POLYNOMIAL ADDITION

AIM: Write a program to perform polynomial addition using arrays.

PROGRAM CODE:

```
#include<stdio.h>
void main()
{
int fir[10],sec[10],res[20],m,n;
int i=0,k=0,j=0;
printf("\n Enter the limit of first polynomial: ");
scanf("%d",&m);
printf("\n Enter the limit of second polynomial: ");
scanf("%d",&n);
printf("\n Enter the coefficent and exponent of first polynomial: ");
for(i=0;i<2*m-1;i=i+2)
{
scanf("%d %d",&fir[i],&fir[i+1]);
}
printf("\n Enter the coefficent and exponent of second polynomial: ");
for(j=0;j<2*n-1;j=j+2)
{
scanf("%d %d",&sec[j],&sec[j+1]);
}
printf("\n The first polynomial is : ");
for(i=0;i<2*m-2;i=i+2)
{
printf("%dx^%d+ ",fir[i],fir[i+1]);
```

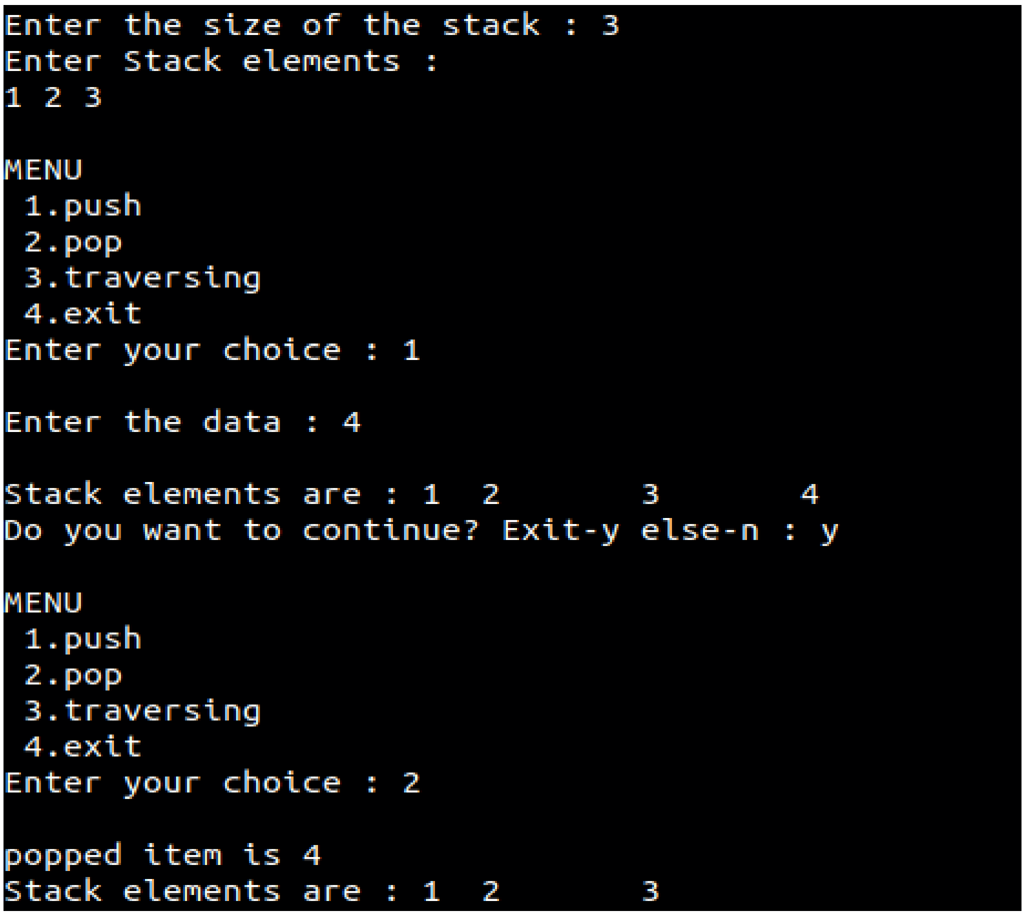
```
}
printf("%dx^%d ",fir[i],fir[i+1]);
printf("\n The second polynomial is : ");
for(j=0;j<2*n-2;j=j+2)
{
printf("%dx^%d+ ", sec[j],sec[j+1]);
}
printf("%dx^%d ",sec[j],sec[j+1]);
for(i=0,j=0,k=0;(i<2*m-1) && (j<2*n-1);)
{
if(fir[i+1]==sec[j+1])
{
res[k]=fir[i]+sec[j];
res[k+1]=fir[i+1];
i=i+2;
j=j+2;
k=k+2;
}
else if(fir[i+1]<sec[j+1])
{
res[k]=sec[j];
res[k+1]=sec[j+1];
j=j+2;
k=k+2;
}
else if(fir[i+1]>sec[j+1])
{
res[k]=fir[i];
res[k+1]=fir[i+1];
```

```
k=k+2;
i=i+2;
}
}
for(;i<2*m-1;i=i+2)
{
res[k]=fir[i];
 res[k+1]=fir[i+1];
 k=k+2;
}
for(;j<2*n-1;j=j+2)
{
res[k]=sec[j];
res[k+1]=sec[j+1];
k=k+2;
}
n=k;
printf("\n After addition : ");
for(i=0;i<k;i=i+2)
{
printf(" %dx^%d",res[i],res[i+1]);
}
}
```

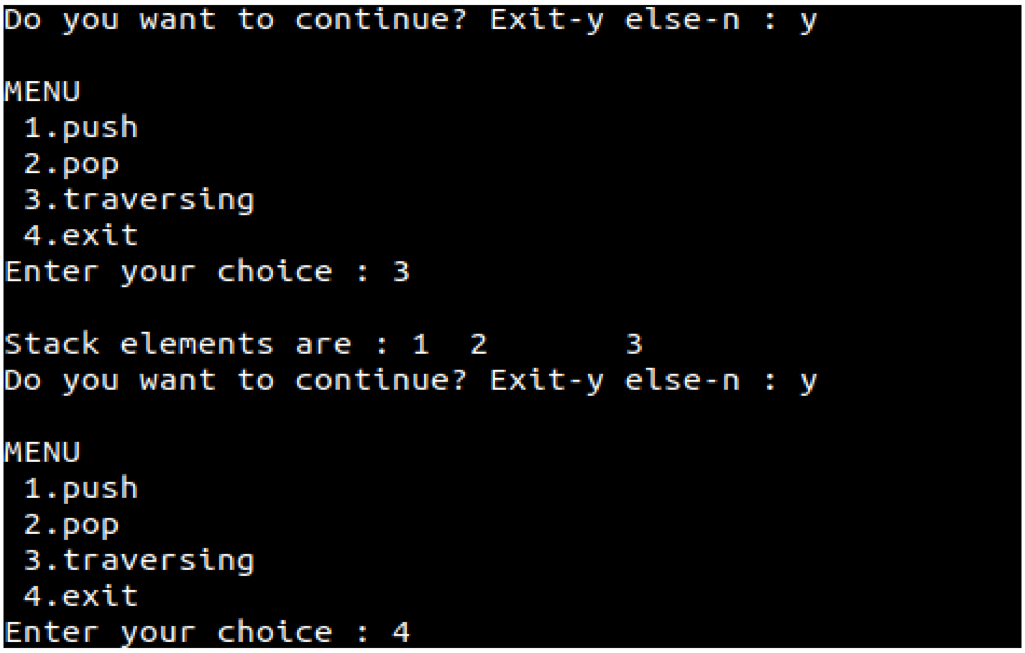
RESULT:

Program executed successfully and output verified.

OUTPUT:



```
Enter the size of the stack : 3
Enter Stack elements :
1 2 3

MENU
 1.push
 2.pop
 3.traversing
 4.exit
Enter your choice : 1

Enter the data : 4

Stack elements are : 1  2       3       4
Do you want to continue? Exit-y else-n : y

MENU
 1.push
 2.pop
 3.traversing
 4.exit
Enter your choice : 2

popped item is 4
Stack elements are : 1  2       3
```



```
Do you want to continue? Exit-y else-n : y

MENU
 1.push
 2.pop
 3.traversing
 4.exit
Enter your choice : 3

Stack elements are : 1  2       3
Do you want to continue? Exit-y else-n : y

MENU
 1.push
 2.pop
 3.traversing
 4.exit
Enter your choice : 4
```

LINEAR STACK OPERATIONS

AIM: Write a menu driven program to perform linear stack operations.

PROGRAM CODE:

```
#include<stdio.h>

#include<stdlib.h>

void pushs();

void pops();

void visit();

int n,top;

int a[25];

void main()

{

int i,n,c;

char d;

printf("Enter the size of the stack : ");

scanf("%d",&n);

if(n>25)

{

printf("\n stack overflow");

exit(0);

}

else

printf("Enter Stack elements :\n");

for(top=0;top<n;top++)

scanf("%d",&a[top]);

do

{
```

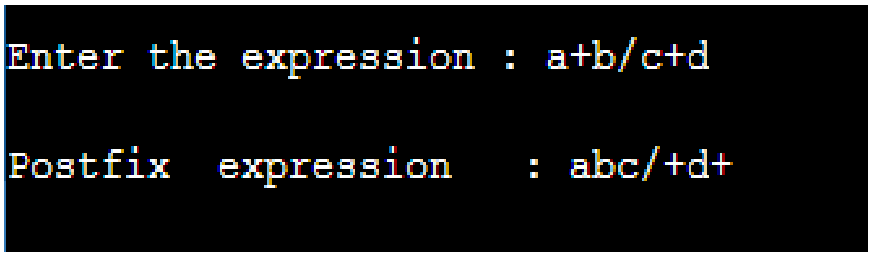
```
printf("\nMENU \n 1.push\n 2.pop \n 3.traversing\n 4.exit\nEnter your choice : ");
scanf("%d",&c);
switch(c)
{
case 1:pushs();
        break;
case 2:pops();
        break;
case 3:visit();
        break;
case 4:exit(0);
default:printf("invalid input");
}
printf("\nDo you want to continue? Exit-y else-n : ");
scanf("%s",&d);
}while(d=='y');
}
void pushs()
{
int data,i;
printf("\nEnter the data : ");
scanf("%d",&data);
if(top>=25)
printf("stack overflow");
else
{
a[top]=data;
top++;
}
```

```
visit(a);
return;
}
void pops()
{
int data;
top=top-1;
if(top==0)
printf("stack underflow");
else
{
data=a[top];
printf("\npopped item is %d",data);
}
visit(a);
return;
}
void visit()
{
printf("\nStack elements are : ");
for(int i=0;i<top;i++)
printf("%d\t",a[i]);
return;
}
```

RESULT:

Program executed successfully and output verified.

OUTPUT:



```
Enter the expression : a+b/c+d

Postfix  expression   : abc/+d+
```

INFIX TO POSTFIX EXPRESSION

AIM: Write a program to convert an infix expression to a postfix expression using stack.

PROGRAM CODE:

```
#include<stdio.h>
#include<ctype.h>
char st[20];
int i,t = -1;
void add(char x)
{
st[++t] = x;
}
char del()
{
if(t == -1)
return -1;
else
return st[t--];
}
int priority(char x)
{
if(x == '(')
return 0;
if(x=='+'||x=='-')
return 1;
if(x=='*'||x=='/')
return 2;
if(x=='$'||x=='#')
```

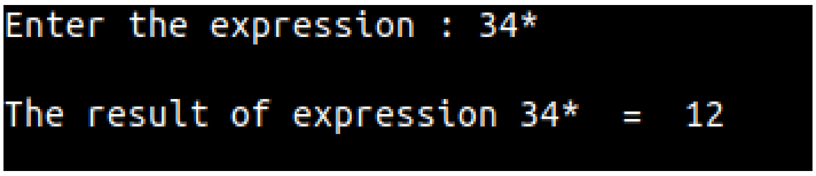
```
return 3;
}
int main()
{
char exp[20];
char e[20], x;
printf("\nEnter the expression : ");
scanf("%s",exp);
printf("\nPostfix  expression   : ");
for(i=0;exp[i]!='\0';i++)
e[i] = exp[i];
i=0;
while(e[i] != '\0')
{
if (isalnum(e[i]))
printf("%c",e[i]);
else if(e[i] == '(')
add(e[i]);
else if(e[i] == ')')
{
while((x = del()) != '(')
printf("%c", x);
}
else
{
while(priority(st[t]) >= priority(e[i]))
printf("%c",del());
add(e[i]);
}
```

```
i++;
}
while(t != -1)
{
printf("%c",del());
}
printf("\n\n");
}
```

RESULT:

Program executed successfully and output verified.

OUTPUT:



```
Enter the expression : 34*

The result of expression 34*  =  12
```

EVALUATION OF POSTFIX EXPRESSION

AIM: Write a program to evaluate a postfix expression using stack.

PROGRAM CODE:

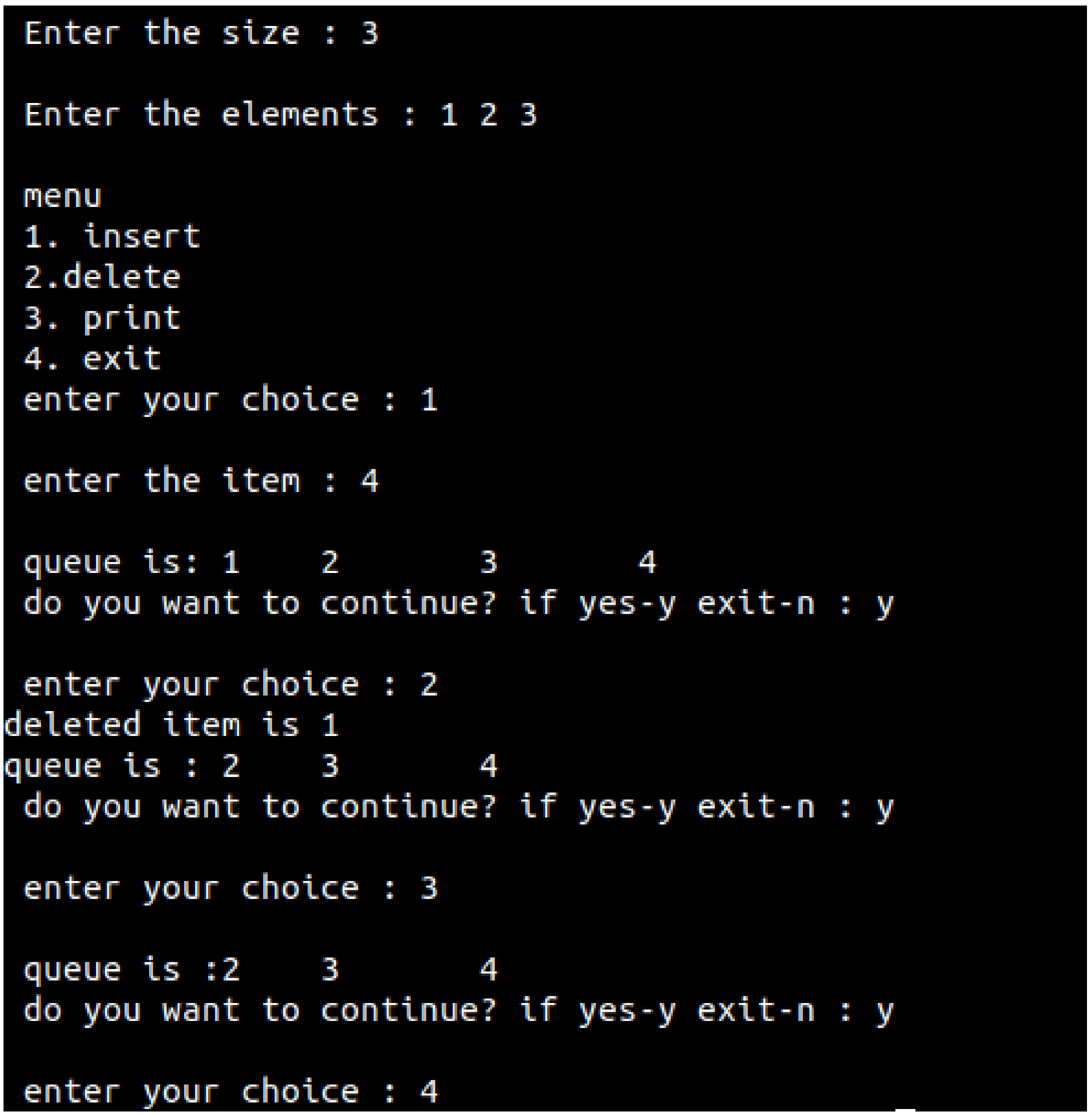
```
#include<stdio.h>
#include<string.h>
char expr[50],ch;
int stk[100];
void main()
{int i,cnt=0,a,b,res;
int top=-1;
printf("Enter your expression : ");
scanf("%s",expr);
for(i=0;i<strlen(expr);i++)
{if(expr[i]>='0'&& expr[i]<='9')
{top++;
stk[top]=(int)(expr[i]-48);
}
if(expr[i]=='+'||expr[i]=='-'||expr[i]=='*'||expr[i]=='/'||expr[i]=='^')
{ch=expr[i];
a=stk[top];
stk[top]='\0';
top--;
b=stk[top];
stk[top]='\0';
top--;
switch(ch)
{
```

```
case '+' :res=b+a;
             break;
case '-' :res=b-a;
            break;
case '*' :res=b*a;
            break;
case '/' :res=b/a;
            break;
case '^' :res=b^a;
            break;
}
top++;
stk[top]=res;
}
}
printf("The result of expression %s = %d \n\n”,expr,stk[top]);
}
```

