Maharaj Vijayaram Gajapathi Raj College of Engineering (Autonomous)

VIJAYARAM NAGAR CAMPUS, VIZIANAGARAM.

CERTIFICATE

This is to certify that this is the bonafide record of the work done in .....

Laboratory by Mr / Ms .....

bearing Regd. No. / Roll No ..... of .....

B. Tech. / M. Tech. course during .....

Total Numbers of Experiments held .....

LAB-IN-CHARGE **HEAD OF THE DEPARTMENT**

INDEX

11	To create a single linked list and perform operations of insertion, deletion, search and traversal		
12	To write a c program to create a circular linked list and perform basic operations like insertion, deletion, search and traversal.		
13	To Write a c program to create a double linked list and perform basic operations insertion, deletion, search and traversal		
14	To implement stack operations using array		
15	To implement stack operations using linked list		
16	To Implement Postfix Evaluation		
17	To implement the queue operations using arrays		
18	To implement the queue operations using linked list. .		
19	To implement circular queues using Arrays		

20	To Implement Binary Tree Creation and Recursive Tree Traversals .		
21	To Create and Implement Binary Search Tree Creation Various Operations on Binary Search Tree (Insertion, Deletion, and Search)		
22	Program to create a graph of an adjacency list.		
23	Program to implement Graph Traversal Breadth First Search		
24	To find the Minimum Spanning tree by using Prim's Algorithm.		
25	To find the Minimum Spanning tree by using Kruskal's Algorithm		
26	To sort the given elements using heap sort technique		

WEEK 1

Question: Write a C program to compute the factorial of N for a given N

Aim: To Determine the Factorial of a given Number Using Recursion

Type your text

Objective : The factorial of a number is the product of all the integers from 1 to that number. Factorial of *n* is denoted by n!.The Objective of the Program is to Find the Factorial of a Number That Uses

Recursion : The process of calling a function

by itself is called recursion and the function

which calls itself is called recursive function.

Main() : Which is the first function of every C program that is responsible for starting the execution and termination of the program

Fact() : Which is Used to Find Factorial of a Number and returns the Value to Main()Function

Source Code:

```
#include<stdio.h>
int fact(int n)
{
    int i ;
    if(n==0|| n==1)
{
    return 1;
}
else
{
    return fact(n-1)*n;
}
}
main()
```

```
{
        int n;
        printf("enter n value");
        scanf("%d",&n);
        if(n<0)
{
                printf("no negative values");
}
        if(n>0)
                printf("factorial of %d is %d ",n,fact(n));
        else
                printf("no other symbols");
}
```

.

Question: Write a C program to generate Fibonacci Series

Aim: to generate the Fibonacci series up to given limits using recursion

Objective: The Fibonacci sequence is a series of numbers in which each number is the sum of the two that precede it. It always starts from 0 and 1.The objective of the program is to generate Fibonacci series that uses

Recursion : The process of calling a function by itself is called recursion and the function which calls itself is called recursive function.

Main() : Which is the first function of every C program that is responsible for starting the execution and termination of the program

Fib() : Which is used to obtain the Fibonacci Series and returns the value to main() function

Source Code:

```
#include<stdio.h>

int fib(int n);
main()
{
int i,n,x;
printf("Enter The Range:\n");
scanf("%d",&n);
printf("Enter The Upto Term To Print\n");
scanf("%d",&x);
printf("fibonaci Series :  \n");
for(i=0;i<(n-x);i++)
{
printf("%d\n",fib(i));
}
}
int fib(int i)
{
if(i==0)
return 0;
if(i==1)
return 1;
else
return(fib(i-1)+fib(i-2));
```

Question: Write a C program to compute GCD of two numbers

Aim: to compute the greatest common divisor of given two numbers using recursion

Objective: The GCD of two numbers is the largest integer that can exactly divide both numbers (without a remainder).The objective of the program is to find the GCD of two numbers that uses

Recursion : The process of calling a function by itself is called recursion and the function which calls itself is called recursive function.

Main() : Which is the first function of every program that is responsible for starting the C execution and termination of the program

GCD() : Which is used to find the GCD of Given two numbers and returns the value to Main() function

Source Code:

```
#include<stdio.h>
int gcd(int,int);
void main()
{
int a,b,result=0;
printf("Enter The Two N.os\n");
scanf("%d%d",&a,&b);
result=gcd(a,b);
printf("The Gcd Of Two N.os Is %d",result);
}
int gcd(int a,int b)
{
 if(b==0)
```

```
return a;
else if(a==0)
return b;
else
return gcd(b,a%b);
}
```

Question: **Write a C program to implement Towers of Hanoi**

Aim: to implement the towers of Hanoi using recursion

Objective: The tower of Hanoi is a mathematical puzzle. It consists of three rods and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top. We have to obtain the same stack on the third rod.

The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules−

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.

Main() : Which is the first function of every program that is responsible for starting the C execution and termination of the program.

Move() : Which is used to find where to move The disk and retuirns the value to main()Function

Source Code:

```
#include<stdio.h>

void move(int,char,char,char);

void main()

{

int n;

printf("Enter The N.o Of Disks\n");

scanf("%d",&n);

move(n,'A','B','C');

}

void move(int n,char src,char dest,char temp)
```

```
{
if(n==1)
printf("Move The Disk From %c to %c\n",src,dest);
else
{
move(n-1,src,temp ,dest);
move(1,src,dest,temp);
move(n-1,temp,dest,src);
}
}
```

WEEK 2

Question: Write a C program for Linear search for a key value in a given list

Type your text

Aim: To search an element in the given list using a linear search technique

Objective: A linear search or sequential search is a method for finding an element within a list. It sequentially checks each element of the list until a match is found or the whole list has been searched. The objectives of the program is to find the required element that uses

Main(): Which is the first function of every C program that is responsible for starting the execution and termination of the program

Source Code:

```
#include<stdio.h>
int fact(int n)
{
        int i ;
        if(n==0|| n==1)
{
        return 1;
}
else
{
        return fact(n-1)*n;
}
}
main()
{
        int n;
```

Type your text

```
        printf("enter n value");
        scanf("%d",&n);
        if(n<0)
{
        printf("no negative values");
}
        if(n>0)
                printf("factorial of %d is %d ",n,fact(n));
        else
                printf("no other symbols");
}
```

Question: Write a C program for Linear search for a key value in a given list Using Recursion

Aim: To search an element in the given list using a linear search technique with recursion

Objective: A linear search or sequential search is a method for finding an element within a list. It sequentially checks each element of the list until a match is found or the whole list has been searched. The objectives of the program is to find the required element with recursion that uses

Main() : Which is the first function of every C program that is responsible for starting the execution and termination of the program.

