Experiment 1

AIM: To Implement Stack ADT using array.

PROGRAM:

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

int i,n,x,top,choice,stack[100];

void push(void);
void pop(void);
void display(void);

int main()
{
 clrscr();
 top=-1;
 printf("Enter the size of STACK [MAX = 100] : ");
 scanf("%d",&n);
 printf("STACK OPERATIONS USING ARRAY \n");
 printf("1.PUSH \n2.POP \n3.DISPLAY \n4.EXIT \n");
 do
 {
  printf("Enter the Choice : ");
  scanf("%d",&choice);
  switch(choice)
  {
   case 1: push();
           break;
   case 2: pop();
           break;
   case 3: display();
           break;
   case 4: printf("\nEXIT POINT \n");
           break;
   default: printf("\nPlease Enter a Valid Choice : (1/2/3/4)");
  }
 }
 while(choice != 4);
 getch();
 return 0;
}

void push()
{
 if(top >= n - 1)
 {
```

```
  printf("\nSTACK is Overflow\n");
 }
 else
 {
  printf("Enter a value to be pushed : ");
  scanf("%d",&x);
  top++;
  stack[top] = x;
 }
}

void pop()
{
 if(top <= -1)
 {
  printf("\nSTACK is Underflow\n");
 }
 else
 {
  printf("\nThe popped element is : %d\n",stack[top]);
  top--;
 }
}

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

OUTPUT:

```
Enter the size of STACK [MAX = 100] : 7
STACK OPERATIONS USING ARRAY
1.PUSH
2.POP
3.DISPLAY
4.EXIT
Enter the Choice : 1
Enter a value to be pushed : 32
Enter the Choice : 1
Enter a value to be pushed : 64
Enter the Choice : 1
Enter a value to be pushed : 128
Enter the Choice : 3

The elements in STACK are :
128
64
32

Press Next Choice
Enter the Choice : 2

The popped element is : 128
Enter the Choice : 4S
```

Experiment 2

AIM: To Convert an Infix Expression to Postfix Expression using Stack ADT.

PROGRAM:

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

int top = -1;
char stack[100];

void push(char);
char pop();
int priority(char);

int main()
{
 char *e,x,exp[100];
 clrscr();
 printf("Enter the expression : ");
 scanf("%s",exp);
 printf("Postfix Expression is : \n\t\t");
 e = exp;
 while(*e != '\0')
 {
  if(isalnum(*e))
  {
   printf("%c ",*e);
  }
  else if(*e == '(')
  {
   push(*e);
  }
  else if(*e == ')')
  {
   while((x = pop()) != '(')
   {
    printf("%c ", x);
   }
  }
  else
  {
   while(priority(stack[top]) >= priority(*e))
   {
    printf("%c ",pop());
   }
   push(*e);
```

```
  }
  e++;
 }
 while(top != -1)
 {
  printf("%c ",pop());
 }
 getch();
 return 0;
}

void push(char x)
{
 stack[++top] = x;
}

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

int priority(char x)
{
 if(x == '(') return 0;
 if(x == '+' || x == '-') return 1;
 if(x == '*' || x == '/') return 2;
 return 0;
}
```

OUTPUT:


Enter the expression : ((A-B)+C*(D/E)+F/(G*H/I))
Postfix Expression is :
A B - C D E / * + F G H * I / / +

Experiment 3

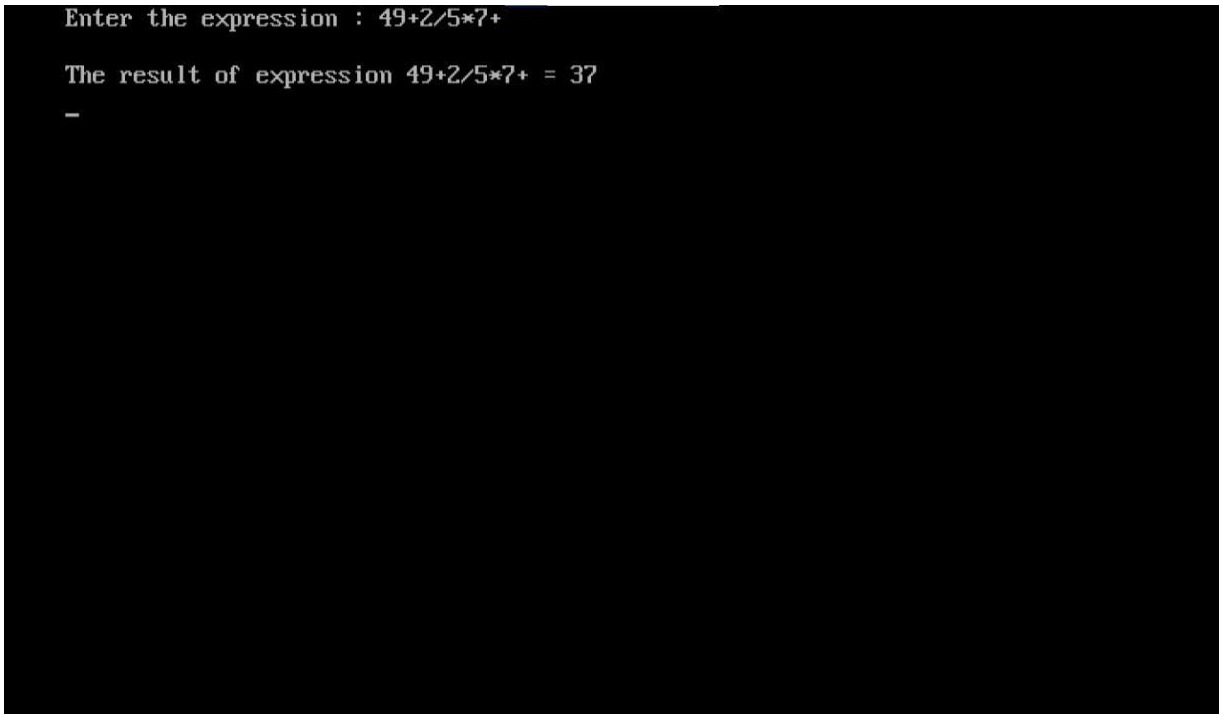
AIM: To Evaluate Postfix Expression using Stack ADT.

PROGRAM:

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

int top = -1,stack[20];

void push(int);
int pop();

int main()
{
 char *e,exp[20];
 int n1,n2,n3,num;
 clrscr();
 printf("Enter the expression : ");
 scanf("%s",&exp);
 e  = exp;
 while(*e != '\0')
 {
  if(isdigit(*e))
  {
   num = *e - 48;
   push(num);
  }
  else
  {
   n1 = pop();
   n2 = pop();
   switch(*e)
   {
    case '+': n3 = n1 + n2;
              break;
    case '-': n3 = n2 - n1;
              break;
    case '*': n3 = n1 * n2;
              break;
    case '/': n3 = n2 / n1;
              break;
   }
   push(n3);
  }
  e++;
 }
 printf("\nThe result of expression %s = %d\n",exp,pop());
```

```
 getch();
 return 0;
}

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

int pop()
{
 return stack[top--];
}
```

