DATA STRUCTURES LAB – LIST OF PROGRAMS

(Common to CSE, AI&ML, CS, DS)

1. Write C program that use both recursive and non-recursive functions to perform Linear search for
a key value in a given list.
2. Write C program that use both recursive and non-recursive functions to perform Binary search for
a key value in a given list.
3. Write a C program that uses functions to perform the following operations on singly linked list.:

i) Creation ii) Insertion iii) Deletion iv) Traversal
4. Write a C program that uses functions to perform the following operations on doubly linked list.:
i) Creation ii) Insertion iii) Deletion iv) Traversal
5. Write a C program that uses functions to perform the following operations on circular linked list.:
i) Creation ii) Insertion iii) Deletion iv) Traversal
6. Write a C program that implement stack (its operations) using
i) Arrays ii) Pointers
7. Write a C program that implement Queue (its operations) using
i) Arrays ii) Pointers
8. Write a C program that Uses Stack Operations to Convert Infix expression into Postfix expression
9. Write a C program that Uses Stack Operations to Evaluate the Postfix expression
10. Write a C program that uses functions to perform the following
i) creating a binary tree of integers ii) Traversing the above binary tree in preorder, inorder and post order
11. Write a C program that uses functions to perform the following operations on Binary search Tree.:
i) Creation ii) Insertion iii) Deletion
12. Write a program that implements the following sorting methods to sort a given list of integers in
ascending order
i) Quick sort ii) Merge sort
13. Write a program to implement the graph traversal methods.

1. Recursive and Non Recursive implementation of Linear search

Aim:- To implement linear search for a key value in a given list by using recursive and non-recursive functions.

Algorithm:-

Step-1: Start

Step-2: Declare any array ‘a’, n,x,ch.

Step-3: Read a value into ‘ch’

Step-4: If ch=1 or ch=2 then

4.1. Read ‘n’ number of elements into array ‘a’

4.2. Read a value into ‘x’ to search the array ‘a’

4.3. If ch=1 call function lsr (a,n,x) // Recursion

4.4. If ch=2 call function lsr (a,n,x) // Non-Recursion

Step-5: Else

5.1: Print “ wrong choice ! Try Again “

Step-6: End If

Step-7: Stop

Recursion: -

Algorithm lsr(int a [ ], int n, int x)

Step-1: If (a [n] = = x) then

Print (“ successful search and Return position “)

Step-2: Else If (n= =0) then

Print (“ unsuccessful search “)

Step-3: Else

Call lsr (a,n-1,x) ;

Step-4: End If

Step-5: End If

Step-6: Stop

Non-Recursion:-

Algorithm lsnr(int a [ ], int x)

Step-1: Declare I and initialize f=0

Step-2: For i =1 to n do

If (a [i] = = x) then

Print “ Successful search and return position “

Set f = 1 and break

Else if (f= = 0) then

Print " unsuccessful search "

End If

Step-3: End for

Step-4: Stop

Program 1. Write C program that use both recursive and non-recursive functions to perform Linear search for a key value in a given list.

```
#include <stdio.h>
#include <conio.h>
void lsr(int a[],int n,int x);
void lsnr(int a[],int n,int x);
void read(int a[],int n);
void display(int a[],int n);

int main()
{
   int a[20],n,x,ch;
   clrscr();
   printf("=====================================================");
   printf("\n\t\t\tMENU");
   printf("\n=====================================================");
   printf("\n1.Linary Search using Recursion method");
   printf("\n2.Linary Search using Non-Recursion method");
   printf("\n\nEnter your Choice:");
   scanf("%d",&ch);
   if(ch==1||ch==2)
    {
    printf("Enter the number of elements :");
    scanf("%d",&n);
    read(a,n);
    printf("\nElements present in the array are:\n\n");
    display(a,n);
    printf("\n\nEnter the element you want to search:\n\n");
    scanf("%d",&x);
    switch(ch)
    {
     case 1:printf("\n**Recursion method**\n");
          lsr(a,n,x);
          getch();
          break;
     case 2:printf("\n**Non-Recursion method**\n");
          lsnr(a,n,x);
          getch();
          break;
    }
    }
    else
    printf("Wrong Choice! Try Again");
 getch();
 return 0;
}

void lsnr(int a[],int n,int x)
```

```
{
    int i,f=0;
    for(i=1;i<=n;i++)
     if(a[i]==x)
     {
      printf("\nThe element %d is present at position %d in the array\n",x,i);
      f=1;
      break;
     }
    if(f==0)
     printf("\nThe element is %d is not present in the array\n",x);
}

void lsr(int a[],int n,int x)
{
    if(a[n]==x)
     printf("\nThe element %d is present at position %d in the array\n",x,n);
    else
    {
     if((n==0))
         printf("The element %d is not found in the array",x);
     else
         lsr(a,n-1,x);
    }
}

void read(int a[],int n)
{
    int i;
    printf("\nEnter the elements:\n");
    for(i=1;i<=n;i++)
     scanf("%d",&a[i]);
}

void display(int a[],int n)
{
    int i;
    for(i=1;i<=n;i++)
     printf("%d\t",a[i]);
}
```

2.Recursive and Non-Recursive implementation of Binary Search

Aim:- To implement Binary Search for a key value in a given list by using recursive and non-recursive functions.

Algorithm:-

Step-1: start

Step-2: Declare an array 'a', n, x, ch and Pos.

Step-3: Read a value into 'ch' as choice.

Step-4: If ch = 1 or ch = 2 then

4.1. Read 'n' number of elements into array 'a'

4.2. Display elements in the given array 'a'

4.3. If ch = 1 then set the 'Pos' value

By calling function bsr (a, 1,n, x) // recursion

4.4. if Pos = -1 then

Print " Element is not found "

4.5. Else

Print " Element is found : Return Pos ".

4.5. End If

4.6. If ch = 2 then call function bsnr (a, n, x) // Non-Recursion

Step-5: Else

5.1. Print " Wrong choice ! Try Again "

Step-6: End If

Step-7: Stop.

Recursion: -
Algorithm bsr (int a [ ], int low, int high, int x)

Step-1: Declare 'mid'

Step-2: If (low ≤ high) then

Mid = (low+high)/2

Check if (x=a [mid]) then

Return bsr (a, low,mid-1, x)

Else

return bsr (a, mid+1, high, x) ;

Step-3: Else

Return -1

Step-4: End If

Step-5: Stop

Non-Recursion: -

Algorithm bsnr (innt a [ ], int n, int x)

Step-1: Declare low,high,mid and initialize found =0

Step-2: Set low = 1 and high = n

Step-3: while (low < = high) do

3.1. mid = (low+high) /2

3.2. check if (x = a [mid]) then

Print " successful Search " found = 1

3.3. Else if (x, a [mid]) then high = mid-1

3.4. Else low =mid+1

3.5. End If

Step-4: End while

Step-5: Check if (found = 0) then

Print " unsuccessful Search "

Step-6: End If

Step-7: Stop

Program 2. Write C program that use both recursive and non-recursive functions to perform Binary search for a key value in a given list.

```
#include <stdio.h>
/* Non-Recursive function*/
void bsnr(int a[],int n,int x)
{
  int low,high,mid,found=0;
  low=1;
  high=n;
  while(low<=high)
  {
    mid=(low+high)/2;
    if(x==a[mid])
    {
   printf("\nThe element %d is present at position %d in list\n",x,mid);
            found=1;
            break;
    }
    else
      if(x<a[mid])
              high=mid-1;
```

```
        else
                low=mid+1;
   }
   if(found==0)
   printf("\nThe element %d is not present in the list\n",x);
}

/* Recursive function*/
int bsr(int a[],int low,int high,int x)
{
  int mid;
  if (low<=high)
  {
    mid=(low+high)/2;
    if (x==a[mid])
      return mid;
    else if (x<a[mid])
      return bsr(a,low,mid-1,x);
    else
      return bsr(a,mid+1,high,x);
   }
   return -1;
}

void read(int a[],int n)
{
   int i;
   printf("\nEnter the elements:\n");
   for(i=1;i<=n;i++)
       scanf("%d",&a[i]);
}

void display(int a[],int n)
{
   int i;
   for(i=1;i<=n;i++)
       printf("%d\t",a[i]);
}

/*main function*/
void main()
{
   int a[20],n,x;
   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==1||ch==2)
   {
     printf("\nEnter the number of elements : ");
```

```
    scanf("%d",&n);
    read(a,n);
    printf("\nElements present in the list are:\n\n");
    display(a,n);
    printf("\n\nEnter the element you want to search:\n\n");
    scanf("%d",&x);
  switch(ch)
  {
    case 1:printf("\nRecursive method:\n");
       pos=bsr(a,1,n,x);
       if(pos==-1)
              printf("Element is not found");
       else
              printf("Element is found at %d position",pos);
       break;

    case 2:printf("\nNon-Recursive method:\n");
       bsnr(a,n,x);
       break;
    }
  }
  else
   printf("Wrong Choice! Try Again");
 getch();
}
```

3. Oeartions on Single Linked List

Aim:- To implement Creation, Insertion, Deletion and Traversal Operations on Single linked list using functions.

Algorithm:-

Step-1: Start

Step-2: Create a node structure for single linked list.