RecursiveLS():which takes in 3 input parameters and returns the position of element in a array which is searched by the user

Source Code:

```
#include<stdio.h>

int ls(int a[],int value,int index,int n)

{

int pos=0;

if(index>=n)

{

return 0;

}

else if(a[index]==value)

{

pos=index+1;

return pos;

}

else
```

```
{
ls(a,value,index+1,n);
}
}
void main()
{
int n,value,pos,i,a[100];
printf("Enter The size Of Array\n");
scanf("%d",&n);
printf("Enter The Values\n");
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
printf("Enter The Element To Be Searched\n");
scanf("%d",&value);
pos=ls(a,value,0,n);
if(pos!=0)
{
printf("Element is found at %d position\n",pos);
}
else
{
printf("Element is not found\n");
}
```

Question: Write a C program for Binary search for a Key value in a given list.

Aim: To search an element in the given list using a binary search technique

Objective: A binary search is a search in which the middle element is calculated to check whether it is smaller or larger than the element which is to be searched the objectives of this program is to find the required element that uses

Main() : Which is the first function of every C program that is responsible for starting the execution and termination of the program

Source Code:

```
#include<stdio.h>
void main()
{
   int i,low,high,mid,n,search,a[100];
   printf("Enter the size of array\n");
   scanf("%d",&n);
   printf("Enter The Elements\n");
   {
   for(i=0;i<n;i++)
      scanf("%d",&a[i]);
   }
      printf("Enter the search element\n");
      scanf("%d",&search);
   low=0;
   high=n-1;
   mid=(low+high)/2;
   while(low<=high)
   {
```

```
        if(a[mid]==search)
        {
           printf("%d is found at %d location\n",search,mid+1);
           break;
        }
        else if(a[mid]<search)
        {
           low=mid+1;
        }
        else
        {
           high=mid-1;
        }
        if(low>high)
        printf("not found\n");
    }
}
```

Question: Write a C program for Binary search for a Key value in a given list. Using recursion

Aim: To search an element in the given list using a binary search technique with recursion.

Objective:

A binary search is a search in which the middle element is calculated to check whether it is smaller or larger than the element which is to be searched the objectives of this program is to find the required element with recursion that uses

Main() : Which is the first function of every C program that is responsible for starting the execution and termination of the program.

RecursiveBS():which takes in 4 input parameters and returns the position of element in a array which is searched by the user

Source Code:

```
#include<stdio.h>
void bs(int[],int,int,int);
void main()
{
   int key,size,i,list[100];
   printf("Enter The size of array\n");
   scanf("%d",&size);
   printf("Enter the elements\n");
   for(i=0;i<size;i++)
   {
  scanf("%d",&list[i]);
   }
   printf("Enter the element to be searched\n");
```

```
scanf("%d",&key);
    bs(list,0,size-1,key);
}
void bs(int list[] ,int low,int high,int key)
{
    int mid;
    mid=(low+high)/2;
if(low>high)
{
  printf("key is not found\n");
  return;
}
 else if(list[mid]==key)
{
 printf("%d is found at %d location\n",key,mid+1);
}
else if(list[mid]>key)
{
    return bs(list,low,mid-1,key);
}
else
 {
    return bs(list,mid+1,high,key);
}
}
```

WEEK 3

Question: **Write a C program for Quick sort, to sort a given list of integers in ascending order**

Aim: To sort the given elements by using quick sort

Objective:

Main ()-Which is the first function of every C program that is responsible for starting the execution and termination of the program

Quick sort: Quicksort is a sorting algorithm based on the divide and conquer approach

Pivot element:

First, quicksort determines something called a pivot, which is a somewhat arbitrary element in the collection

Next, using the pivot point, it partitions (or divides) the larger unsorted collection into two, smaller lists. It uses some smart logic to decide how to do the partitioning. It moves all the elements smaller than the pivot to the left (before) the pivot element, and moves all the elements larger than the pivot to the right (after) the pivot element.

Source code:

```
#include <stdio.h>

#include <conio.h>

#define size 100

int partition(int a[], int beg, int end);

void quick_sort(int a[], int beg, int end);

void main()

{

int arr[size], i, n;

printf("\n Enter the number of elements in the array: ");

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

```
printf("\n Enter the elements of the array: ");
for(i=0;i<n;i++)
{
 scanf("%d", &arr[i]);
}
quick_sort(arr, 0, n-1);
printf("\n The sorted array is: \n");
for(i=0;i<n;i++)
printf(" %d\t", arr[i]);
getch();
}
int partition(int a[], int beg, int end)
{
int left, right, temp, loc, flag;
loc = left = beg;
right = end;
flag = 0;
while(flag != 1)
{
while((a[loc] <= a[right]) && (loc!=right))
right--;
if(loc==right)
 flag =1;
 else if(a[loc]>a[right])
 {
 temp = a[loc];
 a[loc] = a[right];
```

```
 a[right] = temp;
 loc = right;
 }
if(flag!=1)
 {
while((a[loc] >= a[left]) && (loc!=left))
left++;
 if(loc==left)
flag =1;
 else if(a[loc] <a[left])
 {
 temp = a[loc];
a[loc] = a[left];
a[left] = temp;
loc = left;
 }
 }
}
return loc;
}
void quick_sort(int a[], int beg, int end)
{
int loc;
if(beg<end)
{
 loc = partition(a, beg, end);
quick_sort(a, beg, loc-1);
```

```
quick_sort(a, loc+1, end);
}
}
```

Question: Write a C program for Merge sort, to sort a given list of integers in ascending order

Aim: To Sort the Given Elements in the Ascending Order Using Merge Sort

Objective:

Merge sort is one of the most efficient sorting algorithms. It works on the principle of Divide and Conquer. Merge sort repeatedly breaks down a list into several subsists until each sub list consists of a single element and merging those sub lists in a manner that results into a sorted list.. It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves. The merge () function is used for merging two halves. Sorts a given data in O (nLogn) time complexity

Source code:

```
#include <stdio.h>
#include <conio.h>
#define size 100
void merge(int a[], int, int, int);
void merge_sort(int a[],int, int);
void main()
{
int arr[size], i, n;
printf("\n Enter the number of elements in the array : ");
scanf("%d", &n);
printf("\n Enter the elements of the array: ");
for(i=0;i<n;i++)
{
 scanf("%d", &arr[i]);
}
```

```
merge_sort(arr, 0, n-1);
printf("\n The sorted array is: \n");
for(i=0;i<n;i++)
printf(" %d\t", arr[i]);
getch();
}
void merge(int arr[], int beg, int mid, int end)
{
int i=beg, j=mid+1, index=beg, temp[size], k;
while((i<=mid) && (j<=end))
{
 if(arr[i] < arr[j])
 {
temp[index] = arr[i];
i++;
 }
 else
 {
temp[index] = arr[j];
j++;
 }
index++;
}
if(i>mid)
{
 while(j<=end)
 {
```

```
    temp[index] = arr[j];
j++;
index++;
 }
}
else
{
 while(i<=mid)
 {
temp[index] = arr[i];
i++;
index++;
 }
}
for(k=beg;k<index;k++)
arr[k] = temp[k];
}
void merge_sort(int arr[], int beg, int end)
{
int mid;
if(beg<end)
{
mid = (beg+end)/2;
 merge_sort(arr, beg, mid);
merge_sort(arr, mid+1, end);
 merge(arr, beg, mid, end);
}
```

WEEK 4

Question: Write a C program to create a Single linked list and perform basic operations (Insertion,Deletion, Search, Traversal)

Aim: To create a single linked list and perform operations of insertion, deletion, search and traversal

OBJECTIVE: The objective of the program is to conduct all the operations on single linked list that uses

malloc() : which allocate memory for node

struct() : which is used to create node that contains address part and data part.

Insertion : inserting a given element at

1. beginning
2. end
3. at certain position

Deletion : deleting a given element from

1. beginning
2. end
3. at certain position

Traversal : to travel across the list

Source Code:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
```

```
int count=0;
struct node *head=NULL;
void create_ll()
{
struct node *newnode,*ptr;
int num;
printf("Enter -1 To End\n");
printf("Enter The Data\n");
scanf("%d",&num);
while(num!=-1)
{
newnode=(struct node*)malloc(sizeof(struct node));
newnode->data=num;
if(head==NULL)
{
newnode->next=NULL;
head=newnode;
}
else
{
ptr=head;
while(ptr->next!=NULL)
{
ptr=ptr->next;
}
ptr->next=newnode;
newnode->next=NULL;
```

```
}
printf("Enter The Data\n");
scanf("%d",&num);
}
}
void display()
{
struct node *ptr;
ptr=head;
while(ptr!=NULL)
{
printf("%d->",ptr->data);
ptr=ptr->next;
}
printf("NULL");
}
void insert_beg()
{
struct node *newnode;
newnode=(struct node*)malloc(sizeof(struct node));
printf("Enter The Data\n");
scanf("%d",&newnode->data);
newnode->next=head;
head=newnode;
}
void insert_end()
{
```

```
struct node *newnode,*ptr;
newnode=(struct node*)malloc(sizeof(struct node));
printf("Enter The Data\n");
scanf("%d",&newnode->data);
newnode->next=NULL;
ptr=head;
while(ptr->next!=NULL)
{
ptr=ptr->next;
}
ptr->next=newnode;
}
void insert_pos()
{
struct node *newnode,*ptr;
int pos,i=1;
newnode=(struct node*)malloc(sizeof(struct node));
printf("Enter The position\n");
scanf("%d",&pos);
ptr=head;
while(i<pos)
{
ptr=ptr->next;
i++;
}
printf("Enter the data\n");
scanf("%d",&newnode->data);
```

```
newnode->next=ptr->next;
ptr->next=newnode;
}
void delete_beg()
{
struct node *ptr;
ptr=head;
head=head->next;
free(ptr);
}
void delete_end()
{
struct node *prevnode,*ptr;
ptr=head;
while(ptr->next!=NULL)
{
prevnode=ptr;
ptr=ptr->next;
}
prevnode->next=NULL;
free(ptr);
}
void delete_pos()
{
struct node *nextnode,*ptr;
int pos,i=1;
```

```
ptr=head;
printf("Enter The Posistion\n");
scanf("%d",&pos);
if(pos<0||pos>count)
{
printf("Enter The valid Posistion\n");
}
while(i<pos-1)
{
ptr=ptr->next;
i++;
}
nextnode=ptr->next;
ptr->next=nextnode->next;
free(nextnode);
}
void lengthlist()
{
struct node *ptr;
ptr=head;
while(ptr!=NULL)
{
count++;
ptr=ptr->next;
}
printf("Length of list : %d ",count);
}
```

```
void main()
{
int option;
do
{
printf("\n1.Create\n");
printf("2.Display\n");
printf("3.Insert at begining\n");
printf("4.Insert at end\n");
printf("5.Insert at posistion\n");
printf("6.delete at begining\n");
printf("7.Delete at End\n");
printf("8.Delete at any pos\n");
printf("9.length of list\n");
printf("10.Exit\n");
printf("Enter Your Option\n");
scanf("%d",&option);
switch(option)
{
case 1:create_ll();
break;
case 2: display();
break;
case 3: insert_beg();
display();
break;
case 4: insert_end();
```

```
display();
break;
case 5: insert_pos();
display();
break;
case 6: delete_beg();
display();
break;
case 7: delete_end();
display();
break;
case 8: delete_pos();
display();
break;
case 9: lengthlist();
break;
default: printf("Enter valid option\n");
break;
}
}
while(option!=9);
}
```

WEEK 5

Question: Write a C program to create a Circular linked list and perform basic operations(Insertion, Deletion, Search, Traversal)

Aim: To write a c program to create a circular linked list and perform basic operations like insertion, deletion, search and traversal.

