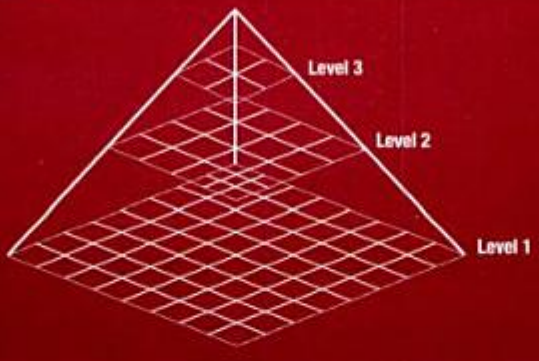
The McGraw-Hill Companies

SIGMA SERIES



Data Structures Using C

1000 Problems and Solutions

Sudipta Mukherjee

Tata McGraw-Hill

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

First reprint 2008
RCLQCRYXRADZX

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

The McGraw-Hill Companies

Contents

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 ggestion 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

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 10th element of an array, and then we will write `a[9] = 24`. As the array index starts from zero, the 10th 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]);
       }
 }
```

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.

```
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'))
            {
```

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++)
                          {
```

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];
       }
```

```
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

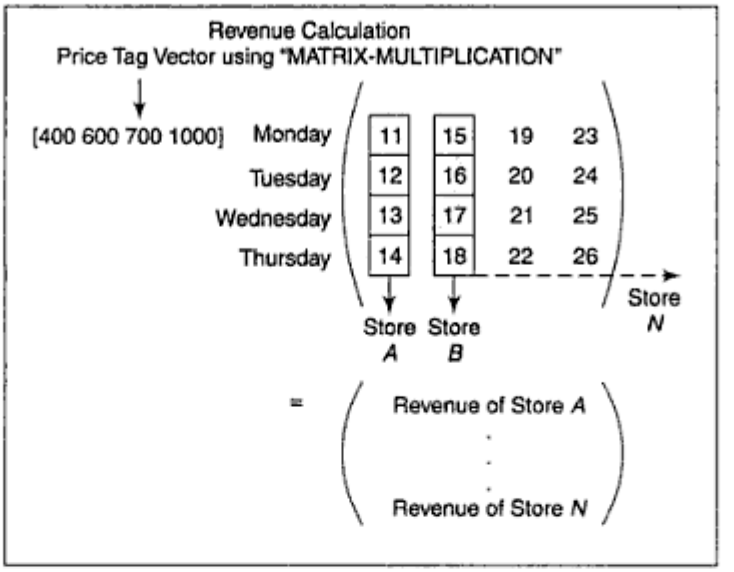


Fig. 1.4

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.

```
            //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.

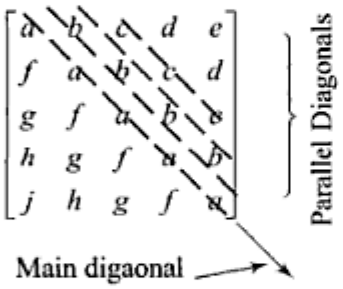


Fig. 1.7

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;
```

```
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);
```

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);
}
```

```
    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++)
```

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

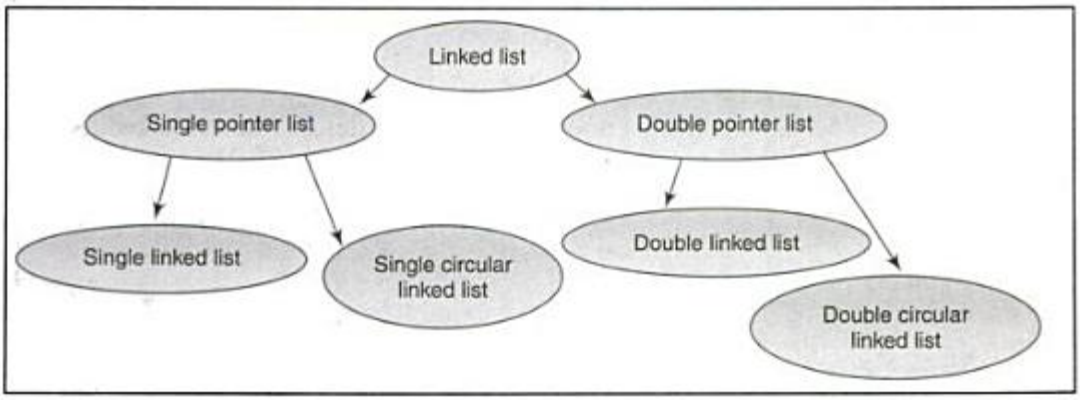


Fig. 3.1 *Classification of linked lists depending on number of pointers needed to implement*

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.

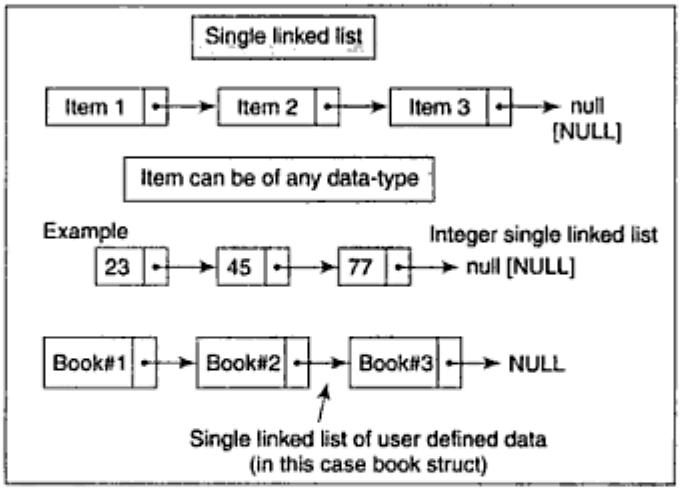


Fig. 3.2

