1. IMPLEMENTATION OF SINGLY LINKED LIST

AIM:

To implement singly linked list using C.

ALGORITHM:

1. Start the program.
2. Create the list by getting no. of nodes and nodes as an input.
3. Display the list.
4. Enter the choice
   a) If choice is 1, Insertion takes place at the first position.
   b) If choice is 2, Insertion takes place after the given node.
   c) If choice is 3, Insert an item into the list at last.
   d) If choice is 4, Delete an item at beginning.
   e) If choice is 5, Delete the given item from the list.
   f) If choice is 6, Delete at end.
   g) If choice is 7, then Exit.
5. Stop the program.

PROGRAM:

```
#include  <stdio.h>
#include <stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node *head=NULL,*p, *t;
void display(void);

int main()
{
 int ch,n,i,x;
 clrscr();
 printf ("Enter the number of nodes:\t");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
  p=malloc(sizeof(struct node));
```

```
 printf("Enter the data of node%d \t",i+1);
 scanf("%d",&p->data);
 p->next=NULL;
 if(head==NULL)
   head=p;
 else
   t->next = p;
 t=p;
 }
display();
printf("Menu");
printf("\n 1.Insert at beginning \n 2.Insert after the given node \n 3.Insert at end \n");
printf(" 4.Delete at beginning \n 5.Delete the given node \n 6.Delete at last \n 7.Exit \n\n");
do
{
 printf("Enter the option \t");
 scanf("%d",&ch);
 t=head;
 switch(ch)
 {
  case 1:
       p=malloc(sizeof(struct node));
       printf("Enter the data to be inserted \t");
       scanf("%d",&p->data);
       p->next=head;
       head=p;
       display();
       break;
case 2:
       printf("Enter the previous node value where the new node to be inserted \t");
       scanf("%d",&x);
       p=malloc(sizeof(struct node));
       printf("Enter the data to be inserted \t");
       scanf("%d",&p->data);
       t=head;
```

```
        while(x!=t->data)
            t=t->next;
        p->next=t->next;
        t->next=p;
        display();
        break;
case 3:
        p=malloc(sizeof(struct node));
        printf("Enter the data to be inserted \t");
        scanf("%d",&p->data);
        t=head;
        while(t->next!=NULL)
            t=t->next;
        t->next=p;
        p->next=NULL;
        display();
        break;
case 4:
        head=head->next;
        display();
        break;
case 5:
        printf("Enter the node value to be deleted \t");
        scanf("%d",&x);
        t=head;
        while(x!=t->data)
        {
            p=t;
            t=t->next;
        }
        if(t==head)
           head=head->next;
        else
            p->next=t->next;
        display();
        break;
```

```
case 6:
        t=head;
        while(t->next!=NULL)
        {
           p=t;
           t=t->next;
        }
        if(t==head)
           head=NULL;
        else
           p->next=NULL;
        display();
        break;
case 7:
        printf("Thank You");
        getch();
 }
}while (ch!=7);
return 0;
}

void display ()
{
 printf("The singly linked list is \n");
 t=head;
 while(t!=NULL)
 {
   printf("%d-->",t->data);
   t=t->next;
 }
 printf("NULL\n\n");
}
```

SAMPLE OUTPUT:

```
Enter the number of nodes:    3
Enter the data of node1       10
Enter the data of node2       20
Enter the data of node3       30
The singly linked list is
10-->20-->30-->NULL

Menu
 1.Insert at beginning
 2.Insert after the given node
 3.Insert at end
 4.Delete at beginning
 5.Delete the given node
 6.Delete at last
 7.Exit
Enter the option        1
Enter the data to be inserted   100
The singly linked list is
100-->10-->20-->30-->NULL
Enter the option        2
Enter the previous node value where the new node to be inserted     20
Enter the data to be inserted   200
The singly linked list is
100-->10-->20-->200-->30-->NULL
Enter the option        3
Enter the data to be inserted   300
The singly linked list is
100-->10-->20-->200-->30-->300-->NULL
Enter the option        4
The singly linked list is
10-->20-->200-->30-->300-->NULL
Enter the option        5
Enter the node value to be deleted      200
The singly linked list is
10-->20-->30-->300-->NULL
Enter the option        6
The singly linked list is
10-->20-->30-->NULL
Enter the option        7
Thank You
```

RESULT:

Thus the program for singly linked list implementation was executed and verified.

2. IMPLEMENTATION OF DOUBLY LINKED LIST

AIM:

To implement doubly linked list using C.

ALGORITHM:

1. Start the program.
2. Create the list by getting no. of nodes and nodes as an input.
3. Display the list.
4. Enter the choice
   a) If choice is 1, Insertion takes place at the first position.
   b) If choice is 2, Insertion takes place after the given node.
   c) If choice is 3, Insert an item into the list at last.
   d) If choice is 4, Delete an item at beginning.
   e) If choice is 5, Delete the given item from the list.
   f) If choice is 6, Delete at end.
   g) If choice is 7, then Exit.
5. Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int data;
 struct node *next;
 struct node *prev;
};
struct node *head=NULL,*p,*q,*t;
void display(void);
void insert_begin(void);
void insert_after(void);
void insert_end(void);
void del_begin(void);
void del(void);
void del_end(void);
int x;

int main()
{
 int ch,n,i;
 clrscr();
 printf ("Enter the number of nodes:\t");
```

```
scanf("%d",&n);
for(i=0;i<n;i++)
{
 p=malloc(sizeof(struct node));
 printf("Enter the data of node%d \t",i+1);
 scanf("%d",&p->data);
 p->next=NULL;
 p->prev=NULL;
 if(head==NULL)
 head=p;
 else
 {
  t->next=p;
  p->prev=t;
  }
 t=p;
 }
display();
printf("Menu");
printf("\n 1.Insert at beginning \n 2.Insert after the given node \n 3.Insert at end \n");
printf(" 4.Delete at beginning \n 5.Delete the given node \n 6.Delete at last \n 7.Exit
\n");
do
{
 printf("\nEnter the option \t");
 scanf("%d",&ch);
 t=head; switch(ch)
 {
   case 1:
      insert_begin();
      display();
      break;
   case 2:
      insert_after();
      display();
      break;
   case 3:
      insert_end();
      display();
      break;
   case 4:
      del_begin();
      display();
      break;
```