OUTPUT:



Experiment 4

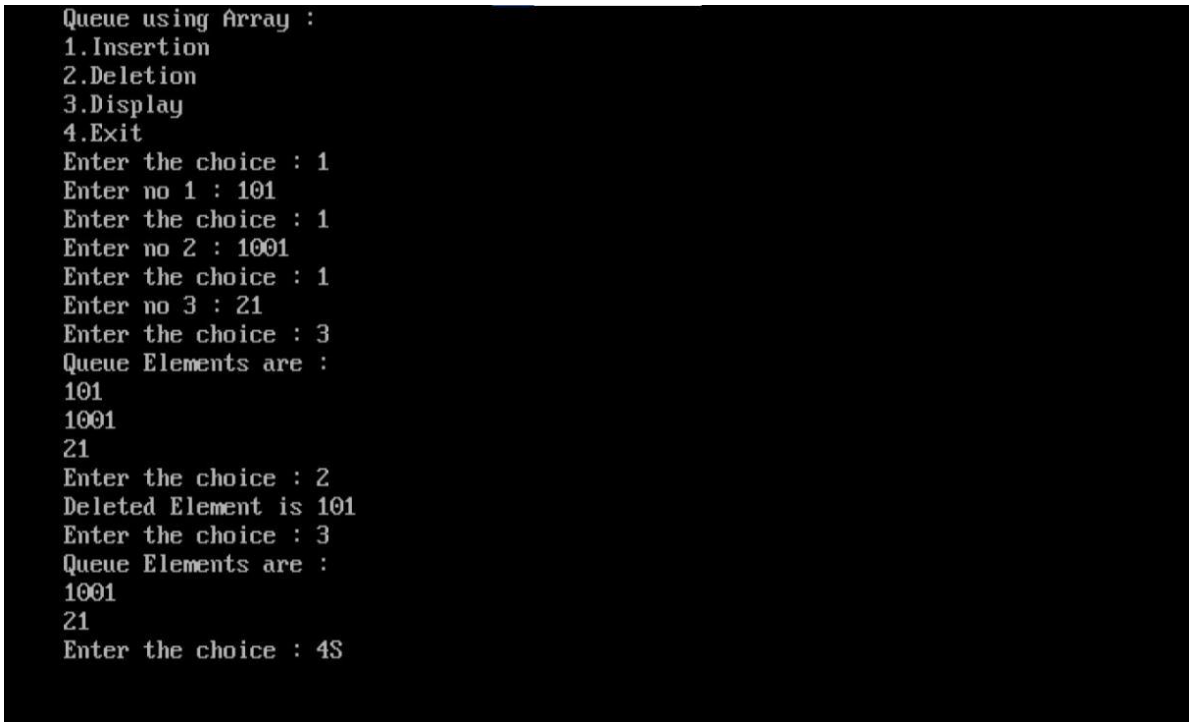
AIM: To Implement Linear Queue ADT using array.

PROGRAM:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define n 5

int main()
{
 int ch = 1,front = 0,rear = 0,i,j = 1,x = n,queue[n];
 clrscr();
 printf("Queue using Array : \n");
 printf("1.Insertion \n2.Deletion \n3.Display \n4.Exit \n");
 while(ch)
 {
  printf("Enter the choice : ");
  scanf("%d",&ch);
  switch(ch)
  {
   case 1: if(rear==x)
           {
            printf("Queue is Full\n");
           }
           else
           {
            printf("Enter no %d : ",j++);
            scanf("%d",&queue[rear++]);
           }
           break;
   case 2: if(front==rear)
           {
            printf("Queue is Empty\n");
           }
           else
           {
            printf("Deleted Element is %d\n",queue[front++]);
            x++;
           }
           break;
   case 3: printf("Queue Elements are : \n");
           if(front==rear)
           {
            printf("Queue is Empty\n");
           }
           else
```

```
            {
             for(i = front;i < rear;i++)
             {
              printf("%d",queue[i]);
              printf("\n");
             }
            }
            break;
   case 4: exit(0);
   default: printf("\nWrong Choice : Please see the options\n");
  }
 }
 getch();
 return 0;
}
```

OUTPUT:



Experiment 5

AIM: To Implement Priority Queue ADT using array.

PROGRAM:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define MAX 5

int front,rear;

void insert_by_priority(int);
void delete_by_priority(int);
void create();
void check(int);
void display_pqueue();
int pri_que[MAX];

int main()
{
 int n,ch;
 clrscr();
 printf("1 - Insert an element into Queue\n");
 printf("2 - Delete an element from Queue\n");
 printf("3 - Display queue elements\n");
 printf("4 - Exit\n");
 create();
 while(1)
 {
  printf("Enter your choice : ");
  scanf("%d",&ch);
  switch(ch)
  {
   case 1: printf("Enter value to be inserted : ");
           scanf("%d",&n);
           insert_by_priority(n);
           break;
   case 2: printf("Enter value to delete : ");
           scanf("%d",&n);
           delete_by_priority(n);
           break;
   case 3: display_pqueue();
           break;
   case 4: exit(0);
   default: printf("\nChoice is Incorrect : Enter a correct choice\n");
  }
 }
```

```
}

void create()
{
 front = rear = -1;
}

void insert_by_priority(int data)
{
 if(rear >= MAX - 1)
 {
  printf("\nQueue Overflow : No more elements can be inserted\n");
  return;
 }
 if((front == -1) && (rear == -1))
 {
  front++;
  rear++;
  pri_que[rear] = data;
  return;
 }
 else
 {
  check(data);
  rear++;
 }
}

void check(int data)
{
 int i,j;
 for(i = 0;i <= rear;i++)
 {
  if(data >= pri_que[i])
  {
   for(j = rear + 1;j > i;j--)
   {
    pri_que[j] = pri_que[j - 1];
   }
   pri_que[i] = data;
   return;
  }
 }
 pri_que[i] = data;
}

void delete_by_priority(int data)
{
 int i;
 if((front==-1) && (rear==-1))
```

```
 {
  printf("\nQueue is Empty : No elements to delete\n");
  return;
 }
 for(i = 0;i <= rear;i++)
 {
  if(data == pri_que[i])
  {
   for(;i < rear;i++)
   {
    pri_que[i] = pri_que[i + 1];
   }
   pri_que[i] = -99;
   rear--;
   if(rear == -1)
   {
    front = -1;
   }
   return;
  }
 }
 printf("\n%d not found in queue to delete\n",data);
}

void display_pqueue()
{
 if((front == -1) && (rear == -1))
 {
  printf("Queue is empty\n");
  return;
 }
 for(;front <= rear;front++)
 {
  printf(" %d ",pri_que[front]);
 }
 printf("\n");
 front = 0;
}
```