Objective: In this program we will learn how circular linked list works

Traversal – access each element of the linked list

Insertion – adds a new element to the linked list

Deletion – removes the existing elements

- While inserting a new node in already created list
- We will how a node is deleted in the list
- Searching for a data element in a list
- The traversal of the created circular linked list

Source Code:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node *start=NULL;
void create_cll()
{
struct node *newnode,*ptr;
int num;
printf("\n enter -1 to end\n");
```

```
printf("\n enter the data\n");
scanf("%d",&num);
while(num!=-1)
{
newnode=(struct node*)malloc(sizeof(struct node));
newnode->data=num;
if(start==NULL)
{
newnode->next=newnode;
start=newnode;
}
else
{
ptr=start;
while(ptr->next!=start)
{
ptr=ptr->next;
}
ptr->next=newnode;
newnode->next=start;
}
printf("\n enter data\n");
scanf("%d",&num);
}
}
void display()
{
```

```
struct node *ptr;
ptr=start;
while(ptr->next!=start)
{
printf("%d->",ptr->data);
ptr=ptr->next;
}
printf("NULL");
}
void insert_beg()
{
struct node *ptr,*newnode;
int num;
printf("enter data");
scanf("%d",&num);
newnode=(struct node*)malloc(sizeof(struct node*));
newnode->data=num;
ptr=start;
while(ptr->next!=start)
ptr=ptr->next;
ptr->next=newnode;
newnode->next=start;
start=newnode;
}
void insert_end()
{
struct node *ptr,*newnode;
```

```
int num;
printf("\n enter data");
scanf("%d",&num);
newnode=(struct node*)malloc(sizeof(struct node));
newnode->data=num;
ptr=start;
while(ptr->next!=start)
ptr=ptr->next;
ptr->next=newnode;
newnode->next=start;
}
void delete_beg()
{
struct node *ptr;
ptr=start;
while(ptr->next!=start)
ptr=ptr->next;
ptr->next=start->next;
free(start);
start=ptr->next;
}
void delete_end()
{
struct node *ptr,*preptr;
ptr=start;
while(ptr->next!=start)
{
```

```
preptr=ptr;
ptr=ptr->next;
}
preptr->next=ptr->next;
free(ptr);
}
void insert_pos()
{
struct node *newnode,*ptr;
int pos;
int i=1;
newnode=(struct node*)malloc(sizeof(struct node));
printf("\n enter position\n");
scanf("%d",&pos);
ptr=start;
while(i<pos)
{
ptr=ptr->next;
i++;
}
printf("\n enter data\n");
scanf("%d",&newnode->data);
newnode->next=ptr->next;
ptr->next=newnode;
}
void delete_pos()
{
```

```
struct node *nextnode,*ptr;
int pos;
int i=1;
ptr=start;
printf("\n enter position\n");
scanf("%d",&pos);
while(i<pos)
{
ptr=ptr->next;
i++;
}
nextnode=ptr->next;
ptr->next=nextnode->next;
free(nextnode);
}
int main()
{
int option;
do
{
printf("\n1.craete\n");
printf("\n2.display\n");
printf("\n3.insert at beg\n");
printf("\n4.insert at end\n");
printf("\n5.insert at pos\n");
printf("\n6.delete at beg\n");
printf("\n7.delete at end\n");
```

```
printf("\n8.delete at pos\n");
printf("\n9.exit\n");
printf("\n enter your option\n");
scanf("%d",&option);
switch(option)
{
case 1:create_cll();
       break;
case 2:display();
       break;
case 3:insert_beg();
       break;
case 4:insert_end();
       break;
case 5:insert_pos();
       break;
case 6:delete_beg();
       break;
case 7:delete_end();
       break;
case 8:delete_pos();
case 9:exit(0);
       break;
default : printf("enter invalid option");
}
}
while(option!=9)
```

Question: Write a C program to create a Double linked list and perform basic operations (Insertion, Deletion, Search, Traversal)

Aim: To Write a c program to create a double linked list and perform basic operations insertion, deletion, search and traversal

Objective:

In this program we will learn how a doubly linked list works

Search – find a node in the linked list

Sort – sort the nodes of the linked list

- Inserting a new node in a doubly linked list at different positions
- Deleting a node at different positions in the created list
- Searching for an element at given position
- The traversal of the created doubly linked list

Source Code:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
struct node *prev;
};
struct node *head,*tail;
void create_dll()
{
struct node *newnode;
int num;
```

```
head=NULL;
printf("\n enter -1 to end\n");
printf("\nenter the data\n");
scanf("%d",&num);
while(num!=-1)
{
newnode=(struct node*)malloc(sizeof(struct node));
newnode->data=num;
newnode->prev=NULL;
newnode->next=NULL;
if(head==NULL)
{
head=tail=newnode;
}
else
{
tail->next=newnode;
newnode->prev=tail;
tail=newnode;
}
printf("\n enter the data\n");
scanf("%d",&num);
}
}
void display()
{
struct node *temp;
```

```
temp=head;
while(temp!=NULL)
{
printf("%d<=>",temp->data);
temp=temp->next;
}
printf("NULL");
}
void insert_beg()
{
struct node *newnode;
newnode=(struct node*)malloc(sizeof(struct node));
printf("\n enter data\n");
scanf("%d",&newnode->data);
newnode->prev=NULL;
newnode->next=head;
head=newnode;
}
void insert_end()
{
struct node *newnode;
newnode=(struct node*)malloc(sizeof(struct node));
printf("\n enter data\n");
scanf("%d",&newnode->data);
tail->next=newnode;
newnode->prev=tail;
tail=newnode;
```

```
}
void insert_pos()
{
struct node *newnode,*temp;
int i=1;
int pos;
printf("\n enter the position\n");
scanf("%d",&pos);
temp=head;
newnode=(struct node*)malloc(sizeof(struct node));
printf("\n enter data\n");
scanf("%d",&newnode->data);
while(i<pos)
{
temp=temp->next;
i++;
}
newnode->prev=temp;
newnode->next=temp->next;
temp->next=newnode;
temp->next->prev=newnode;
}
void delete_beg()
{
struct node *temp;
if(head==NULL)
printf("\n list is empty\n");
```

```
else
{
temp=head;
head=temp->next;
head->prev=NULL;
free(temp);
}
}
void delete_end()
{
struct node *temp;
if(tail==0||head==0)
printf("\n list is empty\n");
else
{
temp=tail;
tail->prev->next=NULL;
tail=tail->prev;
free(temp);
}
}
void delete_pos()
{
struct node *temp;
int pos;
int i=1;
temp=head;
```

```
printf("\n enter position\n");
scanf("%d",&pos);
while(i<pos)
{
temp=temp->next;
i++;
}
temp->prev->next=temp->next;
temp->next->prev=temp->prev;
free(temp);
}
int main()
{
int option;
do
{
printf("\n 1.create\n");
printf("\n 2.display\n");
printf("\n 3.insert at beg\n");
printf("\n 4.insert at end\n");
printf("\n 5.insert at pos\n");
printf("\n 6.delete at beg\n");
printf("\n 7.delete at end\n");
printf("\n 8.delete at pos\n");
printf("\n 9.exit\n");
printf("\n enter your option\n");
scanf("%d",&option);
```

```
switch(option)
{
case 1:create_dll();
        break;
case 2:display();
        break;
case 3:insert_beg();
        display();
        break;
case 4:insert_end();
        display();
        break;
case 5: insert_pos();
        display();
        break;
case 6: delete_beg();
        display();
        break;
case 7: delete_end();
         display();
        break;
case 8: delete_pos();
         display();
         break;
deafault: exit(0);
         break;
}
```

```
}
while(option!=9);
}
```

