total number of nodes in a complete binary tree is given by $2^{(d+1)} - 1$ where d is the degree of the tree.

8.7 HOW TO MODEL A BINARY TREE USING AN ARRAY

Here a binary tree of Integers is modeled using an array of Integers.

The root of the binary tree is assumed to be the first element of the array. And the following two theorems are followed to calculate the child locations.

- The left child will be at a location $2*j + 1$ where j is the location of the main root of the binary tree or any sub-tree of it.
- The right child will be at a location $2*j + 2$ where j is the location of the main root of the binary tree or any sub-tree of it.

How to Add the Root Value

```
void AddRoot(int bt[],int rootvalue)
{
     bt[0]=rootvalue;
}
```

How to Add a Left Child to a Node

```
void AddLeftChild(int bt[], int j,int value)
//whose left child and what is the value of that
{
       bt[ 2*j + 1 ] = value;
}
```

How to Add a Right Child to a Node

```
void AddRightChild(int bt[], int j,int value)
//whose left child and what is the value of that
{
       bt [ 2*j + 2 ] = value;
}
```

How to Find the Parent Location from a Left Child

```
int GetFatherLocationFromLeftChild(int bt[],int loc)
{
      return floor((double)(loc-1)/2);
}
```

How to Find the Parent Location from a Right Child

```
int GetFatherLocationFromRightChild(int bt[],int loc)
{
     return floor((double)(loc-2)/2);
}
```

How to Find if a Child Node is Alone or Not

```
int IsAlone(int bt[],int j)
{
     int alone = NO;
     if(j%2==0)//if the number is in a even location
     //Assuming a positive number Binary Tree
          if(bt[j-1]==0)
               alone = YES;
          if(j%2!=0)//if the number is in a odd location
          //Assuming a positive number Binary Tree
               if(bt[j+1]==0)//0 means that location is blank
                    alone = YES;
     return alone;
}
```

How to Find the Sister/Sibling's Location

```
int GetSisterLocation(int bt[],int j)
{
     if(j%2==0)//if the number is in a even location
          if(IsAlone(bt,j)==NO)
            return j-1;
     else
          if(IsAlone(bt,j)==NO)
            return j+1;
     else
          return -1;//No Sister/Brother/Sibling
}
```

How to Find the Level of a Node in the Binary Tree

```
//Assuming No Duplicates
int GetLevel(int bt[],int size,int no)
{
     int i;
     for(i=0;i<size;i++)
          if(bt[i]==no)
            break;
          if(i%2!=0)
     //Because in the array implementation of binary tree the
     //Right children are always located at the odd places
          return GetFatherLocationFromRightChild(bt,i);
     else
          return GetFatherLocationFromLeftChild(bt,i);

}
```

How to Find Degree of the Binary Tree

```
int Degree(int bt[],int size)
{
//if we get 2 consecutive zero we know
//that we are at the end of the bt.
     int i = 0;
     for(i=0;i<size;i++)
          if(bt[i+1]==0 && bt[i+2]==0)
               break;
     if(i==0)//Root
          return 0;
     else
          return GetLevel(bt,10,bt[i]);
}
```

How to Find the Location of a Number in the Tree

```
int GetLocation(int bt[],int size,int value)
{
     int i=0;
     for(i=1;i<size;i++)
          if(bt[i]==value)
               break;
     return i;
}
```

How to Count the Number of Nodes in the Tree

```
int CountNodes(int bt[],int size)
{
     int i=0;
     int count=0;
     for(i=0;i<size;i++)
     {
          count++;
          if(bt[i+1]==0 && bt[i+2] == 0)
```

```
                break;
    }
    return count;
}
```

How to Find Number of Children of a Node

```
int CountChildren(int bt[],int j)
{
          if(bt[2*j + 1]==0 && bt[2*j+2]==0)
               return 0;
          if((bt[2*j+1]!=0 && bt[2*j+2]==0)
             || (bt[2*j+1]==0 && bt[2*j+2]!=0))
               return 1;
          if(bt[2*j+1]!=0 && bt[2*j+2]!=0)
               return 2;
}
```

How to Count Number of Leaves in the Tree

```
int CountLeaves(int bt[],int size)
{
     int i=0;
     int leaves = 0;
     for(i=0;i<size;i++)
     {
          if(CountChildren(bt,i)==0 && bt[i]!=0)
               leaves++;

     }
     return leaves;
}
```

How to Find the Number of Parent Nodes that has a Single Child

```
int CountParentsWithSingleChildren(int bt[],int size)
{
      int i=0;
      //The edge that connects such a parent to it's only //child looks
      like a stray brunch of the tree.
      int branches = 0;
      for(i=0;i<size;i++)
      {
            if(CountChildren(bt,i)==1)
                  branches++;
      }
      return branches;
}
```

How to Find the Left Child Value for a Node

```
int GetLeftChild(int bt[],int value)
{
     return GetValue(bt,2*GetLocation(bt,10,value)+1);
}
```

How to Find the Right Child Value for a Node

```
int GetRightChild(int bt[],int value)
{
     return GetValue(bt,2*GetLocation(bt,10,value)+2);
}
```

How to Find whether There is a Left Child for a Node

```
int IsThereALeftChild(int bt[],int value)
{
     int left  = GetLeftChild(bt,value);
     if(left!=0)
          return YES;
     else
          return NO;
}
```

How to Find whether a Node is Left Child of Its Parent

There can be couple of implementation possible. The first one takes two arguments. One node is the address and the other is the parent address. Here is the code for this algorithm.

```
int isLeftChild(node *n,node *p)
{
      return p->leftchild == n;
}
```

The other one is more smart. Here we are assuming that the node contains a pointer to the node which is its parent. Here is the implementation.

```
typedef struct node
{
     int data;
     struct node *leftchild;
     struct node *rightchild;
     struct node *parent;
}node;

int isLeftChild(node *n)
{
     return n->parent->leftchild == n;
}
```

The later version is my favourite, because it is much more readable and conceptual in nature.

8.8 HOW TO FIND WHETHER A NODE IS A RIGHT CHILD OF ITS PARENT

There can be couple of implementation possible. The first one takes two arguments. One node address and the other is the parent address. Here is the code for this algorithm.

```
int isRightChild(node *n,node *p)
{
      return p->rightchild == n;
}
```

The other one is more smart. Here we are assuming that the node contains a pointer to the node which is its parent. Here is the implementation.

```
typedef struct node
{
     int data;
     struct node *leftchild;
     struct node *rightchild;
     struct node *parent;
}node;

int isRightChild(node *n)
{
     return n->parent->rightchild == n;
}
```

How to Find whether There is a Right Child for a Node

```
int IsThereARightChild(int bt[],int value)
{
      int right  = GetRightChild(bt,value);
      if(right!=0)
            return YES;
      else
            return NO;
}
```

How to Travel to the Left Child's Location

```
int GoToLeftChildsPlace(int parent)
{
     return 2*parent + 1;
}
```

How to Travel to the Right Child's Location

```
int GoToRightChildPlace(int parent)
{
      return 2*parent + 2;
}
```

How to Model a Binary Tree using Linked List

There are two ways to represent a binary tree using a linked list. The first one modeled by the following structure allows only one way traffic through the node. That means we can only travel from parent to the child. There is no way we can travel back to parents and grandparents.

```
typedef struct node
{
     int data;
     struct node *leftchild;
     struct node *rightchild;
}node;
```

Adding one more pointer to the above structure can make it two way traffic.

```
typedef struct node
{
     int data;
```

```
     struct node *leftchild;
     struct node *rightchild;
     struct node *parent;
}node;
```

The root has no parent. The rest all nodes in the binary tree has their parents.

How to Add a Left Child Node to any Node

```
void AddLeftChild(node *n,int value)
{
     node *r = (node *)malloc(sizeof(node));
     r->data = value;
     r->leftchild = NULL;
     r->rightchild = NULL;
     n->leftchild = r;
}
```

How to Add a Left Child Node to Any Node

```
void AddRightChild(node *n,int value)
{
     node *r = (node *)malloc(sizeof(node));
     r->leftchild = NULL;
     r->rightchild = NULL;
     r->data = value;
     n->rightchild = r;
}
```

8.9 HOW TO FIND THE ADDRESS OF THE SIBLING OF A NODE

If we are trying to get the sibling location from the left side of a node, then we need to get the location of the right child of the parent node.

```
node* Sibling(node *n)
{
      if(n->parent->leftchild->data == n->data)
            return n->parent->rightchild;
      if(n->parent->rightchild->data == n->data)
            return n->parent->leftchild;
}
```

8.10 HOW TO FIND THE ADDRESS OF THE UNCLE OF A NODE

Sibling of parent node is known as *Uncle node* for a child node.

See how a wrapper method can be written to give a conceptual look to the code.

```
node* Uncle(node *n)
{
inart     return Sibling(n->parent);
}
```

8.11 HOW TO TRAVERSE THE TREE "IN-ORDER"

Inorder traversal of the binary tree is traversing the left sub-tree, then the root and then the right sub-tree. Now each sub-tree can have own sub-trees.

```
void inorder(node *n)
{
     if(n->leftchild!=NULL)
          inorder(n->leftchild);
     printf("%d \t",n->data);
     if(n->rightchild!=NULL)
          inorder(n->rightchild);
}
```

How to Traverse the Tree "Pre-order"

```
void preorder(node *n)
{
     printf("%d \t",n->data);
     if(n->leftchild!=NULL)
          preorder (n->leftchild);

     if(n->rightchild!=NULL)
          preorder (n->rightchild);
}
```

How to Traverse the Tree "Post-Order"

```
void postorder(node *n)
{
     if(n->leftchild!=NULL)
          postorder (n->leftchild);

     if(n->rightchild!=NULL)
          postorder (n->rightchild);

     printf("%d \t",n->data);
}
```

How to Count the Number of Nodes in the Binary Tree

```
int CountNodes(node *n)
{
     int Nodes = 0;
     if(n==NULL)
          Nodes = 0;
     else
     {
          Nodes=CountNodes(n->leftchild)
                +CountNodes(n->rightchild)+1;
     }

     return Nodes;
}
```

How to Count the Number of Children a Particular Node has

```
int CountChildren(node *n)
{
     if((n->leftchild !=NULL && n->rightchild ==NULL)
      ||(n->leftchild ==NULL && n->rightchild !=NULL))
          return 1;
```

```
     if(n->leftchild !=NULL && n->rightchild !=NULL)
          return 2;
     else
          return 0;
}
```

How to Count the Number of Internal Nodes in the Binary Tree

An internal node is a node that has got two children.

```
int  InternalNodes = 0;//A Global Variable


int CountInternalNodes(node *n)
{
     if(n->leftchild!=NULL)
          CountInternalNodes(n->leftchild);
     if(n->rightchild!=NULL)
          CountInternalNodes(n->rightchild);
     if(CountChildren(n)==2)
          InternalNodes++;
     return InternalNodes;
}
```

How to Count the Number of Leaves in the Binary Tree

```
int CountLeaves(node *n)
{
     if(n==NULL ||
        (n->leftchild == NULL && n->rightchild == NULL))
          return 1;
     else
          //Tail Recursive call
          return CountLeaves(n->leftchild)+
                 CountLeaves(n->rightchild);
}
```

How to Find if Two Binary Trees are Same or Not

```
int IsSameBinaryTree(node *a,node *b)
{
     if(a==NULL && b==NULL)
          return 1;

     if(a!=NULL && b!=NULL)
          return a->data==b->data
&& IsSameBinaryTree(a->leftchild,b->leftchild)
&& IsSameBinaryTree(a->rightchild,b->rightchild);

     if((a!=NULL && b==NULL) || (a==NULL && b!=NULL))
          return 0;
}
```

8.12 WHAT IS A BINARY SEARCH TREE?

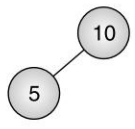
A binary search tree is a binary tree where the elements are ordered in such a way that the root, left and right children follow the relationship as

Left child < Root < Right Child

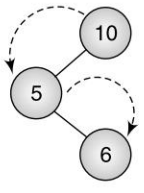
This simple rule makes it easy to search a particular item from a binary tree.
Here is an example creation of binary search tree:


```
Initially

Now trying to add 5
```

The root is 10 and 5 is less than 10, so 5 will be the left child of 10. So after insertion of 5 in the above BST it will look like

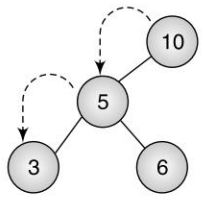


Now let's say we want to add 6 to this tree. 6 is less than 10 so it will find its place in the left sub-tree of the BST. In root of the left sub-tree is 5. 6 is greater than 5 so 6 will be the right child of 5. So after insertion of 6 in the above BST, it will look like



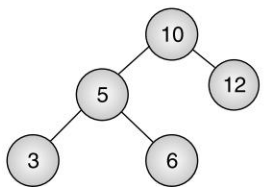
The dotted line displays the path travelled by 6, before finding its place in the tree.

Now let's say we want to put 3 in the above tree. So as 3 is less than 10 it will again go to the left sub-tree and 3 is less than 5 so it will be the left child of 5. So the tree, after insertion of 3 will look like



The dotted line shows the path of 3 before it finds its place in the BST.

So if we add 12 the tree will look like



The Binary Search Tree like any other data structure can hold any type of data. We might want our BST to hold the members of a club. So, in cases like these the mathematical symbol (i.e. '<=', '>') won't work directly and we will have to design a function that can compare two members and depending upon a parameter, can determine who will be the left and who will be the right child of an identified parent node.

How to Add an Element to a Binary Search Tree

```
//Assumes that the root is already there
void Add2BST(node *n,int value)
{
          if(value <=n->data)
          {
               if(n->leftchild!=NULL)
                    Add2BST(n->leftchild,value);
               else
               {
                    node *lc = (node *)malloc(sizeof(node));
                    lc->rightchild=NULL;
                    lc->leftchild = NULL;
                    lc->data = value;
                    n->leftchild = lc;
               }
          }
          if(value >n->data)
          {
               if(n->rightchild!=NULL)
                 Add2BST(n->rightchild,value);
               else
               {
                    node *rc = (node *)malloc(sizeof(node));
                    rc->rightchild=NULL;
                    rc->leftchild = NULL;
                    rc->data = value;
                    n->rightchild = rc;
               }
          }

}
```

How to Check if a Binary Tree is a Binary Search Tree

```
int isBST(node *n)
{
     if(n->leftchild->data>n->data || n->rightchild->data<n->data)
          return 0;
     if(n->leftchild->data < n->data && n->rightchild->data >=n->data)
          return 1;
     else
          return isBST(n->leftchild) && isBST(n->rightchild);
}
```

8.13 HOW TO SEARCH A VALUE IN A BST

Searching in a binary search tree is the most optimized operation possible on the data structure.
First the data is matched with the root node. In case there is a mismatch, we go to the left or right sub-tree depending on whether the number is smaller than root value or greater than it or is it greater than the root. This process rotates recursively until we travel all the nodes of the tree.

```
int SearchBST(node *n,int value)
{
     if(n==NULL)
          return 0;
     else
     {
          //Checking with the root value
          if(n->data == value)
               return 1;
          else
          {
               //If the number is greater than or equal
               //We need to look in the right sub tree
               //recursively for each sub-sub tree
               if(n->data >= value)
                    return SearchBST(n->rightchild,value);
               else
               //Otherwise we need to look into the left subtree
                    return SearchBST(n->leftchild,value);

          }
     }
}
```

How to Delete a Node from a BST

```
void DeleteNode(node *n)
{
    node* temp = n;

     if(n!=NULL)
     {
          printf("deleting %d\n",n->data);
          if (n->leftchild == NULL && n->rightchild!=NULL)
          {
               n = n->rightchild;
               delete temp;
          }
          else
if (n->rightchild == NULL && n->leftchild!=NULL)
{
     n = n->leftchild;
     delete temp;
}
else
```

```
{
     // Node has two children - get max of left subtree
     temp = n->leftchild;
     while (temp->rightchild != NULL)
     {
          temp = temp->rightchild;
     }
     n->data = temp->data;
          DeleteNode(temp);
     }

}
else
     return;

}
```

8.14 WHAT IS THE RIGHT ROTATION ON A BST?

In a binary search tree, pushing a node *N* down and to the right to balance the tree. *N*'s left child replaces *N*, and the left child's right child becomes *N*'s left child.

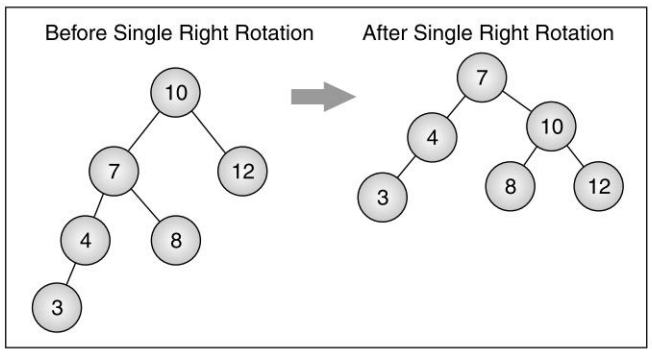


Fig. 8.7 *See the pic above to understand the Single Right Rotation*

How to Rotate a Tree to the Right Once around a Node

```
node* SingleRightRotation(node *n)
{
     node* newroot = n->leftchild;
     n->leftchild = n->leftchild->rightchild;
     newroot->rightchild = n;
     return newroot;
}
```

What is the Left Rotation on a BST?

In a binary search tree*, pushing a* node *N down and to the left to* balance *the tree. N's right* child *replaces N, and the right child's left child becomes N's right child.*

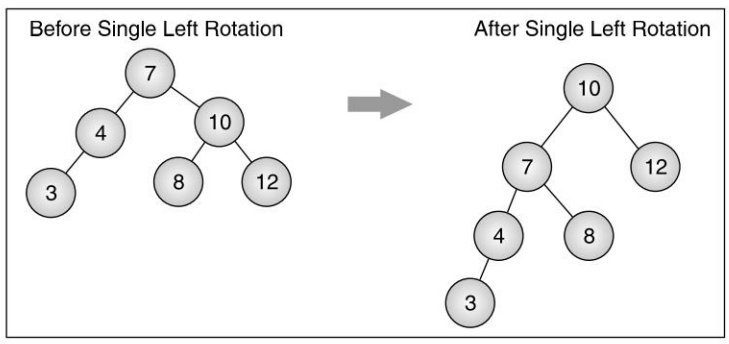


Fig. 8.8 *Double Right Rotation is nothing but two consecutive right rotations*
Double Left Rotation is nothing but two consecutive left rotations

How to Rotate a Tree Once to the Left

```
node* SingleLeftRotation(node *n)
{
     node* newroot = n->rightchild;
     n->rightchild = n->rightchild->leftchild;
     newroot->leftchild = n;
     return newroot;
}
```

Note that the relative ordering of the elements does not change by these rotations. The inorder, preorder, postorder traversing result in same sequence. I mean even after rotation the tree remains as a BST. The balance just get better, nothing else has changed from a conceptual point of view.

8.15 SOME AREAS OF APPLICATION OF BST

1. Information Organization
2. Information Retrieval
3. Information Indexing
4. Creating Associative Containers
5. Displaying Arithmetic Expression in Different Orders
6. Spatial Indexing
7. Binary Image Representation
8. Binary Image Processing
9. Computer Graphics Polygon Rendering
10. Shadow Generation
11. Spatial Vision for Robots.
12. 3D Motion Tracking
13. Proximity Sensing Device
14. Web Browser
15. Windows Explorer
16. Shortest Path
17. Sorting (*See Alphabetical Sort below*)

These are just a few to name. Binary Search Tree and its variations are used in many diverse areas.

8.16 WHAT IS AN EXPRESSION TREE?

An exception tree is nothing but a Binary Tree which represents arithmetic expressions.
The nodes of the tree stores the operators and the binary operands (+,-,*,/). Binary Tree is the best data structure to store this expression, because only the traverse of the tree in post-order can generate the postfix notation which is needed by a postfix calculator that uses stack data structure to calculate the result.

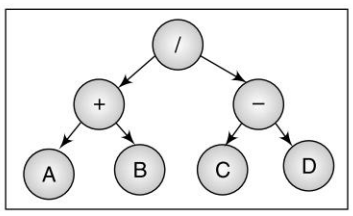


Fig. 8.9 *This is an Exception Tree.*

Some facts about Expression Tree

- Each leaf node represents an operand.
- An internal node is an operator.
- A post-order traversal is performed over the nodes.
- In a post-order traversal, each node is visited after its descendent nodes are visited.

The different orders of traversal of the above expression tree are
Post order ab+cd–/
Pre order /+ab–cd
In order a+b/c–d

How to Represent an Expression Tree as a Binary Tree

```
typedef struct node
{
     int data;
     int isOperator;
     char OperatorSymbol;
     struct node *leftchild;
     struct node *rightchild;
}node;


void AddLeftChild(node *n,int value,char op)
{
     if(value!=0)
     {
          node *r = (node *)malloc(sizeof(node));
          r->data = value;
          r->isOperator = 0;
          r->OperatorSymbol = op;
          r->leftchild = NULL;
          r->rightchild = NULL;
          n->leftchild = r;
     }
     else
     {
          node *r = (node *)malloc(sizeof(node));
          r->data = 0;
          r->isOperator = 1;
          r->OperatorSymbol = op;
          r->leftchild = NULL;
          r->rightchild = NULL;
```

```
            n->leftchild = r;
      }
}

void AddRightChild (node *n,int value,char op)
{
      if(value!=0)
      {
            node *r = (node *)malloc(sizeof(node));
            r->leftchild = NULL;
            r->rightchild = NULL;
            r->data = value;
            r->isOperator = 0;
            r->OperatorSymbol = op;//just a space
            n->rightchild = r;
      }
      else
      {
            node *r = (node *)malloc(sizeof(node));
            r->leftchild = NULL;
            r->rightchild = NULL;
            r->data = 0;
            r->isOperator = 1;
            r->OperatorSymbol = op;
            n->rightchild = r;
      }
}

void postorder(node *n)
{

      if(n->leftchild!=NULL)
            postorder(n->leftchild);

      if(n->rightchild!=NULL)
            postorder(n->rightchild);

      if(n->isOperator==1)
            printf("%c ",n->OperatorSymbol);
      if(n->isOperator==0)
            printf("%d ",n->data);

}
```

This program generates an Expression Tree as

```
int main()
{

      node *root = (node *)malloc(sizeof(node));
      root->isOperator = 1;
```

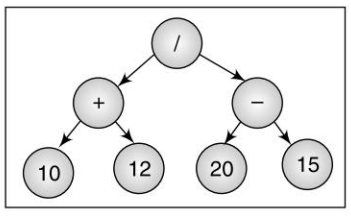


Fig. 8.10

```
root->OperatorSymbol = '/';
root->leftchild = NULL;
root->rightchild = NULL;
root->data = 0;//Assuming Positive Number expression tree

AddLeftChild(root,0,'+');
AddRightChild(root,0,'-');
AddLeftChild(root->leftchild,10,' ');
AddRightChild(root->leftchild,12,' ');
AddLeftChild(root->rightchild,20,' ');
AddRightChild(root->rightchild,15,' ');


postorder(root);
getch();

return 0;
}
```

Example 8.1 ***Write a program to demonstrate Alphabetical Sorting Application of Binary Tree.***

Solution The strategy to represent a dictionary by a binary search tree is as follows.

- If the word to be inserted is alphabetically backward than the root, then it will go to left child/sub-tree
- If the word to be inserted is alphabetically forward than the root, then it will go to right child/sub-tree

A Sample

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

typedef struct node
{
      char data[20];//The word to store per node
      struct node *leftchild;
      struct node *rightchild;
}node;

//Assumes that the root is already there
void Add2BST(node *n,char *value)
{
            if(strcmpi(value ,n->data)<0)
            {
                  if(n->leftchild!=NULL)
      Add2BST(n->leftchild,value);
else
{
      node *lc = (node *)malloc(sizeof(node));
      lc->rightchild=NULL;
      lc->leftchild = NULL;
      strcpy(lc->data,value);
```

```
                n->leftchild = lc;
            }
        }
          if(strcmpi(value,n->data)>0)
          {
                if(n->rightchild!=NULL)
                  Add2BST(n->rightchild,value);
              else
                  {
                        node *rc = (node *)malloc(sizeof(node));
                        rc->rightchild=NULL;
                        rc->leftchild = NULL;
                        strcpy(rc->data,value);
                        n->rightchild = rc;
                  }
            }

}

int SearchBST(node *n,char *value)
{
      if(n==NULL)
            return 0;
      else
      {
            if(n->data == value)
                  return 1;
            else
            {
                  if(n->data >= value)
                        return SearchBST(n->leftchild,value);
                  else
                        return SearchBST(n->rightchild,value);
            }
      }
}

void inorder(node *n)
{
      if(n->leftchild!=NULL)
            inorder(n->leftchild);
      printf("%s \t",n->data);
      if(n->rightchild!=NULL)
            inorder(n->rightchild);
}

int main()
{
      char word[20];
      int i = 0;
      node* root = (node *)malloc(sizeof(node));
      strcpy(root->data,"Water");
      root->leftchild = NULL;
```

```
root->rightchild = NULL;
for(i = 0;i<5;i++)
{
      puts("Enter a word");
      scanf("%s",word);
      Add2BST(root,word);
}
      puts("inorder");
      inorder(root);
      getch();
      return 0;
}
```

This program stores 10 words as they would appear in dictionary and sort them alphabetically.

Here is a sample run of the above program:

Enter a word
Room
Enter a word
Violin
Enter a word
Sun
Enter a word
Monday
Enter a word
Bed
inorder
Bed Monday Room Sun Violin Water

Notice the words are alphabetically sorted.

8.17 WHAT IS A DECISION TREE?

A *decision tree* is a Tree that stores the operations to be performed if the choice selected is either

A few typical examples of a decision trees are shown here:

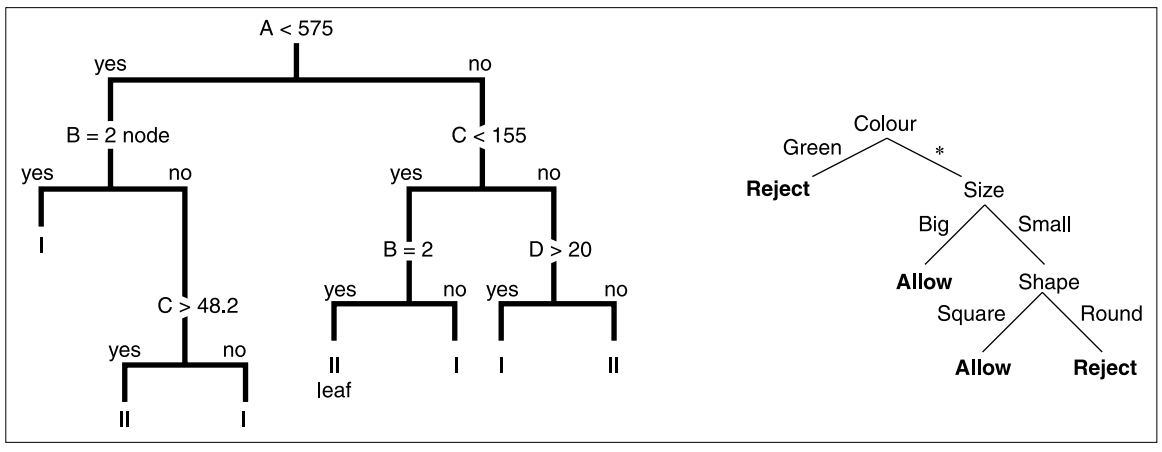


Fig. 8.11

If you notice that the edges of the tree represents the condition.

Decision tree, as the name suggests can help a confused person on the road to get back to the right track.

Decision tree can be used in any kind of problem where deduction of the solution can be done by processing the answers of a set of interdependent questions.

- Finding a place in the citi
- Finding the right school for a kid
- Finding the right career
- Finding the ideal mortgage loan
- Computerized diagnosis of diseases
- Adaptive testing system

And so on.

By Definition a decision tree can have *n* number of children per node where *n* represents the number of degree of freedom (i.e. number of parameters involved) in the problem. For example answer to a particular question like 'Would you like some cold coffee?' might have responses like 'Yes', 'No', 'Might be'. So there would be 3 children for this node. But as we know that any M-ary tree (A tree where a node can have a maximum of M children) can be redesigned using a binary tree, so all decision trees designed for any practical purposes are essentially binary tree.

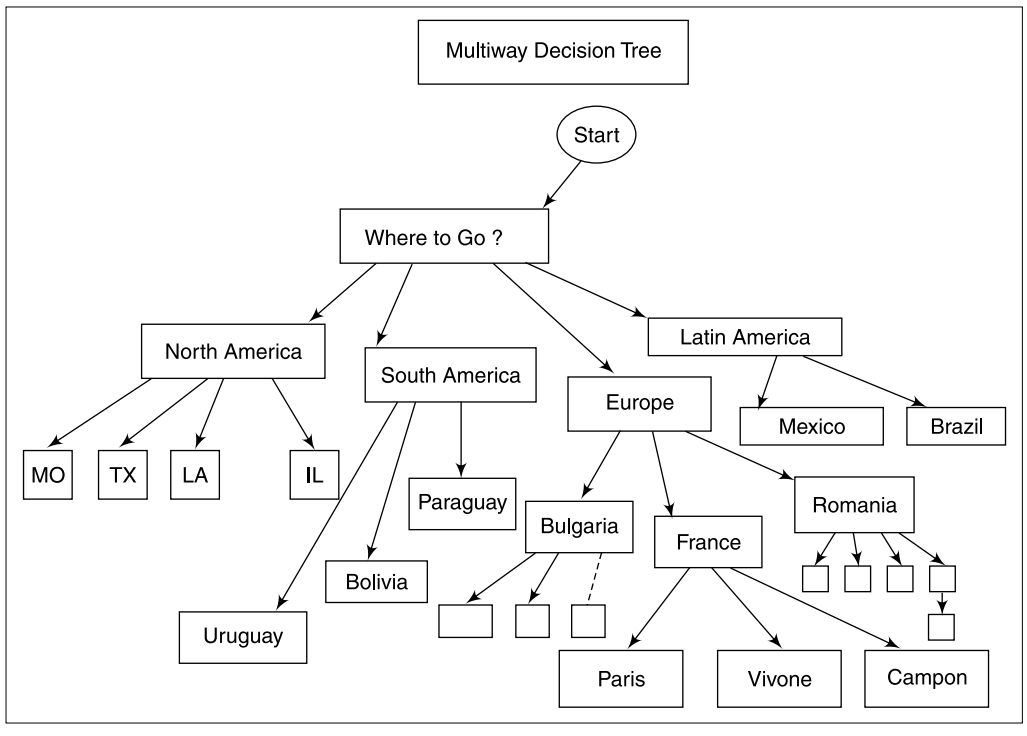


Fig. 8.12 *A Multiway Decision Tree*

8.18 BINARY SEARCH TREE AND GAMES

There are set of games that is played by *adaptive and informed guesses*. For example 'Guess the Price' game. The player is asked to guess the price of something and then if guessed right, another question is asked. Else the player is out of the game. This can be modeled by BSTs.

Example 8.2 ***Write a program for Adaptive Testing using a Binary Search Tree.***

Solution Nowadays almost all online test are adaptive. By adaptive, I mean everybody in the test doesn't get a single set of paper. They all get a different starting question. If they answer that question correct, they get a harder question and if they get that wrong, they get a simpler question. This situation can easily be replicated with a BST where the starting question will be the root of the tree and the easy questions will go to the left sub-tree and the harder questions will find their room in the right sub-tree. We can assign marks to each question. So, as soon as the test ends, actually in no time we can publish the results.

Here is as structure that represents a node in such a BST.

```
typedef struct node
{
     char question[50];
     char answer[20];
     int marks;
     struct node* nexthard;
     struct node* nextsimple;
}node;
```

At the start of the program, the program reads questions, their answers, marks and creates a BST.

The following details are fed to the program per node.

The Hardness Index (HI)

The question statement

The correct answer

Marks to be given for answering this question correctly.

When the user logs in the system the question from the root will be asked, and then depending upon whether correct or wrong answers are given the user will get harder or simpler questions.

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


typedef struct node
{
      char question[50];
      char answer[20];
      int marks;
      int HardIndex;
      struct node* nexthard;
      struct node* nextsimple;
}node;

//Assumes that the root is already there
//This function adds a new question to the
//tree.
void Add2BST(node *n,int HI,char q[],char a[],int m)
{
            if(HI <=n->HardIndex)
            {
                  if(n->nextsimple!=NULL)
                        Add2BST(n->nextsimple,HI,q,a,m);
                  else
                  {
                        node *lc = (node *)malloc(sizeof(node));
```

```
                lc->nexthard=NULL;
                lc->nextsimple = NULL;
                lc->HardIndex = HI;
                strcpy(lc->question,q);
                strcpy(lc->answer,a);
                lc->marks = m;
                n->nextsimple = lc;
                }
          }
            if(HI >n->HardIndex)
            {
                  if(n->nexthard!=NULL)
                    Add2BST(n->nexthard,HI,q,a,m);
                else
                  {
                        node *rc = (node *)malloc(sizeof(node));
                        rc->nexthard=NULL;
                        rc->nextsimple = NULL;
                        rc->HardIndex = HI;
                        strcpy(rc->question,q);
                        strcpy(rc->answer,a);
                        rc->marks = m;
                        n->nexthard = rc;
                  }
          }

}


void quiz(node *n)
{
      char ans[20];
      printf("%s\n",n->question);
      fflush(stdin);
      gets(ans);
      if(strcmpi(n->answer,ans)==0)
      {
            if(n->nexthard!=NULL)
                  quiz(n->nexthard);
      }
      else
      {
            if(n->nextsimple!=NULL)
                  quiz(n->nextsimple);
      }
}


int main()
```

```
{
      node *root = (node *) malloc(sizeof(node));
      strcpy(root->question,"What is your Passport# ?");
      strcpy(root->answer,"E556677");
      root->HardIndex = 10;
      root->marks = 10;
      root->nexthard = NULL;
      root->nextsimple = NULL;
      Add2BST(root,8,"What is your nationality ?","Indian",3);
      Add2BST(root,12,"What is your Passport's Expiry Date ?",
              "31-DEC-2020",12);
      quiz(root);
      getch();
      return 0;
}
```

We can enhance this program by reading a file that stores these questions and their answers. The program will read the details from the file and store them in a binary search tree.

8.19 HOW TO HANDLE MULTIPLE SUBJECTS IN SUCH AN ADAPTIVE TEST

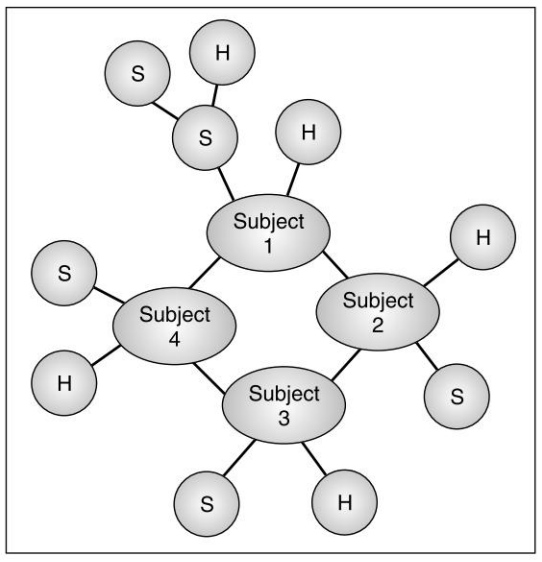


Fig. 8.13

To handle multiple subjects we can design a forest of binary trees where the roots of the tree will contain three more attributes. One link to the next subject tree, one link to the previous subject tree and one string that tells which subject is this tree for.

Notice the above binary forest. This forest holds the binary search tree of questions for four subjects. The right sub-trees store the harder questions and the left sub-trees store the simple questions. We can go on adding questions this way.

We can even segregate the question even better by creating levels of this forest as shown in the figure below later.

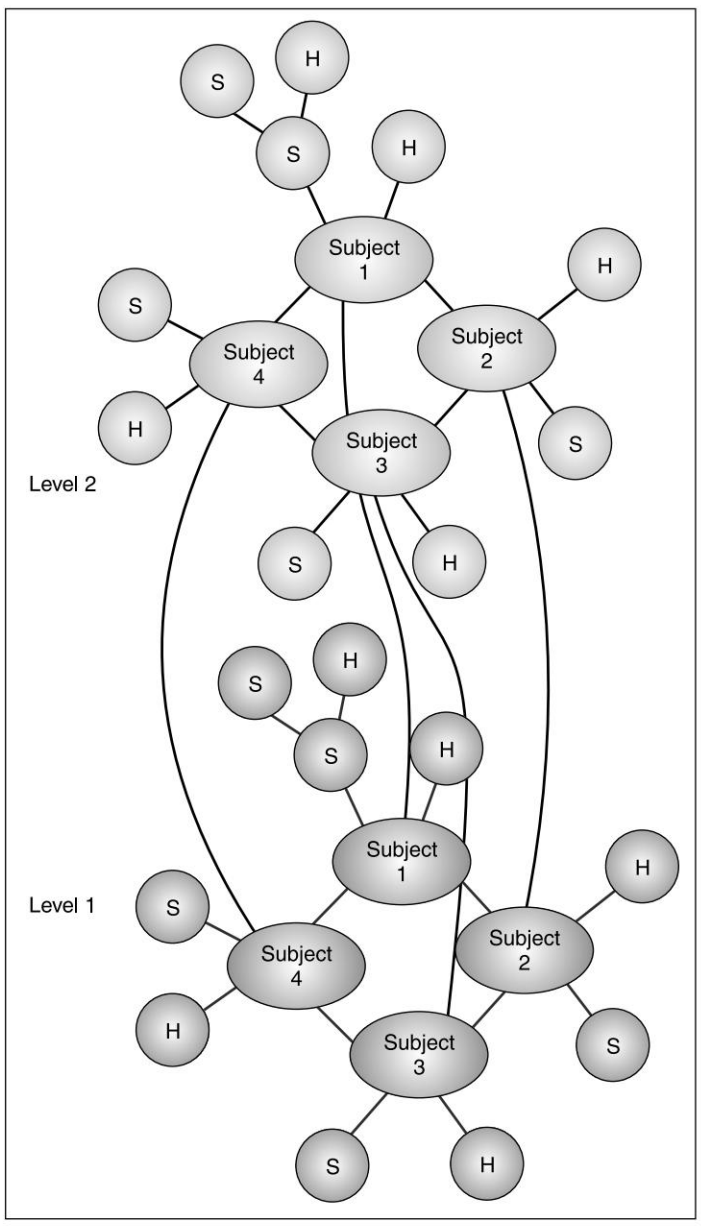


Fig. 8.14

In this illustration you can see how the roots of each level are connected to higher levels. It looks like a lantern with fancy antennas.

We can call this type of a forest generally as ***mn Binary Tree Forest****. Where* ***m*** *is the number of subjects per level and* ***n*** *is the number of levels we create. We can have multiple levels holding papers of different subjects for a university. Starting from one level we can finish on another forming an arch. I call it a* ***binary spider arch****, because each level looks like a spider. We can have multiple such arches*

connected together to hold the question of different subjects of different levels from different universities. Given below is an image that tries to make your understanding better.

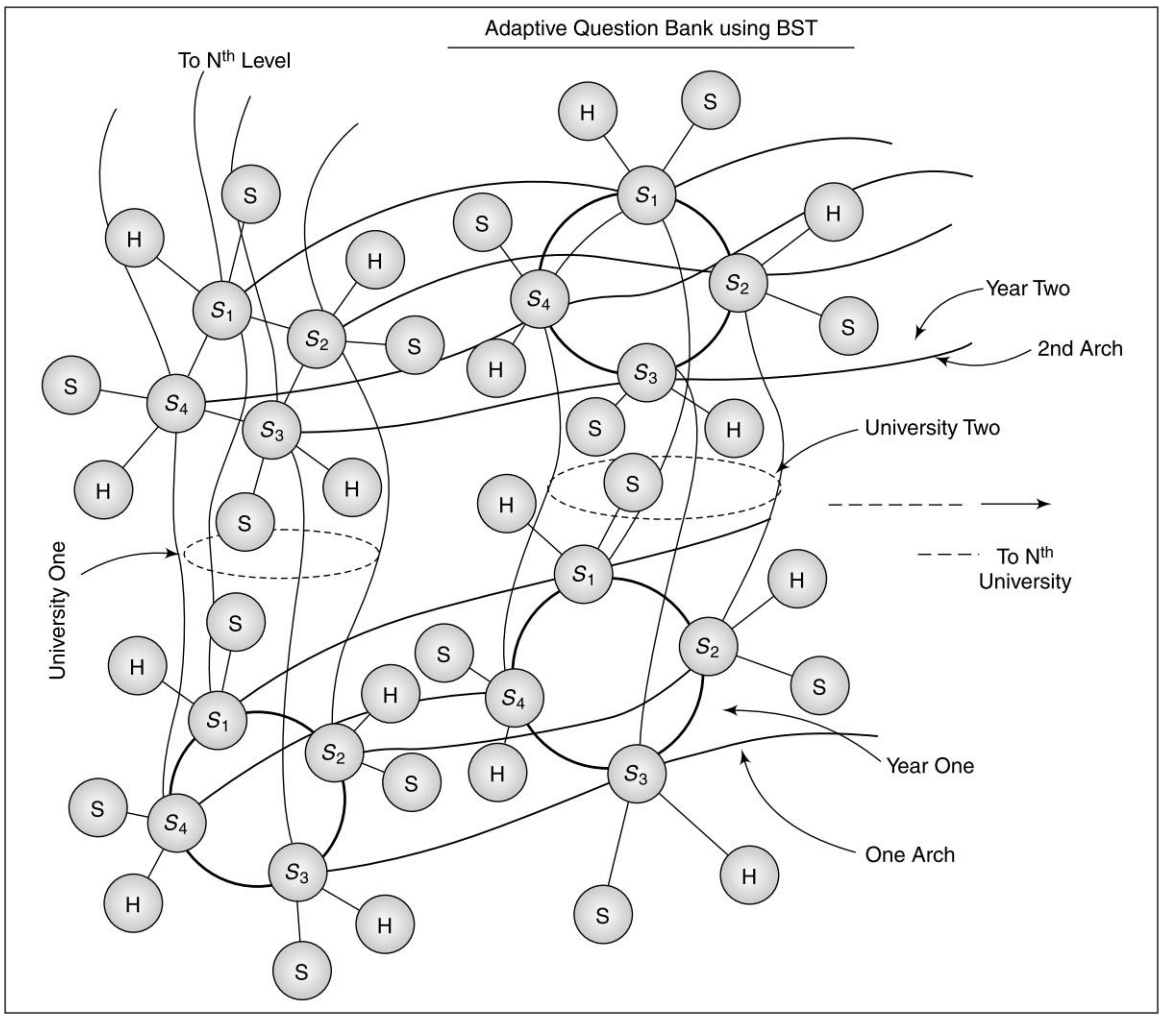


Fig. 8.15

8.20 HOW TO CONVERT A MULTIWAY TREE TO A BINARY TREE

A Multi-way tree is nothing but a tree that can have *n* number of children per node. But there is a way that we can convert any multi-way tree to a binary tree. The steps for converting are as follows

- Keep the leftmost child of any node in the multi-way tree as it is.
- Rest all the children will form the right sub-tree of the left-child node of the binary tree.

Here is an example

Note that the decision tree might have more than two children per internal node. In that case it will be a *M*-ary tree which can be converted to a binary tree as shown in the picture above. This shows that essentially all decision trees can be modeled using binary trees.

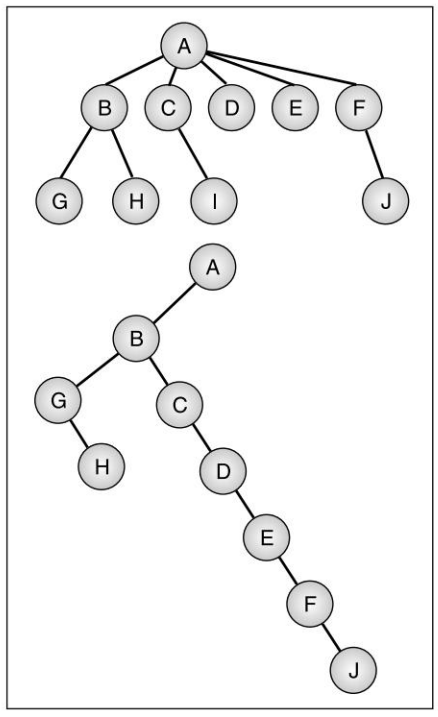


Fig. 8.16

8.21 WHAT IS THE BALANCE FACTOR OF A BST?

While adding the nodes in a BST depending on their value we may end up with some kind of tree where the left sub-tree is quite bigger than that of right sub-tree or vice-versa. Thus the BST becomes unstable or experience imbalance. To measure how much imbalance a BST does have, there is an index called *balancing factor*. The balancing factor of a BST is defined as the difference between the depth of left and right sub-tree. Here is a picture showing a highly imbalanced BST.

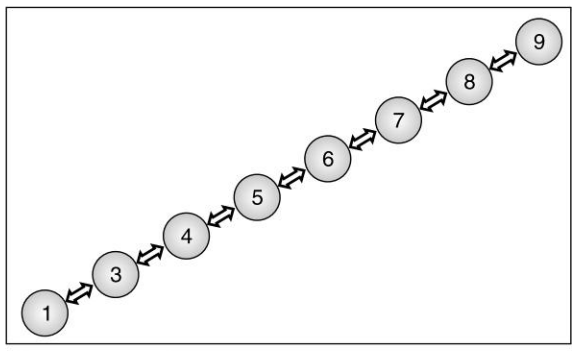


Fig. 8.17 *A highly imbalanced BST*

In this BST, the left sub-tree has 7 levels while the right sub-tree has no level. That means this tree's balancing factor is 0 – 7 = –7.

If the tree has only the root, then the balancing factor is –0

If the tree has same number of levels on both the sub-trees then the balancing factor is 0

If the tree has only one level of difference between the left and right sub-tree, then the balancing factor is 1

If the balancing factor of a tree is anything other than –1, 0 or 1, the tree is said to be imbalanced. Thus the above tree is imbalanced.

How to Find the Balance Factor of a Binary Search Tree

```
//This function finds the depth/height/degree of the left sub tree
//while the supplied argument is the pointer to the root
int DepthLeftTree(node *n)
{
     return Degree(n->leftchild);
}

//This function finds the depth/height/degree of the right sub tree
//while the supplied argument is the pointer to the root
int DepthRightTree(node *n)
{
     return Degree(n->rightchild);
}

//This function returns the Balance Factor of the BST using the above
//two functions.
int BalanceFactor(node *n)
{
     if(n==NULL)
          return -1;//No tree exists!
     else
          return DepthRightTree(n)-DepthLeftTree(n);
}
```

How to Find if a BST is Balanced or Not

```
int IsBalanced(node *n)
{
     if(BalanceFactor(n)==1
        || BalanceFactor(n)==0
        || BalanceFactor(n)==-1)
          return 1;
     else
          return 0;
}
```

8.22 HOW TO BALANCE A BINARY SEARCH TREE

There two types of BST to balance an imbalanced Binary Search Tree.

Self-Balancing Binary Search Tree

Self-balancing binary search tree or **height-balanced binary search tree** is a binary search tree that attempts to keep its *degree*, or the number of levels of nodes beneath the root, as small as possible at all times, automatically. It is one of the most efficient ways of implementing associative arrays, sets, and other data structures.

Some data structures that follow this algorithm are

- AA Tree
- AVL Tree
- Red – Black Tree
- Scapegoat Tree

Self-Organizing Binary Search Tree

- Splay Trees
- Heap

Different Variations of a Heap are

- Binary Heap
- Fibonacci Heap
- Lefist Heap
- Treap
- Skew Heap
- Binomial Heap
- Pairing Heap
- Soft Heap
- Beap

Few of these data structures will be discussed here.

8.23 WHAT IS A SPLAY TREE?

A Splay tree is a Binary Search Tree, which is self-organizing. After every search in the splay tree, the sought item is found at the root of the tree. So unlike other trees where search is a mere passive activity, here in splay tree, searching a value is an active activity, which changes the structure of the tree. So that the most sought item will always be available at the root of the tree. This reduces the searching time drastically.

Splaying is done using *tree rotations*. Left rotation is termed as *Zig* and right rotation is termed as *Zag*.

Here is a picture that shows Zig operation:

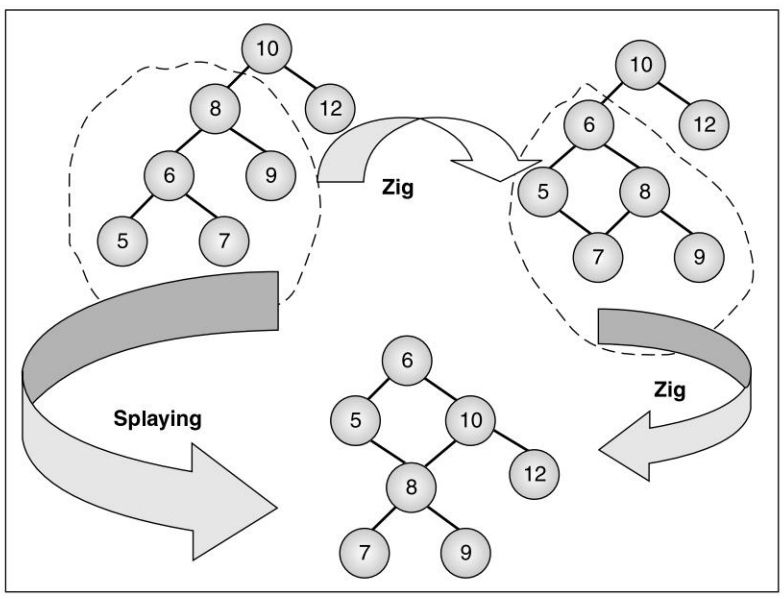


Fig. 8.18

The above picture shows how the BST restructures itself when the number 6 is sought. Please note two points carefully.

- After searching a value in splay tree, if the value is found then the value will occupy the new root in the tree.
- The resulting tree still remains the same BST. I mean the traversal of the tree in any order after splaying doesn't differ from the original one.

Here is a picture that describes the zag operation which is basically left rotation about a node. The nodes above this doesn't experience any kind of change in the location in the tree.