OUTPUT:

```
1 - Insert an element into Queue
2 - Delete an element from Queue
3 - Display queue elements
4 - Exit
Enter your choice : 1
Enter value to be inserted : 25
Enter your choice : 1
Enter value to be inserted : 125
Enter your choice : 1
Enter value to be inserted : 625
Enter your choice : 3
 625  125  25
Enter your choice : 2
Enter value to delete : 125
Enter your choice : 3
 625  25
Enter your choice : S
```

Experiment 6

AIM: To Implement Singly Linked List ADT.

PROGRAM:

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

struct node
{
 int data;
 struct node *next;
};

int data = 0;
int count = 0;
struct node *head = NULL;
struct node *new_node = NULL;
struct node *temp = NULL;
struct node *prev = NULL;

struct node *create_node(int);
void insert_at_beginning(int);
void insert_at_end(int);
void insert_at_position(int,int);
void delete_at_beginning();
void delete_at_end();
void delete_at_position(int);
void print_from_beginning();
void print_from_end(struct node *);
void search_data(int);
void update_node_data(int,int);
void empty_message(void);
int size_of_list();
int getData();
int getPosition();

int main()
{
 int user_choice;
 int data,position;
 char user_active = 'Y';
 clrscr();
 while(user_active == 'Y' || user_active == 'y')
 {
  printf("\n------- Singly Linked List------- \n");
  printf("1.Insert a node at beginning\n");
```

```
printf("2.Insert a node at end\n");
printf("3.Insert a node at given position\n");
printf("4.Delete a node from beginning\n");
printf("5.Delete a node from end\n");
printf("6.Delete a node from given position\n");
printf("7.Print list from beginning\n");
printf("8.Print list from end\n");
printf("9.Search a node data\n");
printf("10.Update a node data\n");
printf("11.Exit\n");
printf("\n");
printf("Enter your choice : ");
scanf("%d",&user_choice);
switch(user_choice)
{
 case 1: printf("Inserting a node at beginning\n");
         data = getData();
         insert_at_beginning(data);
         break;
 case 2: printf("Inserting a node at end\n");
         data = getData();
         insert_at_end(data);
         break;
 case 3: printf("Inserting a node at the given position\n");
         data = getData();
         position = getPosition();
         insert_at_position(data,position);
         break;
 case 4: printf("Deleting a node from beginning\n");
         delete_at_beginning();
         break;
 case 5: printf("Deleting a node from end\n");
         delete_at_end();
         break;
 case 6: printf("Delete a node from given position\n");
         position = getPosition();
         delete_at_position(position);
         break;
 case 7: printf("Printing the list from beginning\n");
         print_from_beginning();
         break;
 case 8: printf("Printing the list from end\n");
         print_from_end(head);
         break;
 case 9: printf("\nSearching the node data");
         data = getData();
         search_data(data);
         break;
 case 10: printf("Updating the node data\n");
          data = getData();
```

```
            position = getPosition();
            update_node_data(data,position);
            break;
   case 11: printf("Program was terminated\n");
            return 0;
   default: printf("Invalid Choice\n");
  }
  printf("\n ............................ \n");
  printf("Do you want to continue? (Y/N) : ");
  fflush(stdin);
  scanf(" %c",&user_active);
 }
 return 0;
}

void empty_message()
{
 printf("List is Empty !\n");
}

void memory_message()
{
 printf("Memory can't be allocated\n");
}

struct node *create_node(int data)
{
 struct node *new_node = (struct node *) malloc(sizeof(struct node));
 if(new_node == NULL)
 {
  memory_message();
  return NULL;
 }
 new_node->data = data;
 new_node->next = NULL;
 return new_node;
}

void insert_at_beginning(int data)
{
 struct node *new_node = NULL;
 new_node = create_node(data);
 if(new_node != NULL)
 {
  new_node->next = head;
  head = new_node;
  printf("Node with data %d was Inserted\n",data);
 }
}
```

```
void insert_at_end(int data)
{
 struct node *new_node = NULL;
 new_node = create_node(data);
 if(new_node != NULL)
 {
  if(head == NULL)
  {
   head = new_node;
  }
  else
  {
   struct node *last = head;
   while(last->next != NULL)
   {
    last = last->next;
   }
   last->next = new_node;
  }
  printf("Node with data %d was Inserted\n",data);
 }
}

void insert_at_position(int data,int pos)
{
 int list_size = 0;
 list_size = size_of_list();
 if((head == NULL) && (pos <= 0 || pos > 1))
 {
  printf("Invalid position to insert a node\n");
  return;
 }
 if((head != NULL) && (pos <= 0 || pos > list_size))
 {
  printf("Invalid position to insert a node\n");
  return;
 }
 new_node = NULL;
 new_node = create_node(data);
 if(new_node != NULL)
 {
  struct node *temp = head;
  int count = 1;
  while(count < pos-1)
  {
   temp = temp->next;
   count += 1;
  }
  if(pos == 1)
  {
```

```
   new_node->next = head;
   head = new_node;
  }
  else
  {
   new_node->next = temp->next;
   temp->next = new_node;
  }
  printf("Node with data %d was Inserted\n",data);
 }
}

void delete_at_beginning()
{
 if(head == NULL)
 {
  empty_message();
  return;
 }
 temp = head;
 data = head->data;
 head = head->next;
 free(temp);
 printf("Node with data %d was Deleted\n",data);
}

void delete_at_end()
{
 if(head == NULL)
 {
  empty_message();
  return;
 }
 temp = head;
 prev = NULL;
 while(temp->next != NULL)
 {
  prev = temp;
  temp = temp->next;
 }
 data = temp->data;
 if(temp == head)
 {
  free(temp);
  head = NULL;
 }
 else
 {
  free(temp);
  prev->next = NULL;
```