Step-3: Initialize Choice = 0

Step-4: While (choice I = 8) do

4.1. Display Menu format and Read ‘choice’.

4.2. If choice = 1 call function insertfront ( )

4.3. If choice = 2 call function insertlast ( )

4.4 If choice = 3 call function randominsert ( )

4.5. If choice = 4 call function deletefront ( )

4.6. If choice = 5 call function deletelast ( )

4.7. If choice = 6 call function randomdelete ( )

4.8. If choice = 7 call function display ( )

4.9. If choice = 8 call functon exit ( )

Step-5: Else

Print “ Invalid choice “

Step-6: End If

Step-7: Stop.

Algorithm insertfront ( )

Step-1: Declare a node * ptr and item

Step-2: If (ptr = NULL) then

Print " overflow "

Step-3: Else

Read a value into 'item'

Set ptr data = item

Set ptr next = head

Set head = ptr

Print " Node inserted "

Step-4: End If

Step-5: Stop

Algorithm insertlast( )

Step-1: Declare nodes * ptr, * temp and item

Step-2: If (ptr = NULL) then

Print " overflow '

Step-3: Else

Read a value into 'item'

Set ptr data = item

If (head = NULL) then

Set ptr next = NULL

Set head = ptr

Print " Node inserted "

Else

Set temp = head

While (temp next ! = NULL) do

temp = temp next

End while

Set temp next = ptr

Set ptr next = NULL

Print (“ Node inserted “)

Step-4: End If

Step-5: Stop

Algorithm randominsert ( )

Step-1: Declarei, loc , item as integers

Step-2: Declare nodes *ptr , * temp.

Step-3: If (ptr = NULL) then

Print “ overflow “

Step-4: Else

Read a value into ‘item’

Set ptr data = item

Read position value into ‘loc’

Set temp = head

For (i=1 to loc-1) do

temp=temp next

if (temp=NULL) then

Print “ can’t Insert “

End for

Set ptr next= temp next

Set temp next = ptr

Print “ Node inserted “

Step-5: End If

Step-6: Stop

Algorithm deletefront ( )

Step-1: Declare node *ptr

Step-2: If (head= NULL) then

Print “List is empty”

Step-3: Else

Set ptr = head, head = ptr next

Free (ptr) and Print “ Node deleted “

Step-4: End If

Step-5: Stop

Algorithm deletelast ( )

Step-1: Declare nodes *ptr, *ptr_1

Step-2:if (head = NULL) then

Print "List is Empty"

Step-3: Else If (head next = NULL) then

Set head = NULL

Free (head) and Print " Node deleted "

Step-4: Else

Set ptr=head

While (ptr next ! = NULL) do

Set ptr_1 = ptr

Set ptr = ptr next

End while

Set ptr_1 next = NULL

Free (ptr) and Print " Node deleted "

Step-5: End If

Step-6: Stop

Algorithm randomdelete ( )

Step-1: Declare nodes *ptr , ptr_1

Step-2: Declare two integers 'doc' and 'i'

Step-3: Read position value into 'loc'

Step-4: Set ptr = head

Step-5: for (i= 1 to loc-1) do

Set ptr_1 = ptr and ptr = ptr next

If (ptr= NULL) then

Print " can't delete and return

End If

Step-6: End for

Step-7: Set ptr_1 next = ptr next

Step-8: free (ptr) and Print "Node deleted "

Step-9: Step

Algorithm display ( )

Step-1: Declare node *ptr

Step-2: Set ptr_1 = head

Step-3: If (ptr = NULL) then

Print “Nothing to print”

Step-4: Else

Print “The List is :”

While (ptr !=NULL) do

Print : ptr data

Ptr=ptr next

End while

Step-5: End If

Step-6: Stop

Program 3. Write a C program that uses functions to perform the following operations on singly linked list: i) Creation ii) Insertion iii) Deletion iv) Traversal

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

void insertfront();
void insertlast();
void randominsert();
void deletefront();
void deletelast();
void randomdelete();
void display();
void main()
{
   int choice=0;
   clrscr();
   while(choice!=8)
   {
    printf("\n\n*********Main Menu*********\n");
    printf("\nChoose one option from the following list ...\n");
    printf("\n===============================================\n");
    printf("\n1.Insert at Front\n2.Insert at Last\n3.Insert at any random location\n4.Delete at Front\n5.Delete at
Last\n6.Delete node after specified location\n7.Display\n8.Exit\n");
    printf("\nEnter your choice?\n");
    scanf("\n%d",&choice);
    switch(choice)
    {
       case 1:insertfront();
              break;
       case 2:insertlast();
```

```
                break;
        case 3:randominsert();
                break;
        case 4:deletefront();
                break;
        case 5:deletelast();
                break;
        case 6:randomdelete();
                break;
        case 7:display();
                break;
        case 8:exit(0);
                break;
        default:printf("Please enter valid choice..");
     }
    }
getch();
}
void insertfront()
{
    struct node *ptr;
    int item;
    ptr = (struct node *) malloc(sizeof(struct node *));
    if(ptr == NULL)
    {
     printf("\nOVERFLOW");
    }
    else
    {
     printf("\nEnter value:");
     scanf("%d",&item);
     ptr->data = item;
     ptr->next = head;
     head = ptr;
     printf("\nNode inserted");
    }

}
void insertlast()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node*)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
     printf("\nOVERFLOW");
    }
    else
    {
     printf("\nEnter value:");
     scanf("%d",&item);
     ptr->data = item;
     if(head == NULL)
     {
        ptr -> next = NULL;
        head = ptr;
```

```
        printf("\nNode inserted");
    }
    else
    {
        temp = head;
        while (temp -> next != NULL)
        {
                temp = temp -> next;
        }
        temp->next = ptr;
        ptr->next = NULL;
        printf("\nNode inserted");

    }
   }
}
void randominsert()
{
   int i,loc,item;
   struct node *ptr, *temp;
   ptr = (struct node *) malloc (sizeof(struct node));
   if(ptr == NULL)
   {
    printf("\nOVERFLOW");
   }
   else
   {
    printf("\nEnter value:");
    scanf("%d",&item);
    ptr->data = item;
    printf("\nEnter position:");
    scanf("\n%d",&loc);
    temp=head;
    for(i=1;i<loc-1;i++)
    {
        temp = temp->next;
        if(temp == NULL)
        {
                printf("\ncan't insert\n");
                return;
        }

    }
    ptr ->next = temp ->next;
    temp ->next = ptr;
    printf("\nNode inserted");
   }
}
void deletefront()
{
   struct node *ptr;
   if(head == NULL)
   {
    printf("\nList is empty\n");
   }
   else
```

```
    {
     ptr = head;
     head = ptr->next;
     free(ptr);
     printf("\nNode deleted from the Front of the list ...\n");
    }
}
void deletelast()
{
    struct node *ptr,*ptr1;
    if(head == NULL)
    {
     printf("\nList is empty");
    }
    else if(head -> next == NULL)
    {
     head = NULL;
     free(head);
     printf("\nOnly node of the list deleted ...\n");
    }

    else
    {
     ptr = head;
     while(ptr->next != NULL)
     {
        ptr1 = ptr;
        ptr = ptr ->next;
     }
     ptr1->next = NULL;
     free(ptr);
     printf("\n Node Deleted from the last ...\n");
    }
}
void randomdelete()
{
    struct node *ptr,*ptr1;
    int loc,i;
    printf("\n Enter position \n");
    scanf("%d",&loc);
    ptr=head;
    for(i=1;i<loc;i++)
    {
     ptr1 = ptr;
     ptr = ptr->next;

     if(ptr == NULL)
     {
        printf("\nCan't delete");
        return;
     }
    }
    ptr1 ->next = ptr ->next;
    free(ptr);
    printf("\nDeleted node %d ",loc);
}
```