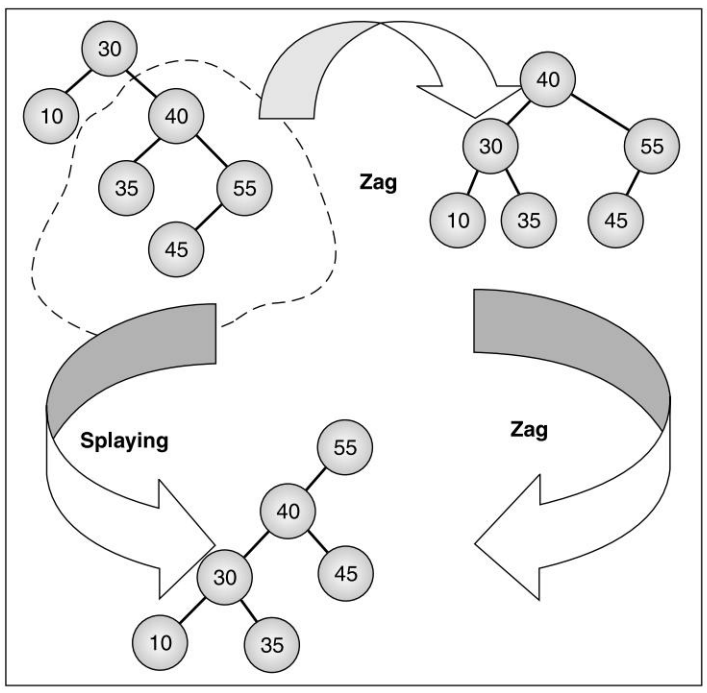


Fig. 8.19

The above image shows searching of 55 in the splay tree. So, after searching 55 becomes the new root and the BST structure maintains. See the left arrow shows what will the tree look like if we search for 55. And the other two arrows show legal steps to arrive at this final BST. Here couple of Zag operations is performed to reach the final splayed binary search tree with the sought element at the root.

If the element being sought is at the right or left arm of the tree (i.e. edge node) then a series of zig or zag operations would be needed to create the splay tree. But in case the sought node are somewhere in between the trees, then a series of zig and zag operations are needed to create the splay tree.

Say we need to search for 45 in the above BST , then first one zig operation and then a series of two zag operations would be needed to make the splayed tree with 45 at the root. On the other hand, if we have to search 7 from the first BST above then we need a zag operation first and then series of two zig operations.

How to Perform Zig Operation on a BST

```
typedef struct node
{
     int data;
     struct node *leftchild;
     struct node *rightchild;
     struct node *parent;
}node;

node* zig(node *n)
{
     //Hold the parent's address
     node* temp = GrandParent(n);
```

```
    n = SingleRightRotation(n->parent);
    temp->leftchild = n;
    return temp;
}
```

How to Perform Zag Operation on a BST

```
node* zag(node *n)
{
    //Hold the parent's address
    node* temp = GrandParent(n);
    n = SingleLeftRotation(n->parent);
    temp->rightchild = n;
    return temp;
}
```

8.24 WHAT IS A HEAP?

A Heap is a tree where all the children of a node are smaller to it. For example see the image below.

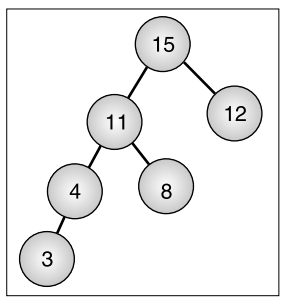


Fig. 8.20

To be precise, the image shown above is that of a binary heap. In a heap, a node may have more than 2 children. But in a binary heap, they can have only two children maximum.

There could be two types of heaps.

- Max Heap
- Min Heap

Max Heap In Max heap the parent is always greater or equal than its children. Greater than equal to does not necessarily have to be the mathematical symbol, because the contents of the heap are not always numerical. The greater than or equal to or less, these all are calculated using some functions.

Say we have got two nodes A and B. A is the parent and B is the child.

Then for a Max- Heap the following condition is always satisfied.

$f(A) >= f(B)$

where f symbolizes the measuring function.

So in a Max Heap the maximum element will always be available at the root.

Min Heap In Min heap just the reverse of Max-Heap happens. Thus the least element is available in the root always.

8.25 WHAT ARE THE DIFFERENT APPROACHES TO CREATE A HEAP?

There is couple of approaches to create a heap.

1. Heapify an existing tree (Not recommended)
2. Create the heap as it grows
   (a) Top down approach
   (b) Bottom up approach

Heapify:

- Create a binary tree as you please
- Change its contents so that it satisfies the heap condition

Top-down Approach

- If the node, you are trying to insert finds its path in the heap, starting from the root then the heap generated is known to be created by Top-down approach.

Bottom-up approach

- If the node, you are trying to insert finds its path in the heap, starting from the leaves of the heap, then the heap generated is known to be created by Bottom-up approach.

8.26 HOW TO IMPLEMENT A BINARY HEAP USING ARRAY

Heap is actually a tree, so we can represent heap the way we did it for tree. The first element of the array will be the root of the heap. The left and the right children will be at 2n + 1 and 2n + 2 location for the n^{th} node. Balancing a heap is done by swapping the elements which are out of order. The parent of the node at location *k* will be found at *a*[floor((*i* – 1)/2)].

Try Yourself: *Try to model a binary heap using array.*

8.27 WHAT IS AN AVL TREE?

AVL Tree is a self balancing binary search tree. This is the first of its kind. In an AVL tree, the heights of the two child subtrees of any node differ by at most one, therefore it is also called height-balanced. Lookup, insertion, and deletion all take *O*(log *n*) time in both the average and worst cases. Additions and deletions may require the tree to be rebalanced by one or more tree rotations.

The AVL tree is named after its two inventors, *G.M. Adelson-Velsky* and *E.M. Landis*, who published it in their 1962 paper "*An algorithm for the organization of information*."

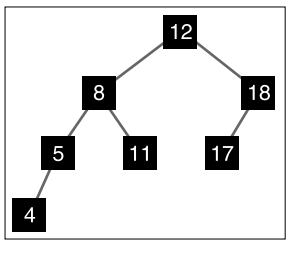


Fig. 8.21 *This is an AVL Tree*

8.28 HOW TO INSERT AN ELEMENT TO AN AVL TREE

Insertion in an AVL tree is just same as inserting that in a BST. But after insertion a series of left or right rotations are performed in order to restore the balance of the tree. In some implementations you will find that the balance factor of a node is stored as an element of the node. That is not a good idea. In this implementation of AVL tree, I am going to use the same node structure defined for the BST above in this chapter.

The rotations are taking place around the root always. In some implementations you will find that rotations are performed over the sub-trees first and then on the root. But that way we actually will spend more time. So, in the following function rotations are done around the root.

```
node* Add2AVL(node *n,int data)
{
     Add2BST(n,data);//Adding to the BST first
//Adjusting the balance by performing
//Left and Right Rotation.
do
{
     if(Degree(n->leftchild)>Degree(n->rightchild))
          n = SingleRightRotation(n);
```

```
        else
            n = SingleLeftRotation(n);
    }while(!(BalanceFactor(n)!=0 || BalanceFactor(n)!=1));

    return n;
}
```

8.29 WHAT IS A BSP TREE?

BSP is the acronym for *Binary Space Partitioning*. This is a method to recursively subdivide a space into convex sets by hyperplanes. This subdivision gives rise to a representation of the scene by means of a tree data structure known as a BSP tree. BSP tree and its variations in 2D (Known as *Quad Tree*), 3D (*k-d Tree*) are very important data structures used in computer graphics. Later in this chapter we will discuss about them.

At the very first, the entire area is denoted as the root of the BSP Tree. Then you go on breaking up the places. Once you break a concave area into two convex (in the best case) or concave polygons, name these areas and they become the children of their parents, which actually represent the entire area.

Here is a picture that shown the BSP Tree generation process.

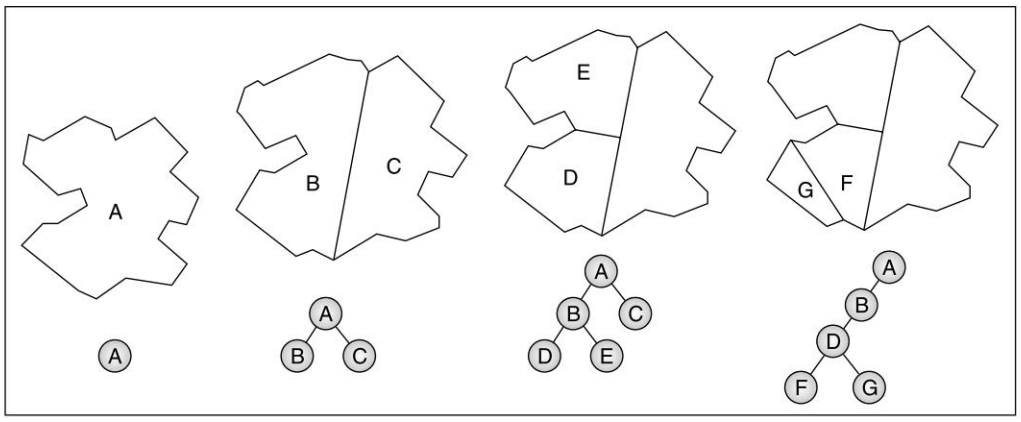


Fig. 8.22 *If you notice carefully you can find that G and F are two convex polygons. So the generation process comes to an end.*

Applications of BSP Tree

BSP Tree or its variations like Quad Tree, K-d Tree are highly applied in

- Image Processing
- Machine Vision
- Computer Graphics
- Computer Games

8.30 WHAT IS A QUAD TREE? A 2D VARIATION OF BSP (*ALSO KNOWN AS* Q-TREE)?

A *Quad Tree* is a tree like data structure where each node has four children. Quadtrees are most often used to partition a two dimensional space by recursively subdividing it into four quadrants or regions.

The regions may be square or rectangular, or may have arbitrary shapes. They are also known as *Q-Trees*.

Quad Trees are used heavily in

- Image Processing
- Spatial Indexing
- Efficient Collision Detection in 2D
- Storing Sparse Data

Generally speaking quad trees could prove to be highly efficient to solve any problem that has got 4 or more (A multiple of 4 probably) degrees of freedom.

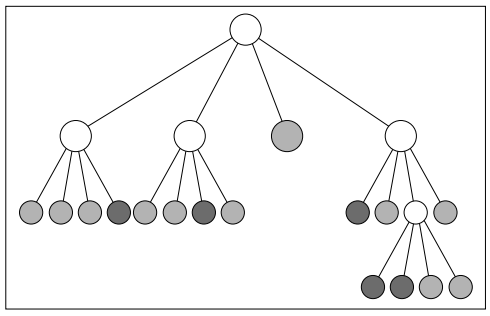


Fig. 8.23

How to Model a Quad Tree using C Structure

```
typedef struct QuadTreeNode
{
     int value;
     struct QuadTreeNode *Parent;
     struct QuadTreeNode *NEChild;
     struct QuadTreeNode *NWChild;
     struct QuadTreeNode *SWChild;
     struct QuadTreeNode *SEChild;
}
QuadTreeNode;
```

This structure models a node of the Quad Tree whose nodes hold integer values. For example this tree can represent a quad tree which can hold block by block representation on a gray level image.

How to Add a North-East Child to This Quad Tree

```
void AddNEChild(QuadTreeNode* parent,int v)
{
     parent->NEChild = (QuadTreeNode*)malloc(sizeof(QuadTreeNode));
     parent->NEChild->value = v;
     parent->NEChild->NEChild = NULL;
     parent->NEChild->NWChild = NULL;
     parent->NEChild->SWChild = NULL;
     parent->NEChild->SEChild = NULL;
     parent->NEChild->Parent = parent;//Both Way connection
}
```

How to Add a North-West Child to This Quad Tree

```
void AddNWChild(QuadTreeNode* parent,int v)
{
     parent->NWChild = (QuadTreeNode*)malloc(sizeof(QuadTreeNode));
     parent->NWChild->value = v;
     parent->NWChild->NEChild = NULL;
     parent->NWChild->NWChild = NULL;
     parent->NWChild->SWChild = NULL;
     parent->NWChild->SEChild = NULL;
     parent->NWChild->Parent = parent;//Both Way connection
}
```

How to Add a South-East Child to This Quad Tree

```
void AddSWChild(QuadTreeNode* parent,int v)
{
      parent->SWChild = (QuadTreeNode*)malloc(sizeof(QuadTreeNode));
      parent->SWChild->value = v;
      parent->SWChild->NEChild = NULL;
      parent->SWChild->NWChild = NULL;
      parent->SWChild->SWChild = NULL;
      parent->SWChild->SEChild = NULL;
      parent->SWChild->Parent = parent;//Both Way connection
}
```

How to Add a South-West Child to this Quad Tree

```
void AddSEChild(QuadTreeNode* parent,int v)
{
      parent->SEChild = (QuadTreeNode*)malloc(sizeof(QuadTreeNode));
      parent->SEChild->value = v;
      parent->SEChild->NEChild = NULL;
      parent->SEChild->NWChild = NULL;
      parent->SEChild->SWChild = NULL;
      parent->SEChild->SEChild = NULL;
      parent->SEChild->Parent = parent;//Both Way connection
}
```

How to Check whether a Node is a North-East Child of a Node

```
int isNEChild(QuadTreeNode *parent,QuadTreeNode *child)
{
     return parent->NEChild == child;
}
```

How to Check whether a Node is a North-West Child of a Node

```
int isNWChild(QuadTreeNode *parent,QuadTreeNode *child)
{
     return parent->NWChild == child;
}
```

How to Check whether a Node is a South-East Child of a Node

```
int isSEChild(QuadTreeNode *parent,QuadTreeNode *child)
{
     return parent->SEChild == child;
}
```

How to Check whether a Node is a South-West Child of a Node

```
int isSWChild(QuadTreeNode *parent,QuadTreeNode *child)
{
     return parent->SWChild == child;
}
```

8.31 HOW TO GET THE NORTH-EAST UNCLE OF A QUAD TREE NODE

Before we discuss the solution of this question, see the picture below to understand the relationship between uncle and nephew.

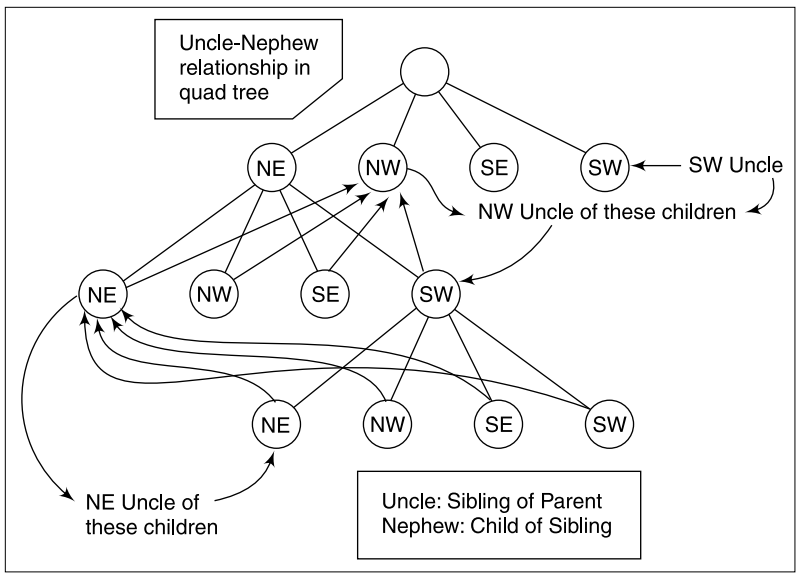


Fig. 8.24

```
QuadTreeNode* NEUncle(QuadTreeNode *Node)
{
     QuadTreeNode* p = NULL;
     p = Node->Parent->Parent->NEChild;
     if(p!=NULL)
          if(p==Node->Parent)
               return Node->Parent;
          else
               return p;
     else
          return NULL;
}
```

How to Get the North-West Uncle of a Quad Tree Node

```
QuadTreeNode* NWUncle(QuadTreeNode *Node)
{
     QuadTreeNode* p = NULL;
     p = Node->Parent->Parent->NWChild;
     if(p!=NULL)
          if(p==Node->Parent)
               return Node->Parent;
          else
               return p;
     else
          return NULL;
}
```

How to Get the South-East Uncle of a Quad Tree Node

```
QuadTreeNode* SEUncle(QuadTreeNode *Node)
{
```

```
QuadTreeNode* p = NULL;
p = Node->Parent->Parent->SEChild;
if(p!=NULL)
     if(p==Node->Parent)
          return Node->Parent;
     else
          return p;
else
          return NULL;
}
```

How to get the South-West Uncle of a Quad Tree Node

```
QuadTreeNode* SWUncle(QuadTreeNode *Node)
{
     QuadTreeNode* p = NULL;
     p = Node->Parent->Parent->SWChild;
     if(p!=NULL)
          if(p==Node->Parent)
               return Node->Parent;
          else
               return p;
     else
          return NULL;
}
```

8.32 HOW ARE IMAGES REPRESENTED USING A QUAD TREE?

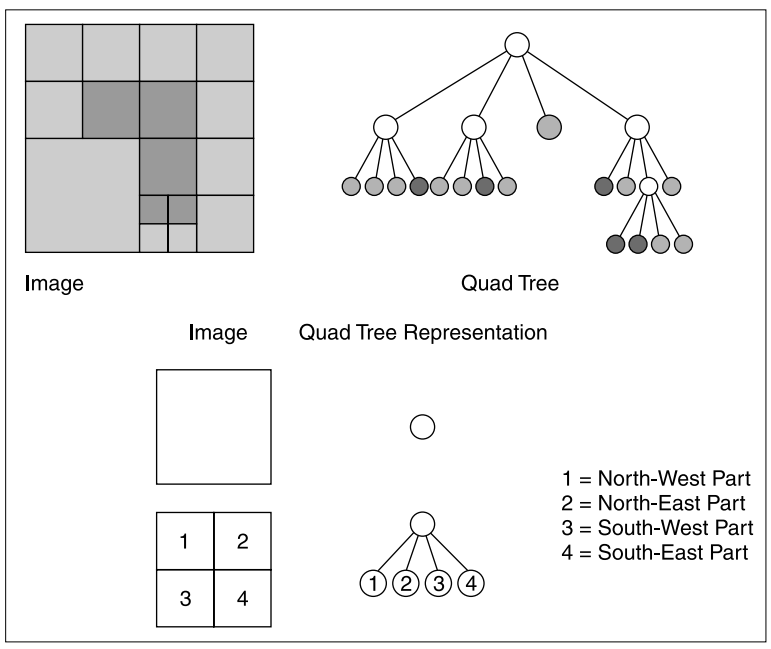


Fig. 8.25

8.33 HOW TO CONVERT A QUAD TREE TO A BINARY TREE

The basic challenge to convert a quad tree to a binary tree is to reduce couple of children per node without changing the actual representation of the quad tree.

Parts of an area that is represented by a square or rectangle can be grouped as

- North West
- North East
- South East
- South West

If we follow the above order of numbering the child of a Quad Tree node then it can be called as a *Clockwise Quad Tree*.

On the other hand, if we segregate the areas like

- North West
- South West
- South East
- North East

Then we can call it as a *Anticlockwise Quad Tree*.

So the directions are first north or south and then east or west. Logically, then we can think of any area as a binary tree that starts from blank (That's the root) and then has two children, north and south. And north and south both will have two children called east and west.

So here is a conceptual diagram of a clockwise quad tree represented as a binary tree.

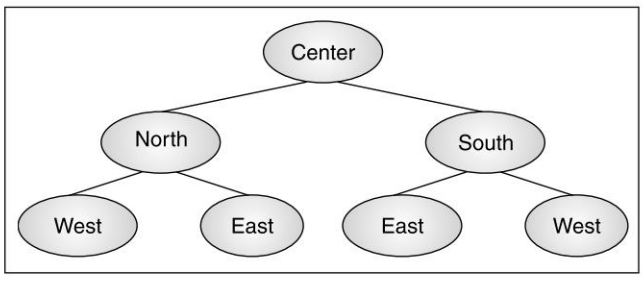


Fig. 8.26

Depending on what is our desired resolution of a binary image, we can break individual quarters into four quarters again and again. Each time we go one step forward, we get a better resolution, by adding one more level to the binary representation of the quad tree. The above image is the only first level of the binary representation. So we can say that the above image is for a

Level 1 Clockwise Binary Tree Representation of a Quad Tree

8.34 SUPERIMPOSING MULTIPLE BINARY IMAGES USING A BINARY TREE

Using the above representation of an image, we can easily superimpose two binary images, but that will have only one condition.

- A Clockwise representation can only be superimposed with a clockwise representation
- An anticlockwise representation can only be superimposed with an anticlockwise representation

These rules are not strict though, they just simplify the operation.

Here are the two pictures and their binary pixel values.

Image A

1	1	1	1	0	0	0	0
0	0	0	1	0	0	0	0
1	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
1	1	1	1	0	0	0	0
0	0	0	1	0	0	0	0
1	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0

Looks like

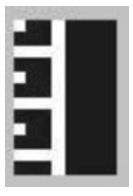


Image B

0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0

Looks like

Their superimposed image is

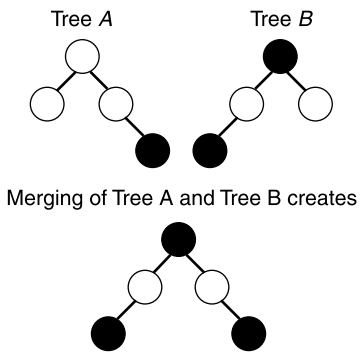
1	**1**	**1**	**1**	**0**	**0**	**1**	**0**
0	**0**	**0**	**1**	**0**	**0**	**0**	**1**
1	**0**	**0**	**1**	**1**	**0**	**0**	**0**
0	**0**	**0**	**1**	**0**	**1**	**0**	**0**

And the superimposed image looks like

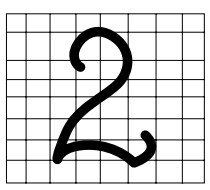
Notice carefully that the superimposing of two images are nothing but the merging of two quad trees that represent the images.

Here is a typical example of merging two binary trees whose contents are either 1 or 0. The node with a content 1 is shown as black and the other one with zero is shown as white.

0	0	0	0	0	0	0	0
0	0	0	**1**	**1**	**1**	**1**	0
0	0	0	**1**	0	0	**1**	0
0	0	0	0	0	**1**	0	0
0	0	0	0	**1**	0	0	0
0	0	0	**1**	0	0	0	0
0	0	**1**	**1**	**1**	**1**	**1**	0
0	0	0	0	0	0	0	0



8.35 HANDWRITING RECOGNITION USING QUAD TREE



See the above 8 * 8 matrix. The first matrix shows a Handwritten 2 and the next one shows a digital representation of it. The blank boxes represent the blank spaces.

We can represent the above matrix as a quad tree. Then by traversing we can say what number is written.

The idea here is to keep a tolerance level. That means there should be more than one quad tree to represent a handwritten number, because people's handwriting differs a lot.

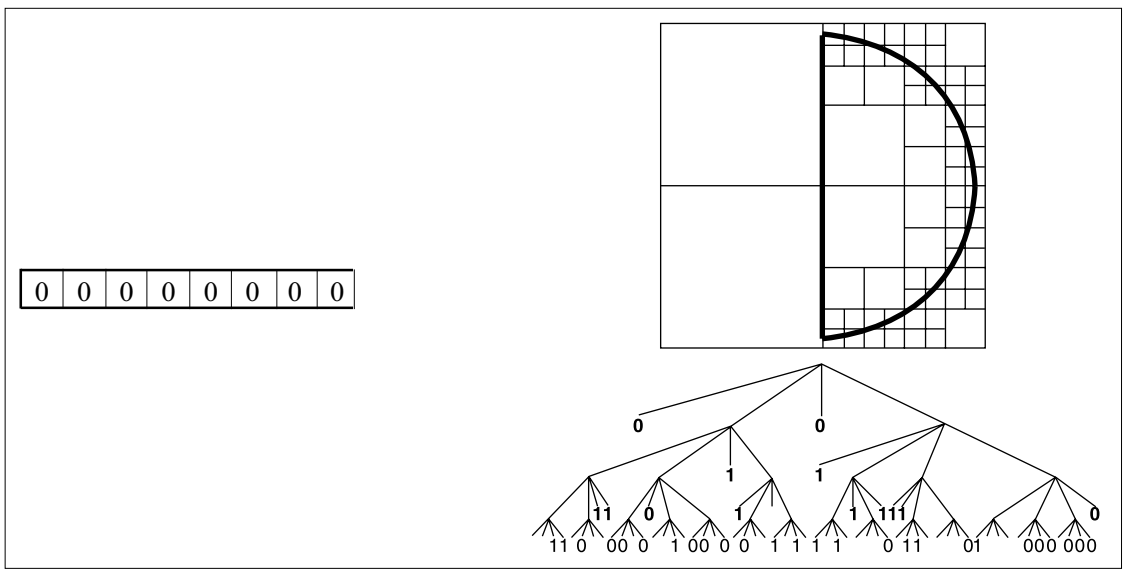


Fig. 8.27

See how the curve is represented with a quad tree.

8.36 HOW TO COMPRESS IMAGES USING A QUAD TREE

The image given below shows the gradual compressed versions of a base gray level photograph.

Fig. 8.28

Let's assume that the first image consists of 512 different segments, then the next level of the compressed image will have 256 different segments, and so on. Thus this image compression can be modeled using a pyramid (Popularly known as *Image Pyramid*). The base of the pyramid holds the image with highest resolution while the tip of the image pyramid holds the point image (Single Pixel Image) with no resolution (Lowest Resolution).

See the illustration below. The cells at the base of the pyramid can be represented by the leaves of a quad tree. Then the next level of the image consists of cell with values which are the mean of 4 leaves/children.

Thus the size of the image is decreased and the resolution as well.

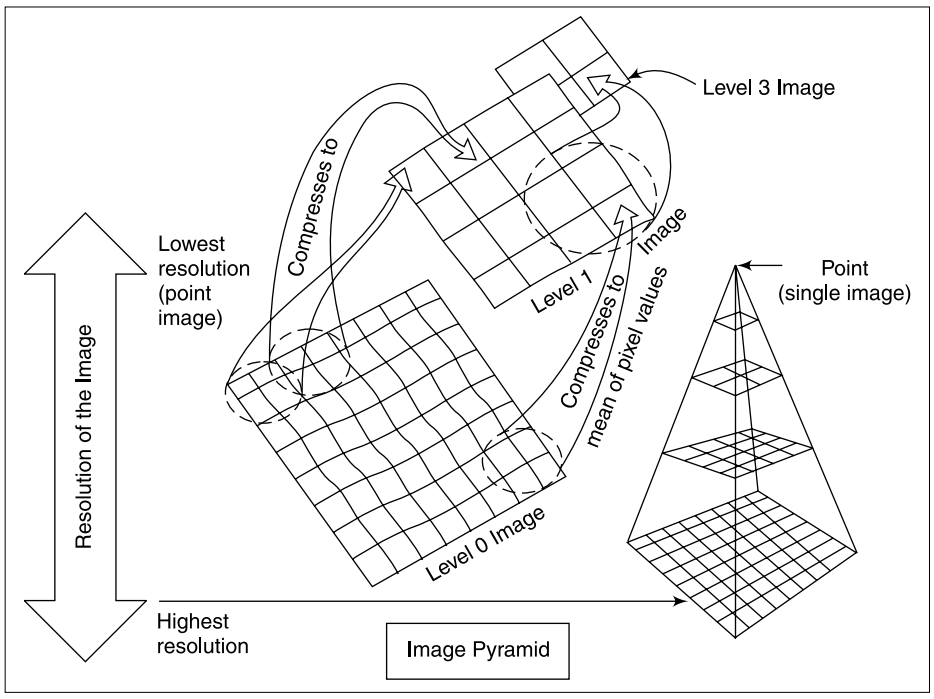


Fig. 8.29 *Image Pyramid Construction of Gray Level Images*

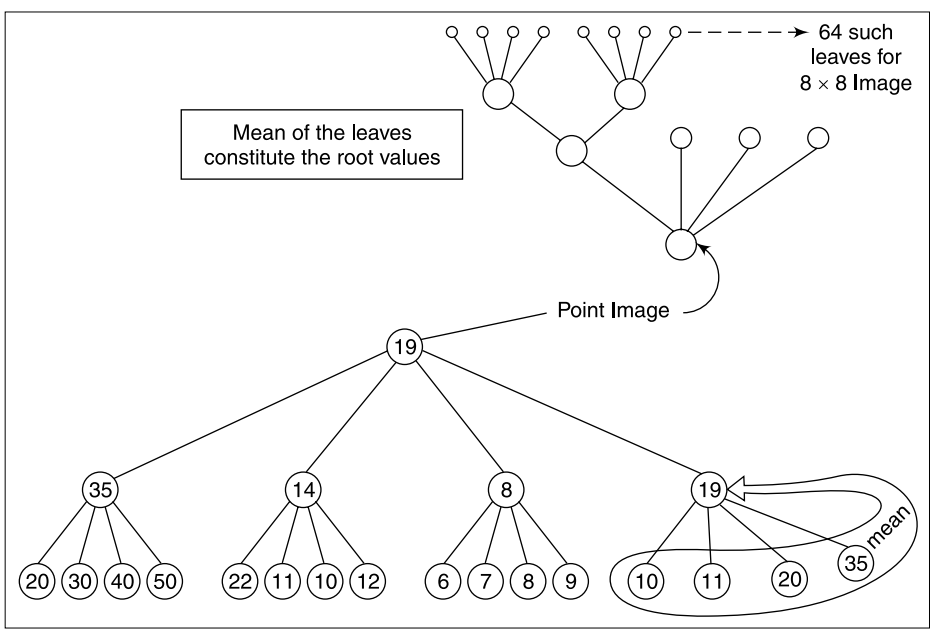


Fig. 8.30 *See how the values of the leaves are used to calculate the gray level value at the parent.*

8.37 WHAT IS AN OCTREE?

An Octree is a variant of BSP tree like a quad tree. Octree is a 3D cousin of quadtree. In this tree, each node can have at the maximum of 8 children. Octrees are more often used to partition a three-dimensional space. This tree is highly used in computer graphics to generate solids. In games, it is used to detect collisions in a 3D environment.

Octree is used in

- 3D Collision detection
- Computer Graphics Generation

See how each node in the octree holds the part of the sphere and we know exactly who are the neighbors of a particular node. Every node has 8 closest neighbors.

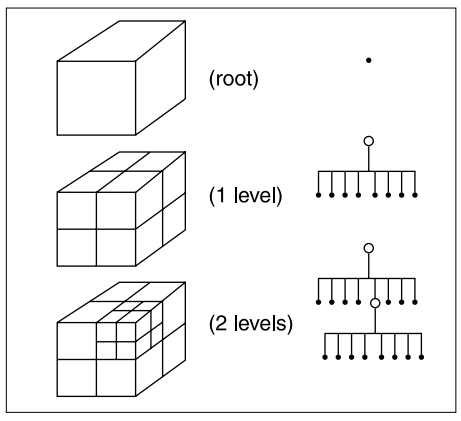


Fig. 8.31 *The image above shows construction of Octree*

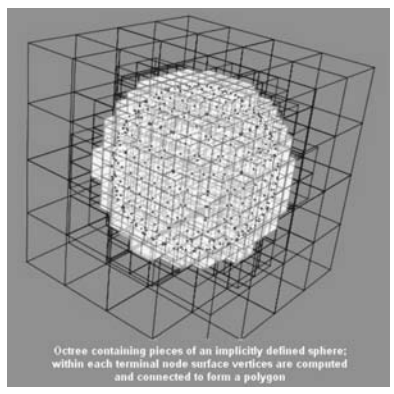


Fig. 8.32 *Image Shows an application of Octree*

How to Model an Octree using C Structure

```
typedef struct OctreeNode
{
     int value;
     struct OctreeNode *Up;
     struct OctreeNode *Down;
     struct OctreeNode *Left;
     struct OctreeNode *Right;
     struct OctreeNode *Front;
     struct OctreeNode *Back;

}OctreeNode;
```

How to Add an Up Neighbor to an Octree Node

```
void AddUp(OctreeNode *Me, int v)
{
     Me->Up = (OctreeNode *)malloc(sizeof
     (OctreeNode));
     Me->Up->value = v;
     Me->Up->Up = NULL;
     Me->Up->Down = Me;
     Me->Up->Front = NULL;
     Me->Up->Back = NULL;
     Me->Up->Right = NULL;
     Me->Up->Left = NULL;
}
```

How to Add a Down Neighbor to an Octree Node

```
void AddDown(OctreeNode *Me, int v)
{
     Me->Down = (OctreeNode *)malloc(sizeof(OctreeNode));
     Me->Down->value = v;
     Me->Down->Up = Me;
     Me->Down->Front = NULL;
     Me->Down->Back = NULL;
     Me->Down->Right = NULL;
     Me->Down->Left = NULL;
}
```

How to Add a Left Neighbor to an Octree Node

```
void AddLeft(OctreeNode *Me, int v)
{
     Me->Left = (OctreeNode *)malloc(sizeof(OctreeNode));
     Me->Left->value = v;
     Me->Left->Up = NULL;
     Me->Left->Down = Null;
     Me->Left->Front = NULL;
     Me->Left->Back = NULL;
     Me->Left->Right = Me;
     Me->Left->Left = NULL;
}
```

8.38 WHAT IS A TRIE?

Tries and suffix trees are the most popular data structures on words. Tries were introduced in 1960 by *Fredkin* as an efficient method for searching and sorting digital data. The name Trie came from *Information Re**trie**val*. This is a special type of tree that stores strings and it facilitates fast retrieval. If we notice carefully we will note that trie is nothing but a *Directed Acyclic Word Graph (DAWG). We will understand this term in Graph Chapter.* The reason, trie is the best data structure when we need to frequently search a string from a collection of strings, is that , the search time is $O(m)$ where m is the length of the string being sought. The search time is independent on the number of strings in the collection or the length of the longest string in the collection.

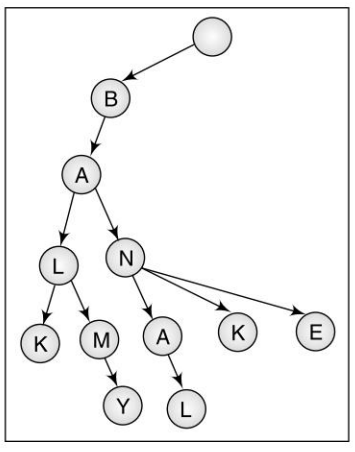


Fig. 8.33

In the above trie, the words BALK, BALMY, BANAL, BANK and BANE are stored. It is noticeable that parts of these strings are identical. This tree is very useful for information retrieval and finds application in the following areas. Numerous trie applications were found such as

- Dictionary Representation (Although binary tree is a better choice)
- Approximate String Matching
- Spell Checking Software
- Dynamic Key Matching (For personal identification on different online systems)
- Dynamic Hashing
- Conflict Resolution Algorithms
- Leader Election Algorithms
- IP addresses lookup
- Coding
- Polynomial Factorization
- Lempel-Ziv compression schemes, and so on

8.39 HOW TO MODEL A TRIE USING LINKED LIST

A trie can be represented easily with the linked list. Here, we will represent a trie which stores the words starting with a particular letter. Each node of such a trie could be represented by the following structure. The next is an array that is meant to hold the address of 26 possible children of the node.

```
typedef struct TrieNode
{
     char c;
     struct TrieNode* Next[26];
}TrieNode;
```

8.40 HOW TO ADD A KEY TO A TRIE

This function determines what the integer equivalent of the character argument passed is. If we pass 'a' to determine, the value returned will be 0 and if we pass 'z' to determine, the value returned will be 25.

```
int Determine(char c)
{
      int i = 0;
      char alphabet[] ={"abcdefghijklmnopqrstuvwxyz"};
      for(i=0;i<26;i++)
            if(c==alphabet[i])
                  return i;
}


void AddAKey(TrieNode *Root,char *Key)
{
     int i = 0;
     int j = 0;
     int index = 0;
      for(i=1;i<strlen(Key);i++)
      {
            index = Determine(Key[i]);
            if(Root->Next[index]==NULL)
            {
             Root->Next[index] = (TrieNode *)malloc(sizeof(TrieNode));
                  Root->Next[index]->c = Key[i];
                  Root = Root->Next[index];
                  for(j=0;j<26;j++)
                        Root->Next[j] = NULL;
            }
            else
                  Root = Root->Next[index];
      }
}
```

8.41 HOW TO SEARCH A KEY IN A TRIE

There are two ways by which we can search a tree. Either by passing the key being sought to the function below or by checking characterwise, the late being more useful to the basic purpose of trie. We will explore the later method next.

```
int Search(TrieNode *Root,char *Key)
{
     enum {NOTFOUND,FOUND};
     int index = 0;
     if(Root==NULL)
          return NOTFOUND;
     else
     {
```

```
for(int i=1;i<strlen(Key);i++)
{
     index = Determine(Key[i]);
     if(Root->Next[index]!=NULL)
          Root=Root->Next[index];
     else
          return NOTFOUND;
}
     }
     return FOUND;
}
```

Example 8.3 ***Write a program that creates a Trie of few words and then allows the user to search a Key as entered and reports whether the Key is present in the Trie or not.***

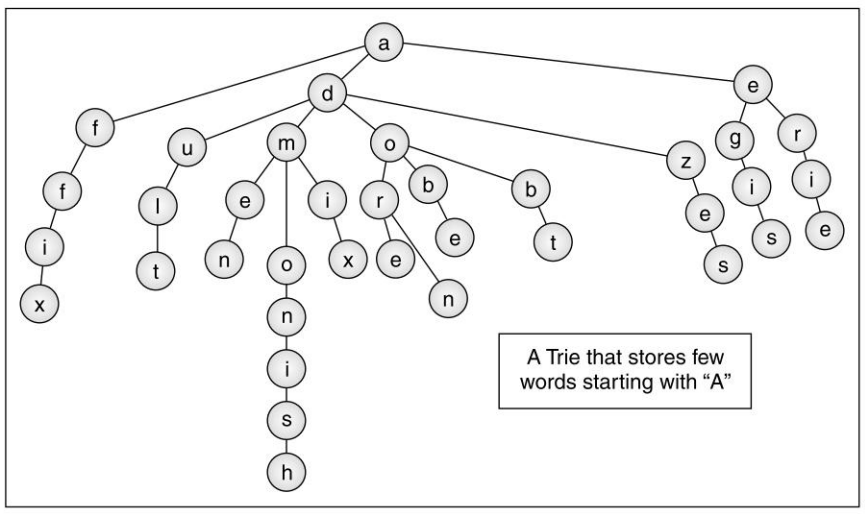


Fig. 8.34

Solution

```
int main()
{
     int i  = 0;
     char c;
     int flag = 0;
     TrieNode *Root=NULL, *Head = NULL;

     Root = (TrieNode *)malloc(sizeof(TrieNode));
//All the words in this trie starts with a
Root->c = 'a';
for(i=0;i<26;i++)
     Root->Next[i]=NULL;
Head = Root;
//Adding few keys to the trie
AddAKey(Root,"admen");
AddAKey(Root,"admix");
AddAKey(Root,"admonish");
```

```
     AddAKey(Root,"adobe");
     AddAKey(Root,"adopt");
     AddAKey(Root,"adore");
     AddAKey(Root,"adorn");
     AddAKey(Root,"adult");
     AddAKey(Root,"adzes");
     AddAKey(Root,"aegis");
     AddAKey(Root,"aerie");
     AddAKey(Root,"affix");
//The picture above displays the Trie created using these words
      printf("%d\n",Search(Head,"adze"));
printf("%c",Root->Next[Determine('d')]->Next[Determine('m')]
->Next[Determine('i')]->Next[Determine('x')]->c);
      puts("\nYour Key");
     //Searching the Trie for Key.
      c = getche();
      if(c=='a')
      {
           for(i=0;;i++)
           {
                c = getche();
                if(c==13)
                     break;
                if(Head->Next[Determine(c)]!=NULL)
                {
                     Head = Head->Next[Determine(c)];
                     continue;
                }
                else
                {
                     puts("\nNo such key");
                     flag = 1;
                     break;
                }
           }
           if(flag==1)
                puts("\nNo Such Key");
           else
                puts("\nFound the Key");
      }
      else
           puts("\nNo Such Key");
      getch();
      return 0;
}
```

Here are a few sample runs of the above program.

Fig. 8.35

As soon as '*n*' is keyed in, the program reports that there is no such key.
Another run,

Fig. 8.36

"adz" is a partial string, i.e. is there in the trie. So the search function is intelligent enough to find out even a substring from the list.

8.42 HOW TO FIND WHETHER A KEY IN A TRIE CAN BE DELETED OR NOT

In order to delete a key from a trie, we need to be sure that the key exists in the trie.

But that is not enough. Even if the key exists in the trie, we can't necessarily delete it, because the last node can be parent of some other node.

```
int CanDeleteThisKey(TrieNode *Root,char *Key)
{
     int index = 0;
     int flag = 0;
     int i = 0;
     int j = 0;
     //The Key doesn't exist in the Trie
     //So we can't delete it.
     if(Search(Root,Key)!=1)
     {

          return 0;
     }
     else
     {
          for(i=0;i<strlen(Key);i++)
          {
               index = Determine(Key[i]);
               if(Root->Next[index]!=NULL)
                    Root = Root->Next[index];
               else
               {
                    flag = 1;
                    break;
               }
          }
          if(flag == 0)
          {
               for(j=0;j<26;j++)
               {
                    if(Root->Next[j]!=NULL)
                    {
                         flag=1;
                         break;
                    }
               }
```

```
                if(flag==1)
                      return 0;
                else
                      return 1;

           }

     }
}
```

8.43 HOW TO USE A TRIE FOR SPELL CHECKING

We are all familiar with the spell check feature of Microsoft Word. We can design this with tries.

Here is a function that scans a trie of words and finds the possible correct words for the wrong word which is wrong only at the last letter like the above word "Fluctuatex".

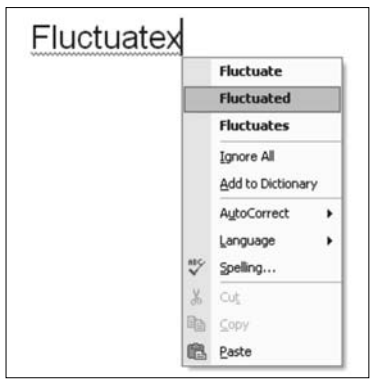


Fig. 8.37

```
void DidYouMeanThese(TrieNode *Root,char *Key)
{
      int i = 0;
      int index = 0;
      char temp[25];
      strcpy(temp,Key);
      //The key is not there in the Trie

      if(Search(Root,Key)!=1)
      {
            for(i=0;i<strlen(Key);i++)
            {
                  index = Determine(Key[i]);
                  if(Root->Next[index]!=NULL)

                        Root = Root->Next[index];
                  else
                        break;
            }//Now we stand at a location
            //Who doesn't have a children to complete this word

            for(int k=0;k<26;k++)
            {

                  if(Root->Next[k]!=NULL)
                  {
                        for(i=0;i<strlen(Key)-1;i++)
                              printf("%c",Key[i]);
                        printf("%c\n",Root->Next[k]->c);
                  }
            }

      }

}
```

When this function is called like

```
DidYouMeanThese(Root,"fluctuatex");
```

We get the following result.

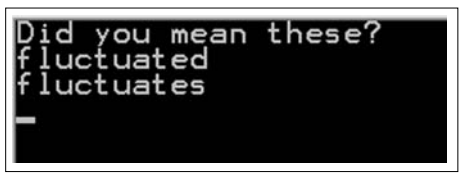


Fig. 8.38

Try Yourself: *Try to change or enhance the above function to spot spelling errors anywhere in the string.*

REVIEW QUESTIONS

1. Add the following numbers to a binary tree 10,20,30,40,50,11,23,45.
2. Draw the binary tree at each step for the above question.
3. In-order traversal of a binary tree yields A, B, C, D, E. Draw the binary tree.
4. Pre-order traversal of a binary tree yields A, B, C, D, E. Draw the binary tree.
5. Post-order traversal of a binary tree yields A, B, C, D, E. Draw the binary tree.
6. What will be the balance factor of the binary tree constructed at question 1 after adding 77 to it.
7. Is 30 alone? If not what is its sibling?
8. How many uncle will 77 have after addition to the tree?
9. Write down the operations (Zig and Zag only) in sequence that will make the tree in question number 1 after addition of 77 balanced.
10. What will be the result of one zag operation on the tree at 1 before addition of 77.
11. What do you mean by B Tree? Where are they applied?

PROGRAMMING PROBLEMS

1. Create a program to demonstrate Red Black Tree.
2. Create a program to demonstrate Heap.
3. Create a program to demonstrate B Trees.
4. Create a program to demonstrate B+ Trees.
5. Create a program to demonstrate B– Trees.
6. Create a program to demonstrate how decision trees can be used to diagnose diseases.
7. Write a function to add an element in a red black tree.
8. Write a function to delete an element from a red black tree.
9. Write a function to add an element in the AA tree.
10. Write a function to delete an element from the AA tree.
11. Write a program to demonstrate operations on a B tree.
12. Write a program to demonstrate operations on a B+ tree.
13. Write a program to demonstrate operations on a B– tree.
14. Write a program to demonstrate operations on a B* tree.
15. Write a program to demonstrate game trees.
16. Write a function to add a question to the binary spider (as mentioned in the chapter).
17. Write a function to delete a question from the binary spider.

9

Graphs
Mathematics to WAN

INTRODUCTION

In this chapter we will learn about a new data structure called graph, which is a more generalized form of tree data structure. Graphs are probably the most efficient and most used data structure to solve the problems in our practical life starting from finding a place in a map to analyze RNA reactions, structural engineering strength analysis, pattern identification in an epidemic etc. In general in a problem that has *n* number of parameters involved can be modeled using a graph. In this chapter the graph representation will be discussed.

9.1 WHAT ARE THE DIFFERENT WAYS GRAPHS ARE REPRESENTED?

There are two well-known ways by which graphs are represented in the computer memory. Namely

- Adjacency Matrix
- Adjacency List

In the first approach of representation the edges in the graphs are represented as values in the matrix. If the graph is directed, then the value of 1 at a location (i, j) indicates that there is an edge between vertex i and vertex j.

Here is an example:

The adjacency matrix is very useful data structure for representing the graph related algorithms in efficient ways. But the only disadvantage of this data structure is that number of vertices needs to be mentioned at the starting of the process.

9.2 HOW TO ADD AN EDGE IN A GRAPH MODELED BY ADJACENCY MATRIX

To add an edge between edge i and j we need to set $M(i, j) = 1$ where M is the adjacency matrix of a graph G. In this chapter we will deal with *Adjacency List of Directed Graph*.

```
int AdjMat[10][10]={0};//Adjacency Matrix of a Graph of 10 nodes

void AddEdge(int startingVertex,int endingVertex)
{
      if(AdjMat[startingVertex][endingVertex]==0)
            AdjMat[startingVertex][endingVertex] = 1;
}
```

9.3 HOW TO REMOVE AN EDGE IN A GRAPH MODELED BY ADJACENCY MATRIX

Removing an edge from a graph is simple as adding it. We shall just have to reassign the element at i, j to zero from 1. Here is the function that removes an edge from a graph represented by adjacency matrix.

```
void RemoveEdge(int startingVertex,int endingVertex)
{
      AdjMat[startingVertex][endingVertex]=0;
}
```

9.4 WHAT IS A PATH MATRIX?

The adjacency matrix tells us whether there exists any direct path (edge) between a pair of nodes or not. It doesn't tell us about the availability of paths between the two nodes. A path matrix is the adjacency matrix raised to different powers. This path matrix when created as the square of the incident matrix denotes the number of paths between the two nodes with three nodes and so on.

How to Find whether There is a Path between Two Nodes or Not

```
If the values of a Path Matrix a

int PathMatrix[MAX][MAX]={0};

void GetPathMatrix2()
{
      int cr = MAX;
      int cc = MAX;
      int i,j,k;
            //loop control

      for( i = 0; i < MAX; i++)
            for( j = 0; j < MAX; j++)
                  for( k = 0; k < MAX; k++)
                        PathMatrix[i][j]+=AdjMat[i][k]*AdjMat[k][j];
}
```

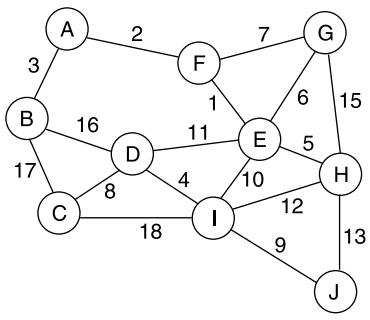
9.5 HOW TO FIND WHETHER A GRAPH IS A TREE OR NOT

A graph is said to be a tree if there exists maximum one path between a pair of nodes or vertices. In other words it can be said that a graph which is minimally connected is called a *Tree*.

```
int isTree(int AdjMat[MAX][MAX])
{
      int i,j;
      for(i=0;i<MAX;i++)
            for(j=0;j<MAX;j++)
                  if(AdjMat[i][j]>0  && AdjMat[j][i]>0)
                        return 0;
      return 1;
}
```

9.6 WHAT IS MINIMUM SPANNING TREE OF A GRAPH?

A simple graph of *n* vertices will have 2^*n* (^ denotes power) number of trees. In a weighted undirected graph the tree that has the minimum total weight is defined as *Minimum Spanning Tree* because this tree spans over all the vertices of the graph and the total cost of traveling this tree is minimal.



The red edges denote the edges of the minimum spanning tree of the graph.

There are two well-known algorithms for calculating the minimum spanning tree of a given graph.

1. Prim's Algorithm
2. Kruskal's Algorithm
3. Reverse Delete Algorithm

9.7 PRIM'S ALGORITHM

This algorithm was discovered in 1930 by mathematician *Vojtěch Jarník* and later in 1957 by computer scientist *Robert C Prim* and again re-discovered by *Dijkstra* in 1959. So this algorithm is also known as **DJP algorithm** or **Prim Jarnik Algorithm**,

The algorithm works as follows.

Step 1 First an arbitary vertex is selected.

Step 2 The vertex with the minimum edge distance from the vertex selected in step 1 is found and the edge connecting these two vertices are added as one edge of the minimum spanning tree.