```
 }
 printf("Node with data %d was Deleted\n",data);
}

void delete_at_position(int pos)
{
 int list_size = 0;
 list_size = size_of_list();
 if((pos <= 0) || (pos > list_size))
 {
  printf("Invalid position to delete a node\n");
  return;
 }
 temp = head;
 prev = NULL;
 count = 1;
 while(count < pos)
 {
  prev = temp;
  temp = temp->next;
  count += 1;
 }
 data = temp->data;
 if(temp == head)
 {
  head = head->next;
  free(temp);
 }
 else
 {
  prev->next = temp->next;
  free(temp);
 }
 printf("Node with data %d was Deleted\n",data);
}

void search_data(int data)
{
 int position = 0;
 int flag = 0;
 struct node *temp = head;
 while(temp != NULL)
 {
  position += 1;
  if(temp->data == data)
  {
   flag = 1;
   break;
  }
  temp = temp->next;
```

```
 }
 if(flag == 0)
 {
  printf("Node with data %d was not found !\n",data);
 }
 else
 {
  printf("Found data at %d position\n",position);
 }
}

void update_node_data(int new_data,int pos)
{
 int list_size = 0;
 list_size = size_of_list();
 if((pos <= 0) || (pos > list_size))
 {
  printf("Invalid position to update a node\n");
  return;
 }
 temp = head;
 count = 1;
 while(count < pos)
 {
  temp = temp->next;
  count += 1;
 }
 temp->data = new_data;
 printf("Updated node data is %d\n",new_data);
}

void print_from_beginning()
{
 if(head == NULL)
 {
  empty_message();
  return;
 }
 temp = head;
 while(temp != NULL)
 {
  printf("%d ",temp->data);
  temp = temp->next;
 }
}

void print_from_end(struct node *head)
{
 if(head == NULL)
 {
```

```
  return;
 }
 print_from_end(head->next);
 printf("%d ",head->data);
}

int size_of_list()
{
 struct node *temp = head;
 int count = 0;
 while(temp != NULL)
 {
  count += 1;
  temp = temp->next;
 }
 return count;
}

int getData()
{
 int data;
 printf("\nEnter Data : ");
 scanf("%d",&data);
 return data;
}

int getPosition()
{
 int pos;
 printf("\nEnter Position : ");
 scanf("%d",&pos);
 return pos;
}
```

OUTPUT:

```
------- Singly Linked List
1.Insert a node at beginning
2.Insert a node at end
3.Insert a node at given position
4.Delete a node from beginning
5.Delete a node from end
6.Delete a node from given position
7.Print list from beginning
8.Print list from end
9.Search a node data
10.Update a node data
11.Exit

Enter your choice : 1
Inserting a node at beginning

Enter Data : 50
Node with data 50 was Inserted


Do you want to continue? (Y/N) : Y
```

```
------- Singly Linked List
1.Insert a node at beginning
2.Insert a node at end
3.Insert a node at given position
4.Delete a node from beginning
5.Delete a node from end
6.Delete a node from given position
7.Print list from beginning
8.Print list from end
9.Search a node data
10.Update a node data
11.Exit

Enter your choice : 2
Inserting a node at end

Enter Data : 100_

Enter Data : 100
Node with data 100 was Inserted


Do you want to continue? (Y/N) : Y

------- Singly Linked List
1.Insert a node at beginning
2.Insert a node at end
3.Insert a node at given position
4.Delete a node from beginning
5.Delete a node from end
6.Delete a node from given position
7.Print list from beginning
8.Print list from end
9.Search a node data
10.Update a node data
11.Exit

Enter your choice : 2
Inserting a node at end

Enter Data : 150_
```

```
Do you want to continue? (Y/N) : Y

------- Singly Linked List
1.Insert a node at beginning
2.Insert a node at end
3.Insert a node at given position
4.Delete a node from beginning
5.Delete a node from end
6.Delete a node from given position
7.Print list from beginning
8.Print list from end
9.Search a node data
10.Update a node data
11.Exit

Enter your choice : 2
Inserting a node at end

Enter Data : 200
Node with data 200 was Inserted


Do you want to continue? (Y/N) : Y
8.Print list from end
9.Search a node data
10.Update a node data
11.Exit

Enter your choice : 7
Printing the list from beginning
50 100 150 200

Do you want to continue? (Y/N) : Y

------- Singly Linked List
1.Insert a node at beginning
2.Insert a node at end
3.Insert a node at given position
4.Delete a node from beginning
5.Delete a node from end
6.Delete a node from given position
7.Print list from beginning
8.Print list from end
9.Search a node data
10.Update a node data
11.Exit

Enter your choice : 4
```

```
Enter your choice : 4
Deleting a node from beginning
Node with data 50 was Deleted


Do you want to continue? (Y/N) : Y

------- Singly Linked List
1.Insert a node at beginning
2.Insert a node at end
3.Insert a node at given position
4.Delete a node from beginning
5.Delete a node from end
6.Delete a node from given position
7.Print list from beginning
8.Print list from end
9.Search a node data
10.Update a node data
11.Exit

Enter your choice : 10
```

```
Enter your choice : 10
Updating the node data

Enter Data : 250

Enter Position : 3
Updated node data is 250


Do you want to continue? (Y/N) : Y

------- Singly Linked List
1.Insert a node at beginning
2.Insert a node at end
3.Insert a node at given position
4.Delete a node from beginning
5.Delete a node from end
6.Delete a node from given position
7.Print list from beginning
8.Print list from end
9.Search a node data
10.Update a node data
11.Exit