```
void display()
{
   struct node *ptr;
   ptr = head;
   if(ptr == NULL)
   {
    printf("Nothing to print");
   }
   else
   {
    printf("\n The List is:\n");
    while (ptr!=NULL)
    {
       printf("%3d",ptr->data);
       ptr = ptr -> next;
    }
   }
}
```

4. Operations on Double Linked List

Aim:- To implement Creation, Insertion, Deletion and Traversal operations on double Linked list using functions.

Algorithm:-

Steps :

1. Start
2. Create a node structure for double linked list
3. Declare the integer variable "choice'
4. While (1) do

   4.1 Display Menu format and read a value into "choice'.

   4.2 If choice = I call function traverse()

   4.3 If choice = 2 call function insert At front()

   4.4 If choice = 3 Call function insert At Ends()

   4.5 If choice = 4 Call function insert At Position()

   4.6 If choice = 5 Call function delete First()

   4.7 If Choice = 6 Call function delete End()

   4.8 If Choice = 7 Call function delete position()

   4.9 If choice = 8 Call function exit ()

4.10. Print "Invalid choice" as default.

5. End while

6. stop.

Algorithm traverse()

Step1: Declare a node * temp

Step2: If (Start =Null) their

print " List is empty"and return

Step3 : Else

set temp= start

print " The elements in the list are:"

while (temp! = NULL) do

print: temp - >info

set temp-> next

end while

step 4: End If

step5: stop.

Algorithm insertAtFront ()

Step1: Declare ‘data’ as integer and a node *temp

Step2: Allocate memory for ‘temp’

Step3: Read a value into data to insert

Step4: set temp-> info = date

steps: set temp-> prev = NULL

step6 : set temp-> next = Start

step7 : set start =temp

step8 : stop

Algorithm insertatEnd()

Step1: Declare data as integer and two nodes * temp, * trav

Step2: Allocate memory for temp.

Step3: set temp-> prev = NULL and temp-> next = NULL

Step4: Read a value into data to insert.

Step5: set temp-> into = date

step6 : set temp-> next =NULL

Step7: set trav = start

Step8: If (start =NULL) then

set start = temp

step9: Else

while (trav -> next != NULL) do

set trav = trav -> next

end while

set temp -> prev = trav

set trav -> next = temp

Step10: End If

Step 11 : stop

Algorithm insertAtPosition()

Step1: Declare two integers data, pos and initialize i = 1

Step2: Declare two nodes *temp and *new node

step3: Allocate memory for new node.

Step4 : set new node-> next = NULL and new node-> Prev = NULL

Step5 : Read position value into 'pos’

Step6 : If (start = NUIL) then

set start = new node

set new node -> prev = NULL

set new node -> next = NULL

step7: Else if (pos = 1) Then

call function insert At Front ()

Step8: Else

Read a value into "data ' to insert

set new node -> info = data

set temp = start

while (i < pos-1) do

temp = temp-> next

i=i+1

End while

set new node -› next = temp -> next

set new node -> Prev =temp

set temp->next = new node

set temp-> next = Prev = new node

Step9: End If

Step10: stop.

Algorithm deleteFirst()

Step1: Declare a node * temp

Step2: If (start = NULL) then

Print "List is empty"

step3: Else

temp=start

start = start-> next

if (start! = NULL) then

start -> Print "List is empty"

free (temp)

End If

Step4 : End If

Step5: stop

Algorithm deleteEnd ()

Step1: Declare a node *temp

Step2: If (Start = NULL) then

Print "List is empty"

stap3: Else: temp = Start

while (temp -> next ! =NULL) do

temp=temp -> next

if (start -> next = NULL) then

Start = NULL

End If

Step4: free (temp)

Step5: End If

Step6: stop.

Algorithm deletePosition()

Step1: Declare two integers pos , i and initialize i =1.

Step2: Declare two nodes * temp, * position

Step3: set temp = start

Step4: if (start =NULL) then

Print "List is empty"

Step5: Else

Read value into ‘pos’

If (pos=1) then

delete first node and return

free (position)

End If

while (i< pos-1) do

temp = temp ->next

i=i+1

End while

Set position =temp ->next

If (position -> next !=NULL)

position-> next-> prev -> next

End If

set temp -> next = postion -> next

free (position)

step6: End If

step7: stop

Program 4.Write a C program that uses functions to perform the following operations on doubly linked list: i) Creation ii) Insertion iii) Deletion iv) Traversal

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int info;
   struct node *prev, *next;
};
struct node* start = NULL;
void traverse()
{
   struct node* temp;
   if (start == NULL)
    {
    printf("\nList is empty\n");
    return;
   }
   temp = start;
   printf("\n The elements in the list are:\n");
   while (temp != NULL)
   {
    printf("%d\t",temp->info);
    temp = temp->next;
   }
}
void insertAtFront()
{
```

```
    int data;
    struct node* temp;
    temp = (struct node*)malloc(sizeof(struct node));
    printf("\nEnter number to be inserted: ");
    scanf("%d", &data);
    temp->info = data;
    temp->prev = NULL;
    temp->next = start;
    start = temp;
}
void insertAtEnd()
{
    int data;
    struct node *temp, *trav;
    temp = (struct node*)malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->next = NULL;
    printf("\nEnter number to be inserted: ");
    scanf("%d", &data);
    temp->info = data;
    temp->next = NULL;
    trav = start;
    if (start == NULL)
     start = temp;
    else
     {
     while (trav->next != NULL)
         trav = trav->next;
     temp->prev = trav;
     trav->next = temp;
     }
}
void insertAtPosition()
{
    int data, pos, i = 1;
    struct node *temp, *newnode;
    newnode = malloc(sizeof(struct node));
    newnode->next = NULL;
    newnode->prev = NULL;
    printf("\nEnter position : ");
    scanf("%d", &pos);
    if (start == NULL)
     {
     start = newnode;
     newnode->prev = NULL;
     newnode->next = NULL;
    }
    else if (pos == 1)
      insertAtFront();
    else
    {
    printf("\nEnter number to be inserted: ");
    scanf("%d", &data);
    newnode->info = data;
    temp = start;
     while (i < pos - 1)
```

```
     {
       temp = temp->next;
       i++;
     }
     newnode->next = temp->next;
     newnode->prev = temp;
     temp->next = newnode;
     temp->next->prev = newnode;
    }
}
void deleteFirst()
{
    struct node* temp;
    if (start == NULL)
     printf("\nList is empty\n");
    else {
     temp = start;
     start = start->next;
     if (start != NULL)
         start->prev = NULL;
     free(temp);
    }
}
void deleteEnd()
{
    struct node* temp;
    if (start == NULL)
     printf("\nList is empty\n");
    temp = start;
    while (temp->next != NULL)
     temp = temp->next;
    if (start->next == NULL)
     start = NULL;
    else
     {
     temp->prev->next = NULL;
     free(temp);
    }
}
void deletePosition()
{
    int pos, i = 1;
    struct node *temp, *position;
    temp = start;
    if (start == NULL)
     printf("\nList is empty\n");
    else
     {
     printf("\nEnter position : ");
     scanf("%d", &pos);
     if (pos == 1)
     {
         deleteFirst();
         if (start != NULL)
                 start->prev = NULL;
         free(position);
```

```
        return;
    }
    while (i < pos - 1)
     {
        temp = temp->next;
        i++;
    }
    position = temp->next;
    if (position->next != NULL)
        position->next->prev = temp;
    temp->next = position->next;
    free(position);
   }
}
void main()
{
   int choice;
   clrscr();
   while (1)
    {

    printf("\n\t 1.To see the list\n");
    printf("\t 2.For insertion at starting\n");
    printf("\t 3.For insertion at end\n");
    printf("\t 4.For insertion at any position\n");
    printf("\t 5.For deletion of first element\n");
    printf("\t 6.For deletion of last element\n");
    printf("\t 7.For deletion of element at any position\n");
    printf("\t 8.To exit\n");
    printf("\n Enter Choice :\n");
    scanf("%d", &choice);

    switch (choice)
     {
    case 1: traverse();
            break;
    case 2:  insertAtFront();
             break;
    case 3:  insertAtEnd();
             break;
    case 4:  insertAtPosition();
             break;
    case 5:  deleteFirst();
             break;
    case 6:  deleteEnd();
             break;
    case 7:  deletePosition();
             break;
    case 8:  exit(1);
             break;
    default: printf("Incorrect Choice. Try Again \n");
              continue;
    }
   }
}
```

5. operations on circular linked list

Aim: - To implement Creation, Insertion, Deletion and Traversal operations on, circular single linked List using functions.

Algorithm:-

Step1: Create a structure of a node for circular Single linked list

Step2: Read an integer variable choice and initialize Choice =o

Step3: while (choice! =7) do

3.1: Display menu format and read a value Into choice.

3.2. if choice = 1 then Call function beginsert()

3:3. If choice = 2 then Call function lastinsert()

3.4 if choice= 3 their call function begindelete()

3.5. If choice = 4 then call function lastdelete()

3.6 if choice = 5 then call function search ()

3.7 if Choice = 6 then call function display ()

3.8. If choice=7 then call function exit (o).

3.9. Print 'Invalid choice" as default

Step4: End wille

Step5: stop

Algorithm beginsert()

Step1: Declare two nodes * Ptr , * temp

Step2: Declare an integer variable ‘item’

Step3: Allocate memory for the node 'ptr'

Step4: If (Ptr = NULL) then

Print " Overflow"

Steps5: Else

Read a value into 'item’ to insert

set ptr -> data = item

If (head= NULL) then

head= ptr, ptr-> next = head

Else

temp= head

while (temp->next ! =head)

temp=temp-> next

End while

Ptr->next= head, temp next = ptr

set head = ptr

End If

Print "Node inserted"

Step 6 : End If

Step 7: Stop

Algorithm lastinsert()

Step1: Declare nodes *ptr and *temp

Step2: Declare an integer variable " item”

Step3: Allocate memory for the node ptr

Step4: If (ptr = NULL) then

print "overflow"

steps5: Else

Read a value into item to Insert

set ptr -> data = item

if (head = NULL) then

set head=ptr, ptr -> next =head

Else

temp=head

while (temp->next! =head) then

set temp=temp-> next

End while

Set temp -> next = ptr, per -> next = head

End If

Pint "Node inserted"

Step6: End If

step7: stop.

