KAKINADA INSTITUTE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF MCA

REG No: …….....

LAB NAME : ……

YEAR : I SEMESTER : I

STUDENT NAME : ……

KAKINADA INSTITUTE OF ENGINEERING AND TECHNOLOGY
(APPROVED BY AICTE, AFFILIATED TO JNTU KAKINADA & GOVT. OF A.P)
Yanam Road, Korangi, Kakinada– 533461, E.G.Dist, Andhra Pradesh
Phone : 0884-2303400,2304050, Fax : 0884-2303869
Website : www.kietgroup.com, E-Mail: kietmcaprojects@gmail.com

KAKINADA INSTITUTE OF ENGINEERING AND TECHNOLOGY

(APPROVED BY AICTE, AFFILIATED TO JNTU KAKINADA & GOVT. OF A.P)
Yanam Road, Korangi, Kakinada– 533461, E.G.Dist, Andhra Pradesh

DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS

CERTIFICATE

This is to certify that the practical work done by Mr./Ms……..bearing the Reg No: …… is a bonafide student of Kakinada Institute of Engineering & Technology, and has successfully completed the..... Laboratory in partial fulfillment of I Year, I Semester of MCA Department during the Year 2020- 2021.

Number of Experiments Conducted :

Number of Experiments Completed :

External Lab Examination Held on : …./..../….

INTERNAL EXAMINER **EXTERNAL EXAMINER**

HEAD OF THE DEPARTMENT

KAKINADA INSTITUTE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF MCA

INDEX

5	23-03-2021	Write recursive program for the following a) Write recursive and non recursive C program for calculation of Factorial of an integer. b) Write recursive and non recursive C program for	34-40	

		calculation of GCD (n, m) c) Write recursive and non recursive C program for Towers of Hanoi: N disks are to be transferred from peg S to peg D with Peg I as the intermediate peg		
6	30-03-2021	a) Write C program that use both recursive and non recursive functions to perform Linear search for a Key value in a given list. b) Write C program that use both recursive and non recursive functions to perform Binary search for a Key value in a given list	41-48	
7	06-04-2021	a) Write C program that implement stack (its operations) using arrays. b) Write C program that implement stack (its operations) using Linked list.	49-67	
8	16-04-2021	a) Write a C program that uses Stack operations to convert infix expression into postfix expression. a) Write C program that implement Queue (its operations) using arrays. b) Write C program that implement Queue (its operations) using linked lists	68-89	
9	20-04-2021	Write a C program that uses functions to create a singly linked list and perform various operations on it.	90-95	
10	27-04-2021	Write a C program to store a polynomial expression in memory using linked list and perform polynomial addition.	96-103	
11	30-04-2021	a) Write a recursive C program for traversing a binary tree in preorder, inorder and postorder. b) Write a non recursive C program for traversing a binary tree in preorder, inorder and postorder.	104-136	

12	04-05-2021	a) Write a C program to implement Prims' algorithm. b) Write a C program to implement Kruskal's algorithm	137-152	
13	07-05-2021	Implementation of Hash table using double hashing as collision resolution function.	153-165	
14	11-05-2021	Implementation of Binary Search trees- Insertion and deletion.	164-166	
15	17-05-2021	a) Write C program that implement Bubble sort, to sort a given list of integers in ascending order. b) Write C program that implement Quick sort, to sort a given list of integers in ascending order. c) Write C program that implement merge sort, to sort a given list of integers in ascending order	167-174	
16				
17				
18				

Program 1(a):

AIM: Write a program in C to display the n terms of even natural number and their sum

```
#include <stdio.h>
void main()
{
int i,n,sum=0;
printf("Input number of terms :
"); scanf("%d",&n);
printf("\nThe even numbers are
:"); for(i=1;i<=n;i++)
{
printf("%d
",2*i);
sum+=2*i;
}
printf("\nThe Sum of even Natural Number upto %d terms : %d \n",n,sum);
}
```

OUTPUT:

```
Input number of terms : 5
The even numbers are:2 4 6 8 10
The Sum of even Natural Number up to 5 terms: 30
```

Program 1(b):

AIM: Write a program in C to display the n terms of harmonic series and their sum.

1 + 1/2 + 1/3 + 1/4 + 1/5 ... 1/n

terms. #include <stdio.h>

```
void main()

{

  int i,n; float s=0.0;

  printf("Input the number of terms :

  "); scanf("%d",&n);

  printf("\n\n");

  for(i=1;i<=n;i++)

  {

     if(i<n)

     { printf("1/%d + ",i); s+=1/(float)i;
     }

    if(i==n)

    { printf("1/%d", i);
    s+=1/(float)i;

    }
    }

       printf("\nSum of Series upto %d terms : %f \n",n,s);

}
```

Output:

Input the number of terms :

5 1/1 + 1/2 + 1/3 + 1/4 +1/5

Sum of Series upto 5 terms : 2.283334

Program 1 (c):

AIM: Write a C program to check whether a given number is an Armstrong number or not.

```
#include
<stdio.h> int
main() {
int num, originalNum, remainder, result
= 0; printf("Enter a three-digit integer: ");
scanf("%d", &num);
originalNum = num;
while (originalNum !=
0) {
remainder = originalNum % 10;
result += remainder * remainder *
remainder; originalNum /= 10;
}
if (result == num)
printf("%d is an Armstrong number.",
num); else
printf("%d is not an Armstrong number.",
num); getch ();
}
```

Output: Enter a three-digit integer:

371 371 is an Armstrong number**.**

Program 1 (d):

AIM: Write a C program to calculate the factorial of a given number

```
#include<stdio.h
>
#include<conio.
h> int
factorial(int);
void main( )
{
int n,k;
printf("enter any
number\n");
scanf("%d",&n);
k=factorial(n);
printf("factorial of %d is %d",n,k);
getch( );
}
int factorial(int x)
{
int fact;
f(x==0||x==1
) eturn(1);
else
{
fact=(x*factorial(x-1));
```

```
getch();
}
}
```

Output: enter any number 4

Factorial of 4 is 24

Program 2 (a):

AIM: Write a program in C for multiplication of two square Matrices.

```
#include
<stdio.h> void
main()
{
 int arr1[50][50],brr1[50][50],crr1[50][50],i,j,k,r1,c1,r2,c2,sum=0;

    printf("\n\nMultiplication of two Matrices
    :\n"); printf("_____\n");

 printf("\nInput the rows and columns of first matrix :
 "); scanf("%d %d",&r1,&c1);
 printf("\nInput the rows and columns of second matrix
 : "); scanf("%d %d",&r2,&c2);
 if(c1!=r2){
   printf("Mutiplication of Matrix is not possible.");
   printf("\nColumn of first matrix and row of second matrix must be same.");
 }
 else
   {
    printf("Input elements in the first matrix
    :\n"); for(i=0;i<r1;i++)
```

```
{
   for(j=0;j<c1;j++)
   {
          printf("element - [%d],[%d] : ",i,j);
          scanf("%d",&arr1[i][j]);
   }
}
printf("Input elements in the second matrix
:\n"); for(i=0;i<r2;i++)
{
   for(j=0;j<c2;j++)
   {
        printf("element - [%d],[%d] : ",i,j);
          scanf("%d",&brr1[i][j]);
   }
}
                                                printf("\nThe First matrix is
                                                      :\n"); for(i=0;i<r1;i++)
          {
          printf("\n");
          for(j=0;j<c1;j++)
   printf("%d\t",arr1[i][j]);
```

```
                }
  printf("\nThe Second matrix is :\n");
                for(i=0;i<r2;i++)
                {
                printf("\n");
                for(j=0;j<c2;j++)
                printf("%d\t",brr1[i][j]);
                }
//multiplication of
    matrix
    for(i=0;i<r1;i++)
       for(j=0;j<c2;j+
        +) crr1[i][j]=0;
         for(i=0;i<r1;i++) //row of first matrix
            {
             for(j=0;j<c2;j++) //column of second matrix
              {
               sum=0;
                for(k=0;k<c1;k+
                +)
                 sum=sum+arr1[i][k]*brr1[k][j
                 ]; crr1[i][j]=sum;
              }
            }
```

```
 printf("\nThe multiplication of two matrices is :
 \n"); for(i=0;i<r1;i++)
   {
     printf("\n");
     for(j=0;j<c2;j+
     +)
     {
       printf("%d\t",crr1[i][j]);
     }
   }
 }
printf("\n\n");
}
```

Output:

Multiplication of two Matrices :

Input the rows and columns of first matrix

: 2 2

Input the rows and columns of second matrix

: 2 2

Input elements in the first

matrix : element - [0],[0] : 1

element - [0],[1] : 2

element - [1],[0] : 3

element - [1],[1] : 4

Input elements in the second

matrix : element - [0],[0] : 5

element - [0],[1] : 6

element - [1],[0] : 7

element - [1],[1] : 8

The First matrix is :

1 2

3 4

The Second matrix is :

5 6

7 8

The multiplication of two matrices

is : 19 22

43 50

Program 2 (b):

AIM: Write a program in C to find transpose of a given matrix. #include <stdio.h> int main() {

```
   int a[10][10], transpose[10][10], r, c,
   i, j; printf("Enter rows and columns:
   "); scanf("%d %d", &r, &c);
  // Assigning elements to the matrix
   printf("\nEnter matrix
   elements:\n"); for (i = 0; i < r; ++i)
      for (j = 0; j < c; ++j) {
         printf("Enter element a%d%d: ", i + 1, j
         + 1); scanf("%d", &a[i][j]);
      }
   // Displaying the matrix a[][]
   printf("\nEntered matrix:
   \n"); for (i = 0; i < r; ++i)
      for (j = 0; j < c; ++j) {
         printf("%d ", a[i][j]);
         if (j == c - 1)
            printf("\n");
      }
```

```
    // Finding the transpose of
    matrix a for (i = 0; i < r; ++i)
        for (j = 0; j < c; ++j) {
            transpose[j][i] =
            a[i][j];
        }
    // Displaying the transpose of matrix
    a     printf("\nTranspose     of     the
    matrix:\n"); for (i = 0; i < c; ++i)
        for   (j   =   0;   j   <   r;   ++j)   {
            printf("%d                    ",
            transpose[i][j]);  if  (j  == r  -
            1)
                printf("\n");
        }
    return 0;
}
```