Enter your choice : 7_
```

```
Enter your choice : 7
Printing the list from beginning
100 150 250

Do you want to continue? (Y/N) : N_
```

Experiment 7

AIM: To Implement Circular Linked List ADT

PROGRAM:

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

struct node
{
 int data;
 struct node *next;
};

struct node *head;

void beginsert();
void lastinsert();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();

int main()
{
 int choice = 0;
 clrscr();
 while(choice != 7)
 {
  printf("\n**********Main Menu**********\n");
  printf("Choose one option from the following list...\n");
  printf("\n===============================================\n");
  printf("1.Insert in beginning\n2.Insert at last\n3.Delete from
  beginning\n4.Delete from last\n5.Search for an
  element\n6.Show\n7.Exit\n\n");
  printf("Enter your choice : ");
  scanf("%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");
   }
 }
 getch();
 return 0;
}

void beginsert()
{
 int item;
 struct node *ptr,*temp;
 ptr = (struct node *) malloc(sizeof(struct node));
 if(ptr == NULL)
 {
  printf("OVERFLOW\n");
 }
 else
 {
  printf("Enter 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("Node Inserted\n");
 }
}

void lastinsert()
{
 int item;
 struct node *ptr,*temp;
 ptr = (struct node *) malloc(sizeof(struct node));
 if(ptr == NULL)
 {
  printf("OVERFLOW\n");
 }
 else
 {
  printf("Enter 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("Node Inserted\n");
 }
}

void begin_delete()
{
 struct node *ptr;
 if(head == NULL)
 {
  printf("UNDERFLOW\n");
 }
 else if(head->next == head)
```

```
 {
  head = NULL;
  free(head);
  printf("Node Deleted\n");
 }
 else
 {
  ptr = head;
  while(ptr->next != head)
  ptr = ptr->next;
  ptr->next = head->next;
  free(head);
  head = ptr->next;
  printf("Node Deleted\n");
 }
}

void last_delete()
{
 struct node *ptr,*preptr;
 if(head == NULL)
 {
  printf("UNDERFLOW\n");
 }
 else if(head->next == head)
 {
  head = NULL;
  free(head);
  printf("Node Deleted\n");
 }
 else
 {
  ptr = head;
  while(ptr->next != head)
  {
   preptr = ptr;
   ptr = ptr->next;
  }
  preptr->next = ptr->next;
  free(ptr);
  printf("Node Deleted\n");
 }
}

void search()
{
```

```
 struct node *ptr;
 int item,i = 0,flag = 1;
 ptr = head;
 if(ptr == NULL)
 {
  printf("Empty List\n");
 }
 else
 {
  printf("Enter item which you want to search : ");
  scanf("%d",&item);
  if(head->data == item)
  {
   printf("Item found at location %d\n",i+1);
   flag = 0;
  }
  else
  {
   while(ptr->next != head)
   {
    if(ptr->data == item)
    {
     printf("Item found at location %d\n",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("Nothing to Print\n");
  }
  else
  {
   printf("Printing Values...\n");
   while(ptr->next != head)
   {
    printf("%d\n",ptr->data);
    ptr = ptr->next;
   }
   printf("%d\n",ptr->data);
  }
}
```

OUTPUT:

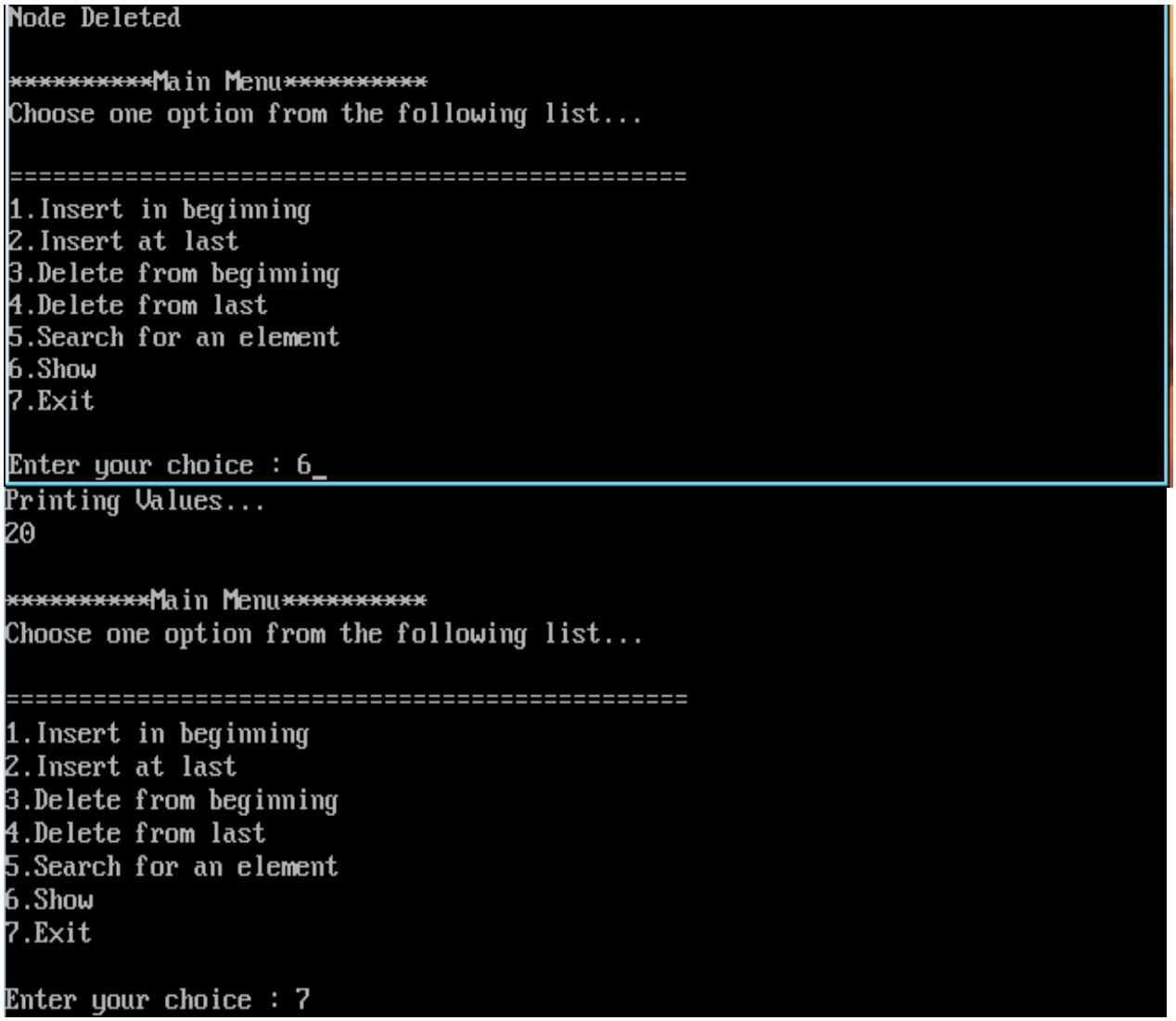
```
**********Main Menu**********
Choose one option from the following list...

================================================
1.Insert in beginning
2.Insert at last
3.Delete from beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice : 1
Enter the node data : 10_
```

```
Node Inserted

**********Main Menu**********
Choose one option from the following list...

================================================
1.Insert in beginning
2.Insert at last
3.Delete from beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice : 2
Enter Data : 20
```

```
Node Inserted

**********Main Menu**********
Choose one option from the following list...

===============================================
1.Insert in beginning
2.Insert at last
3.Delete from beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice : 2
Enter Data : 30
Node Inserted

**********Main Menu**********
Choose one option from the following list...

===============================================
1.Insert in beginning
2.Insert at last
3.Delete from beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice : 6_
Printing Values...
10
20
30

**********Main Menu**********
Choose one option from the following list...

===============================================
1.Insert in beginning
2.Insert at last
3.Delete from beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice : 4
```

```
Node Deleted

**********Main Menu**********
Choose one option from the following list...

===============================================
1.Insert in beginning
2.Insert at last
3.Delete from beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice : 6
Printing Values...
10
20

**********Main Menu**********
Choose one option from the following list...

===============================================
1.Insert in beginning
2.Insert at last
3.Delete from beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice : 5_
```

```
Enter item which you want to search : 10
Item found at location 1

**********Main Menu**********
Choose one option from the following list...

===============================================
1.Insert in beginning
2.Insert at last
3.Delete from beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice : 3
```