Algorithm begindelete()

Step1: Declare node *pts

Step2: if (head= NULL) then

print " Underflow"

step3: Else if (head ->next=head) Then

set head= NULL , free (head)

print "Node deleted "

stepy: Else

set Pts=head

while (Ptr -> next ! = head)

pts = Ptr -> next

End while

set ptr - > next = head->next

free (head)

head =ptr->next

Print " Node deleted"

Step5: End If

Step6: Stop

Algorithm lastdelete ()

step1 : Declare two nodes * Ptr and *preptr

Step2: IF (head= Null) than

Print " Underflows"

Step3: Else if (head -> nest = head) then

set head = NULL, free (head)

print "Node” deleted"

step4: Else

set ptr = head

while (Pte -> next ! = head) do

set Preptr = ptr , ptr=Pty-> next

end while

Preptr -> next = ptr -> next

free (ptr)

Print "Made deleted "

Step5: End If

step6 : stop

Algorithm Search()

Step 1: Declare a node *ptr

Steps2: Declare au integer "item" and initialize i=0 and flag= 1

step3: set Ptr = head

Step4: If (ptr = NULL) then

Print "List is empty"

steps: else

Read a value into 'item' to search

if (head->data = item) then

Print "Successful search" and return location

set flag=o

Else

while (Ptr -> next =head) do

if (Ptr -> date = item) then

Print "Succesfull search" and return location

Set flag=o, break

Else

set flag =1

End If

set i = i +1, Ptr = ptr → next

End while

if (flag !=0) then

Print "Unsuccessful search"

End If

Step6: End If

step7 : stop.

Algorithm display()

Step1: Declare a node *ptr

Step2: set ptr = head

step3: If (head=NULL) then

Print "Nothing to point"

step4: Else

while (Ptr ->next ! head)

print "Data value", data

Ptr = Ptr -> next

End while

print: ptr-> data

steps: End If

step 6: stop

Program 5. Write a C program that uses functions to perform the following operations on circular linked list: i) Creation ii) Insertion iii) Deletion iv) Traversal

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

   void beginsert ();
   void lastinsert ();
   void begin_delete();
   void last_delete();
   void display();
   void search();
   void main ()
   {
      int choice =0;
      while(choice != 7)
      {
         printf("\n*********Main Menu*********\n");
         printf("\nChoose one option from the following list ...\n");
         printf("\n===============================================\n");
         printf("\n1.Insert in begining\n2.Insert at last\n3.Delete from Beginning\n4.Delete from last\n5.Search for an
element\n6.Show\n7.Exit\n");
         printf("\nEnter your choice?\n");
         scanf("\n%d",&choice);
         switch(choice)
         {
            case 1:
            beginsert();
            break;
            case 2:
            lastinsert();
            break;
            case 3:
            begin_delete();
            break;
            case 4:
            last_delete();
            break;
            case 5:
            search();
            break;
            case 6:
            display();
            break;
            case 7:
            exit(0);
            break;
            default:
            printf("Please enter valid choice..");
         }
      }
```

```
}
void beginsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter the node data?");
        scanf("%d",&item);
        ptr -> data = item;
        if(head == NULL)
        {
            head = ptr;
            ptr -> next = head;
        }
        else
        {
            temp = head;
            while(temp->next != head)
                temp = temp->next;
            ptr->next = head;
            temp -> next = ptr;
            head = ptr;
        }
        printf("\nnode inserted\n");
    }

}
void lastinsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW\n");
    }
    else
    {
        printf("\nEnter Data?");
        scanf("%d",&item);
        ptr->data = item;
        if(head == NULL)
        {
            head = ptr;
            ptr -> next = head;
        }
        else
        {
            temp = head;
            while(temp -> next != head)
```

```
            {
                temp = temp -> next;
            }
            temp -> next = ptr;
            ptr -> next = head;
        }

        printf("\nnode inserted\n");
    }

}

void begin_delete()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\nUNDERFLOW");
    }
    else if(head->next == head)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }

    else
    {   ptr = head;
        while(ptr -> next != head)
            ptr = ptr -> next;
        ptr->next = head->next;
        free(head);
        head = ptr->next;
        printf("\nnode deleted\n");

    }
}
void last_delete()
{
    struct node *ptr, *preptr;
    if(head==NULL)
    {
        printf("\nUNDERFLOW");
    }
    else if (head ->next == head)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");

    }
    else
    {
        ptr = head;
        while(ptr ->next != head)
        {
```

```
            preptr=ptr;
            ptr = ptr->next;
        }
        preptr->next = ptr -> next;
        free(ptr);
        printf("\nnode deleted\n");

    }
}

void search()
{
    struct node *ptr;
    int item,i=0,flag=1;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to search?\n");
        scanf("%d",&item);
        if(head ->data == item)
        {
        printf("item found at location %d",i+1);
        flag=0;
        }
        else
        {
        while (ptr->next != head)
        {
            if(ptr->data == item)
            {
                printf("item found at location %d ",i+1);
                flag=0;
                break;
            }
            else
            {
                flag=1;
            }
            i++;
            ptr = ptr -> next;
        }
        }
        if(flag != 0)
        {
            printf("Item not found\n");
        }
    }

}

void display()
{
```

```
    struct node *ptr;
    ptr=head;
    if(head == NULL)
    {
        printf("\nnothing to print");
    }
    else
    {
        printf("\n printing values ... \n");

        while(ptr -> next != head)
        {

            printf("%d\n", ptr -> data);
            ptr = ptr -> next;
        }
        printf("%d\n", ptr -> data);
    }

}
```

6) i) Implementing the operations on Stack Using Arrays

Aim: - To implement Stack operations using Array data structure.

Algorithm:-

Step1: Declare an array stack[100] to store integer values

Step2: Initialize two integers choice= 0 and top=0

Step3 : Declare ' n' as integer to store no of elements

Step4 : Read a value into 'n'.

Step5: while (choice !=4) do

5.1.. Read a value into ' choice'

5.2. If (choice =1) Then call function pushs()

5.3. if (Choice =2) Then call function pop ()

54. If (choice=3) then call function display ()

S5. IF (choice=4) then call function exit (0)

5-6: Print "please Enter valid choice" as default

Step6 : end while

Step7: Stop..

Algorithm push()

Step1: Declare an integer 'val'

Steps: If (top=n) then

print "Overflow"

Step3: Else

Read a value into 'val'

set top= top+1

set stack [top] = val

print: 'val' pushed into the stack

step4: End If

step5: stop

Algorithm for pop()

Step 1: Declare an integer 'val'

Step 2: If(top==0) then

Print :Underflow"

Step 3: Else

val=stack[top]

print: “element poped from the stack”,val

top=top-1

step 4: End If

Step 5: Stop

Algorithm display()

Step 1: For i=1 to top do

Print : Stack[i]

Step 2: End for

Step 3: If(top=0) then

Print “Stack is empty”

Step 4: End If

Step 5: Stop

Program 6 i).Write a C program that implement stack (its operations) using Arrays

```
#include <stdio.h>
int stack[100],i,j,choice=0,n,top=0;
void push();
void pop();
void display();
void main ()
{
   clrscr();
   printf("Enter the number of elements in the stack:");
   scanf("%d",&n);
   printf("*********Stack operations using array*********");
  printf("\n----------------------------------------------\n");
   while(choice != 4)
   {
    printf("\nChoose one from the below options...\n");
    printf("\n1.Push\n2.Pop\n3.display\n4.Exit\n");
    printf("\n Enter your choice:");
    scanf("%d",&choice);
    switch(choice)
    {
       case 1:
       {
              push();
              break;
       }
       case 2:
       {
              pop();
              break;
       }
       case 3:
       {
              printf("\n Elements in the stack are:\n");
```

```
                display();
                break;
        }
        case 4:
                exit(0);
                break;
        default:
        {
                printf("Please Enter valid choice ");
        }
    }
   }
}
void push ()
{
   int val;
   if (top == n )
   printf("\n Overflow");
   else
   {
    printf("Enter the value:");
    scanf("%d",&val);
    top = top +1;
    stack[top] = val;
    printf("%d pushed in to the stack",val);
   }
}

void pop ()
{
   int val;
   if(top == 0)
   printf("Underflow");
   else
   {
   val=stack[top];
   printf("%d poped from the stack",val);
   top=top-1;
   }
}
void display()
{
   for (i=top;i>0;i--)
   {
    printf("%d\n",stack[i]);
   }
   if(top == 0)
   {
    printf("Stack is empty");
   }
}
```

6.ii)Implementing the operations on stack using pointers

Aim:- To implement stack operations using pointers (or) linked lists.

Algorithm:-

Step-1: Create a node structure for stack and declare a node * head

Step-2: Initialize an integer choice =0

Step-3: Print menu format and read a value into 'choice'.

Step-4: If choice =1 then call function push( )

Step-5: If choice =2 then call function pop( )

Step-6: If choice =3 then call function display( )

Step-7: If choice =4 then exit

Step-8: Print "Invalid choice" as default

Step-9: Stop

Algorithm push( )

Step-1: Declare an integer 'val'

Step-2: Declare a node * ptr and allocate memory to it

Step-3: If (ptr =NULL) then

Print "Not able to push the element"

Step-4: Else

Read a value into 'val' to push into the stack

If (head =NULL) then

Ptr ---> val = val

Ptr ---> next = NULL

head = Ptr

Else

Ptr ---> val = val

Ptr ---> next = head

head = ptr

End If

Print: 'val' "Pushed into the stack"

Step-5: End If

Step-6: Stop

Algorithm Pop( )

Step-1: Declare an integer item and a node *Ptr

Step-2: If (head = NULL) then

print " Underflow "

Step-3: Else

item = head ---> val
Ptr = head
head = head ---> next
tree (ptr)
print " Item poped from the stack "

Step-4: End If

Step-5: Stop

Algorithm display ( )

step-1: Declare an integer 'i' and a node *Ptr;

Step-2: Set Ptr = head

Step-3: If (ptr == NULL) then

Print " stack is empty "

Step-4: Else

Print " Elements in the stack "

While (Ptr! = NULL) do

Print: Ptr ---> val

Ptr = Ptr ---> next

End while

Step-5: End If

Step-6: Stop

Program 6 ii). Write a C program that implement stack (its operations) using Pointers