```
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;
```

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;
```

```
        {
                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;
        }
```

```
        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
```

```
            && 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:

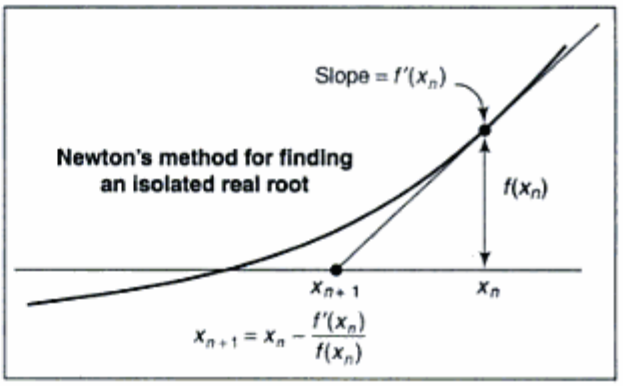


Fig. 5.3

```
#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()
```

```
#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.

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.

```
        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.

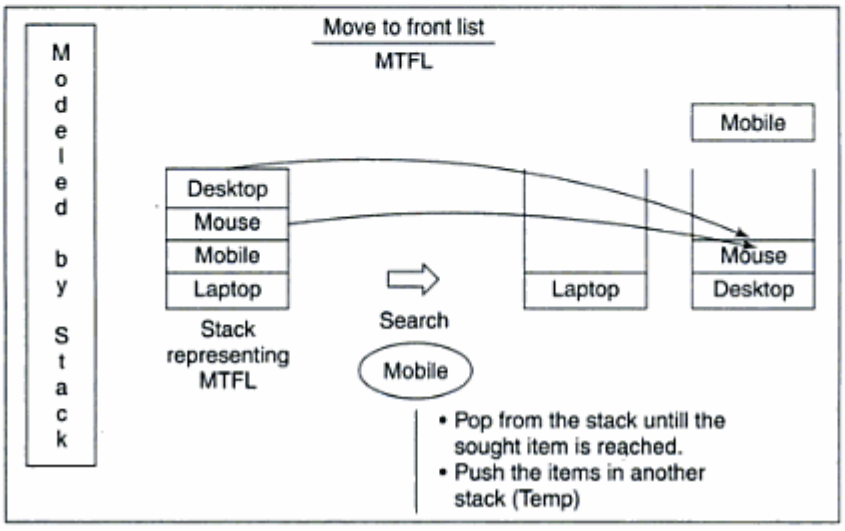


Fig. 6.12

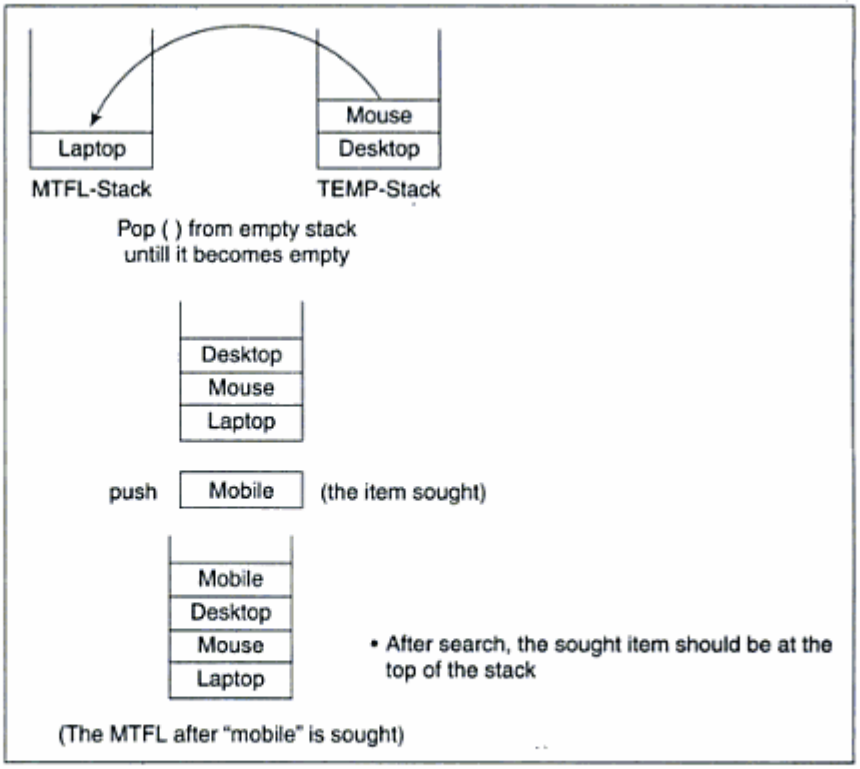


Fig. 6.13