Node Deleted
**********Main Menu**********
Choose one option from the following list...
==========
1.Insert in beginning
2.Insert at last
3.Delete from beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit
Enter your choice : 6
Printing Values...
20
**********Main Menu**********
Choose one option from the following list...
==========
1.Insert in beginning
2.Insert at last
3.Delete from beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit
Enter your choice : 7

Experiment 8

AIM: Implement Stack / Linear Queue ADT using Linked List.

PROGRAM:

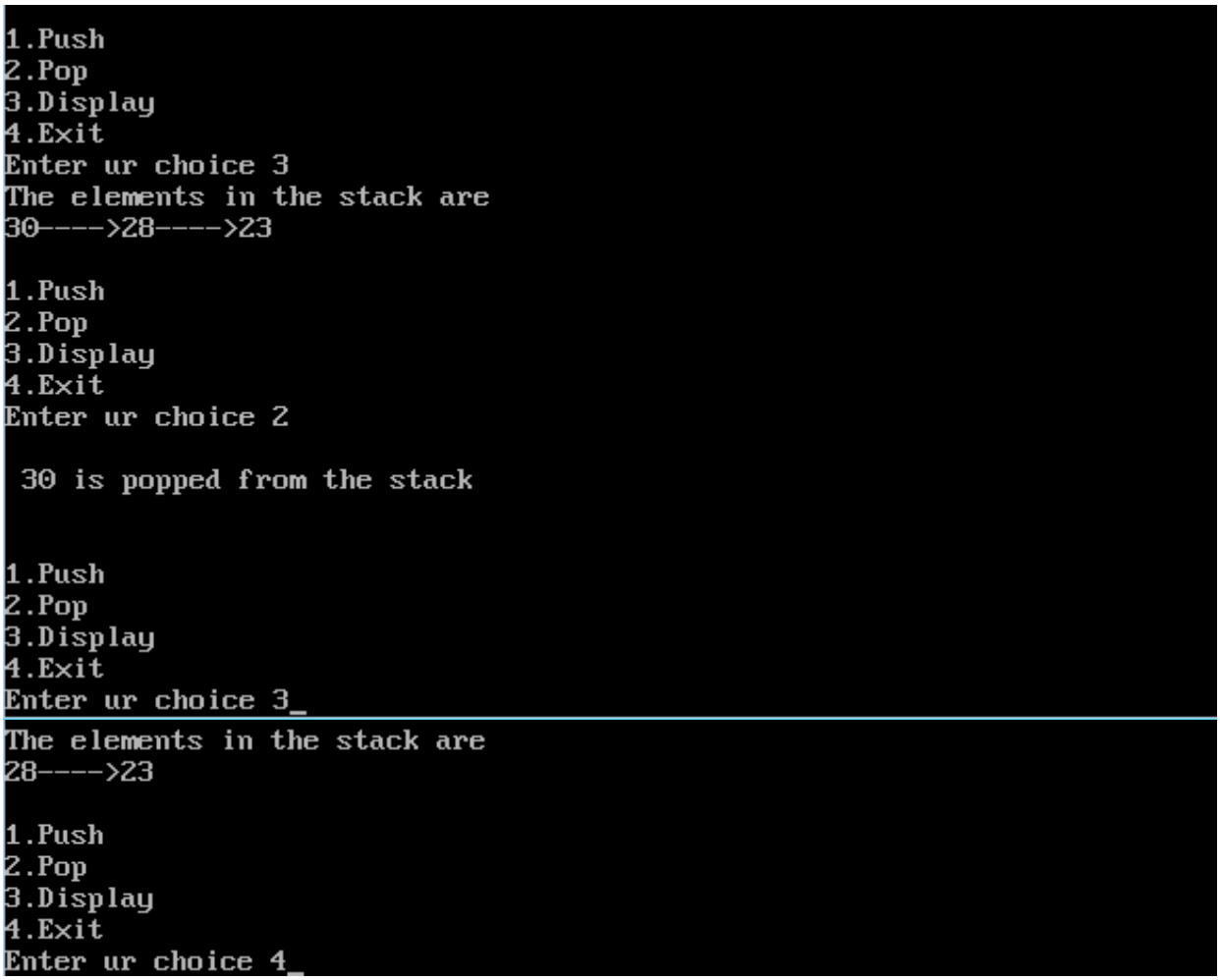
```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#include<malloc.h>
struct node
{ int data;
struct node *next;
}*top,*head,*temp;
void main()
{
void push();
void pop();
void printstack();
int ch;
head=(struct node*)malloc(sizeof(struct node));
head->next=NULL;
do
{ printf("\n\n1.Push\n2.Pop\n3.Display\n4.Exit\n");
printf("Enter ur choice\t");
scanf("%d",&ch);
switch(ch)
{
case 1:
push();
break;
case 2:
pop();
break;
case 3:
printstack();
break;
```

```
case 4:
exit(0);
}
}while(ch<=4);
getch();
}
void push()
{
temp=(struct node*)malloc(sizeof(struct node));
printf("Enter the data to push into the stack \t");
scanf("%d",&temp->data);
temp->next=head->next;
head->next=temp;
top=temp;
}
void pop()
{
if(head->next==NULL)
printf("Stack is empty\n");
else
{
temp=top;
printf("\n %d is popped from the stack\n",top->data);
top=top->next;
head->next=top;
free(temp);
}
}
void printstack(){
if(head->next==NULL)
printf("Stack is empty\n");

else
{
temp=top;
printf("The elements in the stack are\n");
while(temp->next!=NULL)
```

```
{
printf("%d---->",temp->data);
temp=temp->next;
}
printf("%d", temp->data);
}
}
```