WEEK 6

Question: Write a C program for Stack operations using arrays

Aim: To implement stack operations using array

Objective:

The objective of the program is to conduct all operations on stack that uses

push() : inserting an element into the stack; insertion takes place from top end.

pop() : deleting an element from the top of the stack.

peek() : it is used to retrieve the element present at the top of the stack.

Is full() : checks if the stack is full(overflow).

Is empty() : checks if the stack is empty(underflow).

Source Code:

```
#include<stdio.h>
#define N 5
int stack[N];
int top=-1;
void push(int x)
{
if(top==N-1)
printf("overflow");
else
{
top++;
stack[top]=x;
}
}
```

```
void pop()
{
int item;
if(top==-1)
printf("underflow");
else
{
item=stack[top];
top--;
printf("%d is deleted",item);
}
}
void peek()
{
if(top==-1)
printf("stack is empty");
else
printf("%d",stack[top]);
}
void display()
{
int i;
for(i=top;i>=0;i--)
{
printf("%d",stack[i]);
}
}
```

```
void main()
{
int ch,x;
do
{
printf("\nenter choice\n");
printf("\n 1.push\n");
printf("\n 2.pop\n");
printf("\n 3.peek\n");
printf("\n 4.display\n");
scanf("%d",&ch);
switch(ch)
{
case 1:printf("enter element to push");
      scanf("%d",&x);
      push(x);
      break;
case 2:pop();
      break;
case 3:peek();
      break;
case 4:display();
      break;
default:printf("\n invalid choice\n");
}
}
while(ch!=0)
```

Question: Write a C program for Stack operations using linked list

Aim: To implement stack operations using linked list

Objective:

The objective of the program is to conduct all operations on stack that uses

malloc() : which allocate memory for node

struct() : which is used to create node that contains address part and data part.

push() : inserting an element into the stack;insertion takes place from top end.

pop() : deleting an element from the top of the stack.

peek() : it is used to retrieve the element present at the top of the stack.

isfull() : checks if the stack is full(overflow).

is empty() : checks if the stack is empty(underflow).

Source code:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *link;
};
struct node *top=NULL;
void push(int x)
{
struct node *newnode;
newnode=(struct node*)malloc(sizeof(struct node));
```

```
newnode->data=x;
top=newnode;
}
void display()
{
struct node *temp;
temp=top;
if(top==0)
{
printf("\n list is empty\n");
}
else{
while(temp!=0)
{
printf("%d",temp->data);
temp=temp->link;
}
}
}
void pop()
{
struct node *ptr;
ptr=top;
if(top==NULL)
printf("underflow");
else
{
```

```
printf("%d",top->data);
top=top->link;
free(ptr);
}
}
void peek()
{
if(top==0)
printf("stack is empty");
else
printf("top element is %d",top->data);
}
void main()
{
int ch,num;
do
{
printf("\n enter choice:\n");
printf("\n1.push\n");
printf("\n2.pop\n");
printf("\n3.peek\n");
printf("\n4.display\n");
printf("\n5.exit\n");
scanf("%d",&ch);
switch(ch)
{
case 1:printf("\n enter element to be inserted\n");
```

```
        scanf("%d",&num);
        push(num);
        break;
case 2:pop();
        break;
case 3:peek();
        break;
case 4:display();
         break;
case 5:exit(0);
        break;
default:printf("\n invalid choice\n");
}
}
while(ch!=5);
}
```

Question: **Write a C program To Implement Postfix Evaluation**

Aim: To implement Postfix Evaluation

Source Code:

```
#include<stdio.h>
int stack[20];
int top = -1;
void push(int x)
{
stack[++top] = x;
}
int pop()
{
return stack[top--];
}
int main()
{
char exp[20];
char *e;
int n1,n2,n3,num;
printf("Enter the expression : ");
scanf("%s",exp);
e = exp;
while(*e != '\0')
{
if(isdigit(*e))
```

```
{
num = *e - 48;
push(num);
}
else
{
n1 = pop();
n2 = pop();
switch(*e)
{
case '+':
{
n3 = n1 + n2;
break;
}
case '-':
{
n3 = n2 - n1;
break;
}
case '*':
{
n3 = n1 * n2;
break;
}
case '/':
{
```

```
n3 = n2 / n1;
break;
}
}
push(n3);
}
e++;
}
printf("\nThe result of expression %s  =  %d\n\n",exp,pop());
return 0;
}
```

WEEK 7

Question: Write a C program To implement the queue operations using arrays.

Aim: To implement the queue operations using arrays.

Objective:

A queue data structure can be implemented using one dimensional array.To implement a queue using array, create an array of fixed size and take two variables front and rear, rear is the index up to which the elements are stored in the array and front is the index of the first element of the array.Enqueue is the addition of an element to the queue (rear position). Dequeue is removal of an element from the queue (front position).

Enqueue(): add (store) an item to the queue.

dequeue(): remove (access) an item from the queue.

Source code:

```
#include<stdio.h>
#define N 5
int queue[N];
int front=-1;
int rear=-1;
void enqueue(int x)
{
if(rear==N-1)
printf("overflow");
else if((front==-1)&&(rear==-1))
{
front=rear=0;
queue[rear]=x;
}
```

```
else
{
rear++;
queue[rear]=x;
}
}
void dequeue()
{
if(front==-1&&rear==-1)
printf("underflow");
else if(front==rear)
{
printf("%d",queue[front]);
front=rear=-1;
}
else
{
printf("%d is deleted",queue[front]);
front++;
}
}
void display()
{
int i;
if(front==-1&&rear==-1)
printf("queue is empty");
else
```

```
{
for(i=front;i<rear+1;i++)
{
printf("%d",queue[i]);
}
}
}
void peek()
{
if(front==-1&&rear==-1)
printf("queue is empty");
else
printf("%d",queue[front]);
}
void main()
{
int ch,num;
do
{
printf("\n enter choice:\n");
printf("\n1.enqueue\n");
printf("\n2.dequeue\n");
printf("\n3.peek\n");
printf("\n4.display\n");
printf("\n5.exit\n");
scanf("%d",&ch);
switch(ch)
```

```
{
case 1:printf("\n enter element to be inserted\n");
      scanf("%d",&num);
      enqueue(num);
      break;
case 3:dequeue();
      break;
case 4:display();
      break;
case 5:peek();
      break;
}
}
```

Question: **Write a C program for Queue operations using linked list**

Aim: To implement the queue operations using linked list.

Objective:

A queue data structure can be implemented using linked list. In a linked queue, each node of the queue consists of two parts i.e. data part and the link part. In the linked queue, there are two pointers maintained in the memory i.e. front pointer and rear pointer. The front pointer contains the address of the starting element of the queue while the rear pointer contains the address of the last element of the queue. Insertion and deletions are performed at rear and front end respectively.

Front: Get the front item from queue.

Rear: Get the last item from queue.

Source code:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node *front=0;
struct node *rear=0;
void enqueue(int x)
{
struct node *newnode;
newnode=(struct node*)malloc(sizeof(struct node));
```

```
newnode->data=x;
newnode->next=0;
if(front==0&&rear==0)
{
front=rear=newnode;
}
else
{
rear->next=newnode;
rear=newnode;
}
}
void display()
{
struct node *temp;
if(front==0&&rear==0)
printf("queue is empty");
else
{
temp=front;
while(temp!=0)
{
printf("%d->",temp->data);
temp=temp->next;
}
}
}
```

```
void dequeue()
{
struct node *temp;
temp=front;
if(front==0&&rear==0)
printf("underflow");
else
{
printf("%d is deleted",front->data);
front=front->next;
free(temp);
}
}
void peek()
{
if(front==0&&rear==0)
printf("queue ids empty");
else
{
printf("%d",front->data);
}
}
void main()
{
int ch,num;
do
{
```

```
printf("\n enter choice:\n");
printf("\n1.enqueue\n");
printf("\n2.dequeue\n");
printf("\n3.peek\n");
printf("\n4.display\n");
printf("\n5.exit\n");
scanf("%d",&ch);
switch(ch)
{
case 1:printf("\n enter element to be inserted\n");
      scanf("%d",&num);
      enqueue(num);
      break;
case 2:dequeue();
      break;
case 3:display();
      break;
case 4:peek();
      break;
}
}
while(ch!=0);
}
```

Question : **Write a C program for Implement Circular Queue.**

Aim: To implement circular queues using Arrays

Objective:

A circular queue is the extended version of a regular queue where the last element is connected to the first element. Thus forming a circle-like structure. The circular queue solves the major limitation of the normal queue. In a normal queue, after a bit of insertion and deletion, there will be non-usable empty space. Circular Queue works by the process of circular increment i.e. when we try to increment the pointer and we reach the end of the queue, we start from the beginning of the queue.

push(): This function adds the element at the end of the queue.

pop(): This function deletes the first element of the queue.

Source code:

```
#include<stdio.h>
#define N 5
int queue[N];
int front=-1;
int rear=-1;
void enqueue(int x)
{
if(front==-1&&rear==-1)
{
front=rear=0;
queue[rear]=x;
}
else if(((rear=1)%N)==front)
```

```
printf("queue is full");
else
{
rear=(rear+1)%N;
queue[rear]=x;
}
}
void dequeue()
{
if (front==-1&&rear==-1)
printf("queue is empty");
else if(rear==front)
{
printf("%d",queue[front]);
front=rear=-1;
}
else
{
printf("%d",queue[front]);
front=(front+1)%N;
}
}
void display()
{
int i=front;
if(front==-1&&rear==-1)
{
```

```
printf("queue is empty");
}
else
{
printf("queue is :");
while(i!=rear)
{
printf("%d",queue[i]);
i=(i+1)%N;
}
printf("%d",queue[rear]);
}
}
void peek()
{
if(front==-1&&rear==-1)
printf("queue is empty");
else
{
printf("%d is peek element",queue[front]);
}
}
void main()
{
int ch,num;
do
{
```

```
printf("\n enter choice:\n");
printf("\n1.enqueue\n");
printf("\n2.dequeue\n");
printf("\n3.peek\n");
printf("\n4.display\n");
                    printf("\n5.exit\n");
scanf("%d",&ch);
switch(ch)
{
case 1:printf("\n enter element to be inserted\n");
      scanf("%d",&num);
      enqueue(num);
      break;
case 2:dequeue();
      break;
case 4:display();
      break;
case 3:peek();
      break;
default:printf("\n invalid option\n");
       break;
}
}
while(ch!=0);
}
```

WEEK 8

Question: Write a C program to Implement Binary Tree Creation and Recursive Tree Traversals(Depth first Traversals – In-Order, Pre-Order, Post-Order)

Aim: To Implement Binary Tree Creation and Recursive Tree Traversals

Objective:

To know how to use the doubly linked list to make trees by using recursive functions

To know how to print the data in tree by using various methods like: in order, pre order and post order.

Source code:

```
 #include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *left;
struct node *right;
}*root;
void postorder(struct node *temp)
{
        if(temp == NULL)
return;
postorder(temp->left);
postorder(temp->right);
printf("\n%3d\n",temp->data);
}
void inorder(struct node *node)
{
        if(node == NULL)
{
return;
}

inorder(node->left);
```

```
printf("\n%4d\n",node->data);
inorder(node->right);
}
void preorder(struct node * temp)
{
if(temp == NULL)
{
        return;
}
printf("\n%3d\n",temp->data);
preorder(temp->left);
preorder(temp->right);
}

struct node *create();
void main()
{
root = create();
printf("\nthe in order is :\n");
inorder(root);
printf("\nthe post order is :\n");
postorder(root);
printf("\n the pre order is :\n");
preorder(root);
}
struct node *create()
{
struct node *newnode;
int data;
newnode =(struct node *)malloc(sizeof(struct node));
printf("\nEnter data ,if there is no node infront press -1:\n");
scanf("%d",&data);
if(data==-1)
{
return 0;
}
newnode->data = data;
printf("enter data of the left child node :%d",data);
newnode->left = create();
printf("Enter data of the right child node:%d",data);
newnode->right = create();
return newnode;
};
```

WEEK 9

Question: Write a C program to Implement Binary Search Tree Creation Various Operations on Binary Search Tree (Insertion, Deletion, Search)

Aim: To Create and Implement Binary Search Tree Creation Various Operations on Binary Search Tree (Insertion, Deletion, and Search)

Objective:

Binary search tree is a data structure that allows for fast insertion, removal. It is a rooted tree where the nodes of the tree are ordered. If the order is ascending (low to high), the nodes of the left subtree have values that are lower than the root, and the nodes of the right subtree have values that are higher than the root. We have used create_tree() to create a tree ,preorder(),inorder(),postorder() functions to insert elements , total_nodes() to create nodes.

Source code:

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
struct node
{
int data;
struct node *left;
struct node *right;
};
struct node *tree;
void create_tree(struct node *);
struct node *insertElement(struct node *, int);
void preorderTraversal(struct node *);
void inorderTraversal(struct node *);
```

```
void postorderTraversal(struct node *);
int main()
{
int option, val;
struct node *ptr;
create_tree(tree);
clrscr();
do
{
printf("\n ******MAIN MENU******* \n");
printf("\n 1. Insert Element");
printf("\n 2. Preorder Traversal");
printf("\n 3. Inorder Traversal");
printf("\n 4. Postorder Traversal");
printf("\n 5. Exit");
printf("\n\n Enter your option : ");
scanf("%d", &option);
switch(option)
 {
case 1:
printf("\n Enter the value of the new node : ");
scanf("%d", &val);
tree = insertElement(tree, val);
break;
case 2:
printf("\n The elements of the tree are : \n");
preorderTraversal(tree);
```

```
break;
case 3:
printf("\n The elements of the tree are : \n");
inorderTraversal(tree);
break;
case 4:
printf("\n The elements of the tree are : \n");
postorderTraversal(tree);
break;
case 5:
exit(0);
break;
}
}while(option==5);
getch();
return 0;;
}
void create_tree(struct node *tree)
{
tree = NULL;
}
struct node *insertElement(struct node *tree, int val)
{
struct node *ptr, *nodeptr, *parentptr;
ptr = (struct node*)malloc(sizeof(struct node));
ptr–>data = val;
ptr–>left = NULL;
```

```
ptr–>right = NULL;
if(tree==NULL)
{
tree=ptr;
tree–>left=NULL;
tree–>right=NULL;
}
else
{
parentptr=NULL;
nodeptr=tree;
while(nodeptr!=NULL)
 {
parentptr=nodeptr;
if(val<nodeptr–>data)
nodeptr=nodeptr–>left;
else
nodeptr = nodeptr–>right;
 }
if(val<parentptr–>data)
parentptr–>left = ptr;
else
parentptr–>right = ptr;
}
return tree;
}
void preorderTraversal(struct node *tree)
```

```
{
   if(tree != NULL)
{
printf("%d\t", tree–>data);
preorderTraversal(tree–>left);
preorderTraversal(tree–>right);
}
}
void inorderTraversal(struct node *tree)
{
if(tree != NULL)
{
inorderTraversal(tree->left);
printf("%d\t", tree->data);
inorderTraversal(tree->right);
}
}
void postorderTraversal(struct node *tree)
{
if(tree != NULL)
{
postorderTraversal(tree->left);
postorderTraversal(tree->right);
printf("%d\t", tree->data);
}
```

WEEK 10

Question: Write a C program to Implement Graph Creation using adjacency list

Aim: Program to create a graph of an adjacency list.

Objective:

Main ()-Which is the first function of every C program that is responsible for starting the execution and termination of the program.

Adjacency list: An adjacency list is a collection of unordered lists used to represent a finite graph. Each unordered list within an adjacency list describes the set of neighbours of a particular vertex in the graph.

Source code:

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
struct node
{
char vertex;
struct node *next;
};
struct node *gnode;
void displayGraph(struct node *adj[], int no_of_nodes);
void createGraph(struct node *adj[], int no_of_nodes);
int main()
{
    struct node *Adj[10];
int i, no_of_nodes;
printf("\n Enter the number of nodes in G: ");
```

```
scanf("%d", &no_of_nodes);
    for(i = 0; i < no_of_nodes; i++)
 Adj[i] = NULL;
createGraph(Adj, no_of_nodes);
printf("\n The graph is: ");
displayGraph(Adj, no_of_nodes);
deleteGraph(Adj, no_of_nodes);
getch();
    return 0;
}
void createGraph(struct node *Adj[], int no_of_nodes)
{
struct node *new_node, *last;
int i, j, n, val;
    for(i = 0; i < no_of_nodes; i++)
{
 last = NULL;
 printf("\n Enter the number of neighbours of %d: ", i);
 scanf("%d", &n);
 for(j = 1; j <= n; j++)
 {
 printf("\n Enter the neighbour %d of %d: ", j, i);
 scanf("%d", &val);
 new_node = (struct node *) malloc(sizeof(struct node));
 new_node->vertex = val;
 new_node->next = NULL;
 if (Adj[i] == NULL)
```

```
 Adj[i] = new_node;
 else
 last ->next = new_node;
 last=new_node;
 }
}
}
void displayGraph (struct node *Adj[], int no_of_nodes)
{
struct node *ptr;
int i;
    for(i = 0; i < no_of_nodes; i++)
{
 ptr = Adj[i];
 printf("\n The neighbours of node %d are:", i);
 while(ptr != NULL)
 {
 printf("\t%d", ptr->vertex);
 ptr = ptr->next;
 }
}
}
```

Question: Write a C program to Implement Graph Traversal Breadth First Search

Aim: Program to implement Graph Traversal Breadth First Search

Objective:

Breadth-first search (BFS) is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. Extra memory, usually a queue, is needed to keep track of the child nodes that were encountered but not yet explored.