```
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

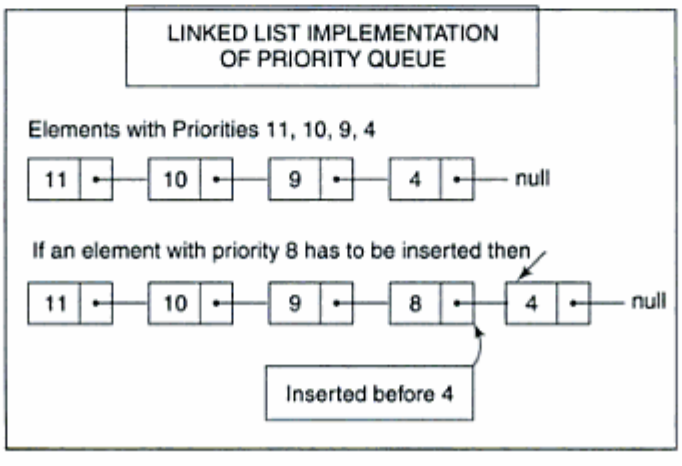


Fig. 7.12

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
```

```
//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 whereto keep(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;
                  }
```

```
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
```

```
QueueNode* push_front(QueueNode *h,pq info)
{
      QueueNode *p = (QueueNode *)malloc(sizeof(QueueNode));
      p->next = h;
      p->data = info;
      return p;
}
```

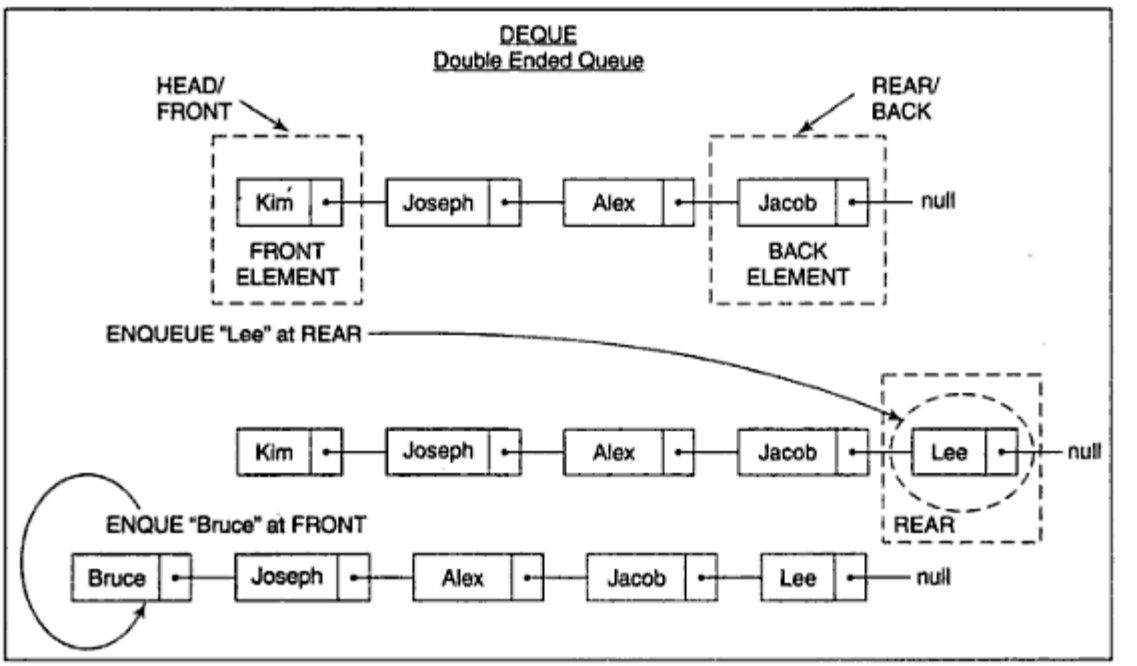


Fig. 7.13

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);
}
```

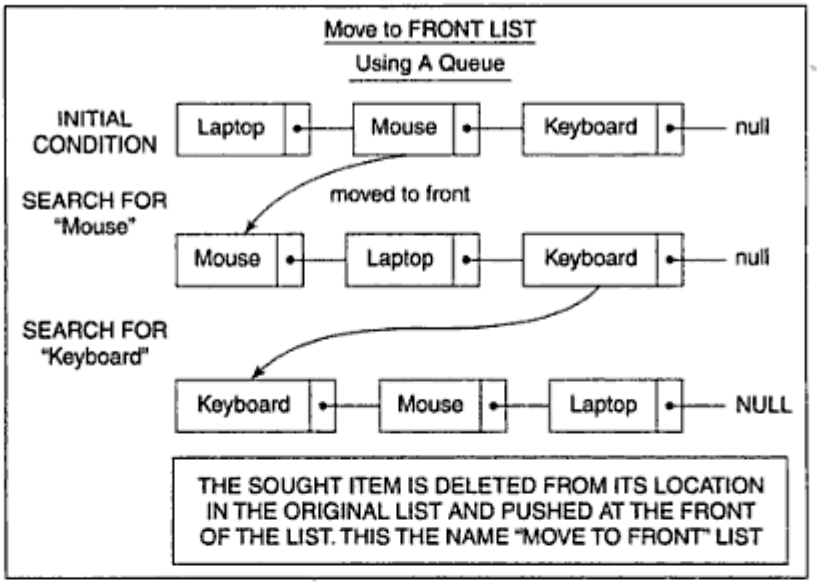


Fig. 7.14

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:

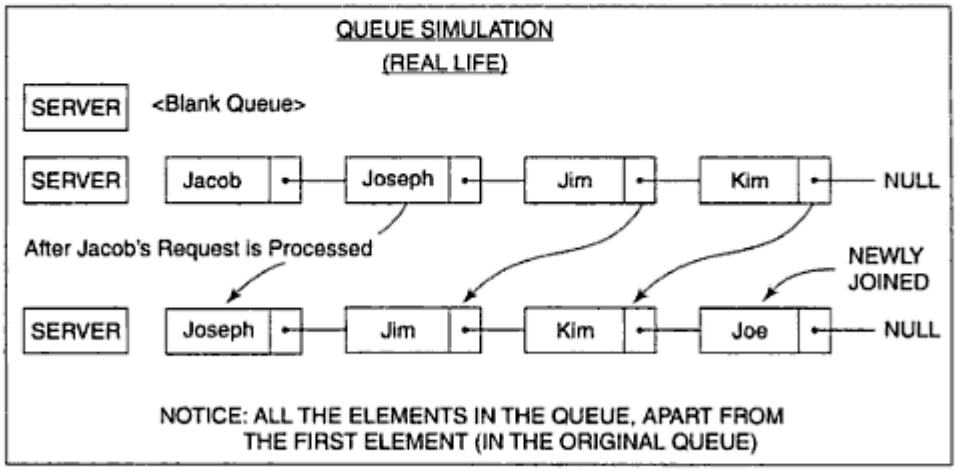


Fig. 7.15

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)
```

8

Trees

Explorer to Genetics!

INTRODUCTION

Some Key Facts about Trees and Jargons

A Binary Tree

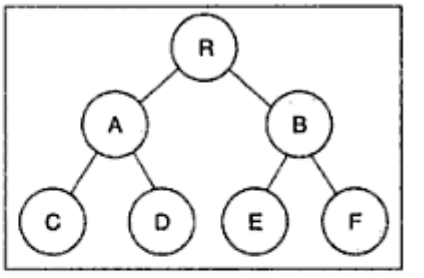


Fig. 8.1

- ***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

$$N = \sum_{k=0}^{H-1} 2\text{^}k$$

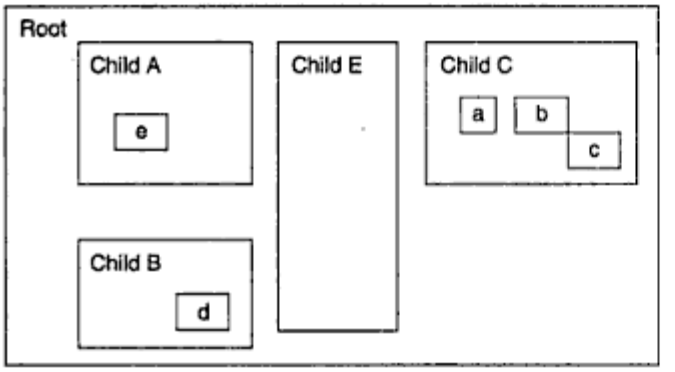
- For a binary tree of height h and nodes n $h <= n <= 2\text{^}h-1$

In the above picture a binary tree is shown. From the above picture we can conclude the following statements:

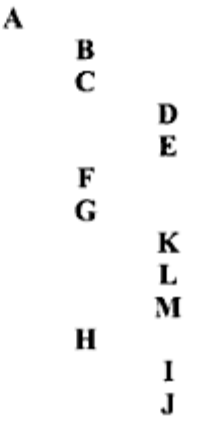
- R is the root of the tree.
- A is the left child of the tree.
- B is the right child of the tree.
- C is the left child of the node A.
- D is the right child of the node A.
- E is the left child of the node B.
- F is the right child of the node B.
- A and B are sisters/brothers/siblings.
- C and D 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



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.

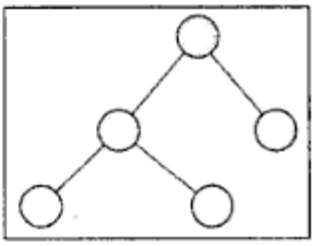


Fig. 8.2

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.

```
        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.

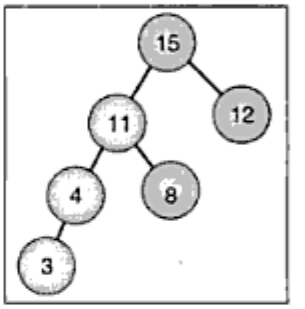


Fig. 8.20

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.

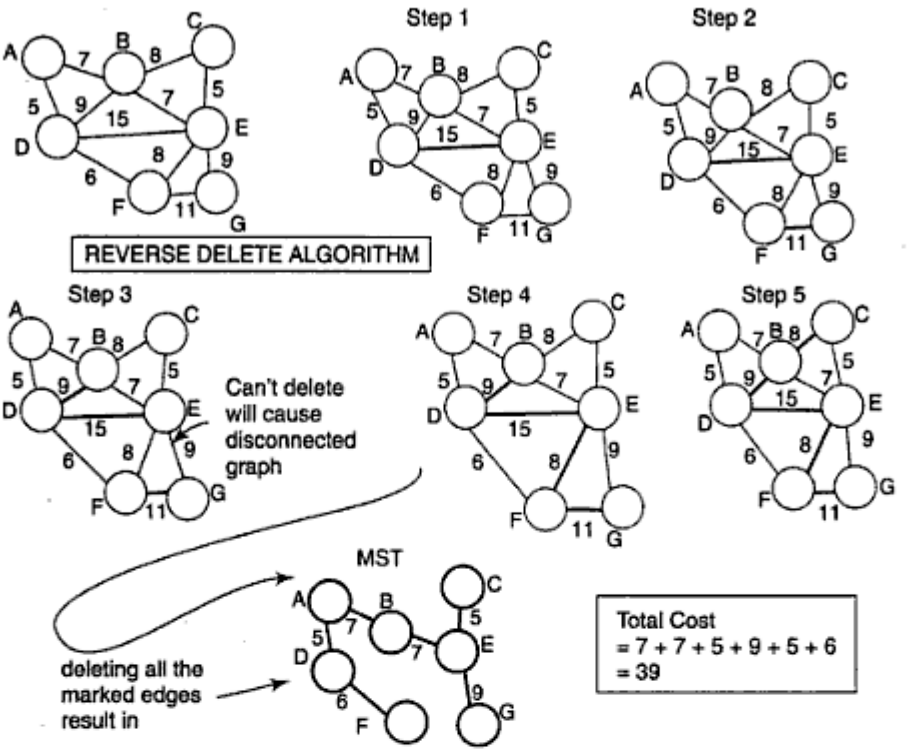
8.25 WHAT ARE THE DIFFERENT APPROACHES TO CREATE A HEAP?

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



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++)
```

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.*

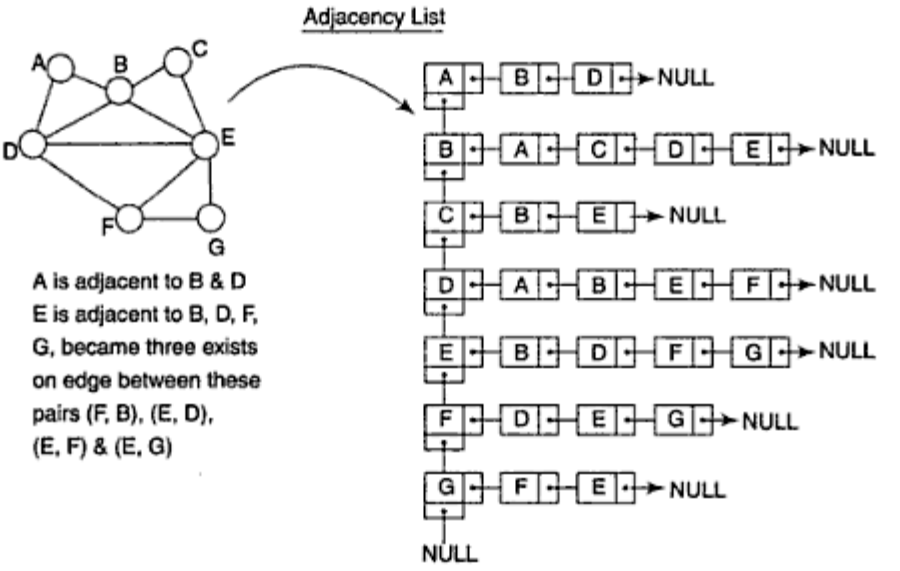
How to Find the Union of Two Graphs Represented using Matrix

```
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
```

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;
      }
}
```

Graphical Representation of Worst-Case Bubble Sort Performance

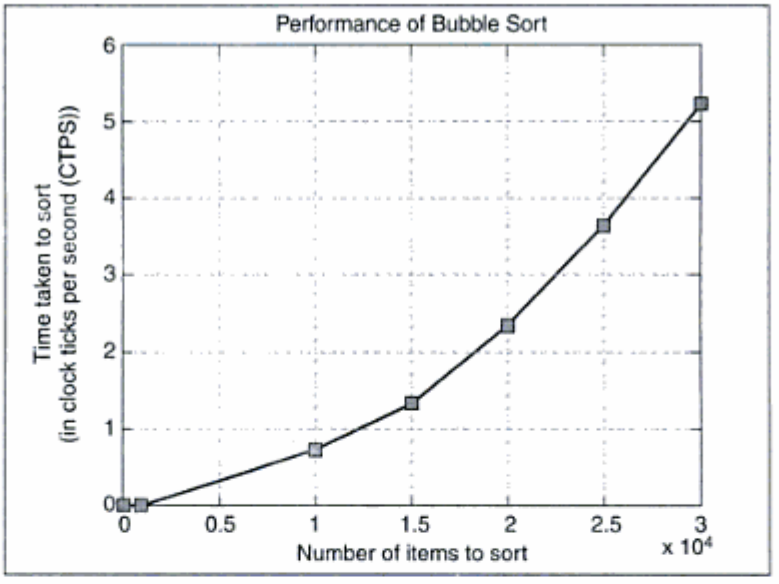


Fig. 10.3 *This graph shows the worst case performance of the switched bubble sort*

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.

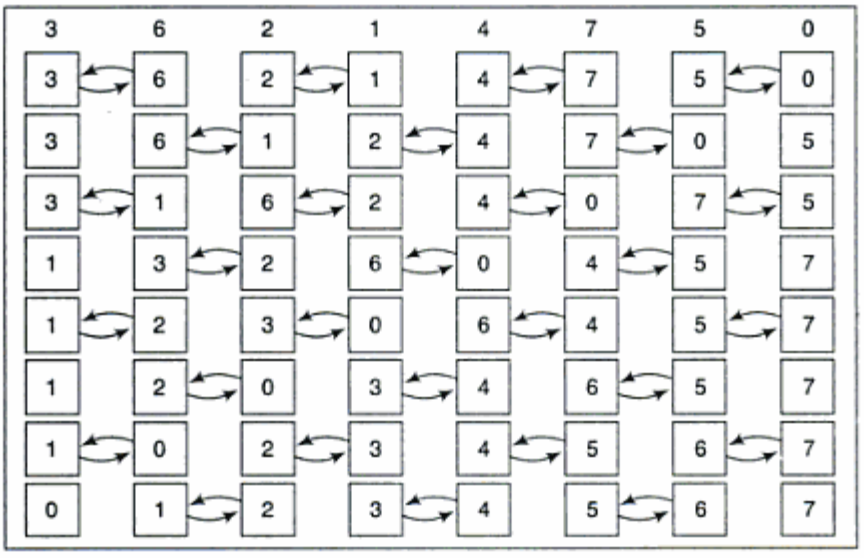


Fig. 10.4

```
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	$\Theta(n)$

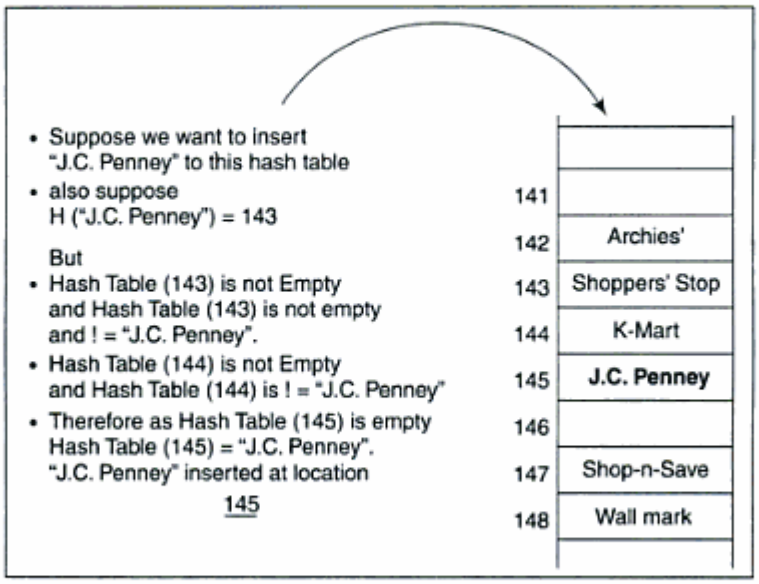


Fig. 11.1

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.

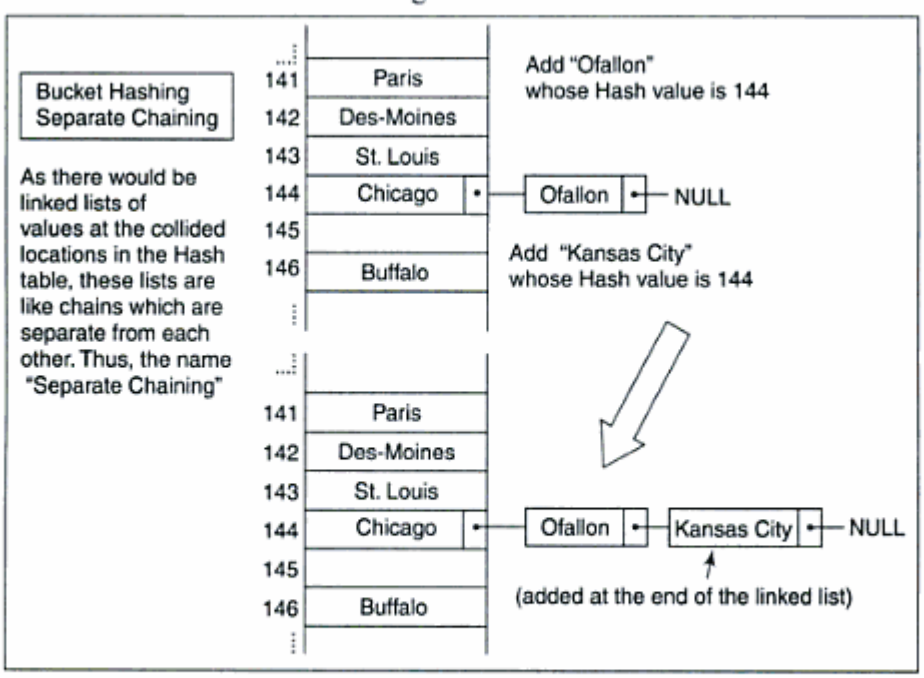


Fig. 11.2

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++;
```

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.

`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;
}
```

```
                    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--;
```

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.

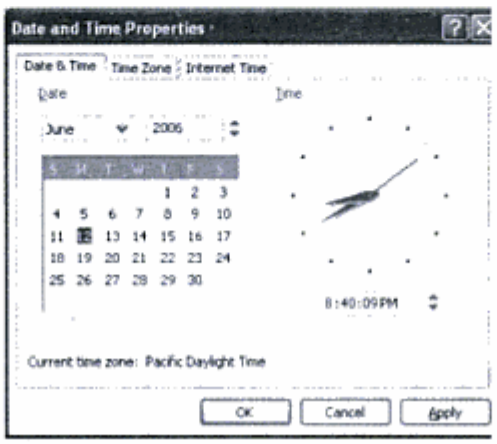
How to Show the Day of the Week in String

```
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
```

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
```

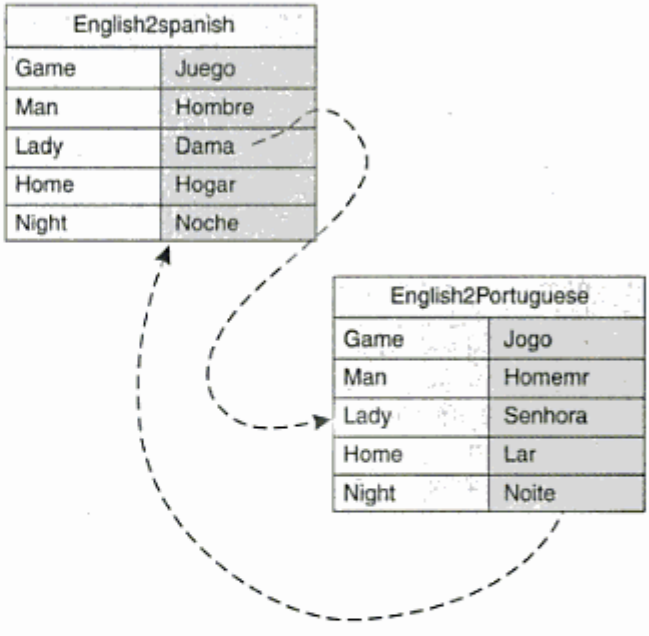
14.6 MULTILANGUAGE WORD MAP

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.**

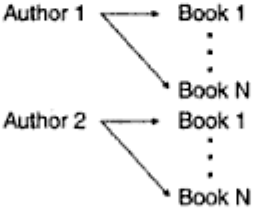
Try Yourself: *Create such a KIM of names and once a name is entered, return the name of the manager of the manager of the employee's name supplied.*

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.



One to Many Map

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;
```

```
        {
                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.

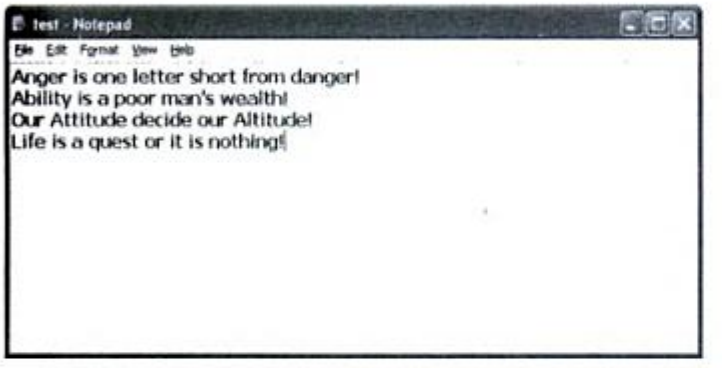
Here is an example.

```
#define USD2GBP 0.517505
#define USD3EUR 0.764435

GBP USD2GBP(USD amount)
{
      GBP temp;
```

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

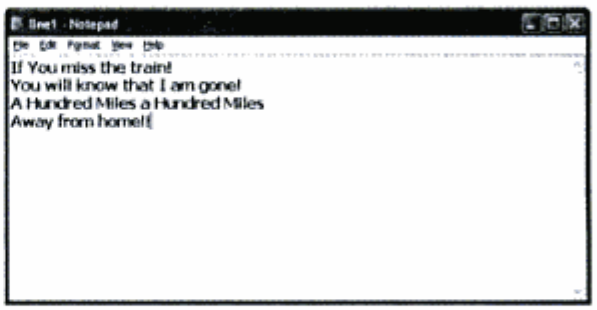
This function prints the first **number** of lines passed to it as a parameter.

```
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()
{
```

```
                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!
```

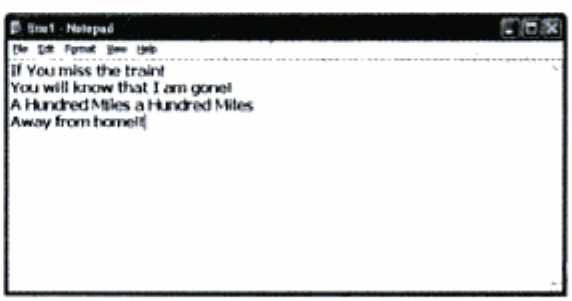
16.5 HOW TO SIMULATE UNIX Grep COMMAND FOR SWITCH –*V*

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,"");
```

the above function gives the following output

```
That I to manhood am arrived so near
To that same lot, however mean or high
```

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

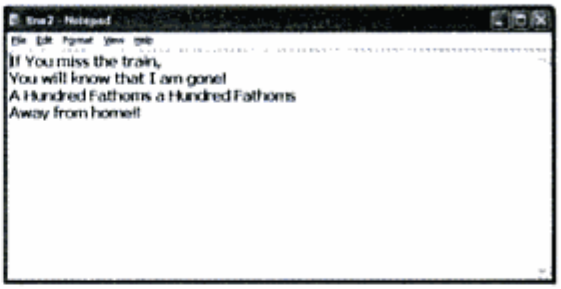
The strategy is to involve a temporary file that will hold the new character version of the old file. After the temporary file is created, we just rename it to the original file and delete the original file.

```
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))
```

```
            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.

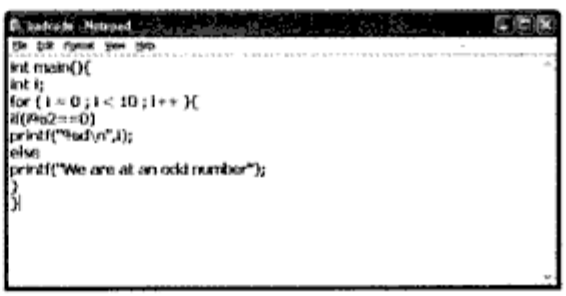
```
You will know that he is gone!
 # You will know that I am gone!
```

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():

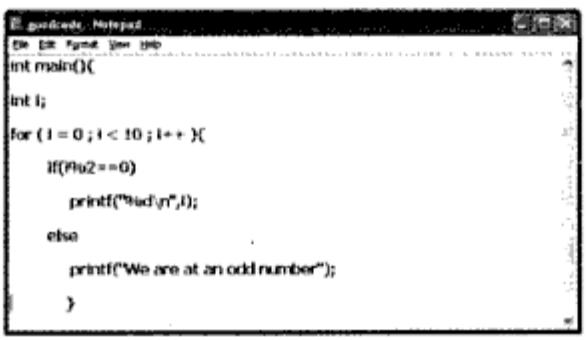
```
void diffv(char *file_1,char *file_2)
{
      char linef1[81];
      char linef2[81];
      FILE *fp1  = fopen(file_1,"r");
      FILE *fp2  = fopen(file_2,"r");
      while(!feof(fp1))
      {
            fgets(linef1,81,fp1);//reading the line from file 1
            fgets(linef2,81,fp2);//reading the line from file 2
            if(strcmpi(linef1,linef2)==0)//Are the lines same?
                  printf("%s = %s\n",linef1,linef2);

      }
}
```

```
        {
                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.

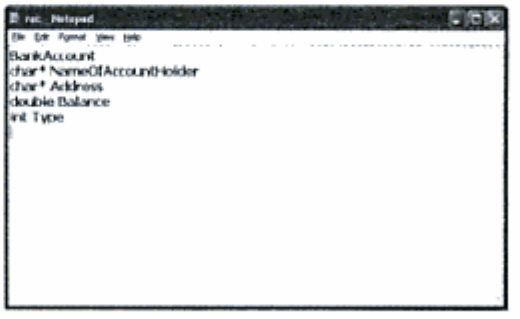


Example 16.6 ***Write a program to find out whether a C++ program uses STL or not?***

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

```
        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.

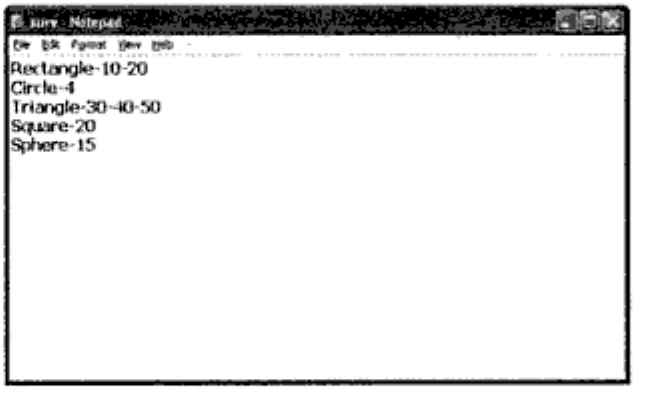


```
      //Sonnets are always of even number of lines. So there will be half
      //the number of lines matching sonnet lines. So for the perfect
      //sonnet like the above one count will always be half the total
      //number of lines. There are different styles of sonnets that might
      //have 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.***



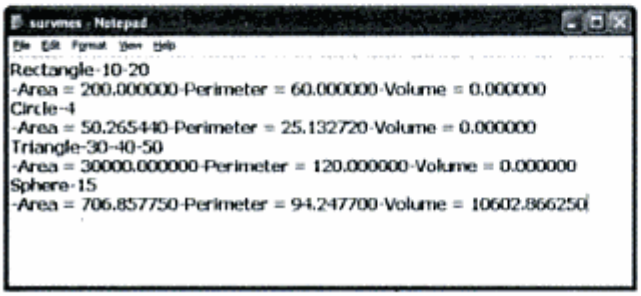
Solution The program will calculate the area, perimeter and volume of the shapes as and when applicable and list them on the console. The program should be able to handle Rectangle, Right Angled Triangle, Squares and Spheres. Here is a sample input file to the program.

```
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++;
```

```
        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.*

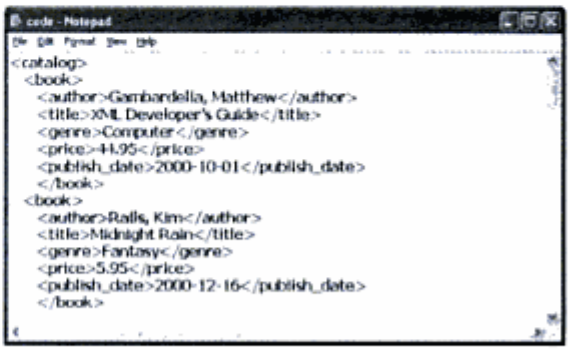
Example 16.11 ***Write a program to send a file to the connected printer.***

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;
```

```
                        //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++;
```

The McGraw-Hill Companies

SIGMA SERIES

Data Structures Using C

1000 Problems and Solutions

This book, meant for undergraduate course on Data Structures, presents numerous solved examples and unsolved problems for better understanding of the subject with greater clarity. Through updated coverage of this subject and simple language employed in this book, students will appreciate many of the practical aspects of Data Structures.

SALIENT FEATURES:

- Uses 'C' language.
- Detailed coverage of date, map and currency.
- Separate chapter on Abstract Data Types
- Includes newly emerging data structures like Saguaro Stack and emulation of DNA reactions

http://www.mhhe.com/mukherjee/ds

Visit us at : www.tatamcgrawhill.com

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

9 780070 667655

Tata McGraw-Hill