```
    case 5:
        del();
        display();
        break;
    case 6:
        del_end();
        display();
        break;
    case 7:
        printf("Thank You");
        getch();
    }
  }while (ch!=7);
 return 0;
 }

void insert_begin()
{
 p=malloc(sizeof(struct node)); printf("Enter the data to be inserted\t");
 scanf("%d",&p->data);
 p->next=head;
 p->prev=NULL;
 head=p;
}

void insert_after()
{
 printf("Enter the previous node value where the new node to be inserted \t");
 scanf("%d",&x);
 p=malloc(sizeof(struct node));
 printf("Enter the data to be inserted\t");
 scanf("%d",&p->data);
 t=head;
 while(x!=t->data)
    t=t->next;
 q=t->next;
 p->next=q;
 p->prev=t;
 t->next=p;
 if(q!=NULL)
    q->prev=p;
  }
```

```
void insert_end()
{
 p=malloc(sizeof(struct node)); printf("Enter the data to be inserted\t");
 scanf("%d",&p->data);
 p->prev=NULL;
 p->next=NULL;
 t=head;
 while(t->next!=NULL)
    t=t->next;
 p->prev=t;
 t->next=p;
 }

void del_begin()
{
 t=head;
 head=head->next;
 head->prev=NULL;
 t->next=NULL;
 free(t);
 }

void del()
{
 printf("Enter the node value to be deleted \t");
 scanf("%d",&x);
 t=head;
 while(x!=t->data)
 {
  p=t;
  t=t->next;
  }
 q=t->next;
 if(t==head)
    head=head->next;
 else if (t->next==NULL)
    p->next=NULL;
 else
 {
  p->next=q;
  q->prev=p;
  }
 free(t);
 }
```

```
void del_end()
{
 t=head;
 while(t->next!=NULL)
 {
  p=t;
  t=t->next;
  }
 if(t==head)
    head=NULL;
 else
 {
  p->next=NULL;
  t->prev=NULL;
  free(t);
  }
 }

void display ()
{
 printf("The doubly linked list is \n");
 t=head;
 printf("NULL<--");


while(t!=NULL)
 {
  printf("%d",t->data);
  t=t->next;
  if (t!=NULL)
    printf("<-->");
  }
 printf("-->NULL\n");
 }
```

SAMPLE OUTPUT:

```
Enter the number of nodes:    3
Enter the data of node1       10
Enter the data of node2       20
Enter the data of node3       30
The doubly linked list is
NULL<--10<-->20<-->30-->NULL
```

```
Menu
 1.Insert at beginning
 2.Insert after the given node
 3.Insert at end
 4.Delete at beginning
 5.Delete the given node
 6.Delete at last
 7.Exit

Enter the option        1
Enter the data to be inserted   100
The doubly linked list is
NULL<--100<-->10<-->20<-->30-->NULL

Enter the option        2
Enter the previous node value where the new node to be inserted     20
Enter the data to be inserted   200
The doubly linked list is
NULL<--100<-->10<-->20<-->200<-->30-->NULL

Enter the option        3
Enter the data to be inserted   300
The doubly linked list is
NULL<--100<-->10<-->20<-->200<-->30<-->300-->NULL

Enter the option        4
The doubly linked list is
NULL<--10<-->20<-->200<-->30<-->300-->NULL

Enter the option        5
Enter the node value to be deleted      200
The doubly linked list is
NULL<--10<-->20<-->30<-->300-->NULL

Enter the option        6
The doubly linked list is
NULL<--10<-->20<-->30-->NULL

Enter the option        7
Thank You
```

RESULT:

Thus the program for doubly linked list implementation was executed and verified.

3. IMPLEMENTATION OF CIRCULAR LINKED LIST

AIM:

To implement circular linked list using C.

ALGORITHM:

1. Start the program.
2. Create the list by getting no. of nodes and nodes as an input.
3. Display the list.
4. Enter the choice
    a) If choice is 1, Insertion takes place at the first position.
    b) If choice is 2, Insertion takes place after the given node.
    c) If choice is 3, Insert an item into the list at last.
    d) If choice is 4, Delete an item at beginning.
    e) If choice is 5, Delete the given item from the list.
    f) If choice is 6, Delete at end.
    g) If choice is 7, then Exit.
5. Stop the program.

PROGRAM:

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

struct node
{
 int data;
 struct node *next;
 };
struct node *head=NULL,*p,*q,*t,*last=NULL;

void display(void);
void insert_begin(void);
void insert_after(void);
void insert_end(void);
void del_begin(void);
void del(void);
void del_end(void);
int x;

int main()
{
 int ch,n,i;
 clrscr();
 printf ("Enter the number of nodes:\t");
```

```
scanf("%d",&n);
t=head;

for(i=0;i<n;i++)
{
 p=malloc(sizeof(struct node));
 printf("Enter the data of node%d \t",i+1);
 scanf("%d",&p->data);
 p->next=NULL;
 if(head==NULL)
 {
  head=p;
  head->next=head;
 }
 else
 {
  t->next=p;
  p->next=head;
  }
 t=p;
 }
last=t;
display();
printf("\nMenu");
printf("\n 1.Insert at beginning \n 2.Insert after the given node \n 3.Insert at end \n");
printf(" 4.Delete at beginning \n 5.Delete the given node \n 6.Delete at last \n 7.Exit
\n");
do
{
 printf("\nEnter the option \t");
 scanf("%d",&ch);
 t=head;
 switch(ch)
 {
   case 1:
      insert_begin();
      display();
      break;

   case 2:
      insert_after();
      display();
      break;
```

```
    case 3:
        insert_end();
        display();
        break;

    case 4:
        del_begin();
        display();
        break;

    case 5:
        del();
        display();
        break;

    case 6:
        del_end();
        display();
        break;

    case 7:
        printf("Thank You");
        getch();
    }
}while (ch!=7);
return 0;
}

void insert_begin()
{
 p=malloc(sizeof(struct node));
 printf("Enter the data to be inserted\t");
 scanf("%d",&p->data);
 p->next=head;
 last->next=p;
 head=p;
}

void insert_after()
{
 printf("Enter the previous node value where the new node to be inserted \t");
 scanf("%d",&x);
 p=malloc(sizeof(struct node));
 printf("Enter the data to be inserted\t");
```

```
 scanf("%d",&p->data);
 t=head;
 while(x!=t->data)
    t=t->next;
 if(t->next!=head)
 {
  q=t->next;
  p->next=q;
  t->next=p;
 }
 else
 {
  t->next=p;
  last=p;
  p->next=head;
 }
}

void insert_end()
{
 p=malloc(sizeof(struct node));
 printf("Enter the data to be inserted\t");
 scanf("%d",&p->data);
 p->next=head;
 t=head;
 while(t->next!=head)
    t=t->next;
 t->next=p;
 last=p;
}


void del_begin()
{
 t=head;
 head=head->next;
 t->next=NULL;
 free(t);
 last->next=head;
}
```

```
void del()
{
 printf("Enter the node value to be deleted \t");
 scanf("%d",&x);
 t=head;
 while(x!=t->data)
 {
  p=t;
  t=t->next;
  }
 q=t->next;

if(t==head)
{
 head=head->next;
 last->next=head;
 }
else if (t->next==head)
{
 p->next=head;
 last=p;
}
else
          p->next=q;
free(t);
}

void del_end()
{
 t=head;
 while(t->next!=head)
 {
  p=t;
  t=t->next;
  }
 if(t==head)
    head=NULL;
else
 {
  p->next=head;
  last=p;
  free(t);
  }
 }
```

```
void display ()
{
 printf("The circular linked list is \n");
 t=head;
 printf("-->");
 while(t!=last)
 {
  printf("%d-->",t->data);
  t=t->next;
  }
  printf("%d-->",t->data);
  }
```