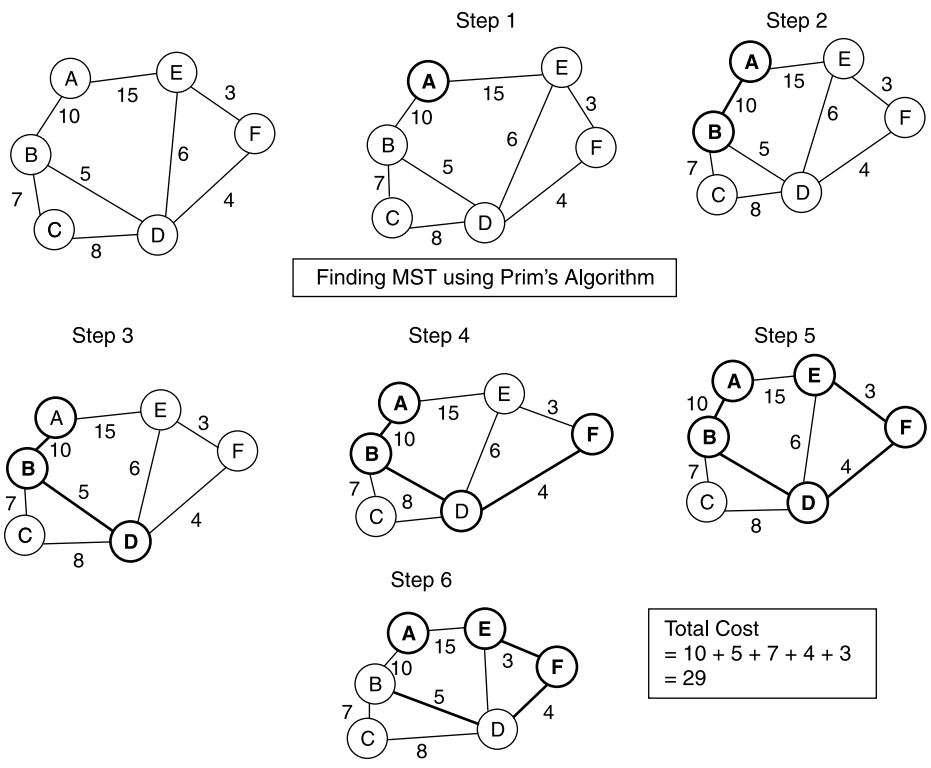
Step 3 Repeat Steps 4 and 5 untill all the vertices are in the spanning tree.

Step 4 Find a vertex which has an edge of minimum edge and not already in the spanning tree.

Step 5 Add that vertex in the spanning tree.

Step 6 Now all the vertices are in the spanning tree so stop.

The figures on next page describes the *minimum spanning tree* (MST) creation using Prim's algorithm.



Notice the figures carefully to understand Prim's algorithm

First A is selected as the arbitrary vertex.

Then B is found to be the closest vertex from A, so B is added and the edge A-B becomes a part of the MST. Later from B, D is the closest vertex and the process goes on. One thing is if both the vertices are already in the MST, then even if their edge (Which is being considered right now) have the smaller weight in comparison with others, will not be added to the MST, because that will cause the tree to have a circuit and the MST will not be a tree any more.

Try Yourself: *Try to implement Prim's algorithm using Adjacency List representation of Graph described below.*

9.8 KRUSKAL'S ALGORITHM TO FIND THE MST

This algorithm is due to the american mathematician *Joseph Bernard Kruskal*. This algorithm is simpler to implement than Prim's. The algorithm works as follows.

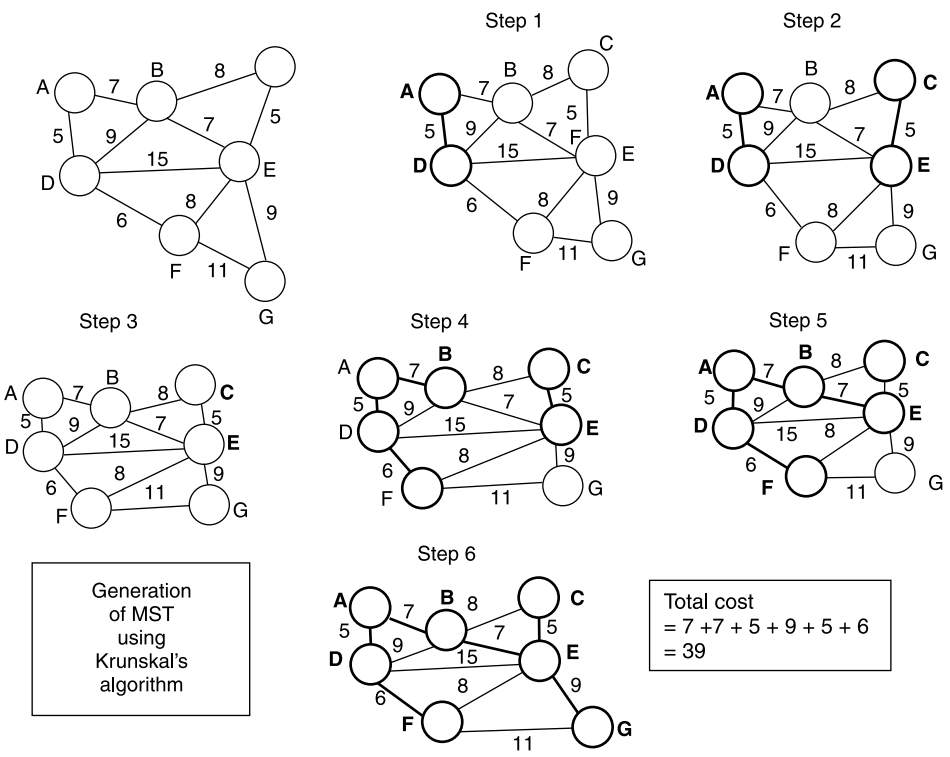
Step 1: The edges are sorted in the ascending order.

Step 2: An empty set of edges is used to create the MST.

Step 3: The edges are traversed and being added to the MST until all the vertices in the graph are traversed.

Step 4: MST is prepared.

The following figure shows how Kruskal algorithm works.



A-D and C-E both these edges have the same weight. So any of these two is selected first. In this case we have selected A-D. Once A-D is selected, obviously the next minimum is C-E. So C-E is added as an edge of the MST. Then the next smallest weight edge is sought and found to be D-F in this case. So D-F is added to the MST. This process continues until all the vertices of the original graph is traversed. Thus we get the MST.

9.9 REVERSE DELETE ALGORITHM TO FIND THE MST

This algorithm works in just the reverse way than Kruskal's. Instead of adding the edges to an empty set, this algorithm deletes the edges from the original graph so that any such deletion doesn't disconnect the graph. At the end of the process we are left with the edges of the MST.

The figure on next page shows how reverse delete algorithm works.

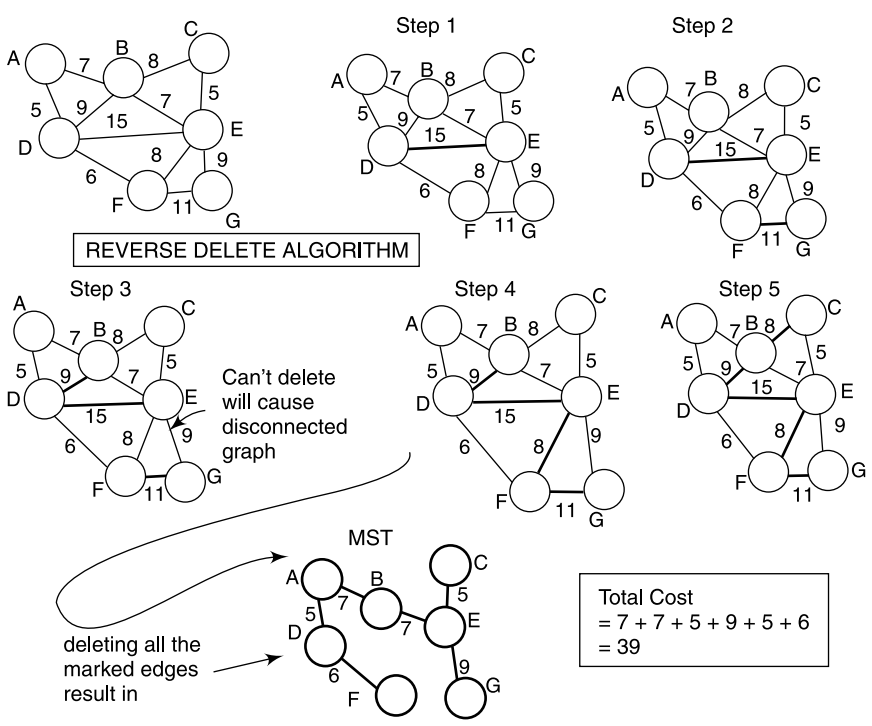
The algorithm works as follows.

Step 1: Sort all the edges in the descending order.

Step 2: Delete edges so that the deletion doesn't leave the graph disconnected.
Stop when deleting any further edge makes the graph disconnected.

Step 3: The remaining edges constitute the MST of the given graph.

Notice in the above figure, the highest weight edge 15 is deleted first and then the second largest weight 11 is deleted and so on until we reach the MST. As the process deletes the highest weight first, thus the name *reverse delete*.

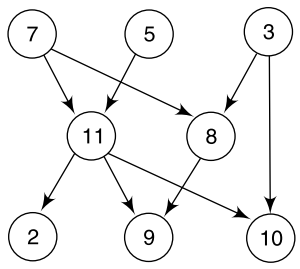


How to Find the Shorted Path using Warshall's Algorithm

```
int minoftwo(int a,int b)
{
      if(a==b)
            return a;
      else
            return a>b?a:b;
}
void WarshallsShortestPath(int AdjMat[MAX][MAX])
{

      int i,j,k;
      int Q[MAX][MAX]={0};
      for(i=0;i<MAX;i++)
      {
            for(j=0;j<MAX;j++)
            {
                  if(AdjMat[i][j]==0)
                        Q[i][j]=INF;
                  else
                        Q[i][j]= AdjMat[i][j];
            }
      }
      for(k=0;k<MAX;k++)
      {
            for(i=0;i<MAX;i++)
```

```
        {
                for(j=0;j<MAX;j++)
                {
                        Q[i][j] = minoftwo(Q[i][j],Q[i][k]+Q[k][j]);
                }
        }
}
```

9.10 WHAT IS A DIRECTED ACYCLIC GRAPH OR DAG?



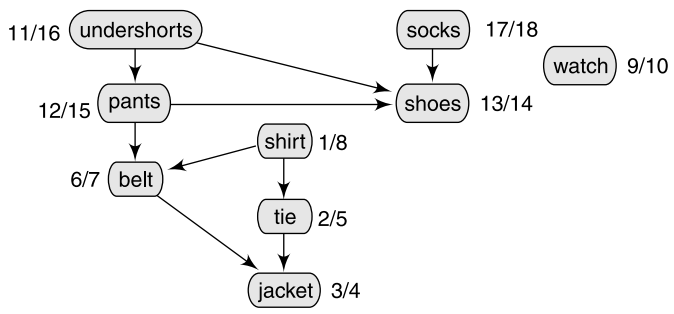
A DAG is a directed graph with no cycles. That means for a vertex *v* in a DAG there is no directed path that starts and ends on *v*. Here is an example of a DAG.

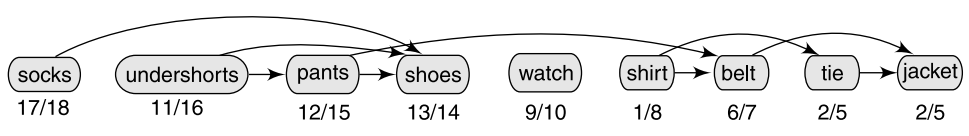
In a DAG the vertex that has only edges ending on it is called a *sink* and the vertex that has only edges starting from it is called a *source*. There can be vertices other than these two types also. As we can see in the above figure, vertices 7, 5, 3 are source vertices and 2, 9, 10 are sink vertices while vertices 11 and 8 are none of these because there are edges starting and ending on them. In the chapter on tree, the trie is a DAG where the content of the vertices are characters.

DAGs find application in many diverse areas. To name a few

1. Bayesian Networks
2. Instruction Scheduling
3. Makefile Instructions
4. Information Categorization System as the folder in a computer network

9.11 WHAT IS TOPOLOGICAL SORTING OF A DAG?





Topological Sorting is a special way to arrange the elements of a DAG.

Topological Sorting is used to design scheduling applications where one job needs to be done first before the other. For example, a course in calculus demands that you understand trigonometry well. So before someone enrolls for calculus course they have to learn trigonometry. In a DAG trigonometry and calculus can be mapped as two nodes where the edge will travel from trigonometry to calculus.

Given a graph the following algorithm calculates the topologically sorted ordering of graph elements.
Step 1: In Degree of each node is calculated
Step 2: All the nodes with in degree = 0 are put in a queue
Step 3: Step 4 and 5 are repeated until the queue is empty
Step 4: The front node N of the queue is removed
Step 5: The following steps are repeated for each neighbor M of node N
  a. In Degree of M is reduced by 1
  b. If In Degree = 0 then add M to the rear of the queue
Step 6: The Queue now has the topological ordering of the elements.

Try Yourself: *Write a function that will accept a DAG and will print its topological ordering. Use Linked Representation (Adjacency List) of Graph.*

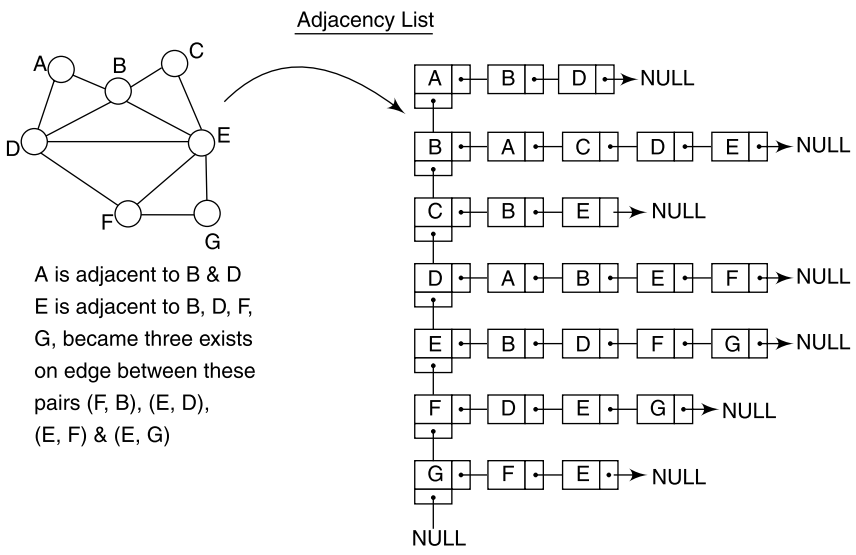
How to Find the Union of Two Graphs Represented using Matrix

```
void Union(int AdjMat1[MAX][MAX],int AdjMat2[MAX][MAX])
{
      int AdjMatUnion[MAX][MAX];
      int i,j;
      for(i=0;i<MAX;i++)
            for(j=0;j<MAX;j++)
                  AdjMatUnion[i][j]=AdjMat1[i][j] | AdjMat2[i][j];
}
```

How to Find the Intersection of Two Graphs Represented using Matrix

```
void Intersection(int AdjMat1[MAX][MAX],int AdjMat2[MAX][MAX])
{
      int AdjMatUnion[MAX][MAX];
      int i,j;
      for(i=0;i<MAX;i++)
            for(j=0;j<MAX;j++)
                  AdjMatUnion[i][j]=AdjMat1[i][j] & AdjMat2[i][j];
}
```

How to Model an Undirected Graph using Adjacency List



```
enum {NO,YES};
//This structure represents the Edge
typedef struct Edge
{
      int StartingVertex;
      int EndingVertex;
}Edge;
//This structure represents the Edge Collections.
typedef struct Edges
{
      Edge edge;
      struct Edges *next;
}Edges;

Edges *GraphEdges = NULL;

//This structure represent a vertex of the Graph
typedef struct node
{
      int data;
      struct node *next;
}node;

//This structure represent the entire Graph.
typedef struct GraphNode
{
      int id;
      GraphNode *next;
      node *connections;
}GraphNode;
```

How to Add a New Vertex in the Graph

```
GraphNode* AddVertex(GraphNode *graph,int newvertex)
{
      //Not Present in the graph already
     if(isVertexPresent(graph,newvertex)==0)
      {
          if(graph==NULL)
          {
               GraphNode *newVertex
               = (GraphNode *)malloc(sizeof(GraphNode));
               newVertex->id = newvertex;
               newVertex->connections = NULL;
               newVertex->next = NULL;
               return newVertex;
          }
          else
          {
               GraphNode *newVertex
               = (GraphNode *)malloc(sizeof(GraphNode));
```

```
            newVertex->id = newvertex;
            newVertex->connections = NULL;
            newVertex->next = NULL;
            graph->next = newVertex;
            //Returning the last node of the graph.
            return newVertex;
      }
}
```

How to Add a New Edge to the Graph

```
Edges* AddEdge(GraphNode *graph
  ,Edges *GraphEdges
  ,int StartingVertex
  ,int EndingVertex)
{
      Edges *e = NULL;
      Edge f;
      if(isVertexPresent(graph,StartingVertex)==0
      && isVertexPresent(graph,EndingVertex)==0)
            printf("Can't add Edge");
      else
      {
            GetAddress(graph,StartingVertex)->connections
        = push_front(GetAddress(graph,StartingVertex)->connections,EndingVertex);

            f.StartingVertex = StartingVertex;
            f.EndingVertex = EndingVertex;
            if(EdgeAlreadyPresent(GraphEdges,f)==NO)
                  GraphEdges =  PushEdgeToFront(GraphEdges,f);
      }
      return GraphEdges;
}
//Adding an edge between vertex 2 and 3 means
//adding an edge between vertex 3 and 2 as well.

Edges* InsertEdge(GraphNode *graph
 ,Edges *GraphEdges
 ,int StartingVertex
 ,int EndingVertex)
{
      GraphEdges = AddEdge(graph,GraphEdges, StartingVertex,EndingVertex);
      GraphEdges = AddEdge(graph,GraphEdges,EndingVertex,StartingVertex);
      return GraphEdges;
}
```

How to Softly Delete an Edge from a Graph

```
Edges* RemoveEdge(Edges *e,int StartingVertex,int EndingVertex)
{
Edges *p = e;
Edges *x = NULL;
for(;p!=NULL;p=p->next)
{
      if((p->edge.StartingVertex == StartingVertex
         && p->edge.EndingVertex == EndingVertex)
      ||(p->edge.StartingVertex == EndingVertex
```

```
            && p->edge.EndingVertex == StartingVertex))
        {
            p->edge.Deleted = 1;
            break;
        }
    }
    return e;
}
```

How to Display the Edges of a Graph

```
void DisplayEdges(Edges *e)
{
    Edges *p = e;
    for(;p!=NULL;p=p->next)
        if(p->edge.Deleted!=1)
            printf("%d-----%d\n",
              p->edge.StartingVertex,p->edge.EndingVertex);

}
```

How to Softly Delete a Vertex from a Graph

```
Edges* RemoveVertex(Edges *e,int Vertex)
{
    Edges *p = e;
    Edges *x = NULL;
    for(;p!=NULL;p=p->next)
    {
        if(p->edge.EndingVertex == Vertex)
            e = RemoveEdge(e,p->edge.StartingVertex,Vertex);
        if(p->edge.StartingVertex == Vertex)
            e = RemoveEdge(e,Vertex,p->edge.EndingVertex);
    }

    return e;
}
```

How to Find Whether a Vertex is Present or Not in the Graph

```
int isVertexPresent(GraphNode *graph,int VertexId)
{
   int Found = NO;
   if(graph==NULL)
       return Found;
   else
   {
       while(graph->next!=NULL)
       {
           if(graph->id==VertexId)
           {
               Found = YES;
               break;
           }
           graph=graph->next;
       }
   }
   return Found;
}
```

How to Find whether an Edge is Present or Not in the Graph

```
int ContainsEdge(GraphNode *graph, int StartingVertex,int
EndingVertex)
{
     int Found = NO;
     int i = 0;
     GraphNode *g = graph;
     node *n = g->connections;
     for(;g!=NULL;g=g->next)
     {
          if(g->id == StartingVertex)
          {
               n = g->connections;
               for(;n!=NULL;n = n->next)
                    if(n->data == EndingVertex)
                    {
                         Found = YES;
                         return Found;
                    }
          }
     }
     return Found;
}
```

How to Find a List of Vertices That Have a Self-Loop around It

```
node* SelfLoopVertices(GraphNode *graph)
{
      node *h = NULL;
      int i=0;
      do
      {
            for(i = 0;i<count(graph->connections);i++)
                  if(graph->connections->data == graph->id)
                        h = push_front(h,graph->id);
            graph = graph->next;
      }while(graph->next !=NULL);
      return h;
}
```

How to Find the Degree of a Node in the Graph

```
//This function counts the number of nodes in the linked list.
int count(node *h)
{
      int numberofnodes=0;
      node *p = h;
      if(p==NULL)
            return 0;
      else
      {
            for(;p!=NULL;p=p->next)
                  numberofnodes++;
            return numberofnodes;
      }
}
```

```
//This function calculates the degree of a node.
int Degree(GraphNode *vertex)
{
      return count(vertex->connections);
}
```

How to Find whether a Vertex is Isolated or Not

```
int isIsolatedVertex(GraphNode *vertex)
{
     if(Degree(vertex)==0)
          return YES;
     else
          return NO;
}
```

What is a Pendant Vertex and How to find if a Vertex is a Pendant?

A vertex is called a Pendant if the degree of the vertex is unity.

```
int isPendant(GraphNode *vertex)
{
      return Degree(vertex)==1;
}
```

How to Display the Graph

```
void display(GraphNode *graph)
{
     do
     {
          printf("\nVertex = %d\n",graph->id);
          while(graph->connections!=NULL)
          {
               printf("\t--%d\t",graph->connections->data);
               graph->connections = graph->connections->next;
          }
          graph = graph->next;
     }while(graph != NULL);
}
```

How to Find whether Vertices of a Graph are a Subset of Vertices of Another

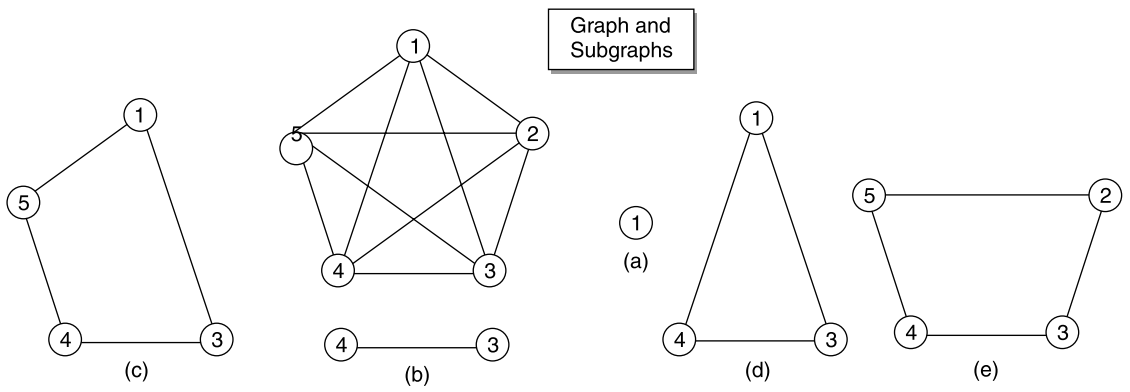
```
int CompareVertices(GraphNode *a,GraphNode *b)
{
      int VerticesMatch = NO;
      GraphNode *temp1 = a;
      GraphNode *temp2 = b;
      for(;temp2!=NULL;temp2 = temp2->next)
      {
            for(;temp1!=NULL;temp1 = temp1->next)
            {
                  if(temp1->id == temp2->id)
                  {
                        VerticesMatch = YES;

                  }
            }
      }
      return VerticesMatch;
}
```

How to Find whether Edges of a Graph are a Subset of Edges of Another

```
int CompareUndirectedEdges(Edges *ea, Edges *eb)
{
      Edges *ta = ea;
      Edges *tb = eb;
      int result = 0;
      if(CountEdges(ta)>=CountEdges(tb))
      {
            for(;tb!=NULL;tb=tb->next)
            {
                  for(;ta!=NULL;ta=ta->next)
                        if((ta->edge.StartingVertex == tb->edge.StartingVertex
                           && ta->edge.EndingVertex == tb->edge.EndingVertex)
                         ||(ta->edge.StartingVertex == tb->edge.EndingVertex
                           && ta->edge.EndingVertex == tb->edge.StartingVertex))
                        result = 1;

            }
      }
      return result;
}
```

How to Find whether a Graph is a Subgraph of Another Graph or not

```
int isSubGraph(GraphNode *a,GraphNode *b,Edges *e1s,Edges *e2s)
{

      if(CountEdges(e2s)>0 && CountEdges(e1s)>0)
      //If both the graphs have got edges then we shall
      //have to test whether the edges and the vertices of
      //the suspected subgraph is a subset of the supplied
      //graph or not
            return CompareVertices(a,b) &&  CompareUndirectedEdges(e1s,e2s);
      if(CountEdges(e2s)==0)
      //If there is no edge in the second graph
      //or the suspected subgraph then we shall just have to
      //check whether the vertices of that graph is a subset of
      //the other or not
            return CompareVertices(a,b);
}
```



How to Find whether a Graph is Euler or Not

If all the vertices of a graph have even degree then the graph is said to be an Euler Graph.

```
int isEuler(GraphNode *graph)
{
      int Euler = 0;
      GraphNode *p = graph;
      for(;p!=NULL;p=p->next)
      {
            if(Degree(p)%2==0)
                  Euler = 1;
            else
                  Euler = 0;
      }
      return Euler;
}
```

How to Find whether a Graph is a Complete Graph or Not

```
int isComplete(GraphNode *graph,Edges *GraphEdges)
{
      //A Complete Graph is also known as Universal Graph or Clique
      int n = Size(graph);
      return CountEdges(GraphEdges) == (n*(n-1)/2);
}
```

9.12 HOW TO FIND WHETHER A GRAPH IS PLANER OR NOT

Graph crossing number is a number that helps to understand whether a graph is planer or not. If the graph crossing number for a graph is zero then the graph is planer else it is not.

```
int GraphCrossingNumber(GraphNode *graph)
{
      double n = Size(graph);
      return (1/4)*floor(n/2)*floor((n-1)/2)*floor((n-2)/2)*floor((n-3)/2);
}

int isPlaner(GraphNode *graph)
{
      return GraphCrossingNumber(graph)==0;
}
```

How to Model a Directed Weighted Graph using Structures

```
typedef struct Edge
{
      int StartingVertex;
      int EndingVertex;
      int Weight;
}Edge;


typedef struct Edges
{
      Edge edge;
```

```
      struct Edges *next;

}Edges;

typedef struct Digraph
{
      Edges *edges;
}Digraph;
```

How to Add an Edge to a Weighted Directed Graph

```
Edges* push_back(Edges *last,Edge e)
{
      if(last==NULL)
      {
            last = (Edges *)malloc(sizeof(Edges));
            last->edge = e;
            last->next = NULL;
            return last;
      }
      else
      {
            Edges *p = (Edges *)malloc(sizeof(Edges));
            p->edge = e;
            p->next = NULL;
            last->next = p;
            return p;
      }
}

Edges* AddAEdge(Digraph d,int StartingVertex,int EndingVertex,int Weight)
{

      Edge e;
      Digraph cd = d;

      e.StartingVertex = StartingVertex;
      e.EndingVertex = EndingVertex;
      e.Weight = Weight;
      for(;cd.edges!=NULL;cd.edges=cd.edges->next)
      {
            if(cd.edges->edge.StartingVertex == EndingVertex
            && cd.edges->edge.EndingVertex == StartingVertex)
            {
                  break;
            }
      }
      d.edges = push_back(d.edges,e);
      return d.edges;
}
```

How to Find out the In-Degree of a Node in a Digraph

The in-degree is the number of edges that end on the given vertex.

```
int InDegree(Digraph d, int Vertex)
{
      int degree=0;
```

```
      Digraph cd =  d;
      for(;cd.edges!=NULL;cd.edges = cd.edges->next)
            if(cd.edges->edge.EndingVertex == Vertex)
                  degree++;
      return degree;
}
```

How to Find out the Out-Degree of a Node in a Digraph

The outdegree is the number of edges that start from the given vertex

```
int OutDegree(Digraph d,int Vertex)
{
      int degree=0;
      Digraph cd =  d;
      for(;cd.edges!=NULL;cd.edges = cd.edges->next)
            if(cd.edges->edge.StartingVertex == Vertex)
                  degree++;
      return degree;
}
```

How to Find the Number of Parallel Edges between a Pair of Nodes

```
int ParallelEdgesCount(Digraph d,int StartingVertex,int EndingVertex)
{
      int ParallelPaths = 0;
      Digraph cd = d;
      for(;cd.edges!=NULL;cd.edges = cd.edges ->next )
            if(cd.edges->edge.StartingVertex == StartingVertex
            && cd.edges->edge.EndingVertex == EndingVertex)
                  ParallelPaths++;
      return ParallelPaths;
}
```

How to Print all the Edges That has Parallel Paths

```
void PrintAllParallelPaths(Digraph d)
{
      Digraph cd = d;
      for(;cd.edges!=NULL;cd.edges=cd.edges->next)
            if(ParallelEdgesCount(d,cd.edges->edge.StartingVertex
                 ,cd.edges->edge.EndingVertex)>1)
                  printf("%d------%d----%d\n"
                ,cd.edges->edge.StartingVertex
                ,cd.edges->edge.EndingVertex
                ,cd.edges->edge.Weight);
}
```

How to Find whether a Digraph is Balanced or Not

If in a directed graph the in-degree and out-degree of all the nodes are equal then the graph is said to be balanced.

```
int isBalanced(Digraph d)
{
      Digraph cd = d;
      for(;cd.edges!=NULL;cd.edges = cd.edges ->next)
```

```
            if(InDegree(d,cd.edges->edge.StartingVertex)
               !=OutDegree(d,cd.edges->edge.StartingVertex)
               ||InDegree(d, cd.edges->edge.EndingVertex)
               !=OutDegree(d,cd.edges->edge.EndingVertex))
            return 0;
      return 1;
}
```

How to Find whether an Edge Exists in a Digraph or Not

```
int DoesThisEdgeExist(Digraph d,int StartingVertex,int EndingVertex)
{
      Digraph cd = d;
      for(;cd.edges!=NULL;cd.edges=cd.edges->next)
            if(cd.edges->edge.StartingVertex == StartingVertex
              && cd.edges->edge.EndingVertex == EndingVertex)
                  return 1;
      return 0;
}
```

How to Find whether a Directed Graph is Symmetric or Not

A directed graph is said to be symmetric, if and only if, there exists an edge from every edge for each opposite edge. For example say *a* and *b* are two vertices of a graph then if there is an edge from *a* to *b* , then there is also an edge from *b* to *a*.

```
int isSymmetric(Digraph d)
{
      Digraph cd = d;
      for(;cd.edges!=NULL;cd.edges=cd.edges->next)
            if(DoesThisEdgeExist(d,cd.edges->edge.EndingVertex,
                cd.edges->edge.StartingVertex)==0)
                  return 0;
      return 1;
}
```

How to Find whether an Edge is a Self-Loop or Not

```
int isThisEdgeASelfLoop(Edge e)
{
      return e.StartingVertex == e.EndingVertex;
}
```

How to Find the Count of Self-Loops in a Digraph

```
int SelfLoopCount(Digraph d)
{
      Digraph cd = d;
      int count = 0;
      for(;cd.edges!=NULL;cd.edges=cd.edges->next)
            if(isThisEdgeASelfLoop(cd.edges->edge)==1)
                  count++;
      return count;
}
```

How to Find whether a Digraph is Regular or Not

A balanced digraph is said to be regular if every vertex has the same in-degree and out-degree as every other vertex.

```
int isRegular(Digraph d)
{
      int regular = 0;
      Digraph cd = d;
      Digraph ccd = cd;
      if(isBalanced(d)==1)
      {
            for(;cd.edges!=NULL;cd.edges = cd.edges->next)
            {
                  for(;ccd.edges!=NULL;ccd.edges = ccd.edges->next)
                  {
                        if(InDegree(d
                           ,cd.edges->edge.StartingVertex)
                           ==InDegree(d,ccd.edges->edge.StartingVertex)
                           && OutDegree(d
                          ,cd.edges->edge.StartingVertex)
                            ==OutDegree(d,ccd.edges->edge.StartingVertex)
                           && InDegree(d
                           ,cd.edges->edge.EndingVertex)
                           ==InDegree(d,ccd.edges->edge.EndingVertex)
                           && OutDegree(d
                          ,cd.edges->edge.EndingVertex)
                           ==OutDegree(d,ccd.edges->edge.EndingVertex))
                           regular =1;
                        else
                        {
                              regular = 0;
                              return regular;
                        }

                  }
            }

      }
      return regular;
}
```

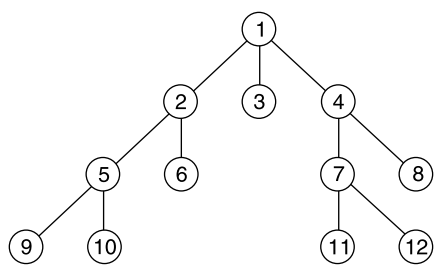
9.13 BREADTH FIRST SEARCH (BFS)

In many graph algorithms the traversal of the graph in a special way is very important. There are two ways the graph is normally traversed. They are *Breadth first search* (BFS) and *Depth first search* (DFS). In this section we will learn about BFS.

The general idea behind the BFS is starting at a node and then go on exploring the neighbors of this node untill we cover all the nodes of the graph. But we have to make sure that a node is not traversed more than once. This is achieved by using a *queue* and by using a *variable status* that depicts what is the status of the current node.

The algorithm is as follows.

Step 1: All the nodes are marked ready to be processed by assigning status = 1



This figure represents the relative ordering of elements being traversed in BFS of a graph.

Step 2: The starting node is put in the queue and its status is changed to waiting or status = 2
Step 3: The steps 4 and 5 are repeated untill the queue is empty.
Step 4: The front element *N* is removed from the queue and its status is marked processed or status = 3
Step 5: All the neighbors of the queue that are in ready status are added to the rear of the queue and their status is changed to the waiting state or status = 2.
Step 6: End. The content of the queue is the BFS.

Here is a pseudo code for BFS
The function takes three arguments

1. The graph
2. The starting node
3. The ending node

```
Parent[start] = -1//cause root doesn't have any parent
Queue = enqueue(queue,start);
while (!isEmpty(Queue))
{
    int u = frontItem(Queue);
    deQueue(Queue);
    // Here is the point where we examine the u th vertex of graph
    // For example:
    if (u == end)
          return 1;
       // Look through neighbors v.
          if (parent[v] == 0)
           {
        // If v is unvisited.
        parent[v] = u;
        enqueue(v);
      }
    }
  }
  return 0;
}
```

Applications of BFS

- Prim's MST algorithm.
- Dijkstra's single source shortest path algorithm.
- Finding all connected components in a graph.
- Finding all nodes within one connected component.
- Copying Collection, ***Cheney's algorithm***.
- Building a web search agent.
- Finding the shortest path between two nodes *u* and *v* (in an unweighted graph).
- Testing a graph for bipartiteness.

Complexities of BFS The space and time complexity of BFS is $O(|V|+|E|)$ where V is the size of vertices and E is the size of edges.

Try Yourself: *Using the adjacency list representation of graph described and defined above try to implement a function that takes a graph and prints its nodes in BFS pattern.*

9.14 DEPTH FIRST SEARCH (DFS)

The general idea of a depth first search is starting at a point we traverse all the neighbor nodes until we reach an end. Once we come to an end we backtrack to start all over again starting from another neighbor of the starting node and travel towards the new neigbors neighbors' direction, and so on. As we have to backtrack, we use a stack unlike a queue in BFS. Otherwise the algorithm is very similar to BFS.

The algorithm is as follows

Step 1: All the nodes are initialized to status 1.
Step 2: The starting node is pushed on the stack and the status of it is changed to waiting or status = 2.
Step 3: Steps 4 and 5 are repeated until the stack is empty.
Step 4: The top node N of the stack is popped and its status is changed to processed status or status = 3.
Step 5: All the neighbors of N that are still in ready status are pushed on the Stack and their status are changed to status = 2 or waiting.
Step 6: Done.

Applications of DFS

- Finding all connected components in a graph.
- Finding all nodes within one connected component
- Solving paths in a maze.

Complexities of DFS The space and time complexity of BFS is $O(|V|+|E|)$ where V is the size of vertices and E is the size of edges.

REVISION OF CONCEPTS

Some Key Facts about Graphs

- A graph is a set of vertices(V:{$v1$, $v2$,…, vn}) and a set of edges(E:{$e1$, $e2$, …, en}) that connect the vertices.
- A vertex is also referred as a node.
- A graph can be either non-directed or directed.
- In a non-directed graph, the edges are not directed from one vertex to the other.
- In directed graphs the edges are directed from one vertex to another.
- If in the directed graph the edges have different weights then the graph is known as Weighted directed graph (WDG).
- An electronic circuit can be represented as a WDG.
- Degree of a node in a non-directed graph is the number of edges meet that node.
- Degree of a node in a directed graph is denoted as the sum of in-degree and out-degree.
- In-degree of a node in a directed graph is the number of edges that end on that vertex.
- Out-degree of a node in a directed graph is the number of edges that start from that node.
- The number of nodes of odd degree in a graph is always even.
- If in a graph, all the nodes have the same degree then the graph is known as a regular graph.
- A node with degree zero is called an isolated vertex.
- A node with degree one is called a pendant node.
- If in a graph any edge starts and ends on the same vertex then that vertex is said to have a self loop.

- Graphs that do not have a self loop are sometimes referred as Simple graph.
- If there are multiple edges in the graph that start and end on same vertices, then these edges are known as Parallel edges.
- A tree is a minimally connected graph.
- A spanning tree of a graph is such a tree that covers all the vertices of the graph
- A minimum spanning tree is a tree that has the minimum edge weight.

REVIEW QUESTIONS

1. What do you mean by isomorphic graphs. What are the rules that makes two graphs isomorphic?
2. Prove that the maximum number of edges in a simple graph with N nodes is $N(N–1)/2$.
3. What can be the maximum degree of a node in a graph of N nodes.
4. Draw the graph of benzene and methane. Represent the molecules by nodes and the chemical bonds by edges.
5. Draw the graph of any electronic circuit.
6. Draw a diagram to explain how traffic signaling can be modeled using graphs.

PROGRAMMING PROBLEMS

1. Demonstrate in a program how MST can be used in different layout design problems. For example computer networking layout in a university or cable tv network in a locality.
2. Demonstrate in a program how an electrical network can be modeled using graphs.
3. Demonstrate in a program how a maze can be modeled using a program.
4. Write a program to implement the Kruskal's algorithm to get the MST of a given graph.
5. Write a program to implement the Prim's algorithm to get the MST of a given graph.
6. Write a program to implement the Reverse-delete algorithm to get the MST of a given graph.
7. Demonstrate how graphs can be used to model a pattern in a geographical survey.
8. Write a program to demonstrate how graphs can be used to map airlines ways between multiple cities.
9. Write a program to demonstrate how graphs can be used to model electric circuits.
10. Write a program to demonstrate how graphs can be used to represent Bayesian Networks.
11. Write a program to find the minimum cabling length in a locality using minimum spanning tree algorithm.
12. Write a program to find if there is more than one path between two vertices in a graph.
13. Write a program to find the shortest edge between every pair of vertices.
14. Write a program to find if there exists a path between two nodes where there are couples of nodes on the path apart from the source and destination.
15. Demonstrate how graphs can be used to model a natural calamity stricken area.

10

Sorting
Micro, Macro, Mammoth

INTRODUCTION

Sorting is a very important part of computer science. Sorting is needed as part of different algorithms and careful selection of a good sorting algorithm can really make a change in an algorithm. In this chapter we will discuss few basic sorting algorithms and analyze their performances against each other. Few sorting algorithms will be discussed and will be left for the reader to implement at the chapter end.

10.1 FUNCTIONS IN THIS CHAPTER

These programs can sort a sequence of numbers either in ascending or in descending order as indicated by the last boolean field.

These functions take the arguments by value, but return the sorted array as pointer to integer. So, in the calling function the change will not be made to the original array. What would have happened if the functions are called by *Call By Reference*? The change can be stored in any integer pointer.

10.2 SORTING ALGORITHMS CLASSIFICATIONS

This figure shows an overall view of different sorting algorithms classified under broad categories. Random sort is the group of most impractical sorting algorithms. On the other hand, *hybrid sort* is the home to the algorithms that have inherited the best qualities of their cousins in other sorting families or uses two or more algorithms as part of a single algorithm (See Fig. 10.1).

10.3 EXCHANGE SORT ALGORITHMS

Explain Bubble Sort Algorithm

Bubble sort is a *comparison sorting algorithm*. It compares consecutive elements one by one till everything is sorted. This is one of the very slow algorithms. Basically the original version of bubble sort

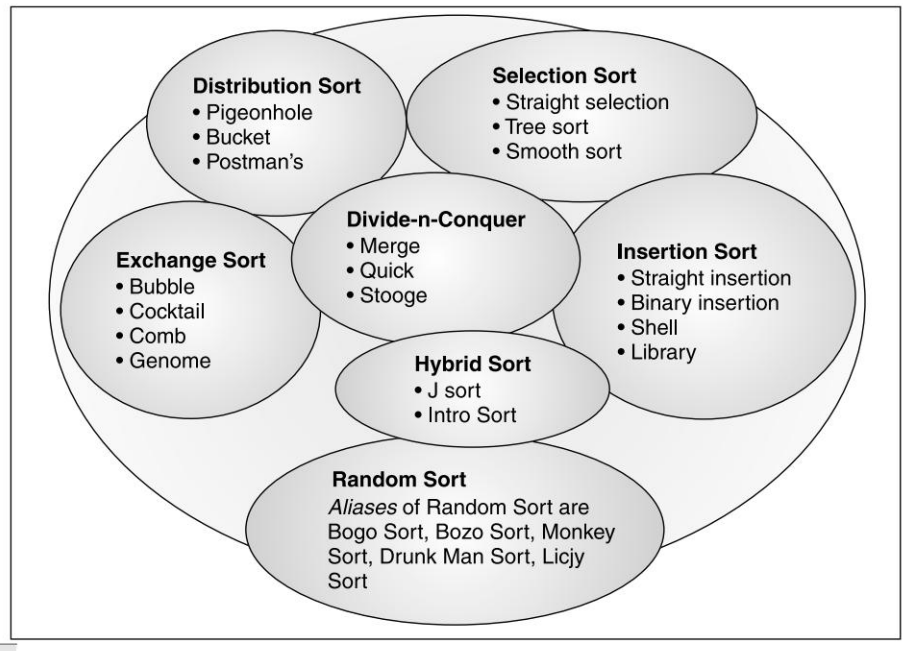


Fig. 10.1

is the slowest of all the comparison sort algorithms as far as the typical worst case performance is concerned.

In case of worst case, it takes one entire pass for an element to find its right place in the sequence. So if there are *N* number of elements in the un-ordered sequence, then it will take *n* passes to sort the element properly. Thus, the worst case performance of bubble sort is in the square of *N*.

Pass 1:

(Given Sequence)

29999	**30000**	29998	29997	29996	29995	29994	29993	29992	29991
29999	29998	**30000**	29997	29996	29995	29994	29993	29992	29991
29999	29998	29997	**30000**	29996	29995	29994	29993	29992	29991
29999	29998	29997	29996	**30000**	29995	29994	29993	29992	29991
29999	29998	29997	29996	29995	**30000**	29994	29993	29992	29991
29999	29998	29997	29996	29995	29994	**30000**	29993	29992	29991
29999	29998	29997	29996	29995	29994	29993	**30000**	29992	29991
29999	29998	29997	29996	29995	29994	29993	29992	**30000**	29991
29999	29998	29997	29996	29995	29994	29993	29992	29991	**30000**

Fig. 10.2

Pass 2:

The sequence generated at the end of Pass 1 is the input sequence to the next pass.

Look carefully below.

At the end of Pass 2 the second highest number will find its place.

Please note the sequence that is generated at the end of Pass 2.

29998 29997 29996 29995 29994 29993 29992 29991 29999 30000

Thus it will take 10 passes to sort this worst case sequence.

Table for the total number of comparisons $n^2 - 2*n + 1$, because in each pass there are $n-1$ comparisons made and $n-1$ passes needed to sort n numbers of elements.

Example 10.1 ***Write a program to demonstrate Bubble Sort.***

Solution

```
enum {ASC,DSC};//Tells the function how to sort ?
enum {NO,YES};//Used as flag in the program..

int* BubbleSort(int array[],int size,int how)
{
     int i=0;
     int j=0;
     int t;
     int k=0;
     //To keep a check whether the
     //array is already sorted or not.
     int swapped=0;
     for(i=0;i<size-1;i++)
     {
          swapped=0;
          for(j=0;j<size-1;j++)
          {
                    //ascending ?
                    if(how == ASC)
                    {
                         if(array[j]>array[j+1])
                         {
                              t = array[j];
                              array[j]=array[j+1];
                              array[j+1]=t;
                              swapped++;//Switched!
                         }
                    }
          }
          //descending ?
          if(how == DSC)
```

```
{
     if(array[j]<array[j+1])
     {
          t = array[j];
          array[j]=array[j+1];
          array[j+1]=t;
          swapped++;
                    }

                }
          }
     }
     return array;
}
```

To call this function,

```
int main()
{
     int *b;
     //a is some integer array of 10 integers
     //Sorting the array in Descending Order.
     b=BubbleSort(a,10,DSC);
     //Now do what you want with the sorted array b
     return 0;
}
```

10.4 WHAT IS THE TIME COMPLEXITY OF BUBBLE SORT?

Worst Case	$\Theta(n^2)$
Average Case	-
Best Case	$\Theta(n)$

Although, bubblesort shares the same worst case time complexity with some of the other algorithms like insertion sort, selection sort, etc. The average case performance of bubble sort is prohibitively low as compared with its $\Theta(n^2)$ family cousins. Thus, *Donald Knuth* commented that this sort has nothing to offer other than a catchy name.

Tabular Representation of Bubble Sort Performance

Number of Items	*Time Taken To Sort (in CTPS)*
10	0
100	0
1000	0.016
10000	0.594
15000	1.031
20000	1.766
25000	2.703
30000	3.828

Graphical Representation of Worst-Case Bubble Sort Performance

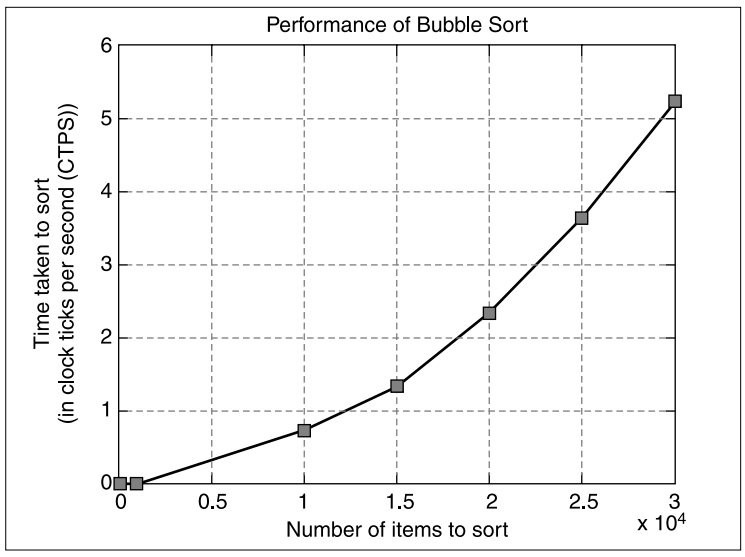


Fig. 10.3 *This graph shows the worst case performance of the switched bubble sort*

It is very clear from the above graph that the bubble sort performance is quadratic in nature. Later, in this chapter we will see how this poor performance algorithm can be modified slightly to give birth to another algorithm known as *Comb Sort* which performs quick sort.

These above plot is generated for 30,000 numbers starting from 30,000 to 1 and sorted ascending. This plot doesn't deal with the average case, instead they deal with the worst case.

The average case performance analysis of a sorting algorithm is mathematically involved and is out of scope of this book.

10.5 WHAT IS ODD-EVEN TRANSPOSITION SORT?

This is a variation of bubble sort where there are two alternating phases, namely, even phase and odd phase.

Even Phase The numbers in the even locations of the array, starting from 0 shift themselves with their immediate right neighbors.

Odd Phase The numbers in the odd locations of the array, starting from 0 shift themselves with their immediate right neighbors.

The phases continue until we reach the totally sorted array of the elements. The first step is an even phase sort.

Here is an example

The algorithm has $\Theta(n^2)$ time complexity.

Example 10.2 ***Write a program to demonstrate Bidirectional Bubble Sort.***

Solution This is a version of bubble sort. Here in each pass the direction of sorting is changed alternatively. This sort is also known as *cocktail shaker sort, shaker sort, ripple sort, shuttle sort and happy hour sort.* This sort improves the performance of the bubble sort.