This non-recursive implementation is similar to the non-recursive implementation of depth-first search, but differs from it in two ways:

It uses a queue (First in First Out) instead of a stack

It checks whether a vertex has been explored before enqueueing the vertex rather than delaying this check until the vertex is dequeued from the queue.

Source code:

```
#include <stdio.h>
#define MAX 10
void breadth_first_search(int adj[][MAX],int visited[],int start)
{
    int queue[MAX],rear=-1,front=-1, i;
queue[++rear] = start;
visited[start] = 1;
while(rear != front)
{
 start = queue[++front];
 if(start == 4)
 printf("5\t");
 else
```

```
 printf("%c \t",start + 65);
        for(i = 0; i < MAX; i++)
 {
            if(adj[start][i] == 1 && visited[i] == 0)
 {
 queue[++rear] = i;
visited[i] = 1;
 }
 }
}
}
int main()
{

    int visited[MAX] = {0};
    int adj[MAX][MAX], i, j;
printf("\n Enter the adjacency matrix: ");
    for(i = 0; i < MAX; i++)
        for(j = 0; j < MAX; j++)
 scanf("%d", &adj[i][j]);
breadth_first_search(adj,visited,0);
    return 0;
}
```

WEEK 11

Question: Minimum Spanning Tree Using Prim's Algorithm

Aim: To find the Minimum Spanning tree by using Prim's Algorithm.

Objective: Spanning tree - A spanning tree is the sub graph of an undirected connected graph.

Minimum Spanning tree - Minimum spanning tree can be defined as the spanning tree in which the sum of the weights of the edge is minimum. The weight of the spanning tree is the sum of the weights given to the edges of the spanning tree.

Main()-Which is the first function of every C program that is responsible for starting the execution and termination of the program.

Prim's()-Prim's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which Form a tree that

It has the minimum sum of weights among all the trees that can be formed from the graph includes every vertex.

Source code:

```
#include<stdio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
void main()
{
        printf("\nEnter the number of nodes:");
        scanf("%d",&n);
        printf("\nEnter the adjacency matrix:\n");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
```

```
{
        scanf("%d",&cost[i][j]);
        if(cost[i][j]==0)
                cost[i][j]=999;
}

visited[1]=1;
printf("\n");
while(ne < n)
{

        for(i=1,min=999;i<=n;i++)

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

        if(cost[i][j]< min)

        if(visited[i]!=0)

        {
                min=cost[i][j];
                a=u=i;
                b=v=j;
        }

        if(visited[u]==0 || visited[v]==0)
        {
```

```
                printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
                mincost+=min;
                visited[b]=1;
            }
            cost[a][b]=cost[b][a]=999;
        }
        printf("\n Minimun cost=%d",mincost);
}
```

Question: Write a C program to Minimum Spanning Tree using Kruskal's Algorithm

Aim: To find the Minimum Spanning tree by using Kruskal's Algorithm

Objective: Spanning tree - A spanning tree is the sub graph of an undirected connected graph.

Minimum spanning tree - Minimum spanning tree can be defined as the spanning tree in which the sum of the weights of the edge is minimum. The weight of the spanning tree is the sum of the weights given to the edges of the spanning tre

Main ()-Which is the first function of every C program that is responsible for starting the execution and termination of the program.

Kruskal ()- to find a subset of the edges that forms a tree and includes every vertex where the total weight of all of the edges is a minimum. Kruskal's algorithm is most suitable for sparse graphs.

Source code:

```
#include<stdio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
{

        printf("\nEnter the no. of vertices:");
        scanf("%d",&n);
        printf("\nEnter the cost adjacency matrix:\n");
```

```
for(i=1;i<=n;i++)
{
        for(j=1;j<=n;j++)
        {
                scanf("%d",&cost[i][j]);
                if(cost[i][j]==0)
                        cost[i][j]=999;
        }
}
printf("The edges of Minimum Cost Spanning Tree are\n");
while(ne < n)
{
        for(i=1,min=999;i<=n;i++)
        {
                for(j=1;j <= n;j++)
                {
                        if(cost[i][j] < min)
                        {
                                min=cost[i][j];
                                a=u=i;
                                b=v=j;
                        }
                }
        }
        u=find(u);
        v=find(v);
        if(uni(u,v))
```

```
                {
                        printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
                        mincost +=min;
                }
                cost[a][b]=cost[b][a]=999;
        }
        printf("\n\tMinimum cost = %d\n",mincost);
        getch();
}
int find(int i)
{
        while(parent[i])
        i=parent[i];
        return i;
}
int uni(int i,int j)
{
        if(i!=j)
        {
                parent[j]=i;
                return 1;
        }
        return 0;
}
```

WEEK 12

Question: **Write a C program to Implement Heap Sort**

Aim: To sort the given elements using heap sort technique

Objective:

Main()-Which is the first function of every C program that is responsible for starting the execution and termination of the program.

Heapify()-It is a method rearranges the elements of an array where the left and right sub-tree of ith element obeys the heap property.

Heapsort()-Heap sort is comparison-based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the minimum element and place the minimum element at the b

Source code:

```
#include<stdio.h>

int temp;

void swap_largest(int arr[], int n, int i)

{

int largest = i;

int left = 2*i + 1;

int right = 2*i + 2;

if (left < n && arr[left] > arr[largest])

largest = left;

if (right < n && arr[right] > arr[largest])

largest = right;

if (largest != i)

{

temp = arr[i];
```

```
arr[i] = arr[largest];
arr[largest] = temp;
swap_largest(arr, n, largest);
}
}
void heap(int arr[], int n)
{
for (int i = n / 2 - 1; i >= 0; i--)
swap_largest(arr, n, i);
for (int i = n - 1; i >= 0; i--)
{
temp = arr[0];
arr[0] = arr[i];
arr[i] = temp;
swap_largest(arr, i, 0);
}
}
int print(int arr[], int n)
{
for(int i = 0; i < n; i++)
{
printf("%d ", arr[i]);
}
}
int main()
{
int n, i;
```

```
scanf("%d", &n);
int arr[n];
for(i = 0; i < n; i++)
{
scanf("%d", &arr[i]);
}
heap(arr, n);
print(arr, n);
}
```