SAMPLE OUTPUT:

```
Enter the number of nodes:    3
Enter the data of node1       10
Enter the data of node2       20
Enter the data of node3       30
The circular linked list is
-->10-->20-->30-->
Menu
 1.Insert at beginning
 2.Insert after the given node
 3.Insert at end
 4.Delete at beginning
 5.Delete the given node
 6.Delete at last
 7.Exit

Enter the option        1
Enter the data to be inserted   5
The circular linked list is
-->5-->10-->20-->30-->
Enter the option        2
Enter the previous node value where the new node to be inserted     10
Enter the data to be inserted   15
The circular linked list is
-->5-->10-->15-->20-->30-->
Enter the option        3
Enter the data to be inserted   35
The circular linked list is
-->5-->10-->15-->20-->30-->35-->
```

```
Enter the option        4
The circular linked list is
-->10-->15-->20-->30-->35-->
Enter the option        5
Enter the node value to be deleted      15
The circular linked list is
-->10-->20-->30-->35-->
Enter the option        6
The circular linked list is
-->10-->20-->30-->
Enter the option        7
Thank You
```

RESULT:

Thus the program for circular linked list implementation was executed and verified.

4. (a) IMPLEMENTATION OF STACK USING ARRAY

AIM:

To implement STACK using array.

ALGORITHM:

1. Start the program.
2. Read the size of array (n) and initialize top=-1.
3. Enter the choice
   a) If choice is 1, push operation by following steps
      - Increment top value by 1 and place the element.
      - If top = n, then display "stack is overflow"
   b) If choice is 2, pop operation by following steps.
      - Display the element which is deleted and decrement top value by 1
      - If top = -1, then display "stack is underflow"
   c) If choice is 3, display the stack elements.
   d) If choice is 4, exit.
4. Stop the program.

PROGRAM:

```
#include <stdio.h>
#define max 10
int stack[max], choice, n, top=-1, x, i;
void push(void);
void pop(void);
void display(void);

int main()
{
   clrscr();
   printf("\n Enter the size of STACK: ");
   scanf("%d",&n);
   printf("\n\t STACK IMPLEMENTATION USING ARRAY");
   printf("\n\t.....");
   printf("\n\t 1.PUSH \n\t 2.POP \n\t 3.DISPLAY \n\t 4.EXIT");
   do
   {
      printf("\n Enter the Choice:");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1:
            push();
            break;
```

```
            case 2:
                pop();
                break;

            case 3:
                display();
                break;

            case 4:
                printf("\n\t EXIT POINT - THANK YOU ");
                break;

            default:
                printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
        }
    }while(choice!=4);
 return 0;
 }

void push()
{
    if(top==n-1)
        printf("\n STACK is over flow \n");
    else
    {
        printf(" Enter a value to be pushed: ");
        scanf("%d",&x);
        top++;
        stack[top]=x;
     }
  }

void pop()
{
    if(top==-1)
        printf("\n\t Stack is under flow \n");
    else
    {
        printf("\n\t The popped elements is %d",stack[top]);
        top--;
     }
 }
```

```
void display()
{
   if(top>=0)
   {
      printf("\n The elements in STACK ");
      for(i=top; i>=0; i--)
         printf("\n%d",stack[i]);
    }
   else
      printf("\n The STACK is empty \n");
}
```

SAMPLE OUTPUT:

```
Enter the size of STACK: 3
        STACK IMPLEMENTATION USING ARRAY
          ----------------------------------
          1.PUSH
          2.POP
          3.DISPLAY
          4.EXIT
 Enter the Choice:1
 Enter a value to be pushed: 10
 Enter the Choice:1
 Enter a value to be pushed: 20
 Enter the Choice:1
 Enter a value to be pushed: 30
 Enter the Choice:3
 The elements in STACK
30
20
10
 Enter the Choice:1
 STACK is over flow

 Enter the Choice:2
 The popped elements is 30
 Enter the Choice:3
```

```
 The elements in STACK
20
10
 Enter the Choice:2
 The popped elements is 20
 Enter the Choice:3
 The elements in STACK
10
 Enter the Choice:2
 The popped elements is 10
 Enter the Choice:3
 The STACK is empty

 Enter the Choice:2
 Stack is under flow

 Enter the Choice:5
 Please Enter a Valid Choice(1/2/3/4)
 Enter the Choice:4
EXIT POINT - THANK YOU
```

RESULT:

Thus the program for stack implementation using array was executed and verified.

4. (b) IMPLEMENTATION OF STACK USING LINKED LIST

AIM:

To implement STACK using linked list.

ALGORITHM:

1. Start the program.
2. Initialize top=NULL.
3. Enter the choice
   a) If choice is 1, push operation by following steps
      - Create node(p) and read the data.
      - If top=NULL, then p->next=NULL.
      - Else p->next=top.
      - Then point the new node by top.

   b) If choice is 2, pop operation by following steps.
      - If top = NULL, then display "stack is underflow"
      - Else display the element ie print top->data and top=top->next

   c) If choice is 3, display the stack elements from top.
   d) If choice is 4, exit.
4. Stop the program.