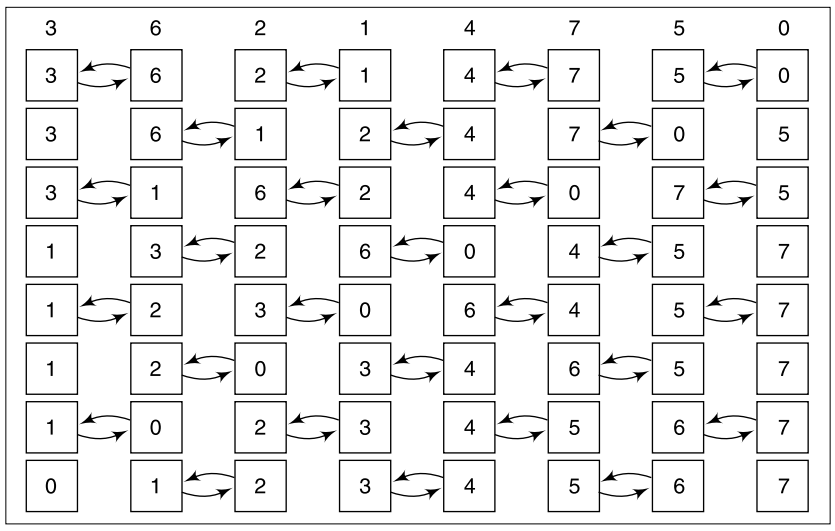


Fig. 10.4

```
int* BidirectionalBubbleSort(int MyArray[],int size,int how)
{
      int i=0;
      int j=0;
      int t=0;
      int k=0;
      int p=0;
      int swapped=0;
      int s=0;
      for(p=0;p<size-1;p++)
      for(i=0;i<size-1;i++)
      {

            if(i%2==0)
            {
                  for(j=0;j<size-1;j++)
                  {
                        if(how == ASC)
                        {
                              if(MyArray[j]>MyArray[j+1])
                              {
                                    t = MyArray[j];
                                    MyArray[j]=MyArray[j+1];
                                    MyArray[j+1]=t;
                                    swapped++;
                        }
            }
            if(how == DSC)
            {
                  if(MyArray[j]<MyArray[j+1])
                  {
```

```
                        t = MyArray[j];
                        MyArray[j]=MyArray[j+1];
                        MyArray[j+1]=t;
                        swapped++;
                  }

            }
      }
            if(i%2!=0)
            {
                  for(j=size-1;j>=0;j--)
                  {
                        if(how == ASC)
                        {
                              if(MyArray[j]>MyArray[j+1])
                              {
                                    t = MyArray[j];
                                    MyArray[j]=MyArray[j+1];
                                    MyArray[j+1]=t;
                                    swapped++;
                              }
                        }
                        if(how == DSC)
                        {
                              if(MyArray[j]<MyArray[j+1])
                              {
                                    t = MyArray[j];
                                    MyArray[j]=MyArray[j+1];
                                    MyArray[j+1]=t;
                                    swapped++;
                              }

                        }
                  }

            }
            }
      }
            return MyArray;

}
```

10.6 WHAT IS THE TIME COMPLEXITY OF BIDIRECTIONAL BUBBLE SORT?

Worst Case	$\Theta(n^2)$
Average Case	-
Best Case	$\Theta(n)$

Tabular Representation of Bidirectional Bubble Sort Performance

Number of Items	*Time Taken To Sort (in CTPS)*
10	0
100	0
1000	0
10000	0.265
15000	0.422
20000	0.734
25000	1.141
30000	1.656

Graphical Representation of Bidirectional Bubble Sort Performance

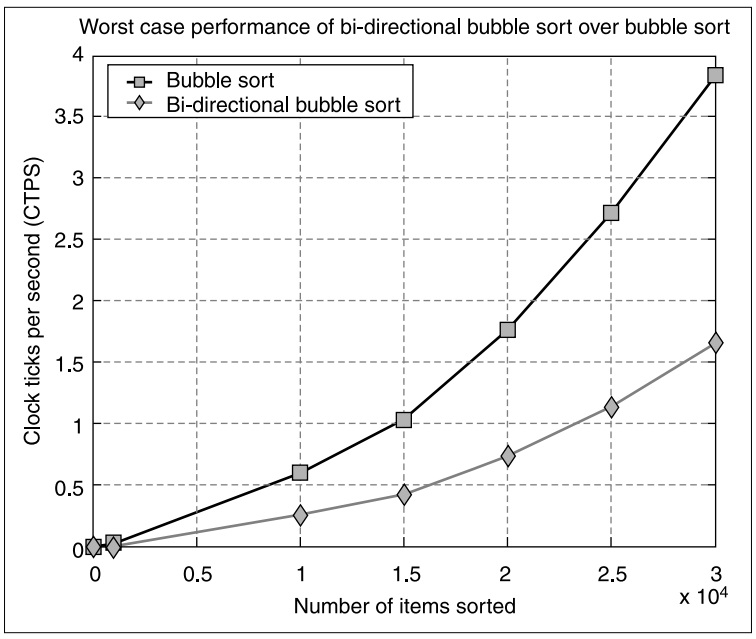


Fig. 10.5

The steeper curve shows the Bubble sort. So, you can clearly see that bidirectional bubble sort is way faster than simple bubble sort. Now, we will discuss about a sorting algorithm called *Comb Sort*, that is developed by a slight modification of bubble sort. But worst case performance of comb sort is way better than that of bubble sort.

Example 10.3 ***Write a program to demonstrate Comb Sort (also known as Doboseiwicz Sort).***

Solution Before we discuss about comb sort, we need to learn a well-known problem of bubble sort. The problem is known as *Turtles and Rabbits*.

Turtles and Rabbits

The large values at the beginning of a sequence takes not much time to reach their correct positions in

the list when the list is being sorted in an ascending order by bubble sort. These large values are known as *rabbits*.

On the other hand, the smaller values at the end of the sequence takes longer to crawl up to the front (assuming that we are doing an ascending sort). These values are known as *turtles*.

They take more time to crawl up to their correct position in an ascending sequence, because in bubble sort the comparison is always done between two consecutive (with a distance or gap 1) elements.

To combat this *Turtle-n-Rabbits* problem, *Stephen Lacey* and *Richard Box* came up with an algorithm, called *Comb sort*. In 1980, *Doboseiwicz* also invented the same algorithm. So, it is also known as *Doboseiwicz sort*. In comb sort, the comparison between the two elements is done but with a longer gap between them, instead of one unlike bubble sort.

This helps the rabbits to jump to the end of the list faster and it kills the turtles at the end easily. Comb sort does this by resetting the gap between the two values of the list. The update gap function below does that. Richard and Stephen proved that a shrink factor (by which factor the gap between two elements to compare reduces) of 1.3 gives satisfactory results.

With a shrink factor of 1.3, there are only three possible ways for the list of gaps to end: (9,6,4,3,2,1), (10,7,5,3,2,1) , (11,8,6,4,3,2,1). Only the last of these endings kills all turtles before the gap becomes 1. Therefore, significant speed improvements can be made if the gap is set to 11 whenever it becomes either 9 or 10. This variation of comb sort is called *Comb Sort 11*.

```
int update_gap(int gap)
{
    gap = (gap * 10) / 13;
    if(gap == 9 || gap == 10)
        gap = 11;
      if(gap < 1)
            return 1;
      return gap;

}

int* CombSort(int array[],int size)
{
      int temp=0;
      int i=0;
      int gap = size;
      int swapped;
      do
      {
            swapped = NO;
            gap = update_gap(gap);
            for(i=0;i< size - gap;i++)
            {
if(array[i] > array[i + gap])
{
      //Moving the Turtle to the top
      //Helping the Rabbits go bottom
      swapped = YES;
      temp = array[i];
      array[i] = array[i + gap];
```

```
                        array[i + gap] = temp;
                    }
                }

        }while (gap > 1 || swapped);//Tells you when to stop!

        return array;//returns the sorted array.
}
```

10.7 WHAT IS THE TIME COMPLEXITY OF COMB SORT?

Worst Case	**Θ(n log n)**
Average Case	**Θ(n log n)**
Best Case	**Θ(n log n)**

Tabular Representation of Comb Sort Performance

Number of Items	*Time Taken To Sort (in CTPS)*
10	0
100	0
1000	0
10000	0
15000	0
20000	0.016
25000	0
30000	0.015

Graphical Representation of Worst-Case Comb Sort Performance

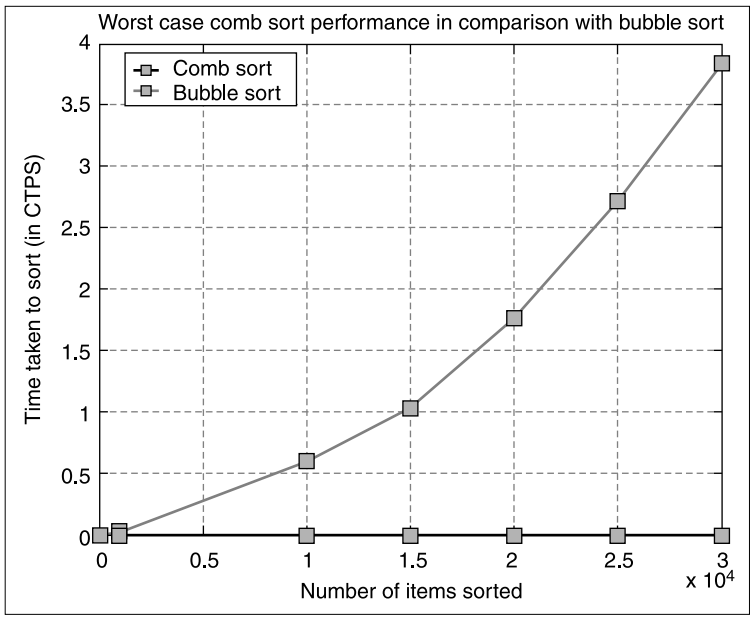


Fig. 10.6

Notice: How a small change can lead to high performance
Comb sort worst case performance is comparative with more complex and speedy algorithms like quick sort or flash sort.

10.8 INSERTION SORT ALGORITHMS

Example 10.4 ***Write a program to demonstrate Straight Insertion Sorting.***

Solution *Straight insertion sorting* is the simplest *in-situ (Read in-place)* sorting algorithm. An *in-situ* algorithm is a sorting algorithm where the element finds its place in the sorted sequence without any kind of comparison or exchange.

Starting from $I = 2$ up to the ith element, the elements are placed in place. Then the counter is incremented and the elements find their place.

Initial values
44 **55** 12 42 94 18 06 67
$I = 2$ **55** 44 **12** 42 94 18 06 67
$I = 3$ **12** 44 55 42 94 18 06 67
And so on
$I = 7$ 06 12 18 42 44 55 94 67
$I = 8$ 06 12 18 42 44 55 67 94

```
int* InsertionSort(int array[],int size,int how)
{
      int i;
      int j;
      int value;
      for(i=1;i<size;i++)
      {
            value = array[i];
            if(how==ASC)
            {
                  for(j=i-1;j>=0 && value<array[j];j--)
                        array[j+1]= array[j];
                  array[j+1]=value;
            }
            if(how==DSC)
            {
                  for(j=i-1;j>=0 && value>array[j];j--)
                        array[j+1]= array[j];
                  array[j+1]=value;
            }
      }
      return array;
}
```

10.9 WHAT IS THE TIME COMPLEXITY OF INSERTION SORT?

Worst Case	$\Theta(n^2)$
Average Case	-
Best Case	$\Theta(n)$

Tabular Representation of Insertion Sort Performance

Number of Items	*Time Taken To Sort (in CTPS)*
10	0
100	0
1000	0
10000	0.250
15000	0.156
20000	0.219
25000	0.296
30000	0.344

Graphical Representation of Worst Case Performance of Straight Insertion Sort

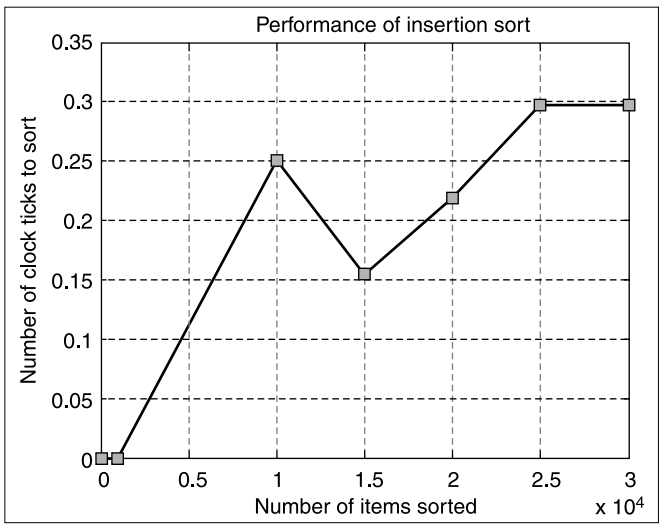


Fig. 10.7

This curve is clearly much more steeper than the previous graph. This proves that insertion sort is better than bubble sort as the number of elements increases. But for smaller amount of elements we can choose any of these two and it doesn't make any interpretable difference.

10.10 COMPARISON WITH THE IDEAL *O*(*N*^2) CURVE

Insertion Sort, normally means, the straight insertion sort, not any other variant of it.

Example 10.5 ***Write a program to demonstrate Binary Insertion Sorting.***

Solution The obvious development over the straight insertion sort is an algorithm that finds the *in-situ* location of the element. Binary search is a better way to search an item from a sorted list. If you notice carefully, you will find that in straight insertion sort, the destination sequence, in which the new item has to be inserted is already sorted. Therefore, we can use binary search to find its *in-situ* location.

So as the algorithm uses binary search to find the new *in-situ* location of the item being inserted thus the name binary insertion is justified.

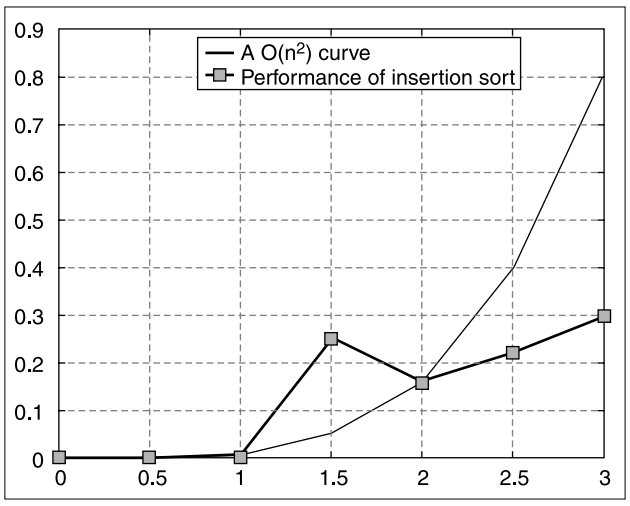


Fig. 10.8

```
int* BinaryInsertionSort(int array[],int size)
{
      int i=0;
      int j=0;
      int left=0;
      int right=0;
      int middle=0;
      int temp=0;
      for(i=1;i<size;i++)
      {
            temp = array[i];
            left = 0;
            right = i;
            while(left<right)
            {
                  middle = (left+right)/2;
                  if(temp>=array[middle])
                        left = middle+1;
                  else
                        right = middle;
            }
            for(j=i;j>left;--j)
            {
                  temp = array[j-1];
                  array[j-1]=array[j];
                  array[j]=temp;
            }
      }
      return array;
}
```

10.11 WHAT IS THE TIME COMPLEXITY OF BINARY INSERTION SORT?

Worst Case	$\Theta(n^2)$
Average Case	$\Theta(n^2)$
Best Case	$\Theta(n \log n)$

Tabular Representation of Binary Insertion Sort Performance

Number of Items	*Time Taken To Sort*
10	0
100	0
1000	0
10000	0.265
15000	0.203
20000	0.282
25000	0.375
30000	0.453

Graphical Representation of Binary Insertion Sort Performance

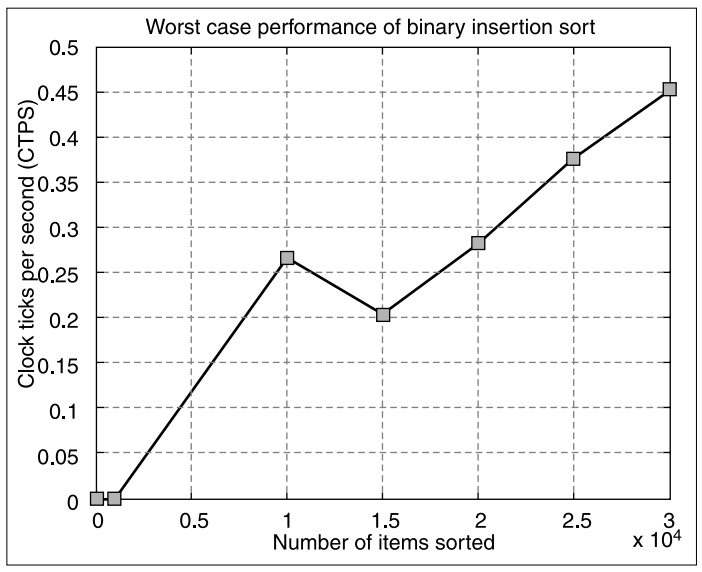


Fig. 10.9

If you notice carefully, then you will understand that binary insertion sort will also have a O(n^2) time complexity.

10.12 PROBLEMS WITH INSERTION SORT: SHIFTING

Although insertion sort looks very intuitive, it is not a very good algorithm because in case of a very

large sequence the insertion of an element in a sub-sequence involves a very expensive operation called *shifting*.

Selection sort, is a sorting algorithm that addresses this issue.

To combat the problem of shifting there is a version of insertion sort popularly known as Library Sort due to its similarity to shifting books on a library shelf.

10.13 EXPLAIN THE LIBRARY SORT ALGORITHM (*ALSO KNOWN AS* GAPPED INSERTION SORT)

Suppose a librarian were to store his books alphabetically on a long shelf, starting with the A's at the left end, and continuing to the right along the shelf with no spaces between the books until the end of the Z's. If the librarian acquired a new book that belongs to the B section, once he finds the correct space in the B section, he will have to move every book over, from the middle of the Bs all the way down to the Zs in order to make room for the new book. This is an insertion sort. However, if he were to leave a space after every letter, as long as there was still space after B, he would only have to move a few books to make room for the new one. This is the basic principle of the Library Sort.

The only disadvantage of this algorithm is the requirement of the extra space of order $O(n)$.

Example 10.6 ***Write a program to demonstrate Shell Sort.***

Solution This is a variation of insertion sort, discovered by *D.L.Shell* in 1959.

First of all elements which are 4 position apart are grouped and sorted separately. This process is called a *4 Sort*. In the following example of eight items, each group contains exactly 2 elements. After this first pass, the elements are re-grouped into the groups with elements two position apart. Then they are sorted again. This process is known as *2-sort*. Finally, in a third pass all elements are sorted in an ordinary sort or *1-sort*.

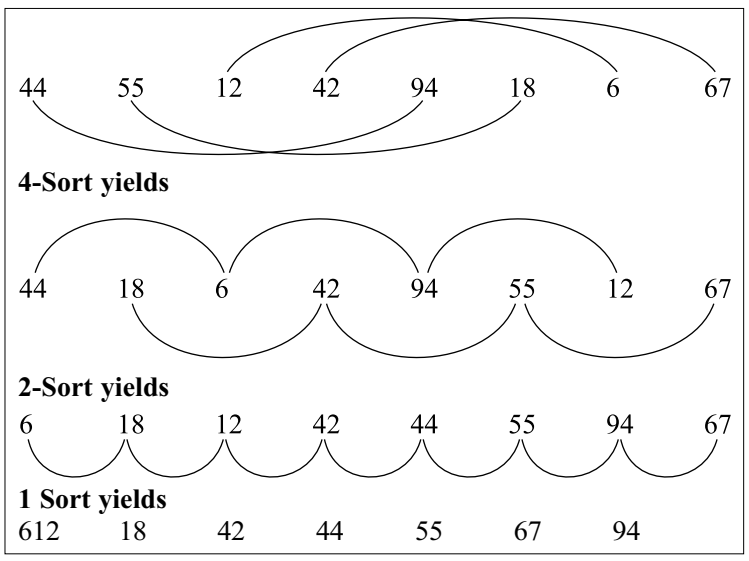


Fig. 10.10

```
int* ShellSort(int array[],int size)
{
      int temp;
      int span;
      int i;
      int swapped;
      span = size/2;
      do
      {
            do
            {
                  swapped=0;
                  for(i=0;i<size-span;i++)
                  {
                        if(array[i]>array[i+span])
                        {
                              temp=array[i];
                              array[i]=array[i+span];
                              array[i+span]=temp;
                              swapped=1;
                        }
                  }
            }while(swapped);
      }while(span=span/2);
      return array;
}
```

10.14 WHAT IS THE TIME COMPLEXITY OF SHELL SORT?

Worst Case	$\Theta(n^{1.5})$
Average Case	$\Theta(n^2)$
Best Case	$\Theta(n)$

Tabular Representation of Shell Sort Performance

Number of Items	*Time Taken To Sort*
10	0
100	0
1000	0
10000	0
15000	0
20000	0
25000	0.016
30000	0

You can see that the worst-case performance of shell sort is comparable to that of quick sort.

Graphical Representation of Worst-Case Shell Sort Performance

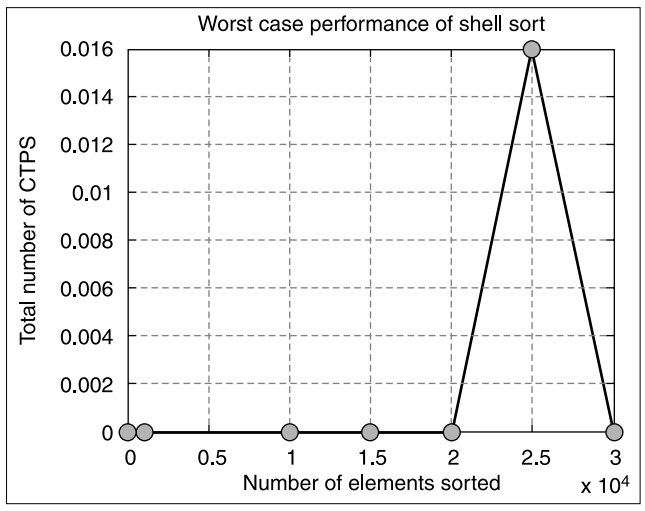


Fig. 10.11

10.15 SELECTION SORT ALGORITHMS

Example 10.7 ***Write a program to demonstrate Straight Selection Sorting.***

Solution This method is based on the following principle:

1. Select the minimum element in a sub-sequence.
2. Exchange it with the first element.

```
int* SelectionSort(int MyArray[],int length, int how)
{
      int i,j,min,minat;
      if(how == ASC)
      {
            for(i=0;i<length-1;i++)
            {
                  minat=i;
                  min=MyArray[i];

                    for(j=i+1;j<length;j++)
                    {
                          if(min>MyArray[j])
                          {
                                minat=j;
                                min=MyArray[j];
                      }
                }
                int temp=MyArray[i] ;
                MyArray[i]=MyArray[minat];
```

```
                MyArray[minat]=temp;
        }
}
else
{
            for(i=0;i<length-1;i++)
            {
                  minat=i;
                  min=MyArray[i];

                  for(j=i+1;j<length;j++)
                  {
                    if(min<MyArray[j])
                    {
                          minat=j;
                          min=MyArray[j];
                    }
                }
                int temp=MyArray[i] ;
                MyArray[i]=MyArray[minat];
                MyArray[minat]=temp;
          }
      }

      return MyArray;
}
```

10.16 WHAT IS THE TIME COMPLEXITY OF SELECTION SORT?

Worst Case	$\Theta(n^2)$
Average Case	$\Theta(n^2)$
Best Case	$\Theta(n^2)$

Tabular Representation of Selection Sort Performance

Number of Items	*Time Taken To Sort*
10	0
100	0
1000	0.015
10000	0.391
15000	0.593
20000	0.985
25000	1.484
30000	2.078

Graphical Representation of Selection Sort Performance

Fig. 10.12

It is very clear from the graph that the performance of selection sort is also quadratic but it is better than the bubble sort.

We may conclude that straight selection sort is to be preferred over straight insertion sort although in that

10.17 WHAT IS BINGO SORT?

Bingo Sort is a variation of selection sort that repeatedly looks through the remaining items to find the greatest item moving all items with that value to their final location. This is more efficient if there are many duplicate values.

To see why it is more efficient, consider one value. Selection sort does one pass through remaining items for each item moved. Bingo sort does two passes for each value (not item): one pass to find the next biggest value, and one pass to move every item with that value to its final location. Thus if on average there are more than two items with each value, bingo sort may be faster.

10.18 HYBRID SORT ALGORITHMS

Any sorting algorithm that uses two or more algorithms to sort the elements of a sequence is called a hybrid sort. One such example is J-Sort.

10.19 WHAT IS J-SORT?

J Sort is a sorting algorithm that uses strand sort for fewer than 40 elements and shuffle sort for more than 40 elements. So J Sort is basically strand sort for an array of 40 elements or less and it is a hybrid of two sorting algorithms *strand sort* and *shuffle sort* for more than 40 elements.

Try Yourself: *Implement J Sort for arbitrary list of numbers. Use linked lists.*

10.20 DIVIDE-N-CONQUER SORTING ALGORITHMS

In this section those sorting techniques will be discussed that use a divide and conquer strategy to sort the array by splitting the array in many sub lists.

10.21 HOW TO WRITE A FUNCTION TO DEMONSTRATE QUICK SORT

The name of this sorting algorithm is really justified. Quick sort algorithm was developed by *C.A.R. Hoare* that, on average makes $O(n \log n)$ Comparisons to sort n elements. However in the worst case it makes $\mathbf{\Theta(n^2)}$ comparisons. Typically, Quick sort is way faster than its $O(n \log n)$ family cousins. But this is not a stable algorithm.

Quick sort follows the *Divide-and-Conquer* strategy like merge sort. Thus, Quick sort is *v*

1. It picks up an element from the list. The element is known as *pivot*.
2. Re order the elements so that the elements which are less than the pivot comes before it and the rest elements which are greater than the pivot comes next to it. After this partitioning, pivot will find its final place in the list.
3. Recursively perform tasks 1 and 2 for the sub lists of smaller and larger numbers.

```
int* QuickSort(int array[],int first,int last)
{
      int temp;
      int low,high,pivot;
      low = first;
      high = last;
      pivot = array[(first+last)/2];
      do
      {
            while(array[low]<pivot)
                  low++;
            while(array[high]>pivot)
                  high--;
            if(low<=high)
            {
                  temp = array[low];
                  array[low++] = array[high];
                  array[high--]=temp;
            }
      }while(low<=high);
      if(first<high)
            QuickSort(array,first,high);
      if(low<last)
            QuickSort(array,low,last);
      return array;
}
```

10.22 WHAT IS THE TIME COMPLEXITY OF QUICK SORT?

Worst Case	$\mathbf{\Theta(n^2)}$
Average Case	$\mathbf{\Theta(n \log n)}$
Best Case	$\mathbf{\Theta(n \log n)}$

Tabular Representation of Quick Sort Performance

Number of Items	*Time Taken To Sort*
10	0
100	0
1000	0
10000	0
15000	0
20000	0
25000	0
30000	0.016

10.23 HOW TO SELECT THE PIVOT IN QUICK SORT

Selecting the pivot is a very important activity as far as the performance of Quick sort is concerned. In some implementations the first element of the list is chosen as the pivot. In some implementations the median of the first three or three randomly chosen number is selected as pivot. This implementation is called *Sample Sort*.

According to research, if the pivot is selected to be the middle element of the list, then the Quick sort performance is optimal.

If the array is almost sorted, quick sort takes longer than expected and often goes out to perform by lower speed algorithms. To avoid this, sometimes the entries are scrambled before those are fed to quick sort.

Example 10.8 ***Write a program to demonstrate Merge Sort.***

Solution

```
void MergePass(int array[],int top,int size,int bottom)
{
      int temp[100];
      int f = top;
      int s = size + 1;
      int t = top;
      int upper;
      while(f<=size && s<=bottom)
      {
            if(array[f]<=array[s])
            {
                  temp[t]=array[f];
                  f++;
            }
            else
            {
                  temp[t]=array[s];
                  s++;
            }
            t++;
      }
      if(f<=size)
      {
```

```
{
      for(;f<=size;f++)
      {
            temp[t]=array[f];
            t++;
      }
}
else
      {
            for(;s<=bottom;s++)
            {
                  temp[t]=array[s];
                  t++;
            }
      }
      for(upper=top;upper<=bottom;upper++)
      {
            array[upper]=temp[upper];
      }
}

int* MergeSort(int array[],int m,int n)
{
      int mid;
      if(m!=n)
      {
            mid = (m+n)/2;
            MergeSort(array,m,mid);
            MergeSort(array,mid+1,n);
            MergePass(array,m,mid,n);

      }
      return array;
}
```

10.24 WHAT IS THE TIME COMPLEXITY OF MERGE SORT?

Worst Case	**O(n log n)**
Average Case	-
Best Case	**O(n log n)**

Example 10.9 *Write a program to demonstrate Stooge Sort.*

Solution

```
int* StoogeSort(int array[],int start,int end)
{
      int temp=0;
      int t;

      if(end-start==1)
      {
```

```
        if(array[end]<array[start])
        {
                temp = array[end];
                array[end]=array[start];
                array[start]=temp;
        }
}
if(end-start>1)
        {
                t = (end-start+1)/3;
                StoogeSort(array,start,end-t);
                StoogeSort(array,start+t,end);
                StoogeSort(array,start,end-t);
        }

        return array;
}
```

10.25 WHAT IS THE TIME COMPLEXITY OF STOOGE SORT?

Worst Case	$\Theta(n^{2.7})$

10.26 DISTRIBUTION SORTING ALGORITHMS

These algorithms deals with the numbers by first sorting them into different regions and then sort the individual regions and then concatenate. Which number will land up to which place is strictly part. That way, merge sort is a special type of distribution sort.

10.27 BUCKET SORT

Bucket sort is a specialization of *pigeonhole sort*. It works by partitioning the numbers in the array by a finite number of buckets. Each bucket is then sorted individually either by using a different algorithm or by using bucket sort. Bucket sort runs in linear time $O(n)$.

This figure gives an idea how bucket sort works.

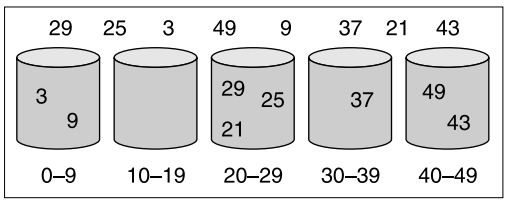


Fig. 10.13

Bucket sort works as follows:

Set up an array of initially empty buckets the size of the range.

Go over the original array, putting each object in its bucket.

Sort each non-empty bucket.

Put elements from non-empty buckets back into the original array.

To find out which number will go in which bucket a function can be written like msbits(x, k) which will return the k most significant digit of x.

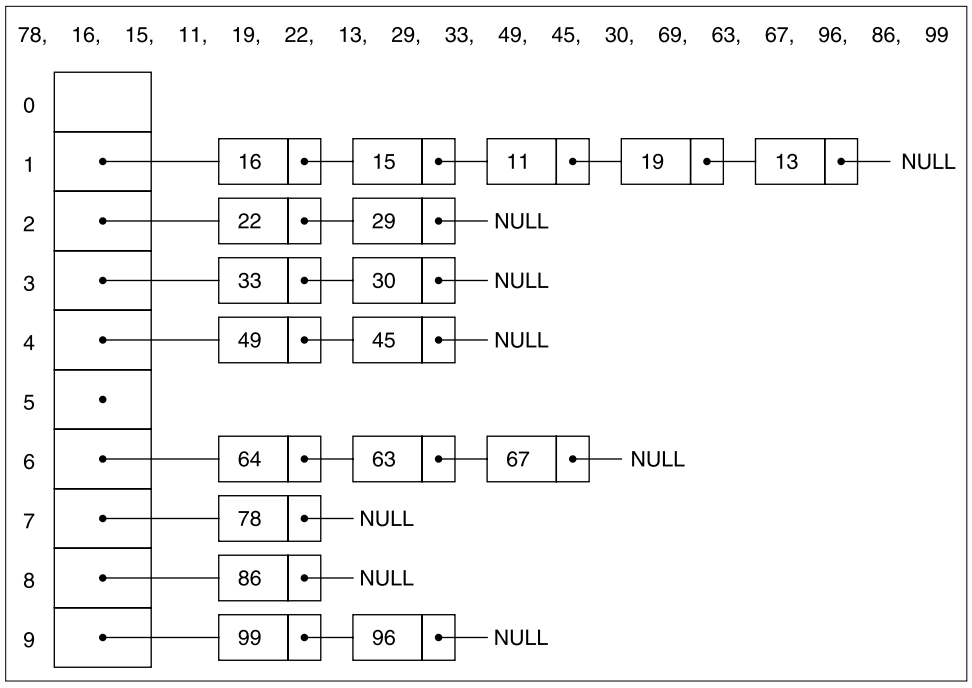


Fig. 10.14

Try Yourself: *Use jagged linked list to store the numbers as shown above. Then sort each linked list separately and then merge them into one linked list sequentially to obtain the sorted numbers.*

10.28 PERFORMANCE COMPARISONS OF THE SORTING ALGORITHMS WITH *O*(*n*^2) TIME COMPLEXITY

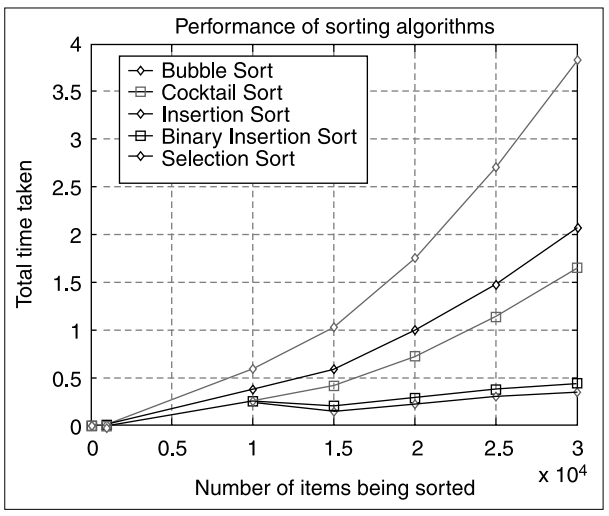


Fig. 10.15

From the graph, which is produced with the values from the tables above, we can easily conclude that binary insertion sort is clearly the best algorithm of all those that are compared here and bubble sort is definitely the worst. From the graph, fortunately we can see a clear demarcation of performance of these sorting algorithms.

So, as far as the worst case performance is concerned these sorting algorithms have the following relationship.

Bubble Sort < Selection Sort < Cocktail Sort < Insertion Sort < Binary Insertion Sort

But we have seen earlier that how a slight modification in the bubble sort could generate algorithms like comb sort that is equal to quick sort as far as the worst case is considered.

Shell sort, comb sort, quick sort has almost the same performance curve. In the beginning of this chapter the sort algorithms are classified according to the technique of operations. They can also be classified according to their performance or time complexity. Broadly there are two groups. Algorithms with worst case time complexity $\Theta(n^2)$ and algorithms with time complexity $\Theta(n \log n)$. Clearly, the algorithms with $\Theta(n \log n)$ is better than their $\Theta(n^2)$ cousins.

10.29 PERFORMANCE COMPARISONS OF THE SORTING ALGORITHMS WITH $O(n \log n)$ TIME COMPLEXITY

Clearly, quick sort is the best of the three. For a certain range heap and merge behave the same way. But after that threshold, merge sort out-performs heap sort due to the heavy recursion overhead in the later.

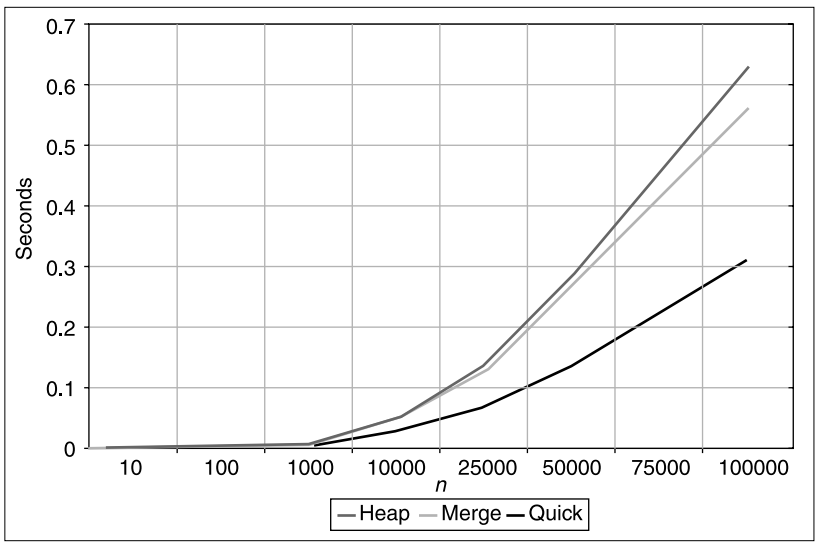


Fig. 10.16

10.30 BOGO SORT AND FRIENDS

Bogo sort is probably the most inefficient sorting technique one rational human being could ever imagine. Bogo sort uses two steps to sort the elements of the array.

1. It throws the number randomly.
2. Checks whether the numbers are in order or not.
3. If they are not in order then it generates another randomization of them.
4. The above steps continue till the array is sorted.

Due to its tremendous inefficiency it is jokingly called *stupid sort, bozo sort, blort sort, monkey sort, random sort and drunk man sort*.

Bozo sort (a Variation of Stupid Sort) **Bozo sort** is another sorting algorithm based on the random numbers. If the list is not in order, it picks two items at random and swaps them, then checks to see if the list is sorted. It also faces the same pseudo-random problems as bogosort—it may never terminate.

10.31 TREE SORT

Binary Tree can be used to sort numbers very easily. Suppose we have to sort numbers 170, 45, 75, 90, 2, 24, 802 , 66 then we can create a binary tree with these numbers as shown in Fig. 10.17.

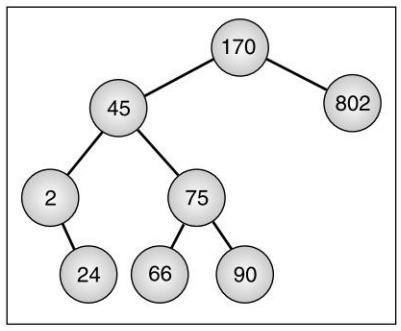


Fig. 10.17

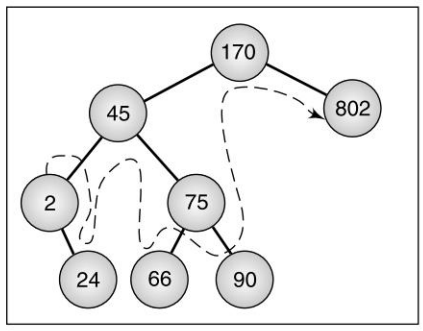


Fig. 10.18

Now if we traverse the tree in-order as shown by the broken line in Fig. 10.18
we will get the numbers sorted in increasing order as
2, 24, 45, 66, 75, 90, 170, 802.
On the other hand, if we traverse the tree using post order as shown by the broken line in the Fig. 10.19, we will get the numbers in decreasing order as
802, 170, 90, 75, 66, 45, 24, 2

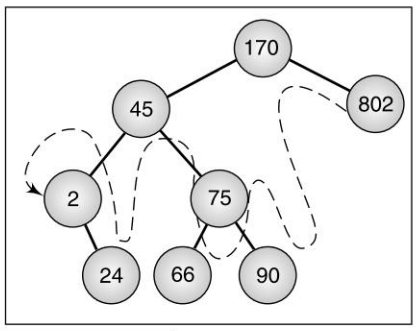


Fig. 10.19

10.32 LEXICOGRAPHIC SORT

Before we understand what is *Lexicographic sort*, we need to understand what is *Lexicographic order*.

Lexicographic Order A *d*-tuple is a sequence of *d* keys (*k*1,*k*2,*k*3, … , *kd*) where key *ki* is said to be the i^{th} dimension of the tuple.

Lexicographic Sort Let *Ci* be the comparator that compares two tuples by their i^{th} dimension.

Let StableSort(S,C) be a stable sorting algorithm that uses comparator C.

Lexicographic sort sorts *d* tuples in lexicographic order by executing StableSort algorithm *d* times once per dimension.

Lexicographic sort runs in $O(dT(n))$ time where $T(n)$ is the stable sort time complexity.

An example

Input tuple sequence (7,4,6) (5,1,5) (2,4,6) (2,1,4) (3,2,4)

Pass 1: (2,1,4) (3,2,4) (5,1,5) (7,4,6) (2,4,6)

Pass 2: (2,1,4) (5,1,5) (3,2,4) (7,4,6) (2,4,6)

Pass 3: (2,1,4) (2,5,6) (3,2,4) (5,1,5) (7,2,4)

10.33 RADIX SORT

Radix Sort is a special type of Lexicographic that uses bucket sort as the StableSort algorithm. Radix sort is applicable to tuples where the keys in each dimension *i* are integers in the range [0 to *N*–1]

Radix Sort can be of different types depending on the digit on which the sorting is being performed. When the sorting is performed on the *Most Significant Digit* then the sorting is known as *MSD Radix Sort*. On the other hand, when the sorting is done on the least significant bit then the sorting is known as *Least Significant Bit Radix Sort*.

Let's say we have a list of numbers like

120, 340,43,2,304,502,230,1

If we extend the above numbers for 3 digits, then the numbers are

120,340,043,002,304,502,230,001

So, after sorting the Most Significant Digit we get

502,340,304,230,120,43,2,1

Please note that only after first pass the numbers are sorted in descending order. This has happened due to the special nature of this sequence. Radix sort uses bucket sort as the stable sorting algorithm.

10.34 ADDRESS CALCULATION SORT USING HASHING

If we can calculate the location of a value from a set of variables to sort from, then there will be no involvement of comparison of one value with the other. Thus the program will be faster because there will be no comparisons involved. We can use hash functions to calculate the locations of a value. In applying a hashing function to the sorting process, a particular kind of hashing function will be required. Let us assume that we have a hashing function *H* with the property

$x1 < x2$, implies that $H(x1) <= H(x2)$

A function which exhibits this property is called a *non-decreasing* or *order preserving*, hashing function. When such a function is used to hash a particular key into a particular number to which some previous keys have already been hashed (That is a collision occurs) then the new key is placed in the set of colliding records so as to preserve the order of the keys. The result of hashing and inserting the sample keys using a non decreasing hashing function in which all keys in the ranges 1–20, 21–40, 41–60, 61–80 and 81–100 are each hashed into a different set as sorting proceeds. This is shown in the following figure

42, 23, 74, 11, 65, 57, 94, 36, 99, 87, 70, 81, 61

A general algorithm for this sorting process follows:

1. Initialized hash entries to NULL.
2. Repeat through step 4 while there are still input records.
3. Input a hash record.
4. Insert table into appropriate linked list.
5. Concatenate the non-empty linked lists into one.

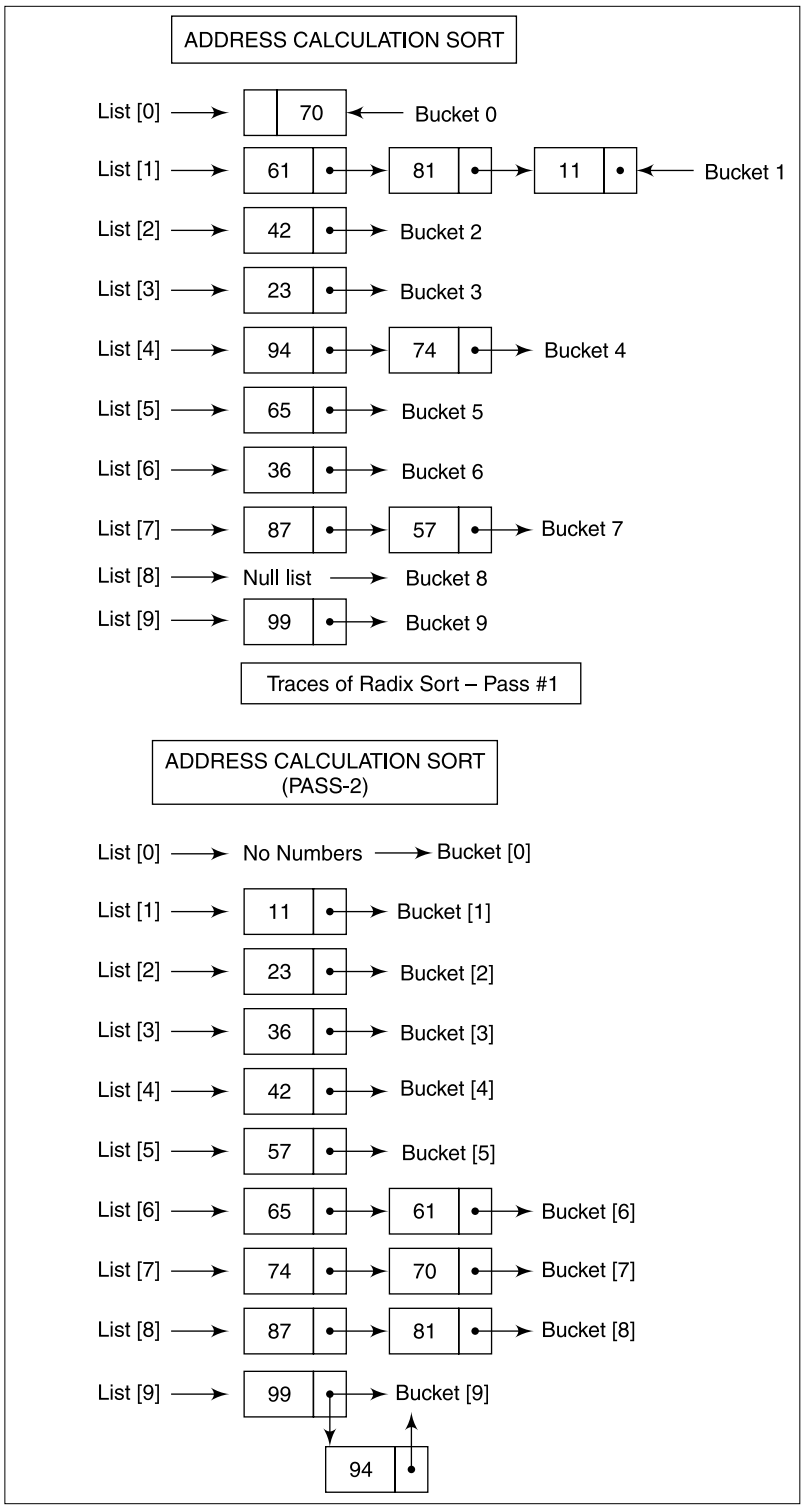

ADDRESS CALCULATION SORT
List [0]
70
Bucket 0
List [1]
61
81
11
Bucket 1
List [2]
42
Bucket 2
List [3]
23
Bucket 3
List [4]
94
74
Bucket 4
List [5]
65
Bucket 5
List [6]
36
Bucket 6
List [7]
87
57
Bucket 7
List [8]
Null list
Bucket 8
List [9]
99
Bucket 9
Traces of Radix Sort – Pass #1
ADDRESS CALCULATION SORT
(PASS-2)
List [0]
No Numbers
Bucket [0]
List [1]
11
Bucket [1]
List [2]
23
Bucket [2]
List [3]
36
Bucket [3]
List [4]
42
Bucket [4]
List [5]
57
Bucket [5]
List [6]
65
61
Bucket [6]
List [7]
74
70
Bucket [7]
List [8]
87
81
Bucket [8]
List [9]
99
Bucket [9]
94

Fig. 10.20

10.35 APPLICATION OF SORTING

Sorting is one of the most fundamental processes that are involved in all the computer programs in one way or the other. Here are some generic areas where sorting is the key to the solution.

- Clustering
- Business clustering
- Finding shortest path
- Finding the most wanted DVD in the city
- Finding the greatest scorer in the online pool competition
- Finding the largest shape when their dimensions are given
- Finding the largest file in the hard disk
- Detecting cancer cells by statistical analysis

10.36 WHAT IS CLUSTERING?

Clustering is an operation to group a set of elements depending on the value of a particular parameter. For example, we want to find out what is the number of college students who studies data structure. So, the subject being studied is the parameter and the students who studies that form the cluster. So, a loose definition of clustering could be the process of organizing objects into groups whose members are similar in some way.

Sorting is the key to clustering. Suppose we have a set of student records and we have to cluster them. We can write a program that would interactively find the clusters of all the different subjects, for example, a cluster of data structure students, a cluster of numerical methods, students, etc.

10.37 BUSINESS CLUSTERING

There are always some stakeholders for a business. If the business performs well, then their stakeholders will also gain from it. For example, the better be the tourism business the better the transportation business because the transportation is a supporting business to tourism. Government of a nation may want to find what is the business that can grow together.

Business clustering is a technique that uses sorting to find out several numerical values (For example, share prices) that tell about the prosperity of a company to find out what other business is also growing/ has the potential to grow along with it.

10.38 FINDING THE SHORTEST PATH

This is one of the most used examples and used by GPS enabled car driving assistants. For example, if you want to go to a place in a city from a hotel in the same city. There could be *n* number of ways to reach that point. You calculate the Euclidian distance between them and store them in an array and find the shortest path length.

10.39 FINDING THE MOST WANTED DVD IN THE CITY

A DVD supplying company may want to find out what is most wanted by their customers in a city. They could easily post a survey in the local online newspaper. Readers could vote online. Depending on the polling results, the most wanted DVD in the city can be found. These type of problems are very obvious applications of sorting and Merge Sort is one of the most suited algorithms for these types of activities, because the user base is large so we will probably get more response than could be loaded in a computer memory. External sorting techniques could be used.

10.40 FINDING THE GREATEST ONLINE SCORER IN THE ONLINE POOL COMPETITION

Suppose, few people are playing online pool from different geographical locations and their scores are being constantly monitored and stored in a database. At the end of the game if a user clicks “Find Rank”

button, all the player data will be sorted and the user's position will be displayed amongst the playing members. This is done by sorting the players by their score.

10.41 FINDING THE LARGEST SHAPE WHEN THEIR DIMENSIONS ARE GIVEN

A surveyor finds it difficult to calculate the area and perimeter of all the shapes that he measures on the fly. So, he writes back the dimensions measured on a flat file. The sorting application program reads that and finds out what shape has the maximum area/volume.

REVISION OF CONCEPTS

Some Key Facts about Sorting and Terminology

- ***Sorting***:
  - In computer science and related fields (probably everything under the sun) sorting normally means to order a set of variables either in ascending or in descending order.
  - This means ordering a set of elements.
  - Supposedly, 25% of all CPU cycles are spent sorting.
- Sorting is fundamental to most other algorithmic problems, for example, binary search.
- *Many* different approaches lead to useful sorting algorithms, and these ideas can be used to solve many other problems.
- **CTPS**: Clock Ticks Per Second
  - This is a measure of how many CPU clock ticks occur within a second.
  - For different system the CTPS will be different.
- **Stable:** A sort algorithm is said to be stable, if it reserves the relative order of records with equal keys.
- ***Spatial Complexity:***
  - This discusses about
    - Does a particular algorithm use prohibitive amount of memory?
    - Is the target hardware ok as far as memory is concerned?
  - Massive Spatial Complexity is not a big issue in PCs but when we are coding for embedded software.
- ***Time Complexity:***
  - Time Complexity of an algorithm is the measure of the time it takes to complete a set of tasks.
  - There are few ways to quantify this measure. One of them is to use functional notation like Big O and Omega.
- ***Big O***: This is a notation to denote the time complexity.
  - In sorting the operations being performed depend on the number of entries in the array or sequence.
  - So if it is found that the sorting algorithms take a time that is proportional to the number of elements then the time complexity of the algorithm is given by **Θ(n)** and in case we see that the time complexity is directly proportional to the square of the number of elements being sorted then the time complexity is **Θ(n^2).**
- ***Best Time Complexity Possible***: **Θ(n)** because one needs to travel all the elements of the list at least once.

- ***Worst Time Complexity Possible*** $\Theta(n^2)$
- ***External Sorting:***
  - Sometimes the data to sort becomes so large that it becomes difficult to load the entire data in the computer memory to sort them. In such cases external storage devices like tape drive, etc. are used.
  - Merge sort is typically used for external sorting.
- ***Parallel Sorting***: In a system with multiple processors, the operations could be distributed easily between multiple processors. Suppose there are *n* number of operations to be performed in order to sort a sequence and there are *p* number of processors. So each processor will be loaded with *n*/*p* number of operations and ideally it will take *n* time less time.

REVIEW QUESTIONS

1. What is the time complexity of strand sort?
2. What is the time complexity of J-Sort?
3. What is the time complexity of shuffle sort?
4. Does the time complexity of Quick Sort depends on the pivot?
5. Is Quick sort a good algorithm to use if the array is almost sorted?
6. Which one is better 1000N or *O*(*n*^2)?