OUTPUT:

```
1.Push
2.Pop
3.Display
4.Exit
Enter ur choice 1
Enter the data to push into the stack   23


1.Push
2.Pop
3.Display
4.Exit
Enter ur choice 1
Enter the data to push into the stack   28


1.Push
2.Pop
3.Display
4.Exit
Enter ur choice 1
Enter the data to push into the stack   30_
```


1.Push
2.Pop
3.Display
4.Exit
Enter ur choice 3
The elements in the stack are
30---->28---->23
1.Push
2.Pop
3.Display
4.Exit
Enter ur choice 2
30 is popped from the stack
1.Push
2.Pop
3.Display
4.Exit
Enter ur choice 3
The elements in the stack are
28---->23
1.Push
2.Pop
3.Display
4.Exit
Enter ur choice 4

Experiment 9

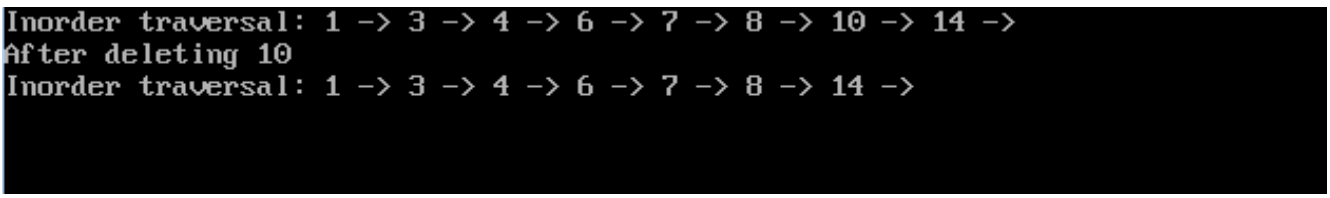
AIM: Implement Binary Search Tree ADT using Linked List.

PROGRAM:

```
#include <stdio.h>
#include<conio.h>
#include <stdlib.h>
struct node
{
int key;
struct node *left, *right;
};
struct node *newNode(int item)
{
struct node *temp = (struct node *)malloc(sizeof(struct node));
temp->key = item;
temp->left = temp->right = NULL;
return temp;
}
void inorder(struct node *root)
{
if (root != NULL)
{
inorder(root->left);
printf("%d -> ", root->key);
inorder(root->right);
}
}
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;
}
struct node *minValueNode(struct node *node) {
struct node *current = node;
while (current && current->left != NULL)
current = current->left;
return current;
}
struct node *deleteNode(struct node *root, int key) {
if (root == NULL) return root;
if (key < root->key) root->left = deleteNode(root->left, key);
else if (key > root->key)
root->right = deleteNode(root->right, key);
else {
if (root->left == NULL) {
struct node *temp = root->right;
```

```
free(root);
return temp;
} else if (root->right == NULL) {
struct node *temp = root->left;
free(root);
return temp;
}
struct node *temp = minValueNode(root->right); root->key = temp-
>key;
root->right = deleteNode(root->right, temp->key); }
return root;
}
void main() {
clrscr();
struct node *root = NULL;
root = insert(root, 8);
root = insert(root, 3);
root = insert(root, 1);
root = insert(root, 6);
root = insert(root, 7);
root = insert(root, 10);
root = insert(root, 14);
root = insert(root, 4);
printf("Inorder traversal: ");
inorder(root);
printf("\nAfter deleting 10\n");
root = deleteNode(root, 10);
printf("Inorder traversal: ");
inorder(root);
getch();
}
```

OUTPUT:



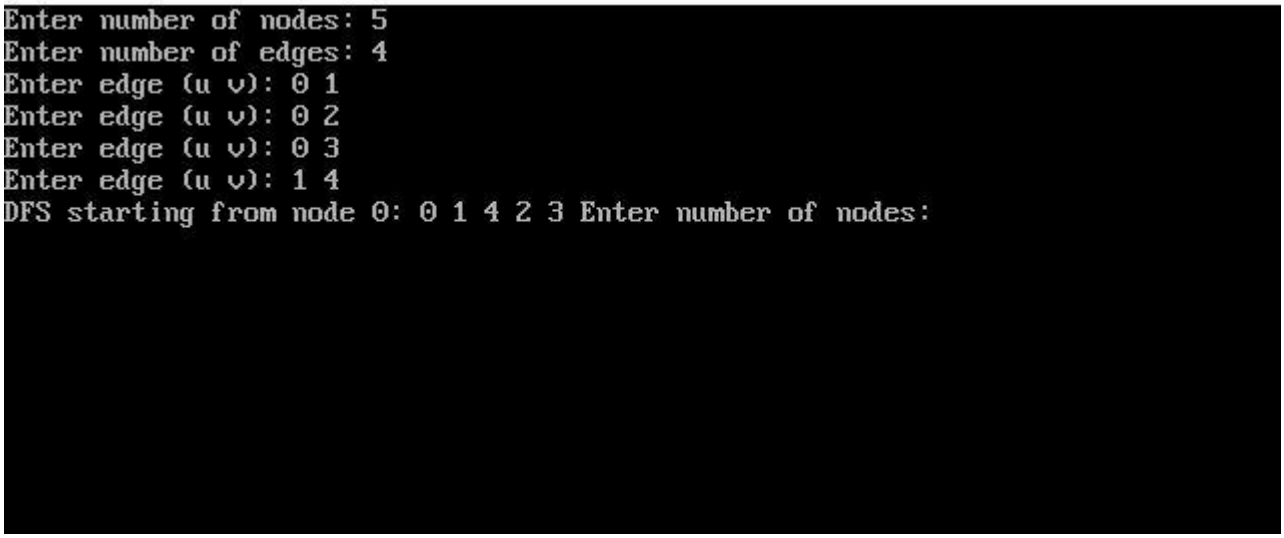
EXPERIMENT 10

C CODE FOR BFS

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

#define MAX 100

// Graph structure using an adjacency matrix
int adj[MAX][MAX]; // Adjacency matrix to represent the graph
int visited[MAX]; // Array to mark visited nodes
int nodes; // Number of nodes in the graph

// Queue structure for BFS
int queue[MAX], front = -1, rear = -1;

// Function to insert an element into the queue
void enqueue(int v) {
if (rear == MAX - 1) {
printf("Queue Overflow\n");
return;
}
if (front == -1) {
front = 0;
}
rear++;
queue[rear] = v;
}

// Function to remove and return an element from the queue
int dequeue() {
if (front == -1 || front > rear) {
printf("Queue Underflow\n");

return -1;
}
int v = queue[front];
front++;
return v;
}

// Function to check if the queue is empty
int isQueueEmpty() {
return front == -1 || front > rear;
}