Output:

```
Enter rows and
columns: 2 3
Enter matrix
elements: Enter
element a11: 1 Enter
element a12: 4
```

Enter element a13:

0 Enter element

a21: -5 Enter

element a22: 2

Enter element a23:

7

Entered matrix:

1 4 0

-5 2 7

Transpose of the matrix:

1 -5

4 2

0 7

Program 3 (a) :

AIM: Write a program in C to check whether a number is a prime number or not using the function.

```
#include
<stdio.h>
#include
<conio.h>     void
main()
{
    int
    num,res=0;
    clrscr();
    printf("\nENTER A
    NUMBER: ");
    scanf("%d",&num);
    res=prime(num);
    if(res==0)
            printf("\n%d IS A PRIME NUMBER",num);
    else
            printf("\n%d IS NOT A PRIME
    NUMBER",num); getch();
}
int prime(int n)
{
    int i;
```

```
for(i=2;i<=n/2;i+
+)
```

```
    {
        if(n%i!=0)
            continue;
        else
            return 1;

    }
    return 0;
}
```

Output

ENTER A NUMBER 7

7 IS A PRIME NUMBER

Program 3 (b):

AIM: Write recursive program which computes the nth Fibonacci number, for appropriate values of n.

```
/** C Program to find the nth number in Fibonacci series using
recursion*/ #include <stdio.h>
int
fibo(int);
int main()
{
   int num;
   int
   result;
   printf("Enter the nth number in fibonacci
   series: "); scanf("%d", &num);
   if (num < 0)
   {
      printf("Fibonacci of negative number is not possible.\n");
   }
   else
   {
      result = fibo(num);
      printf("The %d number in fibonacci series is %d\n", num, result);
   }
   return 0;
```

```
}
int fibo(int num)
{
   if (num == 0)
   {
      return 0;
   }
   else if (num == 1)
   {
      return 1;
   }
   else
   {
      return(fibo(num - 1) + fibo(num - 2));
   }
}
```

Output:

Enter the nth number **in** fibonacci

series: 8 The 8 number **in** fibonacci

series is 21

Enter the nth number **in** fibonacci

series: 12 The 12 number **in** fibonacci

series is 144

Program 3 (c):

AIM: Write a program in C to add numbers using call by reference.

```
#include <stdio.h>
long addTwoNumbers(long *, long
*); int main()
{
  long fno, sno, sum;
  printf("\n\n Pointer : Add two numbers using call by
  reference:\n"); printf("_____\n");
  printf(" Input the first number :
  "); scanf("%ld", &fno);
  printf(" Input the second number : ");
  scanf("%ld", &sno);
  sum = addTwoNumbers(&fno, &sno);
  printf(" The sum of %ld and %ld is %ld\n\n", fno, sno,
  sum); return 0;
}
long addTwoNumbers(long *n1, long *n2)
{
  long sum;
  sum = *n1 +
  *n2; return
  sum;
```

```
}
```

Output:

Pointer : Add two numbers using call by reference:

Input the first number : 5

Input the second number :

6 The sum of 5 and 6 is 11

Program 4 (a):

AIM: Write a program in C to append multiple lines at the end of a text file.

```
#include
<stdio.h> int
main ()
{
 FILE *
 fptr; int
 i,n;
 char str[100];
 char
 fname[20];
 char str1;

      printf("\n\n Append multiple lines at the end of a text file
      :\n"); printf("_____\n");
      printf(" Input the file name to be opened :
      "); scanf("%s",fname);
  fptr = fopen(fname, "a");
  printf(" Input the number of lines to be written :
  "); scanf("%d", &n);
  printf(" The lines are :
  \n"); for(i = 0; i <
  n+1;i++)
  {
```

```
fgets(str, sizeof str, stdin);
```

```
    fputs(str, fptr);
  }
  fclose (fptr);
//----- Read the file after appended -------
        fptr = fopen (fname, "r");
        printf("\n The content of the file %s is :\n",fname);
        str1 = fgetc(fptr);
        while (str1 != EOF)
                {
                        printf ("%c",
                        str1); str1 =
                        fgetc(fptr);
                }
    printf("\n\n");
    fclose (fptr);
//_____End of reading __________
  return 0;
}
```

Output:

Append multiple lines at the end of a text file :

Input the file name to be opened :

test.txt Input the number of lines to be

written : 3

The lines are :

test line 5

test line 6

test line 7

The content of the file test.txt is :

test line 1

test line 2

test line 3

test line 4

test line 5

test line 6

test line 7

Program 4 (b):

AIM: Write a program in C to copy a file in another name.

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

void main()
{
        FILE *fptr1, *fptr2;
        char ch, fname1[20], fname2[20];

        printf("\n\n Copy a file in another name
        :\n"); printf("_____\n");

        printf(" Input the source file name :
        "); scanf("%s",fname1);

        fptr1=fopen(fname1,
        "r"); if(fptr1==NULL)
        {
                printf(" File does not found or error in
                opening.!!"); exit(1);
        }
```

```
printf(" Input the new file name :
"); scanf("%s",fname2);
fptr2=fopen(fname2, "w");
if(fptr2==NULL)
{
        printf(" File does not found or error in
        opening.!!"); fclose(fptr1);
        exit(2);
}
while(1)
{
        ch=fgetc(fptr1
        ); if(ch==EOF)
        {
                break;
        }
        else
        {
                fputc(ch, fptr2);
        }
 }
```

```
        printf(" The file %s  copied successfully in the file %s. \n\n",fname1,fname2)
        ; fclose(fptr1);
        fclose(fptr2);
        getchar();
}
```

Output:

Copy a file in another name :

Input the source file name :

test.txt Input the new file name :

test1.txt

The file test.txt copied successfully in the file test1.txt.

Program 5 (a):

AIM:

Write recursive and non recursive C program for calculation of Factorial of an integer.

```
#include<stdio.h>
long int multiplyNumbers(int
n); int main() {
   int n;
   printf("Enter a positive integer:
   "); scanf("%d",&n);
   printf("Factorial of %d = %ld", n,
   multiplyNumbers(n)); return 0;
}
long int multiplyNumbers(int
   n) { if (n>=1)
      return n*multiplyNumbers(n-
   1); else
      return 1;
}
```

Output:

```
Enter a positive integer:
6 Factorial of 6 = 720
```

Program 5 (b):

AIM:

Write recursive and non recursive C program for calculation of GCD (n, m) #include <stdio.h>

```
int hcf(int n1, int
n2); int main() {
   int n1, n2;
   printf("Enter two positive integers:
   "); scanf("%d %d", &n1, &n2);
   printf("G.C.D of %d and %d is %d.", n1, n2, hcf(n1,
   n2)); return 0;
}
int hcf(int n1, int n2)
   { if (n2 != 0)
      return hcf(n2, n1 %
   n2); else
      return n1;
}
```

Output:

```
Enter two positive integers: 366
                                        60
G.C.D of 366 and 60 is 6.
```

Program 5 (c):

AIM:

Write recursive and non recursive C program for Towers of Hanoi: N disks are to be transferred from peg S to peg D with Peg I as the intermediate peg.

```
// recursive function to solve
   // Tower of Hanoi puzzle
   static void towerOfHanoi(int n, char
                from_rod, char to_rod, char
                aux_rod1,
                        char aux_rod2)
   {
      if (n ==
         0)
         return;
      if (n == 1) {
         Console.WriteLine("Move disk " +
                  n + " from rod " + from_rod
                  +
                     " to rod " + to_rod);
         return;
      }

      towerOfHanoi(n - 2, from_rod, aux_rod1,
```

```
aux_rod2, to_rod);
```

```
    Console.WriteLine("Move disk " + (n - 1)
                + " from rod " + from_rod
                + " to rod " + aux_rod2);
    Console.WriteLine("Move disk " +
    n +
                 " from rod " + from_rod
                 + " to rod " + to_rod);
    Console.WriteLine("Move disk " + (n
    - 1)
                + " from rod " + aux_rod2
                 + " to rod " + to_rod);
    towerOfHanoi(n - 2, aux_rod1,
    to_rod,
                   from_rod, aux_rod2);
  }

  // Driver method
  public static void Main()
  {
    int n = 4; // Number of disks

    // A, B, C and D are names of
    rods towerOfHanoi(n, 'A', 'D',
    'B', 'C');
  }
```

```
}
```

Output:

Move disk 1 from rod A to rod D Move disk 2 from rod A to rod B Move disk 1 from rod D to rod B Move disk 3 from rod A to rod C Move disk 4 from rod A to rod D Move disk 3 from rod C to rod D Move disk 1 from rod B to rod C Move disk 2 from rod B to rod D Move disk 1 from rod C to rod D

Program 6 (a):

AIM:

Write C program that use both recursive and non recursive functions to perform Linear search for a Key value in a given list

```
/*Write a C program Linear Search Using Recursive
Functions*/ #include<stdio.h>
#include<conio.h>45
int
lin_search(int[],int,int);
main()
{
   int a[100],n,i,ele;
   clrscr();
   printf("Enter
   Size");
   scanf("%d",&n);
   printf("Enter %d
   elements",n);
   for(i=0;i<n;i++)
   scanf("%d",&a[i]);
   printf("The array
   elements"); for(i=0;i<n;i++)
   printf("%4d",a[i]);
   printf("\nEnter element to
   search"); scanf("%d",&ele);
   i=lin_search(a,n-
   1,ele); if(i!=-1)
   printf("\nThe element %d found at %d
   location",ele,i+1); else
   printf("\nThe element %d is not
   found",ele); getch();
}
int lin_search(int x[100],int n,int ele)
```

```
{
   if(n<=0)
   return -1;
   if(x[n]==ele
   ) return n;
   else
   return lin_search(x,n-1,ele);
}
```

Output:

```
Enter Size5
Enter 5 elements25
30
12
54
60
The array elements

25 30 12 54  60
Enter element to

search 60

The element 60 found at 5 location
```

Program 6 (b):

AIM:

Write C program that use both recursive and non recursive functions to perform Binary search for a Key value in a given list.