PROGRAMMING PROBLEMS

1. How does quick sort perform if the data is almost sorted?
2. Write a program to demonstrate MSD Radix Sort. The program should be able to sort numbers of *n* digits, where *n* may be any arbitrarily long or short.
3. Write a program to demonstrate LSD Radix Sort. The program should be able to sort numbers of *n* digits, where *n* may be any arbitrarily long or short.
4. Write a program to demonstrate Counting Sort Algorithm.
5. Discuss the algorithm for 2-way Merge Sort.
6. Create a program to do a performance analysis of all these sorting algorithms and find out the best sorting algorithm for a given sequence of integers.
7. Prove the time complexity of Insertion Sort.
8. Write a program to find the top 10 rented CDs in a CD library.
9. Write a program to find the business cluster. Assume that a share value chart is given where the company names and their share prices are written. All you have to do is to identify the companies that have some relation in share prices.
10. Write a program to demonstrate J Sort.
11. Write a program to demonstrate Bucket Sort.
12. Write a program to demonstrate Polyphase Merge Sort.
13. Compare their performance on a graph.
14. Compare all the sorting algorithm discussed on a single graph.

11

Hashing
Accident or Choice?

INTRODUCTION

In computer science searching for a particular string from a pool of strings is an activity that is very common. So far whatever searching algorithm we studied, the fastest ever was $O(m)$ where m is the length of the string being sought. This could be intuitively the optimal search time complexity, because one must have to look through all the characters of a string to declare a match from a pool of strings. But in some programming language, we can use associative arrays to store key value pair (which is also shown in the Map chapter of this book) so that search can be done in $O(1)$ time. In programming languages that do not support in-built associative arrays, like C, Hashing let us create a map where each string/value is mapped to a reproducible unique key by a function known as *Hashing function*. A Hash function takes data of arbitary size using a perfect Hash function H( ) we can map x to y as $y = H(x)$. such that for no values of $x1$ and $x2$, $H(x1) = H(x2)$.

Some features that are desirable are as follows:

$y = H(x)$ should be one directional so that the reverse cannot be reproduced.

$H(x1)$ and $H(x2)$ will not be same ever.

11.1 CONCEPT OF COLLISION AND ITS RESOLUTION

Almost all Hash functions in use are imperfect that means we might end up a situation where $H(x1) = H(x2)$ when $x1$ and $x2$ are different.

When such an occasion occurs, this is termed as *collision*. There are several ways a collision can be prevented. One way is called '*Open Addressing*' and the other is called '*Separate Chaining*'.

In terms of collision the philosophy is 'first come, first serve'. So the item that is hashed first will get the location in the hash table first. We have to do rehash with the second and subsequent values that collide with the already hashed element. The following example illustrates everything.

It might also happen that the generated hash code of the new string is out of the hash table maximum range. Say 149 in this case. Then we need to start looking at the front of the Hash Table. This scheme of

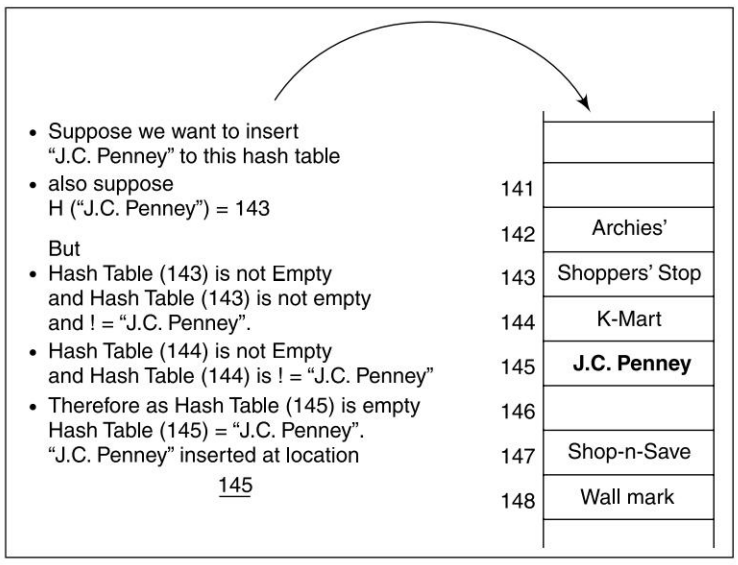


Fig. 11.1

collision resolution is known as *Linear Probing*. Due to the hash function that calculates the location of the element in the array derives it the result in linear time.

This collision resolution technique is known as *Linear Probing*. The name is due to the *linear time* it takes to travel across the hash table before an initially collided string finds its home.

There is another technique to combat collision which is known as *Separate Chaining*.
In this technique all collided items are added at the end of a linked list whose header is plugged at the hashed location in the hash table. The following illustration makes it clear.

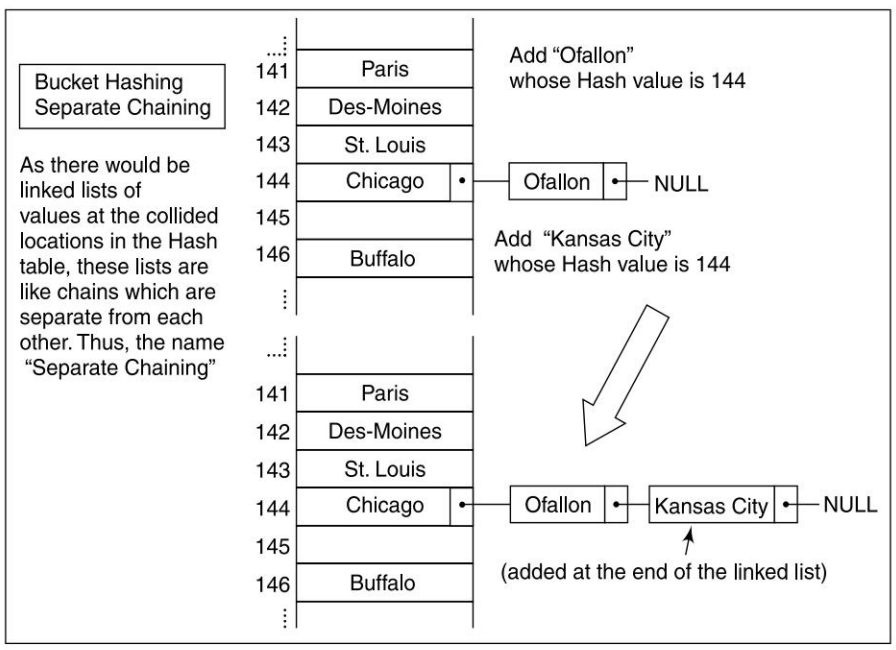


Fig. 11.2

The advantage of separate chaining is that we can store multiple values but the disadvantage is we need additional storage and the linked list maintenance overhead.

11.2 SOME KEY FACTS AND JARGONS ABOUT HASHING

Hashing It is the trick of organizing data elements by encoding them and it finds applications from design of associative containers like dictionary to cryptography.

Hash Function A hash function is a reproducible way to convert some data into some sort of digital fingerprint.

Hash Table A hash table is nothing but a map (For understanding map, read the chapter on map after ADT) where the locations of a newly added entry is determined by the hash functions.

Load Factor This is the measure of how much loaded a hash table is at a certain time. The measure is given by the ratio of occupied cells and number of total cells in the hash tables. Typically when the load factor falls to ½ then the size of the hash table is doubled.

Hash List As the name suggests it is a list of hash codes.

Hash Tree This is a tree of hash codes of different data blocks of a file. Mainly used in p2p network to verify the authenticity of a media. Hash trees are mostly designed binary.

Hash Chain This is hash of hash of hash and so on of a given key. This technique was first discovered by *Leslie Lamport*. This hashing technique is normally used for OTP generations.

OTP *(One Time Password)* This is a scheme where passwords are generated using hash chains.

Collision When two elements are hashed to the same location on a hash table.

Collision Free Hashing This is a hashing function that doesn't map any two entries to the same hash code.

One Way Hash Function This is a cryptographic hash function that is a collision free hash for which calculation of inverse hash function is computationally infeasible. For example,

If $y = H(x)$ and H is a one way hash function, then it is not possible to find a function G such that $G(y) = x$. or in other words $G = H^{(-1)}$

Coalesced Hashing This is a scheme to handle collisions using the best approach of open addressing and separate chaining.

Universal Hashing A hashing method in which hashing functions are generated randomly at runtime so that no particular set of keys is likely to produce a bad distribution of elements in the hash table. Because the hash functions are generated randomly, even hashing the same set of keys during different executions may result in different measures of performance.

Bloom Filter This is a hash table of k different entries where each entry is mapped to the table using a different hash function. So in total there will be k different hash functions to map k different entries.

How to Demonstrate Linear Probing Technique on Integer

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

```
//Definition of our Hash Function
#define HASH(x) x%10

//Maximum number of elements that the Hash Table can hold.
#define MAX 10

int Values[MAX];
int LinearProbedPosition(int number)
{
      int HashedIndex = HASH(number);
      int n = 0;
      if(Values[HashedIndex]==0)
            return HashedIndex;
      else
      {
            do
            {
                  HashedIndex++;
                  if(Values[HashedIndex]==0)
                        return HashedIndex;
                  else
                        continue;
            }while(HashedIndex!=MAX-1);

            if(HashedIndex==MAX)
            {
                  HashedIndex=0;
                  do
                  {
                        n++;
                        HashedIndex++;
                  }while(Values[HashedIndex]!=0 || n!=MAX-1);
                  if(n!=MAX-1)
                        return HashedIndex;
                  else
                        return -1;
            }

      }
}

int main()
{
      int n = 0;
      int number;
      while(n!=MAX)
      {
            number = 0;
            printf("Enter number :");
            scanf("%d",&number);
            Values[LinearProbedPosition(number)]=number;
            n++;
```

```
            Values[LinearProbedPosition(number)]=number;
            n++;
      }
      for(n=0;n<MAX-1;n++)
            printf("%d\n",Values[n]);
      getch();
      return 0;
}
```

The main disadvantage of this linear probing method is that we can't store more than the array can hold. So, once the array is full then no elements can be added to it. But the advantage of this system is that the method is very simple and doesn't require extra space for storing the numbers mapped to the same location; unlike separate chaining method that will be described next.

How to Demonstrate Quadratic Probing

Quadratic probing is a method to rehash the collided number. In quadratic probing the i^{th} hash value of the number x is given by

$$h(x,i) = h(x) + c1 * i + c2 * i^2 \ (\mathbf{mod}\ m)$$

For example

$$h(x,i) = h(x) + i + i^2 \ (\mathbf{mod}\ m)$$

Then the probe sequence will be h(x), h(x)+1, h(x)+3, h(x)+6 etc for m = 2^n
It has been found that for c1 = c2 = ½ all the values for hashing are distinct in the range [0 to m–1]

11.3 HOW TO DEMONSTRATE THE SEPARATE CHAINING METHOD FOR HASHING ELEMENTS IN A HASH TABLE

To remove the disadvantage of the linear probing method we create a linked list of numbers to be attached for each node in the hash table. Once the cell in the hash table is filled then the numbers will be pushed in the linked list associated with the hash table cell.

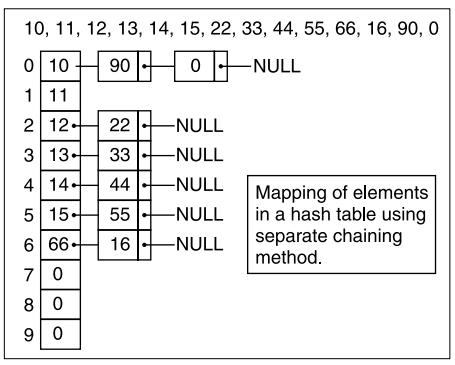


Fig. 11.3

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

#define HASH(x) x%10 //A Very simple Hash
 function which uses module
#define MAX 10

typedef struct node
{
      int data;
      struct node *next;
}node;

typedef struct number
{
      node *list;
      int data;
```

```
      int count;
}number;

number Numbers[MAX];

node* push_front(node *last,int data)
{
      if(last==NULL)
      {
            last = (node *)malloc(sizeof(node));
            last->data = data;
            last->next = NULL;
            return last;
      }
      else
      {
            node *p = (node *)malloc(sizeof(node));
            p->data = data;
            p->next = last;
            return p;
      }
}

int main()
{

      int number;
      int hashedindex;
      do
      {
            number = 0;
            hashedindex = 0;
            printf("Enter number ");
            scanf("%d",&number);
            hashedindex=HASH(number);
            if(Numbers[hashedindex].data==0)
                  Numbers[hashedindex].data = number;
            else
            {
                  Numbers[hashedindex].list =
                  push_front(Numbers[hashedindex].list,number);
                  Numbers[hashedindex].count++;

            }
}while(number!=0);

for(hashedindex=0;hashedindex<MAX;hashedindex++)
{
      printf("%d->",Numbers[hashedindex].data);
```

```
        for(;Numbers[hashedindex].list!=NULL;
         Numbers[hashedindex].list
           = Numbers[hashedindex].list->next)
              printf("%d-",Numbers[hashedindex].list->data);
        printf("<Total Count=%d>",Numbers[hashedindex].count+1);
        printf("\n");
}
        getch();
        return 0;
}
```

This is a sample run of the above program.

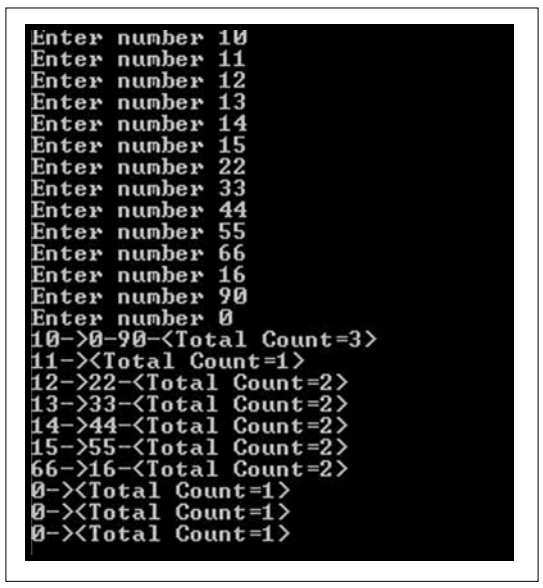


Fig. 11.4

Note that the searching in a hash table mapped with separate chaining is very simple because the location to look for a search item will always be the same as its hashed location. So we might write a wrapper function search as follows.

```
int SearchItem(int s)
{
      return HASH(s);
}
```

11.4 WHAT IS COALESCED HASHING?

Both of the above methods to resolve collision has some disadvantages as discussed above. *Coalesced Hashing* is a technique that tries to blend the advantages of both linear probing and separate chaining.

Here are the steps that coalesced follows to hash an element in a hash table.

1. The hash code of the element is calculated
2. If the location in the hash table that is mapped isn't blank then the search ends there and the element is entered there.
3. If the location is already occupied then the search is carried starting from the front of the hash table to find out a blank space.
4. Once the blank space is found the item is placed there and a pointer is attached to it from this location to the original hashed location. This approach gives the benefits of using both the approach *Linear Probing* and *Separate Chaining*.

Note in the above picture how the elements are hashed in the hash table.

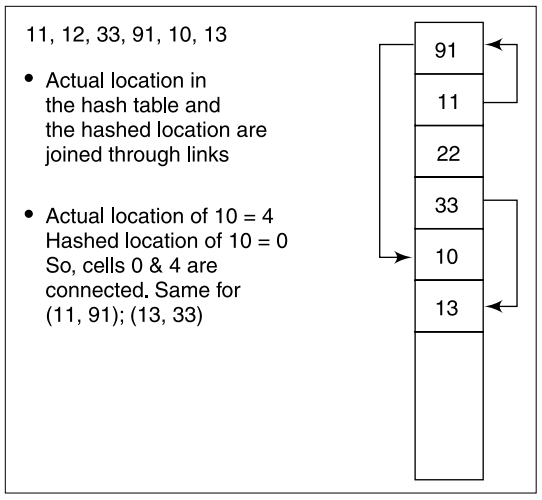


Fig. 11.5

How to Demonstrate Coalesced Hashing (*Linked Hashing*)

```
#include <stdio.h>
#include <malloc.h>
#include <conio.h>
#define MAX 10
#define HASH(X) X%MAX

typedef struct Cell
{
      int location;
      struct Cell *next;
}Cell;

typedef struct Node
{
      int data;
      Cell *nextcells;
}Node;

Node nodes[20]={0};

Cell* push_front_Cell(Cell *last,int loc)
{
      if(last==NULL)
      {
      last = (Cell *)malloc(sizeof(Cell));
      last->location = loc;
      last->next = NULL;
      return last;
}
else
{
      Cell *p = (Cell *)malloc(sizeof(Cell));
```

```
            p->location = loc;
            p->next = last;
            return p;
      }

}

void AddToHashTable(int x)
{
      int i = 0;
      int index = HASH(x);
    //We assume that the Hash Table will only contain non-zero entries.
      if(nodes[index].data == 0)
      {
            //If the Hashed location is free,
            //add the element there
            nodes[index].data = x;
            nodes[index].nextcells = NULL;
      }
      else
      {
            for(i=0;i<MAX;i++)
            {
                  if(nodes[i].data == 0)
                  {
                        //This is the first empty cell
                        nodes[i].data = x;
                        //Associate the original location with the
                        //hashed location
                        nodes[index].nextcells =
                        push_front_Cell(nodes[index].nextcells,i);
                        break;
                  }

            }
      }
}


void display_locations(Cell *front)
{
      Cell *h = front;
      for(;h!=NULL;h=h->next)
            printf("%d-",h->location);
}

void display()
{
      int i = 0;
      for(i=0;i<MAX;i++)
      {
            printf("We are at %d\n",i);
            printf("Value in this cell = %d\n",nodes[i].data);
            puts("Other values which were hashed to this cell are at
                  the locations ");
```

```
            display_locations(nodes[i].nextcells);
            printf("\n");
      }
}

int main()
{
      int i;
      int v[]={11,22,33,91,10,13};
      for(i=0;i<6;i++)
      {
            AddToHashTable(v[i]);
      }
      display();
      getch();
      return 0;
}
```

The output of the above program is as shown below.

```
We are at 0
Value in this cell = 91
Other values which were hashed to this cell are at the locations
4
We are at 1
Value in this cell = 11
Other values which were hashed to this cell are at the locations
0
We are at 2
Value in this cell = 22
Other values which were hashed to this cell are at the locations

We are at 3
Value in this cell = 33
Other values which were hashed to this cell are at the locations
5
We are at 4
Value in this cell = 10
Other values which were hashed to this cell are at the locations

We are at 5
Value in this cell = 13
Other values which were hashed to this cell are at the locations

We are at 6
Value in this cell = 0
Other values which were hashed to this cell are at the locations

We are at 7
Value in this cell = 0
Other values which were hashed to this cell are at the locations

We are at 8
Value in this cell = 0
Other values which were hashed to this cell are at the locations

We are at 9
Value in this cell = 0
Other values which were hashed to this cell are at the locations
```

Fig. 11.6

How to Search an Item from a Coalesced Hashed Table

```
int Search(int s)
{
      Cell *temp;
      int i;
      for(i=0;i<MAX;i++)
```

```
{
      if(nodes[i].data==s)
            return i;
      else
      {
            //Not found in the cell,
            //Lets search in the node's chain
            temp = nodes[i].nextcells;
            for(;temp!=NULL;temp=temp->next)
                  if(nodes[temp->location].data  == s)
                        return temp->location;
      }
   }

}
```

11.5 WHAT ARE THE VARIATIONS OF COALESCED HASHING (*LINKED HASHING*)?

The method described above is known as *Standard Coalesced Hashing.*

The method where the collided item is placed right next to the hashed location in the table like linear probing, is known as '*Early Insertion Standard Coalesced Hashing*' or **EISCH**. A generaralization of the standard coalesced Hashing method which we call general coalesced hashing, adds extra positions to the hash table that can be used for list nodes in the case of collisions but not for initial hash locations. Unlike the situation with standard coalesced hashing, the EISCH method yields worse retrieval times, than if elements are added, at the end of the chain in general coalesced hashing. A combination of these two techniques called '*Varied Insertion Coalesced Hashing*' yield the best results.

11.6 WHAT IS A HASH CHAIN AND WHAT IS ITS UTILIZATION FOR OTP?

Hash Chain as the name suggests is a chain of hashes. Basically this can be viewed as a cascaded system where the output of one hash function is the input to another. Typically the same hash function is used to generate hash chains. The number of levels is known as the *degree of the chain*. For example

h(h(h(h(h(h(x))))))

is a *Hash Chain* of degree 6. This is normally written using a short hand notation as h^6(x). Hash Chains can be used to generate one time passwords.

This scheme was first suggested by *Leslie Lompert* to generate *One Time Passwords* (OTPs). OTP is a special way in security where the passwords are usable only one time. The system generates the passwords and sends them to users through handheld wireless devices as shown in the figure above.

The user is given a password and every time say h^n() th hash code is saved in the database and matched against the n^{th} hash code of the supplied password that user typed.

The main advantage of OTP is that even if some hacker sees the last password and types that in, that is not going to work. As once the n^{th} hash code is matched then the password will be the $n–1^{th}$ hash code and will be given to the legal user. Thus OTP assures better security than that of static password.

How to Demonstrate Hash Chain and OTP Generation

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

```
#include <conio.h>
#include <string.h>

#define HASH(x) x+1//A very simple hash function

char spass[11]={"8923781234"};

int ToInt(char c)
{
      char digs[]={"1234567890"};
      int i;
      for(i=0;i<10;i++)
            if(c==digs[i])
                  return i;
}

char ToChar(int i)
{
      char digs[]={"1234567890"};
      return digs[i];
}
char* h(char pass[11])
{
      int i;
      char temp[11];

      for(i=0;i<10;i++)
            temp[i]=
              ToChar(HASH(ToInt(pass[i]))>=10?0:HASH(ToInt(pass[i])));
      temp[i]='\0';
      return temp;
}

int main()
{

      char t[11];
      strcpy(t,h(spass));
      puts(t);
      strcpy(t,h(t));
      puts(t);
      strcpy(t,h(t));
      puts(t);
      strcpy(t,h(t));
      puts(t);
      strcpy(t,h(t));
      puts(t);
      strcpy(t,h(t));
      puts(t);
      strcpy(t,h(t));
      puts(t);
      getch();
      return 0;
}
```

The output of this program is as shown below

```
9034892345
0145903456
1256014567
2367125678
3478236789
4589347890
5690458901
```

Try Yourself: *Try to improve the hash function that operates on the digits of the number so that even if a hacker gets to see few passwords in series, it still remain difficult to decode the generating sequence.*

11.7 HOW IS A HASH TREE USED TO CHECK THE DATA INTEGRITY OF A MEDIA DOWNLOADED FROM A PEER-TO-PEER (P2P) NETWORK

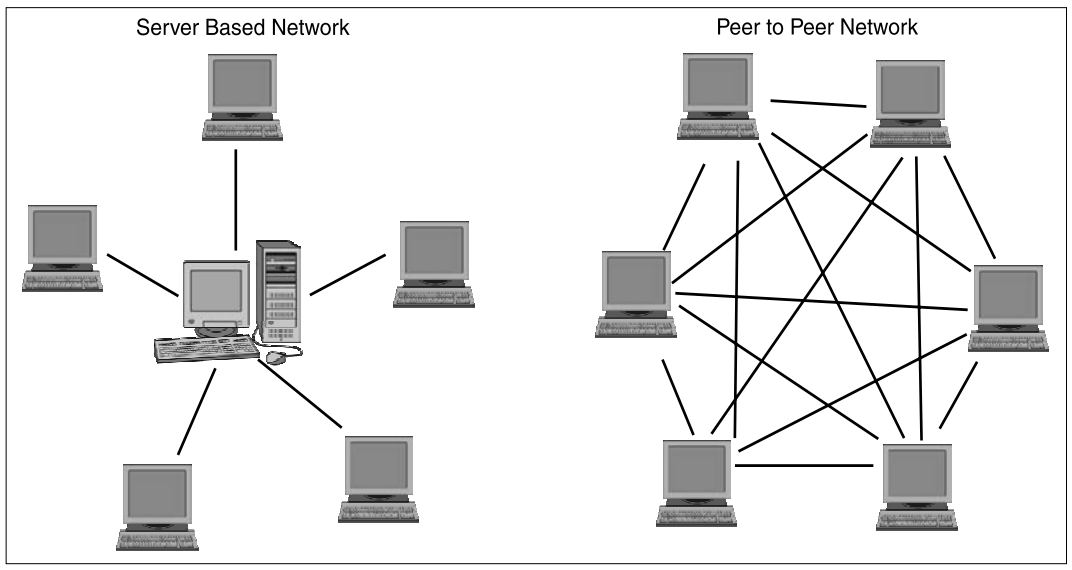
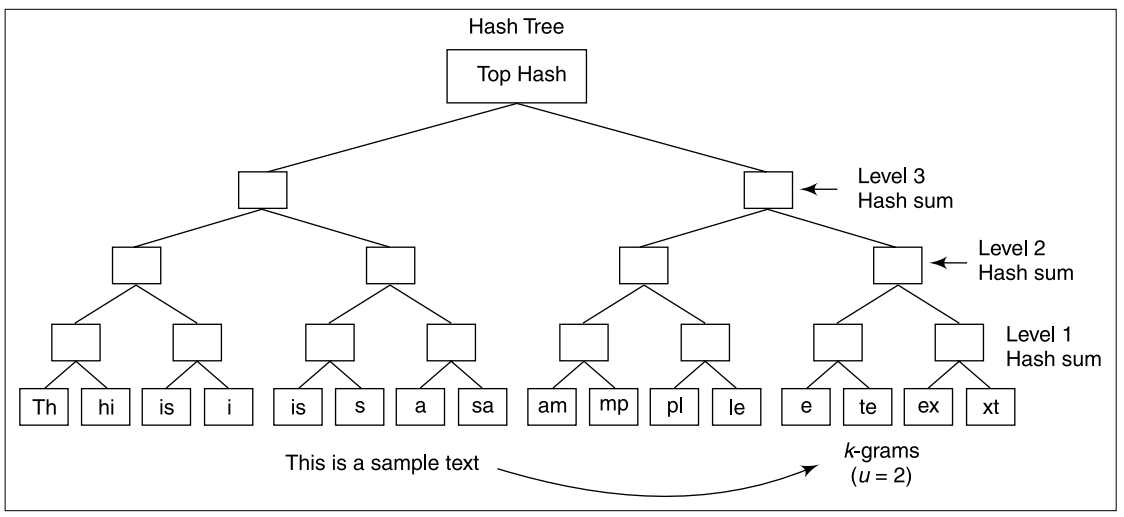


Fig. 11.7

In a peer-to-peer (a.k.a **P2P**) network as shown in the figure above each peer is connected with every other peer which makes it possible for faster downloads and sharing programs easy. To check the data integrity of the data read from a peer's disk is verified against a pre-evaluated source. This checking is done using *Hash Trees*.

Before we discuss how this can be done using hash trees, let's have a close look at what is a hash Tree. A *Hash Tree* is nothing but the tree of hash codes of *k*-grams of a document or media or some kind of data which can be broken into sequential *k* grams.

Suppose we are downloading a story through a p2p network from a peer's system and "This is a sample text" is a line in the story. To check the data integrity the hash sum of the *k* grams are first calculated. *Hash Sum* is nothing but the sum of the hash codes of each consecutive *k* grams. For example, the first box from the left in level 1 hash sum is the summation of hash codes **`H("Th")`** and **`H("hi")`** where H is the hash function being operated on the *k* grams.



Thus this calculation progresses until we get one value which is known as *Top Hash*. To check the data integrity of data being read from a unknown source in a p2p network, the *top hash* for that source is compared with that of the reliable source and if they match then we conclude that the unknown source's data is ok to download because it has passed the data integrity test. Otherwise the data is not good.

REVIEW QUESTIONS

1. What do you mean by adding salt to the password?
2. Name a few cryptographic hash functions and discuss their typical applications.
3. What is the time complexity for looking up an entry in linearly probed hash map?
4. What is the time complexity for looking up an entry in quadratically probed hash map?
5. What is the time complexity for adding an element in a hash map using linear probing?
6. Which hashing technique is better for a growing list. Justify your answer.

PROGRAMMING PROBLEMS

Implement a data integrity checker using hash trees. Use *k*-grams function defined in the chapter on strings.

12

ADT

Delivered Inbuilt Plumbing!

12.1 THE BLACK-BOX CONCEPT

We have used the `sqrt()` function to find the root of a number. But we probably didn't give any deeper thought to the implementation logic of this function. There are many different ways the `sqrt()` function can be implemented. All that bothers a client code programmer is the result and the input to the function. As long as these two remain same, nobody bothers much about the implementation logic inside. Programmers use this as a black box.

12.2 ADT

ADT is an acronym for *Abstract Data Type*. An ADT is nothing but the model of a real life entity, may be existent may be imaginary entity. Mathematically speaking an ADT is a data structure that supports some operations which can be thought of either straight simple mathematical operations or is a combination of Mathematical and Boolean Operations.
For example, Linked List is an ADT because

- It allows mathematical operations (Like Add an Element, Delete an Element, etc)
- It allows Boolean operations (Check if an element is present or not etc)
- It hides the implementation from its users.

Please recall that in the linked list chapter, once you define a linked list of integers, and say **`push_back(list, 10)`**, 10 will be added at the end of the list. As a programmer you need not bother what is the implementation logic of **`push_back()`**, just like the **`sqrt()`** example at the start of this Chapter. Thus the implementation of linked list shows abstraction capabilities (i.e. Hiding the actual implementation from the application programmers).

So we can conclude that ADT has the following properties.

- It shows *abstraction capabilities* (For more information of abstraction from a programming perspective, please refer to a good text on *Object Oriented Programming*)
- It allows Mathematical and/Or Boolean Operation

Increased usage of ADT in programs has few great advantages,

- It increases the readability of the program.
- It increases the flexibility of the program.
- It allows creating re-usable building blocks for more complex logic implementation. For example, once the Linked List ADT is defined, creating a *Binary Search Tree*(BST) ADT would be an easy task, because the BST is built with linked list ADT.

12.3 ADT DESIGN IN C

To define an ADT in C, the most basic building block could be either a C Structure or an array. Rest all Data Structures can be built from these. C does not have any ADT apart from Array. In C++ *Standard Template Library* (STL) there are a lot of ADT.

12.4 DESIGNING YOUR OWN ADT

An ADT constitute of data or attributes and methods/functions/procedures. Methods can be broadly classified into two categories, namely

- Active Methods
  - Methods that change the data of the ADT
- Passive Methods
  - Methods that doesn't affect the data of the ADT

Passive Methods can be classified into two other categories.

- Accessor Methods
  - Accessor Methods are methods that are used to access the elements of the ADT. They just access the data. They don't change the data. Thus they are passive method.
- Predictors/Search Methods (In some Self Organizing ADTs like MTF List, the 'Search' method is not a 'Passive Method'. It not only searches an item, but also puts the item at the front of the list. In such cases the search method is not a Passive Method)
  - A Predictor is a method that returns true or false depending on any particular given evaluation criterion for all items in the ADT.

Active Methods can also be of the following types

- Methods to add new element in the ADT
- Methods to delete existing element in the ADT
- Methods to update existing element in the ADT.

When designing an ADT, you need to know what you want that ADT to contain? What will be the operations needed on the data of the ADT? The answer to the first question will tell you about the data of the ADT and the answer to the second question gives the idea of the operations, methods of the ADT.

Let's take the example of the Linked List functions described in the linked list chapter.

`push_back()`

- This method puts an element at the end of the list.
- Active method.
- A kind of add operation.

`push_front()`

- This method puts an element at the front of the list.
- Active method.
- A kind of add operation.

`pop_back()`

- This method deletes the last element of the list.
- Active method.
- A kind of delete operation.

`pop_front()`

- This method deletes the first element of the list.
- Active method.
- A kind of delete operation.

`delete_at()`

- This is a method that deletes the element at the given location.
- A kind of delete operation.

`delete_range()`

- This method deletes a range of elements.
- A kind of Delete Operation.

`delete_alternate()`

- This method deletes alternate elements from the list.
- A kind of delete operation.

`front_element()`

- This method gets the first element of the list.
- A kind of the accessor method.

`back_element()`

- This method gets the last element of the list.
- A kind of the accessor method.

`get_value()`

- This method gets the value of the nth element in the list.
- A kind of accessor method.

`frequency()`

- This method gets the number of occurrences of an element in the list.
- A kind of accessor method and of statistical importance.

`findmax()`

- This method gets the maximum integer in the integer linked list.
- A variation of the accessor method and of statistical importance.

`findmin()`

- This method gets the minimum integer in the integer linked list.
- A variation of the accessor method and of statistical importance.

`swap()`

- This method swaps two nodes contents.
- A kind of update operation.

swap_head_tail()

- This method swaps the first and the last element of the list.
- A kind of update operation.

merge()

- This method merges two linked list.
- A kind of update operation.

count()

- This method counts the number of nodes in the linked list.
- This is an accessor operation.

In the next chapters Date, Map and Currency we will learn how to define our own ADTs as and when required.

REVIEW QUESTIONS

1. What type of method is push()?
2. What type of method is peek() at stack top?
3. What type of method is enqueue()?
4. What type of method is deque()?
5. What type of method is displayQueue() which displays a queue?
6. What do you mean by a Heterogeneous Container?
7. What kind of method can change the content of a container?

PROGRAMMING PROBLEMS

1. Create an ADT "Dictionary" that will have methods for the following operations
   (a) To add a word to it
   (b) List all the words starting with a letter
   (c) Search for a word
   (d) Delete a word
   (e) Update a word
2. Create an ADT "Bag" that will be able to hold element of different types. And will have methods for the following operations
   (a) Adding an element to the bag
   (b) Deleting an element from the bag
   (c) Updating an element in the bag
   (d) Searching an element in the bag
   (e) Finding out the type of the element in the bag
3. What is the type of the method push_front() in linked list?
4. What is the type of the method pop() in stack?
5. What is the type of the method pop_back() in linked list?
6. What is the type of the method pop_front() in linked list?
7. What is the type of the method search() in MTF?
8. What is the type of the method in find_first_if() method in linked list?
9. Create a program to demonstrate binary tunnel (Refer Chapter on Tree. A Binary Tunnel is nothing but a linked list of Binary Spider).

13

Date

Today was Tomorrow!!

INTRODUCTION

Date is an integral part of any solution that we develop. Many programming languages offer APIs(*Application Programming Interfaces*) to deal with the dates. There is a built-in data structure in Turbo C called date, (discussed about this structure in structure chapter). Actually, this structure is used in the today() function (that will be discussed in this chapter) which will return the day of week for today. The functions described in this chapter are written using Turbo C 3.5. So it is recommended to use Turbo C 3.5 or Higher to compile and Test these functions

The structure used to represent date objects is

```
typedef struct Date
{

      int day;
      int month;
      int year;
} Date;
```

Only in the function today() the built-in system structure **date** is used. In this chapter we will learn how to design date related functions and how to use them to solve different real world problems. At the end of the chapter we will learn how to interact with the built-in structure date to deal with system date.

How to Check whether a Year is a Leap Year or Not

```
enum{NO,YES};
int isleapyear(int year)
{
      int leap = NO;
      if(((year%4==0) && (year%100!=0)) || (year%400==0))
                  leap = YES;
      return leap;
}
```

Please notice that how the function uses enum variables to increase the readability of the code.

Here is a call to this function

```
int x = isleapyear(2008);
if(x==1)
      printf("Leap Year");
else
      printf("Non Leap Year");
```

This will print Leap Year to the console as 2008 is a leap year.

How to Find the Date of Tomorrow of any Date

```
Date tomorrow(Date d)
{
      switch(d.month)
      {
            case 1://Jan
            case 3://Mar
            case 5://May
            case 7://July
            case 10://Oct
            case 8:if(d.day<31)//Aug
                    d.day++;
                   else//If day is 31, ie, last day
                   {
                         //Then move tomorrow will be
                         //the first day of the next month
                         d.day = 1;
                         d.month++;

                   }
                   break;
            case 12:if(d.day<31)
                      d.day++;
                    else//If we are on the last day of the year
                    {
                     //Tomorrow will be
                     //first January of the next year
                      d.month = 1;
                      d.day = 1;
                      d.year++;
                    }
                    break;
            case 2:if(isleapyear(d.year))
                    {
//in case leap year
   if(d.day<29)
      d.day++;
   else
   {
//we are in March
      d.day=1;
```

```
      d.month=3;
   }
}
else//The year is not leap year
{
   if(d.day<28)//the day is less than 28
     //one more day to go
     d.day++;
                     //we are on 28th Feb of a non leapyear
                   else
                   {
                     //we are in March
                     d.day=1;
                     d.month=3;
                   }
                 }
                 break;
          case 4://April
          case 6://June
          case 9://September
          case 11:if(d.day<30)//November
                    d.day++;
                 else
                 {
                      d.day = 1;
                      d.month++;
                 }
                 break;

       }
       return d;
}
```

Here is a typical call to the function.
//Here we want to find tomorrow of 28th Feb 2008 which is leap year.

```
Date d,td;
d.day = 28;
d.month = 2;
d.year = 2008;
td = tomorrow(d);
printf("Tomorrow of %d-%d-%d is %d-%d-
%d",d.day,d.month,d.year,td.day,td.month,td.year);
```

The output of this code snippet will be
Tomorrow of 28-2-2008 is 29-2-2008

How to Find the Date of Yesterday of any Date

```
Date yesterday(Date d)
{
      switch(d.month)
      {
      case 1: if(d.day>1)
                  d.day--;
```

```
            else
            {
                  //If standing on 1st January
                  //we want to find what was yesterday
                  //then yesterday will be 31st December
                  //the previous year.
                  d.day=31;
                  d.month=12;
                  d.year--;
            }
            break;

      //Finding yesterday for a date in March
      //if it is not the first date of the month
      //we just slide backward otherwise
      //we need to check whether the year is leap year?
      //If the year is leap year then previous day of March
      //first will be 29th Feb else it will be 28th Feb.
      case 3: if(d.day>1)
                d.day--;
              else
                {
                    if(isleapyear(d.year))
                    {
                          d.day=29;
                          d.month=2;
                    }
                    else
                    {
                          d.day=28;
                          d.month=2;
                    }
                }
                break;
      case 2://Feb
      case 5://May
      case 7://July
      case 8://Aug
      case 10://Oct
      case 12:if(d.day>1)
                    d.day--;
              else//we reached the first date of the month
              {
                    //The previous month's last day is 30
                    d.day=30;
                    d.month--;
              }
              break;
      case 4://April
      case 6://June
      case 9://September
      //Havn't we reached the start of the month
```

```
case 11:if(d.day>1)
            d.day--;
      else//we are at the start of the month
      {
            //the last month had 31 days.
            d.day=31;
            d.month--;
      }
            break;
      }
      return d;
}
```

How to Calculate the Difference of Days between Two Dates in the Same Year

```
int daysbetween(Date d1,Date d2)
{
      int diff=0;
      int i=0;
      diff+=d2.day-d1.day;//Initial day difference
      for(i=d1.month;i<d2.month;i++)
      {
            //If month is Jan, March, May, July, Aug,
            // October or December, add 31 to the difference
            if(i==1 || i==3 || i==5 || i==7 || i==8
            || i== 10 || i==12)
                  diff+=31;
            //February and Leap Year, add 29 days
            if(i==2 && isleapyear(d1.year))
                  diff+=29;
            else
            //February and not leap year, add 28 days
                  diff+=28;
            //rest all months, i.e,
            ///April, June, September and November
            //add 30
            if(i==4 || i==6 || i==9 || i==11)
                  diff+=30;
      }
      return diff;
}
```

Please note that this function calculates the difference of days between two dates in a year. This function serves as a basic building block of other date related functions.

Try This: *How can you change this function so that it tells the difference between two dates in different or same years.*

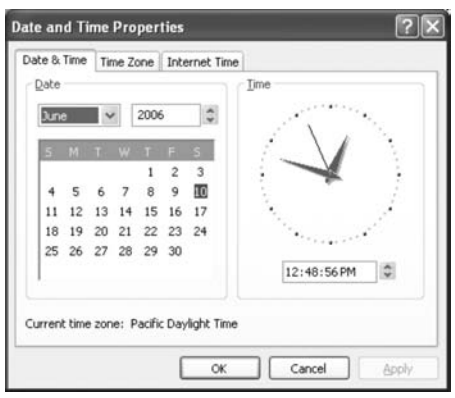
13.1 HOW TO FIND THE DAY OF WEEK (SUN, MON, ETC)

The normal strategy to solve this problem is to find the day difference between the two particular dates,

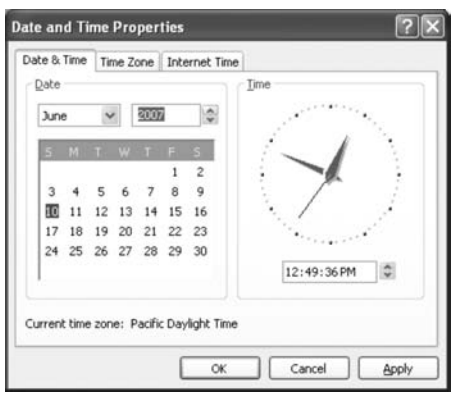
i.e. target date and source date. *Target date* is the date for what we want to know the dayofweek and *Source Date* is the date which we use for reference, and then dividing the total with 7. The remainder of this division tells you the day of week for that date starting from the day of week of the source/base date. First January 1900 was Monday. So if the remainder is 2, then the target date is Wednesday, (which is 2 days ahead from Monday).

But this above process is time consuming. It involves multiplication by 365.

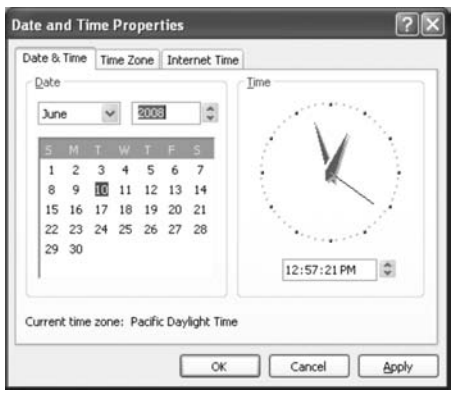
A different approach is used in this function. Please look into the screenshots of windows calendar carefully.

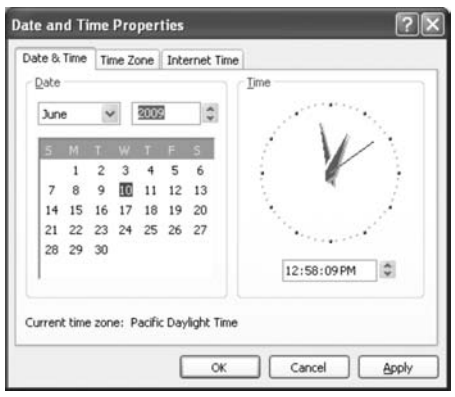


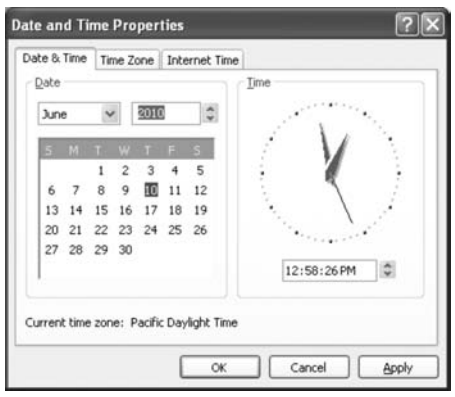
Showing 10-Jun-2006

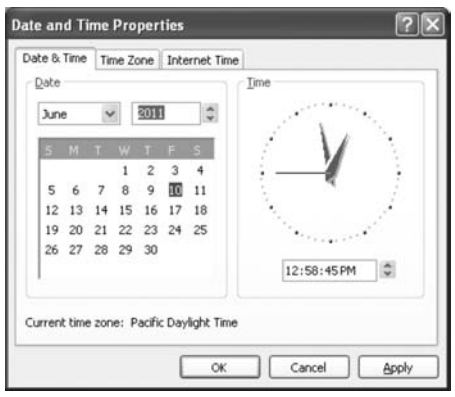


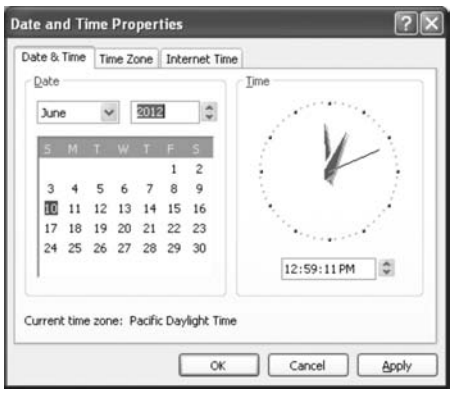
Showing 10-Jun-2007











Have you noticed from the above screenshots that a particular day, shifts towards right as the year increases. When we go from one non-leap-year to a leap-year then the day of week of a particular shifts two days towards right by two units. See the change for the transformation 2007 to 2008 and 2011 to 2012.

So by the above theory, we can calculate what will the day of week for first January of any given year without any multiplication, just by adding 1 and 2 and shifting that many days towards right starting from Monday, as 1-January-1900 was Monday.

Suppose we have to find out the day of week for the date 10-Jun-2014.

Then first by using the above formula, we have to find the day of week for the date 1-Jan-2014. and then adding the days difference using daysbetween() function defined above.

```
int dayofweek(Date d)
{
      Date t;
      int i;
      int sum=0;
      for(i=1900;i<d.year;i++)
      {
            if(isleapyear(i))
                  sum+=2;
            else
                  sum++;
      }
      t.day = 1;
      t.month = 1;
      t.year = d.year;
      sum+=daysbetween(t,d);
      return sum%7;
}
```

This function returns 1 for Tuesday and so on till 5 for Saturday, 0 or 7 for Monday, and rest all values returned are interpreted as Sunday.

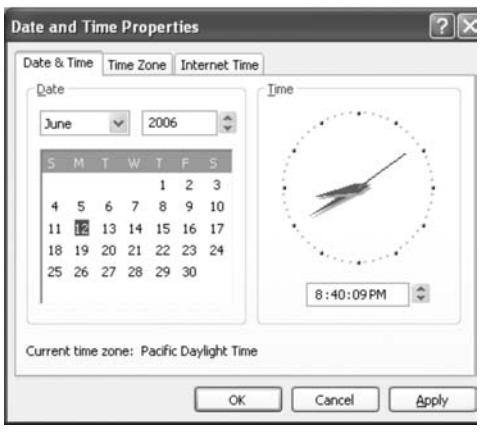
How to Show the Day of the Week in String

```
char* daystring(int dow)
{
      char *d;
      switch(dow)
      {
            case 1:strcpy(d,"Tuesday");break;
            case 2:strcpy(d,"Wednesday");break;
            case 3:strcpy(d,"Thursday");break;
            case 4:strcpy(d,"Friday");break;
            case 5:strcpy(d,"Saturday");break;
            case 7:
            case 0:strcpy(d,"Monday");break;
            default:strcpy(d,"Sunday");break;
      }
      return d;
}
```

This function will be used to display the Day of Week in Human readable format.
Here is a call

```
Date d;
d.day = 12;
d.month = 6;
d.year = 2006;
printf("%d-%d-%d is %s",d.day,d.month,d.year,daystring(dayofweek(d)));
```

This code snippet will display
12-6-2006 is Monday



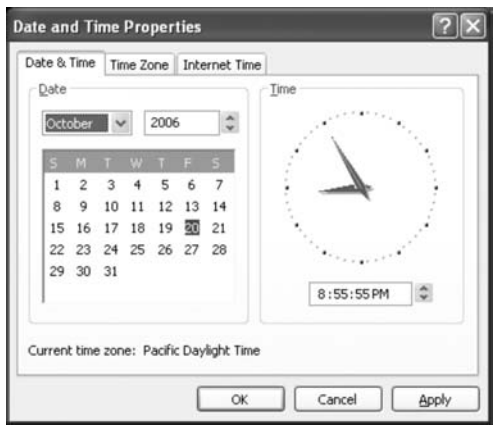
The screenshot proves the answer.

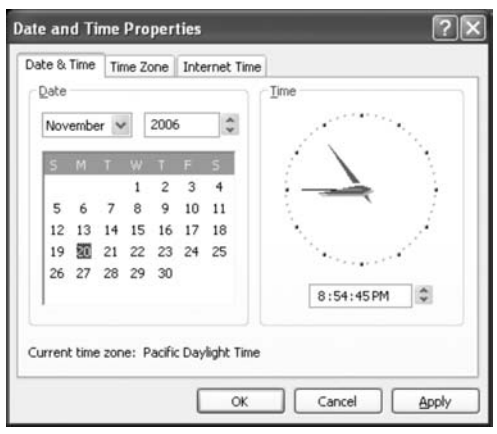
13.2 HOW TO FIND THE DATE OF THE NEXT NTH SUNDAY, FROM A GIVEN DATE

//Given a day in string format, this function returns the corresponding DAYOFWEEK as //an integer

```
int resolvedow(char *day)
{
      int dow=0;
      if(strncmpi(day,"Tuesday",strlen(day))==0)
            dow = 1;
      if(strncmpi(day,"Wednesday",strlen(day))==0)
            dow = 2;
      if(strncmpi(day,"Thursday",strlen(day))==0)
            dow = 3;
      if(strncmpi(day,"Friday",strlen(day))==0)
            dow = 4;
      if(strncmpi(day,"Saturday",strlen(day))==0)
            dow = 5;
      if(strncmpi(day,"Sunday",strlen(day))==0)
            dow = 6;
      return dow;

}
//This function returns the date of Nth X day starting from
//a particular date. For example, if we want to find the 5th
```

```
//Monday starting from 20-OCT-2006 then the call to this
//function will be like
//Date d,d1;
//d.day = 20;
//d.month = 10;
//d.year = 2006
//d1 = nextNthXday(d,5,”Mon”);//OK
//d1 = nextNthXday(d,5,”Monday”);//Ok
//The function is case insensitive.
//You have to write those many characters for the day of
//week, as many as needed to differentiate from others.
//Like in the above case if we write “Mo” for Monday the
//function understands what we mean. We need not Write the
//entire day of week string.
//The above call returned d1.day = 20, d.month = 11,
//d.year = 2006

//Here are two screenshots of windows calendar
```





Look into the screenshots and count the 5th Monday starting from 20-OCT-2006 and see whether the program calculated it right as 20-NOV-2006.

```
Date nextNthXday(Date d,int N,char *day)
{
      Date counter;
      int total = 0;
      //Initializing loop counter to the starting date.
      counter.day = d.day;
      counter.month = d.month;
      counter.year = d.year;


      for(;;counter=tomorrow(counter))
      {
       if(resolvedow(daystring(dayofweek(counter)))
            ==resolvedow(day))
                  total++;
            if(total==N)//are we on the target date
                  break;
      }
      return counter;//return the date
}
```

13.3 HOW TO FIND THE DATE OF THE PREVIOUS NTH SUNDAY, FROM A GIVEN DATE

Here is the function. Notice carefully that there is only one word difference in the whole function. Instead of tomorrow, we have to use yesterday, because we are going backwards unlike the previous case where we were going towards right in timeline.