PROGRAM:

```
// Implementation of stack using linked list
#include <stdio.h>
#include <stdlib.h>
struct Node
{
 int data;
 struct Node *next;
};
struct Node *top = NULL;

void push(int value)
{
 struct Node *p;
 p = malloc(sizeof(struct Node));
 p->data = value;
```

```
 if (top == NULL)
    p->next = NULL;
 else
    p->next = top;
 top = p;
 }

void pop()
{
 if (top == NULL)
    printf("\n Stack is Underflow \n");
 else
 {
  struct Node *t = top;
  printf("The Popped element is %d \n ",t->data);
  top = top->next;
  free(t);
  }
 }

void display()
{
 if (top == NULL)
    printf("\nStack Underflow\n");
 else
 {
  printf("The stack is \n");
  struct Node *t = top;
  while (t->next != NULL)
  {
   printf("%d--->", t->data);
   t = t->next;
   }
  printf("%d--->NULL\n", t->data);
  }
 }

int main()
{
 int choice, value;
 //clrscr();
 printf("\n Implementation of Stack using Linked List ");
 printf("\n ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ \n");
 printf(" 1. Push \n 2. Pop \n 3. Display \n 4. Exit \n");
```

```
while (1)
{
 printf("Enter your choice : ");
 scanf("%d",  &choice);
 switch (choice)
{
   case 1:
      printf("\nEnter the value to insert: ");
      scanf("%d", &value);
      push(value);
      break;
   case 2:
      pop();
      break;
   case 3:
      display();
      break;
   case 4:
      printf("\n Thank You \n");
      //getch();
      exit(0);
      break;
   default:
      printf("\n Wrong Choice \n");
   }
 }
}
```

SAMPLE OUTPUT:

```
Implementation of Stack using Linked List
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1. Push
2. Pop
3. Display
4. Exit
Enter your choice : 1
Enter the value to insert: 10
Enter your choice : 1
Enter the value to insert: 20
Enter your choice : 1
Enter the value to insert: 30
Enter your choice : 3
```

```
The stack is
30--->20--->10--->NULL
Enter your choice : 2
The Popped element is 30
 Enter your choice : 2
 The Popped element is 20
 Enter your choice : 2
 The Popped element is 10
 Enter your choice : 2
 Stack is Underflow
Enter your choice : 3
Stack Underflow
Enter your choice : 4
Thank You
```

RESULT:

Thus the program for stack implementation using linked list was executed and verified.

5. (a) IMPLEMENTATION OF QUEUE USING ARRAY

AIM:

To implement QUEUE using array.

ALGORITHM:

1. Start the program.
2. Read the size of array (n) and initialize front = -1 & rear = -1.
3. Enter the choice
   a) If choice is 1, insertion operation by following steps
      - If rear = n-1, then display "Queue is overflow"
      - Else increment rear value by 1 and place the element.
      - If front = -1, then assign front = 0

   b) If choice is 2, deletion operation by following steps.
      - Display the element (which is deleted) in a[front] and increment front value by 1
      - If front = -1 or front > rear, then display "Queue is underflow"

   c) If choice is 3, display the Queue elements.
   d) If choice is 4, exit.
4. Stop the program.

PROGRAM

```
// Implementation of Queue using array
#include <stdio.h>
#include <stdlib.h>
#define MAX 3

void insert();
void del();
void display();

int a[MAX], value, rear = - 1, front = - 1;

int main()
{
 int choice;
 clrscr();
 printf("MENU \n");
 printf("1.Insert an element to queue \n");
 printf("2.Delete element from queue \n");
 printf("3.Display all elements of queue \n");
```

```
 printf("4.Quit \n");
 while (1)
 {
  printf("\nEnter your choice : ");
  scanf("%d", &choice);
  switch(choice)
  {
    case 1:
        insert();
        break;
    case 2:
        del();
        break;
    case 3:
        display();
        break;
    case 4:
        printf("Thank You");
        getch();
        exit(1);
    default:
        printf("Wrong choice \n");
    }
  }
 }

void insert()
{
 if(rear == MAX - 1)
    printf("Queue Overflow \n");
 else
 {
  if(front== - 1)
        front = 0;
  printf("Enter the value to be inserted into queue : ");
  scanf("%d", &value);
  rear = rear + 1;
  a[rear] = value;
  }
 }

void del()
{
 if(front == - 1 || front > rear)
```

```
 {
  printf("Queue Underflow \n");
  front = rear = -1;
  return;
  }
 else
 {
  printf("Element deleted from queue is : %d \n", a[front]);
  front = front + 1;
  }
 }

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

SAMPLE OUTPUT:

```
Menu
1.Insert an element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit

Enter your choice :  1
Enter the value to be inserted into queue : 10
Enter your choice :  1
Enter the value to be inserted into queue : 20
Enter your choice :  1
Enter the value to be inserted into queue : 30
Enter your choice :  3
Queue is :
10        20        30
```

```
Enter your choice :  2
Element deleted from queue is : 10

Enter your choice :  2
Element deleted from queue is : 20

Enter your choice :  2
Element deleted from queue is : 30

Enter your choice : 3
Queue is empty

Enter your choice :  1
Enter the value to be inserted into queue : 50
Enter your choice :  1
Enter the value to be inserted into queue : 60
Enter your choice :  3
Queue is :
50      60

Enter your choice : 4
Thank You
```

RESULT:

Thus the program for QUEUE implementation using array was executed and verified.

5. (b) IMPLEMENTATION OF QUEUE USING LINKED LIST

AIM:

To implement QUEUE using linked list.

ALGORITHM:

1. Start the program.
2. Initialize front=NULL and rear=NULL.
3. Enter the choice
   a) If choice is 1, insertion operation by following steps
      - Create node(p)and read the data & assign p->next=NULL..
      - Else assign rear->next=p and rear=p.
      - Then point the new node „p‟ by rear.
   b) If choice is 2, deletion operation by following steps.
      - If front = NULL, then display “Queue is underflow”
      - Else display the element in front->data and assign front=front->next
   c) If choice is 3, display the queue elements from front to rear.
   d) If choice is 4, exit.
4. Stop the program.

PROGRAM

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

struct Node
{
  int data;
  struct Node *next;
}*front = NULL,*rear = NULL;

void insert(int);
void delete();
void display();

void main()
{
  int choice, value;
  clrscr();
  printf("\n:: Queue Implementation using Linked List ::\n");
  printf("\n****** MENU ******\n");
  printf(" 1. Insert\n 2. Delete\n 3. Display\n 4. Exit\n");
```

```
  while(1)
  {
    printf("\n Enter your choice: ");
    scanf("%d",&choice);
    switch(choice)
    {
          case 1:
                printf("Enter the value to be insert: ");
                scanf("%d", &value);
                insert(value);
                break;
          case 2:
                delete();
                break;
          case 3:
                display();
                break;
          case 4:
                printf("\nThank you\n");
                exit(0);
          default:
                printf("\n Wrong selection!!! Please Enter Correct Choice!!!\n");
    }
  }
}

void insert(int value)
{
  struct Node *p;
  p = malloc(sizeof(struct Node));
  p-> data = value;
  p-> next = NULL;
  if(front == NULL)
    front = rear = p;
  else
  {
    rear -> next = p;
    rear = p;
   }
  printf("\n Insertion is Success!!!\n");
}
```

```
void delete()
{
  if(front == NULL)
    printf("\n Queue is Underflow!!!\n");
  else
  {
    struct Node *temp = front;
    front = front -> next;
    printf("\n Deleted element: %d\n", temp->data);
    free(temp);
   }
}

void display()
{
  if(front == NULL)
    printf("\n Queue is Empty!!!\n");
  else
  {
   struct Node *temp = front;
   while(temp->next != NULL)
   {
        printf("%d--->",temp->data);
           temp = temp -> next;
    }
    printf("%d--->NULL\n",temp->data);
  }
}
```