// Function to perform BFS
void BFS(int start) {
// Mark the start node as visited and enqueue it
visited[start] = 1;
enqueue(start);

printf("BFS Traversal: ");
```

```
while (!isQueueEmpty()) {
// Dequeue a node and print it
int v = dequeue();
printf("%d ", v);

// Visit all adjacent nodes of dequeued node
for (int i = 0; i < nodes; i++) {
if (adj[v][i] == 1 && !visited[i]) {
visited[i] = 1; // Mark as visited
enqueue(i); // Enqueue adjacent node
}

}
}
printf("\n");
}

int main() {
int edges, u, v;

// Input number of nodes
printf("Enter number of nodes: ");
scanf("%d", &nodes);

// Input number of edges
printf("Enter number of edges: ");
scanf("%d", &edges);

// Initialize adjacency matrix and visited array
for (int i = 0; i < nodes; i++) {
for (int j = 0; j < nodes; j++) {
adj[i][j] = 0;
}
visited[i] = 0;
}

// Input edges
for (int i = 0; i < edges; i++) {
printf("Enter edge (u v): ");
scanf("%d %d", &u, &v);
adj[u][v] = 1;
adj[v][u] = 1; // For undirected graph
}

// Perform BFS starting from node 0
BFS(0);

return 0;
}
```

OUTPUT:

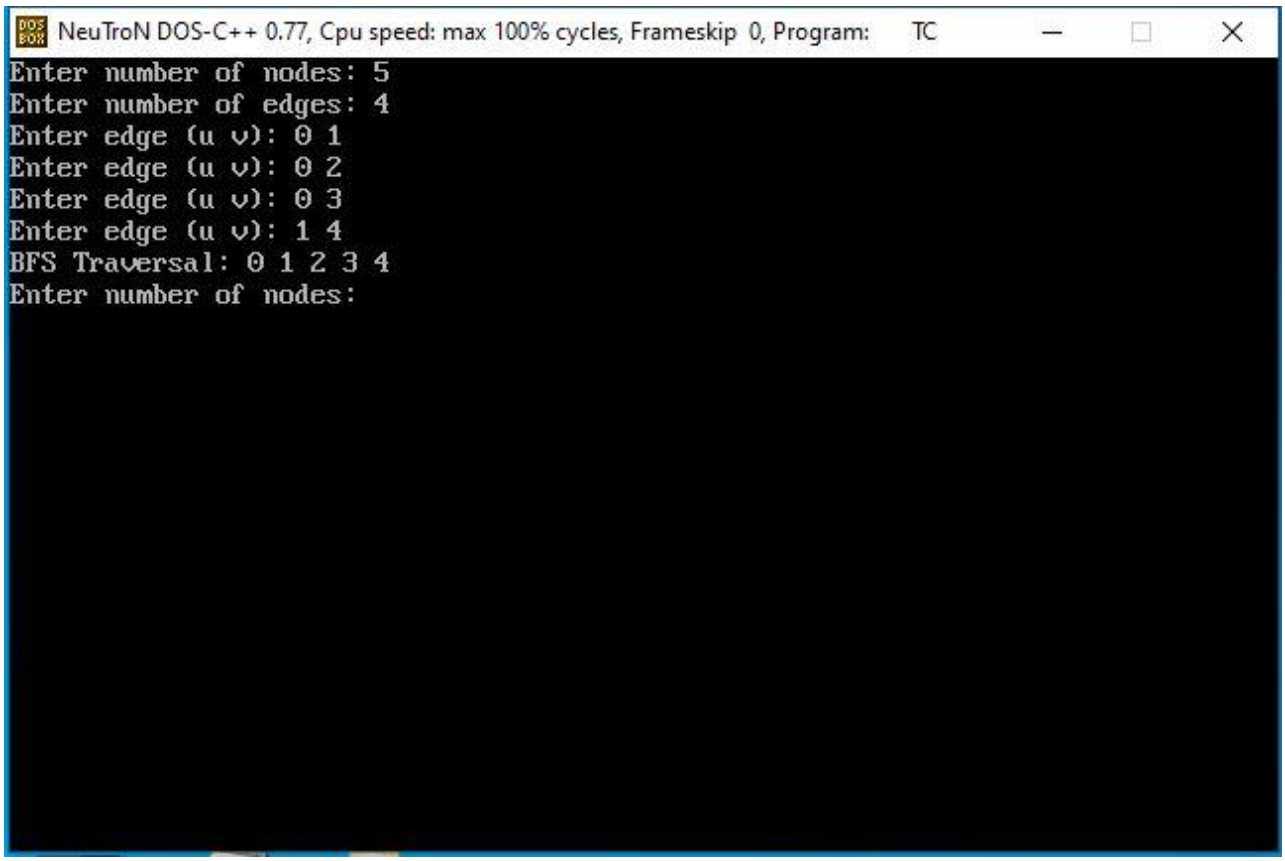


DFS C CODE

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

#define MAX 100

// Graph structure using an adjacency matrix
int adj[MAX][MAX];
int visited[MAX];
int nodes;

// Function to perform DFS
void DFS(int v) {
// Mark the node as visited
visited[v] = 1;
printf("%d ", v); // Output the node

// Visit all adjacent vertices that are not visited
for (int i = 0; i < nodes; i++) {
if (adj[v][i] == 1 && !visited[i]) {
DFS(i);
}
}
}

int main() {
int edges, u, v;

// Input number of nodes
```

```
printf("Enter number of nodes: ");
scanf("%d", &nodes);

// Input number of edges
printf("Enter number of edges: ");
scanf("%d", &edges);

// Initialize adjacency matrix to 0
for (int i = 0; i < nodes; i++) {
for (int j = 0; j < nodes; j++) {
adj[i][j] = 0;
}
}

// Initialize visited array to 0
for (int i = 0; i < nodes; i++) {
visited[i] = 0;
}

// Input edges
for (int i = 0; i < edges; i++) {
printf("Enter edge (u v): ");
scanf("%d %d", &u, &v);
adj[u][v] = 1;
adj[v][u] = 1; // For undirected graph
}

// Perform DFS starting from node 0
printf("DFS starting from node 0: ");
DFS(0);

return 0;
}
```

OUTPUT:


NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter number of nodes: 5
Enter number of edges: 4
Enter edge (u v): 0 1
Enter edge (u v): 0 2
Enter edge (u v): 0 3
Enter edge (u v): 1 4
BFS Traversal: 0 1 2 3 4
Enter number of nodes: