# LECTURE NOTES

## ON

## DATA STRUCTURES THROUGH C- LAB

## B.TECH MCA I YEAR I SEMESTER
## (JNTUA-R09)


**Mrs. N.HEMALATHA**
**ASST.PROFESSOR**




**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**
CHADALAWADA RAMANAMMA ENGINEERING COLLEGE
**CHADALAWADA NAGAR, RENIGUNTA ROAD, TIRUPATI (A.P) - 517506**

**JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR.**

**MCA Semester I** **Lab 3**

**(9F00106) DATA STRUCTURES THROUGH C- LAB**

Objectives:

- To make the student learn a programming language.
- To teach the student to write programs in C to solve typical problems.
- To introduce the student to simple linear data structures such as lists, stacks, queues.

**Recommended Systems/Software Requirements:**

- Intel based desktop PC with ANSI C Compiler and Supporting Editors

**Exercise l**.

**a)** Write a C program to find the sum of individual digits of a positive integer.

**b)** A Fibonacci sequence is defined as follows: the first and second terms in the sequence are 0 and 1. Subsequent terms are found by adding the preceding two terms in the sequence. Write a C program to generate the first n terms of the sequence.

**c)** Write a C program to generate all the prime numbers between 1 and n, where value of n is supplied.

**Exercise 2**.

**a)** Write a C program to calculate the following Sum:

Sum=$1-x^2/2! +x^4/4!-x^6/6!+x^8/8!-x^{10}/10!$

**b)** Write a C program to find the roots of a quadratic equation.

**c)** Write C program that uses both recursive and non-recursive functions

i) To find the factorial of a given integer.

ii) To find the GCD (greatest common divisor) of two given integers.

iii) To solve Towers of Hanoi problem.

**Exercise 3**

**a)** Write a C program to find both the largest and smallest number in a list of integers.

**b)** Write a C program that uses functions to perform the following:

i) Addition of Two Matrices ii) Multiplication of Two Matrices

**c)** Write a C program that uses functions to perform the following operations:

i) To insert a sub-string in to a given main string from a given position.

ii) To delete n Characters from a given position in a given string.

**d)** Write a C program to determine if the given string is a palindrome or not

**Exercise 4**

**a)** Write a C program that displays the position or index in the string S where the string T begins, or – 1 if S doesn't contain T.

**b)** Write a C program to count the lines, words and characters in a given text.

**c)** Write a C program to generate Pascal's triangle.

**d)** Write a C program to construct a pyramid of numbers.

**Exercise 5**

**a)** Write a C program which copies one file to another.

**b)** Write a C program to reverse the first n characters in a file.

(Note: The file name and n are specified on the command line.)

**c)** Write a C programme to display the contents of a file.

**d)** Write a C programme to merge two files into a third file ( i.e., the contents of the first file followed by those of the second are put in the third file)

**Exercise 6**
Write a C program that uses functions to perform the following operations.:
i) Creation ii) Insertion iii) Deletion iv) Traversal
on
**a)** singly linked list **b)** doubly linked list **c)** circular linked list

**Exercise 7**
**a)**Write C programs that implement stack (its operations) using
i) Arrays ii) Pointers
**b)**Write C programs that implement Queue (its operations) using
i) Arrays ii) Pointers

**Exercise 8**
Write a C program that uses Stack operations to perform the following:
i) Converting infix expression into postfix expression
ii) Evaluating the postfix expression

**Exercise 9**
Write a C program that implements the following sorting methods to sort a given list of integers in ascending order
i) Bubble sort
ii) Selection sort

**Exercise 10**
Write C program that implements the following sorting method to sort a given list of integers in ascending order:
i)Quick sort
ii)Merge sort

**Exercise 11**
Write C programs that use both recursive and non recursive functions to perform the following searching operations for a Key value in a given list of integers :
i) Linear search ii) Binary search

**Exercise 12**
Write C programs to implement the Lagrange interpolation and Newton- Gregory forward interpolation.
Write C programs to implement the linear regression and polynomial regression algorithms.

**Exercise 13**
Write C programs to create BST and perform operations on it.
Write C programs to implement recursive and non recursive Tree traversal techniques.

**Exercise 14**
Write C programs to implement Trapezoidal and Simpson methods.
Write C programs to implement Heap Sort.

**REFERENCES:**

1. **The Spirit of C, an introduction to modern programming**, M.Cooper, Jaico Publishing House.
2. **Mastering** C, K.R. Venugopal and S.R. Prasad, TMH Publications.
3. **Computer Basics and C Programming,** V. Rajaraman, PHI Publications

# Sum of Individual Digits of a Positive Integer

**1. a) Aim:** To write a C program to find the sum of individual digits of a positive integer.

**Program:**

```
#include<stdio.h>
#include<conio.h>
void main ()
{
        int n, sum=0, t;
        clrscr();
        printf("Enter a number: ");
        scanf("%d",&n);
        while( n != 0)
        {
                t = n%10;
                sum = sum + t;
                n = n/10;
         }
         printf("Sum of individual digits is: %d ",sum);
         getch();
}
```

**Output:**

```
Enter a number: 123
Sum of individual digits is: 6
```

# Generation of Fibonacci Sequence

**1. b) Aim:** A Fibonacci sequence is defined as follows: the first and second terms in the sequence are 0 and 1. Subsequent terms are found by adding the preceding two terms in the sequence. To write a C program to generate the first n terms of the sequence.

**Program:**

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int num1=0, num2=1,n,counter,fab;
        clrscr();
        printf("\nENTER LENGTH OF SERIES FOR 1 TO N : ");
        scanf("%d",&n);
        if(n<1)
                printf("\n THE FIBONACCI SERIES IS NOT POSSIBLE");
        else if(n==1)
        {
                printf("\n<------------------FIBONACCI SERIES------------");
                printf("\n %d",num1);
        }
        else if(n==2)
        {
                printf("\n<------------FIBONACCI SERIES---------->\n");
                printf("%d \t %d", num1,num2);
        }
        else
        {
                printf("\n <----------FIBONACCI SERIES------->\n");
                printf("%d \t %d",num1,num2);
                for(counter = 1; counter <= n-2; counter++)
                {
                        fab=num1 + num2;
                        printf("\t%d",fab);
                        num1=num2;
                        num2=fab;
                }
        }
        getch();
}
```

**Output:**

```
ENTER LENGTH OF SERIES FOR 1 TO N : 5
<-----------FIBONACCI SERIES--------->
0       1       1       2       3
```

# Generation of Prime Numbers

**1. c) Aim:** To Write a C program to generate all the prime numbers between 1 and n, where n is a value supplied by the user.

**Program:**

```
#include<stdio.h>
#include<conio.h>
void main ()
{
        int i, n, j, c = 0;
        clrscr();
        printf("Enter n value : ");
        scanf("%d",&n);
        printf("\nThe prime numbers are \n");
        for(i=2;i<=n;i++)
        {
                c = 0;
                for ( j=2;  j<i; j++)
                {
                        if (i%j == 0)
                                c = 1;
                }
                 if ( c == 0)
                {
                        printf("\t%d ",i);
                 }
        }
        getch();
}
```

**Output:**

```
Enter n value : 10
        2       3       5       7
```

# Calculating the Sum of given Series

**2. a) Aim:** To write a C program to calculate the following Sum:
Sum=1-$x^2$/2! +$x^4$/4!-$x^6$/6!+$x^8$/8!-$x^{10}$/10!

**Program:**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
        int i,j;
        float x,y,v,f=1,sum=1;
        clrscr();
        printf("Enter X value: ");
        scanf("%f",&x);
         for(i=1;i<=5;i++)
         {
                y=pow(-1,i)*pow(x,2*i);
                for(j=1;j<=2*i;j++)
                {
                        f=f*j;
                       v=y/f;
                }
                sum=sum+v;
        }
        printf("The sum of series is %f: ",sum);
        getch();
}
```

**Output:**

```
Enter X value: 2
The sum of series is: -0.668518
```

# Roots of a Quadratic Equation

**2. b) Aim:** To write a C program to find the roots of a quadratic equation.

**Program:**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
        float a,b,c,root1,root2,dis;
        clrscr();
        printf("\nEnter values of a,b,c : ");
        scanf("%f%f%f",&a,&b,&c);
        dis=b*b-4*a*c;
        if(dis>0)
        {
                root1=-b+sqrt(dis)/2*a;
                root2=-b-sqrt(dis)/2*a;
                printf("\n*****ROOTS ARE REAL & UNEQUAL*****\n");
                printf("\n root1=%f\n root2=%f",root1,root2);
        }
        else if (dis==0)
        {
                root1=root2=-b/2*a;
                printf("\n*****ROOTS ARE REAL & EQUAL*****\n");
                 printf("\nroot1=%f\n root2=%f",root1,root2);
        }
        else
                printf("\n Imaginary Roots.");
        getch();
}
```

**Output:**

```
Case 1: Enter values of a,b,c :  1       2        1

*****ROOTS ARE REAL & EQUAL*****

root1=-1.000000
root2=-1.000000
Case 2: Enter values of a,b,c :  1       8        2

*****ROOTS ARE REAL & UNEQUAL*****

root1=-4.258343
root2=-11.741657
```

# Factorial of a given Integer

**2. c) Aim:** To write a c program to find out factorial of a given number using non recursive and recursive functions

**Program:**

```
#include<stdio.h>
#include<conio.h>
int fact_nonrec(int  );
int fact_rec(int  );
void main()
{       int a,ch,f;
        clrscr();
        do
        {       printf("\n1.Factorial using non recursion");
                printf("\n2.Factorial using recursion");
                printf("\n3.Exit");
                printf("\nEnter your choice:");
                scanf("%d",&ch);
                switch(ch)
                {       case 1: printf("\nEnter a number:");
                                scanf("%d",&a);
                                f=fact_nonrec(a);
                                printf("\nFactorial of a given nmber is:%d",f);
                                break;
                        case 2: printf("\nEnter a number:");
                                scanf("%d",&a);
                                f=fact_rec(a);
                                printf("\nFactorial of a given nmber is:%d",f);
                                break;
                        default:printf("\nProgram terminated");
                                break;
                }
        }while(ch>=1&&ch<3);
        getch();
}
int fact_nonrec(int a)
{       int f=1,i;
        for(i=1;i<=a;i++)
        {       f=f*i;
        }
        return(f);
}
int fact_rec(int n)
{       if(n<1)
                return 1;
        else
                 return(n*fact_rec(n-1));
}
```

**OUTPUT:-**

1.Factorial using non recursion
2.Factorial using recursion
3.Exit
Enter your choice:1
Enter a number:5
Factorial of a given nmber is:120

1.Factorial using non recursion
2.Factorial using recursion
3.Exit
Enter your choice:2
Enter a number:5
Factorial of a given nmber is:120

1.Factorial using non recursion
2.Factorial using recursion
3.Exit
Enter your choice:3
Program terminated

# GCD of Two given Integers

**2. d) Aim:** To write a c program to find out GCD of given two numbers using non recursive and recursive functions

**PROGRAM:-**