```
#include <stdio.h>
#include <stdlib.h>
void push();
void pop();
void display();
struct node
{
int val;
struct node *next;
};
struct node *head;

void main ()
{
   int choice=0;
   clrscr();
   printf("\n*********Stack operations using linked list*********\n");
```

```
    printf("\n----------------------------------------------\n");
    while(choice != 4)
    {
        printf("\n\nChose one from the below options...\n");
     printf("\n1.Push\n2.Pop\n3.Display\n4.Exit");
        printf("\n Enter your choice \n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            {
                push();
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {
                display();
                break;
            }
            case 4:
            {
                  printf("Exit....");
                  break;
        }
        default:
        {
                  printf("Please Enter valid choice ");
        }
    };
}
}
void push ()
{
    int val;
    struct node *ptr = (struct node*)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
     printf("not able to push the element");
    }
    else
    {
     printf("Enter the value");
     scanf("%d",&val);
     if(head==NULL)
     {
         ptr->val = val;
         ptr -> next = NULL;
         head=ptr;
     }
     else
     {
```

```
        ptr->val = val;
        ptr->next = head;
        head=ptr;

    }
    printf("%d pushed into the stack",val);

   }
}

void pop()
{
   int item;
   struct node *ptr;
   if (head == NULL)
   {
    printf("Underflow");
   }
   else
   {
    item = head->val;
    ptr = head;
    head = head->next;
    free(ptr);
    printf("%d popped from the stack",item);

   }
}
void display()
{
   int i;
   struct node *ptr;
   ptr=head;
   if(ptr == NULL)
   {
      printf("Stack is empty\n");
   }
   else
   {
    printf("Elements in the Stack are: \n");
      while(ptr!=NULL)
      {
          printf("%d\n",ptr->val);
          ptr = ptr->next;
      }
   }
}
```

7. i) Implementing the operations on Queues using Arrays

Aim:- To implement operations on queue using array data structure

Algorithm:-

Step-1: start

Step-2: Define a constant ‘maxsize‘ and the value of maxsize = 5

Step-3: Declare and initialize two integers front = -1, rear = -1

Step-4: Declare an integer array queue[maxsize]

Step-5: Declare an integer ‘choice ‘ to read choice

Step-6: Print “Menu Format “ and read a value into choice

Step-7: If choice = 1 then call function insertq( )

Step-8: If choice = 2 then call function deleteq( )

Step-9: If choice = 3 then call function display( )

Step-10: If choice = 4 then call function exit(0)

Step-11: Print “ Invalid choice “ as default

Step-12: Stop

Algorithm insertq( )

Step-1: Declare an integer ‘item’

Step-2: If (rear = maxsize-1)

Print “overflow” and return

Step-3: Else

Read a value into ‘item’

If (front = -1 && rear = -1) then

Set front = 0, rear = 0

Else

rear = rear+1

End If

queue [rear = item]

print “Item inserted “

Step-4: End If

Step-5: Stop

Algorithm deleteq( )

Step-1: Declare an integer variable ‘item’

Step-2: If ((front = -1) or (front > rear)) then

Print “overflow” and return

Step-3: Else

item = queue [front]

if (front = rear)

front = -1 , rear = -1

Else

front = front+1

End If

Print "Item deleted"

Step-4: End If

Step-5: Stop

Algorithm display( )

Step-1: Declare an integer 'I'

Step-2: If (rear = -1) then

Print "Queue is empty "

Step-3: Else

Print "Elements in the stack are"

For i = front to rear do

Print : que[i]

End for

Step-4: End If

Step-5: Stop

Program 7 i). Write a C program that implement Queue (its operations) using Arrays

```
#include<stdio.h>
#include<stdlib.h>
#define maxsize 5
void insertq();
void deleteq();
void display();
int front=-1,rear=-1;
int queue[maxsize];
void main ()
{
   int choice;
   clrscr();
   while(choice != 4)
   {
    printf("\n************************Main Menu****************************\n");
    printf("\n=================================================================\n");
    printf("\n1.Insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");
    printf("\nEnter your choice:");
    scanf("%d",&choice);
    switch(choice)
    {
       case 1:insertq();
```

```
            break;
      case 2:deleteq();
            break;
      case 3:display();
            break;
      case 4:exit(0);
            break;
      default:printf("\nEnter valid choice\n");
    }
   }
}

void insertq()
{
   int item;
   if(rear == maxsize-1)
   {
    printf("\nOVERFLOW\n");
    return;
   }
   printf("\n Enter the element:");
   scanf("%d",&item);
   if(front == -1 && rear == -1)
   {
    front = 0;
    rear = 0;
   }
   else
   {
    rear = rear+1;
   }
   queue[rear] = item;
   printf("\n%d inserted ",item);
}
void deleteq()
{
   int item;
   if (front == -1 || front > rear)
   {
    printf("\nUNDERFLOW\n");
    return;
   }
   else
   {
    item = queue[front];
    if(front == rear)
    {
      front = -1;
      rear = -1 ;
    }
    else
    {
      front = front + 1;
    }
    printf("\n%d deleted ",item);
   }
```

```
}

void display()
{
   int i;
   if(rear == -1)
   {
    printf("\nEmpty queue\n");
   }
   else
   {   printf("\n Elements in the Queue are:\n");
    for(i=front;i<=rear;i++)
    {
        printf("%d ",queue[i]);
    }
   }
}
```

7.ii) Implementing the operations on Queue using Pointers

Aim:- To implement Queue operations by using pointers or linked lists

Algorithm:-

Step-1: Start

Step-2: Create a node structure for Queue

Step-3: Declare two nodes *front and *rear

Step-4: Declare an integer variable ‘choice’

Step-5: Print Menu format and read a value into ‘choice’

Step-6: If choice = 1 then call function insertq( )

Step-7: If choice = 2 then call function deleteq( )

Step-8: If choice = 3 then call function display( )

Step-9: if choice = 4 then call function exit(0)

Step-10: Print “Invalid choice” as default

Step-11: Stop

Algorithm insertq( )

Step-1: Declare a node *ptr and an integer ‘item’

Step-2: Allocate memory for ‘ptr’

Step-3: If (ptr = NULL) then

Print “overflow” and return

Step-4: Else

Read a value into ‘item’

If (front = NULL) then

Set front = rear = ptr

Set front ---> next = rear ---> next = NULL

Else

rear ---> next = ptr

rear = ptr

rear ---> next = NULL

End If

Step-5: End If

Step-6: Stop

Algorithm deleteq( )

Step-1: Declare a node *ptr

Step-2: If (front = NULL) then

Print “Underflow” return

Step-3: Else

Ptr = front

front = front ---> next

tree(ptr)

Step-4: End If

Step-5: Stop

Algorithm display ( )

Step-1: Declare a node *ptr and set ptr = front

Step-2: If (front = NULL) then

Print “Queue is empty”

Step-3: Else

Print “Elements in the Queue are:”

While (ptr! = NULL) do

Print : ptr ---> data

Ptr = ptr ---> next

End while

Step-4: End If

Step-5: Stop

Program 7 ii). Write a C program that implement Queue (its operations) using Pointers

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *front;
struct node *rear;
void insertq();
void deleteq();
void display();
void main()
{
   int choice;
   clrscr();
   while(choice != 4)
   {
    printf("\n*************************Main Menu*****************************\n");
    printf("\n========================================================================\n");
    printf("\n1.Insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");
    printf("\nEnter your choice:");
    scanf("%d",&choice);
    switch(choice)
    {
       case 1:insertq();
                break;
       case 2:deleteq();
                break;
       case 3:display();
                break;
       case 4:exit(0);
                break;
       default:printf("\nEnter valid choice\n");
    }
   }
}
void insertq()
{
   struct node *ptr;
   int item;
   ptr = (struct node *) malloc (sizeof(struct node));
   if(ptr == NULL)
   {
    printf("\nOVERFLOW\n");
    return;
   }
   else
```

```
    {
     printf("\nEnter value:");
     scanf("%d",&item);
     ptr -> data = item;
     if(front == NULL)
     {
         front = ptr;
         rear = ptr;
         front -> next = NULL;
         rear -> next = NULL;
     }
     else
     {
         rear -> next = ptr;
         rear = ptr;
         rear->next = NULL;
     }
    }
}
void deleteq()
{
    struct node *ptr;
    if(front == NULL)
    {
     printf("\nUNDERFLOW\n");
     return;
    }
    else
    {
     ptr = front;
     front = front -> next;
     free(ptr);
    }
}
void display()
{
    struct node *ptr;
    ptr = front;
    if(front == NULL)
    {
     printf("\nEmpty queue\n");
    }
    else
    {   printf("\nElements in the Queue are:\n");
     while(ptr != NULL)
     {
         printf("%d ",ptr -> data);
         ptr = ptr -> next;
     }
    }
}
```

8.Conversion of infix expression into postfix expression

Aim :- To convert infix expression into postfix expression by using stacks.