```
/* Binary search program in C using both recursive and non recursive
functions */ #include <stdio.h>
#define MAX_LEN 10
/* Non-Recursive function*/
void b_search_nonrecursive(int l[],int num,int ele)
{
  int l1,i,j, flag =
  0; l1 = 0;
  i = num-1;
  while(l1 <=
  i)
  {
    j = (l1+i)/2;
    if( l[j] ==
    ele)
    {
   printf("\nThe element %d is present at position %d in
       list\n",ele,j); flag =1;
       break;
```

```
    }
    else
     if(l[j] <
       ele) l1 =
       j+1;
     else
       i = j-1;
  }
  if( flag == 0)
  printf("\nThe element %d is not present in the list\n",ele);
}
/* Recursive function*/
int b_search_recursive(int l[],int arrayStart,int arrayEnd,int a)
{
 int m,pos;
 if (arrayStart<=arrayEnd)
 {
  m=(arrayStart+arrayEnd)
  /2; if (l[m]==a)
    return m;
  else if
  (a<l[m])
    return b_search_recursive(l,arrayStart,m-1,a);
```

```
    else
      return b_search_recursive(l,m+1,arrayEnd,a);
   }
   return -1;
}
void read_list(int l[],int n)
{
   int i;
   printf("\nEnter the
   elements:\n"); for(i=0;i<n;i++)
      scanf("%d",&l[i]);
}
void print_list(int l[],int n)
{
    int i;
   for(i=0;i<n;i+
   +)
      printf("%d\t",l[i]);
}
/*main function*/
main()
{
```

```
int l[MAX_LEN], num,
ele,f,l1,a; int ch,pos;
//clrscr();
printf("==========================================
=======");
printf("\n\t\t\tMENU");
printf("\n==========================================
======");
printf("\n[1] Binary Search using Recursion method");
printf("\n[2] Binary Search using Non-Recursion
method"); printf("\n\nEnter your Choice:");
scanf("%d",&ch
); if(ch<=2 &
ch>0)
{
 printf("\nEnter the number of elements :
 "); scanf("%d",&num);
 read_list(l,num);
 printf("\nElements present in the list
 are:\n\n"); print_list(l,num);
 printf("\n\nEnter the element you want to
 search:\n\n"); scanf("%d",&ele);
```

```
  switch(ch)
  {
   case                1:printf("\nRecursive
     method:\n");
     pos=b_search_recursive(l,0,num,e
     le); if(pos==-1)
     {
     printf("Element is not found");
     }
     else
     {
     printf("Element is found at %d position",pos);
     }
     //getch();
     break;
   case 2:printf("\nNon-Recursive
     method:\n");
     b_search_nonrecursive(l,num,ele);
     //getch();
     break;
   }
  }
//getch();
```

```
}
```

OUTPUT:

```
==============================================

                ======== MENU

==================================================
[1] Binary Search using Recursion method
[2] Binary Search using Non-Recursion
method Enter your Choice:1
Enter the number of elements
: 5 Enter the elements:
12
22
32
42
52
Elements present in the list
are: 12      22     32     42
        52
Enter the element you want to
search: 42
Recursive method:
Element is found at 3 position
```

Program 7(a):

AIM:

Write C program that implement stack (its operations) using arrays.

```
#include<stdio.h>
int
stack[100],choice,n,top,x,i;
void push(void);
void pop(void);
void
display(void); int
main()
{
   //clrscr()
   ; top=-1;
   printf("\n Enter the size of
   STACK[MAX=100]:"); scanf("%d",&n);
   printf("\n\t STACK OPERATIONS USING ARRAY");
   printf("\n\t_____");
   printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t
   4.EXIT"); do
   {
      printf("\n Enter the
      Choice:");
```

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

```
switch(choice)
{
   case 1:
   {
      push()
      ;
      break;
   }
   case 2:
   {
      pop();
      break
      ;
   }
   case 3:
   {
      display(
      ); break;
   }
   case 4:
   {
      printf("\n\t EXIT POINT
      "); break;
   }
```

```
        default:
        {
           printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
        }
      }
   }
   while(choice!=4
   ); return 0;
}
void push()
{
   if(top>=n-1)
   {
      printf("\n\tSTACK is over flow");
   }
   else
   {
      printf(" Enter a value to be
      pushed:"); scanf("%d",&x);
      top++;
      stack[top]=
      x;
   }
```

```
}
void pop()
{
   if(top<=-1)
   {
      printf("\n\t Stack is under flow");
   }
   else
   {
      printf("\n\t The popped elements is
      %d",stack[top]); top--;
   }
}
void display()
{
   if(top>=0)
   {
      printf("\n The elements in STACK
      \n"); for(i=top; i>=0; i--)
         printf("\n%d",stack[i]);
      printf("\n Press Next
      Choice");
   }
```

```
  else
  {
     printf("\n The STACK is empty");
  }
}
```

Output:

```
Enter the size of
      STACK[MAX=100]:10 STACK
      OPERATIONS USING ARRAY

      1.PUSH
      2.POP
      3.DISPLAY
      4.EXIT
 Enter the Choice:1
 Enter a value to be
 pushed:12 Enter the
 Choice:1
 Enter a value to be
 pushed:24 Enter the
 Choice:1
 Enter a value to be
 pushed:98 Enter the
 Choice:3
 The elements in STACK
```

98

24

12

Press Next

Choice Enter the

Choice:2

The popped elements is

98 Enter the Choice:3

The elements in

STACK 24

12

Press Next

Choice Enter the

Choice:4

EXIT POINT

Program 7(b):

AIM:

Write C program that implement stack (its operations) using Linked list

```
/*
 * C Program to Implement a Stack using Linked List
 */
#include
<stdio.h>
#include
<stdlib.h>    struct
node
{
   int info;
   struct node *ptr;
}*top,*top1,*temp;
int topelement();
void push(int
data); void pop();
void  empty();
void
display();
void
destroy();
void
stack_count();
```

```
void create();
```

```
int count =
0; void
main()
{
   int no, ch, e;
   printf("\n  1  -
   Push");
   printf("\n 2 - Pop");
   printf("\n 3 - Top");
   printf("\n 4 - Empty");
   printf("\n 5 - Exit");
   printf("\n 6 - Dipslay");
   printf("\n 7 - Stack
   Count"); printf("\n 8 -
   Destroy stack"); create();
   while (1)
   {
      printf("\n Enter choice :
      "); scanf("%d", &ch);
      switch (ch)
      {
      case 1:
         printf("Enter data :
         "); scanf("%d",
         &no);
```

```
    push(no)
    ; break;
case 2:
    pop();
    break
    ;
case 3:
    if (top == NULL)
        printf("No elements in
    stack"); else
    {
        e = topelement();
        printf("\n Top element : %d", e);
    }
    break
; case
4:
    empty()
    ; break;
case 5:
    exit(0)
; case 6:
    display(
    ); break;
```

```
        case 7:
           stack_count(
           ); break;
        case 8:
           destroy()
           ; break;
        default :
           printf(" Wrong choice, Please enter correct
           choice "); break;
        }
     }
}
/* Create empty stack */
void create()
{
   top = NULL;
}
/* Count stack elements */
void stack_count()
{
   printf("\n No. of elements in stack : %d", count);
}
```

```
/* Push data into stack */
void push(int data)
{
   if (top == NULL)
   {
      top =(struct node *)malloc(1*sizeof(struct
      node)); top->ptr = NULL;
      top->info = data;
   }
   else
   {
      temp =(struct node *)malloc(1*sizeof(struct
      node)); temp->ptr = top;
      temp->info =
      data; top = temp;
   }
   count++;
}
/* Display stack elements */
void display()
{
```

```
    top1 = top;
    if (top1 == NULL)
    {
       printf("Stack is
       empty"); return;
    }
    while (top1 != NULL)
    {
       printf("%d ", top1-
       >info); top1 = top1-
       >ptr;
    }
 }
/* Pop Operation on stack */
void pop()
{
    top1 = top;
    if (top1 == NULL)
    {
       printf("\n Error : Trying to pop from empty
       stack"); return;
    }
    else
```

```
        top1 = top1->ptr;
    printf("\n Popped value : %d", top-
    >info); free(top);
    top =
    top1;
    count--;
}
/* Return top element */
int topelement()
{
    return(top->info);
}
/* Check if stack is empty or not */
void empty()
{
    if (top == NULL)
        printf("\n Stack is
    empty"); else
        printf("\n Stack is not empty with %d elements", count);
}
/* Destroy entire stack */
void destroy()
{
```

```
    top1 = top;
    while (top1 != NULL)
    {
       top1 = top-
       >ptr; free(top);
       top = top1;
       top1 = top1->ptr;
    }
    free(top1);
    top =
    NULL;
    printf("\n All stack elements
    destroyed"); count = 0;
}
```

Output:

1 - Push

2 - Pop

3 - Top

4 - Empty

5 - Exit

6 - Dipslay

7 - Stack

Count 8 -

Destroy stack

Enter choice :

1 Enter data :

56

Enter choice :

1 Enter data :

80

Enter choice : 2

Popped value :

80 Enter choice

: 3

Top element :

56 Enter choice

: 1 Enter data :

78

Enter choice :

1 Enter data :

90

Enter choice :

6 90 78 56

Enter choice : 7

No. of elements in stack

: 3 Enter choice : 8

All stack elements

destroyed Enter choice : 4

Stack is

empty Enter

choice : 5

Program 8 (a):

AIM:

Write a C program that uses Stack operations to convert infix expression into postfix expression.