```
#include<stdio.h>
#include<conio.h>
void main()
{       int a,b,c,ch;
        clrscr();
        do
        {       printf("\n1.Gcd using non recursion");
                printf("\n2.Gcd using recursion");
                printf("\n3.Exit");
                printf("\nEnter your choice:");
                scanf("%d",&ch);
                switch(ch)
                {       case 1: printf("\nEnter a and b:\n");
                                scanf("%d%d",&a,&b);
                                c=gcd_nonrec(a,b);
                                printf("\nGCD of given two numbrs is:%d",c);
                                break;
                        case 2: printf("\nEnter a and b:\n");
                                scanf("%d%d",&a,&b);
                                c=gcd_rec(a,b);
                                printf("\nGCD of given two numbrs is:%d",c);
                                break;
                        default:printf("\nProgram terminated");
                                break;
                }
        }while(ch>=1&&ch<3);
        getch();
}
int gcd_nonrec(int a,int b)
{       int c,i;
        if(a>b)
        {       for(i=1;i<=a;i++)
                {       if(a%i==0&&b%i==0)
                                c=i;
                }
        }
        else
        {
                for(i=1;i<=b;i++)
                {       if(a%i==0&&b%i==0)
                        {
                                c=i;
                        }
                }
        }
```

```
        return(c);
}
int gcd_rec(int m,int n)
{
        if(n>m)
        {
                return (gcd_rec(n,m));
        }
        else if(n==0)
        {
                return (m);
        }
        else
        {
                return (gcd_rec(n,m%n));
        }
}
```

**OUTPUT:**

```
1.Gcd using non recursion
2.Gcd using recursion
3.Exit
Enter your choice:1
Enter a and b:
2       8

GCD of given two numbrs is:2
1.Gcd using non recursion
2.Gcd using recursion
3.Exit

Enter your choice:2
Enter a and b:
28      56
GCD of given two numbrs is:28

1.Gcd using non recursion
2.Gcd using recursion
3.Exit
Enter your choice:3
Program terminated
```

# Towers of Hanoi

**2. e) AIM:-** To write a c program for Towers of Hanoi

**PROGRAM:-**

```
#include<stdio.h>
#include<conio.h>
void towers(int,char,char,char);
int main(void)
{
        int n;
        clrscr();
        printf("\nEnter no.of disks:");
        scanf("%d",&n);
        towers(n,'s','d','t');
        getch();
}
void towers(int n,char source,char dest,char temp)
{
        static int step=0;
        if(n==1)
                printf("\nStep%d:move from %c to %c",++step,source,dest);
        else
        {
                towers(n-1,source,temp,dest);
                printf("\nstep%d:move from %c to %c",++step,source,dest);
                towers(n-1,temp,dest,source);
        }

}
```

**OUTPUT:-**

```
Enter no.of disks:3

Step1:move from s to d
step2:move from s to t
Step3:move from d to t
step4:move from s to d
Step5:move from t to s
step6:move from t to d
Step7:move from s to d
```

# Largest and Smallest of Numbers

**3. a) Aim:** To write a c program to find minimum and maximum value from the given input.

**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
void  main()
 {
        int a[20],i,n,min,max;
        clrscr();
        printf("Enter Size of Array: ");
        scanf("%d",&n);
        printf("Enter Elements into Array: ");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
         }
        max=0;
        for(i=0;i<n;i++)
        {
                if(a[i]<min)
                        min=a[i];
                if(a[i]>max)
                        max=a[i];
        }
        printf(" min is : %d \n max is : %d",min,max);
        getch();
}
```

**Output:**

```
Enter Size of Array: 5
Enter Elements into Array: 10          20       30       40       50
min is : 10
max is : 50
```

# Matrix Operations

**3. b) Aim:** To write a c program to perform matrix addition, subtraction and multiplication by using functions

**Program:**

```
#include<stdio.h>
#include<conio.h>
int a[10][10],b[10][10],c[10][10],i,j,k,r1,r2,c1,c2,ch;
void main()
{
        do
        {       clrscr();
                printf("\n1.Matrix addition");
                printf("\n2.Matrix subtraction");
                printf("\n3.Matrix multiplication");
                printf("\n4.Exit");
                printf("\nEnter your choice");
                scanf("%d",&ch);
                switch(ch)
                {       case 1: printf("\nEnter no.of rows and columns");
                                scanf("%d%d",&r1,&c1);
                                printf("\nEnter %dx%d elements into matrix-A",r1,c1);
                                mat_read(a,r1,c1);
                                printf("\nEnter no.of rows and columns");
                                scanf("%d%d",&r2,&c2);
                                printf("\nEmter%dx%d elements into matrix-B",r2,c2);
                                mat_read(b,r2,c2);
                                clrscr();
                                printf("\n%dx%d matrix-A is:",r1,c1);
                                mat_disp(a,r1,c1);
                                printf("\n%dx%d matrix-B is:",r2,c2);
                                mat_disp(b,r2,c2);
                                mat_add(a,b,r1,c1);
                                printf("\nAddition of matrices is:");
                                mat_disp(c,r1,c1);
                                break;
                        case 2: printf("\nEnter no.of rows and columns");
                                scanf("%d%d",&r1,&c1);
                                printf("\nEnter %dx%d elements into matrix-A",r1,c1);
                                mat_read(a,r1,c1);
                                printf("\nEmter%dx%d elements into matrix-B",r1,c1);
                                mat_read(b,r1,c1);
                                clrscr();
                                printf("\n%dx%d matrix-A is:",r1,c1);
                                mat_disp(a,r1,c1);
                                printf("\n%dx%d matrix-B is:",r1,c1);
                                mat_disp(b,r1,c1);
                                mat_sub(a,b,r1,c1);
                                printf("\nSubtraction of matrices is:");
                                mat_disp(c,r1,c1);
```

```
                        break;
                case 3: printf("\nEnter no.of rows and columns of matrix_A");
                        scanf("%d%d",&r1,&c1);
                        printf("\nEnter no.of rows and columns of matrix_B");
                        scanf("%d%d",&r2,&c2);
                        if(c1!=r2)
                        {
                                printf("\nMatrix multiplication is not possiable");
                        }
                        else
                        {
                                printf("\nEnter %dx%d elements into matrix-A",r1,c1);
                                mat_read(a,r1,c1);
                                printf("\nEmter%dx%d elements into matrix-B",r2,c2);
                                mat_read(b,r2,c2);
                                clrscr();
                                printf("\n%dx%d matrix-A is:",r1,c1);
                                mat_disp(a,r1,c1);
                                printf("\n%dx%d matrix-B is:",r2,c2);
                                mat_disp(b,r2,c2);
                                mat_mul(a,b,r1,c1,r2,c2);
                                printf("\nMultiplication of matrices is:");
                                mat_disp(c,r1,c2);
                                break;
                        }
                }
                getch();

        }while(ch>0&&ch<4);
}
mat_read(int temp[10][10],int row,int col)
{       for(i=0;i<row;i++)
        {
                for(j=0;j<col;j++)
                {
                        scanf("%d",&temp[i][j]);
                }
        }
        return;
}
mat_disp(int temp[10][10],int row,int col)
{       for(i=0;i<row;i++)
        {
                for(j=0;j<col;j++)
                {
                        printf("%d\t",temp[i][j]);
                }
                printf("\n\t\t");
        }
        return;
}
```

```
mat_add(int a[10][10],int b[10][10],int r1,int c1)
{       for(i=0;i<r1;i++)
        {
                for(j=0;j<c1;j++)
                {
                        c[i][j]=a[i][j]+b[i][j];
                }
        }
        return;
}
mat_sub(int a[10][10],int b[10][10],int r1,int c1)
{
        for(i=0;i<r1;i++)
        {
                for(j=0;j<c1;j++)
                {
                        c[i][j]=a[i][j]-b[i][j];
                }
        }
        return;
}
mat_mul(int a[10][10],int b[10][10],int r1,int c2,int r2)
{       for(i=0;i<r1;i++)
        {
                for(j=0;j<c2;j++)
                {
                        c[i][j]=0;
                for(k=0;k<r2;k++)
                        c[i][j]+=a[i][k]*b[j][k];
                }
        }
        return;
}
```

**OUTPUT:-**

```
1.Matrix addition
2.Matrix subtraction
3.Matrix multiplication
4.Exit
Enter your choice1
Enter no.of rows and columns: 2        2

Enter 2x2 elements into matrix-A
1
2
3
4
Enter no.of rows and columns: 2        2

Enter 2 x 2 elements into matrix-B
5
6
```

```
7
8

2 x 2 matrix-A is:1       2
                  3       4

2 x 2 matrix-B is:5       6
                  7       8

Addition of matrices is:6       8
                        10      12
```

# Inserting a Sub-String into Main String

**3. c) Aim:** To write a c program to insert a given string into the main string.

**PROGRAM:-**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
        char str[20],str1[20],str2[20]="";
        int n;
        clrscr();
        printf("Enter string: ");
        gets(str);
        printf("Enter second string: ");
        gets(str1);
        printf("Enter the position where the substring is to be inserted: ");
        scanf("%d",&n);
        strncpy(str2,str,n);
        strcat(str2,str1);
        strcat(str2,str+n);
        puts(str2);
        getch();
}
```

**Output:**

```
Enter string: Siva Reddy
Enter second string: Kumar
Enter the position where the substring is to be inserted: 5
Siva KumarReddy
```

# Deleting Characters in a String

**3. d) Aim:** To write a c program to delete a n characters in a given string.

**PROGRAM:-**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
        char str[20],str2[20]="";
        int n,pos;
        clrscr();
        printf("Enter string: ");
        gets(str);
        printf("Enter position: ");
        scanf("%d",&pos);
        printf("Enter number of characters to be deleted: ");
        scanf("%d",&n);
        strncpy(str2,str,pos);
        strcat(str2,str+pos+n);
        puts(str2);
        getch();
}
```

**Output:**

```
Enter string: Venkataramana
Enter position: 0
Enter number of characters to be deleted: 7
ramana
```

# String Palindrome

**3. e) Aim:** To write a C program to determine if the given string is a palindrome or not

**Program:**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
        char s1[20],s2[20];
        clrscr();
        printf("Enter a String : ");
        gets(s1);
        strcpy(s2, s1);
        strrev(s1);
        printf("\nThe reversed string is : %s", s1);
        if ( strcmp(s1,s2) == 0)
                printf("\nThe given string is palindrome");
        else
                printf("\nThe given string is not a palindrome");
        getch();
}
```

**Output:**

```
Enter a String : madam
The reversed string is : madam
The given string is palindrome
```

# Displaying the position or index in the String

**4. a) Aim:** To display the position or index of the string S where the string T begins, or -1 if S doesn't contain T.

**Program:**

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
void main()
{
        char s[30], t[20];
        char *found;
        clrscr();
        puts("Enter the first string: ");
        gets(s);
        puts("Enter the string to be searched: ");
        gets(t);
        found=strstr(s,t);
        if(found)
                printf("Second String is found in the First String at %d position.\n",found-s);
        else
                printf("-1");
        getch();
}
```

**Output:**

```
Enter the first string: Kiran Kumar
Enter the string to be searched: ran
Second String is found in the First String at 2 position.
```

# Counting the lines, words and characters in a given Text

**4. b) Aim:** To count the lines, words and characters in a given text.

**Program:**

```
#include<conio.h>
#include<string.h>
#include<stdio.h>
void main()
{
        char ch;
        int line=0,chars=0, words=0;
        clrscr();
        printf(" Enter text : \n");
        while((ch=getchar())!='@')
        {

                chars++;
                if(ch=='\n')
                        line++;
                if(ch=='\t' || ch=='\n'|| ch==' ')
                        words++;

        }
        printf(" no of lines are %d\n  no of words are %d \n no of characters are %d",line,words,
chars);
        getch();
 }
```

**Output:**

```
Enter text :
This is a C Program
@
no of lines are : 1
no of words are: 5
no of characters are: 20
```

# Pascal's Triangle

**4. c) Aim:** To generate and display Pascal triangle.

**Program:**

```
#include<stdio.h>
#include<conio.h>
void main ()
{
        int a[30][30],i,j,k,n,spec=25;
        clrscr();
        printf("Enter the value of n : ");
        scanf("%d",&n);
        for ( i=0; i<n; i++ )
        {
                for ( k=(spec-(2*i)); k>0;  k-=2 )
                {
                        printf(" ");
                }
                for ( j=0; j<=i; j++ )
                {
                        if ( j==0 || j==i )
                                a[i][j]=1;
                        else
                                a[i][j]=a[i-1][j-1]+a[i-1][j];
                                printf("%d ",a[i][j]);
                }
                printf("\n");
          }
        getch();
}
```

**Output:**

```
Enter the value of n : 3
          1
       1     1
    1     2     1
```

# Pyramid of Numbers

**4. d) Aim:** To generate and display a pyramid of numbers.

**Program:**

```
#include<stdio.h>
#include<conio.h>
void pyramid ( int );
void main ()
 {
        int n;
        clrscr();
        printf(" Enter the value of n : ");
        scanf("%d",&n);
        pyramid(n);
        getch();
}
void pyramid ( int n )
{
        int i,j;
        for ( i=0; i<n; i++ )
        {
                for ( j=n-1; j>i; j-- )
                        printf("  ");
                for ( j=i; j>=0; j-- )
                        printf(" %d",j);
                for ( j=1; j<=i; j++ )
                        printf(" %d",j)
                printf("\n");
        }
}
```

**Output:**

```
Enter the value of n : 4
     0
   101
  21012
3210123
```

# Copying a File

**5. a) Aim:** To write a C program which copies one file to another.

**Program:**

```
#include<stdio.h>
void main()
{
	FILE *ft, *fs;
	fs = fopen("a.txt", "r");
	ft = fopen("b.txt", "w");
	if( fs== NULL)
	{
		printf("\nSource File Opening Error");
		exit(1);
	}
	if( ft== NULL)
	{
		printf("\nTarget File Opening Error");
		exit(1);
	}
	while( !feof(fs) )
	{
		fputc(fgetc(fs), ft);
	}
	printf("\nSource file copied to Target file successfully...");
	fcloseall();
	getch();
}
```

**Output:**

Source file copied to Target file successfully...

# Reversing the first n characters in a File

**5. b) AIM:-** To write a C program to reverse the first n characters in a file.

**PROGRAM:-**

```
#include<stdio.h>
#include<conio.h>
void main()
{       FILE *fp;
        int n;
        char ch;
        clrscr();
        fp=fopen("text1.dat","r");
        if( fp == NULL)
        {
                printf("\nSource File Opening Error");
                exit(1);
        }
        printf("\nEnter number of characters to reverse : ");
        scanf("%d", &n);
        printf("\nThe reverse order of the content: ");
        while(n>=0)
        {
                fseek(fp,n,0);
                printf("%c\t",fgetc(fp));
                n= n-2;
        }
        fclose(fp);
        getch();
}
```

**OUTPUT:-**

Enter number of characters to reverse :Original content of the file: 7
The reverse order of the content: g f e d c b a

# Displaying the contents of a file

**5. c) Aim:** To write a C program to display the contents of a file.

**Program:**

```
#include<stdio.h>
 #include<conio.h>
 void main()
 {
        FILE *fp;
        char ch;
        clrscr();
        fp=fopen("text1.dat","r");
        if( fp == NULL)
        {
                printf("\nSource File Opening Error");
                exit(1);
        }
        while(  (ch= fgetc(fp) ) !=  EOF)
        {
                printf("%c", ch);
        }
        fclose(fp);
        getch();
 }
```

**Output:**

```
This is first line
This is second line
This is last line
```

# Merging two Files

**5.d) AIM**:- To write a c program to merge two files into another file

**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
void main()
{
        FILE *fp1,*fp2,*fp3;
        char ch;
        clrscr();
        fp1=fopen("text1.dat","r");
        fp3=fopen("text3.dat","w");
        while(!feof(fp1))
        {
            ch=getc((fp1));
            putc(ch,fp3);
        }
        fclose(fp1);
        fclose(fp3);
        fp2=fopen("text2.dat","r");
        fp3=fopen("text3.dat","a");
        while(!feof(fp2))
        {
            ch=getc((fp2));
            putc(ch,fp3);
        }
        fclose(fp2);
        fclose(fp3);
        fp3=fopen("text3.dat","r");
        printf("\ncontent of the merged file:");
        while(!feof(fp3))
        {
            printf("%c",getc(fp3));

        }
        fclose(fp3);
}
```

**OUTPUT:-**

```
Enter content of file1:abcd.
Enter content of file2:efgh.
content of file1:abcd
content of file2:
efgh
content of the merged file:abcd
efgh
```

# Operations on Singly Linked List

**6. a) AIM:-** To write a c program that uses functions to perform all possible operations on singly linked list.

**PROGRAM:-**

```
#include<stdio.h>
#include<alloc.h>
typedef struct llist
{
    int info;
    struct llist *link;
}stype ;
stype *head;
int len= 0;
void main()
{
        int ch=1, pos , elem;
        while(ch>0 && ch < 7)
        {       clrscr();
                printf("Linked List");
                printf("\n1.Initializing list to empty");
                printf("\n2.Determining whether the list is empty or not");
                printf("\n3.Displaying length of the list");
                printf("\n4.Inserting given element at particular position of the list");
                printf("\n5.Deleting the given element from the list");
                printf("\n6.Traversing the list elements");
                printf("\n7.Exit");
                printf("\nEnter your choice...");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1 :  initialize_list();
                                break;
                        case 2 : empty_list();
                                break;
                        case 3 : display_list();
                                break;
                        case 4 : printf("\nEnter element to be inserted : ");
                                scanf("%d",&elem);
                                printf("\nEnter position to be inserted : ");
                                scanf("%d",&pos);
                                insert_list(elem,pos);
                                break;
                        case 5 : printf("\nEnter position for deletion : ");
                                scanf("%d",&pos);
                                delete_list(pos);
                                break;
                        case 6 : traverse_list();
                                break;
                        default : printf("\nProgram Terminated");
```

```
                                        break;
                        }
                        getch();
                }
        }
initialize_list()
{       head = NULL;
        len = 0;
        printf("\nList is initialized");
        return;
}
empty_list()
{       if (len == 0)
                printf("\nList is empty");
        else
                printf("\nList is not empty");
        return;
}
display_list()
{
        printf("\nThe length of the list is : %d",len);
        return;
}
insert_list(int elem,int pos)
{       stype *node,*p,*q;
        int i;
        if(pos<1 || pos>len+1)
        {
                printf("\nThe position is invalid, hence no element to be inserted");
                return;
        }
        node = (stype *)malloc(sizeof(stype));
        node->info = elem;
        if ( pos == 1)
        {
                node->link = head;
                head = node;
        }
        else
        {
                p = head;
                for(i=1;i<=pos-2;i++)
                        p = p->link;
                q = p->link;
                p->link = node;
                node->link = q;
        }
        len = len + 1;
        printf("\nElement inserted successfully");
        return;
}
```

```
delete_list(int pos)
{
        stype *temp,*p,*q;
        int i;
        if ( len == 0)
        {       printf("\nSingly Linked List is empty");
                return;
        }
        if( pos <1  || pos > len)
        {
                printf("\nPostion is invalid");
                return;
        }
        if (pos == 1)
        {
                temp = head;
                head = head->link;
                free(temp);
        }
        else
        {
                p = head;
                for(i=1;i<=pos-2;i++)
                        p = p->link;
                temp = p->link;
                q = (p->link)->link;
                p->link = q;
                free(temp);
        }
        len = len - 1;
        printf("\nElement deleted successfully");
        return;
}
traverse_list()
{
        stype *p;
        if (len == 0)
        {
                printf("\nThe list is empty");
                return;
        }
        printf("\nThe elements in the list are : ");
        p = head;
        while( p!= NULL)
        {
                printf("%d\t",p->info);
                p = p->link;
        }
        return;
}
```

**OUTPUT:-**