SAMPLE OUTPUT

```
:: Queue Implementation using Linked List ::

****** MENU ******
 1. Insert
 2. Delete
 3. Display
 4. Exit
 Enter your choice: 1
 Enter the value to be insert: 10
 Insertion is Success!!!

 Enter your choice: 1
 Enter the value to be insert: 20
 Insertion is Success!!!
```

```
Enter your choice: 1
Enter the value to be insert: 30
Insertion is Success!!!

Enter your choice: 3
10--->20--->30--->NULL

Enter your choice: 2
Deleted element: 10

Enter your choice: 3
20--->30--->NULL

Enter your choice: 2
Deleted element: 20

Enter your choice: 2
Deleted element: 30

Enter your choice: 3
Queue is Empty!!!

Enter your choice: 2
Queue is Underflow!!!

Enter your choice: 5
Wrong selection!!! Please Enter Correct Choice!!!

Enter your choice: 4
Thank you
```

RESULT:

Thus the program for QUEUE implementation using linked list was executed and verified.

6.(a) IMPLEMENTATION OF BUBBLE SORT

AIM :

To implement Bubble Sorting technique to sort a given list of integers in ascending order.

DESCRIPTION:

Bubble sort is a sorting algorithm that compares two adjacent elements and swaps them until they are in the intended order.

Just like the movement of air bubbles in the water that rise up to the surface, each element of the array move to the end in each iteration. Therefore, it is called a bubble sort.

ALGORITHM :

1. Starting from the first index, compare the first and the second elements.
2. If the first element is greater than the second element, they are swapped.
3. Now, compare the second and the third elements. Swap them if they are not in order.
4. The above process goes on until the last element in the unsorted list..
5. After each iteration, the largest element among the unsorted elements is placed at the end.
6. In each iteration, the comparison takes place up to the last unsorted element.
7. The array is sorted after completion of (n-1) iterations for „n‟ elements.

PROGRAM

```
#include  <stdio.h>
#include <conio.h>
#define MAX 10

void printArray(int a[], int n)
{
  printf("[");
  for(int i = 0; i < n; i++)
     printf(" %d ",a[i]);
  printf("]\n");
 }

void main()
{
  int a[MAX], n, i, j, temp;
  clrscr();
  printf("Enter the total number of elements in an array : \t ");
```

```
   scanf("%d",&n);
   printf("Enter %d elements one by one \n",n);
   for(i = 0; i < n; i++)
        scanf("%d",&a[i]);

   printf("\n Input Array : ");
   printArray(a,n);
   for(i = 0; i < n-1; i++)
   {
      printf("\n Iteration %d: \n\n",(i+1));
      for(j = 0; j < n-1-i; j++)
      {
        printf("\t\t Items compared: [%d, %d] ", a[j],a[j+1]);
        if(a[j] > a[j+1])
        {
          temp = a[j];
          a[j] = a[j+1];
          a[j+1] = temp;
          printf(" => swapped [%d, %d]\n",a[j],a[j+1]);
         }
       else
          printf(" => not swapped\n");
      }
    printf("\n\t Array after this iteration: ");
    printArray(a,n);
    }
   printf("\nOutput Array: ");
   printArray(a,n);
   getch();
}
```

SAMPLE OUTPUT

```
Enter the total number of elements in an array :        5
Enter 5 elements one by one
16
2
4
10
3
Input Array : [ 16 2 4 10 3 ]
```

Iteration 1:

Items compared: [16, 2] => swapped [2, 16]
Items compared: [16, 4] => swapped [4, 16]
Items compared: [16, 10] => swapped [10, 16]
Items compared: [16, 3] => swapped [3, 16]

Array after this iteration: [2 4 10 3 16]

Iteration 2:

Items compared: [2, 4] => not swapped
Items compared: [4, 10] => not swapped
Items compared: [10, 3] => swapped [3, 10]

Array after this iteration: [2 4 3 10 16]

Iteration 3:

Items compared: [2, 4] => not swapped
Items compared: [4, 3] => swapped [3, 4]

Array after this iteration: [2 3 4 10 16]

Iteration 4:

Items compared: [2, 3] => not swapped

Array after this iteration: [2 3 4 10 16]

Output Array: [2 3 4 10 16]

RESULT:

Thus the program for Bubble Sort was executed and verified successfully.

6.(b) IMPLEMENTATION OF SELECTION SORT

AIM :

To implement Selection Sort technique to sort a given list of integers in ascending order.

DESCRIPTION:

Selection sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.

ALGORITHM :

1. Set the first element as minimum.
2. Compare minimum with the second element. If the second element is smaller than minimum, assign the second element as minimum.
3. Compare minimum with the third element. Again, if the third element is smaller, then assign minimum to the third element otherwise do nothing. The process goes on until the last element.
4. After each iteration, minimum is placed in the front of the unsorted list.
5. For each iteration, indexing starts from the first unsorted element.
6. Step 1 to 3 are repeated until all the elements are placed at their correct positions.

PROGRAM

```
// Implementation of selection sort
#include <stdio.h>
#include <conio.h>
void printArray(int arr[], int size)
{
  printf("[ ");
  for(int i = 0; i < size; i++)
        printf("%d ",arr[i]);
  printf("]\n");
 }

int main()
{
   int a[20], i, j, n, min, t;
   clrscr();
   printf("Enter the total elements in the array \t");
   scanf("%d",&n);
```

```
 for(i=0;i<n;i++)
 {
  printf("Enter the element%d \t: ",i+1);
  scanf("%d",&a[i]);
 }
printf("\n Input Array : ");
printArray(a,n);

for (i = 0; i < n-1; i++)
{
    printf("\n Iteration %d: \n",(i+1));
    min = i;
    for (j = i+1; j < n; j++)
       if (a[j] < a[min])
          min = j;
    if(i!=min)
    {
       t = a[min];
       a[min] = a[i];
       a[i] = t;
       printf("\t Items compared: [%d, %d]  & swapped ",a[min],a[i]);
     }
    else
       printf("\t Not swapped ");
 }

printf("\n\n Output Array: ");
printArray(a,n);
getch();
return 0;
}
```

SAMPLE OUTPUT

```
Enter the total elements in the array  5
Enter the element1      : 45
Enter the element2      : 65
Enter the element3      : 1
Enter the element4      : 24
Enter the element5      : 15

 Input Array : [ 45 65 1 24 15 ]

 Iteration 1:
        Items compared: [45, 1]  & swapped
 Iteration 2:
        Items compared: [65, 15]  & swapped
 Iteration 3:
        Items compared: [45, 24]  & swapped
 Iteration 4:
        Not swapped

 Output Array: [ 1 15 24 45 65 ]
```

RESULT:

Thus the program for Selection Sort was executed and verified successfully.

6.(c) IMPLEMENTATION OF INSERTION SORT

AIM :

To implement Insertion Sorting method to sort a given list of integers in ascending order.

DESCRIPTION:

- Insertion sort is a sorting algorithm that places an unsorted element at its suitable place in each iteration.
- Insertion sort works similarly as we sort cards in our hand in a card game.
- We assume that the first card is already sorted then, we select an unsorted card. If the unsorted card is greater than the card in hand, it is placed on the right otherwise, to the left. In the same way, other unsorted cards are taken and put in their right place.

ALGORITHM :

1. The first element in the array is assumed to be sorted. Take the second element and store it separately in key.
2. Compare key with the first element. If the first element is greater than key, then key is placed in front of the first element.
3. Now, the first two elements are sorted.
4. Take the third element and compare it with the elements on the left of it. Placed it just behind the element smaller than it. If there is no element smaller than it, then place it at the beginning of the array.
5. Similarly, place every unsorted element at its correct position.

PROGRAM

```
#include <stdio.h>
#include <conio.h>
void printArray(int arr[], int size)
{
  printf("[ ");
  for(int i = 0; i < size; i++)
        printf("%d ",arr[i]);
  printf("]\n");
 }
int main()
{
   int a[20], i, j, n, NewElement, flag;
   clrscr();
   printf("Enter the total number of elements in an array : \t");
   scanf("%d",&n);
```

```
    for(i = 0 ; i < n ; i++)
    {
     printf("Enter the element%d \t: ", i+1);
     scanf("%d",&a[i]);
     }
    printf("\n\n Input Array : ");
    printArray(a,n);

    for (i = 1; i < n; i++)
    {
       NewElement = a[i];
       j = i - 1;
       flag=0;
       while (j >= 0 && a[j] > NewElement)
       {
          a[j + 1] = a[j];
          j = j - 1;
          flag=1;
        }
       a[j + 1] = NewElement;
       printf("\n Iteration %d : \n",i);
       if(flag == 1)
          printf("\t %d is taken & inserted \n", NewElement);
       else
          printf("\t %d is taken & kept at same position \n", NewElement);
       printf("\t\t Array after the iteration : ");
       printArray(a,n);
     }

   printf("\n Output Array: ");
   printArray(a,n);
   getch();
   return 0;
}
```

SAMPLE OUTPUT

```
Enter the total number of elements in an array :        5
Enter the element1      14
Enter the element2      10
Enter the element3      5
Enter the element4      30
Enter the element5      24


Input Array : [ 14 10 5 30 24 ]

Iteration 1 :
        10 is taken & inserted
                Array after the iteration : [ 10 14 5 30 24 ]

Iteration 2 :
        5 is taken & inserted
                Array after the iteration : [ 5 10 14 30 24 ]

Iteration 3 :
        30 is taken & kept at same position
                Array after the iteration : [ 5 10 14 30 24 ]

Iteration 4 :
        24 is taken & inserted
                Array after the iteration : [ 5 10 14 24 30 ]

Output Array: [ 5 10 14 24 30 ]
```

RESULT:

Thus the program for Insertion Sort was executed and verified successfully.

7.(a) IMPLEMENTATION OF LINEAR SEARCH

AIM :

To implement linear search using non-recursive and recursive function.

DESCRIPTION :

- Linear search is a very basic and simple search algorithm. In Linear search, we search an element or value in a given array by traversing the array from the starting, till the desired element or value is found.
- This procedure is also applicable for unsorted data set. Linear search is also known as sequential search. It is named as linear because its time complexity is O(n).

ALGORITHM :

1. Read the array.
2. Traverse the array using a for loop.
3. In every iteration, compare the target value with the current value of the array.
    - If the values match, return the position.
    - If the values do not match, move on to the next array element.
4. If no match is found, return -1.

PROGRAM

```
#include <stdio.h>
void printResult(int);
int LinearSearch(int array[], int n, int x)
{
  for (int i = 0; i < n; i++)
    if (array[i] == x)
    return i;
  return -1;
 }

int RecursiveLSearch(int arr[], int value, int index, int n)
{
   if(index >= n)
      return 0;
   else if (arr[index] == value)
      return index;
   else
      return RecursiveLSearch(arr, value, index+1, n);
```

```
    return -1;
 }

int main()
{
    int n, value, array[30];
    printf("Enter the total elements in the array ");
    scanf("%d", &n);

    printf("Enter the array elements \n");
    for (int i = 0; i < n; i++)
        scanf("%d", &array[i]);

    printf("Enter the element to search : ");
    scanf("%d", &value);

    int result = LinearSearch(array, n, value);
    printf("\n\n Result : Linear Search using Non-recursive function \n");
    printResult(result);

    int pos = RecursiveLSearch(array, value, 0, n);
    printf("\n\n Result : Linear Search using Recursive function \n");
    printResult(pos);

    return 0;
   }




void printResult(int X)
{
    if (X != -1)
        printf("\t Element found at position %d ", X+1);
    else
        printf("\t Element not found");
 }
```

SAMPLE OUTPUT

```
Enter the total elements in the array 6
Enter the array elements
35
48
2
14
68
42
Enter the element to search : 2
Result : Linear Search using Non-recursive function
        Element found at position 3

 Result : Linear Search using Recursive function
        Element found at position 3
```

RESULT:

Thus the program for Linear Search using non-recursive and recursive was executed and also verified successfully.

7.(b) IMPLEMENTATION OF BINARY SEARCH

AIM :

To implement binary search using non-recursive and recursive function.

DESCRIPTION :

A binary search is usually a search algorithm that helps in finding the exact position of the element using a short line of codes. It is also known as a half interval search, as we select or eliminate the half.

ALGORITHM :

1. Input the array and key value to be found.
2. Sort the array by using any one of the sorting technique.
3. Find the middle value.
4. If array[middle] = key value, print "value found".
5. If array[middle] < element, search in the right side of middle.
   else search in the left side of middle.
6. Repeat the above procedure until start>last.

PROGRAM

```
#include<stdio.h>
void printResult(int);
int BinarySearch (int array[], int start, int last, int element)
{
   while (start <= last)
   {
      int middle = (last + start)/2;
      if (array[middle] == element)
         return middle;
      if (array[middle] < element)
         start = middle + 1;
      else
         last = middle - 1;
    }
   return -1;
 }

int RecursiveBSearch (int array[], int start, int last, int element)
{
   while (start <= last)
   {
      int middle = (last + start)/2;
```

```
        if (array[middle] == element)
            return middle;
        if (array[middle] > element)
                return RecursiveBSearch(array, start, middle-1, element);
        return RecursiveBSearch(array, middle+1, last, element);
     }
    return -1;
 }
int main(void)
{
    int a[30], n, element, temp, option;
    printf("Enter the total number of elements in the array : ");
    scanf("%d",&n);
    printf("Enter the elements one by one \n");
    for(int i=0; i<n; i++)
       scanf("%d",&a[i]);
    printf("Enter the element to be searched : ");
    scanf("%d",&element);
    for(int i = 0; i < n-1; i++)
    {
      for(int j = 0; j < n-1-i; j++)
      {
       if(a[j] > a[j+1])
       {
          temp = a[j];
          a[j] = a[j+1];
          a[j+1] = temp;
         }
       }
     }
    printf("The sorted Array is \n");
    for(int i = 0; i < n; i++)
      printf(" %d ",a[i]);

    int ans = BinarySearch(a, 0, n-1, element);
    printf("\n\n Result : Binary Search using Non-recursive function \n");
    printResult(ans);

    int result = RecursiveBSearch(a, 0, n-1, element);
    printf("\n Result : Binary Search using Recursive function \n");
    printResult(result);

    return 0;
    }
```

```
void printResult(int X)
{
   if(X == -1 )
      printf("\t Element not found in this array ");
   else
      printf("\t Element found at position %d \n",X+1);
 }
```

SAMPLE OUTPUT

```
Enter the total number of elements in the array : 6
Enter the elements one by one
85
95
55
16
3
42
Enter the element to be searched : 16
The sorted Array is
 3  5  16  42  55  85

 Result : Binary Search using Non-recursive function
   Element found at position 3

 Result : Binary Search using Recursive function
   Element found at position 3
```

RESULT:

Thus the program for Binary Search using recursive and non-recursive was executed and also verified successfully.

8. IMPLEMENTATION OF TREE TRAVERSAL

AIM :

To implement tree traversal in binary search tree.

ALGORITHM :

1. Start the program.
2. Create a binary search tree for the given number of nodes.
3. Read the traversal option.
4. For in-order, follow the following steps.
   i. Traverse left subtree
   ii. Visit the root and print the data.
   iii. Traverse the right subtree
5. For pre-order, follow the following steps.
   i. Visit the root and print the data.
   ii. Traverse left subtree
   iii. Traverse the right subtree
6. For post-order, follow the following steps.
   i. Traverse left subtree
   ii. Traverse the right subtree
   iii. Visit the root and print the data.
7. Stop the program.

PROGRAM

```
#include  <stdio.h>
#include <conio.h>
struct node
{
  int key;
  struct node* left;
  struct node* right;
};
// Create a node
struct node* newNode(int item)
{
  struct node *temp = malloc(sizeof(struct node));
  temp->key = item;
  temp->left = temp->right = NULL;
  return temp;
}
```

```
// Inorder Traversal
void inorder(struct node* root)
{
  if (root != NULL)
  {
    inorder(root->left);
    printf("%d ", root->key);
    inorder(root->right);
  }
}

// Postorder Traversal
void postorder(struct node* root)
{
  if (root != NULL)
  {
    postorder(root->left);
    postorder(root->right);
    printf("%d  ", root->key);
  }
}

// Preorder Traversal
void preorder(struct node* root)
{
  if (root != NULL)
  {
    printf("%d ", root->key);
    preorder(root->left);
    preorder(root->right);
  }
}

// Insert a node
struct node* insert(struct node* node, int key)
{
  if  (node == NULL)
        return newNode(key);
  if (key < node->key)
        node->left = insert(node->left, key);
  else
        node->right = insert(node->right, key);
  return node;
}
```

```
int main()
{
 int ch,data,n,i;
 struct node *root = NULL;
 clrscr();
 printf("\n 1.Create \n 2.In-order \n 3.Pre-order \n 4.Post-order \n 5.Exit ");
 do
 {
  printf("\n\n Enter the choice \t");
  scanf("%d",&ch);
  switch(ch)
  {
   case 1:
         printf("\n Enter the no. of nodes to be inserted : ");
         scanf("%d",&n);
         for(i=0;i<n;i++)
         {
          printf("Enter the data %d : ",i+1);
          scanf("%d",&data);
          root = insert(root,data);
          }
         break;

case 2:
         printf("Inorder traversal: ");
         inorder(root);
         break;
   case 3:
         printf("Preorder traversal: ");
         preorder(root);
         break;
   case 4:
         printf("Postorder traversal: ");
         postorder(root);
         break;
   case 5:
         printf("End of Program - Thank you");
         getch();
         exit(0);
   default:
         printf("WRONG CHOICE : Enter correct choice ");
   }
 }while(1);
}
```

SAMPLE OUTPUT

```
1.Create
2.In-order
3.Pre-order
4.Post-order
5.Exit

Enter the choice        1
Enter the no. of nodes to be inserted : 8
Enter the data 1    30
Enter the data 2    20
Enter the data 3    10
Enter the data 4    5
Enter the data 5    35
Enter the data 6    40
Enter the data 7    45
Enter the data 8    15

Enter the choice        2
Inorder traversal: 5  10  15  20  30  35  40  45

Enter the choice        3
Preorder traversal: 30  20  10  5  15  35  40  45

Enter the choice        4
Postorder traversal: 5  15  10 20  45  40  35  30

Enter the choice        5
End of Program - Thank you
```

RESULT:

Thus the program for Tree Traversal was executed and verified.

9. IMPLEMENTATION OF GRAPH TRAVERSAL

AIM :

Write a program to implement the graph traversal methods.

ALGORITHM :

BFS (Breadth First Search)

1. Define a Queue of size total number of vertices in the graph.
2. Select any vertex as starting point for traversal. Visit that vertex and insert it into the Queue.
3. Visit all the non-visited adjacent vertices of the vertex which is at front of the Queue and insert them into the Queue.
4. When there is no new vertex to be visited from the vertex which is at front of the Queue then delete that vertex.
5. Repeat steps 3 and 4 until queue becomes empty.
6. When queue becomes empty, then produce final spanning tree by removing unused edges from the graph

DFS (Depth First Search)

1. Define a Stack of size total number of vertices in the graph.
2. Select any vertex as starting point for traversal. Visit that vertex and push it on to the Stack.
3. Visit any one of the non-visited adjacent vertices of a vertex which is at the top of stack and push it on to the stack.
4. Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack.
5. When there is no new vertex to visit then use back tracking and pop one vertex from the stack.
6. Repeat steps 3, 4 and 5 until stack becomes Empty.
7. When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph

PROGRAM

```
#include<stdio.h>
int top=-1,q[20],stack[20],front=-1,rear=-1,arr[20][20],visited[20];

main()
{
 int i,j,n,ch,s;
 printf("Enter the Number of Vertices \n");
 scanf("%d",&n);
```

```
 for(i=1;i<=n;i++)
 {
   for(j=1;j<=n;j++)
   {
    printf("Enter 1 if Vertex %d has a edge with Vertex %d, Else 0 \t",i,j);
    scanf("%d",&arr[i][j]);
   }
 }
 printf("Enter the stating vertex \t");
 scanf("%d",&s);
 do
 {
  for(i=0;i<=n;i++)
      visited[i]=0;
  printf("\n Enter the Option   (1. BFS   2. DFS    3. Exit \n");
  scanf("%d",&ch);
  switch(ch)
  {
   case 1:
      bfs(s,n);
      break;
   case 2:
      dfs(s,n);
      break;
   case 3:
      printf("End of Program - Thank you");
      getch();
   }
  } while(ch!=3);
}

void add(int item)
{
   if(rear==-1)
      front++;
   q[++rear]=item;
   }

int Delete()
{
   int k;
   if ((front>rear)||(front==-1))
      return (0);
```

```
    else
        {
         k=q[front++];
           return(k);
        }
 }

void push( int item )
{
   stack[ ++top ] = item;
}

int pop()
{
   int k;
   if ( top == -1 )
      return ( 0 );
   else
        {
           k = stack[ top-- ];
           return ( k );
        }
 }

bfs(int s,int n)
{
 int i,p;
 add(s);
 visited[s]=1;
 p=Delete();
 if(p!=0)
          printf("%d ",p);
 while(p!=0)
 {
  for(i=1;i<=n;i++)
  {
   if((arr[p][i]!=0)&&(visited[i]==0))
   {
    add(i);
    visited[i]=1;
    }
  }
 p=Delete();
```

```
 if(p!=0)
       printf("%d ",p);
 }
 for(i=1;i<=n;i++)
 {
  if(visited[i]==0)
       bfs(i,n);
  }
 }

dfs(int s,int n)
{
 int k,i;
 push(s);
 visited[s]=1;
 printf("%d ",s);
 k=pop();
 while(k!=0)
 {
  for(i=1;i<=n;i++)
  {
   if((arr[k][i]!=0)&&(visited[i]==0))
   {
    push(i);
    visited[i]=1;
    printf("%d ",i);
    }
   k=pop();
   }
  }
 for(i=1;i<=n;i++)
 {
  if(visited[i]==0)
          dfs(i,n);
  }
 }
```

SAMPLE OUTPUT

```
Enter the Number of Vertices
5
Enter 1 if Vertex 1 has a edge with Vertex 1, Else 0  0
Enter 1 if Vertex 1 has a edge with Vertex 2, Else 0  1
Enter 1 if Vertex 1 has a edge with Vertex 3, Else 0  1
Enter 1 if Vertex 1 has a edge with Vertex 4, Else 0  0
Enter 1 if Vertex 1 has a edge with Vertex 5, Else 0  0
Enter 1 if Vertex 2 has a edge with Vertex 1, Else 0  1
Enter 1 if Vertex 2 has a edge with Vertex 2, Else 0  0
Enter 1 if Vertex 2 has a edge with Vertex 3, Else 0  0
Enter 1 if Vertex 2 has a edge with Vertex 4, Else 0  1
Enter 1 if Vertex 2 has a edge with Vertex 5, Else 0  1
Enter 1 if Vertex 3 has a edge with Vertex 1, Else 0  1
Enter 1 if Vertex 3 has a edge with Vertex 2, Else 0  0
Enter 1 if Vertex 3 has a edge with Vertex 3, Else 0  0
Enter 1 if Vertex 3 has a edge with Vertex 4, Else 0  0
Enter 1 if Vertex 3 has a edge with Vertex 5, Else 0  0
Enter 1 if Vertex 4 has a edge with Vertex 1, Else 0  0
Enter 1 if Vertex 4 has a edge with Vertex 2, Else 0  1
Enter 1 if Vertex 4 has a edge with Vertex 3, Else 0  0
Enter 1 if Vertex 4 has a edge with Vertex 4, Else 0  0
Enter 1 if Vertex 4 has a edge with Vertex 5, Else 0  0
Enter 1 if Vertex 5 has a edge with Vertex 1, Else 0  0
Enter 1 if Vertex 5 has a edge with Vertex 2, Else 0  1
Enter 1 if Vertex 5 has a edge with Vertex 3, Else 0  0
Enter 1 if Vertex 5 has a edge with Vertex 4, Else 0  0
Enter 1 if Vertex 5 has a edge with Vertex 5, Else 0  0
Enter the stating vertex          1
Enter the Option   (1. BFS   2. DFS    3. Exit
1
1 2 3 4 5
Enter the Option   (1. BFS   2. DFS    3. Exit
2
1 2 3 4 5
Enter the Option   (1. BFS   2. DFS    3. Exit
3
End of Program - Thank you
```

RESULT:

Thus the program for Graph Traversal was executed and verified.