```
#include<stdio.h
>
#include<ctype.h
> char
stack[100]; int
top = -1;
void push(char x)
{
   stack[++top] = x;
}
char pop()
{
   if(top == -1)
      return -
   1; else
      return stack[top--];
}
int priority(char x)
{
   if(x == '('
```

```
        return 0;
    if(x == '+' || x == '-
        ') return 1;
    if(x == '*' || x == '/')
        return 2;
    return 0;
}
int main()
{
    char
    exp[100];
    char *e, x;
    printf("Enter the expression :
    "); scanf("%s",exp);
    printf("\n");
    e = exp;
    while(*e != '\0')
    {
        if(isalnum(*e))
            printf("%c
            ",*e);
        else if(*e ==
            '(')
            push(*e);
        else if(*e == ')')
```

```
        {
           while((x = pop()) !=
              '(') printf("%c ", x);
        }
        else
        {
           while(priority(stack[top]) >=
              priority(*e)) printf("%c ",pop());
           push(*e);
        }
        e++
        ;
     }
     while(top != -1)
     {
        printf("%c ",pop());
     }return 0;
}
```

Output:

Enter the expression :

a+b*c a b c * +

Enter the expression : ((4+8)(6-5))/((3-

2)(2+2)) 4 8 + 6 5 - 3 2 - 2 2 + /

Program 8 (b):

AIM:

Write C program that implement Queue (its operations) using arrays.

```
/*
 * C Program to Implement a Queue using an Array
 */
#include
<stdio.h>
#define MAX 50
void insert();
void
delete();
void
display();
int
queue_array[MAX];
int rear = - 1;
int front = -
1; main()
{
   int
   choice;
   while (1)
   {
      printf("1.Insert element to queue \n");
```

```
printf("2.Delete element from queue
\n"); printf("3.Display all elements of
queue \n");
```

```
printf("4.Quit \n");
printf("Enter your choice :
"); scanf("%d", &choice);
switch (choice)
{
   case 1:
   insert()
   ;
   break;
   case 2:
   delete()
   ; break;
   case 3:
   display(
   ); break;
   case 4:
   exit(1);
   default
   :
   printf("Wrong choice \n");
} /* End of switch */
                                                    } /* End of while */
                                                  } /* End of main() */
```

```
void insert()
{
   int add_item;
   if (rear == MAX - 1)
   printf("Queue Overflow
   \n"); else
   {
      if (front == - 1)
      /*If queue is initially empty
      */ front = 0;
      printf("Inset the element in queue :
      "); scanf("%d", &add_item);
      rear = rear + 1;
      queue_array[rear] =
      add_item;
   }
} /* End of insert() */

void delete()
{
   if (front == - 1 || front > rear)
   {
      printf("Queue Underflow \n");
```

```
        return ;
    }
    else
    {
        printf("Element deleted from queue is : %d\n",
        queue_array[front]); front = front + 1;
    }
} /* End of delete() */

void display()
{
    int i;
    if (front == - 1)
        printf("Queue is empty
    \n"); else
    {
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf("%d ",
            queue_array[i]);
        printf("\n");
    }
}
```

Output:

1.Insert element to queue

2.Delete element from

queue 3.Display all

elements of queue 4.Quit

Enter your choice : 1

Inset the element in queue :

10 1.Insert element to queue

2.Delete element from

queue 3.Display all

elements of queue 4.Quit

Enter your choice : 1

Inset the element in queue :

15 1.Insert element to queue

2.Delete element from

queue 3.Display all

elements of queue 4.Quit

Enter your choice : 1

Inset the element in queue :

20 1.Insert element to queue

2.Delete element from

queue 3.Display all

elements of queue

4.Quit

Enter your choice : 1

Inset the element in queue :

30 1.Insert element to queue

2.Delete element from

queue 3.Display all

elements of queue 4.Quit

Enter your choice : 2

Element deleted from queue is :

10 1.Insert element to queue

2.Delete element from

queue 3.Display all

elements of queue 4.Quit

Enter your choice :

3 Queue is :

15 20 30

1.Insert element to queue

2.Delete element from

queue 3.Display all

elements of queue 4.Quit

Enter your choice : 4

Program 8 (c):

AIM:

Write C program that implement Queue (its operations) using linked lists

```
/*
 * C Program to Implement Queue Data Structure using Linked List
 */
#include
<stdio.h>
#include
<stdlib.h>

struct node
{
   int info;
   struct node *ptr;
}*front,*rear,*temp,*front1;

int
frontelement();
void         enq(int
data); void deq();
void empty();
void display();
void create();
void
```

```
queuesize();
```

```
int count = 0;

void main()
{
   int no, ch, e;

   printf("\n 1 - Enque");
   printf("\n 2 - Deque");
   printf("\n 3 - Front
   element"); printf("\n 4 -
   Empty"); printf("\n 5 - Exit");
   printf("\n 6 - Display");
   printf("\n 7 - Queue
   size"); create();
   while (1)
   {
      printf("\n Enter choice :
      "); scanf("%d", &ch);
      switch (ch)
      {
      case 1:
```

```
    printf("Enter data :
    "); scanf("%d",
    &no); enq(no);
    break
; case
2:
    deq();
    break
; case
3:
    e =
    frontelement(); if
    (e != 0)
        printf("Front element : %d",
    e); else
        printf("\n No front element in Queue as queue is empty");
    break
; case
4:
    empty();
    break
; case
5:
    exit(0)
; case 6:
```

```
display();
break;
```

```
        case 7:
           queuesize();
           break
        ;
        default:
           printf("Wrong choice, Please enter correct choice ");
           break;
        }
     }
}

/* Create an empty queue */
void create()
{
  front = rear = NULL;
}

                                        /* Returns queue size
                                        */
void queuesize()
{
   printf("\n Queue size : %d", count);
}
```

```
/* Enqueing the queue */
void enq(int data)
{
   if (rear == NULL)
   {
      rear = (struct node *)malloc(1*sizeof(struct
      node)); rear->ptr = NULL;
      rear->info  =
      data; front =
      rear;
   }
   else
   {
      temp=(struct node *)malloc(1*sizeof(struct
      node)); rear->ptr = temp;
      temp->info =
      data; temp->ptr
      = NULL;

      rear = temp;
   }
   count++;
}
```

```
/* Displaying the queue elements */
void display()
{
   front1 = front;

   if ((front1 == NULL) && (rear == NULL))
   {
      printf("Queue is
      empty"); return;
   }
   while (front1 != rear)
   {
      printf("%d ", front1-
      >info); front1 = front1-
      >ptr;
   }
   if (front1 == rear)
      printf("%d", front1-
      >info);
}

/* Dequeing the queue */
void deq()
{
```

```
front1 = front;

if (front1 == NULL)
{
   printf("\n Error: Trying to display elements from empty
   queue"); return;
}
else
   if (front1->ptr != NULL)
   {
      front1 = front1->ptr;
      printf("\n Dequed value : %d", front-
      >info); free(front);
      front = front1;
   }
   else
   {
      printf("\n Dequed value : %d", front-
      >info); free(front);
      front =
      NULL; rear
      = NULL;
   }
```

```
        count--;
}

/* Returns the front element of queue */
int frontelement()
{
   if ((front != NULL) && (rear !=
       NULL)) return(front->info);
   else
       return 0;
}
/* Display if queue is empty or not */
void empty()
{
    if ((front == NULL) && (rear ==
       NULL)) printf("\n Queue empty");
   else
      printf("Queue not empty");
}
```

Output:

1 - Enque

2 - Deque

3 - Front

element 4 -

Empty

5 - Exit

6 - Display

7 - Queue **size**

Enter choice : 1

Enter data : 14

Enter choice : 1

Enter data : 85

Enter choice : 1

Enter data : 38

Enter choice : 3

Front element :

14 Enter choice :

6

14 85 38

Enter choice :

7 Queue **size**

: 3 Enter

choice : 2

Dequed value :

14 Enter choice

: 6 85 38

Enter choice : 7

Queue **size** : 2

Enter choice : 4

Queue not

empty Enter

choice : 5

Program 9:

AIM:

Write a C program that uses functions to create a singly linked list and perform various operations on it.

```
/* Simple Singly(Single) Linked List Example Program Using Functions in C*/

/* Data Structure Programs,C Linked List Examples */


#include <stdio.h>

#include

<malloc.h>

#include

<stdlib.h> struct

node {

 int value;

 struct node *next;

};

void

insert(int);

void

display();

typedef struct node DATA_NODE;

DATA_NODE *head_node, *first_node, *temp_node = 0;


int main() {

 int loop =
```

1; int data;

```
  first_node = 0;
  printf("Singly(Single) Linked List Example - Using
  Functions\n"); while (loop) {
   printf("\nEnter Element for Insert Linked List (-1 to Exit ) :
   \n"); scanf("%d", &data);

   if (data >= 0)
    {
    insert(data)
    ;
   } else {
    loop =
    0;
    temp_node->next = 0;
   }
  }
  display(
  ); return
  0;
}
 void insert(int data) {
  temp_node = (DATA_NODE *) malloc(sizeof (DATA_NODE));

  temp_node->value = data;
```

```
if (first_node == 0) {
```

```
    first_node = temp_node;
   } else {
    head_node->next = temp_node;
   }
   head_node =
   temp_node;
   fflush(stdin);
  }
  void
   display() {
   int count =
   0;
   temp_node = first_node;
   printf("\nDisplay Linked List :
   \n"); while (temp_node != 0) {
    printf("# %d # ", temp_node-
    >value); count++;
    temp_node = temp_node -> next;
   }
   printf("\nNo Of Items In Linked List : %d", count);
  }
```

Output:

```
  Singly(Single) Linked List Example - Using Functions


  Enter Element for Insert Linked List (-1 to Exit ) :
```

555

Enter Element for Insert Linked List (-1 to
Exit) : 444

Enter Element for Insert Linked List (-1 to
Exit) : 333

Enter Element for Insert Linked List (-1 to
Exit) : 222

Enter Element for Insert Linked List (-1 to
Exit) : 111

Enter Element for Insert Linked List (-1 to Exit) :
-1

Display Linked List :
555 # # 444 # # 333 # # 222 # # 111
No Of Items In Linked List : 5

Program 10:

AIM: Write a C program to store a polynomial expression in memory using linked list and perform polynomial addition.

```
#include
<iostream>
#include
<iomanip.h> using
namespace std;


struct poly
  { int
  coeff;
  int
  pow_val;
  poly* next;
};


class add {
  poly *poly1, *poly2, *poly3;


public:
  add() { poly1 = poly2 = poly3 =
  NULL; } void addpoly();
  void display();
};
```

```
void add::addpoly()
{
   int i, p;
   poly *newl = NULL, *end = NULL;
   cout << "Enter highest power for x\n"; cin >> p;
   //Read first poly
   cout << "\nFirst Polynomial\n"; for (i = p; i >= 0;
      i--) { newl = new poly;
      newl->pow_val = p;
      cout << "Enter Co-efficient for degree" << i << ":: "; cin >> newl-
      >coeff; newl->next = NULL;
      if (poly1 ==
         NULL) poly1
         = newl;
      else
         end->next =
      newl; end = newl;
   }

   //Read Second poly
   cout << "\n\nSecond Polynomial\n"; end = NULL; for (i = p; i >=
      0; i--) { newl = new poly;
      newl->pow_val = p;
```

```
    cout << "Enter Co-efficient for degree" << i << ":: "; cin >> newl-
    >coeff; newl->next = NULL;
    if (poly2 ==
       NULL) poly2
       = newl;
    else
       end->next =
    newl; end = newl;
}

//Addition Logic
poly *p1 = poly1, *p2 =
poly2; end = NULL;
while (p1 != NULL && p2 !=
    NULL) { if (p1->pow_val == p2-
    >pow_val) {
       newl = new poly;
       newl->pow_val =
       p--;
       newl->coeff = p1->coeff + p2-
       >coeff; newl->next = NULL;
       if (poly3 ==
          NULL) poly3
          = newl;
```

```
else
```

```
end->next = newl;
```

```
            end = newl;
        }
        p1 = p1-
        >next; p2 =
        p2->next;
    }
}

void add::display()
{
    poly* t = poly3;
    cout << "\n\nAnswer after addition is :
    "; while (t != NULL) {
        cout.setf(ios::showpos);
        cout << t->coeff;
        cout.unsetf(ios::showpo
        s); cout << "X" << t-
        >pow_val; t = t->next;
    }
}
int main()
{
    add obj;
```

```
    obj.addpoly(
    );
    obj.display();
}
```

Output:

```
Enter the total number of terms in the polynomial:4
Enter the COEFFICIENT and EXPONENT in DESCENDING ORDER
Enter the Coefficient(1):
3 Enter the
exponent(1): 4 Enter
the Coefficient(2): 7
Enter the exponent(2):
3 Enter the
Coefficient(3): 5 Enter
the exponent(3): 1
Enter the Coefficient(4):
8 Enter the
exponent(4): 0

First polynomial : 3(x^4)+7(x^3)+5(x^1)+8(x^0)

Enter the total number of terms in the polynomial:5
Enter the COEFFICIENT and EXPONENT in DESCENDING ORDER
Enter the Coefficient(1):
7 Enter the
```

exponent(1): 5 Enter

the Coefficient(2): 6

Enter the exponent(2):

4 Enter the

Coefficient(3): 8 Enter

the exponent(3): 2

Enter the Coefficient(4):

9 Enter the

exponent(4): 1 Enter

the Coefficient(5): 2

Enter the exponent(5):

0

Second polynomial : 7(x^5)+6(x^4)+8(x^2)+9(x^1)+2(x^0)

Resultant polynomial after addition:

7(x^5)+9(x^4)+7(x^3)+8(x^2)+14(x^1)+10(x^0)

Program 11(a):

AIM:

Write a recursive C program for traversing a binary tree in preorder, inorder and postorder.

```
/*
 * C Program to Traverse the Tree Recursively
 */
#include
<stdio.h>
#include
<stdlib.h>

struct node
{
   int a;
   struct node *left;
   struct node
   *right;
};

void generate(struct node **,
int); void infix(struct node *);
void postfix(struct node
*); void prefix(struct node
*); void delete(struct
```

```
node **);
```

```
int main()
{
   struct node *head = NULL;
   int choice = 0, num, flag = 0, key;

   do
   {
      printf("\nEnter your choice:\n1. Insert\n2. Traverse via infix\n3.Traverse
via prefix\n4. Traverse via postfix\n5. Exit\nChoice: ");
      scanf("%d",
      &choice);
      switch(choice)
      {
      case 1:
         printf("Enter element to insert:
         "); scanf("%d", &num);
         generate(&head, num);
         break
      ; case
      2:
         infix(head)
         ; break;
      case 3:
```

```
            prefix(head
            ); break;
        case 4:
            postfix(head
            ); break;
        case 5:
            delete(&head);
            printf("Memory Cleared\nPROGRAM
            TERMINATED\n"); break;
        default: printf("Not a valid input, try again\n");
        }
    } while (choice !=
    5); return 0;
}

void generate(struct node **head, int num)
{
    struct node *temp = *head, *prev = *head;

    if (*head == NULL)
    {
        *head = (struct node *)malloc(sizeof(struct node));
```

```
    (*head)->a = num;
    (*head)->left = (*head)->right = NULL;
}
else
{
    while (temp != NULL)
    {
        if (num > temp->a)
        {
            prev = temp;
            temp = temp->right;
        }
        else
        {
            prev = temp;
            temp = temp->left;
        }
    }
    temp = (struct node *)malloc(sizeof(struct
    node)); temp->a = num;
    if (num >= prev->a)
    {
```

```
            prev->right = temp;
        }
        else
        {
            prev->left = temp;
        }
    }
}

void infix(struct node *head)
{
    if (head)
    {
        infix(head->left);
        printf("%d ", head-
        >a); infix(head-
        >right);
    }
}

void prefix(struct node *head)
{
    if (head)
```

```
    {
        printf("%d ", head-
        >a); prefix(head-
        >left); prefix(head-
        >right);
    }
}

void postfix(struct node *head)
{
    if (head)
    {
        postfix(head->left);
        postfix(head->right);
        printf("%d ", head-
        >a);
    }
}

void delete(struct node **head)
{
    if (*head != NULL)
    {
        if ((*head)->left)
```

```
        {
           delete(&(*head)->left);
        }
        if ((*head)->right)
        {
           delete(&(*head)->right);
        }
        free(*head);
     }
}
```

Output:

Enter your choice:

1. Insert
2. Traverse via infix
3. Traverse via prefix
4. Traverse via postfix
5. Exit

Choice:

1

Enter element to insert: 5

Enter your choice:

1. Insert

2. Traverse via infix

3. Traverse via prefix

4. Traverse via postfix

5. Exit

Choice:

1

Enter element to insert: 3

Enter your choice:

1. Insert

2. Traverse via infix

3. Traverse via prefix

4. Traverse via postfix

5. Exit

Choice:

1

Enter element to insert: 4

Enter your choice:

1. Insert

2. Traverse via infix

3. Traverse via prefix

4. Traverse via postfix

5. Exit

Choice: 1

Enter element to insert: 6

Enter your choice:

1. Insert
2. Traverse via infix
3. Traverse via prefix
4. Traverse via postfix
5. Exit

Choice:

1

Enter element to insert: 2

Enter your choice:

1. Insert
2. Traverse via infix
3. Traverse via prefix
4. Traverse via postfix
5. Exit

Choice:

2

2 3 4 5 6

Enter your choice:

1. Insert

2. Traverse via infix

3. Traverse via prefix

4. Traverse via postfix

5. Exit

Choice:

3

5 3 2 4 6

Enter your choice:

1. Insert

2. Traverse via infix

3. Traverse via prefix

4. Traverse via postfix

5. Exit

Choice:

4

2 4 3 6 5

Enter your choice:

1. Insert

2. Traverse via infix

3. Traverse via prefix

4. Traverse via postfix

5. Exit

Choice:

5

Memory Cleared

Program 11(b):

AIM:

Write a non recursive C program for traversing a binary tree in preorder, inorder and postorder

/* C Program for Inorder Preorder Postorder traversal of Binary Tree */

```
#include<stdio.h
>
#include<stdlib.h
>  #define   MAX
50

struct node
{
      struct node
      *lchild; int info;
      struct node *rchild;
};

struct node *insert_nrec(struct node *root, int
ikey ); void nrec_pre(struct node *root);
void nrec_in(struct node *root);
void nrec_post(struct node
*root);
void display(struct node *ptr,int level);
```

```
struct node
*queue[MAX]; int front=-
1,rear=-1;
void insert_queue(struct node
*item); struct node *del_queue();
int queue_empty();

struct node
*stack[MAX]; int top=-
1;
void push_stack(struct node
*item); struct node *pop_stack();
int stack_empty();

int main( )
{
      struct node *root=NULL,
      *ptr; int choice,k;

      while(1)
      {
            printf("\n");
            printf("1.Insert\n")
            ;
```

```
        printf("2.Display\n");
        printf("3.Preorder
        Traversal\n"); printf("4.Inorder
        Traversal\n");
        printf("5.Postorder
        Traversal\n");
        printf("6.Quit\n");
        printf("\nEnter your choice : ");
        scanf("%d",&choice);

        switch(choice)
        {

        case 1:
              printf("\nEnter the key to be inserted :
              "); scanf("%d",&k);
              root = insert_nrec(root,
              k); break;

    case 2:
              printf("\n");
              display(root,0
              ); printf("\n");
              break;
```

```
        case 3:
                nrec_pre(root)
                ; break;

        case 4:
                nrec_in(root)
                ; break;

        case 5:
                nrec_post(root
                ); break;

        case 6:
                exit(1);

        default:
                printf("\nWrong choice\n");
        }/*End of switch*/
}/*End of while */

return 0;
```

```
}/*End of main( )*/


struct node *insert_nrec(struct node *root, int ikey)
{
      struct node *tmp,*par,*ptr;

      ptr = root;
      par =
      NULL;

      while( ptr!=NULL)
      {
            par = ptr;
            if(ikey < ptr->info)
                  ptr = ptr-
                  >lchild;
            else if( ikey > ptr-
                  >info ) ptr = ptr-
                  >rchild;
            else
            {
                  printf("\nDuplicate
                  key"); return root;
```

```
            }
        }

        tmp=(struct node *)malloc(sizeof(struct
        node)); tmp->info=ikey;
        tmp-
        >lchild=NULL;
        tmp-
        >rchild=NULL;

        if(par==NULL)
            root=tmp;
        else if( ikey < par-
            >info ) par-
            >lchild=tmp;
        else
            par->rchild=tmp;

        return root;
    }/*End of insert_nrec( )*/



    void nrec_pre(struct node *root)
    {
        struct node *ptr = root;
```

```
        if( ptr==NULL )
        {
                printf("Tree is
                empty\n"); return;
        }
        push_stack(ptr);
        while( !stack_empty()
        )
        {
                ptr = pop_stack();
                printf("%d ",ptr-
                >info); if(ptr-
                >rchild!=NULL)
                        push_stack(ptr-
                >rchild); if(ptr-
                >lchild!=NULL)
                        push_stack(ptr->lchild);
        }
        printf("\n");
}/*End of nrec_pre*/

void nrec_in(struct node *root)
{
        struct node *ptr=root;
```

```
 if( ptr==NULL )
 {
       printf("Tree is
       empty\n"); return;
 }
 while(1)
 {
while(ptr->lchild!=NULL )
       {
             push_stack(ptr
             ); ptr = ptr-
             >lchild;
       }

       while( ptr->rchild==NULL )
       {
             printf("%d ",ptr->info);
             if(stack_empty())
                   return;
             ptr = pop_stack();
       }
       printf("%d ",ptr->info);
       ptr = ptr->rchild;
```

```
        }
        printf("\n");
}/*End of nrec_in( )*/

void nrec_post(struct node *root)
{
        struct node *ptr =
        root; struct node *q;

        if( ptr==NULL )
        {
                printf("Tree is
                empty\n"); return;
        }
        q =
        root;
        while(1)
        {
                while(ptr->lchild!=NULL)
                {
                        push_stack(ptr
                        ); ptr=ptr-
                        >lchild;
                }
```

```
        while( ptr->rchild==NULL || ptr->rchild==q )
        {
            printf("%d ",ptr->info);
            q = ptr;
            if( stack_empty()
                ) return;
            ptr = pop_stack();
        }
        push_stack(ptr
        ); ptr = ptr-
        >rchild;
    }
    printf("\n");
}/*End of nrec_post( )*/


/*Functions for implementation of
queue*/ void insert_queue(struct node
*item)
{
    if(rear==MAX-1)
    {
        printf("Queue Overflow\n");
```

```
        return;
    }
    if(front==-1) /*If queue is initially
        empty*/ front=0;
    rear=rear+1;
    queue[rear]=item
    ;
}/*End of insert()*/

struct node *del_queue()
{
    struct node *item;
    if(front==-1 || front==rear+1)
    {
        printf("Queue
        Underflow\n"); return 0;
    }
    item=queue[front]
    ; front=front+1;
    return item;
}/*End of del_queue()*/

int queue_empty()
```

```
{
    if(front==-1 ||
        front==rear+1) return
        1;
    else
        return 0;
}

/*Functions for implementation of
stack*/ void push_stack(struct node
*item)
{
    if(top==(MAX-1))
    {
        printf("Stack
        Overflow\n"); return;
    }
    top=top+1;
    stack[top]=item
    ;
}/*End of push_stack()*/

struct node *pop_stack()
{
    struct node *item;
```

```
    if(top==-1)
    {
        printf("Stack Underflow. \n");
        exit(1);
    }
    item=stack[top]
    ; top=top-1;
    return item;
}/*End of pop_stack()*/

int stack_empty()
{
    if(top==-1)
        return 1;
    else
        return 0;
} /*End of stack_empty*/


void display(struct node *ptr,int level)
{
    int i;
```

```
    if(ptr == NULL )/*Base
        Case*/ return;
    else
  {
        display(ptr->rchild,
        level+1); printf("\n");
        for (i=0; i<level;
            i++) printf("
                    ");
        printf("%d", ptr->info);
        display(ptr->lchild,
        level+1);
    }
}
```

Output:

/* C Program for Inorder Preorder Postorder traversal of Binary Tree */

1.Insert

2.Displa

y

3.Preorder

Traversal 4.Inorder

Traversal

5.Postorder

Traversal 6.Quit

Enter your choice : 1

Enter the key to be inserted

: 7 1.Insert

2.Display

3.Preorder

Traversal 4.Inorder

Traversal

5.Postorder

Traversal 6.Quit

Enter your choice : 1

Enter the key to be inserted

: 5 1.Insert

2.Display

3.Preorder

Traversal 4.Inorder

Traversal

5.Postorder

Traversal 6.Quit

Enter your choice : 1

Enter the key to be inserted

: 6 1.Insert

2.Display

3.Preorder

Traversal

4.Inorder

Traversal

5.Postorder

Traversal 6.Quit

Enter your choice : 1

Enter the key to be inserted

: 4 1.Insert

2.Display

3.Preorder

Traversal 4.Inorder

Traversal

5.Postorder

Traversal 6.Quit

Enter your choice : 1

Enter the key to be inserted

: 9 1.Insert

2.Display

3.Preorder

Traversal 4.Inorder

Traversal

5.Postorder

Traversal 6.Quit

Enter your choice : 1

Enter the key to be inserted

: 8 1.Insert

2.Display

3.Preorder

Traversal 4.Inorder

Traversal

5.Postorder

Traversal 6.Quit

Enter your choice : 1

Enter the key to be inserted :

11 1.Insert

2.Display

3.Preorder

Traversal 4.Inorder

Traversal

5.Postorder

Traversal 6.Quit

Enter your choice :

2 11

9

8

7

6

5

4

1.Insert

2.Displa

y

3.Preorder

Traversal 4.Inorder

Traversal

5.Postorder

Traversal 6.Quit

Enter your choice :

3 7 5 4 6 9 8 11

1.Insert

2.Displa

y

3.Preorder

Traversal 4.Inorder

Traversal

5.Postorder

Traversal 6.Quit

Enter your choice :

4 4 5 6 7 8 9 11

1.Insert

2.Displa

y

3.Preorder

Traversal 4.Inorder

Traversal

5.Postorder

Traversal 6.Quit

Enter your choice : 5

4 6 5 8 11 9 7

1.Insert

2.Displa

y

3.Preorder

Traversal 4.Inorder

Traversal

5.Postorder

Traversal 6.Quit

Enter your choice : 6

Program 12(a):

AIM:

Write a C program to implement Prims' algorithm

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

#define infinity 9999
#define MAX 20

int G[MAX][MAX],spanning[MAX][MAX],n;

int prims();

int main()
{
    int i,j,total_cost;
    printf("Enter no. of
    vertices:"); scanf("%d",&n);

    printf("\nEnter the adjacency matrix:\n");
```

```
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        scanf("%d",&G[i][j]);

        total_cost=prims();
        printf("\nspanning tree
        matrix:\n");

        for(i=0;i<n;i++)
        {
                printf("\n");
                for(j=0;j<n;j+
                +)
                        printf("%d\t",spanning[i][j]);
        }
        printf("\n\nTotal cost of spanning
        tree=%d",total_cost); return 0;
}
int prims()
{
        int cost[MAX][MAX];
        int u,v,min_distance,distance[MAX],from[MAX];
```

```
int visited[MAX],no_of_edges,i,min_cost,j;

//create cost[][]
matrix,spanning[][]
for(i=0;i<n;i++)
        for(j=0;j<n;j++)
        {
                if(G[i][j]==0)
                        cost[i][j]=infinity;
                else
                        cost[i][j]=G[i][j];
                        spanning[i][j]=
                        0;

        }

//initialise visited[],distance[] and
from[] distance[0]=0;
visited[0]=1;

for(i=1;i<n;i++)
{
        distance[i]=cost[0][i
        ]; from[i]=0;
```

```
        visited[i]=0;
}

min_cost=0;        //cost of spanning tree
no_of_edges=n-1;              //no. of edges to be
added

while(no_of_edges>0)
{
        //find the vertex at minimum distance from the
        tree min_distance=infinity;
        for(i=1;i<n;i++)
                if(visited[i]==0&&distance[i]<min_distance)
                {
                        v=i;
                        min_distance=distance[
                        i];
                    }

        u=from[v];

        //insert the edge in spanning
        tree
        spanning[u][v]=distance[v];
```

```
            spanning[v][u]=distance[
            v]; no_of_edges--;
            visited[v]=1;

            //updated the distance[]
            array for(i=1;i<n;i++)
                  if(visited[i]==0&&cost[i][v]<distance[i])
                  {
                        distance[i]=cost[i][v
                        ]; from[i]=v;
                  }

            min_cost=min_cost+cost[u][v];
      }

      return(min_cost);
}
```

Output

```
Enter no. of vertices:6
Enter the adjacency matrix:
0 3 1 6 0 0
```

3	0	5	0	3	0
1	5	0	5	6	4
6	0	5	0	0	2
0	3	6	0	0	6

0 0 4 2 6 0

spanning tree matrix:

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

0 0 4 2 0 0

Total cost of spanning tree=13

Program 12(b):

AIM:

Write a C program to implement Kruskal's algorithm.

```
/* C Program for Minimum Spanning Tree using Kruskal's Algorithm
Example */


#include<stdio.h
>
#include<stdlib.h
>


#define MAX 100
#define NIL -1


struct edge
{
    int u;
    int v;
    int weight;
    struct edge *link;
}*front = NULL;


void make_tree(struct edge
tree[]); void insert_pque(int i,int
j,int wt); struct edge *del_pque();
```

```
int isEmpty_pque(
); void
create_graph();

int n; /*Total number of vertices in the graph */

int main()
{
      int i;
      struct edge tree[MAX]; /* Will contain the edges of spanning
      tree */ int wt_tree = 0; /* Weight of the spanning tree */

      create_graph();

      make_tree(tree);

      printf("\nEdges to be included in minimum spanning tree are
      :\n"); for(i=1; i<=n-1; i++)
      {
            printf("\n%d->",tree[i].u);
            printf("%d\n",tree[i].v);
            wt_tree += tree[i].weight;
      }
```

```
        printf("\nWeight of this minimum spanning tree is : %d\n", wt_tree);


        return 0;

}/*End of main()*/

void make_tree(struct edge tree[])
{
        struct edge *tmp;
        int v1,v2,root_v1,root_v2;
        int father[MAX]; /*Holds father of each vertex */
        int i,count = 0; /* Denotes number of edges included in the tree */

        for(i=0; i<n; i++)
                father[i] =
                NIL;

        /*Loop till queue becomes empty or
        till n-1 edges have been inserted in the
        tree*/ while( !isEmpty_pque( ) && count <
        n-1 )
        {
                tmp =
                del_pque(); v1 =
                tmp->u;
```

```
        v2 = tmp->v;

        while( v1 !=NIL )
        {
              root_v1 = v1;
              v1 =
              father[v1];
        }
        while( v2 != NIL  )
        {
              root_v2 = v2;
              v2 =
              father[v2];
        }

        if( root_v1 != root_v2 )/*Insert the edge (v1, v2)*/
        {
           count++;
              tree[count].u = tmp->u;
              tree[count].v = tmp->v;
              tree[count].weight = tmp-
              >weight;
              father[root_v2]=root_v1;
        }
    }
```

```
    if(count < n-1)
    {
        printf("\nGraph is not connected, no spanning tree
        possible\n"); exit(1);
    }

}/*End of make_tree()*/

/*Inserting edges in the linked priority queue
*/ void insert_pque(int i,int j,int wt)
{
    struct edge *tmp,*q;

    tmp = (struct edge *)malloc(sizeof(struct
    edge)); tmp->u = i;
    tmp->v = j;
    tmp->weight = wt;

    /*Queue is empty or edge to be added has weight less than first
    edge*/ if( front == NULL || tmp->weight < front->weight )
    {
```

```
            tmp->link =
            front; front =
            tmp;
      }
      else
      {
            q = front;
            while( q->link != NULL && q->link->weight <= tmp-
                  >weight ) q = q->link;
            tmp->link = q-
            >link; q->link =
            tmp;
            if(q->link == NULL) /*Edge to be added at the
                  end*/ tmp->link = NULL;
      }
}/*End of insert_pque()*/

/*Deleting an edge from the linked priority
queue*/ struct edge *del_pque()
{
      struct edge
      *tmp; tmp =
      front;
      front = front-
      >link; return tmp;
```

```
}/*End of del_pque()*/

int isEmpty_pque( )
{
      if ( front ==
            NULL )
            return 1;
      else
            return 0;
}/*End of isEmpty_pque()*/

void create_graph()
{
      int i,wt,max_edges,origin,destin;

      printf("\nEnter number of vertices :
      "); scanf("%d",&n);
      max_edges = n*(n-1)/2;

      for(i=1; i<=max_edges; i++)
      {
            printf("\nEnter edge %d(-1 -1 to quit):
            ",i); scanf("%d %d",&origin,&destin);
```

```
        if( (origin == -1) && (destin == -
            1) ) break;
        printf("\nEnter weight for this edge :
        "); scanf("%d",&wt);
        if( origin >= n || destin >= n || origin<0 || destin<0)
        {
            printf("\nInvalid
            edge!\n"); i--;
        }
        else
            insert_pque(origin,destin,wt);
    }
}
```

Output:

```
/* C Program for Minimum Spanning Tree using Kruskal's Algorithm
Example */


Enter number of vertices : 6
Enter edge 1(-1 -1 to quit):
1 1 Enter weight for this
edge : 2 Enter edge 2(-1 -1
to quit): 0 2 Enter weight for
this edge : 3 Enter edge 3(-
1 -1 to quit): 0 3
```

Enter weight for this edge : 1

Enter edge 4(-1 -1 to quit): 2 4

Enter weight for this edge : 2

Enter edge 5(-1 -1 to quit): 2 5

Enter weight for this edge : 5

Enter edge 6(-1 -1 to quit): 1 5

Enter weight for this edge : 3

Enter edge 7(-1 -1 to quit): 1 4

Enter weight for this edge : 1

Enter edge 8(-1 -1 to quit): 3 5

Enter weight for this edge : 2

Enter edge 9(-1 -1 to quit): 4 2

Enter weight for this edge : 1

Enter edge 10(-1 -1 to quit): 5

2 Enter weight for this edge :

2 Enter edge 11(-1 -1 to quit):

1 3 Enter weight for this edge

: 2 Enter edge 12(-1 -1 to

quit): -1 -1

Edges to be included in minimum spanning tree are :

0->3

1->4

4->2

5->2

0->1

Weight of this minimum spanning tree is

: 6 Process returned 0

Program 13:

AIM:

Implementation of Hash table using double hashing as collision resolution function

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

struct data
{
        int key;
        int
        value;
};

struct hashtable_item
{

        int flag;
        /*
         * flag = 0 : data not present
         * flag = 1 : some data already present
         * flag = 2 : data was present,but deleted
        */
        struct data *item;

};

struct hashtable_item
*array; int max = 7;
int size = 0;
int prime =
3;

int hashcode1(int key)
{
        return (key % max);
```

```
}

int hashcode2(int key)
{
        return (prime - (key % prime));
}

void insert(int key, int value)
{
        int hash1 =
        hashcode1(key); int
        hash2 = hashcode2(key);

        int index = hash1;

        /* create new data to insert */
        struct data *new_item = (struct data*) malloc(sizeof(struct
        data)); new_item->key = key;
        new_item->value = value;

        if (size == max)
      {
                printf("\n Hash Table is full, cannot insert more items
                \n"); return;
        }

        /* probing through other array elements
        */ while (array[index].flag == 1) {
                if (array[index].item->key == key)
           {
                        printf("\n Key already present, hence updating its value
                        \n"); array[index].item->value = value;
                        return;
                }
                index = (index + hash2) %
                max; if (index == hash1)
           {
                        printf("\n Add is failed \n");
```

```
                return;
            }
            printf("\n probing \n");

        }

        array[index].item =
        new_item; array[index].flag
        = 1;
        size++;
        printf("\n Key (%d) has been inserted \n", key);

}

/* to remove an element from the array
*/ void remove_element(int key)
{
        int          hash1          =
        hashcode1(key);         int
        hash2 = hashcode2(key);
        int index = hash1;

        if (size == 0)
    {
            printf("\n Hash Table is empty
            \n"); return;
        }

        /* probing through other elements
        */ while (array[index].flag != 0)
    {
            if (array[index].flag == 1 && array[index].item->key == key)
        {
                array[index].item =
                NULL; array[index].flag
                = 2;
                size--;
                printf("\n Key (%d) has been removed \n",
                key); return;
            }
```

```
                index = (index + hash2) %
                max; if (index == hash1)
            {
                        break;
                }
        }

        printf("\n Key (%d) does not exist \n", key);

}

int size_of_hashtable()
{
        return size;
}

/* displays all elements of array
*/ void display()
{
        int i;
        for (i = 0; i < max; i++)
      {
                if (array[i].flag != 1)
            {
                        printf("\n Array[%d] has no elements \n", i);
                }
                else
            {
                        printf("\n Array[%d] has elements \n Key (%d) and Value
                        (%d)
\n", i, array[i].item->key, array[i].item->value);
                }
        }
}

/* initializes array
*/ void init_array()
{
```

```
        int i;
        for(i = 0; i < max; i++)
    {
            array[i].item =
            NULL; array[i].flag
            = 0;
        }
        prime = get_prime();
}

/* returns largest prime number less than size of
array */ int get_prime()
{
        int i,j;
        for (i = max - 1; i >= 1; i--)
    {
            int flag = 0;
            for (j = 2; j <= (int)sqrt(i); j++)
        {
                if (i % j == 0)
             {
                    flag++;
                }
            }
            if (flag == 0)
        {
                return i;
            }
        }
        return 3;

}

void main()
{
        int choice, key, value, n,
        c; clrscr();
```

```
        array = (struct hashtable_item*) malloc(max *
sizeof(struct hashtable_item));
        init_array();

                                                do {
                 printf("Implementation of Hash Table in C with Double
Hashing.\n\n");
                printf("MENU-: \n1.Inserting item in the Hash
                     Table" "\n2.Removing item from the Hash
                     Table" "\n3.Check the size of Hash Table"
                     "\n4.Display Hash Table"
                     "\n\n Please enter your choice-

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

                switch(choice)
           {

                case 1:

                    printf("Inserting element in Hash
                    Table\n"); printf("Enter key and value-
                    :\t");
                    scanf("%d %d", &key,
                    &value); insert(key, value);

                    break

                ; case 2:

                    printf("Deleting in Hash Table \n Enter the key to
                    delete-:"); scanf("%d", &key);
                    remove_element(key

                    ); break;

                case 3:
```

```
                n = size_of_hashtable();
                printf("Size of Hash Table is-:%d\n",

                n); break;

            case 4:

                display(

                ); break;

            default:

                 printf("Wrong Input\n");

            }

            printf("\n Do you want to continue-:(press 1 for
            yes)\t"); scanf("%d", &c);

        }while(c == 1);

        getch();

}
```

Output:

```
Implementation of Hash Table in C with Double
Hashing MENU-:
1. Inserting item in the Hash Table
2. Removing item from the Hash Table
3. Check the size of Hash Table
4. Display Hash Table

Please enter your choice-
: 3 Size of hash table is-:
0
```

```
Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Double
Hashing MENU-:
1. Inserting item in the Hash Table
2. Removing item from the Hash Table
3. Check the size of Hash Table
4. Display Hash Table

Please enter your choice-: 1
Inserting element in hash
table Enter key and value-:
12                              1

 Key (12) has been inserted

Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Double
Hashing MENU-:
1. Inserting item in the Hash Table
2. Removing item from the Hash Table
3. Check the size of Hash Table
4. Display Hash Table

Please enter your choice-: 1
Inserting element in Hash
Table Enter key and value-:
47                              1

 probing

 Key (47) has been inserted

Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Double
Hashing MENU-:
1. Inserting item in the Hash Table
2. Removing item from the Hash Table
3. Check the size of Hash Table
```

```
4. Display Hash Table

Please enter your choice-: 1
Inserting element in Hash
Table Enter key and value-:
61                                  1

 probing

 Key (61) has been inserted


Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Double
Hashing MENU-:
1. Inserting item in the Hash Table
2. Removing item from the Hash Table
3. Check the size of Hash Table
4. Display Hash Table

Please enter your choice-
: 3 Size of hash table is-:
3

Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Double
Hashing MENU-:
1. Inserting item in the Hash Table
2. Removing item from the Hash Table
3. Check the size of Hash Table
4. Display Hash Table

Please enter your

choice-: 4 Array[0] has

no elements

Array[1] has
elements-: 47 (key)
and 1 (value)
```

Array[2] has elements-:
61 (key) and 1

(value) Array[3] has

elements-:

Array[4] has no elements

Array[5] has elements-:

12 (key) and 1 (value)

Array[6] has no elements

Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Double Hashing MENU-:
1. Inserting item in the Hash Table
2. Removing item from the Hash Table
3. Check the size of Hash Table
4. Display Hash Table

Please enter your choice-
: 2 Deleting in hash table
Enter the key to delete-:

61 Key (61) has been

removed

Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Double Hashing MENU-:
1. Inserting item in the Hash Table
2. Removing item from the Hash Table
3. Check the size of Hash Table
4. Display a Hash Table

Please enter your choice-
: 2 Deleting in hash table
Enter the key to delete-:

61 This key does not

exist

Do you want to continue-:(press 1 for yes) 2

Program 14:

AIM:

Implementation of Binary Search trees- Insertion and deletion.

```
// C program to demonstrate
// insert operation in binary
// search
tree using
System;
class BinarySearchTree{
        // Class containing left and
// right child of current node
// and key value
public class
Node
{
        public int key;
        public Node left, right;

        public Node(int item)
        {
                key = item;
                left = right = null;
        }
}

// Root of
BST Node
root;
// Constructor
BinarySearchTre
e()
{
        root = null;
}
// This method mainly calls
insertRec() void insert(int key)
{
        root = insertRec(root, key);
}
```

```
// A recursive function to insert
// a new key in BST
Node insertRec(Node root, int key)
{
        // If the tree is empty,
        // return a new
        node if (root ==
        null)
        {
                root = new
                Node(key); return
                root;
        }
// Otherwise, recur down the
        tree if (key < root.key)
                root.left = insertRec(root.left,
        key); else if (key > root.key)
                root.right = insertRec(root.right, key);

        // Return the (unchanged) node
        pointer return root;
}
// This method mainly calls
InorderRec() void inorder()
{
        inorderRec(root);
}
// A utility function to
// do inorder traversal of
BST void
inorderRec(Node root)
{
        if (root != null)
        {
                inorderRec(root.left);
                Console.WriteLine(root.ke
                y); inorderRec(root.right);
        }
}
```

```
// Driver Code
public static void Main(String[] args)
{
        BinarySearchTree tree = new BinarySearchTree();

        /* Let us create following
                BST 50
        /         \
        30       70
        / \ / \
20 40 60 80
*/
        tree.insert(50
        );
        tree.insert(30
        );
        tree.insert(20
        );
        tree.insert(40
        );
        tree.insert(70
        );
        tree.insert(60
        );
        tree.insert(80
        );

        // Print inorder traversal of the
        BST tree.inorder();
}
}
```

Output:

```
20
30
40
50
60
70
80
```

Program 15(a):

AIM:

Write C program that implement Bubble sort, to sort a given list of integers in ascending order.

```
/** C program to sort N numbers in ascending order using Bubble sort
 * and print both the given and the sorted
array */ #include <stdio.h>
#define MAXSIZE
 10 void main()
{
   int
   array[MAXSIZE];
   int i, j, num, temp;

   printf("Enter the value of num
   \n"); scanf("%d", &num);
   printf("Enter the elements one by one
   \n"); for (i = 0; i < num; i++)
   {
      scanf("%d", &array[i]);
   }
   printf("Input array is
   \n"); for (i = 0; i < num;
   i++)
   {
      printf("%d\n", array[i]);
   }
   /* Bubble sorting begins */
   for (i = 0; i < num; i++)
   {
      for (j = 0; j < (num - i - 1); j++)
      {
         if (array[j] > array[j + 1])
         {
            temp = array[j];
            array[j] = array[j +
            1];
```

```
            array[j + 1] = temp;
          }
        }
    }
    printf("Sorted array
    is...\n"); for (i = 0; i < num;
    i++)
    {
       printf("%d\n", array[i]);
    }
}
```

Output:

```
Enter the value of
num 6
Enter the elements one by
one 23
45
67
89
12
34
Input array is
23
45
67
89
12
34
Sorted array is...
12
23
34
45
67
89
```

Program 15(b):

AIM:

Write C program that implement Quick sort, to sort a given list of integers in ascending order.

```
/*
* C Program to Perform Quick Sort on a set of Entries from a File
* using Recursion
*/
#include <stdio.h>
void quicksort (int [], int,

int); int main()

{
   int
   list[50];
   int size, i;

   printf("Enter the number of elements:
   "); scanf("%d", &size);
   printf("Enter the elements to be
   sorted:\n"); for (i = 0; i < size; i++)
   {
      scanf("%d", &list[i]);
   }
   quicksort(list, 0, size - 1);
   printf("After applying quick
   sort\n"); for (i = 0; i < size; i++)
   {
      printf("%d ", list[i]);
   }
   printf("\n");

   return 0;
}
```

```
void quicksort(int list[], int low, int high)
{
   int pivot, i, j,
   temp; if (low <
   high)
   {
      pivot  =
      low; i =
      low;
      j = high;
      while (i <
      j)
      {
         while (list[i] <= list[pivot] && i <= high)
         {
            i++;
         }
         while (list[j] > list[pivot] && j >= low)
         {
            j--;
         }
         if (i < j)
         {
            temp         =
            list[i]; list[i] =
            list[j]; list[j] =
            temp;
         }
      }
      temp = list[j];
      list[j] =
      list[pivot];
      list[pivot] =
      temp;
      quicksort(list, low, j -
      1); quicksort(list, j + 1,
      high);
   }
}
```

Output:

```
Enter the number of elements:
6 Enter the elements to be
sorted: 67
45
24
98
12
38
After applying quick
sort 12 24 38 45 67 98
```

Program 15(c):

AIM:

Write C program that implement merge sort, to sort a given list of integers in ascending order

```
/*
 * C Program to Input Few Numbers & Perform Merge Sort on them
using Recursion
 */

#include <stdio.h>

void mergeSort(int [], int, int,
int); void partition(int [],int, int);

int main()
{
   int
   list[50];
   int i, size;

   printf("Enter total number of
   elements:"); scanf("%d", &size);
   printf("Enter the
   elements:\n"); for(i = 0; i <
   size; i++)
   {
      scanf("%d", &list[i]);
   }
   partition(list, 0, size - 1);
   printf("After merge
   sort:\n"); for(i = 0;i < size;
   i++)
   {
      printf("%d  ",list[i]);
   }

  return 0;
```

```
}

void partition(int list[],int low,int high)
{
   int mid;

   if(low < high)
   {
      mid = (low + high) / 2;
      partition(list, low, mid);
      partition(list, mid + 1, high);
      mergeSort(list, low, mid,
      high);
   }
}

void mergeSort(int list[],int low,int mid,int high)
{
   int i, mi, k, lo, temp[50];

   lo =
   low; i =
   low;
   mi = mid + 1;
   while ((lo <= mid) && (mi <= high))
   {
      if (list[lo] <= list[mi])
      {
         temp[i] =
         list[lo]; lo++;
      }
      else
      {
         temp[i] =
         list[mi]; mi++;
      }
      i++;
   }
   if (lo > mid)
```

```
  {
    for (k = mi; k <= high; k++)
    {
      temp[i] =
      list[k]; i++;
    }
  }
  Else v c
  {
    for (k = lo; k <= mid;
 k++) x            derpjt {
      temp[i] = list[k];c. xl
      ,m.llk i++;
    }
  }

  for (k = low; k <= high; k++)
  {
    list[k] = temp[k];
  }
}
```

Output

Enter total number of elements:5 Enter the elements:
12
36
22
76
54
Aftem mcc r merge sort: 12 22 36 54 76