```
Date prevNthXday(Date d,int N,char *day)
{
      Date counter;
      int total = 0;
      counter.day = d.day;
      counter.month = d.month;
      counter.year = d.year;


      for(;;counter=yesterday(counter))
     {                                                    //Change
            //integer comparison is much
            //faster than string
            if(resolvedow(daystring(dayofweek(counter)))
           ==resolvedow(day))
                  total++;
            if(total==N)
                  break;

      }
      return counter;
}
```

13.4 WRAPPER FUNCTIONS: INCREASES THE READABILITY OF YOUR CODE

These above two functions can be used from within various wrapper functions. A *Wrapper function* is a function which calls a complicated function using some hardcoded values to return some specific values.

Suppose I want to find what will be the date of the next Friday, I can write like

Next_Friday_date = nextNthXday(current_date,1,"Fri");

But here I have to give 3 variables to the function, which I disliked. So I have written few wrapper functions to find the immediate next *X*day and immediate past *X*day. Where *X* represents DAYOFWEEK (SUN,MON, etc).

```
Date nextSUNDAY(Date d)
{
      return nextNthXday(d,1,"Sun");
}

Date lastSUNDAY(Date d)
{
      return prevNthXday(d,1,"SuN");
}
```

And so on for other days of the week.

How to Find the DAYOFWEEK for a Particular Date in the Previous Year

```
int prevyearsameday(Date d)
{
      d.year--;//Lets go back one year
      return dayofweek(d);
}
```

How to Find the DAYOFWEEK for a Particular Date in the Next Year

```
int nextyearsameday(Date d)
{
      d.year++;
      return dayofweek(d);
}
```

Example 13.1 ***Find the DAYOFWEEK for a particular date in the next month? Let me clarify the question. Suppose we are on 22-APR-2006, and we want to find the DAYOFWEEK of 22-MAY-2006.***

Solution Here is the code.

```
//This function returns the last date of the month of the
//date object passed to it as argument.
//For example
//if we call
//Date d;
//d.day = 2;
//d.month  = 2;
//d.year = 2008
//So if we call lastday() like
//printf("Last date of Feb 2008 is %d-%d-%d\n"
//,lastday().day,lastday().month,lastday().year);
//Then the output will be
//Last date of Feb 2008 is 29-2-2008
```

```
Date lastday(Date d)
{
      switch(d.month)
      {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:d.day=31;
                  break;
            case 2:if(isleapyear(d.year))
                        d.day=29;
                   else
                        d.day=28;
                   break;
            case 4:
            case 6:
            case 9:
            case 11:d.day=30;
                     break;
       }
       return d;

}


int nextmonthsameday(Date d)
{
      //Lets hold the day before tomorrow overrides it
      int x = d.day;
      //We have to go to the next month
      //so that is nothing but tomorrow() of
      //the last date of the current month
      //or the month passed.
      d = tomorrow(lastday(d));
      d.day = x;//lets reassign the original day
      return dayofweek(d);//returns DAYOFWEEK as integer
}
```

Example 13.2 ***Sometimes we have to find the date of the n^{th} X day in the next/previous month. For example the clerk at the bank may tell you, "Sorry Sir, We are not able to do this today, could you please come on second saturday next month?" Now you will start thinking what will be the date. Here is the code that may help you.***

Solution

```
Date nextmonthNthXday(Date d,int N,char *day)
{
      return nextNthXday(tomorrow(lastday(d)),N,day);
}
```

Now let me explain how to call this function. Suppose now it is October and you want to find 3rd Friday of November of the same year. You need to pass any date in October so the function can find the first date of November.

```
Date d;
d.day = 4;//The day, Any day between the range of days
d.month  = 10;//The month you are in
d.year = 2001;
//Now call the function
d = nextmonthNthXday(d,3,"Friday");
//d will contain the date of the third Friday in November
//the same year
```

You might have already understood that as we discussed above we can write some Wrapper functions here also.

How to Find the Last X Day of the Next Month

```
Date nextmonthlastXday(Date d,char *day)
{
      //There can be maximum 5 weeks
      //in a month and the last
      //X day may fall on this last
      Date t[5];
      d = tomorrow(lastday(d));
      t[0] = nextNthXday(d,1,day);
      t[1] = nextNthXday(d,2,day);
      t[2] = nextNthXday(d,3,day);
      t[3] = nextNthXday(d,4,day);
      t[4] = nextNthXday(d,5,day);
      if(t[4].month == d.month)
            return t[4];
      else
            return t[3];
}
```

How to Find the First X Day of the Next Month

//This function can be used to answer queries like
//what will be the date of the first Monday next month and so

```
Date nextmonthfirstXday(Date d,char *day)
{
      d = tomorrow(lastday(d));
      return nextNthXday(d,1,day);
}
```

How to Find the DAYOFWEEK of 29th February of all the Leap Years in a given Range

We have written a function to find whether a year is leap year or not, and we have also written a function to return the DAYOFWEEK of a day. Now how to combine these two functions to write a function that will accept two arguments as starting and ending year. And it displays all those years which are leap year and also tells the DAYOFWEEK of 29th February on those years.

Here is the code of the function

```
void displayleapyears(int s,int f)
{
      Date d;
      int i=0;
      d.day=29;
      d.month = 2;
      for(i=s;i<=f;i++)
            if(isleapyear(i))
            {
                  d.year = i;
                  printf("%d 29-FEB-%d will be %s\n"
                  ,i,i,daystring(dayofweek(d)+1));
            }
}
```

When called like
displayleapyears(2006,2080) ;
it generates the following output

2008 29-FEB-2008 will be Friday
2012 29-FEB-2012 will be Wednesday
2016 29-FEB-2016 will be Monday
2020 29-FEB-2020 will be Saturday
2024 29-FEB-2024 will be Thursday
2028 29-FEB-2028 will be Tuesday
2032 29-FEB-2032 will be Sunday
2036 29-FEB-2036 will be Friday
2040 29-FEB-2040 will be Wednesday
2044 29-FEB-2044 will be Monday
2048 29-FEB-2048 will be Saturday
2052 29-FEB-2052 will be Thursday
2056 29-FEB-2056 will be Tuesday
2060 29-FEB-2060 will be Sunday
2064 29-FEB-2064 will be Friday
2068 29-FEB-2068 will be Wednesday
2072 29-FEB-2072 will be Monday
2076 29-FEB-2076 will be Saturday
2080 29-FEB-2080 will be Thursday

How to Find whether a Date is in Past Compared to Another Date

```
int isPast(Date d1,Date d2)
{
      //returns 1 if d2 is in past that d1
      if(d1.year>d2.year
```

```
      || (d1.year==d2.year && d1.month>d2.month)
       ||(d1.year==d2.year && d1.month==d2.month
         && d1.day>d2.day))
            return 0;
      else
            return 1;
}
```

How to Check whether Two Dates are Same or Not

```
//this function returns 1 if both the dates are same
int isPresent(Date d1,Date d2)
{
      if(d1.day == d2.day && d1.month == d2.month
       && d1.year == d2.year)
            return 1;
      else
            return 0;
}
```

How to Check whether a Date is in Future in Respect to Another Date

```
//this function returns 1 if d1 is at future of
int isFuture(Date d1,Date d2)
{
      if(!isPresent(d1,d2) && !isPast(d1,d2))
            return 1;
      else
            return 0;
}
```

How to Get the Particular Date After or Before a Number of Days

Let me explain this. Suppose we are on 2-Jan-2007, and we want to find what will be the date after 90 days or before 60 days. This function helps to solve these type of problems.

```
Date adddays(Date d,int N)
{
       int c=0;
       Date counter;
       counter.day = d.day;
       counter.month = d.month;
       counter.year = d.year;
       if(N>0)//We are going to the future
             for(;c<N;c++,counter=tomorrow(counter));
       else
             for(;c<abs(N);c++,counter=yesterday(counter));
       return counter;
}
```

To call this function,

```
Date d;
d.day = 1;
```

```
d.month = 2;
d.year = 2002;
d = adddays(d,20);
printf("%d-%d-%d\n", d.day,d.month,d.year);
```

This will print 21-2-2002

Try Yourself: *Try calling this function with such inputs so that there is a migration between years in both directions.*

How to Get the Particular Date After or Before a Number of Months

Let me explain this. Suppose we are on 2-Jan-2007, and we want to find what will be the date after 9 months or before 14 months. By month, I mean 30 days. This function helps to solve these types of problems.

```
Date addmonths(Date d,int N)
{
      return adddays(d,N*30);
      return d;
}
```

Notice, how the adddays() function us being used.

How to Get the Particular Date After or Before a Number of Years

Let me explain this. Suppose we are on 2-Jan-2007, and we want to find what will be the date after 9 years or before 14 years.

```
Date addyears(Date d,int N)
{
      d.year+=N;
      return d;
}
```

13.5 INTERACTION WITH THE SYSTEM BUILT IN `Date` STRUCTURE

As discussed in Chapter 2: Structures, and in the front of this chapter, the date structure is built in. There are two functions getdate() and setdate() to read and write system time. As the ultimate motto for this API design is abstraction of core logic from application programmers, so a function named today() is introduced as a wrapper of getdate(). Here is the code.

```
Date today()
{
   Date td;//Our Date structure
   struct date d;//System date structure

   getdate(&d);//getting the date
   td.year = d.da_year;
   td.day  = d.da_day;
   td.month = d.da_mon;
   return td;
}
```

So if we want to get today's date we need not write getdate anymore we can write like

```
printf("Today's date is %d-%d-
%d\n",today().day,today().month,today().year);
```

This approach increases the readability of the code.
Now let's write another interface for setting the system date.

```
void setsystemdate(Date d)
{
   struct date reset;
   reset.da_year = d.year;
   reset.da_day = d.day;
   reset.da_mon = d.month;
   setdate(&reset);

}
```

This above function will reset the system date. BE VERY CAREFUL TO RUN THIS. ALWAYS USE A FUTURE DATE TO TEST THIS. IN CASE YOU USE SOME PAST DATE, SOME IMPORTANT FILES CAN BE MISSING FROM YOUR DISK.

13.6 INTERACTION WITH THE REAL WORLD

Till now we have created date objects from within our code. But in the real life this will happen only rarely. There we shall have to interact with the outer world and we shall have to accept dates as strings. Then we shall have to tokenize them. If you have ever worked with Oracle you know that there is a date function called to_date() that takes two strings. First one describes the date as a string and the second one tells the function in which format the date is entered. Let me explain this a little more. If I say the date is 10-2-1978 then it may refer to October 2nd 1978 or 10th February 1978. So the phenomenon creates an ambiguity. That is where the format string comes to help.

The format strings will be like

- dd-mm-yyyy
- mm-dd-yyyy
- dd-mon-yyyy
- mon-dd-yyyy
- yyyy-mon-dd
- mon-yyyy-dd
- dd-yyyy-mm //Not that popular though
- Only a number telling the number of the date in the year (For example, if I say 35th Day of the year, it will always represent the date 4th February and so on)

Here is the function code for to_date.

```
int tomonth(char *mon)
{
      if(strncmpi(mon,"Jan",3)==0)
            return 1;
if(strncmpi(mon,"Feb",3)==0)
      return 2;
if(strncmpi(mon,"Mar",3)==0)
      return 3;
if(strncmpi(mon,"Apr",3)==0)
      return 4;
```

```
if(strncmpi(mon,"May",3)==0)
      return 5;
if(strncmpi(mon,"Jun",3)==0)
      return 6;
if(strncmpi(mon,"Jul",3)==0)
      return 7;
if(strncmpi(mon,"Aug",3)==0)
      return 8;
      if(strncmpi(mon,"Sep",3)==0)
            return 9;
      if(strncmpi(mon,"Oct",3)==0)
            return 10;
      if(strncmpi(mon,"Nov",3)==0)
            return 11;
      if(strncmpi(mon,"Dec",3)==0)
            return 12;
}


Date to_date(char *date,char *f)
{
      int i;
      char *formats[]
            ={
             "dd-mm-yyyy",//2-3-2004
             "mm-dd-yyyy",//4-3-2004
              "dd-mon-yyyy",//2-oct-1976
              "mon-dd-yyyy",//july-31-1978
              "yyyy-mon-dd",//1966-Oct-20
              "mon-yyyy-dd",//Sep-1989-21
              //365-2005, This is the last date of the year
              "NUM-yyyy"
      };
      Date d;
      char *p;
      int dayofyear=0;
      for(i=0;i<6;i++)
            if(strcmpi(formats[i],f)==0)
                  break;
      switch(i)
      {
            case 0:p=strtok(date,"-");
              d.day = atoi(p);
              p = strtok(NULL,"-");
              d.month = atoi(p);
              d.year = atoi(strtok(NULL,"-"));
  break;
      case 1:p=strtok(date,"-");
            d.month = atoi(p);
            p = strtok(NULL,"-");
            d.day = atoi(p);
            d.year = atoi(strtok(NULL,"-"));
            break;
```

```
            case 2:p=strtok(date,"-");
                   d.day = atoi(p);
                   p = strtok(NULL,"-");
                   d.month = tomonth(p);
                   d.year = atoi(strtok(NULL,"-"));
                   break;
            case 3:p=strtok(date,"-");
                   d.month = tomonth(p);
                   p = strtok(NULL,"-");
                   d.day = atoi(p);
                   d.year = atoi(strtok(NULL,"-"));
                   break;
            case 4:p=strtok(date,"-");
                   d.year = atoi(p);
                   p = strtok(NULL,"-");
                   d.month = tomonth(p);
                   d.day = atoi(strtok(NULL,"-"));
                   break;
            case 5:p=strtok(date,"-");
                   d.month = tomonth(p);
                   p = strtok(NULL,"-");
                   d.year = atoi(p);
                   d.day = atoi(strtok(NULL,"-"));
                   break;
            case 6:p=strtok(date,"-");
                   dayofyear = atoi(p);
                   p = strtok(NULL,"-");
                   d.year = atoi(p);
                   d.day = 1;
                   d.month = 1;
                   d = adddays(d,dayofyear-1);
                   break;
      }
      return d;

}
```

Here is how to call this function.

```
Date d = to_date("2006-DEC-31","yyyy-mon-dd");
```

This will create the date as d.day = 31, d.month = 12 and d.year = 2006.

```
Date d = to_date("365-2005","nuM-yyyy");
```

This will create the date as d.day = 31, d.month = 12 and d.year = 2005

REVIEW QUESTIONS

1. What is the function that compares two dates.
2. How will you find the total difference in days between 2nd October 1972 and 31st October 2007.
3. How will you find how many Thursdays are there between 2nd October 1980 and 2nd October 2007
4. How will you find how many months between 1st January 2005 and 1st January 2008 have 5 Sundays?
5. Someone is born on 29th February 2000 then when should she celebrate her 16th birthday? What will be the day of week for her 20th birthday?

PROGRAMMING PROBLEMS

1. Accept a list of dates from the user and return only Sundays.
2. *Couple on Longest Holiday Problem: There is a couple. Let's name them Jack and Jill.* Jack's company works in shifts and for a particular year Jack finds that he will get holidays on second Saturday and Third Sunday every month. On the other hand, the company where Jill works has a fixed holiday list for each employee. Jill's company keep closed for every Saturday and Sunday. Accept the holiday (say thanksgiving day) list of Jill's company. Now with all this information can you write a program that will tell them the date in a year they should plan to enjoy the longest holiday together?
3. Ask the user his date of birth and then tell the day of week of his birthday.
4. Ask the user his date of birth and then calculate his age and display that in "You are walking on earth for past X year Y months and Z days".
5. A museum is kept closed on every 3rd Friday if the year is a Leap Year. Else it is kept closed on every 2nd Friday starting from first January. Saturday and Sunday are by default holidays. One very old artwork had been stolen from that museum in the year 1996. Police have arrested a person who is telling that someone from the museum helped him. There are 4 curators of the museum. They work in a rotational shift starting from the first person on 1st January. Can you write a program to help police identify the guilty curator ? [Hint: The person working longest can be the culprit.]
6. There is some improvement. Police now got the information that the man was seen in and around museum starting from July to September. Now accordingly modify your program written for 5.
7. Using the dayofweek() function described in the text and other functions as you may need, can you write a program that will print a customized calendar. A customized calendar is a calendar where the user can set what is the range of dates that he wants to see. For example, in UNIX there is a command cal. If you write cal 2067 then it will print the whole calendar for 2067. If you write cal 10 2067 then it will display the calendar for October 2067. If you just write cal it will display the current month's calendar. Can you write such a command based program? Use PgUp, PgDn to scroll vertically or you can use arrow keys.
8. Write a wrapper function to find the next Sunday.
9. Write a wrapper function to find the previous Thursday.
10. Write a function to find the date of last Sunday of a year.
11. Write a function to find the date of the first Monday of a year.
12. Write a function to accept two dates in string format. Pass them to **`to_date()`** function and then check whether there are *N* Sundays between them or not. [*N* will be provided.]
13. Write a function nextFriday() that will return the next Friday.
14. Write a function nextMonthsFridays() that will return the Fridays of the next month as a date array
15. Write a function Fridays() that will return all the Fridays of the year passed as argument.
16. Write a function GetBDaysDOW() that will accept two years as arguments and a date as birthday and will return the Date of Week of birthdays in all those years inclusive of the two.
17. Write a function Age() that accepts two dates and returns the time passed between these two in terms of years, months and days.
18. Write a function Older() that accepts two birthdates and return 0 or 1 depending upon which is older.
19. Write a function Younger()that accepts two birthdates and return 0 or 1 depending upon which is younger.
20. Write a function Get5WeeksMonths() that will accept a year and return the months that has 5 week for that particular year.
21. Write a function to_string() that will be complementary to to_date().

14

Map
Phonebook, Dictionary, Cryptography

INTRODUCTION

Map is the name given to any table that stores a key-value pair. A value from the map is accessed using a key. The map doesn't allow duplicate keys. The key of the map can be of any data type. The value of the map can be of any data type. A multimap is a map or maps.

A Map doesn't allow duplicate entries, but a multimap does because it actually stores the duplicate values on different maps. Maps are good if they are array implemented, because in that case they offer O(1) time complexity. C does not have associative arrays as built in data structure. Maps are used to design associative arrays in C. Associative arrays or maps find application in many real life problems from a simple synonym antonym mapping to a complex random block ciphering.

In this chapter we will first learn how to declare a map and then will discuss about the diverse usage of this data structure. A map is basically a structure with couple of elements, one represents the key and the other represents the value.

About the Methods in This Chapter The methods described in this chapter are close cousins of the methods defined in the linked list chapter. Here the map is defined using the linked list.

14.1 HOW TO REPRESENT A MAP

Here a map has been designed using a linked list of buddy structures. The value in the key-value pair that are stored in a map could be anything. Here a buddy structure is stored as the value and each buddy is associated with a key.

```
typedef struct Buddy
{
      char name[20];
      char phone[11];
}Buddy;

//Node of the Map
typedef struct node
{
      Buddy pal;//value
      int key;//key
      struct node *next;
}node;
```

How to Add a Buddy at the End of the Buddy List

```
node* push_back(node *last,Buddy pal,int key)
{
      if(last==NULL)
      {
            last = (node *)malloc(sizeof(node));
            last->pal = pal;
            last->key = key;
            last->next = NULL;
            return last;
      }
      else
      {
            //creating enough space for a Buddy!
            node *p = (node *)malloc(sizeof(node));
            if(p)
            {
                  last->next = p;
                  p->pal = pal;
                  p->key = key;
                  p->next = NULL;
            }
            return p;
      }
}
```

How to Add a Buddy at the Front of the Buddy List

```
node* push_front(node *h,Buddy pal,int key)
{
      node *p = (node *)malloc(sizeof(node));
      p->next = h;
      p->pal = pal;
      p->key = key;
      return p;
}
```

How to Delete the Last Buddy in the List

```
//Delete the last node
node* pop_back(node *h)
{
	node *p = h;
	node *r = p;
	for(;;)
	{
		if(p->next->next==NULL)
		{
			free(p->next->next);
			p->next = NULL;
			break;
		}
		p=p->next;
	}
	return r;
}
```

How to Delete the First Buddy in the List

```
//deletes the first element
node* pop_front(node *h)
{
	//identifying the first node that has to be freed
	node *x = h;
	node *p = h->next;
	free(x);//freeing the memory space.
	return p;
}
```

How to Delete a Buddy/Map Entry at a Particular Index

```
node* delete_at(node *h,int location)
{
	int c=0;
	node *p=h;
	node *r=p;
	node *x;
	if(location<count(h))
	{
		//loop till we find the location
		for(;p!=NULL;p=p->next)
		{
			c++;
			//human readabale location
			//if you want to delete the fifth element
			//it is actually the fourth in the list
			//so human readable location is 5
			//but actual location is 4
			if(c==location-1)
				break;
}
```

```
      //Identify which memory location to free
      x = p->next;
      p->next = p->next->next;
      //free that location
      free(x);

}
      return r;
}
```

How to Delete a Range of Buddies/Map Entries

```
//deletes a particular range of elements
node* delete_range(node *h,int start,int finish)
{
      node *p=h;
      int c=0;
      int k=0;
      for(c=start,k=0;c<=finish;c++,k++)
            delete_at(h,c-k);
      return p;
}
```

How to Delete Alternate Entries of Buddies/Map Entries

```
//deletes alternate elements from the list
node* delete_alternate(node *h,int start,int finish)
{
      node *p=h;
      int c=0;
      for(c=start;c<=finish;c++)
            delete_at(h,c);
      return p;
}
```

Notice carefully, how delete at has been used from inside delete_alternate

How to Get the First Friend/Entry in the Buddy List/Map

```
node* first(node *h)
{
      return h;
}

//returns the front element of the list
Buddy front_element(node *h)
{
      return first(h)->pal;
}
```

How to Get the Last Friend/Entry in the Buddy List/Map

```
node* last(node *h)
{
      node *p = h;
      for(;p->next!=NULL;p=p->next);
```

```
      return p;
}

//returns the back element of the list
Buddy back_element(node *h)
{
      return last(h)->pal;
}
```

How to Count the Number of Buddies in the List

```
int count(node *h)
{
      int numberofnodes=0;
      node *p = h;
      if(p==NULL)
            return 0;
      else
      {
            for(;p!=NULL;p=p->next)
                  numberofnodes++;
            return numberofnodes;
      }
}
```

How to Replace a Particular Entry in the Map

```
node* replace(node *h,int location,Buddy pal)
{
      int c=0;
      node *p=h;
      node *r=p;
      node *q=(node *)malloc(sizeof(node *));
      if(location<count(h))
      {
            //
            for(;p!=NULL;p=p->next)
            {
                  c++;
                  if(c==location-1)
                        break;
            }

            q = p->next;
            q->next = p->next->next;
            q->pal = pal;
      }
      return r;
}
```

How to Swap Two Contents of the Map at two Different Locations

```
//This function swaps any two elements in the list
node* swap(node *h,int loc_a,int loc_b)
```

```
{
   Buddy temp;
   node *p=h;
   temp = get_address(h,loc_a)->pal;
   p=replace(p,loc_a,get_value(h,loc_b));
   p=replace(p,loc_b,temp);
   return p;
}
```

How to Swap Head and Tail of a Map

```
//This function swaps the head and the tail
//or the front and the back element of the list
node* swap_head_tail(node *h)
{
      node *p = h;
      Buddy temp = p->pal;
      p->pal = back_element(h);
      for(;p->next!=NULL;p=p->next);
      p->pal = temp;
      return p;
}
```

14.2 HOW TO DEFINE A PREDICATOR OVER A MAP AND USE IT FROM A CLIENT

A *predicator* is a function that returns true or false depending on some condition.

```
//This is a client display function
//that iterates through the map
//and checks if the buddy satisfies the
//condition (The name starts with a S) or not
//If the buddy satisfies, then it will display
//the buddy details. Else it will not display
//the buddy's name and number.
void display_if(node *h)
{
      node *p = h;
      for(;p!=NULL;p=p->next)
      //Refer the function startsWith in Chapter on String
            if(startsWith(p,"S"))
                  printf("Key %d  Name %s Phone %s\n ",p->key,
                      p->pal.name, p->pal.phone);

}
```

14.3 HOW TO KNOW WHO IS WHO'S FRIEND FROM THE BUDDY LIST

From the above buddy list representation it is not possible to know if some buddy of your phonebook is also a buddy to another person in the list. To maintain this we can keep an array of integers that will store the keys of a buddy's pals. The structure will change like

```
typedef struct node
```

```
{
      Buddy pal;
      int key;
      int pals[10];
      struct node *next;
}node;
```

The pals array will store the keys of the persons who are friend to this buddy.

The above implementation could prove to be handy in many different situations.

For example, we can write a function like this

```
Buddy[] GetPalsOfThisBuddy(int BuddyKey)
{

  //This function will return an array of Buddy
  //Structures that will represent the Buddies
  //For the buddy whose Key is supplied.

}
```

14.4 HOW TO DESIGN A RANDOM CIPHER ENCODER USING A MAP

Cipher Text is the product of a mathematical operation (Mostly Linear Translation) over a plain text. For example a *" +1 Cipher of a text"* moves all the characters in the text to its right by one character. If we get *z* it makes it *a* (i.e. Circular, when we go out of alphabet, we start from beginning , *z* + 1 = *a* and so on).

A cipher text generated like this is highly prone to hacking risks because computers are incredibly fast on checking all these additive, subtractive ciphers and decode them..

One way to protect our strings is to encrypt it with a random cipher. *Random Cipher* is a technique that replaces a whole word with a totally different pattern or sequence of characters. So that the Hacker will never have the slightest clue about what is going on.

Say the word, random will be replaced by the pattern "**4A&$)<**"

So we can store the words and their equivalent *Random cipher replacements* in a map.

When a string will be given to us, we can just stroll through the string and replace the words (Keys in the map) with their *Random Cipher Encoded Values*. (values in the map). To decode such an encrypted string we just use a map with reversed key value pair. Articles and prepositions would also be part of the map. For verbs, to represent the past tense, the mapped value will be concatenated with #$ for example the value for Like is say "J8*p" then an instance of 'liked' in the original string will be replaced with "J8*p#$".

To make things worse for the hacker we can do an additive or subtractive (or some kind of linear ciphering) over the generated encoded string from the first pass.

The below application uses the following maps

Now if we convert the string

I had a plate of Pasta. I liked. I will have again.

we will get,

@ T#@ I*@F2 O% ^^&tZ. @ (x)21#$. @ mul2 Y^&k 89GjU.

Assuming single space between words.

Notice the #$ is appended due to past tense.

Key	*Value*
A	*
An	**
The	***
Has	U&9
Have	Y^&k
Had	T#@
Like	(x)21
Pasta	^^&tZ
Normally	Qxzm*iiz
Will	mul2
Again	89GjU
Plate	I*@F2
I	@
Of	**O%**

Encryption Map

Key	*Value*
*	A
**	An
***	The
U&9	Has
Y^&k	Have
T#@	Had
(x)21	Like
^^&tZ	Pasta
Qxzm*iiz	Normally
mul2	Will
89GjU	Again
I*@F2	Plate
@	I
O%	**Of**

Decryption Map

14.5 APPLICATION OF MAP OF MAPS

A phonebook is designed using a map given earlier in this chapter. We can change the code so that multiple user will access it. First, we will define the above map that will be used for the phonebook and then we will describe another map that will hold a pointer to the phonebook map and a string key. It will look like

```
typedef struct Buddy
{
      char name[20];
      char phone[11];
}Buddy;

//Node of the Map
typedef struct node
{
      Buddy pal;//value
      int key;//key
      struct node *next;
}node;


typedef struct MultiMapNode
{
     string key;
     node* BuddyList
     struct MultiMapNode *next;
}MultiMapNode;
```

Here string in the above structure denotes the identification of the phonebook. This way we can write a program to make room for multiple people to store their phonebook.

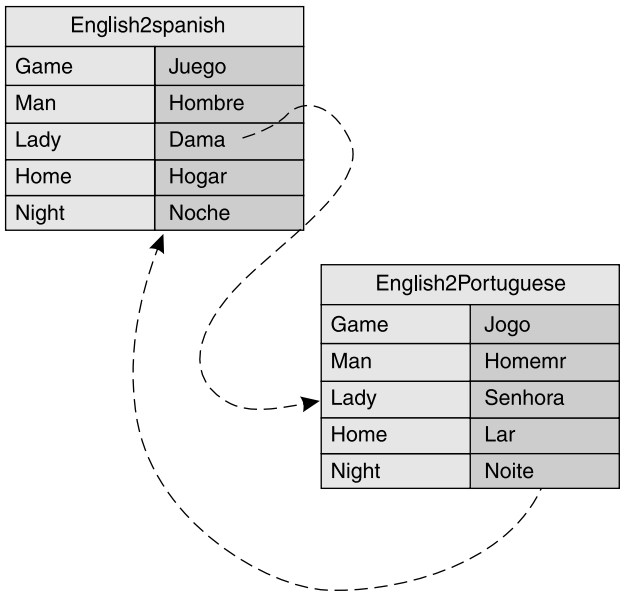
14.6 MULTILANGUAGE WORD MAP

Famous lines are translated in many languages. A simple multimap structure can be used to store the synonyms of a word, in different languages. Each map will demonstrate a map for translation from one language to the other. For example the first map will map words from English to Spanish the second from English to Portuguese, and so on. Now if a word is given in any language, without any more information the program will be able to tell the following two informations

1. Which language is it from (assuming the word is from any of these three languages)
2. What is its meaning

If you want to extend you could do that by adding maps for synonyms and antonyms and homonyms (if any exists for the given word)

Suppose the word 'Dama' is entered. We will seek the word list in the English2Spanish map. Looking at it we can find out that Dama means Lady in Spanish. Now to know its synonym in Portuguese we have to go to the English2Portuguese map and return the value of the pair whose key is Lady. English is the common language kept in the two maps. Similarly, when someone says 'Noite' you look into English2Spanish map and find no match. Then you search English2Portuguese map and find that 'Noite' means 'Night' in Portuguese. Now if you want to find out what is the Spanish synonym of 'Noite' you need to go back to English2Spanish map and get the value for the key 'Night'.



This approach could be quite useful for simple sentence translation from one language to an other.

But as the sentence becomes bigger, grammar of different languages differs almost unpredictably. So for Big sentence this could not be used.

Another Typical Application of this approach is to translate cheque amount written in one language to the other. For example a German will write

Zwanzig Fünf Tausend neun hundert und siebzig acht

For

Twenty Five Thousand nine hundred and seventy eight

But if you have the map that maps one to hundred and the conjunctions like and etc., then you can easily map the German words to English.

14.7 KEY INTERLINKED MAP (KIM)

If in a map the keys are interlinked then that map can be called as *Key Interlinked Map* or KIM. There can be many ways that a key of a map is related to the other part of it. A very easy way to relate a key to another, is to create the value of the current the key of the other. For example the following map holds the name of few individuals as key, value pair. The key value holds the employee's name and the value holds his/her boss's name.

Employee	*Manager*
Sam	Otto
Joan	Kim
Otto	Joan
Kim	**John**

So from this map we can say that **Joan** is two levels up than **Sam.**

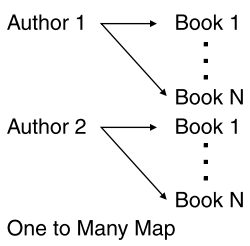
Try Yourself: *Create such a KIM of names and once a name is entered, return the name of the manager of the manager of the employee's name supplied.*

REVIEW QUESTIONS

1. What do you mean by map or maps? How can that be used to solve different association problems in different industry.
2. What is the time complexity of search function in a map?
3. What is the time complexity of insert function in a map?
4. What is the time complexity to look up in an interlinked map?

PROGRAMMING PROBLEMS

1. How can we implement a dictionary using maps where for each word, meaning, synonyms, antonyms will be stored. A word can have many antonyms and synonyms. So you might consider using linked list to store them.
2. Implement a new data structure called SortedDictionary where the entries will be added according to the sorted key values. For example an entry (Key="City" Value="O'Fallon") will be followed by an entry (Key="BigCity" Value="St. Louis"). That means St. Louis will appear before O'Fallon in the map.
3. Implement a word translator with the concepts described in the chapter. Try to use SortedDictionary. This will save you time for search.
4. How will you implement a one to many map? Let's visualize an example of one to many mapping.



As shown in the figure, an author might have written *N* number of books. So when storing the Author–Book couple in a map, that map becomes a "One to Many" map, because in this case the author (Which is a single Key, Maps to different Books). *In the text we discussed about Maps with no duplicate keys.*

5. Write a function to add an entry in an interlinked map.
6. Write a function to delete an entry in an interlinked map.
7. Write a program to implement English to Spanish Numeric Translation.
8. Write a program to demonstrate how maps can be used to represent if–else blocks effectively.
9. Write a program that uses the concept of program 4 and extend the idea to show the premium of an insurance.
10. Write a program to demonstrate how a map can be used to represent in a decision tree.
11. Write a program to demonstrate how a map can be used to represent basic political atlas.

15

Currency
No Primitive Please!

INTRODUCTION

Currencies need special attention in any program as they can't be efficiently represented using the primitive data types like float or double. All present *relational database systems* like Oracle, SQL Server, MySQL etc. support Currency Data Type. Creating a new *currency data type* for each currency is a good programming practice because it gives the program much more professional look and the code becomes much more readable and easily understandable. For example to represent the salary of an employee in USA, we can write **USD salary**, which makes much more sense rather than just writing float salary. So we should not use primitive data types when we know that the variable that we are going to deal with will/may have some other behaviors than that of a primitive data type which though at a first look might seem to be a close match.

In this chapter we will discuss how we can create different currency data type using primitive C building blocks (read structures) and then we will write different functions that will be operating on these currency data types.

How to Model USD Currency as a Structure

```
typedef struct USD
{
      int dollars;
      float cents;
}USD;
```

How to Add Two Amounts in USD

```
USD Add(USD amount_1, USD amount_2)
{

      USD temp;
```

```
temp.dollars = amount_1.dollars + amount_2.dollars;
temp.cents = amount_1.cents + amount_2.cents;

if(temp.cents>100)
{
      temp.cents = temp.cents - 100;
      temp.dollars+=1;
}//Highest possible cent amount is
      //98 cents in any addition.


      return temp;
}
```

How to Convert a String to a Corresponding USD Amount

```
#include <string.h>
#include <conio.h>
#include <ctype.h>


USD StringToUSD(char amount[])
{
      //$3,44,500
      USD temp;
      double dbldollaramount;
      int intdollaramount;
      char samount[15];
      int i = 0;
      int j=0;
      for(i=0;i<strlen(amount);i++)
      {
            if((amount[i]!='$' && amount[i]!=',')
         && (toascii(amount[i])>=48
          || toascii(amount[i])<=57
          || amount[i]=='.'))
            {
                  samount[j]=amount[i];
                  j++;
            }

      }
      samount[j]='\0';
      dbldollaramount = atof(samount);
      temp.dollars = dbldollaramount;
      intdollaramount = dbldollaramount;
      temp.cents = dbldollaramount-intdollaramount;
      temp.cents*=100;
      return temp;
}
```

See how this function is called from a *client code* (*A client code is a code that calls another code. For example we will call this function from main function then main function is the client code for this function*)

```
int main()
{
      USD a = StringToUSD("$45.78 USD");
      USD b = StringToUSD("$78.99 USD");
      USD c = Add(a,b);
      printf("%d Dollars %.0f  Cents",c.dollars ,c.cents);
      getch();
      return 0;
}
```

The output of the above program is
124 Dollars 77 Cents

Note how clean the code looks. And the StringToUSD is a very versatile function. I have given all kind of dirty inputs to it but it successfully constructs the USD object correctly every time. I have tried the following conversions.

```
int main()
{
      USD a = StringToUSD("$13,560.23 United States Dollars");
      printf(" The amount + 1 Dollar  = %d Dollars %.0f
              Cents",a.dollars + 1,a.cents);
      getch();

      return 0;
}
```

The output of the above program is
The amount + 1 Dollar = 13561 Dollars 23 Cents

See how the StringToUSD() function commas.

Try Yourself: *Modify the StringToUSD() function such that it returns a structure called result containing a USD object and a character array. The character array will store the conversion comment, stating whether it is successful or not. So the calling of this function will look like*

```
typedef struct Result
{
      USD amount;
      Char *comment;
}Result;

Result StringToUSD(char *string)
{
      Result result_of_this_conversion;

      // Place your code here
      // Populate the elements of the result structure.

      Return result_of_this_conversion;
}
```

So while calling this function we can write

```
Result Conversion = StringToUSD(“$12,569.45 US Dollars”);
if(strcmpi(Conversion.Comments,”Successful”)==0)
```

```
{
      //Do whatever you want with the converted amount
      //Conversion.amount
}
else
{
      //Find out what exactly has happened by checking
      //the result comment
      //and tell the user where exactly he/she went wrong.
}
```

How to Check if Two USD Amounts are Equal or Not

```
int isEqual(USD a, USD b)
{
      return a.dollars==b.dollars && a.cents == b.cents;
}
```

Try to give the function cascaded input (*A Cascaded input is an input generated in a system from another system*) like

```
USD b;
b.dollars = 44;
b.cents =24;
if ( isEqual (Add(StringToUSD("$40.24"),StringToUSD($4.00"))
     ,b)  = = 1)
{
      //Do Something
}

else
{
      //Do Something else
}
```

How to Check if One USD is Greater than the Other

enum {NO,YES}; //an Enum type variable is used to make the code more readable.

```
int isGreater(USD a,USD b)
{
      int Greater = NO;

      if(a.dollars>b.dollars)
            Greater = YES;
      if(a.dollars == b.dollars)
      {
            if(a.cents>b.cents)
                  Greater = YES;
      }

       return Greater;
}
```

How to Check if One USD is Lesser than the Other

```
enum {NO,YES};
//an Enum type variable is used to make the code more readable.
```

```
int isLesser(USD a,USD b)
{
      int Lesser = NO;

      if(a.dollars<b.dollars)
            Lesser = YES;
      if(a.dollars == b.dollars)
      {
            if(a.cents<b.cents)
                  Lesser = YES;
      }

      return Lesser;
}
```

Let's define few other structures depicting some other currencies.

How to Model GBP Currency as a Structure

```
typedef struct GBP
{
      int pound;
      float penny;
}GBP;
```

How to Model Euro Currency as a Structure

```
typedef struct Euro
{
      int eurodollars;
      float eurocents;
}Euro;
```

How to Find the Highest USD Amount from a Series Supplied

```
USD GetMeMax( USD amounts[],int size)
{
      int i;
      USD GreatestAmount = amounts[0];
      for(i=1;i<size;i++)
      {
            if(isGreater(amounts[i],GreatestAmount)==1)
                  GreatestAmount = amounts[i];

      }
      return GreatestAmount;
}
```

How to Find the Least USD Amount from a Series Supplied

```
USD GetMeMin( USD amounts[],int size)
{
      int i;
      USD LeastAmount = amounts[0];
      for(i=1;i<size;i++)
```

```
      {
            if(isLesser(amounts[i],LeastAmount)==1)
                  LeastAmount = amounts[i];

      }
      return LeastAmount;
}
```

A Client Code that Uses the Above Methods (*Putting It Together*)

```
int main()
{
      USD amounts[] =
     {
       StringToUSD("$45.23"),
       StringToUSD("$244.22 Dollars"),
       StringToUSD("$12,441.19 Dollars")
     };//Carefully notice how the array is initialized!
      USD max = GetMeMax(amounts,3);
      USD min = GetMeMin(amounts,3);

      printf("Greatest amount is %d Dollars and %.0f
      Cents\n",max.dollars,max.cents);
      printf("Least amount is %d Dollars and %.0f
      Cents\n",min.dollars,min.cents);


      getch();

      return 0;
}
```

The output of the above client program is

```
Greatest amount is 12441 Dollars and 19 Cents
Least amount is 45 Dollars and 23 Cents
```

15.1 A PRACTICAL APPLICATION: GETTING THE LOWEST BID AMOUNT

Suppose you are hired by a bidding company to find out what is the lowest bid amount from a list of amounts collected online. Now there lies a challenge. When you are allowing your user base to enter the amount in whatever way they please, you need to make sure that finally the amounts are stored correctly.

We can convert the amount entered by them with their names or web login credentials (LoginID and password). And store these details in a flat file. Now we can read that file and store the values in an array of structure (*That will depict a bid, with the identification of the client who bid that amount and the amount*).

Here is the structure to store a *Bid*:

```
typedef struct Bid
{
      Char UserID[10];
      Char Password[10];
      USD  BidAmount;
}
```

Here is the approach that reads the bids from a flat file (*A notepad file is sometimes referred as a flat file, to distinguish it from other file structures*) which has the following format.

UserID Password BidAmount

Read from File

Using strtok () function or some other function extract the tokens user id, password and the bid amount and then store them in an array of bid like

```
strcpy(bids[i].UserID,/* Put the first token here */);
strcpy(bids[i].Password, /* Put the second token here */);
bids[i].BidAmount = StringToUSD(/* Put the third token here */);
```

Once you have the bid array created, you can find what is the least bid amount by a code like

USD LowestBidAmount = bids[i].BidAmount;

```
//Assuming there are 10 rows in the file./for 10 clients
for(i=1;i<10;i++)
{
      if(bids[i] .BidAmount<LowestBidAmount)
      {
            LowestBidAmount = bids[i].BidAmount;
      }
}
```

Now that you have the LowestBidAmount, you can tell who proposed this lowest bill just by strolling the list once more.

```
for(i=1;i<10;i++)
{
      if(bids[i].BidAmount==LowestBidAmount)
      {
            printf("User ID = %s ",bids[i].UserID);
      }
}
```

After you define other currencies like USD, say GBP, EURO etc, then we can even handle the above problem with multiple type of currencies. What I mean is we can find out the max or min amount from a list of amounts in different currencies.

15.2 HOW TO CONVERT USD TO GBP AND VICE VERSA

Currency conversion is probably the most performed operation on any currency ever modeled.

A typically ideal programmatic conversion from one currency to another follows the following set of techniques.

- Decide to have a base currency.
- Declare constants that stores the conversion rates from this currency to others.
- Use these conversion constants across the program.
- When the rates change, we just need to change these programming constants.

Here is an example.

```
#define USD2GBP 0.517505
#define USD3EUR 0.764435

GBP USD2GBP(USD amount)
{
      GBP temp;
```

```
      float t = amount.dollars + amount.cents/100;
      t*=USD2GBP;
      int ct = t;
      temp.pound = ct;
      temp.penny = t - ct;
      temp.penny*= 100;
      return temp;
}
```

Rates are taken from www.xe.com

The reverse would be similar, to find the equivalent USD amount for an amount given in GBP

Here is the function that will perform this conversion:

```
USD GBP2USD(GBP amount)
{
      USD temp;
      float t = amount.pound + amount.penny/100;
      t/=USD2GBP;
      int ct = t;
      temp.dollars = ct;
      temp.cents  = t - ct;
      temp.cents*= 100;
      return temp;
}
```

Similar methods can be written for other currencies.

Try Yourself: *Accept a list of Conversion Requests in the following format:*

10 GBP to USD
2.4 USD to EURO
22.23 EURO to GBP

And output as

10 GBP = x Dollars and y cents

And so on..

15.3 HOW TO CONVERT USD TO GBP AND VICE VERSA DATEWISE

The conversion rates vary datewise. So we might want to use different rates for different dates. In this way we can even compare performance of a currency with respect to others over a span of time. To represent such changing rate, a map whose keys are date and values as conversion rates can be used.

The entries of the map would be objects of the structure

```
typedef struct Rate
{
      Date d;
      double USD2GBP;
      double USD2EURO;
      //and so on
}Rate;
```

REVIEW QUESTIONS

1. What will be the output of the following code snippet?

```
if(isLesser(stringtoUSD("$145.34"),USD2GBP(stringtoUSD("123")))
      puts("Share price increased");
else
      puts("Oh! No! the price is still the same");
```

2. If we want to find out the equivalent GBP amount for a USD amount for different conversion rates using a function then how will the function signature look like?
3. `x = stringToUSD("$6000.45").dollars`; What can you conclude about the data type of *x* from this statement.

PROGRAMMING PROBLEMS

1. Write a program that will read a list of amounts in USD and GBP and will return the maximum amount.
2. Write a program that will read a list of amounts in USD and GBP and will return the minimum amount.
3. Write a program that will read a list of bidding amounts associated with a bidder name. A Bidder can put multiple bids though. The program should output the bidder with the least bid amount. Assume that the bidding amounts can be either in USD or in GBP. If nothing is mentioned then the amount is in USD.
4. Create a currency Euro.
5. Create a currency Lira.
6. Create a currency Peso.
7. Create a currency Rubble.
8. Create a currency AUD.
9. Write a function that will convert from USD to any of these as passed as argument.
10. Write a function to find out the maximum amount from a list of amounts in all such currencies.
11. Write a function to find out the minimum amount from a list of amounts in all such currencies.

16

File Handling

Seed, Save, Share

INTRODUCTION

Whatever we do in our real life, we need to store the details in hard drives for future purposes. Starting from storing the transaction details for a pizza outlet to music storage for ipod Nano, every bit of research is being affected by how we can store the information in the disk. Files are also used heavily in game programming. One such example is described at the end of this chapter. Reading a file and analyzing the words in it, we can classify the document. "Google News" is a great example of this technique. In this chapter we will discuss about the library functions that C offers for file handling. After that we will discuss how files can be used to solve many diverse problems, which are very similar in one aspect. All these problems have a need for data storage in some way or the other.

It is advised that you read chapter on string before you go ahead with this chapter because we will use some functions written in String chapter here in this chapter. Anyways, you can continue and whenever you find the references, jump to chapter on Strings for instant clarifications.

16.1 WHAT IS FILE?

FILE is a built-in structure that represents the file in C. In C any file is a stream of characters. To do any kind of operation on a FILE in C, we need to create a FILE pointer that will be the indicator of the FILE.

Example 16.1 ***Write a program to demonstrate fopen(), fclose(), fputc() and fgetc().***

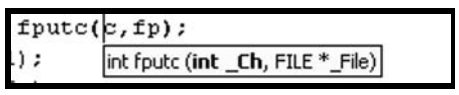
Solution fopen() opens a file in different mode namely *r* (read mode) , *w* (write mode) and *a* (append mode). If the file does not exist and we open it in '*a*' mode then the file will be created before writing. If we open an existing file in *w* mode then the existing file will be overwritten. The '*r*' mode is used to read the file starting from the first character. fopen() takes the FILE pointer parameter to stroll through the file. Fopen() signature is shown in the box below.

```
fp = fopen("C:\\test.txt","r");
puts("This
```

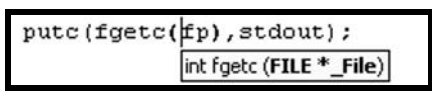
FILE * fopen (**const char *_Filename**, const char *_Mode);

fclose() signature is shown in the box below.

fputc() signature is shown in the box below.



fgetc() signature is shown in the box below.




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

int main()
{
      FILE *fp;
      char c;
      fp = fopen("C:\\test.txt","w");
      do
      {
            c = getche();//Reading a character from the screen
            if(c==13)//Let the loop run till we hit "Enter"
                  break;
            else
             //Let's put the character in the file
             //pointed by FILE pointer fp
                  fputc(c,fp);
      }while(1);
      fclose(fp);
      puts("File written!");
      fp = fopen("C:\\test.txt","r");
      puts("This is what you have written in the file");
      while(!feof(fp))
      {
            //Getting characters one by one from the text file
            //and displaying them on the console
            putc(fgetc(fp),stdout);
      }
                  fclose(fp);

                  getch();
                  return 0;
}
```

Example 16.2 *Write a program to demonstrate fgets() and fputs().*

Solution `fgets()` signature is shown in the box below.

```
fgets(line,81,fp);
puts(l
```
char * fgets (**char *_Buf**, int _MaxCount, FILE *_File)

`fputs()` signature is shown in the box below

```
fputs("Water can flow and it can crash.\n",fp);
fputs(
```
int fputs (**const char *_Str**, FILE *_File)

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

int main()
{
   FILE *fp;
   char line[80];
   fp = fopen("C:\\test.txt","w");
   fputs("Water can flow and it can crash.\n",fp);
   fputs("Be water my friend!",fp);
   fclose(fp);
   fp = fopen("C:\\test.txt","r");
   while(!feof(fp))
   {
         fgets(line,81,fp);
         puts(line);

   }
   fclose(fp);
   getch();
   return 0;
```

The output of this program will be

```
Water can flow and it can crash.
Be water my friend!
```

Example 16.3 ***Write a program to demonstrate fprintf() and fscanf().***

Solution `fprintf()` signature is shown below in the box.

```
fprintf(fp,"%s\t%d\n",name,age);
count++;
```
int fprintf (**FILE *_File**, const char *_Format, ...)

`fscanf()` signature is shown below in the box

```
fscanf(fp,"%s\t%d\n",name,&age);
printf(
```
int fscanf (**FILE *_File**, const char *_Format, ...)

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

int main()
{
      FILE *fp;
      int count = 0;
      char name[20];
      int age = 0;
      fp = fopen("C:\\test.txt","w");
      do
      {
            fflush(stdin);
            puts("Enter name and age of the person ");
            scanf("%s%d",name,&age);
            fprintf(fp,"%s\t%d\n",name,age);
            count++;

      }while(count!=3);

      fclose(fp);

      fp = fopen("C:\\test.txt","r");
      while(!feof(fp))
      {
            fscanf(fp,"%s\t%d\n",name,&age);
            printf("%s\t%d\n",name,age);
      }
      fclose(fp);
      getch();
      return 0;
}
```

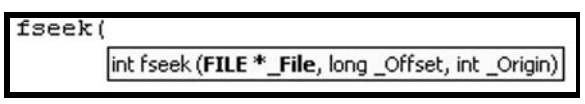
The same pattern is used to read and write

This is a sample run output of the above program.

```
Enter name and age of the person
Sam
27
Enter name and age of the person
Andy
26
Enter name and age of the person
Viks
28
Sam     27
Andy    26
Viks    28
```

Example 16.4 ***Write a program to demonstrate the use of fseek() and ftell().***

Solution fseek() is a function that allows us to move forward or backward from the three different locations (beginning of the file, current location of the file pointer and the end of the file) in the file. Here is the function signature of the fseek().



the origin is from where we want to travel. It can have either of these three set integer values 0, 1 or 2.

If the origin is SEEK_SET that means we want to move from beginning of the file.

If the origin is SEEK_CUR that means we want to move from current location in file.

If the origin is SEEK_END that means we want to move from end of file.