```
Linked List
1.Initializing list to empty
2.Determining whether the list is empty or not
3.Displaying length of the list
4.Inserting given element at particular position of the list
5.Deleting the given element from the list
6.Traversing the list elements
7.Exit
Enter your choice...1
List is initialized
Enter your choice…2
List is empty
Enter your choice…3
Length of the list is 0
Enter your choice…4
Enter element to be inserted 10
Enter position to be inserted 1
Element inserted successfully
Enter element to be inserted 20
Enter position to be inserted 2
Element inserted successfully
Enter your choice…5
Enter position for deletion 2
Element deleted successfully
Enter your choice…6
The elements in the list are :   10
```

# Operations on Doubly Linked List

**6. b) AIM:-** To write a c program that uses functions to perform all possible operations on doubly linked list.

**PROGRAM:-**

```
#include<stdio.h>
#include<alloc.h>
typedef struct llist
{       int info;
        struct llist *right, *left;
}stype ;
stype *head1, *head2;
int len= 0;
void main()
{       int ch=1, pos , elem;
        while(ch>0 && ch < 8)
        {       clrscr();
                printf("Doubly Linked List");
                printf("\n1.Initializing list to empty");
                printf("\n2.Determining whether the list is empty or not");
                printf("\n3.Displaying length of the list");
                printf("\n4.Inserting given element at particular position of the list");
                printf("\n5.Deleting the given element from the list");
                printf("\n6.Traversing the list elements from left to right");
                printf("\n7.Traversing the list elements from right to left");
                printf("\n8.Exit");
                printf("\nEnter your choice...");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1 :  initialize_list();
                                  break;
                        case 2 : empty_list();
                                  break;
                        case 3 : display_list();
                                  break;
                        case 4 : printf("\nEnter element to be inserted : ");
                                  scanf("%d",&elem);
                                  printf("\nEnter position to be inserted : ");
                                  scanf("%d",&pos);
                                  insert_list(elem,pos);
                                  break;
                        case 5 : printf("\nEnter position for deletion : ");
                                  scanf("%d",&pos);
                                  delete_list(pos);
                                  break;
                        case 6 : traverselr_list();
                                  break;
                        case 7 : traverserl_list();
                                  break;
```

```
                default : printf("\nProgram Terminated");
            }
            getch();
        }
}
initialize_list()
{       head1 = NULL;
        Head2 = NULL;
        len = 0;
        printf("\nList is initialized");
        return;
}
empty_list()
{       if (len == 0)
            printf("\nList is empty");
        else
            printf("\nList is not empty");
        return;
}
display_list()
{       printf("\nThe length of the list is : %d",len);
        return;
}
insert_list(int elem,int pos)
{       stype *node,*p,*q;
        int i;
        if(pos<1 || pos>len+1)
        {
            printf("\nThe position is invalid, hence no element can be inserted");
            return;
        }
        node = (stype *)malloc(sizeof(stype));
        node->info = elem;
       node->left = NULL;
        node->right = NULL;
        if( len == 0)
        {
            head1 = node;
            head2 = node;
        }
        else
        {
            if ( pos == 1)
            {       node->right = head1;
                    head1->left = node;
                    head1 = node;
            }
            else if ( pos == len+1)
            {       head2->right = node;
                    node->left = head2;
                    head2 = node;
```

```
                }
                else
                {       p = head1;
                        for(i=1;i<=pos-2;i++)
                                p = p->right;
                        q = p->right;
                        p->right = node;
                        node->left = p;
                        node->right = q;
                        q->left = node;
                }
        }
        len = len + 1;
        printf("\nElement inserted successfully");
        return;
}
delete_list(int pos)
{       stype *temp,*p,*q;
        int i;
        if ( len == 0)
        {       printf("\nSingly Linked List is empty");
                return;
        }
        if( pos <1  || pos > len)
        {       printf("\nPostion is invalid");
                return;
        }
        if (len == 1)
        {       temp = head1;
                head1 = NULL;
                head2 = NULL;
                free(temp);
        }
        else
        {       if (pos ==1)
                {
                        temp = head1;
                        head1 = head1->right;
                        head1->left  = NULL;
                        free(temp);
                }
                else if (pos == len)
                {       temp = head2;
                        head2 = head2->left;
                        head2->right  = NULL;
                        free(temp);
                }
                else
                {       p = head1;
                        for(i=1;i<=pos-2;i++)
                                p = p->right;
```

```
                        temp = p->right;
                        q = (p->right)->right;
                        p->right = q;
                        q->left = p;
                        free(temp);
                }
        }
        len = len - 1;
        printf("\nElement deleted successfully");
        return;
}
traverselr_list()
{       stype *p;
        if (len == 0)
        {       printf("\nThe list is empty");
                return;
        }
        printf("\nThe elements in the list are : ");
        p = head1;
        while( p!= NULL)
        {
                printf("%d\t",p->info);
                p = p->right;
        }
        return;
}
traverserl_list()
{       stype *p;
        if (len == 0)
        {       printf("\nThe list is empty");
                return;
        }
        printf("\nThe elements in the list are : ");
        p = head2;
        while( p!= NULL)
        {
                printf("%d\t",p->info);
                p = p->left;
        }
        return;
}
```

**OUTPUT:-**

Doubly Linked List
1.Initializing list to empty
2.Determining whether the list is empty or not
3.Displaying length of the list
4.Inserting given element at particular position of the list
5.Deleting the given element from the list
6.Traversing the list elements from left to right
7.Traversing the list elements from right to left

8.Exit
Enter your choice...1
List is initialized
Enter your choice…2
List is empty
Enter your choice…3
Length of the list is 0
Enter your choice…4
Enter element to be inserted 10
Enter position to be inserted 1
Element inserted successfully
Enter element to be inserted 20
Enter position to be inserted 2
Element inserted successfully
Enter your choice…5
Enter position for deletion 2
Element deleted successfully
Enter your choice…6
The elements in the list are : 10

# Operations on Circular Linked List

**6. c) AIM:-** To write a c program that uses functions to perform all possible operations on circular linked list.

**PROGRAM:-**

```
#include<alloc.h>
typedef struct clist
{       int info;
        struct clist *link;
}stype;
stype *head;
int len;
void main()
{   int ch=1, pos , elem;
    while(ch>0 && ch < 7)
    {   clrscr();
        printf("Circular Linked List");
        printf("\n1.Initializing circular linked list to empty");
        printf("\n2.Determining whether the clircular list is empty or not");
        printf("\n3.Displaying length of the circular list");
        printf("\n4.Inserting given element at particular position of the circular list");
        printf("\n5.Deleting the given element from the circular list");
        printf("\n6.Traversing the circular list elements");
        printf("\n7.Exit");
        printf("\nEnter your choice...");
        scanf("%d",&ch);
        switch(ch)
        {       case 1 :  initialize_list();
                          break;
                case 2 : empty_list();
                          break;
                case 3 : display_list();
                          break;
                case 4 : printf("\nEnter element to be inserted : ");
                          scanf("%d",&elem);
                          printf("\nEnter position to be inserted : ");
                          scanf("%d",&pos);
                          insert_list(elem,pos);
                          break;
                case 5 : printf("\nEnter position for deletion : ");
                          scanf("%d",&pos);
                          delete_list(pos);
                          break;
                case 6 : traverse_list();
                          break;
                default : printf("\nProgram Terminated");
                                  break;
        }
        getch();
    }
```

```
}
initialize_list()
{   head→info = NULL;
    head  →  link = head;
    len = 0;
    printf("\nCircular List is initialized");
    return;
}
empty_list()
{   if (len == 0)
        printf("\nCircular List is empty");
    else
        printf("\nCircular List is not empty");
    return;
}
display_list()
{   printf("\nThe length of the Circular list is : %d",len);
    return;
}
insert_list(int elem,int pos)
{   stype *node,*p,*q;
    int i;
    if(pos<1 || pos>len+1)
    {   printf("\nThe position is invalid, hence no element to be inserted");
        return;
    }
    node = (stype *)malloc(sizeof(stype));
    node→info = elem;
    if ( pos == 1)
    {   (node->link) = (head->link);
        (head->link) = node;
    }
    else
    {   p = (head->link);
        for(i=1;i<=pos-2;i++)
                p = p->link;
        q = p->link;
        p->link = node;
        node->link = q;
    }
    len = len + 1;
    printf("\nElement inserted successfully");
    return;
}
delete_list(int pos)
{   stype *temp,*p,*q;
    int i;
    if ( len == 0)
    {   printf("\nCircular Linked List is empty");
        return;
    }
```

```
    if( pos <1  || pos > len)
    {   printf("\nPostion is invalid");
        return;
    }
    if (pos == 1)
    {   temp = (head->link);
        (head->link) = (head->link)->link;
        free(temp);
    }
    else
    {   p = (head->link);
        for(i=1;i<=pos-2;i++)
                p = p->link;
        temp = p->link;
        q = (p->link)->link;
        p->link = q;
        free(temp);
    }
    len = len - 1;
    printf("\nElement deleted successfully");
    return;
}
traverse_list()
{   stype *p;
    if (len == 0)
    {   printf("\nThe list is empty");
        return;
    }
    printf("\nThe elements in the list are : ");
    p = (head->link);
    while( p!= NULL)
    {   printf("%d\t",p->info);
        p = p->link;
    }
    return;
}
```

**OUTPUT:-**

Circular Linked List
1.Initializing Circular Linked list to empty
2.Determining whether the Circular list is empty or not
3.Displaying length of the circular list
4.Inserting given element at particular position of the Circular list
5.Deleting the given element from the Circular list
6.Traversing the Circular list elements
7.Exit
Enter your choice...1
Circular List is initialized
Enter your choice…2
Circular List is empty
Enter your choice…3

```
Length of the Circular list is 0
Enter your choice…4
Enter element to be inserted 10
Enter position to be inserted 1
Element inserted successfully
Enter element to be inserted 20
Enter position to be inserted 2
Element inserted successfully
Enter your choice…5
Enter position for deletion 2
Element deleted successfully
Enter your choice…6
The elements in the Circular list are :          10
```

# Stack Operations using Arrays

**7. a) AIM:-** To write a c program to implement all possible operations on stack using arrays.

**PROGRAM:-**