Algorithm:-

Steps:

1. Start
2. Define a constant SIZE and set SIZE=100.
3. Declare a character array stack[SIZE]
4. Declare an integer variable top and assign top=-1
5. Declare two characters arrays infix[SIZE] and postfix[SIZE]
6. Read an infix expression into ‘infix’ array.
7. Call function Infix to postfix(infix,postfix)
8. Display the postfix expression from the array ‘postfix’
9. Stop

Algorithm infix to postfix (char infix_exp[], char post_exp[])

Steps:

1. Declare two integers I,j
2. Declare two characters x,item
3. Push(‘(’)
4. Use function strcat(infix_exp,”)”)
5. Set i=0 and j=0
6. Set item =infix_exp[i]
7. While (item!=’ \o’)do

If (item=’ (‘)then

Push (item)

Else if (is digit(item)!! Is alpha(item) then

Postfix_exp[j]=item

j=j+1

If (is_operator(item)=1)then

X=pop()

While (is_operator(x)=1 precedence(x) >=precendence(item))

postfix_exp[j]=x

j=j+1

x=pop()

Endwhile

else

print “ivalid expression!”

Exit(1)

Set i=i++and item =infix_exp[i]

End if

8. End while

9. if (top >0)then

   Print “invalid expression”

   Exit

10. End if

11. Stop

Algorithm for push (char item)

Steps:

1. If (top>= SIZE-1) then

   Print “ stack overflow”

2. else

   top =top +1

   stack[top]=item

3. end if

4. stop

Algorithm for char pop()

Steps:

1. Declare a charater variable ‘item’
2. If (top < 0)then

   Print “ stack underflow: invalid infix expression

   Return 1

3. Else
   Item = stack[top]
   Top = top-1
   Return (item)
4. End if
5. Stop

Algorithm for is_operator(char symbol)

Steps:

1. If(symbol=’^’ | symbol = ‘*’ | symbol =’/’ |symbol=’+’|symbol = ‘- ‘) then
   Return 1
2. Else
   Return 0
3. Stop

Algorithm for precedence (char symbol)

Steps:

1. If (symbol=’ ^’)then
   Return 3
2. Else if (symbol = ‘*’ || symbol =’ /’) then
   Return 2
3. Else if (symbol = ‘+’ || symbol =’ -’) then
   Return 1
4. Else
   Retrun 0
5. End if
6. Stop

Program 8. Write a C program that Uses Stack Operations to Convert Infix expression into Postfix expression

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define SIZE 100
char stack[SIZE];
int top=-1;
void push(char item)
{
    if (top >= SIZE - 1)
      printf("\nStack Overflow.");
    else
     {
     top = top + 1;
     stack[top] = item;
     }
}
char pop()
{
    char item;
    if (top < 0)
     {
     printf("stack under flow: invalid infix expression");
     return 1;
    }
    else
     {
     item = stack[top];
     top = top - 1;
     return (item);
    }
}
int is_operator(char symbol)
{
    if (symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol == '-')
     return 1;
    else
```

```
        return 0;
}
int precedence(char symbol)
{
    if (symbol == '^')
     return (3);
    else if (symbol == '*' || symbol == '/')
     return (2);
    else if (symbol == '+' || symbol == '-')
     return (1);
    else
       return (0);
}
void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
    int i, j;
    char item;
    char x;
    push('(');
    strcat(infix_exp, ")");
    i = 0;
    j = 0;
    item = infix_exp[i];
    while (item != '\0')
    {
     if (item == '(')
       push(item);
     else if (isdigit(item) || isalpha(item))
      {
         postfix_exp[j] = item;
         j++;
      }
     else if (is_operator(item) == 1)
     {
         x = pop();
         while (is_operator(x) == 1 && precedence(x) >= precedence(item))
          {
                 postfix_exp[j] = x;
                 j++;
                 x = pop();
          }
         push(x);
         push(item);
     }
     else if (item == ')')
     {
         x = pop();
         while (x != '(')
         {
                 postfix_exp[j] = x;
                 j++;
                 x = pop();
         }
     }
     else
     {
```

```
        printf("\nInvalid infix Expression.\n");
        exit(1);
    }
    i++;
    item = infix_exp[i];
   }

   if (top > 0)
    {
    printf("\nInvalid infix Expression.\n");
    exit(1);
   }
   if (top > 0)
    {
    printf("\nInvalid infix Expression.\n");
    exit(1);
    }
   postfix_exp[j] = '\0';
}
void main()
{
   char infix[SIZE], postfix[SIZE];
   clrscr();
   printf("ASSUMPTION: The infix expression contains single letter variables and single digit constants only.\n");
   printf("\nEnter Infix expression : ");
   gets(infix);
   InfixToPostfix(infix, postfix);
   printf("Postfix Expression: ");
   puts(postfix);
   getch();
}
```

9. Evaluation of postfix expression using stack operation

Aim:- To evaluate the given postfix expression by using operation of stack

Algorithm:-

1. Declare an integer constant SIZE and set SIZE=40
2. Declare a character array postfix[SIZE]
3. Declare an integer array stack[SIZE] and set top=1
4. Declare integers i,a,b,result, peval
5. Declare character variable ‘ch’
6. For i=0 to SIZE do
   Stack[i]=-1
7. End for
8. Read a postfix expression into ‘postfix’
9. For i=0 to postfix[i]!=’\0’ do
   Ch=postfix[i]
   If(isdigit(ch))
      Push (ch_’0’)
   Elseif (ch==’t’||ch==’-‘||ch==’*’||ch==’/’)
      b = pop()

a =pop()
if ch=' +' then set result=a+b & return
if ch=' -' then set result=a-b & return
if ch=' *' then set result=a*b & return
if ch=' /' then set result=a/b & return
if ch=' %' then set result=a%b & return
push (result)
end if

10. End for
11. Peval=pop()
12. print("postvaluation" : peval)
13. stop

Program 9. Write a C program that Uses Stack Operations to Evaluate the Postfix expression

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#define SIZE 40
int pop();
void push(int);
char postfix[SIZE];
int stack[SIZE],top=-1;
void main()
{
int i, a, b, result, pEval;
char ch;
clrscr();
for(i=0; i<SIZE; i++)
  stack[i] = -1;
printf("\nEnter a postfix expression: ");
scanf("%s",postfix);
for(i=0; postfix[i] != '\0'; i++)
{
 ch = postfix[i];
 if(isdigit(ch))
   push(ch-'0');
 else if(ch == '+' || ch == '-' || ch == '*' || ch == '/')
  {
   b = pop();
   a = pop();
   switch(ch)
    {
     case '+': result = a+b;
           break;
     case '-': result = a-b;
           break;
     case '*': result = a*b;
           break;
     case '/': result = a/b;
           break;
     case '%': result = a%b;
           break;
    }
   push(result);
  }
 }
```

```
 pEval = pop();
 printf("\nThe postfix evaluation is: %d\n",pEval);
 getch();
}
void push(int n)
{
 if (top < SIZE -1)
   stack[++top] = n;
 else
  {
   printf("Stack is full!\n");
   exit(-1);
  }
}
int pop()
{
 int n;
 if (top > -1)
  {
   n = stack[top];
   stack[top--] = -1;
   return n;
   }
  else
  {
   printf("Stack is empty!\n");
   return -1;
  }
}
```

10.Implementing the operations on a binary tree

Aim:-To implement Creation,Inorder,Preorder and postorder traversal operations on a binary tree using functions.

Algorithm:-

Step-1: Start

Step-2: Create a node structure to implement binary tree operations.

Step-3: Declare a node *root and call function create (1)

Step-4: Call function insertLeft (root,4)

Step-5: Call function insertRight (root,6)

Step-6: Call function insertLeft (root--->left,42)

Step-7: Call function insertRight (root--->left,3)

Step-8: Call function insertLeft (root--->right,2)

Step-9: Call function insertRight (root--->right,33)

Step-10: Call function InorderTraversal (root) to display the elements in the tree in inorder fashion

Step-11: Call function PreorderTraversal (root) to display the elements in the tree in in preorder fashion

Step-12: Call function postorderTraversal (root) to display the elements in the tree in postorder fashion

Step-13: Stop

Algorithm inorderTraversal (Struct node *root)

Step-1: If (root = NULL) then

Return

Step-2: Else

inorderTraversal (root--->left)

print :root--->item

inorderTraversal (root--->right)

Step-3: End If

Step-4: Stop

Algorithm PreorderTraversal (Struct node *root)

Step-1: If (root = NULL) then

return

Step-2: Else

Print: root--->item

PreorderTraversal (root--->left)

PreorderTraversal (root--->right)

Step-3: End If

Step-4: Stop

Algorithm PostorderTraversal (Struct node *root)

Step-1: If (root = NULL) then

return

Step-2: Else

PostorderTraversal (root --->left)

PostorderTraversal (root --->right)

Print: root--->item

Step-3: End If

Step-4: Stop

Algorithm Create (int value)

Step-1: Allocate memory for 'newNode'

Step-2: newNode --->item = value

Step-3: newNode --->left = NULL

Step-4: newNode --->right = NULL

Step-5: return newNode

Step-6: Stop

Algorithm insertLeft (Struct node *root,int value)

Step-1: root ---> left = Create(value)

Step-2: return root ---> left

Step-3: Stop

Algorithm insertRight (Struct node *root,int value)

Step-1: root --->right = Create(value)

Step-2: return root ---> right

Step-3: Stop

Program 10. Write a C program that uses functions to perform the following
i) creating a binary tree of integers ii) Traversing the above binary tree in preorder, inorder and post order

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

```
struct node
 {
  int item;
  struct node* left;
  struct node* right;
};
void inorderTraversal(struct node* root)
 {
  if (root == NULL) return;
  inorderTraversal(root->left);
  printf("%d ", root->item);
  inorderTraversal(root->right);
}
void preorderTraversal(struct node* root)
{
  if (root == NULL) return;
  printf("%d ", root->item);
  preorderTraversal(root->left);
  preorderTraversal(root->right);
}
void postorderTraversal(struct node* root)
 {
  if (root == NULL) return;
  postorderTraversal(root->left);
  postorderTraversal(root->right);
  printf("%d ", root->item);
}
struct node* create(int value)
 {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->item = value;
  newNode->left = NULL;
  newNode->right = NULL;

  return newNode;
}
struct node* insertLeft(struct node* root, int value)
 {
  root->left = create(value);
  return root->left;
}
struct node* insertRight(struct node* root, int value)
 {
  root->right = create(value);
  return root->right;
}

void main()
 {
  struct node* root = create(1);
  clrscr();
  insertLeft(root,4);
  insertRight(root,6);
  insertLeft(root->left,42);
  insertRight(root->left,3);
  insertLeft(root->right,2);
```

```
  insertRight(root->right,33);
  printf("Traversal of the inserted binary tree \n");
  printf("Inorder traversal \n");
  inorderTraversal(root);
  printf("\nPreorder traversal \n");
  preorderTraversal(root);
  printf("\nPostorder traversal \n");
  postorderTraversal(root);
  getch();
}
```

11) Implementing the operations on Binary search Tree

Aim:- To implement Create,insert and delete operations on binary search trees using functions

Algorithm:-

Step-1: Start

Step-2: Create a node structure to implement binary search tree operations

Step-3: Declare a node *root

Step-4: Call function new_node and assign to 'root'

Step-5: Call function 'insert' to insert new nodes

Step-6: Call function 'inorder' to display the nodes

Step-7: Call function 'delete' to remove the nodes

Step-8: Stop

Algorithm new_node (int X)

Step-1: Create a node *temp and allocate memory to it

Step-2: temp ---> data = X

Step-3: temp ---> left_child = NULL

Step-4: temp ---> right_child = NULL

Step-5: return temp

Algorithm insert (Struct node *root,int X)

Step-1: If (root == NULL) then

Return new_node (X)

Step-2: Else If (X > root ---> data) then

root ---> right_child = insert (root ---> right_child,X)

Step-3: Else

root ---> left_child = insert (root ---> left_child,X)

Step-4: End If

Step-5: return root

Step-6: Stop

Algorithm delete (Struct node *node root,int X)

Step-1: If (root = NULL) then

return NULL

Step-2: If (X > root ---> data) then

root ---> right_child = delete (root ---> right_child,X)

Step-3: Else if (X < root ---> data) then

root ---> left-child = delete (root---> left_child,X)

Step-4: Else if (root ---> left_child = NULL && root ---> right_child = NULL) then

tree (root)

return NULL

Step-5: Else if (root ---> left_child = NULL !! root ---> right_child = NULL) then

Create a node *temp

If (root ---> left_child = NULL) then

temp = root ---> right_child

Else

temp = root ---> left_child

End If

tree(root)

return temp

Step-6: Else

Struct node *temp = find_minimum (root ---> right_child)

root ---> data = temp ---> data

root ---> right_child = delete (root ---> right_child,temp ---> data)

Step-7: End If

Step-8: Return root

Step-9: Stop

Algorithm inorder (Struct node * root)

Step-1: If (root ! = NULL) then

Inorder (root ---> left_child)

Print : root ---> data

Inorder (root ---> right_child)

Step-2: End If

Step-3: Stop

Algorithm find_minimum (struct node * root)

Step-1: If (root = NULL) then

return NULL

Step-2: Else if (root ---> left_child ! = NULL) then

return find_minimum (root ---> left_child)

Step-3: End If

Step-4: return root

Program 11. Write a C program that uses functions to perform the following operations on Binary search Tree: i) Creation ii) Insertion iii) Deletion

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

struct node* new_node(int x)
{
  struct node *temp;
  temp = malloc(sizeof(struct node));
  temp -> data = x;
  temp -> left_child = NULL;
  temp -> right_child = NULL;
```

```
  return temp;
}
struct node* search(struct node * root, int x)
 {
  if (root == NULL || root -> data == x)
    return root;
  else if (x > root -> data)
    return search(root -> right_child, x);
  else
    return search(root -> left_child, x);
}

struct node* insert(struct node * root, int x)
 {
  if (root == NULL)
    return new_node(x);
  else if (x > root -> data)
    root -> right_child = insert(root -> right_child, x);
  else
    root -> left_child = insert(root -> left_child, x);
  return root;
}

struct node* find_minimum(struct node * root)
 {
  if (root == NULL)
    return NULL;
  else if (root -> left_child != NULL)
    return find_minimum(root -> left_child);
  return root;
}

struct node* delete(struct node * root, int x)
 {
  if (root == NULL)
    return NULL;
  if (x > root -> data)
    root -> right_child = delete(root -> right_child, x);
  else if (x < root -> data)
    root -> left_child = delete(root -> left_child, x);
  else
   {
    if (root -> left_child == NULL && root -> right_child == NULL)
    {
      free(root);
      return NULL;
    }

    else if (root -> left_child == NULL || root -> right_child == NULL)
     {
      struct node *temp;
      if (root -> left_child == NULL)
     temp = root -> right_child;
      else
     temp = root -> left_child;
      free(root);
```

```
      return temp;
    }

    else
     {
      struct node *temp = find_minimum(root -> right_child);
      root -> data = temp -> data;
      root -> right_child = delete(root -> right_child, temp -> data);
    }
  }
  return root;
}

void inorder(struct node *root)
{
  if (root != NULL)
  {
    inorder(root -> left_child);
    printf(" %d ", root -> data);
    inorder(root -> right_child);
  }
}

void main()
 {
  struct node *root;
  clrscr();
  root = new_node(20);
  insert(root,5);
  insert(root,1);
  insert(root,15);
  insert(root,9);
  insert(root,7);
  insert(root,12);
  insert(root,30);
  insert(root,25);
  insert(root,40);
  insert(root,45);
  insert(root,42);
  printf("\nInorder Traversal After Insertion is:\n");
  inorder(root);
  printf("\n");
  root = delete(root,1);
  root = delete(root,40);
  root = delete(root,45);
  root = delete(root,9);
  printf("\n Inorder Traversal after Deletion is:\n");
  inorder(root);
  printf("\n");
  getch();
}
```

12.i). Implementation of Quick sort

Aim:- To implement Quick sort method to sort a given list of integers in ascending order.

Algorithm:-

Step-1: Start

Step-2: Initialize an integer array 'a'

Step-3: Declare an integer 'n' to store no. of elements of an array 'a'

Step-4: Call print Arr (a,n) functon to display elements before sorting.

Step-5: Call quick (a,o,n-1) function

Step-6: Call Print Arr(a,n) function to display elements after sorting.

Step-7: Stop

Algorithm quick(int a[ ], int start, int end)

Step-1: If (start < end) then

P= Partition(a, start, end)

quick (a, start, P-1)

quick (a, P+1, end)

Step-2: End If

Step-3: Stop

Algorithm PrintArr (int a [ ], int n)

Step-1: Declare an integer variable 'i'

Step-2: for i=0 to n-1 do

print:a[i]

Step-3: End for

Step-4: Stop

Algorithm partition (int a[ ], int start, int end)

step-1: Declare an integer 'pivot' and set pivot = a[end]

Step-2: Declare an integer 'I' and set i=(start − 1)

Step-3: Declare two integers variables j, temp

Step-4: for j = start to end-1 do

If (a [j]< pivot) then

```
i = i+1
        temp = a [ i ]
        a[ i ] = a[ j ]
        a[ j ] = temp
   end if
```

Step-5: End for

Step-6: temp = a[i+1]

Step-7: a [i+1] = a[end]

Step-8: a [end] = temp

Step-9: return (i+1)

Step-10: Stop

12. i) Write a program that implements the Quick sort method to sort a given list of integers in ascending order i) Quick sort ii) Merge sort

```
#include <stdio.h>
int partition (int a[], int start, int end)
{
   int pivot = a[end];
   int i = (start - 1),j,temp;
   for ( j = start; j <= end - 1; j++)
   {
    if (a[j] < pivot)
    {
        i++;
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
   }
   temp= a[i+1];
   a[i+1] = a[end];
   a[end] = temp;
   return (i + 1);
}
void quick(int a[], int start, int end)
{
   if (start < end)
   {
    int p = partition(a, start, end);
    quick(a, start, p - 1);
    quick(a, p + 1, end);
   }
}
void printArr(int a[], int n)
{
   int i;
   for (i = 0; i < n; i++)
    printf("%3d", a[i]);
}
```

```
void main()
{
   int a[] = { 24, 9, 29, 14, 19, 27 };
   int n = sizeof(a) / sizeof(a[0]);
   clrscr();
   printf("Before sorting array elements are: \n");
   printArr(a, n);
   quick(a, 0, n-1);
   printf("\nAfter sorting array elements are: \n");
   printArr(a, n);
   getch();
}
```

12. ii) Implementation of Merge Sort

Aim:- To implement merge sort method to sort a given list of integers in ascending order.

Algorithm:-

Step-1: Initialize an integer array 'a'

Step-2: Declare an integer variable 'n' to store no. of elements of an array 'a'

Step-3: Call function print Array (a,n) to display the array elements before sorting

Step-4: Call merge sort (a,o,n-1) function

Step-5: Call function Print Array (a,n) to display the array elements after sorting

Step-6: Stop

Algorithm merge sort(int a [ ], int low, int high)

Step-1: If (low<high) then

mid = (low+high)/2

mergesort (a,low,mid)

mergesort (a,mid+1,high)

merge (a,low,mid,high)

Step-2: End If

Step-3: stop

Algorithm merge (int a [ ], int low, int mid, int high)

Step-1: Declare 3 integers i, j and k.

Step-2: Initialize n_1 mid-low+ and n_2 = high-mid

Step-3: Declare two integers array L [10] and R[10].

Step-4: for i=0 to n_1-1 do

L [i] = a[low+i]

Step-5: End for

Step-6: for j=0 to n_2 -1 do

R[j] = a[mid+1+j]

Step-7: End for

Step-8: Set i=j=0 and k=low

Step-9: while (i<n_1 && j < n_2) do

If (L [i] < = R[j]) then

A[k] = L [i]

i = i+1

Else

A [k] = R[j]

J = j+1

End If

K = k+1

Step-10: End while

Step-11: while (i<n_1) do

a[k] = L [i]

i = i+1, k = k+1

Step-12: End while

Step-13: while (j<n_2) do

a[k] = R [j]

j = j+1, k = k+1

Step-14: End while

Step-15: Stop

Algorithm Print array(int a[ ], int n)

Step-1: Declare an integer ‘i’

Step-2: for (i =0 to n-1) do

Print : a[i]

Step-3: End for

Step-4: Stop

12. ii) Write a program that implements the Merge sort method to sort a given list of integers in ascending order

```
#include<stdio.h>
void merge(int a[], int low, int mid, int high);
void mergeSort(int a[], int low, int high)
{
 if (low<high)
 {
  int mid = (low+high) / 2;
  mergeSort(a,low, mid);
  mergeSort(a,mid+1,high);
  merge(a,low,mid,high);
 }
}
void merge(int a[], int low, int mid, int high)
{
 int i,j,k;
 int n1 = mid-low+1;
 int n2 = high-mid;
 int L[10],R[10];
 for (i=0;i<n1;i++)
  L[i] = a[low + i];
 for (j = 0; j < n2; j++)
  R[j] = a[mid + 1 + j];
 i = 0;
 j = 0;
 k = low;
 while (i < n1 && j < n2)
  {
  if (L[i] <= R[j])
  {
   a[k] = L[i];
   i++;
  }
  else
   {
   a[k] = R[j];
   j++;
  }
  k++;
 }

 while (i < n1)
  {
  a[k] = L[i];
  i++;
  k++;
 }

 while (j < n2)
  {
  a[k] = R[j];
  j++;
  k++;
 }
}
```

```
void printArray(int a[], int n)
{
 int i;
 for (i = 0; i <n; i++)
  printf("%3d", a[i]);
 printf("\n");
}
void main()
{
 int a[] = { 28, 95, 13, 56, 6, 17 };
 int n = sizeof(a) / sizeof(a[0]);
 clrscr();
 printf("Given array is: \n");
 printArray(a,n);

 mergeSort(a, 0,n-1);

 printf("\nSorted array is: \n");
 printArray(a,n);
 getch();
}
```

13. i) Implementation of Depth First search (DFS)

Aim:- To implement depth first search graph traversal technique

Algorithm:-

Step-1: Start

Step-2: Declare array a [20][20], reach[20]

Step-3: Declare an integer ‘n’ to store the no. of vertices

Step-4: Declare two integer i and j

Step-5: Initialize an integer count = 0

Step-6: Read 'n' to store no. of vertices of graph

Step-7: for (i =1 to n) do

Reach [i] = 0

For (j = 1 to n) do

a[i] [j] =0

End for

Step-8: End for

Step-9: Display the adjacency matrix

Step-10: Call function dfs (1)

Step-11: for (i =1 to n) do

If (reach[i]) then

Count = count+1

End if

Step-12: End for

Step-13: if (count = n) then

Print " Graph is not connected "

Step-14: Else

Print " graph is not connected "

Step-15: End If

Step-16: Stop

Algorithm dfs (int V)

Step-1: Declare an integer 'i'

Step-2: Initialize reach [V] =1

Step-3: for (i =1 to n) do

If (a [v] [i] && ! reach [i]) then

Print "visited order " : V i

dfs (1)

end if

Step-4: end for

Step-5: Stop

Algorithm deleteq( )

Step-1: Declare a node *ptr

Step-2: If (front = NULL) then

Print "Underflow" return

Step-3: Else

Ptr = front

front = front ---> next

tree(ptr)

Step-4: End If

Step-5: Stop

13.i). Write a program to implement the DFS graph traversal method

```
#include<stdio.h>
#include<conio.h>
int a[20][20],reach[20],n;
 void dfs(int v)
  {
    int i;
    reach[v]=1;
    for (i=1;i<=n;i++)
      if(a[v][i] && !reach[i])
      {
             printf("\n %d->%d",v,i);
             dfs(i);
      }
    }
   void main()
   {
    int i,j,count=0;
    clrscr();
    printf("\n Enter number of vertices:");
    scanf("%d",&n);
    for (i=1;i<=n;i++)
     {
             reach[i]=0;
             for (j=1;j<=n;j++)
               a[i][j]=0;
    }
    printf("\n Enter the adjacency matrix:\n");
    for (i=1;i<=n;i++)
      for (j=1;j<=n;j++)
       scanf("%d",&a[i][j]);
    dfs(1);
    printf("\n");
    for (i=1;i<=n;i++)
     {
             if(reach[i])
               count++;
    }
    if(count==n)
      printf("\n Graph is connected"); else
      printf("\n Graph is not connected");
```

```
getch();
}
```

13. ii) Implementation of Breadth First Search (BFS)

Aim:- To implement breadth first search (BFS) graph traversal technique.

Algorithm:-

Step-1: Start

Step-2: Declare three integers n, i, j

Step-3: Declare two integer arrays visited [10] and queue[10]

Step-4: Initialize two integers front = -1 and rear = -1

Step-5: Declare an array adj [10][10]

Step-6: Declare an integer 'V' to store starting vertex

Step-7: Read a value into 'n' to store no. of vertices

Step-8: for (i =1 to n) do

queue [i] = 0

visited [i] = 0

Step-9: end for

Step-10: Read the values into the adjacency matrix adj [i] [j]

Step-11: Read a value into 'V'

Step-12: call function bfs (v)

Step-13: Print " The nodes which are reachable are :"

Step-14: for (i = 1 to n) do

If (visited [i]) then

Print : i

Else

Print " BFS is not Possible "

End if

Step-15: end for

Step-16: Stop

Algorithm bfs (int V)

Step-1: for (i= 1 to n) do

If (adj [v] [i] && ! visited [i]) then

Queue [rear+1] =1

Else if (front < = rear) then

Visited [queue [front]] =1

Bfs (queue [front+1]]

End if

Step-2: end for

Step-3: Stop

13.ii). Write a program to implement the BFS graph traversal method

```
#include <stdio.h>
int n, i, j, visited[10], queue[10], front = -1, rear = -1;
int adj[10][10];
void bfs(int v)
{
   for (i = 1; i <= n; i++)
    if (adj[v][i] && !visited[i])
        queue[++rear] = i;
   if (front <= rear)
   {
    visited[queue[front]] = 1;
    bfs(queue[front++]);
   }
}

void main()
{
   int v;
   clrscr();
   printf("Enter the number of vertices: ");
   scanf("%d", &n);
   for (i = 1; i <= n; i++)
   {
    queue[i] = 0;
    visited[i] = 0;
   }
   printf("Enter graph data in matrix form:    \n");
   for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
        scanf("%d", &adj[i][j]);
   printf("Enter the starting vertex: ");
   scanf("%d", &v);
   bfs(v);
   printf("The node which are reachable are:    \n");
   for (i = 1; i <= n; i++)
    if (visited[i])
        printf("%d\t", i);
    else
        printf("BFS is not possible. Not all nodes are reachable");
  getch();
}
```