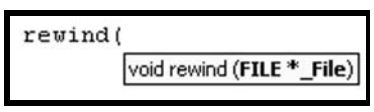
Ftell() returns the current value of the position indicator of the _File pointer. For binary streams, the value returned corresponds to the number of bytes from the beginning of the file. For text streams, the value is not guaranteed to be the exact number of bytes from the beginning of the file, but the value returned can still be used to restore the position indicator to this position using fseek.

Here is a program that measures the size of a file using these two functions.

```
#include <stdio.h>

int main ()
{
  FILE * fp;
  long size;

  fp = fopen("test.txt","rb");//Assume that the file exists


    fseek (fp, 0, SEEK_END);
    size=ftell (fp);
    fclose (fp);
    printf ("Size of test.txt: %ld bytes.\n",size);

  return 0;
}
```

16.2 WHAT DOES THE FUNCTION rewind() DO

This function is a wrapper function. If sometime while reading the file, we want to start all over again from the starting of the file, then we can call this function. The box below shows the signature of this function.



rewind() is basically same to a call to fseek(fp,0L,SEEK_SET);

How to Write a Function wcl() to Find the Line Count of a File

```
int wcl(char *filename)
{
      int count = 1;
```

```
FILE *fp;
fp = fopen(filename,"r");
//The loop will rotate till the end of the file
while(!feof(fp))
{
      //Are we done with this line?
      //If yes then the line counter needs to be
      //increased by unity
            if(fgetc(fp)=='\n')
                  count++;
      }
      fclose(fp);
      return count++;//Return the total line count
}
```

How to Write a Function wcw() to Find the Word Count of a File

Assuming that the words in the text file are separated by blank spaces,

```
int wcw(char *filename)
{
      int count = 0;
      char word[20];
      FILE *fp;
      fp = fopen(filename,"r");
      while(!feof(fp))
      {
           //reading a word
           fscanf(fp,"%s",word);
                  count++;
      }
      fclose(fp);
      return count;
}
```

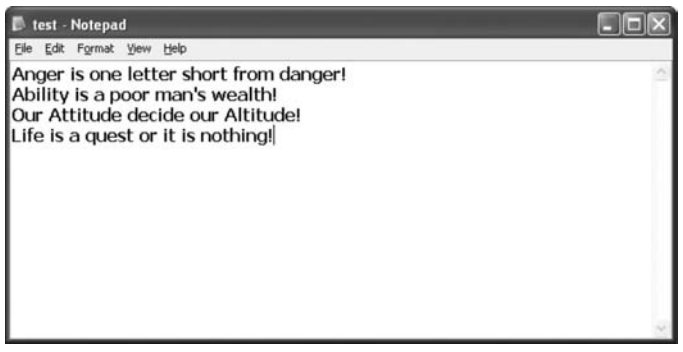
How to Write a Function wcc() to Find the Word Count of a File

```
int wcc(char *filename)
{
      int count = 0;
      FILE *fp;
      fp = fopen(filename,"r");
      while(!feof(fp))
      {
            fgetc(fp);
            count++;
      }
      fclose(fp);
      return count++;

}
```

How to Write a Function Client Code that Uses These Three Functions

```
int main()
{
      printf("Lines = %d  Word = %d  Characters =

%d\n",wcl("C:\\test.txt"),wcw("C:\\test.txt"),wcc("C:
\\test.txt"));
      getch();
      return 0;
}
```

The contents of the file C:\\test.txt was this



When the client program is run, we get the following output

```
Lines = 4  Word = 26  Characters = 139
```

How to Simulate UNIX head Command

This function prints the first **number** of lines passed to it as a parameter.

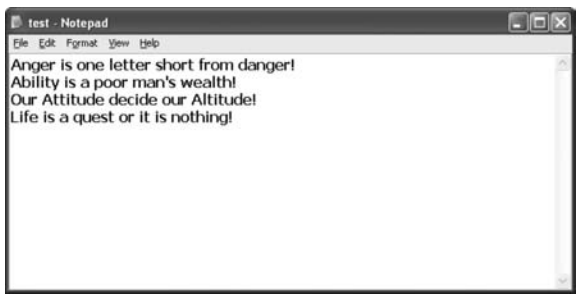
```
void head(char *filename,int number)
{
      int count = 0;
      char line[81];
      FILE *fp;
      fp = fopen(filename,"r");
      while(!feof(fp))
      {
            fgets(line,81,fp);
            count++;
            puts(line);
            if(count==number)
                  break;
      }
      fclose(fp);
}
```

Here is a client code to call the function.

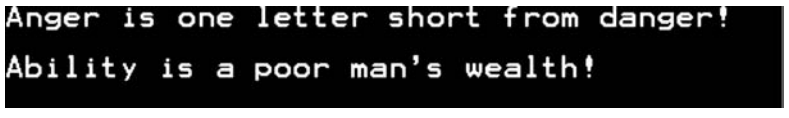
```
int main()
{
```

```
	head("C:\\test.txt",2);
	getch();
	return 0;
}
```

test.txt contains these lines



And the output of the client code is



which are the first two lines of the text file test.txt.

How to Simulate UNIX tail Command

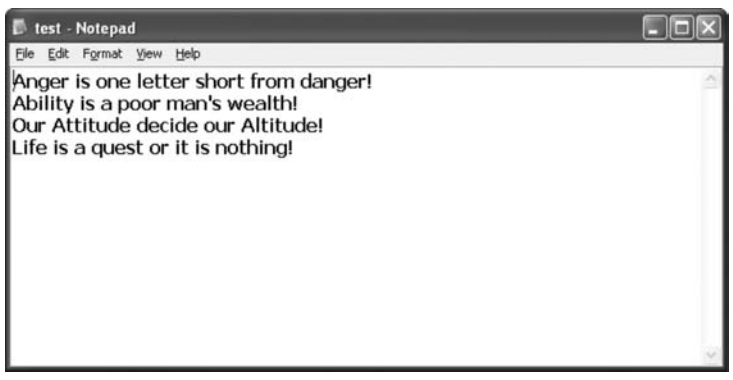
```
void tail(char *filename,int number)
{
	int count = 0;
	char line[81];
	//See how the wcl function is being used to calculate the
	//number of lines in the file. Whenever writing a new function
	//we should keep in mind that the function can be used as a
	//building block.
	int totalnumberoflines = wcl(filename);
	int startingnumber = totalnumberoflines - number + 1;
	FILE *fp;
	fp = fopen(filename,"r");
	while(!feof(fp))
	{
		fgets(line,81,fp);
			count++;
		if(count>=startingnumber)
			puts(line);
	}
	fclose(fp);

}
```

Notice the bolded part that shows how the wcl() function is being used to calculate the total number of lines in the file which is the central part of the tail() function.

Here is a client code to test the tail() function

```
int main()
{
	tail("C:\\test.txt",2);
	getch();
	return 0;
}
```

The content of test.txt was the following lines



And the output of the above client code is

```
Our Attitude decide our Altitude!
Life is a quest or it is nothing!
```

which are the last 2 lines of the file test.txt.

16.3 HOW TO SIMULATE UNIX cat COMMAND

This function appends the contents of the source file at the end of the destination file.

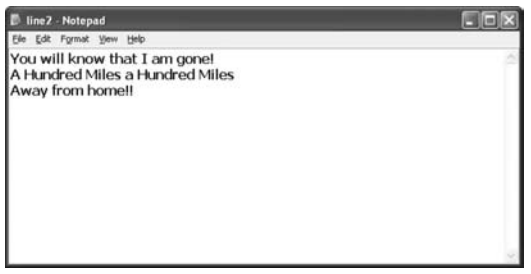
void cat(char *destinationfile,char *sourcefile)

```
void cat(char *destinationfile,char *sourcefile)
{
	FILE *fp1 = fopen(destinationfile,"a");
	FILE *fp2 = fopen(sourcefile,"r");
	fputc('\n',fp1);
	while(!feof(fp2))
	{
		fputc(fgetc(fp2),fp1);
	}
	fclose(fp1);
	fclose(fp2);
}
```

There are two files line1.txt and line2.txt whose contents are as follows.

Line1.txt

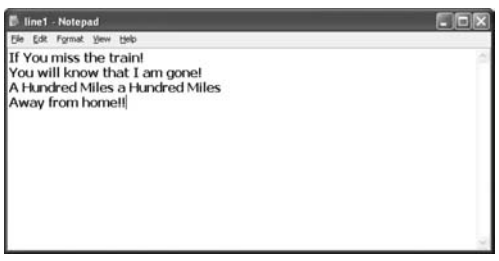
Line2.txt



Here is a client code that concatenates these two files.

```
int main()
{
      cat("C:\\line1.txt","C:\\line2.txt");
      getch();
      return 0;
}
```

and here is the content of line1.txt after this main() is executed.



Try Yourself: *Write a wrapper function that will use this* ***cat****() function to concatenate more file contents at the end of one target file. The signature of your wrapper function should be like* `cat(char *targetfilename, String *files)`

The second argument is a linked list of filenames that represents the source files.

16.4 HOW TO SIMULATE UNIX grep COMMAND WITH EXACT MATCH

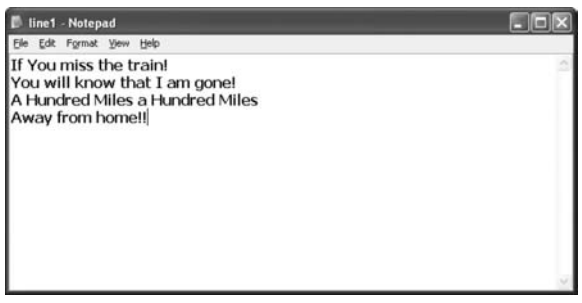
This function returns 1 if the pattern is a substring of the string.

```
int DoesItExist(char *string,char *pattern)
{
      //strstr() returns the pointer that marks the start of pattern
      //in string. So if pattern exist as a substring in string then
      //strstr(string,pattern) will not return a NULL pointer.
      return strstr(string,pattern)!=NULL?1:0;
}
```

This function prints all the lines of the file where searchstring is found to be part of the line.

```
void grep(char *filename,char *searchstring)
{
      char line[81];
FILE *fp = fopen(filename,"r");
while(!feof(fp))
{
      fgets(line,81,fp);
            //Is the sought string a substring of
            //the line read?
```

```
            if(DoesItExist(line,searchstring))
            puts(line);
}
      fclose(fp);
}
```



Here is a client code to test this function.

```
int main()
{
      grepv("C:\\line1.txt","You");
      getch();
      return 0;
}
```

The content of the file line 1.txt

The demo program will print only those lines where the word "You" is present. The output of the program is shown below.

```
If You miss the train!
You will know that I am gone!
```

16.5 HOW TO SIMULATE UNIX Grep COMMAND FOR SWITCH –*V*

This function searches for all those lines in the text file that doesn't have the search string as part of it. This is basically the complimentary version of grep.

```
void grepv(char *filename,char *searchstring)
{
      char line[81];
      FILE *fp = fopen(filename,"r");
      while(!feof(fp))
      {
            fgets(line,81,fp);
                  if(!DoesItExist(line,searchstring))
                  puts(line);
      }

      fclose(fp);
}
```

Here is a client code that calls this function.

```
int main()
{
      grepv("C:\\line1.txt","You");
      getch();
      return 0;

}
```

The content of line1.txt is

The above client code is written to get those lines where "You" is not present. The output of the program is shown below.

```
A Hundred Miles a Hundred Miles
Away from home!!
```

How to Print Those Lines of a Text File that Contain a Word Starting with a Given Prefix

```
void greps(char *filename,char *searchstring)
{
      char word[20];
      char line[81];
      char cline[81];

      int found = 0;
      FILE *fp = fopen(filename,"r");
      while(!feof(fp))
      {
            found = 0;
            String *h=NULL;
            strcpy(line,"");
            strcpy(cline,"");
            fgets(line,81,fp);
            //making a copy of the line.
            //because at the end of the split
            //the line variable will be lost.
            strcpy(cline,line);
            h = split(line);
            //h now contains all the words in line.
            for(;h!=NULL;h=h->next)
            {
                  //checking whether a word starts with the searchstring
                  if(startsWith(h->s,searchstring)==1)
                  {
                        found = 1;
                        break;
                  }
            }
            if(found==1)
                  puts(cline);
      }
      fclose(fp);
}
```

How to Print Those Lines of a Text File that Contain a Word Ending with a given Suffix

```
void grepe(char *filename,char *searchstring)
```

```
{
      char word[20];
      char line[81];
      char cline[81];

      int found = 0;
      FILE *fp = fopen(filename,"r");
      while(!feof(fp))
      {
            found = 0;
            String *h=NULL;
            strcpy(line,"");
            strcpy(cline,"");
            fgets(line,81,fp);
            //Holding the copy of the line
            //in another container. Otherwise when
            //we copy the next
            strcpy(cline,line);
            h = split(line);//Refer the split function in string chapter
            for(;h!=NULL;h=h->next)
            {
                  //Refer the endsWith function in the string chapter
                  if(endsWith(h->s,searchstring)==1)
                  {
                        found = 1;
                        break;
                  }
            }
            if(found==1)
                  puts(cline);
      }

      fclose(fp);
}
```

How to Print Those Lines of a Text File that do not Contain a Word Starting with a Given Prefix

```
void grepsv(char *filename,char *searchstring)
{
      char word[20];
      char line[81];
      char cline[81];
      int found = 0;
      FILE *fp = fopen(filename,"r");
      while(!feof(fp))
      {
            found = 0;
            String *h=NULL;
            strcpy(line,"");
            strcpy(cline,"");
```

```
fgets(line,81,fp);
strcpy(cline,line);
h = split(line);
for(;h!=NULL;h=h->next)
{
      if(startsWith(h->s,searchstring)==1)
      {
            found = 1;
            break;
      }
}
if(found==0)
      puts(cline);
}

fclose(fp);
}
```

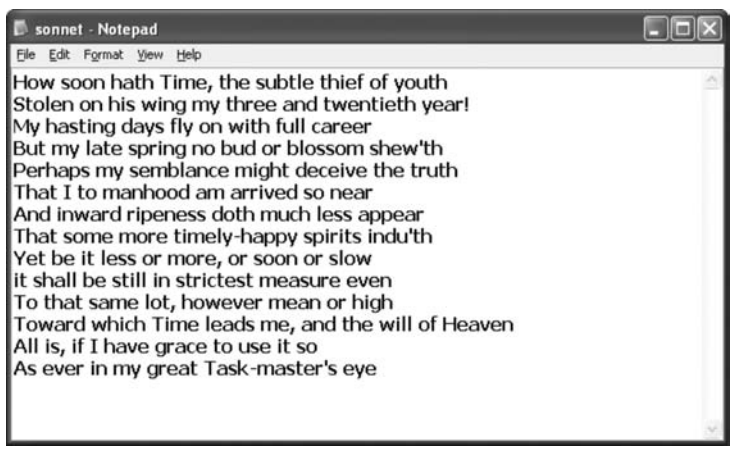
How to Print Those Lines of a Text File That do not Contain a Word Ending with a Given Suffix

```
void grepev(char *filename,char *searchstring)
{
      char word[20];
      char line[81];
      char cline[81];

      int found = 0;
      FILE *fp = fopen(filename,"r");
      while(!feof(fp))
      {
            found = 0;
            String *h=NULL;
            strcpy(line,"");
            strcpy(cline,"");
            fgets(line,81,fp);
            strcpy(cline,line);
            h = split(line);
            for(;h!=NULL;h=h->next)
            {
                  if(endsWith(h->s,searchstring)==1)
                  {
                        found = 1;
                        break;
                  }
            }
            if(found==0)//Notice the change
                  puts(cline);
      }
      fclose(fp);
}
```

How to Print Those Lines of a File That Contain a Word or a Phrase that Matches a Particular Pattern Represented by Asterisk Wildcard Notation

```
void wildgrep(char *filename,char *searchstring)
{
      char line[50];
      char cline[50];
      String *toks = NULL;
      FILE *fp = fopen(filename,"r");
      while(!feof(fp))
      {
            toks = NULL;
            fgets(line,50,fp);
            strcpy(cline,line);
            //Notice how these string functions are used heavily here
            toks = split(line," ");
            for(;toks!=NULL;toks=toks->next)
            {
                  if(wildCharMatch(toks->s,searchstring)==1)
                        puts(cline);
            }

      }

      fclose(fp);
}
```

The sonnet.txt file content is as shown below.



When called by the following client code,

```
int main()
{
      wildgrep("C:\\sonnet.txt","*am*");
      getch();
      return 0;
}
```

the above function gives the following output

because the pattern *am* matches with “am” and “same”.

16.6 HOW TO PRINT THOSE LINES OF A FILE THAT CONTAIN A WORD THAT SOUNDS LIKE A GIVEN WORD

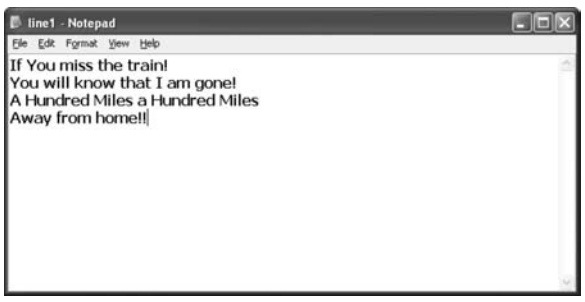
```
void wildgrep(char *filename,char *searchstring)
{
      char line[50];
      char cline[50];
      String *toks = NULL;
      FILE *fp = fopen(filename,"r");
      while(!feof(fp))
      {
            toks = NULL;
            fgets(line,50,fp);
            strcpy(cline,line);
            //Notice how these string functions are used heavily here
            toks = split(line," ");
            for(;toks!=NULL;toks=toks->next)
            {
                  //refer chapter on string for isSameSoundex() function.
                  if(isSameSoundex(toks->s,searchstring)==1)
                        puts(cline);
            }

      }

      fclose(fp);
}
```

16.7 HOW TO REPLACE A CHARACTER IN A FILE WITH ANOTHER CHARACTER

The strategy is to involve a temporary file that will hold the new character version of the old file. After the temporary file is created, we just rename it to the original file and delete the original file.

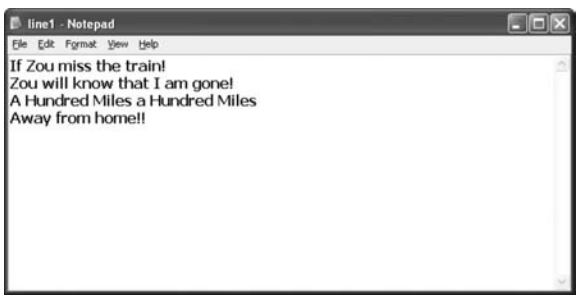
```
void trc(char *filename,char oldchar,char newchar)
{
      char c;
      char temp[20];
      FILE *fp = fopen(filename,"r");
      FILE *fw = tmpfile();
      tmpnam(temp);
      fw = fopen(temp,"a");
      while(!feof(fp))
```

```
{
      c = fgetc(fp);
      if(c==oldchar)
            fputc(newchar,fw);
      else
            fputc(c,fw);
}
fclose(fp);
fclose(fw);
unlink(filename);
rename(temp,filename);
}
```



Before the call to this function the contents of line1.txt was as given in the first figure
After the call to tr() as in the client code below

```
int main()
{
      trc("C:\\line1.txt",'Y','Z');
      getch();
      return 0;
}
```

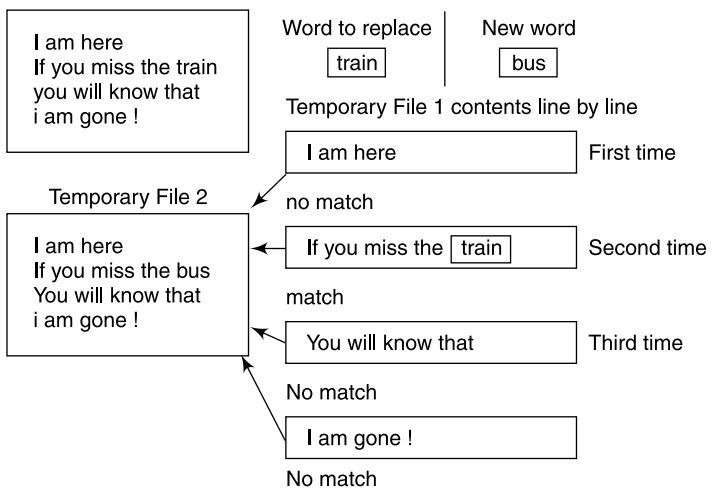


the content of line1.txt becomes as shown alongside,

16.8 HOW TO REPLACE A WORD IN A FILE WITH ANOTHER WORD

This function replaces a word in a text file with another given word. The program uses two temporary files for this purpose. The function works as follows.

Lines are read one by one from the source file where the word needs to be replaced. The lines are then written to a temporary file one by one and the words are read from these lines. If any of the word read from these words match with the word to be replaced, then the new word is written in another temporary file. Otherwise the words from these lines are written. At the end the second temporary file is renamed by the original file's name and the original file is deleted. Thus we get a new file where the word is replaced with another one.




```
void trw(char *filename,char* oldword,char *newword)
{
      char word[20];
```

```
char temp[20];
char temp2[20];
char line[80];
FILE *fp = fopen(filename,"r");
FILE *fw = tmpfile();
FILE *fw2 = tmpfile();
tmpnam(temp);
tmpnam(temp2);
//Creating the first temporary file
fw = fopen(temp,"a");

while(!feof(fp))
{
      //Creating the second temporary file
      fw2 = fopen(temp2,"w");
      fgets(line,81,fp);
      fprintf(fw2,"%s",line);
      fclose(fw2);
      fw2 = fopen(temp2,"r");
      while(!feof(fw2))
      {
            fscanf(fw2,"%s ",word);
            if(strcmpi(word,oldword)==0)
                  fprintf(fw,"%s ",newword);
            else
                  fprintf(fw,"%s ",word);
      }
      fprintf(fw,"\n");
      fclose(fw2);
}
fclose(fp);
fclose(fw);
unlink(filename);
rename(temp,filename);

}
```

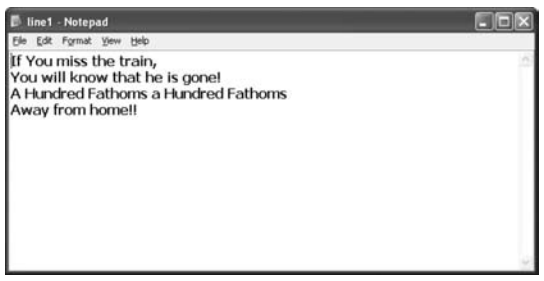
16.9 HOW TO COMPARE TWO TEXT FILES LINE BY LINE

This function prints all those lines where two files are different:

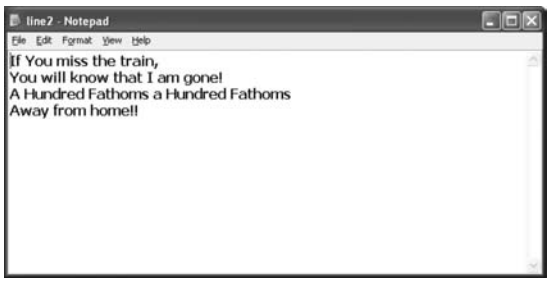
```
void diff(char *file_1,char *file_2)
{
      char linef1[81];
      char linef2[81];
      //Opening first file
      FILE *fp1  = fopen(file_1,"r");
      //Opening the second file
      FILE *fp2  = fopen(file_2,"r");
      while(!feof(fp1))
      {
            fgets(linef1,81,fp1);//reading line from the first file
```

```
            fgets(linef2,81,fp2);//reading line from the second file
            if(strcmpi(linef1,linef2)!=0)//If they are different
                  printf("%s # %s\n",linef1,linef2);

      }
}
```

Contents of line1.txt is



Contents of line2.txt is



When called by the client code below

```
int main()
{
      diff("C:\\line1.txt","C:\\line2.txt");
      getch();
      return 0;
}
```

we get the below output.

```
You will know that he is gone!
 # You will know that I am gone!
```

16.10 HOW TO PRINT SAME LINES OF TWO FILES

This function will print only those lines that are same in both the files. This is just the complement of the diff function. Note the naming convention of complement functions. diff() and diffv() differs by only a 'v' at the end which is the short form of negative because diffv() = !diff():

```
void diffv(char *file_1,char *file_2)
{
      char linef1[81];
      char linef2[81];
      FILE *fp1  = fopen(file_1,"r");
      FILE *fp2  = fopen(file_2,"r");
      while(!feof(fp1))
      {
            fgets(linef1,81,fp1);//reading the line from file 1
            fgets(linef2,81,fp2);//reading the line from file 2
            if(strcmpi(linef1,linef2)==0)//Are the lines same?
                  printf("%s = %s\n",linef1,linef2);

      }
}
```

16.11 HOW TO COPY A FILE FROM A SOURCE TO A DESTINATION

This function copies the content of the sourcefile to the target file.

```
void cp(char *sourcefile,char *targetfile)
{
      FILE *sfp = fopen(sourcefile,"r");
      FILE *tfp = fopen(targetfile,"w");
      while(!feof(sfp))
      {
            //reading a character from source file
            //and putting that in the target file
            fputc(fgetc(sfp),tfp);
      }
      fclose(sfp);
      fclose(tfp);
}
```

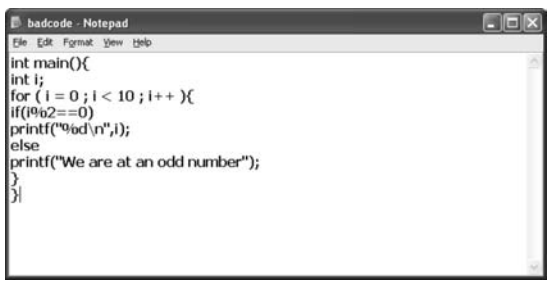
Example 16.5 ***Write a program to autocorrect the indentation of a badly written C code.***

Solution Programs that correct the indentation of a program are known as *Code–Beautifier*.

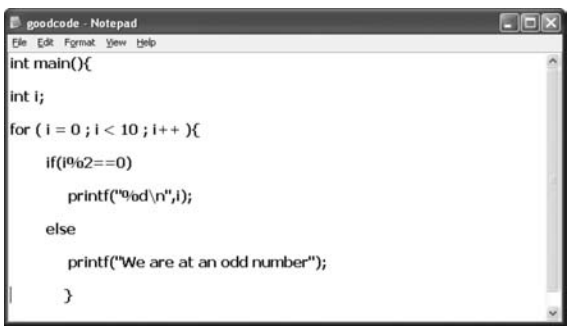
```
typedef struct Line
{
      char s[50];
      int spaceatfront;
      struct Line *next;
      struct Line *prev;
}Line;


Line* push_back_Line(Line *last,char *s,int space)
{
      if(last==NULL)
      {
            last = (Line *)malloc(sizeof(Line));
            strcpy(last->s,s);
            last->spaceatfront = space;
            last->next = NULL;
            last->prev = NULL;
            return last;
      }
      else
      {
            Line *p = (Line *)malloc(sizeof(Line));
            last->next = p;
            p->spaceatfront = space;
            p->prev = last;

            strcpy(p->s,s);
            p->next = NULL;
```

```
            return p;
      }

}

int AssignSpace(Line *l)
{
      int count = 0;
      l = l->prev;
      for(;l!=NULL;l=l->prev)
            if(startsWith(l->s,"if")==1
             ||startsWith(l->s,"for")==1
             ||startsWith(l->s,"{")==1)
                  count+=l->spaceatfront+4;
      return count;
}

int AssignSpaceForElse(Line *l)
{

      l = l->prev;
      for(;l!=NULL;l=l->prev)
            if(startsWith( l->s ,"if")==1)
                  break;
      return l->spaceatfront;
}

int main(int argc,char *argv[])
{

      char line[50];
      FILE *fp = fopen("C:\\badcode.txt","r");
      FILE *fg = fopen("C:\\goodcode.txt","w");

      int count = 0;
      int i = 0;
      Line *h=NULL,*hc=NULL,*hc2=NULL;
      while(!feof(fp))
      {
            fgets(line,50,fp);
            h = push_back_Line(h,line,0);
            count++;
            if(count==1)
                  hc = h;
      }
hc2= hc;
fclose(fp);

for(;hc->next!=NULL;hc=hc->next)
{
      if(startsWith(hc->s,"if")==1 || startsWith(hc->s,"for")==1)
```

```
{
      hc->spaceatfront = AssignSpace(hc);

      for(i=0;i<hc->spaceatfront;i++)
            fprintf(fg,"%c",' ');
}
if(startsWith(hc->s,"else")==1)
{
      hc->spaceatfront = AssignSpaceForElse(hc);

      for(i=0;i<hc->spaceatfront;i++)
            fprintf(fg,"%c",' ');
   }
   else
   {
      if(hc->prev!=NULL)
      {
        hc->spaceatfront = AssignSpace(hc);

      for(i=0;i<hc->spaceatfront;i++)
            fprintf(fg,"%c",' ');
      }
   }
   fprintf(fg,"%s\n",hc->s);
}

fclose(fg);
getch();

return 0;
}
```



Here is the badly indented code on which this indenter was run.
After the program is run, the generated good indented code looks as below.



Example 16.6 ***Write a program to find out whether a C++ program uses STL or not?***

Solution STL is the acronym for Standard Template Library. This library holds some generic data structures and algorithm. A student might want to scan through the c/c++ codes that he has on his hard

drive for the programs written with STL. The following functions scans a program file and returns 1 if the program uses STL else it returns 0.

```
int isSTL(char *filename)
{
      FILE *fp = fopen(filename,"r");
      char word[20];
      while(!feof(fp))
      {
            fscanf(fp,"%s",word);
            //If any of the listed header is found in the
            //program listing then the program uses STL
            if(strcmp(word,"<list>")==0
            || strcmp(word,"<vector>")==0
            || strcmp(word,"<map>")==0
            || strcmp(word,"<set>")==0
            || strcmp(word,"<algorithms>")==0
            || strcmp(word,"<queue>")==0
            || strcmp(word,"<stack>")==0
            || strcmp(word,"<deque>")==0)
                  return 1;
      }
      fclose(fp);
}

int main()
{
      printf("STL Indicator = %d\n",isSTL("C:\\prog.txt"));
      getch();
      return 0;
}
```

Example 16.7 ***Write a program to find out whether a console based program is written in C/C++, Java, C# or Visual Basic.***

Solution To identify whether a program is written in C,C++,Java or Visual Basic it will be enough and sufficient to look at the main() function/sub-signature lines.

```
int isCorCPP(char *filename)
{
      int isC = 0;
      FILE *fp = fopen(filename,"r");
      char line[81];
      while(!feof(fp))
      {
            fgets(line,81,fp);
            //These are the forms of legal main() functions in C or C++
            if(startsWith(line,"main")==0
                || strcmp(line,"int main()")==0
                  || strcmp(line,"int main(void)")==0
                  || strcmp(line,"void main()")==0
                  || strcmp(line,"void main(void)")==0
```

```
            || strcmp(line,"void main(int argc,char *argv[])")==0
            || strcmp(line,"int main(int argc,char *argv[])")==0
            || strcmp(line,"int main(void){")==0
            || strcmp(line,"void main(){")==0
            || strcmp(line,"void main(void){")==0
            || strcmp(line,"void main(int argc,char *argv[]){")==0
            || strcmp(line,"int main(int argc,char *argv[]){")==0)
{
                  isC = 1;
                  break;
            }
      }
      fclose(fp);
      return isC;
}



int isJava(char *filename)
{

      int isJ = 0;
      FILE *fp = fopen(filename,"r");
      char line[81];
      while(!feof(fp))
      {
            fgets(line,81,fp);
            //This is the only legal form of main() in Java
            if(startsWith(line,"public static void main")==1)
            {
                  isJ = 1;
                  break;
            }
      }
      fclose(fp);
      return isJ;
}

int isCSharp(char *filename)
{

      int isCS = 0;
      FILE *fp = fopen(filename,"r");
      char line[81];
      while(!feof(fp))
      {
            fgets(line,81,fp);
            //This is the only legal form of Main() in C#
            if(startsWith(line,"static void Main")==1)
            {
                  isCS = 1;
```

```
                  break;
            }
      }

      fclose(fp);
      return isCS;
}

int isVB(char *filename)
{

      int VB = 0;
      FILE *fp = fopen(filename,"r");
      char line[81];
      while(!feof(fp))
      {
            fgets(line,81,fp);
            //This is the only legal form of Main() in VB
            if(startsWith(line,"Sub Main()")==1)
            {
                  VB = 1;
                  break;
            }
      }

      fclose(fp);
      return VB;
}
```

Example 16.8 ***Write a program that will read a notepad file which will contain the definition of an entity like Bank Account, Student, etc., and then generate the code for creating a single linked list of this type. The program should also write the structure definitions.***

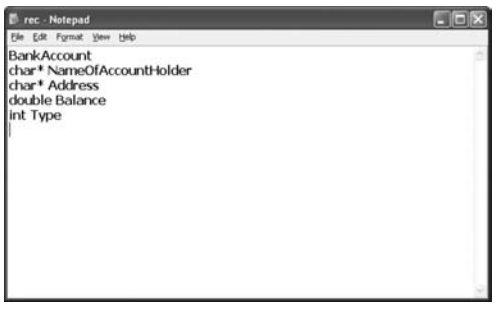
Solution

```
typedef struct Entry
{
      char datatype[10];
      char variablename[256];
      struct Entry *next;
}Entry;

Entry* push_back_Entry(Entry *last,Entry e)
{
      if(last==NULL)
      {
            last = (Entry *)malloc(sizeof(Entry));
            strcpy(last->datatype ,e.datatype);
            strcpy(last->variablename , e.variablename);
            last->next = NULL;
            return last;
      }
```

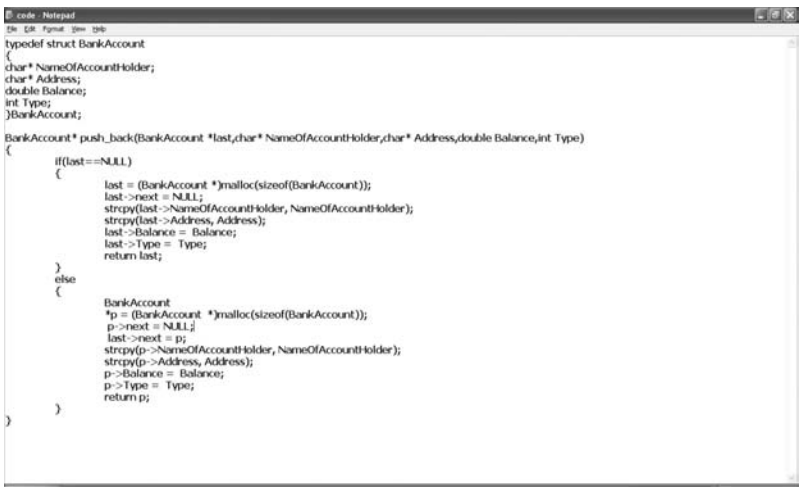
```
else
{
      Entry *p = (Entry *)malloc(sizeof(Entry));
      last->next = p;
      p->next = NULL;
      strcpy(p->datatype ,e.datatype) ;
      strcpy(p->variablename , e.variablename) ;
      return p;
      }
}

      int main()
      {
          char line[50];
          int count=0;
            char rec[20];
            Entry *ens = NULL,e;
            Entry *cens = NULL,*cens2=NULL,*cens3=NULL,*cens4 = NULL;
            String *h = NULL;
            String *hc = NULL;
            String *toks = NULL;
             FILE *fp = fopen("C:\\rec.txt","r");
            FILE *fw = fopen("C:\\code.txt","w");
            while(!feof(fp))
            {
                  fgets(line,50,fp);
                  h = push_back_String(h,line);
                  count++;
                  if(count == 1)
                        hc = h;
            }
            fclose(fp);
            count = 0;
            strcpy(rec,hc->s);
            hc = hc->next;
            for(;hc!=NULL;hc=hc->next)
            {
                  toks = split(hc->s," ");
                  strcpy(e.datatype , toks->s);
                  strcpy(e.variablename , toks->next->s);
                  ens = push_back_Entry(ens,e);
                  count++;
                  if(count == 1)
                        cens = ens;
            }
            cens2 = cens;
            cens3 = cens;
            cens4 = cens;

            fprintf(fw,"typedef struct %s",rec);
            fprintf(fw,"\n{\n");
            for(;cens4!=NULL;cens4=cens4->next)
                  fprintf(fw,"%s %s;\n",cens4->datatype
                   ,cens4->variablename);
            fprintf(fw,"\n}");
```

```
        fprintf(fw,"%s;\n\n\n\n",rec);
        //push_back generation
        fprintf(fw,"%s* push_back(%s *last",rec,rec);
        for(;cens->next!=NULL;cens=cens->next)
              fprintf(fw,",%s %s",cens->datatype,cens->variablename);
        fprintf(fw,",%s %s)\n",cens->datatype,cens->variablename);
        fprintf(fw,"{\n");
        fprintf(fw,"\tif(last==NULL)");
        fprintf(fw,"\n\t{");
        fprintf(fw,"\n\t\tlast = (%s *)malloc(sizeof(%s));",rec,rec);
        fprintf(fw,"\n\t\tlast->next = NULL;");
        for(;cens2!=NULL;cens2=cens2->next)
        {
              if(strstr(cens2->datatype,"char *")!=NULL ||
        strstr(cens2->datatype,"char*")!=NULL)
                    fprintf(fw,"\n\t\tstrcpy(last->%s, %s);"
 ,cens2->variablename,cens2->variablename);
              else
                    fprintf(fw,"\n\t\tlast->%s =  %s;"
,cens2->variablename,cens2->variablename);
        }
        fprintf(fw,"\n\t\treturn last;");
        fprintf(fw,"\n\t}");
        fprintf(fw,"else");
        fprintf(fw,"\n{");
        fprintf(fw,"\n\t%s *p = (%s *)malloc(sizeof(%s));",rec,rec,rec);
        fprintf(fw,"\n\t p->next = NULL;");
        fprintf(fw,"\n\t last->next = p;");

        for(;cens3!=NULL;cens3=cens3->next)
        {
              if(strstr(cens3->datatype,"char *")!=NULL
 || strstr(cens3->datatype,"char*")!=NULL)
                    fprintf(fw,"\n\t\tstrcpy(p->%s, %s);"
,cens3->variablename,cens3->variablename);
              else
                    fprintf(fw,"\n\t\tp->%s =  %s;",cens3-
>variablename,cens3->variablename);
        }
        fprintf(fw,"\n\t\treturn p;");
        fprintf(fw,"\n\t}");
        fprintf(fw,"\n}");
        fclose(fw);
        getch();
        return 0;
}
```

The above program takes the following notepad as input.



and it generates the following code as output:



```
typedef struct BankAccount
{
char* NameOfAccountHolder;
char* Address;
double Balance;
int Type;
}BankAccount;

BankAccount* push_back(BankAccount *last,char* NameOfAccountHolder,char* Address,double Balance,int Type)
{
        if(last==NULL)
        {
        last = (BankAccount *)malloc(sizeof(BankAccount));
        last->next = NULL;
        strcpy(last->NameOfAccountHolder, NameOfAccountHolder);
        strcpy(last->Address, Address);
        last->Balance = Balance;
        last->Type = Type;
        return last;
        }
        else
        {
                BankAccount
                *p = (BankAccount *)malloc(sizeof(BankAccount));
                p->next = NULL;
                last->next = p;
                strcpy(p->NameOfAccountHolder, NameOfAccountHolder);
                strcpy(p->Address, Address);
                p->Balance = Balance;
                p->Type = Type;
                return p;
        }
}
```

The beauty of this program is that we can make it read any record and it will create a flawless push_back() method for that particular type. This saves a lot of coding effort.

Example 16.9 ***Write a program to check whether content of a file is a sonnet or not.***

Solution A sonnet is a song of 14 or more lines where alternate lines end with a rythmic word. Here is a Perfect Sonnet (*Def: Total number of matching pair or lines is half that of the total lines of the poem*) by Horatio R Palmer.

Angry words! O let them never
From the tongue unbridled slip
May the heart's best impulse ever
Check them ere they soil the lip
Love one another thus saith the Savior
Children obey the Father's blest command
Love each other, love each other
'Tis the Father's blest command
Love is much too pure and holy
Friendship is too sacred far
For a moment's reckless folly
Thus to desolate and mar
Angry words are lightly spoken
Bitterest thoughts are rashly stirred
Brightest links of life are broken
By a single angry word

This function extracts a substring from the given string, starting from index start till the index reaches end.

```
char* substring(char *string,int start,int end)
{
      char temp[30];
      int i,j;
      for(i=start,j=0;i<=end;i++,j++)
            temp[j] = string[i];
      temp[j] = '\0';
      return temp;

}
```

This function returns 1 if two lines passed to it can be identified by two matching sonnet lines. For example from the above sonnet, "*Angry words are lightly spoken" **and** "Brightest links of life are broken"* are two matching sonnet lines.

```
int isSonnetLines(char *a,char *b)
{
      if(strcmp(a,b)==0)
            return 1;
      if(strcmp(substring(a,strlen(a)-3,strlen(a)-1)
       ,substring(b,strlen(b)-3,strlen(b)-1))==0)
            return 1;

      if(strcmp(substring(a,strlen(a)-2
        ,strlen(a)-1),substring(b,strlen(b)-2,strlen(b)-1))==0)
            return 1;

      if(a[strlen(a)-1]==b[strlen(b)-1])
            return 1;
      else
            return 0;
}
```

There are many styles of sonnet writing. The above example was written by following one such style. The client code below uses the above two functions. It reads a text file and displays a message reporting whether the text file content is a sonnet or not. If you notice carefully the above sonnet contains 16 lines and 8 matching pair of sonnet lines. This has been captured in the following client code.

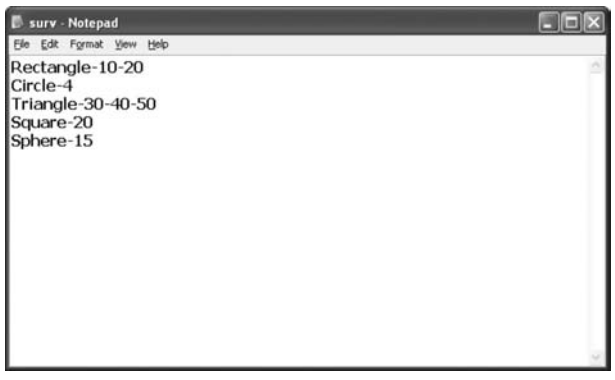
```
int main()
{
      String* Lines = NULL,*CLines = NULL,*CLines2=NULL;
      int count = 0;
      int total = 0;
      char line[81];
      FILE *fp = fopen("C:\\sonnet.txt","r");
      while(!feof(fp))
      {
            fgets(line,81,fp);
            Lines = push_back_String(Lines,line);
            count++;
           if(count==1)
                 CLines = Lines;
      }
      fclose(fp);
      CLines2 = CLines;
      count=0;
      //Reading the total number of lines in the Sonnet
      for(;CLines2!=NULL;CLines2=CLines2->next)
            total++;
      //Finding matching sonnet line pairs
      for(;CLines->next->next!=NULL;CLines=CLines->next)
      {
            if(isSonnetLines(CLines->s,CLines->next->next->s)==1)
                  count++;
      }

      printf("Count = %d Total = %d\n",count,total);
```

```
//Sonnets are always of even number of lines. So there will be half
//the number of lines matching sonnet lines. So for the perfect
//sonnet like the above one count will always be half the total
//number of lines. There are different styles of sonnets that might
//have couple of lines of mismatch from a perfect sonnet style. So
//the following logic accomodates that.
if(count==total/2 || count == total/2 -1 || count == total/2-2)
      puts("It is a Sonnet.");
      else
            puts("It is not a Sonnet.");

      getch();
      return 0;
}
```

Try Yourself: *The Italian Style Sonnets don't match the pattern taken as the example here. Write a program to check whether a Sonnet is an Italian Style Sonnet or not.*

File handling I/O function details: Please refer Online Learning Center Slides for more Theoretical details on the file handling I/O functions.

Example 16.10 ***Write a program that reads a notepad file that contains geometrical measures of different shapes in a delimited format.***



Solution The program will calculate the area, perimeter and volume of the shapes as and when applicable and list them on the console. The program should be able to handle Rectangle, Right Angled Triangle, Squares and Spheres. Here is a sample input file to the program.

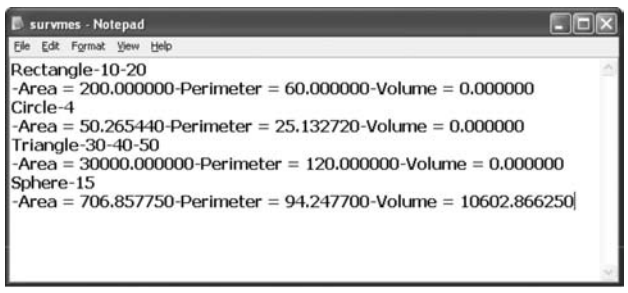
```
typedef struct Shape
{
      //This will store description like Square-20
      //(This means a square of length 20)
      char Description[50];
      double Area;
      double Perimeter;
      double Volume;//If applicable, I mean if the shape is 3D
      struct Shape *next;
}Shape;

//This puts a Shape at the end of a Shape Linked List
Shape* push_back_Shape(Shape *last,Shape s)
{
if(last==NULL)
{
      last = (Shape *)malloc(sizeof(Shape));
      last->next = NULL;
      strcpy(last->Description , s.Description );
      last->Area = s.Area;
      last->Perimeter = s.Perimeter ;
      last->Volume = s.Volume;
```

```
      return last;
}

else
{
      Shape *p = (Shape *)malloc(sizeof(Shape));
      p->next = NULL;
      last->next = p;
            strcpy(p->Description,s.Description);
            p->Area = s.Area;
            p->Perimeter = s.Perimeter;
            p->Volume = s.Volume ;
            return p;
      }
}
int main()
{
      int count = 0;
      int x,y,z;
      String *l=NULL,*cl = NULL;
      String *toks = NULL;
      Shape s;
      Shape *sl = NULL,*csl = NULL;
      FILE *fp = fopen("C:\\surv.txt","r");
      char line[30];
      while(!feof(fp))
      {
            fgets(line,81,fp);
            l = push_back_String(l,line);
            count++;
            if(count==1)
                  cl = l;
      }
      fclose(fp);
      count = 0;
      for(;cl!=NULL;cl=cl->next)
      {
            x=0;
            y=0;
            z=0;
            strcpy(s.Description,cl->s);
            //Is it a Rectangle?
            if(startsWith(cl->s,"Rec")==1)
            {
                  toks = split(cl->s,"-");
                  x = atof(toks->next->s);
y = atof(toks->next->next->s);

s.Area = x*y;
s.Volume = 0;
s.Perimeter = 2*(x+y);
sl = push_back_Shape(sl,s);
count++;
```

```
	if(count==1)
		csl = sl;
}

//Is it a Triangle?
if(startsWith(cl->s,"Tri")==1)
{
	toks = split(cl->s,"-");
	x = atof(toks->next->s);
	y = atof(toks->next->next->s);
	z = atof(toks->next->next->next->s);
	//Assuming Right Angled Triangles
	s.Area = 0.5 * x * y * z;
	s.Volume = 0;
	s.Perimeter = x + y + z;
	sl = push_back_Shape(sl,s);
	count++;
	if(count==1)
		csl = sl;
}

//Is it a Circle?
if(startsWith(cl->s,"Cir")==1)
{
	toks = split(cl->s,"-");
	x = atof(toks->next->s);
	//Assuming Right Angled Triangles
	s.Area = 3.14159 * x * x;
	s.Volume = 0;
	s.Perimeter = 2 * 3.14159 * x;//integer estimate
	sl = push_back_Shape(sl,s);
	count++;
	if(count==1)
		csl = sl;
}
//Is it a sphere?
if(startsWith(cl->s,"Sph")==1)
{
	toks = split(cl->s,"-");
	x = atof(toks->next->s);
	//Assuming Right Angled Triangles
	s.Area = 4/3 * 3.14159 * x * x;
	s.Volume = 4/3 * 3.14159 * x * x * x;
	s.Perimeter = 2* 3.14159 * x;
	sl = push_back_Shape(sl,s);
	count++;
	if(count==1)
		csl = sl;
	}


}
```

```
fp = fopen("C:\\survmes.txt","w");
      for(;csl!=NULL;csl=csl->next)
            fprintf(fp,"%s-%f-%f-%f\n"
             ,csl->Description
             ,csl->Area
            ,csl->Perimeter
            ,csl->Volume);
      fclose(fp);
            puts("Done");

      getch();
      return 0;
}
```

The above program generates the following output.



The surveyor can just collect the information and use this above program to generate this '-' delimited notepad file which can be exported to excel.

Here is the Function That Finds the Shape That Has the Maximum Area

```
Shape* MaxArea(Shape *shapes)
{
      Shape *cs = shapes,*ccs = shapes;
      double maxArea = cs->Area;
      for(;cs!=NULL;cs=cs->next)
      {
            if(cs->Area>maxArea)
                  maxArea = cs->Area;
      }

      for(;ccs!=NULL;ccs=ccs->next)
            if(ccs->Area==maxArea)
                  return ccs;
}
```

Here is the Function That Finds the Shape That has the Minimum Area

```
Shape* MinArea(Shape *shapes)
{
      Shape *cs = shapes,*ccs = shapes;
      double minArea = cs->Area;
      for(;cs!=NULL;cs=cs->next)
      {
            if(cs->Area<minArea)
                  minArea = cs->Area;
      `
```

```
	}

	for(;ccs!=NULL;ccs=ccs->next)
		if(ccs->Area==minArea)
			return ccs;

}
```

Here is the Function That Finds the Shape That has the Maximum Perimeter

```
Shape* MaxPerimeter(Shape *shapes)
{

	Shape *cs = shapes,*ccs = shapes;
	double MaxPerimeter = cs->Area;
	for(;cs!=NULL;cs=cs->next)
	{
		if(cs->Perimeter>MaxPerimeter)
			MaxPerimeter = cs->Area;
	}

	for(;ccs!=NULL;ccs=ccs->next)
		if(ccs->Perimeter==MaxPerimeter)
			return ccs;

}
```

Here is the Function That Finds the Shape That has the Minimum Area

```
Shape* MinArea(Shape *shapes)
{
	Shape *cs = shapes,*ccs = shapes;
	double minArea = cs->Area;
	for(;cs!=NULL;cs=cs->next)
	{
		if(cs->Area<minArea)
			minArea = cs->Area;
	}

	for(;ccs!=NULL;ccs=ccs->next)
		if(ccs->Area==minArea)
			return ccs;

}
```

Here is the Function That Finds the Shape That has the Minimum Perimeter

```
Shape* MinPerimeter(Shape *shapes)
{
	Shape *cs = shapes,*ccs = shapes;
double minPerimeter = cs->Area;
for(;cs!=NULL;cs=cs->next)
{
	if(cs->Perimeter<minPerimeter)
		minPerimeter = cs->Area;
}

for(;ccs!=NULL;ccs=ccs->next)
```

```
            if(ccs->Perimeter==minPerimeter)
                  return ccs;
}
```

Here is the Function that Finds the Shape That has the Minimum Volume

```
Shape* MinVolume(Shape *shapes)
{
      Shape *cs = shapes,*ccs = shapes;
      double minVol = cs->Volume;
      for(;cs!=NULL;cs=cs->next)
      {
            if(cs->Volume<minVol)
                  minVol = cs->Volume;
      }

      for(;ccs!=NULL;ccs=ccs->next)
            if(ccs->Volume==minVol)
                  return ccs;
}
```

Try Yourself: *Use the functions* **MaxArea(),MaxPerimeter(),MaxVolume(), MinArea(), MinPerimeter() and MinVolume()** *listed above so that the surveyer can type commands to get the max volume, min area of a list of shapes. The commands will be* **mav,miv,maa,mia,map,mip** *where mav stands for maximum volume and so on. The user should be able to give the commands like* **Survey mav("shapes.txt")** *where* **shapes.txt** *is the file that contains shape description like surv.txt shown above.*

Example 16.11 ***Write a program to send a file to the connected printer.***

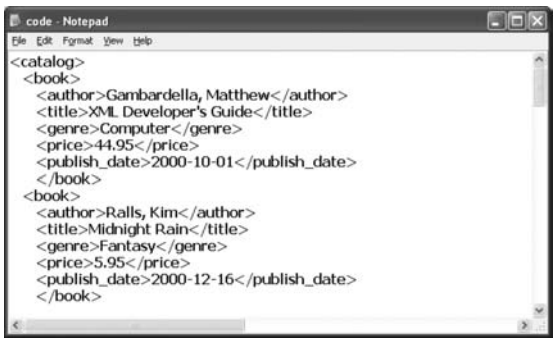
Solution This program works only in Turbo C++ compilers.

```
#include <stdio.h>

int main()
{
      FILE *fp = fopen("C:\\sonnet.txt","r");
      while(!feof(fp))
      {
            fprintf(stdprn,"%c", fgetc(fp));
      }
      fclose(fp);

      return 0;
}
```

Example 16.12 ***Write a program that will read an XML file as shown below and will create a rough database table schema from that which can be basis for further development.***

Solution The following program reads the above XML file and outputs the table creation SQL query.

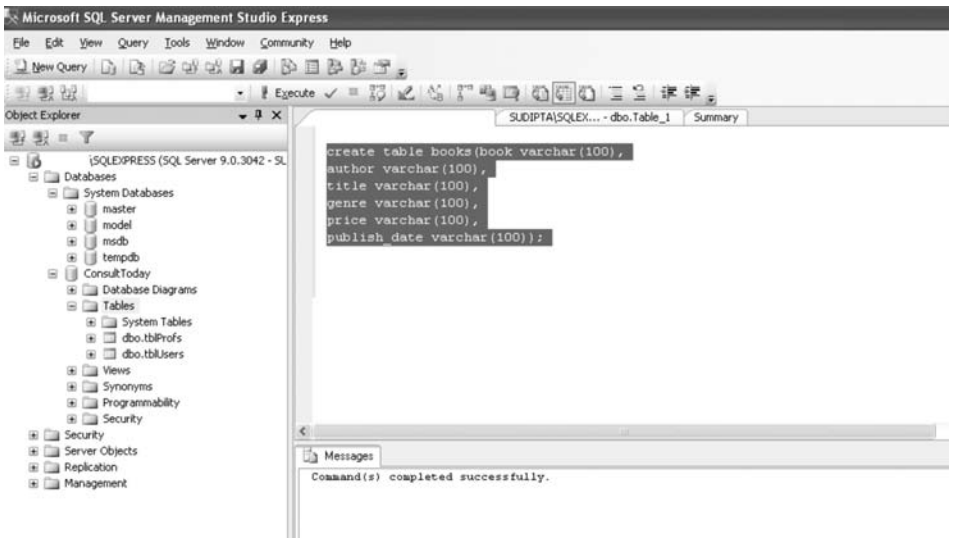
```
int main()
{
      int count = 0;
      int index = 0;
```



```
char size[6];
char type[10];
char tag[20];
String *XML_Schema = NULL;
String *CXML_Schema = NULL;

String *xmls = NULL,*cxmls = NULL;
String *toks = NULL;
FILE *fp = fopen("C:\\code.txt","r");
char line[81];
while(!feof(fp))
{
      fgets(line,81,fp);
      xmls = push_back_String(xmls,line);
      count++;
      if(count==1)
            cxmls = xmls;

}
fclose(fp);
count = 0;
for(;cxmls!=NULL;cxmls=cxmls->next)
{
      toks = split(cxmls->s,">" );
      if(strchr(toks->s,'<')!=NULL)
      {
            index = strchr(toks->s,'<')-toks->s+1;

            strcpy(tag,substring(toks->s,index
                        ,strlen(toks->s)));
            //printf("Tag = %s\n",tag);
            if(strchr(tag,'/')==NULL)
            {
                  XML_Schema =
                   push_back_String(XML_Schema,tag);

                  count++;
                  if(count==1)
                  {
                        CXML_Schema = XML_Schema;

                  }

            }

       //Assuming there is a Grand Mother Tag, like Catalog
       //which can serve as the database
      //name in SQL Server.

     else
     {
            CXML_Schema= CXML_Schema->next;
            printf("create table NewXMLSchema(");
            for(;
                  CXML_Schema->next!=NULL
```

```
                    ;CXML_Schema=CXML_Schema->next)
                {

                printf("%s
                 varchar(100),\n",CXML_Schema->s);
                }
                                printf("%s varchar(100));",CXML_Schema->s);

                                break;
                            }


                    }
                }
                getch();
                return 0;
}
```

When run with the above sample XML, the program outputs the following query to create the corresponding table schema in SQLserver.



This screenshot shows that the generated query is bugfree and executes in SQL Server Express Edition successfully.

16.12 FILE HANDLING IN CONSOLE-BASED GAMES

Now we will demonstrate application of text files in console based games. The program below allows the user to play Sokoban. It is an old Japanese strategy game. Originally developed by a company called *Thinking Rabbit* in Japan in 1980. This game own *game of the year* award at that time. The basic philosophy of the game is very simple. In Japanese "Sokoban" means "*Warehouse keeper*". All you have to do is to push all the boxes ($) to the goal squares (.) by giving commands to sokoban (@). If sokoban (The player) is on a goal square then the symbol of the goal square will be changed to (+). But you can't do the following.

(a) Can't make sokoban(@) or boxes($) move past the walls (#) of the shop.
(b) Can't push two or more boxes together.
(c) Can't pull any box.

In all the cases above, the game will terminate ! You can make your sokoban move by all the four cursor keys or by *l* (left), *r*(right), *u* (up), *d*(down). Or by 2 ,4 ,6 ,8 as in the mobile phones. The places where you should place the boxes($) are known as goal squares and are denoted by full stop(.). While your sokoban moves over a goal square it takes the shape of plus sign (+) and when a box is placed over a goal square it becomes an star(*).

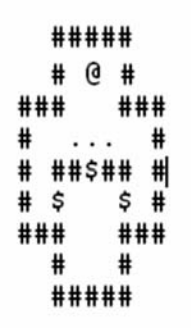
The program reads the map files of the warehouses from the notepad files and loads them.

The game has 71 levels. If you want to play any particular level just press *g*. The program will ask you to enter the level you want to play. After you enter the number the program will load the map file you requested and be ready to accept your commands. In case you need any help simply press ? and you will have all the online help on your screen.

Here is a typical Map file that are read from Notepad Files. The # denotes the border/walls of the warehouse.

```
/*This program implements the old japanese game sokoban*/
/*This prpgram will only compile and run on Turbo C++ 3.5 or more under DOS*/

/*Header File inclusions*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <process.h>
#include <conio.h>
#include <dos.h>

#define Musik sound(490);delay(30);nosound();

/*-----------------------Function declarations-----------------------------*/
/*Shows Menu for the User*/
void Show_Menu();
/*avoid blinking*/
void no blink please(int,int,int,int);
/*draws walls*/
void draw_walls(int[],int[],int);
/*draws goals*/
void draw_goals(int[],int[],int);
/*draw boxes*/
void draw_boxes(int[],int[],int[],int [],int);
/*checks for collision*/
int check_collision(int[],int[],int[],int[],int,int,int,int);
/*checks for sokoban on goal*/
int issokoban_on_goal(int,int,int[],int [],int);
/*are all the boxes properly placed*/
int is_goal_reached(int,int,int[],int [],int);
/*wheather there is any box or not ? */
int is_any_box_there(int [],int[],int,int,int);
/*Checks if the box is movable to up or not ?*/
```

```
int is_movable_to_up(int,int,int[],int[],int);
/*Checks if the box is movable to down or not ?*/
int is_movable_to_down(int,int,int[],int[],int);
/*/*Checks if the box is movable to left or not ?*/
int is_movable_to_left(int,int,int[],int[],int);
/*Checks if the box is movable to right or not ?*/
int is_movable_to_right(int,int,int[],int[],int);
/*****************************************************************************/

/************** Global Variable ***************/
/*Holds the Map File Identification number*/
int counter=0;
/*-----------------------------------------------*/
/************************ MAIN PROGRAM STARTS HERE ************************/
int main()
{
      /*Holds names of the map files*/
      char *Mapfile[72]={
                  "soko001.txt",
                  "soko002.txt",
                  "soko003.txt",
                  "soko004.txt",
                  "soko005.txt",
                  "soko006.txt",
                  "soko007.txt",
                  "soko008.txt",
                  "soko009.txt",
                  "soko010.txt",
                  "soko011.txt",
                  "soko012.txt",
                  "soko013.txt",
                  "soko014.txt",
                  "soko015.txt",
                  "soko016.txt",
                  "soko017.txt",
                  "soko018.txt",
                  "soko019.txt",
                  "soko020.txt",
                  "soko021.txt",
                  "soko022.txt",
                  "soko023.txt",
                  "soko024.txt",
                  "soko025.txt",
                  "soko026.txt",
                  "soko027.txt",
                  "soko028.txt",
                  "soko029.txt",
                  "soko030.txt",
                  "soko031.txt",
                  "soko032.txt",
                  "soko033.txt",
                  "soko034.txt",
                  "soko035.txt",
                  "soko036.txt",
                  "soko037.txt",
                  "soko038.txt",
```



```
                "soko039.txt",
                "soko040.txt",
                "soko041.txt",
                "soko042.txt",
                "soko043.txt",
                "soko044.txt",
                "soko045.txt",
                "soko046.txt",
                "soko047.txt",
                "soko048.txt",
                "soko049.txt",
                "soko050.txt",
                "soko051.txt",
                "soko052.txt",
                "soko053.txt",
                "soko054.txt",
                "soko055.txt",
                "soko056.txt",
                "soko057.txt",
                "soko058.txt",
                "soko059.txt",
                "soko060.txt",
                "soko061.txt",
                "soko062.txt",
                "soko063.txt",
                "soko064.txt",
                "soko065.txt",
                "soko066.txt",
                "soko067.txt",
                "soko068.txt",
                "soko069.txt",
                "soko070.txt",
                "soko071.txt"
        };
FILE *fp;
int flag=0;//return value of check_collision() is stored
int w;//stores index number of the box sokoban is currently pushing
char ch;// stures character that are read from the map file
char Commands[100];//Command keys are stored for Undo
int NumberOfMoves=0;//Number of moves are stored here
/*goal_x[] and goal_y[] stores the co_ordinate values
  of the goal_squares('.') of the current map file*/
int goal_x[10],goal_y[10],
/*walls_x[] and walls_y[] stores the co_ordinate values
  of the boundaries('#') of the current map file*/
 walls_x[500],walls_y[500],
/*sbx[] and sby[] stores the co_ordinate values
  of the boxes('$') of the current map file*/
 sbx[10],sby[10],
 /*skx,sky store the x and y co-ordinates of the
   sokoban's initial position in the map*/
 skx,sky,
 /*loop variables*/
    i=0,j=0,l=0;
/*clears the screen*/
clrscr();
/*Hides Cursor*/
```

```
_setcursortype(_NOCURSOR);
/*Loads one Map_File after another */
for(;counter<71;)
{
      /*Open a map file for reading*/
      fp=fopen(Mapfile[counter],"r");
      do
      {
      /*Scans a character at a time*/
      fscanf(fp,"%c",&ch);
      printf("%c",ch);

      if(ch=='.')//is it a goal_square
      {
            //stores the co-ordinates of the goal_squares
            goal_x[i]=wherex();
            goal_y[i]=wherey();
            i++;
      }

      if(ch=='@')//is it sokoban itself
      {
            //stores the co-ordinates of the
            //sokoban
            skx=wherex();
            sky=wherey();
      }
      if(ch=='$')//is it a box
      {
            //stores the
            //co-ordinates of the boxes
            sbx[j]=wherex();
            sby[j]=wherey();
            j++;
      }

      if(ch=='#')//if it is a boundary element
      {
            //stores the
            //co-ordinates of the boundary walls
            walls_x[l]=wherex();
            walls_y[l]=wherey();
            l++;
      }

}while(!feof(fp));//continues untill the EOF is reached

clrscr();
do
{
      //is the sokoban pushing against wall or
      //is it pushing a box against a wall
      flag=check_collision
      (walls_x,walls_y,sbx,sby,w,l,skx,sky);
      if(flag==0)//No Collision
      {
            gotoxy(30,18);
```

```
                        printf("Sokoban Position : (%d,%d)",skx,sky);
                        gotoxy(30,20);
                        printf("Level : %d",counter+1);
                        gotoxy(30,22);
                        printf("Number of Moves : %d",NumberOfMoves);
                        gotoxy(15,24);
                        printf("Press Spacebar to jump to Next Level and Q/E to
exit");
                        //is sokoban pushing any box ?
                        w=is_any_box_there(sbx,sby,skx,sky,j);
                        //draws the walls
                        draw_walls(walls_x,walls_y,l);
                        //draws the goals
                        draw_goals(goal_x,goal_y,i);
                        //draws the boxes with proper sign
                        draw_boxes(sbx,sby,goal_x,goal_y,j);
                        //checks if the sokoban is on a goal_square
                        Show_Menu();
                        if(issokoban_on_goal(skx,sky,goal_x,goal_y,j))
                        {
                              gotoxy(skx,sky);
                              printf("+");//sokoban player is on a goal square
                        }
                        else
                        {
                              gotoxy(skx,sky);
                              printf("@");
                        }
                        //Checks if the boxes are properly placed
                        //on the goal squares
                        if(is_goal_reached(sbx[w],sby[w],goal_x,goal_y,j))
                        {
                              gotoxy(sbx[w],sby[w]);
                              printf("*");//box is on a goal square
                        }
                        ch=getch();
                        //Up arrow key or U or u or 8 to move up
                        if(ch==72||toupper(ch)=='U'||ch=='8')
                        {
                              Musik;
                              no_blink_please(skx,sky,sbx[w],sby[w]);
                              sky--;//sokoban moves one step down
                              //is sokoban tries to move a box
                              //if yes what is the box index ?
                              w=is_any_box_there(sbx,sby,skx,sky,j);
                              //Sokoban is trying to push a box
                              //let's see if it is movable or not in
                              //that direction or not
                              if(w!=-1&&!is_movable_to_up(skx,sky,sbx,sby,j))
                              {
                                    clrscr();
                                    gotoxy(20,15);
                                    printf("Game Over ! Sokoban pushed two boxes
together!");
                                    getch();
                                    exit(0);
                              }
```

```
                        //There is a box sokoban tries to move
                        //and it is movable !!
                        if(w!=-1)
                        {
                              //Sokoban pushes it one step down
                              sby[w]=sky-1;
                              NumberOfMoves++;
                        }
                              //sokoban is not pushing any box
                              //it is simply moving
                              else
                                    NumberOfMoves++;
                        }

                        //Down arrow key or D or d or 2 to move down
                        if(ch==80||toupper(ch)=='D'||ch=='2')
                        {
                              Musik;
                              no_blink_please(skx,sky,sbx[w],sby[w]);
                              sky++;
                              w=is_any_box_there(sbx,sby,skx,sky,j);
                              if(w!=-1&&!is_movable_to_down(skx,sky,sbx,sby,j))
                              {
                                    clrscr();
                                    gotoxy(20,15);
                                    printf("Game Over ! Sokoban pushed two boxes
together!");
                                    getch();
                                    exit(0);
                              }
                              if(w!=-1)
                              {
                                    sby[w]=sky+1;
                                    NumberOfMoves++;
                              }
                              else
                                    NumberOfMoves++;
                        }
                        //Left arrow key or L or l or 4 to move left
                        if(ch==75||toupper(ch)=='L'||ch=='4')
                        {
                              Musik;
                              no_blink_please(skx,sky,sbx[w],sby[w]);
                              skx--;
                              w=is_any_box_there(sbx,sby,skx,sky,j);
                              if(w!=-1&&!is_movable_to_left(skx,sky,sbx,sby,j))
                              {
                                    clrscr();
                                    gotoxy(20,15);
                                    printf("Game Over !
                                    Sokoban pushed two boxes together!");
                                    getch();
                                    exit(0);
                        }
                        if(w!=-1)
                        {
                              sbx[w]=skx-1;
                              NumberOfMoves++;
```

```
		}
		else
			NumberOfMoves++;
	}
	//Right arrow key or R or r or 6 to move right
	if(ch==77||toupper(ch)=='R'||ch=='6')
	{
		Musik;
		no_blink_please(skx,sky,sbx[w],sby[w]);
		skx++;
		w=is_any_box_there(sbx,sby,skx,sky,j);
		if(w!=-1&&!is_movable_to_right(skx,sky,sbx,sby,j))
		{
			clrscr();
			gotoxy(20,15);
			printf("Game Over ! Sokoban pushed two boxes
together!");
			getch();
			exit(0);
		}
		if(w!=-1)
		{
			sbx[w]=skx+1;
			NumberOfMoves++;
		}
		else
			NumberOfMoves++;
	}

	if(tolower(ch)=='g')//Jumps to load a requested map
	{
		int level;
		gotoxy(30,20);
		_setcursortype(_SOLIDCURSOR);
		printf("Enter Level :");
		scanf("%d",&level);
		counter=level-1;
		if(counter<=-1||counter>=71)
		{
			clrscr();
			gotoxy(20,22);
			printf("No Such Map File!!");
			getch();
			//Map file Id
			//is re initialized
			//to starting Map file
			counter=0;
			main();
		}
		main();
	}
	//? for Help on sokoban
	if(ch=='?') //Help
	{
		gotoxy(25,4);
		printf("-------------- H E L P ------------------");
		gotoxy(25,5);
```

```
sokoban");                          printf("Use Arrow Keys or L/R/U/D or 4-8-6-2 to move

                                    gotoxy(25,6);
                                    printf("Sokoban can't push two box together");
                                    gotoxy(25,7);
                                    printf("Sokoban can't walk over a box or walls");
                                    gotoxy(25,8);
                                    printf("Sokoban can't move or dislodge a box from a
corner");
                                          gotoxy(25,9);
                                          printf("Press Enter to return to game");
                                          gotoxy(25,10);
                                          printf("Press ? to See HELP again");
                                          getch();
                                    }

                                    if(ch==8)//backspace
                                    {
                                          counter--;
                                          main();
                                    }
                                    if(ch==32)//Spacebar is pressed
                                      //to load the next map file
                                    {
                                          counter++;
                                          main();
                                    }
                                    //Press 'Q'/'q'/'E'/'e' to quit
                                    if(toupper(ch)=='Q'||toupper(ch)=='E')
                                          exit(0);
                              }

                              //if collision occurs then Game is Over
                              else
                              {
                                    clrscr();
                                    gotoxy(20,15);
                                    printf("Game Over ! Sokoban or the Box hit the walls !!");
                                    getch();
                                    exit(0);
                                    main();
                              }
                             } while(1);//Always true
                        }
                       getch();
            return 0;
            }

/************************  MAIN PROGRAM ENDS HERE ************************/
```

16.13 FUNCTION DEFINITIONS

Function : `draw_walls()`

```
  ======================
/*Draws the outside boundary of the map*/
/*-----------------------------------------------------------------------*/
```

```
void draw_walls(int walls_x[],int walls_y[],int bound)
{
      for(int b=0;b<bound;b++)
      {
            gotoxy(walls_x[b],walls_y[b]);
            printf("#");
      }
}
/*********************************************************************
```

Function : `draw_goals()`

```
  ======================
/*Draws the goal square of the map*/
/*--------------------------------------------------------------------*/
void draw_goals(int goal_x[],int goal_y[],int bound)
{
      for(int b=0;b<bound;b++)
      {
            gotoxy(goal_x[b],goal_y[b]);
            printf(".");
      }
}
/*********************************************************************
```

Function : `draw_boxes()`

```
  ======================
  Draws the boxes of the map with appropriate sign
  $ for all the boxes not yet reached any goal_squares
  and * for all of those who reached any of the goal_squares
/*--------------------------------------------------------------------*/
void draw_boxes(int sbx[],int sby[],int goal_x[],int goal_y[],int bound)
{
      for(int b=0;b<bound;b++)
      {       if(is_goal_reached(sbx[b],sby[b],goal_x,goal_y,bound))
            {
                  gotoxy(sbx[b],sby[b]);
                  printf("*");
            }
            else
            {
                  gotoxy(sbx[b],sby[b]);
                  printf("$");
            }
      }
}
/*/*******************************************************************
```

Function : `check_collision()`

```
   ======================
/*Checks whether a collision between sokoban and wall
  or box and wall has occurred or not and also checks
  that if the sokoban is trying to move two sokobans at the same time*/
/*--------------------------------------------------------------------*/
```

```
int check_collision(int walls_x[],int walls_y[],
                int sbx[],int sby[],int box_index,
                int walls_bound,int sokoban_x,int sokoban_y)
{
      int status=0;
      for(int b=0;b<walls_bound;b++)
      {
            if((sokoban_x==walls_x[b]&&sokoban_y==walls_y[b])||
               (sbx[box_index]==walls_x[b]&&sby[box_index]==walls_y[b]))
               {
                  status=1;
                  break;
               }
      }
      return status;
}

/*/*****************************************************************
```

Function : `issokoban_on_goal()`

```
   ============================

/*Checks if the sokoban is on any goal_square or not
  if yes it automatically changes to a '+' sign
  else it travels as a '@'*/
/*-----------------------------------------------------------------*/
int issokoban_on_goal(int skx,int sky,int goal_x[],int goal_y[],int bound)
{
      int status=0;
      for(int loop=0;loop<bound;loop++)
      {
            if(skx==goal_x[loop]&&sky==goal_y[loop])
            {
                  status=1;
                  break;
            }
            else
                  status=0;
      }
       return status;
}

/*/*****************************************************************
```

Function : `is_goal_reached()`

```
   ===========================
/*Checks if the box sokoban is currently pushing reaches any of
  the given goal_squares of the map or not.*/
/*-----------------------------------------------------------------*/
int is_goal_reached(int box_x,int box_y,int goal_x[],int goal_y[],int bound)
{
      int flag=0;
```

```
for(int loop=0;loop<bound;loop++)
{
      if(box_x==goal_x[loop]&&box_y==goal_y[loop])
      {
            flag=1;
            break;
      }
}
return flag;
}

/*********************************************************************
```

Function : is_any_box_there()

```
   ============================
/*Returns the index number of the box sokoban is trying to push
 if it exists, else return -1 to indicate that sokoban is not
 pushing any box.*/
/*-------------------------------------------------------------------*/

int is_any_box_there(int sbx[],int sby[],int skx,int sky,int bound)
{
      int position=-1;//No box is being pushed by sokoban
      for(int loop=0;loop<bound;loop++)
      {
            if(skx==sbx[loop]&&sky==sby[loop])
            {
                  position=loop;
                  break;
            }
      }
return position;
}

/*/*******************************************************************
```

Function : is_movable_to_up()

```
   ============================
/*Checks if the box sokoban is push can be shifted upward or not*/
/*-------------------------------------------------------------------*/
int is_movable_to_up(int sokoban_x,int sokoban_y,int sbx[],int sby[],int bound)
{
      int status=1;
      for(int loop=0;loop<bound;loop++)
      {
            if(sokoban_y-1==sby[loop]&&sokoban_x==sbx[loop])
            {
                  status=0;
                  break;
            }
      }
      return status;
}
/*/*******************************************************************
```

Function : is_movable_to_down()

```
   ============================
/*Checks if the box sokoban is push can be shifted downward or not*/
/*-------------------------------------------------------------------*/
int is_movable_to_down(int sokoban_x,int sokoban_y,
                   int sbx[],int sby[],int bound)
{
      int status=1;
      for(int loop=0;loop<bound;loop++)
      {
            if(sokoban_y+1==sby[loop]&&sokoban_x==sbx[loop])
            {
                  status=0;

                  break;
            }
      }
      return status;
}
/*/*****************************************************************
```

Function : is_movable_to_left()

```
   ============================
/*Checks if the box sokoban is push can be shifted towards left or not*/
/*-------------------------------------------------------------------*/
int is_movable_to_left(int sokoban_x,int sokoban_y,
                   int sbx[],int sby[],int bound)
{
      int status=1;
      for(int loop=0;loop<bound;loop++)
      {
            if(sokoban_x-1==sbx[loop]&&sokoban_y==sby[loop])
            {
                  status=0;
                  break;
            }
      }
      return status;
}

/*/*****************************************************************
```

Function : is_movable_to_right()

```
   ==============================
/*Checks if the box sokoban is push can be shifted towards right or not*/
/*-------------------------------------------------------------------*/

int is_movable_to_right(int sokoban_x,int sokoban_y,int sbx[],int sby[],int bound)
{
      int status=1;
      for(int loop=0;loop<bound;loop++)
      {
            if(sokoban_x+1==sbx[loop]&&sokoban_y==sby[loop])
            {
                  status=0;
                  break;
            }
      }
      return status;
```

```
}

/*This function shows the menu*/
void Show_Menu()
{
      gotoxy(25,4);
      printf("---------------- User Menu ------------------------ ");
      gotoxy(25,5);
      printf("Use Arrow Keys or L/R/U/D or 4-8-6-2 to move sokoban");
      gotoxy(25,6);
      printf("L/l/4/Left arrow key to move sokoban LEFT");
      gotoxy(25,7);
      printf("R/r/6/Right arrow key to move sokoban RIGHT");
      gotoxy(25,8);
      printf("U/u/8/Up arrow key to move sokoban UP");
      gotoxy(25,9);
      printf("D/d/2/Down arrow key to move sokoban DOWN");
      gotoxy(25,10);
      printf("g/G to load a particular map file");
      gotoxy(25,11);
      printf("m/M to see the number of moves till now");
      gotoxy(25,12);
      printf("? for HELP ");
}

/*This function avoids blinking*/
void no_blink_please(int sokoban_x,int sokoban_y,
            int box_x,int box_y)
{
      gotoxy(sokoban_x,sokoban_y);
      printf(" ");
      gotoxy(box_x,box_y);
      printf(" ");
}
```

The map files will be available from OLC

REVIEW QUESTIONS

1. Which of the following statements will make us loose the previous content of file abc.txt
   (a) FILE *p = fopen("abc.txt","r") (b) FILE *p = fopen("abc.txt","w");
   (c) FILE *p = fopen("abc.txt","w+"); (d) FILE *p = fopen("abc.txt","a");
2. Which file input/output function allows the formatted file output.
3. Which of the following can take the variable number of arguments
   (a) fprintf() (b) fgets() (c) scanf() (d) vprintf()
4. What does the function ftell() do?
5. Can we check whether the file we are trying to open already exists or not? If yes, how?
6. Which of the following statements is/are false
   (a) fseek() returns an integer
   (b) SEEK_SET sets the location of the pointer to the current position
   (c) __FILE__ returns the name of the file name we are currently on
   (d) rewind() is a wrapper function
7. What is a File?
8. What could be different modes of opening a file?

9. What are the differences between *w+* and *w* mode?
10. What are the difference between *a* and *w* mode?
11. Where is the mode *wb* stands for?

PROGRAMMING PROBLEMS

1. Write a program to demonstrate the use of fread and fwrite.
2. Write a program to offer a command line version of *cp* where multiple target files can be given as input.
3. Write a program to offer a command line version of cat where multiple target files can be given as input.
4. Write a program that accepts an HTML file and reports any missing tag.
5. Update the above program to accommodate the allowance for Orphan Tags. A tag is called orphan tag when the starting tag end is not required. For example </br> is an orphan tag. It doesn't require a opening tag.
6. Write a program that accepts a list of strings along with some kind of parent child relationship and creates a XML file from it.
7. Write a program that simulates an online banking experience using multiple files for maintaining users personal and their bank account details. The bank account generation should be automatic.
8. Write a program that simulates a share index monitoring system. There will be different files that users can point to for the application. These files will contain the share information of different companies in a delimited file [Hint: *Use String structure and split() funcition*]
9. Write a program to simulate a model of a toll-plaza automation. There are different charges for different types of vehicles. These charges can be modified. Be creative with functionalities. More you offer the better. One direction is to provide a report for vehicles per day and the total toll tax earned.
10. Write a program to read a VB code from a notepad file and change the variable names according to a given variable naming guideline.
11. Write a program that will read a document and will encrypt it.
12. Write a program that will decrypt the above encrypted file.
13. Write a program to read a C program and print the interdependency of the functions.
14. Write a program to read a file and then generate all the k-grams of it. Keep a provision such that k can be supplied by the user.
15. Write a program to read a delimited file with the following pattern as shown in the figure below.

After reading create separate files that will hold the users in different domains. If the above file is given to the program as input the following two files should be generated.

Note the file name is same as the domain name.
16. Write a function that will calculate which domain has the maximum users.
17. Write a function that will calculate which domain has maximum popularity among teenagers.
18. Write a function that will show trend of gender preference of domain names in decreasing order.

Appendix A
Project Ideas!

"Everything begins with an idea!"
—Earl Nightingale
"Great ideas often receive violent opposition from mediocre minds."
—Albert Einstein
"Ideas not coupled with action never become bigger than the brain cells they occupied."
—Arnold H. Glasgow

IDEA #1: IDENTIFYING A SET

The idea here is to create sets of different objects. The program will ask the user to enter a few names of related objects and then it will first tell what is that object and then will list a few more objects of that set. For example, if the user enters guitar then the program should first output that "Guitar is an object of stringed musical instrument" and then list a few more stringed musical instruments, for example, "Electric Guitar", "Hawaiian Guitar", "Harp", "Sitar", "Sarod" , "Mandolin", etc.

[Hint]: Create a trie to store the strings and fetch from there. If a string is matched against Musical Instrument Trie then the string is a musical instrument and so on.

IDEA #2: ATM

Imagine that you have been hired by a bank to design a software to plug into their ATM machine. Customers will key in their card number and PIN. The customer base of the bank is very large. So you might want to store them in a special way/format for minimum retrieval time. Once the customer keys in proper card number and PIN then the screen will show four options

Withdraw Cash
Deposit Cash
Balance Check

Mini Statement
Exit

These operations should be the same as they behave in ATMs. For example, if I want to withdraw 23.50, it should not allow me. Because the ATM can only dispatch currency notes in the denomination of 10 or multiples of 10.

Keep a provision in your program to change the denomination of currency to make the program universal. Different countries have different currency denominations.

IDEA #3: LINE EDITOR

Create an editor like vi or sam or pico in UNIX using linked lists. A line editor is nothing but a linked list of lines, where lines are nothing but a linked list of words, and words are nothing but a linked list of characters. So, create different structures to represent character, word and line and add different functions to support normal editing experience.

IDEA #4: POS(POINT OF SALE) FULLY CUSTOMIZABLE

Almost every store these days are computerized and maintain a small program known as POS (Point of Sale), that allows then to check out items for customers. Create a POS program that can be customized for any kind of store starting from a shoe store to a departmental chain like WAL-MART. The following are the minimum desired feature of the POS software. Moreover, you will be creative logically, more good.

The program will allow for *credit card payment*. (You might want to use the credit card validation program in the chapter on String) .

The user (The counter person) will enter the product code and the system will verify the product code and will produce a go ahead signal. The system will calculate the total amount and show it on the screen. Once payment is made the system will generate a bill listing all the item description and their price.

The system will maintain a transaction log for 3 months in case a customer comes back and want to exchange something with a valid bill.

IDEA #5: CUSTOMIZED ADDRESS BOOK WITH LITTLE SQL SUPPORT

When more than one person uses a computer it is not judicial to use the windows address book for storing personal friend details. Create a program to allow the users to create different user ids and passwords and maintain their own personal friend list/scrap book etc. The program will have a predefined set of fields for storing details of friends. Once a new user is created, then the program will ask the user to add columns to the address book. Suppose the predefined fields are “Name”, “Address”, “Phone”,”Email” and “Blog Site”. Now a new user is created and he doesn’t want to enter “Blog Site” details for his friends. So the columns he adds for his very own customized address book are Name, Address, Phone and Email. These program will encrypt the passwords and store the user id/password combo in one file and the details of friends in separate files.

While searching the program we need to support little SQL feature. The program should only support “Where <column_name> = “value” feature

Search can be like

Select name, phone from friends where name = ‘sam’ and in these sort of cases the query will only print the column which were mentioned, as here name and phone. If we give a ‘*’ then all the columns should be printed as in SQL like

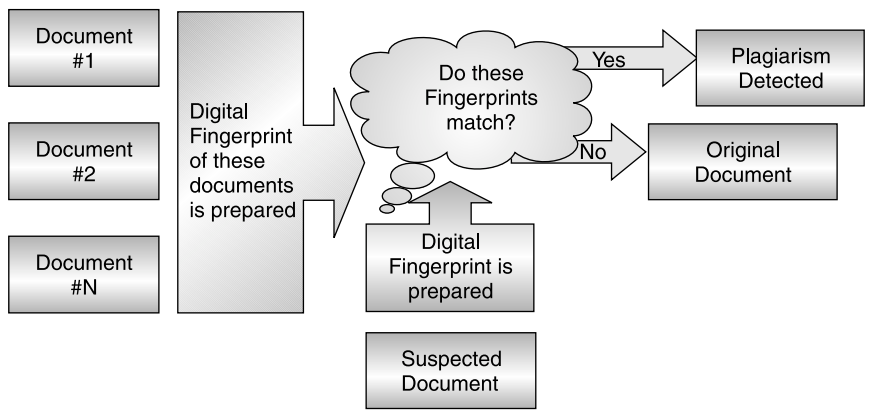
Select * from friends where name = ‘sam’

IDEA #6: CREATION OF MAGIC SQUARES

Write a program that accepts a number and create a magic square as NxN matrix. A Magic Square is a square that sums to the same number in all directions. The magic square for 3 is

8 1 6
3 5 7
4 9 2

IDEA #7: DETECTION OF PLAGIARISM



Sometimes students are found to copy from different source files without the prior permission of the author or the publisher. This activity is illegal as per the law for protecting the intellectual property is concerned. This type of activity is commonly referred as Plagiarism and the person who performs this act is known as Plagiarist.

There are several techniques to detect whether a suspected document is from a plagiarist or not. But most of them are copyrighted. We will discuss about a topic called "Winnowing." The plagiarism involved document can be a combination from so many different documents. So, searching for string match in all those documents is practically not feasible. Winnowing avoids this by a smart hashing technique.

The idea behind winnowing lies in the fact that every document, no matter how big it is has a close to unique digital fingerprint (Which is a combination of k-gram hash codes) that makes it different from other documents.

Here are the steps for winnowing.

1. Remove all the white space and punctuation from the text.
2. Create *k* grams of the lines.
3. Hash those *k* gram words.
4. Take Hash code of *N* blocks and that constitutes the digital fingerprint of the document.

Latter prepare the fingerprint of the suspected document and match them. If they match then the document is really from a plgiarist otherwise it is the original work of its author.

IDEA #8: *D*YNAMIC *A*DAPTIVE *Q*UIZ *S*YSTEM (DAQS)

Create a program that will allow users to load quiz papers in a delimited notepad file. The notepad file

will be of such format that the program knows which question to ask next, if the answer is correct and if it is wrong. (You might want to refer the binary spider data structure described in chapter on trees).

At the end of the quiz the program should create a report telling how many questions are answered correct and how many questions are answered wrong.

IDEA #9: SIMPLE ENGLISH TO C CODE GENERATOR (SETCOGEN)

This is probably going to be the most difficult and fun-to-achieve project. The idea is that the user will write the requirement in plain and structured English and the program will convert it to a C code. Here is an example "English Program" that a layman might write.

Ask "What is the radius"
Calculate area
Print "Area of the Circle is"

The output will be the following C code.

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

int main()
{
      int radius = 0;
      printf("What is the radius ");
      scanf("%d",&radius);
      printf("Area of the Circle is %f\n",3.14159*radius*radius);
      getch();
      return 0;
}
```

IDEA #10: DETECTING AUTHOR'S GENDER (DAG)

If you notice carefully, you will find that men and women use very disjoint sets of words to express the same feeling or experience. This fact can be helpful to detect the gender of the author. Women use more pronouns while men use more nouns. Work of women authors are more involved while that of male authors are informative. Women do not use the definite article very often, while that is a very prominent characteristic of male authors. There are so many such features that help us distinguish male and female authors.

Steps for this project

Step #1: Collect samples of male and female writings for the same topic.
Step #2: Distinguish between their writing style and try to extract features.
Step #3: Design an algorithm that will use the features from step two to detect the gender of an author's gender.
Step #4: Feed different inputs to your algorithm and see how it performs.
Step #5: Keep track of all the incidents where your algorithm fails and where it successfully detects the gender of the author. Plot a graph to test the success rate of your algorithm.

Appendix B
Bibliography

Books are the bees which carry the quickening pollen from one to another mind.
—James Russell Lowell

- Donald E. Knuth, *The Art of Computer Programming* (vols I, II and III)
- Jeffrey Esacov, Tom Weiss, *Data Structures—An Advanced Approach using C,* Prentice Hall Software Series
- Yedidyah Langsam, Moshe J. Augenstein, Aaron M. Tenenbaum, *Data Structures using C and C++*, PHI
- Niklaus Wirth, *Algorithms + Data Structures = Programs*, PHI
- Mark Allen Weiss, *Data Structure and Problem Solving using C++*, Addison Wesley Longman
- Norsingh Deo, *Graph Theory with Applications to Engineering and Computer Science*, PHI
- Graham A. Stephen, *String Searching Algorithms*, World Scientific
- Fred S. Roberts, *Graph Theory and its Applications to Problems of Society*
- Seymour Lipschutz, *Theory and Problems of Data Structures,* TMH
- Jean-Paul Trembly, Paul G. Sorenson, *An Introduction to Data Structures with Application.*
- Brian W. Kernighan, Dennis M. Ritchie, *The C Programming Language*
- Ivor Horton, *Beginning C++ the complete language,* Wrox
- Tod Golding, *Professional .NET 2.0 Generics,* Wrox
- James B. Scarborough, *Numerical Mathematical Analysis,* Oxford & IBH Publishing Co.
- Nick Parlante, *Linked List Basics*
- Sanjay Goswami, Susmita Sur Kolay, *Virtual Molecular Computing, Emulating DNA Molecules*
- Bart, Preneel, *Analysis and Design of Cryptographic Hash Functions*

Index

a Hyperbola 78
a Parabola 74
A variant 210
abstraction capabilities 445
Accessor Methods 446
Ackermann's Function 216
Active
  Methods 446
  Search Operation 273
AddAEdge 394
adddays 463
AddEdge 379, 387
addmonths 464
Address Calculation Sort using Hashing 426
AddVertex 386
addyears 464
Adjacency
  List 378, 385
  Matrix 378
ADT 445
ADT Design in C 446
Aitken's Array 233
An
  External Node 326
  Internal Node 326
Anagram 202
Ancestor 325
appendQ 300
Application of Sorting 428
apply 3D Arrays 8
Array 1
Array
  Elements 4
  Using Pointer 2
associative arrays 469
autocorrect the indentation 507
AVL Tree 359

back_element() 447
Balance Factor 354
Ballot Problem 237
Bayesian Networks 384
Bell Triangle 233
Bidirectional Bubble Sort 404
Bidirectional Bubble Sort Performance 407
Big O 429
Binary 326
Binary
  Insertion Sort Performance 413
  Insertion Sorting 411
  Recursion 208
Binary Search Tree 337
Bingo Sort 418
Binomial Coefficient 209
bisection method 223
Black-Box Concept 445
Bloom Filter 433
Bogo Sort and friends 424
Bowley's Index Number 26
Bozo sort 425
Breadth First Search (BFS) 396
Brothers 326

BSP Tree 360
Bubble Sort Algorithm 400
Bucket Sort 422
Business Clustering 428

C-Style String 165
Cash Counter 318
Cassini's Formula 210
cat 496
Catalan's Triangle 235
check_collision() 533
Child 325
chord of a curve 226
Cipher Text 54
Circularity 208
Coalesced Hashing 433, 437
Collision 433
collision and Its resolution 431
Collision Free Hashing 433
Comb Sort 407
Comb Sort Performance 409
CompareUndirectedEdges 391
CompareVertices 390, 391
Complete Binary Tree 327
Complex Number 85
Complexity of
  Bidirectional Bubble Sort 406
  Binary Insertion Sort 413
  Bubble Sort 403
  Insertion Sort 410
  Merge Sort 421
  Quick Sort 419
  Shell Sort 415
Cone 77
ContainsEdge 389
count 389
count() 448
CountEdges 391
countWords 185
Cousin 326
cp 507
create a rough database table schema 522
CTPS 429
Currency 479
Cylinder 76

DAG 384
Date 63, 449
DAY of WEEK 453
daysbetween 453
daystring 455
Decision Tree 347
decrypt 188
Degree 325, 390
degree of the chain 441
delete_alternate() 447
delete_at() 447
delete_range() 447
deleteQ 300
DeleteWords 186
Depth First Search (DFS) 398
Deque 316
dequeue 286
Descendent 326
diff 505
diffv 506
Directed Acyclic 384
Discrete Event Simulations 323
display 390
DisplayEdges 388
displayleapyears 462
displayQ 301
displaystring 190
Distribution Sorting Algorithms 422
Divide-n-Conquer 419
Doboseiwicz Sort 407
DoesItExist 497
DoesThisEdgeExist 395
Double Linked List 107
draw_boxes() 533
draw_goals() 533
draw_walls() 532
Drawn Complete Binary Tree 328
Dyck Path 236

Elements of the Array 3
Ellipse 75
encrypter 55
enqueue 285
Entringer Numbers 239
enum 70
Euclid's algorithm 211
Exchange Sort 400
Exponential Recursion 208
Expression Tree 343
External Sorting 430
External Variables 5

factorial 209
FCFS 322

fclose() 488
FCLS 322
fflush 94
fgetc() 488
fgets() 489
Fibonacci Numbers 209
FIFO 322
FIFO(First In First Out) 284
File 63
File Handling 488
File Handling in Console-Based Games 524
Finding the Most Wanted DVD in the City 428
Finding the Shortest Path 428
findmax() 447
findmin() 447
first in, last out list 247
fopen() 488
fprintf() 490
fputc() 488
fputs() 489
frequency() 447
front_element() 447
fscanf() 490

Gapped Insertion Sort 414
get_value() 447
GetPathMatrix2 379
Grandchild 326
Grandparent 326
GraphCrossingNumber 393
Graphs 378
Greatest Online Scorer 428
grep 497
grepe 499
grepev 501
greps 499
grepsv 500
grepv 498

Hadamard product 31
Hash
  Chain 433
  Function 433
  List 433
  Table 433
  Tree 433, 443
Hashing 431, 433
Hashing function 431
head 494
Heap 358
Height 325
HHI Index 46
Hofstadter Conway $10,000 Sequence 239
Hybrid Sort 418

InDegree 394
Information Categorization 384
Inorder Predecessor 326
Inorder Successor 326
InsertEdge 387
Insertion Sort Algorithms 410
Instruction Scheduling 384
Intersection 385
inverse Ackermann's Function 216
Inverse of a Square Matrix 34
Inverse Quadratic Interpolation 225
is_any_box_there() 535
is_goal_reached() 534
is_movable_to_down() 536
is_movable_to_left() 536
is_movable_to_right() 536
is_movable_to_up() 535
isAnagram 203
isBalanced 394
isComplete 392
isCorCPP 510
isCSharp 511
isEmpty 301
isEuler 392
isFull() 285
isFuture 463
isIsolatedVertex 390
isJava 511
isleapyear 449
isPast 462
isPendant 390
isPlaner 393
isPrefix 179
isPresent 463
isRegular 396
isSameSoundex 206
issokoban_on_goal() 534
isSonnetLines 516
isSTL 510
isSubGraph 391
isSubsequence 181
isSuffix 180
isSymmetric 395
isThisEdgeAselfLoop 395
isTree 380

IsValidCheckDigit 196
isVB 512
isVertexPresent 386, 388

J-Sort 418
Joseph Stein's Algorithm 211

Key Interlinked Map 477
kgrams 193
Koch Curve 243
Koch Snowflake 243
Kronecker product 32
Kruskal's Algorithm to find 381

Lagrange's Interpolation 21
Largest Shape 429
Laspeyre's Index Number 26
lastday 460
LCFS 322
LCLS 322
Leaf 326
Left Child 325
Left Sub Tree 325
LeibniTz Harmonic Triangle 238
Level 325
Lexicographic Order 425
Lexicographic Sort 425
Library Sort 414
LIFO 282
LIFO queue 284
LILO 322
Linear
  Congruential Method 212
  Probing 432
  Probing Technique 433
  Queue 284
  Recursion 208
linked linear queue 284
Linked List 105
Load Factor 433
Losanitsch's Triangle 237
Lucas' Theorem 210

M-ary 326
Makefile Instructions 384
Map 469
Mapfile 526
Marshall–Edward Index Number 27
Matrix Multiplication 30
Max Heap 358
MAX-PQ 298
Merge Sort 420
merge() 448
MIN – PQ 298
Min Heap 358
Minimum Spanning Tree 380
MostUsedWord 202
MRA 282
MTFL 271
Muller's Method 227
Multiway Tree 353
Mutual Recursion 208

Nested Recursion 208
Newton-Raphson Method 222
Newton's Forward Difference Interpolation 20
nextmonthfirstXday 461
nextmonthlastXday 461
nextmonthNthXday 460
nextmonthsameday 460
nextNthday 458
nextyearsameday 459
Node 325
Not- So-Typically 328
Number is happy or not 229

Octree 370
Odd-Even Transposition Sort 404
One Time Passwords 441
One Way Hash Function 433
Open Addressing 431
OTP 433

Paasche's Index Number 26
pad 192
padleft 190
padright 191
Palindrome 13
Parallel Sorting 430
Parent 325
Parenthesis Matcher 258
Pascal Triangle 232
Passive Methods 446
PathMatrix 379
Peep 282
Peirce Triangle 233
Perfect Binary Tree 327
Performance Comparisons of the Sorting Algorithms 423, 424
Phonebook Simulation 86

Pitfalls of Recursion 209
Pivot in Quick Sort 420
Polygon using Point Structure 64
pop 247
pop_back() 446
pop_front() 447
postfix 247
postfix expression 254
Postorder Predecessor 326
Postorder Successor 326
Predictors/Search Methods 446
Prefix of a String 179
Preorder Predecessor 326
Preorder Successor 326
prevNthday 458
prevyearsameday 459
Prim's Algorithm 380
PrintAllParallelPaths 394
Priority Queue 298
Prism 78
push 247
push_back 189
push_back() 446
push_front() 446

Q Sequence 240
Q-tree 360
Quad Tree 360
quadratic congruent method 212
Quadratic Probing 435
querystring 87
Queue 284
Quick Sort 419
Quick Sort Performance 420

Radix Sort 426
Random Cipher Encoder 475
Random Number Generation using Recursion 212
Recursion 208
Recursive Definition 208
Regression Line on X or Y 23
Regula-Falsi Method 224
RemoveEdge 379, 387
RemoveVertex 388
resolvedow 456
Reverse Delete Algorithm 382
rewind() 492
Right Child 326
Right Sub Tree 325
Root 325
RSO 322

Saguaro Stack 265
Scheduling Appointments 309
searchQ 301
secant 226
Secant Method 226
SEEK_CUR 492
SEEK_END 492
SEEK_SET 492
Selection Sort Algorithms 416
SelfLoopCount 395
selfLoopVertices 389
SentenceCase 200
Separate Chaining 431, 432, 435
Serpinski Triangle 240
setsystemdate 465
Shell Sort 414
Shell Sort Performance 415
Shifting Property 210
showmenu 94
Siblings 326
Simson's Relation 210
Simulate 318
Single Linked List 106
Sisters 326
Sokoban 524
Sorting 400
Soundex 205
SoundexCode 205
Spatial Complexity 429
Spell Checking 376
Splay Tree 356
Splaying 356
split 184
Stable 429
Stack 247
Stacktop Element 249
startsWith 182
Stooge Sort 421
stpcpy() 170
Strassen's algorithm 30
strcat() 168
strchr() 176
strcmp() 169
strcmpi() 169
strcpy() 170
strcspn() 177
strdup() 178
Strictly Binary Tree 327

String Initialization 166
StringToUSD 480
strlen() 168
strlwr() 172
strncat() 169
strncmp() 169
strncmpi() 170
strncpy() 171
strnset() 175
Strong Binary Tree 328
strrev ( ) 174
strset() 175
strspn() 177
strstr() 176
strtok() 178
Structures 61
strupr() 172
Stupid Sort 425
Subsequence of a String 181
substring 190
Suffix of a String 180
Swap 121, 250
swap() 447
swap_head_tail() 447
Switchbox Routing Problem 258

tail 495
Tail Recursion 209
Tail Recursive Algorithm 211
TAK Function 216
the Bernoulli Triangle 234
Time 63
Time
  Complexity 429
  Complexity of Comb Sort 409
  Complexity of Stooge Sort 422
to_date 466
today() 464
Toeplitz matrix 38
ToggleCase 200
Tokenizer 44
tomonth 465
Tomorrow 450
Top Hash 444
Topological Sorting 384
Tower of Hanoi 217
Transpose of a Matrix 34
trc 503
Tree 325
Tree Sort 425
Trees 325
Trie 371
trimleft 190
trimright 190
trw 504
Turtles and Rabbits 407
Two Elements 250
Two-dimensional Array 3
typedef 61

Uncle 326
Union 385
Universal Hashing 433
Upper Triangular 35
USD Currency 479

Von Numann's Middle Squaring Method 213

Warshall's Algorithm 383
wcc() 493
wcl() 492
wcw() 493
Weak Binary Tree 328
What is Clustering 428
wheretokeep 314
wildCharMatch 183
wildgrep 502
WordFrequency 201
WordHistogram 201
wordWrap 187
Worst Case
  Performance of Straight Insertion Sort 411
  Bubble Sort Performance 404
  Comb Sort Performance 409
Wrapper Function 291

yesterday 451

Zag 356
Zig 356