```
#include<stdio.h>
#include<conio.h>
#define size 5
int stack[size+1], top=0, i, data;
void stack_intilize();
void stack_empty();
void stack_full();
void stack_top();
void stack_bottom();      /* function declaration */
void stack_size();
void stack_push();
void stack_pop();
void stack_traverse();
void main()
{       int n, data;
        do
        {       clrscr();
                printf("\n\t\tMENU");
                printf("\n1.Intializing stack to empty");
                printf("\n2.Checking whether stack is empty or not");
                printf("\n3.Checking whether stack is full or not");
                printf("\n4.Displaying element at top of the stack");
                printf("\n5.Displaying element at bottom value of the stack");
                printf("\n6.Displaying number of elements in the stack ");
                printf("\n7Push element into the stack");
                printf("\n8.Pop element from the stack");
                printf("\n9.Displaying the values in the stack”);
                printf("\n10.exit");
                printf("\nEnter your choice:");
                scanf("%d",&n);
                switch(n)
                {       case 1 :stack_intilize();
                                break;
                        case 2 :stack_empty();
                                break;
                        case 3 :stack_full();
                                break;
                        case 4 :stack_top();
                                break;
                        case 5 :stack_bottom();
                                break;
                        case 6 :stack_size();
                                break;
                        case 7 :
                                printf("\Enter the Element to insert into the stack:");
                                scanf("%d",&data);
```

```
                        stack_push(data);
                        break;
                case 8 :stack_pop();
                        break;
                case 9 :stack_traverse();
                        break;
                }
                getch();
        }while(n>=1&&n<=9);
}
void stack_intilize()
{       top=0;
        printf("\nStack  is intialized....");
}
void stack_empty()
{       if(top==0)
                printf("\nStack is empty");
        else
                printf("\nStack is not empty");
}
void stack_full()
{
        if(top==size)
                printf("\nStack is full");
        else
                printf("\nStack is not full..");
}
void stack_top()
{       if(top==0)
                printf("\nStack is empty");
        else
                printf("\nElement at top of the stack is :%d",stack[top]);
}
void stack_bottom()
{
        if(top==0)
                printf("\nstack is empty");
        else
                printf("\nElement at the bottom of the stack is :%d",stack[1]);
}
void stack_size()
{
        printf("\nNumber of elements in the  stack  is :%d",top);
}
void stack_push(int data)
{       if(top==size)
                printf("\nThe stack is already full..");
        else
        {
                top++;
                stack[top]=data;
```

```
        }
}
void stack_pop()
{       if(top==0)
                printf("\nThe stack is empty");
        else
        {
                printf("\The popped elemnt is:%d",stack[top]);
                top--;
        }
}
void stack_traverse()
{       if(top==0)
                printf("\nThe stack is empty");
        else
        {
                printf("\Elements in the stack");
                for(i=top;i>0;i--)
                        printf("%d\t",stack[i]);
        }
}
```

**OUTPUT**:

```
                        MENU
1.Intializing stack to empty
2.Checking whether stack is empty or not
3.Checking whether stack is full or not
4.Displaying element at top of the stack
5.Displaying element at bottom value of the stack
6.Displaying number of elements in the stack
7Push element into the stack
8.Pop element from the stack
9.Displaying the values in the stack
10.exit
Enter your choice:1
Stack is intialized....
Enter your choice:7
Enter the Element to insert into the stack:10
Enter your choice:7
Enter the Element to insert into the stack:20
Enter your choice:7
Enter the Element to insert into the stack:30
Enter your choice:2
Stack is not empty
Enter your choice:3
Stack is not full..
Enter your choice:4
Element at top of the Stack is :30
Enter your choice:5
Element at Bottom of the  stack is :10
Enter your choice:6
```

```
The stack size is :3
Enter your choice:9
Elements in the stack
30      20      10
```

# Stack Operations using Pointers

**7. b) AIM:-** To write a c program to implement all possible operations on stack using pointers.

**PROGRAM:-**

```
#include<stdio.h>
#include<alloc.h>
#include<conio.h>
typedef struct stack
{
        int info;
        struct stack *link;
}stype;
stype *top;
int len=0;
void main()
{
        int ch=1,elem;
        while(ch>0&&ch<8)
        {
                clrscr();
                printf("\nstack using pointers ");
                printf("\n1.Initializing stack to empty");
                printf("\n2.Determining whether the stack is empty or not");
                printf("\n3.Displaying length of the stack");
                printf("\n4.Displaying element at top of the stack");
                printf("\n5.Pushing an element into the stack");
                printf("\n6.Poping an element at the top of the stack");
                printf("\n7.Traversing the elements of the stack");
                printf("\n8.Exit");
                printf("\nEnter your choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:initialize_stack();
                                break;
                        case 2:empty_stack();
                                break;
                        case 3:length_stack();
                                break;
                        case 4:top_stack();
                                break;
                        case 5:printf("\nEnter element to be pushed");
                              scanf("%d",&elem);
                              push_stack(elem);
                                break;
                        case 6:pop_stack();
                                break;
                        case 7:traverse_stack();
                                break;
                        default:printf("\nProgram terminated");
```

```
                                        break;
                        }
                        getch();
                }
        }
initialize_stack()
{
        top=NULL;
        len=0;
        printf("\nStack is initialized");
        return;
}
empty_stack()
{
        if(len==0)
                printf("\nStack is empty");
        else
                printf("\nStack is not empty");
        return;
}
length_stack()
{
        printf("\nThe length of the stack:%d",len);
        return;
}
top_stack()
{
        if(len==0)
                printf("\nstack is empty");
        else
                printf("\n Element at top of the stack:%d",top->info);
        return;
}
push_stack(int elem)
{       stype *node;
        int i;
        node=(stype*)malloc(sizeof(stype));
        node->info=elem;
        node->link=top;
        top=node;
        len=len+1;
        printf("\nElement pushed successfully");
        return;
}
pop_stack()
{
        stype *temp;
        if(len==0)
        {
                printf("\nstack is empty no element to pop");
                return;
```

```
        }
        printf("\n The element to be poped is:%d",top->info);
        temp=top;
        top=top->link;
        free(temp);
        len=len-1;
        printf("\nElement poped successfully");
        return;
}
traverse_stack()
{
        stype *p;
        if(len==0)
        {
                printf("\nThe stack is empty,hence no element to traverse");
                return;
        }
        printf("\nThe elements in stack are:");
        p=top;
        while(p!=NULL)
        {
                printf("%d\t",p->info);
                p=p->link;
        }
        return;
}
```

**OUTPUT:-**

```
stack using pointers
1.Initializing stack to empty
2.Determining whether the stack is empty or not
3.Displaying length of the stack
4.Displaying element at top of the stack
5.Pushing an element into the stack
6.Poping an element at the top of the stack
7.Traversing the elements of the stack
8.Exit
Enter your choice:1
Stack is initialized

Enter your choice:2
Stack is empty
Enter your choice:3
The length of the stack:0
Enter your choice:4
stack is empty
Enter your choice:5
Enter element to be pushed10
Element pushed successfully
Enter your choice:5
Enter element to be pushed20
Element pushed successfully
```

```
Enter your choice:5
Enter element to be pushed30
Element pushed successfully
Enter your choice:6
The element to be poped is:30
Element poped successfully
Enter your choice:7
The elements in stack are:20    10
Enter your choice:8
Program terminated
```

# Queue Operations using Arrays

**7. c) AIM:-** To write a c program to implement all possible operations on queue using arrays.

**PROGRAM:-**

```
#include<stdio.h>
#include<conio.h>
#define size 5
int queue[size+1];
int rear=0,front=1;
void q_intilize();
void q_empty();
void q_full();
void q_front();
void q_rear();      /* function declaration */
int q_length();
void q_push();
void q_pop();
void q_traverse();
void main()
{
        int n,data;
        do
        {       clrscr();
                printf("\n\t\tMENU");
                printf("\n1.Intializing queue to empty");
                printf("\n2.Determining whether queue is empty or not");
                printf("\n3.Determining whether queue is full or not");
                printf("\n4.Displaying element at rear of the queue");
                printf("\n5.Displaying element at front of the queue");
                printf("\n6.Displaying number of elements in the queue ");
                printf("\n7.Inserting elements into queue");
                printf("\n8.Deleting elements from queue");
                printf("\n9.Displaying the elements in the queue");
                printf("\n10.exit");
                printf("\nEnter your choice:");
                scanf("%d",&n);
                switch(n)
                {
                        case 1 :q_intilize();
                                break;
                        case 2 :q_empty();
                                break;
                        case 3 :q_full();
                                break;
                        case 4 :q_rear();
                                break;
                        case 5 :q_front();
                                break;
                        case 6 :q_length();
                                break;
```

```
                        case 7 :
                                printf("\Enter the Element to insert into the queue:");
                                scanf("%d",&data);
                                q_push(data);
                                break;
                        case 8 :q_pop();
                                break;
                        case 9 :q_traverse();
                                break;

                }
                getch();
        }while(n>=1&&n<=9);
}
void q_intilize()
{       front=1;
        rear=0;
        printf("\nQueue is intialized....");
}
void q_empty()
{       if(rear==0)
                printf("\nQueue is empty");
        else
                printf("\nQueue is not empty");
}
void q_full()
{       if(rear==size)
                printf("\nQueue is full");
        else
                printf("\nQueue is not full..");
}
void q_front()
{       if(rear==0)
                printf("\nQueue is empty");
        else
                printf("\nElement at front of the Queue is :%d",queue[front]);
}
void q_rear()
{       if(rear==0)
                printf("\nQueue is empty");
        else
                printf("\nElement at rear of the Queue is :%d",queue[rear]);
}
int q_length()
{
        printf("\nNumber of elements in the queue is  :%d",rear);
}
void q_push(int data)
{       if(rear==size)
                        printf("\nThe Queue is already full..");
        else
```

```
        {
                rear++;
                queue[rear]=data;
        }
}
void q_pop()
{       int i;
        if(rear==0)
                printf("\nThe Queue is empty");
        else
        {
                printf("\The deleted element is:%d",queue[front]);
                for(i=2;i<=rear;i++)
                {
                        queue[i-1]=queue[i];
                }
                rear--;
        }
}
void q_traverse()
{       int i;
        if(rear==0)
                printf("\nThe queue is empty");
        else
        {
                printf("\nElements in the queue");
                for(i=front;i<=rear;i++)
                        printf("%d\t",queue[i]);
        }
}
```

**OUTPUT:-**

```
                MENU
1.Intializing queue to empty
2.Determining whether queue is empty or not
3.Determining whether queue is full or not
4.Displaying  element at rear of the queue
5.Displaying element at front of the queue
6.Displaying number of elements in the queue
7.Inserting elements into queue
8.Deleting elements from queue
9.Displaying the elements in the queue
10.exit
Enter your choice:1
Queue is intialized....
Enter your choice:7
Enter the Element to insert into the queue:1
Enter your choice:7
Enter the Element to insert into the queue:2
```

```
Enter your choice:7
Enter the Element to insert into the queue:3
Enter your choice:2
Queue is not empty
Enter your choice:3
Queue is not full..
Enter your choice:4
Element at rear of the queue is :3
Enter your choice:5
Element at front of the queue is :1
Enter your choice:6
Number of elements in the queue is :3
Enter your choice:9
Elements in the queue
1       2       3
Enter your choice:8
The deleted element is:1
```

# Queue Operations using Pointers

**7. d) AIM:-** To write a c program to implement all possible operations on queue using pointers.

**PROGRAM:-**

```
#include<stdio.h>
#include<alloc.h>
#include<conio.h>
typedef struct queue
{       int info;
        struct queue *link;
}stype;
stype *front,*rear;
int len=0;
void main()
{       int ch=1,pos,elem;
        while(ch>0&&ch<9)
        {       clrscr();
                printf("\nQueuee");
                printf("\n1.Initializing queue to empty");
                printf("\n2.Determining whether the queue is empty or not");
                printf("\n3.Displaying length of the queue");
                printf("\n4.Displaying element at rear of the queue");
                printf("\n5.Displaying element at front of the queue");
                printf("\n6.Inserting given element at the rear of the queue");
                printf("\n7.Deleting  an element at front of the queue");
                printf("\n8.Traversing the queue elements");
                printf("\n9.Exit");
                printf("\nEnter your choice:");
                scanf("%d",&ch);
                switch(ch)
                {       case 1:initialize_queue();
                               break;
                        case 2:empty_queue();
                               break;
                        case 3:length_queue();
                               break;
                        case 4:rear_queue();
                               break;
                        case 5:front_queue();
                               break;
                        case 6:printf("\nEnter element to be inserted:");
                              scanf("%d",&elem);
                              insert_queue(elem);
                               break;
                        case 7:delete_queue();
                               break;
                        case 8:traverse_queue();
                               break;
                        default:printf("\nProgram terminated");
                               break;
```

```
                }
                getch();
        }
}
initialize_queue()
{
        front=NULL;
        rear=NULL;
        len=0;
        printf("\nQueue is initialized");
        return;
}
empty_queue()
{
        if(len==0)
                printf("\nQueue is empty");
        else
                printf("\nQueue is not empty");
        return;
}
length_queue()
{
        printf("\nThe length of the queue:%d",len);
        return;
}
rear_queue()
{
        if(len==0)
                printf("\nQueue is empty,hence no element at rear of the queue");
        else
                printf("\nElement at rear of the queue is:%d",rear->info );
        return;
}
front_queue()
{       if(len==0)
                printf("\nQueue is empty,hence no element at front of the queue");
        else
                printf("\nElement at front of the queue is:%d",front->info );
        return;
}
insert_queue(int elem)
{
        stype *node;
        node=(stype*)malloc(sizeof(stype));
        node->info=elem;
        node->link=NULL;
        if(len==0)
        {
                front=node;
                rear=node;
        }
```

```
        else
        {
                (rear->link)=node;
                rear=node;
        }
        len=len+1;
        printf("\nElement inserted successfully");
        return;
}
delete_queue()
{       stype *temp;
        if(len==0)
        {       printf("\nQueue is empty,hence no element to delete");
                return;
        }
        printf("\nThe element to be deleted is:%d",front->info);
        temp=front;
        if(len==1)
        {       front=NULL;
                rear=NULL;
        }
        else
        {
                front=front->link;
        }
        free(temp);
        len=len-1;
        printf("\nElement deleted successfully");
        return;
}
traverse_queue()
{       stype *p;
        if(len==0)
        {
                printf("\nQueue is empty,hence no element to traverse");
                return;
        }
        printf("\nThe elements in the queue are:");
        p=front;
        while(p!=NULL)
        {
                printf("%d\t",p->info);
                p=p->link;
        }
        return;
}
```

**OUT PUT:-**

Queue
1.initializing queue to empty
2.Determining whether the queue is empty or not
3.Displaying length of the queue

4.Displaying element at rear of the queue
5.Displaying element at front of the queue
6.Inserting given element at the rear of the queue
7.Deleting  an element at front of the queue
8.Traversing the queue elements
9.Exit
Enter your choice:1
Queueis initialized
Enter your choice:6
Enter element to be inserted:10
Element inserted successfully
Enter your choice:6
Enter element to be inserted:20
Element inserted successfully
Enter your choice:6
Enter element to be inserted:30
Element inserted successfully
Enter your choice:7
The element to be deleted is:10
Element deleted successfully
Enter your choice:8
The elements in the queue are:20 30
Enter your choice:9
Program terminated

# Converting Infix expression into Postfix Expression

**8. a) AIM:** To write a c program that converts given infix expression into postfix expression using stack.

**PROGRAM:-**

```
#include<conio.h>
#include<string.h>
#define MAX 20
char stack[MAX];
int top=0;
char pop();
void push(char);
int prcd(char);
int isoperator(char);
void convertip(char [],char []);
void main()
{       char infix[20],postfix[20];
        clrscr();
        printf("Enter the valid infix string:\n");
        gets(infix);
        convertip(infix,postfix);
        printf("The corresponding postfix string is: ");
        puts(postfix);
        getch();
}
void push(char item)
{       top++;
        stack[top]=item;
}
char pop()
{       char a;
        a=stack[top];
        top--;
        return a;
}
void convertip(char infix[],char postfix[])
{       int i,symbol,j=0;
        stack[++top]='#';
        for(i=0;i<strlen(infix);i++)
        {       symbol=infix[i];
                if(symbol=='(')
                        push(symbol);
                else if(symbol==')')
                {       while(stack[top]!='(')
                        {       postfix[j]=pop();
                                j++;
                        }
                        pop();//pop out (.
                }
                else if (isoperator(symbol)==1)
```

```
                {       while(stack[top] != '#' && prcd(symbol)<=prcd(stack[top]))
                        {       postfix[j]=pop();
                                j++;
                        }
                        push(symbol);
                }//end of else.
                else if(isoperator(symbol)==0)
                {       postfix[j]=symbol;
                        j++;
                }
        }//end of for.
        while(stack[top]!='#')
        {       postfix[j]=pop();
                j++;
        }
        postfix[j]='\0';//null terminate string.
}
int prcd(char symbol)
{       if(symbol == '^' || symbol == '$')
                return 6;
        else if(symbol == '*' || symbol == '/')
                return 4;
        else if(symbol == '+' || symbol == '-')
                return 2;
        else if(symbol == '(' || symbol == ')' || symbol == '#')
                return 1;
        else
                return 0;
}
int isoperator(char symbol)
{       if(symbol == '^' || symbol == '$' || symbol == '+' || symbol == '-' || symbol == '*' || symbol
== '/' || symbol == '(' || symbol == ')')
                return 1;
        else
                return 0;
}
```

**OUTPUT:-**

```
Enter the valid infix string:
a+b*c
The corresponding postfix string is: abc*+
```

# Evaluating the Postfix Expression

**8. b) AIM:-** To write a c program that evaluates given postfix expression

**PROGRAM:-**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
float  stack[20];
int top=0;
float pop();
void push(float);
int isoperator(char);
float evaluate(char []);
float calculate(float,char,float);
void main()
{       char postfix[20];
        float solution;
        clrscr();
        printf("Enter the valid postfix string:\n");
        gets(postfix);
        solution = evaluate(postfix);
        printf("The result value is : %f",solution);
        getch();
}
void push(float item)
{       top++;
        stack[top]=item;
}
float pop()
{       float a;
        a=stack[top];
        top--;
        return a;
}
int isoperator(char symbol)
{       if(symbol == '+' || symbol == '-' || symbol == '*' || symbol == '/' )
                return 1;
        else
                return 0;
}
float evaluate(char postfix[])
{       int i;
        char symbol;
        float oper1,oper2,result;
        for(i=0;i<strlen(postfix);i++)
        {
                symbol = postfix[i];
                if (isoperator(symbol)==0)
                {
                        push(symbol-48);
```

```
                }
                else
                {
                        oper2 = pop();
                        oper1 = pop();
                        result = calculate(oper1,symbol,oper2);
                        push(result);
                }
        }//end of for.
        result = pop();
        return(result);
}
float calculate(float oper1,char symbol,float oper2)
{
        switch(symbol)
        {
                case '+' : return(oper1+oper2);
                case '-' : return(oper1-oper2);
                case '*' : return(oper1* oper2);
                case '/' : return(oper1/oper2);
                default : return(0);
        }
}
```

**OUTPUT:-**

```
Enter the valid postfix string:
235*+
The result value is : 17.000000
```

# Bubble Sort

**9. a) AIM :-** To write a c program to sort n numbers using Bubble Sort

**PROGRAM:-**

```
#include<stdio.h>
void main()
{
        int arr[20],i,n;
        clrscr();
        printf("\nEnter number of elements : ");
        scanf("%d",&n);
        printf("Enter elements : ");
        for(i=1;i<=n;i++)
                scanf("%d",&arr[i]);
        bubble_sort(arr,n);
        getch();
}
bubble_sort(int a[],int n)
{
        int i,j,temp;
        for(i=1;i<=n-1;i++)
        {
                for(j=1;j<=n-i;j++)
                {
                        if(a[j]>a[j+1])
                        {
                                temp  = a[j];
                                a[j]  = a[j+1];
                                a[j+1]= temp;
                        }
                }
        }
        printf("\nThe sorted list is : ");
        for(i=1;i<=n;i++)
                printf("%5d",a[i]);
        return;
}
```

**OUTPUT:**

```
Enter number of elements : 5
Enter elements : -2 0 5 3 1
The sorted list is :    -2    0    1    3    5
```

# Selection Sort

**9. b) AIM :-** To write a c program to sort n numbers using Selection Sort

**PROGRAM:**

```
#include<stdio.h>
void main()
{
        int arr[20],i,n;
        clrscr();
        printf("\nEnter number of elements : ");
        scanf("%d",&n);
        printf("Enter elements : ");
        for(i=1;i<=n;i++)
                scanf("%d",&arr[i]);
        selection_sort(arr,n);
        getch();
}
selection_sort(int a[],int n)
{
        int i,j,temp,min,loc;
        for(i=1;i<=n-1;i++)
        {
                min=a[i];
                loc=i;
                for(j=i+1;j<=n;j++)
                {
                        if(min > a[j])
                        {
                                min = a[j];
                                loc = j;
                        }
                }
                temp=a[i];
                a[i]=a[loc];
                a[loc]=temp;
        }
        printf("The sorted list is : ");
        for(i=1;i<=n;i++)
                printf("%5d",a[i]);
        return;
}
```

**OUTPUT:**

```
Enter number of elements : 5
Enter elements : -2 0 5 3 1
The sorted list is :    -2    0    1    3    5
```

# Quick Sort

**10. a) AIM :-** To write a c program to sort n numbers using Quick Sort

**PROGRAM:-**

```
#include<stdio.h>
void main()
{       int i,n,a[20];
        clrscr();
        printf("\nEnter number of elements : ");
        scanf("%d",&n);
        printf("\nEnter elements : ");
        for(i=1;i<=n;i++)
                scanf("%d",&a[i]);
        quick_sort(a,1,n);
        printf("\nThe sorted elements are : ");
        for(i=1;i<=n;i++)
                printf("%5d",a[i]);
        getch();
}
quick_sort(int a[],int lb,int ub)
{       int key,i,j,flag=1,temp;
        if(lb<ub)
        {       key=a[lb];
                i=lb;
                j=ub+1;
                while(flag)
                {       i=i+1;
                        while(a[i]<key &&i<=ub)
                                i++;
                        j--;
                        while(a[j]>key &&j>=lb)
                                j=j-1;
                        if(i<j)
                        {       temp = a[i];
                                a[i] = a[j];
                                a[j] = temp;
                        }
                        else
                                flag=0;
                }
                temp=a[lb];
                a[lb]=a[j];
                a[j]=temp;
                quick_sort(a,lb,j-1);
                quick_sort(a,j+1,ub);
        }

        return;
}
```

**OUTPUT:**

```
Enter number of elements : 5
Enter elements : 5 3 4 1 2
The sorted elements are :    1   2   3   4   5
```

# Merge Sort

**10. b) AIM :-** To write a c program to sort n numbers using Merge Sort

**PROGRAM:-**

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i,n,a[20];
        clrscr();
        printf("\nEnter number of elements : ");
        scanf("%d",&n);
        printf("\n Enter elements : ");
        for(i=1;i<=n;i++)
                scanf("%d",&a[i]);
        merge_sort(a,1,n);
        printf("\n The sorted elements are : ");
        for(i=1;i<=n;i++)
                printf("%5d",a[i]);
        getch();
}
merge_sort(int a [],int start,int finish)
{
        int size,mid;
        size=finish-start+1;
        if(size<=1)
                return;
        mid=(start+finish)/2;
        merge_sort(a,start,mid);
        merge_sort(a,mid+1,finish);
        merge(a,start,mid+1,finish);
        return;
}
merge(int a[],int s, int m, int f)
{       int i,j,k,temp[20];
        i=s;    j=m;    k=1;
        while(i<m && j<=f)
        {       if(a[i]<=a[j])
                {       temp[k]=a[i];
                        i=i+1;k=k+1;
                }
                else
                {       temp[k]=a[j];
                        j=j+1;k=k+1;
                }
        }
        if(i >= m)
        {
                while(j<=f)
                {
```

```
                temp[k]=a[j];
                j=j+1;          k=k+1;
            }
        }
        else
        {
            while(i<m)
            {
                temp[k]=a[i];
                i=i+1;k=k+1;
            }
        }
        for(i=1;i<=k-1;i++)
            a[s+i-1]=temp[i];
        return;
}
```

**OUTPUT:**

Enter number of elements : 5
Enter elements : 3 5 1 4 2
The sorted elements are : 1 2 3 4 5

# Linear Search

**11. a) AIM:-** To write a c program that use both recursive and non- recursive functions to perform linear search operation for a key value in a given list of integers.

**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int ch=1,n,key,i,a[20];
        clrscr();
        printf("\nEnter number of  elements : ");
        scanf("%d",&n);
        printf("\nEnter elements : ");
        for(i=1;i<=n;i++)
                scanf("%d",&a[i]);
        printf("\nEnter key element : ");
        scanf("%d",&key);
        do
        {
                printf("\n1.Linear search using recursion");
                printf("\n2.Linear search without using recursion");
                printf("\n3.Exit");
                printf("\nEnter your choice...");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1 : lsearch_recursion(a,n,key);
                                 break;
                        case 2 : lsearch_nonrecursion(a,n,key);
                                 break;
                        default : printf("\nProgram Terminated");
                }
                getch();
        }while(ch>0 && ch<3);
}
lsearch_recursion(int a[],int n,int key)
{
        int f = 0;
        if (a[n] == key)
        {
                printf("\nElement is found at location : %d",n);
                f = 1 ;
        }
        else
        {
                if ( (n==0) && (f==0) )
                        printf("\nElement is not found");
                else
                        lsearch_recursion(a,n-1,key);
```

```
        }
        return;
}
lsearch_nonrecursion(int a[],int n,int key)
{
        int i;
        a[n+1]=key;
        i=1;
        while(a[i]!=key)
                i++;
        if(i<=n)
                printf("\nElement is found in location : %d",i);
        else
                printf("\nElement is not found");
        return;
}
```

**OUTPUT:**

```
Enter number of  elements : 5
Enter elements : 10 20 30 40 50
Enter key element : 40

1.Linear search using recursion
2.Linear search without using recursion
3.Exit
Enter your choice...1
Element is found at location : 4

1.Linear search using recursion
2.Linear search without using recursion
3.Exit
Enter your choice...2
Element is found in location : 4

1.Linear search using recursion
2.Linear search without using recursion
3.Exit
Enter your choice...3
Program Terminated
```

# Binary Search

**11. b) AIM:-** To write a c program that use both recursive and non- recursive functions to perform binary search operation for a key value in a given list of integers

**PROGRAM:-**

```
#include<conio.h>
void main()
{       int ch=1,n,key,i,a[20];
        clrscr();
        printf("\nEnter number of  elements : ");
        scanf("%d",&n);
        printf("\nEnter elements : ");
        for(i=1;i<=n;i++)
                scanf("%d",&a[i]);
        printf("\nEnter key element : ");
        scanf("%d",&key);
        do
        {       printf("\n1.Binary search using recursion");
                printf("\n2.Binary search without using recursion");
                printf("\n3.Exit");
                printf("\nEnter your choice...");
                scanf("%d",&ch);
                switch(ch)
                {       case 1 : printf("\nElement is found at location %d", bsearch_recursion (a,
1 ,n, key));
                                 break;
                        case 2 : bsearch_nonrecursion(a,n,key);
                                 break;
                        default : printf("\nProgram Terminated");
                }
                getch();
        }while(ch>0 && ch<3);
}
bsearch_recursion(int a[],int low,int high,int key)
{       int loc,mid;
        if(low>high)
                loc=0;
        else
        {
                mid=(low+high)/2;
                if(key<a[mid])
                        loc = bsearch_recursion(a,low,mid-1,key);
                else if(key>a[mid])
                        loc = bsearch_recursion(a,mid+1,high,key);
                else
                        loc = mid;
        }
        return(loc);
}
bsearch_nonrecursion(int a[],int n,int key)
```

```
{       int start = 1, mid,end = n, flag = 0;
        while( start <= end)
        {       mid = (start + end) / 2;
                if ( a[mid]  == key )
                {
                        printf("\nElement is found at location : %d", mid);
                        flag = 1;
                        break ;
                }
                else
                {
                        if ( key > a[mid] )
                                start = mid + 1;
                        else
                                end = mid -  1 ;
                }
        }
        if ( flag == 0)
                printf("\nElement not found");
        return;
}
```

**OUTPUT:**

```
Enter number of  elements : 5
Enter elements : 10 20 30 40 50
Enter key element : 20
1.Binary search using recursion
2.Binary search without using recursion
3.Exit
Enter your choice...1
Element is found ar location :2
1.Binary search using recursion
2.Binary search without using recursion
3.Exit
Enter your choice...2
Element is found at location : 2
1.Binary search using recursion
2.Binary search without using recursion
3.Exit
Enter your choice...3
Program Terminated
```

# Binary Search Tree and its Operations

**12) AIM:-** To write a c program to create Binary Search Tree and perform operations on it.

**PROGRAM:**

```
#include<stdio.h>
#include<alloc.h>
typedef struct node
{       struct node *left;
        int info;
        struct node *right;
}stype;
stype *root=NULL,*par,*cur,*newnode;
void main()
{
        int choice,number;
        do
        {       clrscr();
                printf("\n\t\tBinary Search Tree operations");
                printf("\n1.Initializing BST to empty");
                printf("\n2.Inserting an element into BST");
                printf("\n3.Deleting an element from BST");
                printf("\n4.Inorder Traversal");
                printf("\n5.Exit");
                printf("\nEnter your choice : ");
                scanf("%d",&choice);
                switch(choice)
                {       case 1 : initialize();
                                 break;
                        case 2 : printf("\nEnter number to insert into BST : ");
                                 scanf("%d",&number);
                                 insert(number);
                                 break;
                        case 3 : printf("\nEnter number to delete from BST : ");
                                 scanf("%d",&number);
                                 delete_bst(number);
                                 break;
                        case 4 : inorder(root);
                                 break;
                        default : printf("\nProgram Terminated");
                }
                getch();
        }while(choice>0 && choice<5);
}
initialize()
{       root=NULL;
        printf("\nThe Binary Search Tree is initialized");
        return;
}
insert(int elem)
{       newnode=(stype *)malloc(sizeof(stype));
```

```
        newnode->info=elem;
        newnode->left=NULL;
        newnode->right=NULL;
        if(root == NULL)
                root = newnode;
        else
        {       cur = root;
                while(cur !=NULL)
                {       par=cur;
                        if(elem<=cur->info)
                                cur=cur->left;
                        else
                                cur=cur->right;
                }
                if(elem<=par->info)
                        par->left=newnode;
                else
                        par->right=newnode;
        }
        printf("\nThe element inserted into BST successfully");
        return;
}
inorder(stype *root)
{       if(root!=NULL)
        {
                inorder(root->left);
                printf("%4d",root->info);
                inorder(root->right);
        }
        return;
}
delete_bst(int elem)
{
        stype *temp;
        if (root == NULL)
        {
                printf("\nTree is empty, hence deletion cannot be performed");
                return;
        }
        cur = root;
        par = NULL;
        while(cur != NULL && cur->info !=elem)
        {
                par = cur;
                if (elem <= (cur->info) )
                        cur = cur->left;
                else
                        cur = cur->right;
        }
        if (cur == NULL)
        {
```

```
            printf("\nGiven element is not found in BST and hence it cannot be deleted");
            return;
        }
        if ( cur == root )
        {
            if (cur->left == NULL && cur->right == NULL)
                root = NULL;
            else if (cur->left != NULL && cur->right == NULL)
                root = root->left;
            else if (cur->left == NULL && cur->right != NULL)
                root = root->right;
            else
            {
                temp = (root->left);
                root = root->right;
                cur = root;
                while ( cur->left != NULL)
                    cur = cur->left;
                cur->left = temp;
            }
        }
        else
        {
            if ( cur->left == NULL && cur->right == NULL)
            {
                if( (cur->info) <= (par->info))
                    par->left = NULL;
                else
                    par->right = NULL;
            }
            else if ( cur->left != NULL && cur->right == NULL)
            {
                if( (cur->info) <= (par->info))
                    par->left = cur->left;
                else
                    par->right = cur->left;
            }
            else if ( cur->left == NULL && cur->right != NULL)
            {
                if( (cur->info) <= (par->info))
                    par->left = cur->right;
                else
                    par->right = cur->right;
            }
            else
            {
                temp = (cur->left);
                if( (cur->info) <= (par->info))
                    par->left = cur->right;
                else
                    par->right = cur->right;
```

```
                        cur = (cur->right);
                        while( cur->left != NULL)
                                cur = cur->left;
                        cur->left = temp;
                }
        }
        printf("\nThe element deleted from BST successfully");
        return;
}
```

**OUTPUT:**

```
        Binary Search Tree operations
1.Initializing BST to empty
2.Inserting an element into BST
3.Deleting an element from BST
4.Inorder Traversal
5.Exit
Enter your choice : 1
The Binary Search Tree is initialized
Enter your choice : 2
Enter number to insert into BST : 10
The element inserted into BST successfully
Enter your choice : 3
Enter number to delete from BST : 30
The element deleted from BST successfully
Enter your choice : 4
10      20      40
```

# Recursive Traversal Techniques on BST

**13. AIM:-** To write a C program to implement BST Recursive Traversal Techniques

**PROGRAM:**

```
#include<stdio.h>
#include<alloc.h>
typedef struct node
{
        struct node *left;
        int info;
        struct node *right;
}stype;
stype *root=NULL,*par,*cur,*newnode;
void main()
{       int choice,number;
        do
        {
                clrscr();
                printf("\n\t\tBST Recursive Traversal Techniques");
                printf("\n1.Initializing BST to empty");
                printf("\n2.Inserting element into BST");
                printf("\n3.Preorder Traversal");
                printf("\n4.Inorder Traversal");
                printf("\n5.Postorder Traversal");
                printf("\n6.Exit");
                printf("\nEnter your choice : ");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1 : initialize();
                                 break;
                        case 2 : printf("\nEnter number to insert into BST : ");
                                 scanf("%d",&number);
                                 insert(number);
                                 break;
                        case 3 : preorder(root);
                                 break;
                        case 4 : inorder(root);
                                 break;
                        case 5 : postorder(root);
                                 break;
                        default : printf("\nProgram Terminated");
                }
                getch();
        }while(choice>0 && choice<6 );
}
initialize()
{
        root=NULL;
        printf("\nThe Binary Search Tree is initialized");
```

```
        return;
}
insert(int elem)
{
        newnode=(stype *)malloc(sizeof(stype));
        newnode->info=elem;
        newnode->left=NULL;
        newnode->right=NULL;
        if(root == NULL)
                root = newnode;
        else
        {
                cur = root;
                while(cur != NULL)
                {
                        par = cur;
                        if(elem<=cur->info)
                                cur=cur->left;
                        else
                                cur=cur->right;
                }
                if(elem<=par->info)
                        par->left=newnode;
                else
                        par->right=newnode;
        }
        printf("\nThe element inserted into BST successfully");
        return;
}
preorder(stype *root)
{
        if(root!=NULL)
        {
                printf("%4d",root->info);
                preorder(root->left);
                preorder(root->right);
        }
        return;
}

inorder(stype *root)
{
        if(root!=NULL)
        {
                inorder(root->left);
                printf("%4d",root->info);
                inorder(root->right);
        }
        return;
}
postorder(stype *root)
```

```
{
        if(root!=NULL)
        {
                postorder(root->left);
                postorder(root->right);
                printf("%4d",root->info);
        }
        return;
}
```

**OUTPUT:**

```
            BST Recursive Traversal Techniques
1.Initializing BST to empty
2.Inserting element into BST
3.Preorder Traversal
4.Inorder Traversal
5.Postorder Traversal
6.Exit
Enter your choice : 1
The Binary Search Tree is initialized
Enter your choice : 2
Enter number to insert into BST : 10
The element inserted into BST successfully
Enter your choice : 4
10   20   30   40
Enter your choice : 5
  40  30  20  10
```

# Non- Recursive Traversal Techniques on BST

**14. AIM**:- To write a C program to implement BST Non Recursive Traversal Techniques

**PROGRAM:**

```
#include<stdio.h>
#include<alloc.h>
typedef struct node
{       struct node *left;
        int info;
        struct node *right;
}stype;
stype *root=NULL,*par,*cur,*newnode;
void main()
{       int choice,number;
        do
        {       clrscr();
                printf("\n\t\tBST NON-Recursive Traversal Techniques");
                printf("\n1.Initializing BST to empty");
                printf("\n2.Inserting element into BST");
                printf("\n3.Preorder Traversal");
                printf("\n4.Inorder Traversal");
                printf("\n5.Postorder Traversal");
                printf("\n6.Exit");
                printf("\nEnter your choice : ");
                scanf("%d",&choice);
                switch(choice)
                {       case 1 : initialize();
                                 break;
                        case 2 : printf("\nEnter number to insert into BST : ");
                                 scanf("%d",&number);
                                 insert(number);
                                 break;
                        case 3 : preorder(root);
                                 break;
                        case 4 : inorder(root);
                                 break;
                        case 5 : postorder(root);
                                 break;
                        default : printf("\nProgram Terminated");
                }
                getch();
        }while(choice>0 && choice<6 );
}
initialize()
{       root=NULL;
        printf("\nThe Binary Search Tree is initialized");
        return;
}
insert(int elem)
{       newnode=(stype *)malloc(sizeof(stype));
```

```
        newnode->info=elem;
        newnode->left=NULL;
        newnode->right=NULL;
        if(root == NULL)
                root = newnode;
        else
        {       cur = root;
                while(cur != NULL)
                {       par = cur;
                        if(elem<=cur->info)
                                cur=cur->left;
                        else
                                cur=cur->right;
                }
                if(elem<=par->info)
                        par->left=newnode;
                else
                        par->right=newnode;
        }
        printf("\nThe element inserted into BST successfully");
        return;
}
preorder(stype *root)
{       stype *ptr,*stack[30];
        int top;
        if ( root == NULL)
        {       printf("BST is empty, there are no nodes to traverse");
                return;
        }
        ptr = root;
        top = 0;
        top = top+1;
        stack[top] = NULL;
        while(ptr != NULL)
        {       printf("%d\t",ptr->info);
                if( ptr->right != NULL)
                {
                        top = top + 1;
                        stack[top] = ptr->right;
                }
                if (ptr->left != NULL)
                        ptr = ptr->left;
                else
                {
                        ptr = stack[top];
                        top = top - 1;
                }
        }
        return;
}
inorder(stype *root)
```

```
{       stype *ptr,*stack[30];
        int top;
        if ( root == NULL)
        {
                printf("BST is empty, there are no nodes to traverse");
                return;
        }
        ptr = root;
        top = 0;
        top = top+1;
        stack[top] = NULL;
    bst_lt :    while(ptr != NULL)
        {
                top = top+1;
                stack[top] = ptr;
                ptr = ptr->left;
        }
        ptr = stack[top];
        top = top - 1;
        while(ptr != NULL)
        {
                printf("%d\t",ptr->info);
                if(ptr->right != NULL)
                {
                        ptr = ptr->right;
                        goto bst_lt;
                }
                else
                {
                        ptr = stack[top];
                        top = top - 1;
                }
        }
        return;
}
postorder(stype *root)
{       stype *ptr ,*stack[30], *dummy;
        int top;
        if ( root == NULL)
        {
                printf("BST is empty, there are no nodes to traverse");
                return;
        }
        ptr = root;
        top = 0;
        top = top+1;
        stack[top] = NULL;
    pt_lt :     while(ptr != NULL)
        {
                top = top+1;
                stack[top] = ptr;
```

```
                if ( ptr->right != NULL)
                {
                        top = top+1;
                        dummy = ptr->right;
                        dummy->info = -(dummy->info);
                        stack[top] =  dummy;
                }
                ptr = ptr->left;
        }
        ptr = stack[top];
        top = top - 1;
        while(ptr->info > 0)
        {
                printf("%d\t",ptr->info);
                ptr = stack[top];
                top = top - 1;
        }
        if (ptr->info < 0)
        {
                dummy = ptr;
                dummy->info = -(dummy->info);
                ptr = dummy;
                goto pt_lt;
        }
        return;
}
```

**OUTPUT:**

```
        BST NON-Recursive Traversal Techniques
1.Initializing BST to empty
2.Inserting element into BST
3.Preorder Traversal
4.Inorder Traversal
5.Postorder Traversal
6.Exit
Enter your choice : 1
The Binary Search Tree is initialized
Enter your choice : 2
Enter number to insert into BST : 10
The element inserted into BST successfully
Enter your choice : 4
10   20   30   40
Enter your choice : 5
  40  30  20  10
```