RESULT:

Program executed successfully and output verified.

OUTPUT:



```
 Enter the size : 3

 Enter the elements : 1 2 3

 menu
 1. insert
 2.delete
 3. print
 4. exit
 enter your choice : 1

 enter the item : 4

 queue is: 1    2       3       4
 do you want to continue? if yes-y exit-n : y

 enter your choice : 2
deleted item is 1
queue is : 2    3       4
 do you want to continue? if yes-y exit-n : y

 enter your choice : 3

 queue is :2    3       4
 do you want to continue? if yes-y exit-n : y

 enter your choice : 4
```

LINEAR QUEUE OPERATIONS

AIM: Write a menu driven program to perform linear queue operations.

PROGRAM CODE:

```
#include<stdio.h>
#include<stdlib.h>
int f,r,front,rear;
int n,q[50];
void main()
{
int a,item,x,i,loc=0,c=0,z;
char k;
f=r=-1;
printf("\n Enter the size : ");
scanf("%d",&n);
if(n>25)
printf("\n Queue overflow");
else
printf("\n Enter the elements : ");
while(r<n)
{
if(r==-1)
r=0;
scanf("%d",&q[r]);
r++;
}
rear--;
printf("\n menu\n 1. insert\n 2.delete\n 3. print\n 4. exit");
```

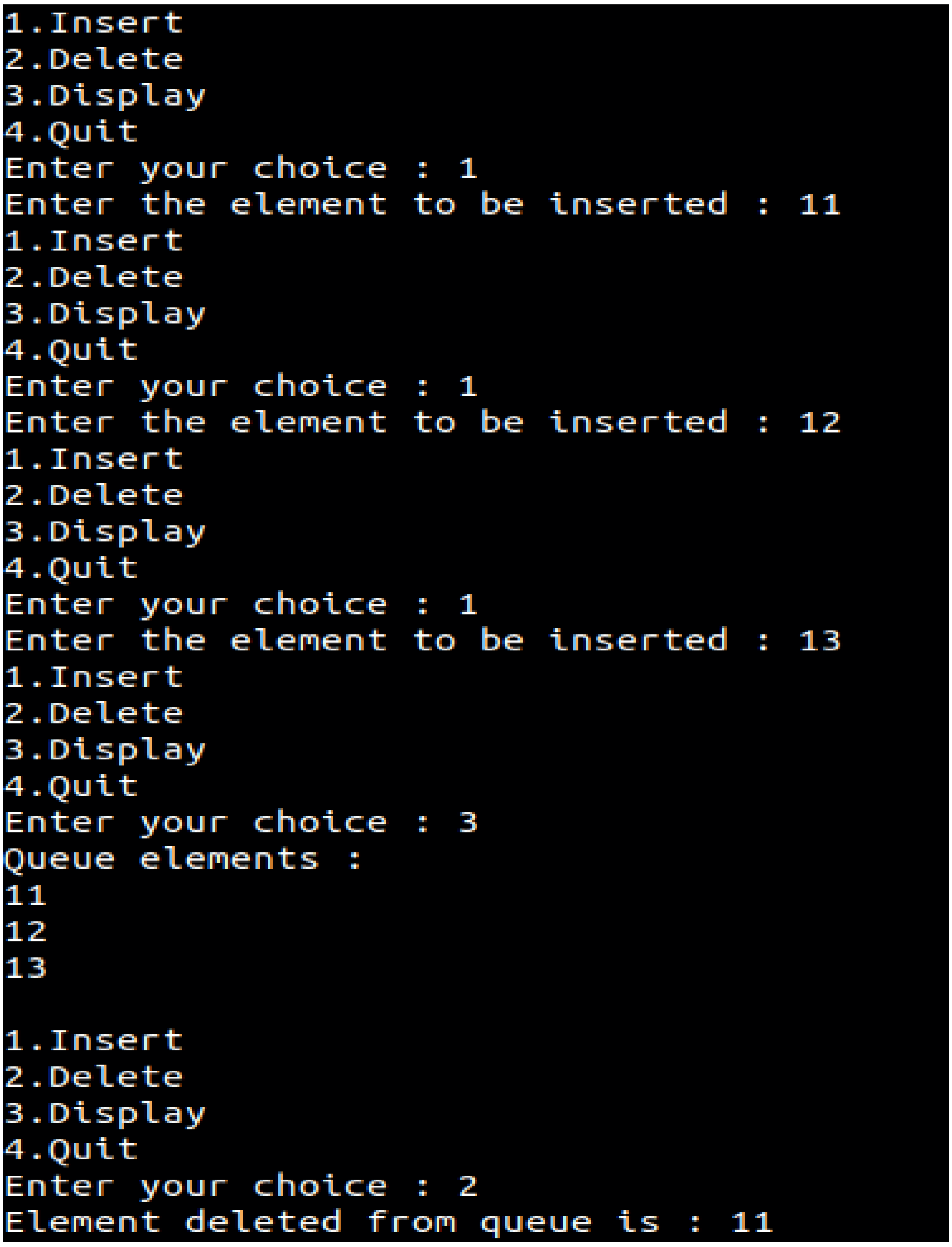
```
do
{
printf("\n enter your choice : ");
scanf("%d",&a);
switch(a)
{
case 1: printf("\n enter the item : ");
        scanf("%d",&item);
        if(rear>25)
        printf("\n queue overflow");
        else
        {
        rear++;
        q[r]=item;
        }
        printf("\n queue is: ");
        for(i=f+1;i<=r;i++)
        printf("%d\t",q[i]);
        break;
 case 2: f++;
        if(f==r)
        printf("\n queue underflow");
        else
        {f++;
        x=q[front];
        c++;
        }
        if(f!=r)
       {
```

```
        printf("deleted item is %d\n",x);
        printf("queue is : ");
        for(i=f;i<=r;i++)
        printf("%d\t",q[i]);
        }
        else
        printf("queue is empty");
        break;
case 3: if(f!=r)
            {printf("\n queue is :");
            for(i=f;i<=r;i++)
            printf("%d\t",q[i]);
            }
            else
            printf("queue is empty");
            break;
default: exit(0);
}
printf("\n do you want to continue? if yes-y exit-n : ");
scanf("%s",&k);
}while(k=='y');
}
```

RESULT:

Program executed successfully and output verified.

OUTPUT:



```
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 1
Enter the element to be inserted : 11
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 1
Enter the element to be inserted : 12
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 1
Enter the element to be inserted : 13
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 3
Queue elements :
11
12
13

1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 2
Element deleted from queue is : 11
```

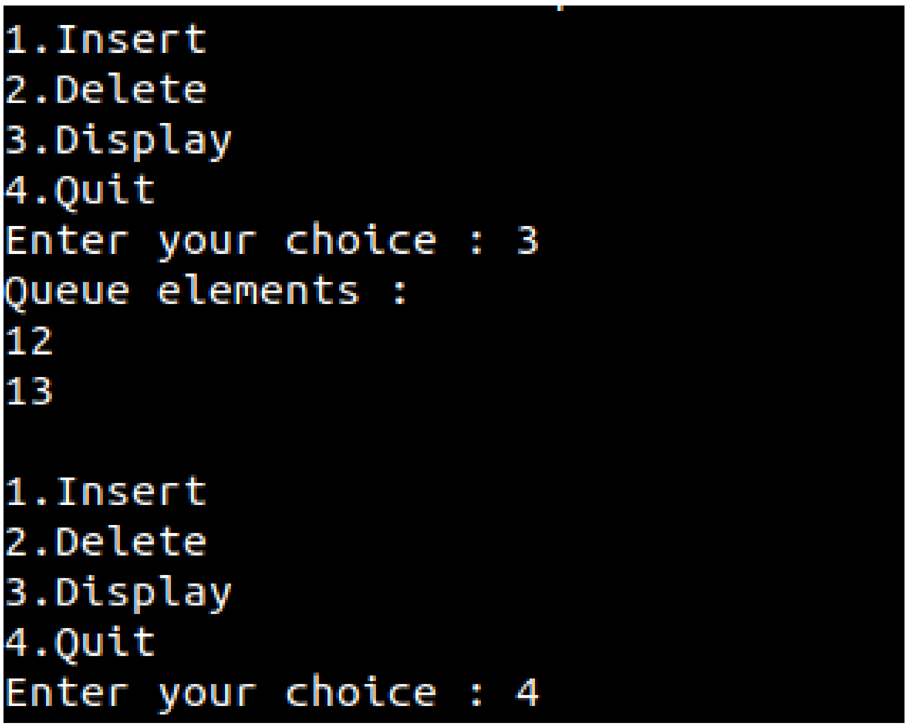
LINEAR CIRCULAR QUEUE OPERATIONS

AIM: Write a menu driven program to perform linear circular queue operations.

PROGRAM CODE:

```
#include<stdio.h>
# define MAX 5
int cqueue_arr[MAX];
int front = -1;
int rear = -1;
void insert(int item)
{
if((front == 0 && rear == MAX-1) || (front == rear+1))
{
printf("Queue Overflow \n");
return;
}
if(front == -1)
{
front = 0;
rear = 0;
}
else
{
if(rear == MAX-1)
rear = 0;
else
rear = rear+1;
}
```

OUTPUT:



```
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 3
Queue elements :
12
13

1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 4
```

```
cqueue_arr[rear] = item ;
}
void deletion()
{
if(front == -1)
{
printf("Queue Underflow \n");
return ;
}
printf("Element deleted from queue is : %d\n",cqueue_arr[front]);
if(front == rear)
{
front = -1;
rear=-1;
}
else
{
if(front == MAX-1)
front = 0;
else
front = front+1;
}
}
void display()
{
int front_pos = front,rear_pos = rear;
if(front == -1)
{
printf("Queue is empty \n");
```

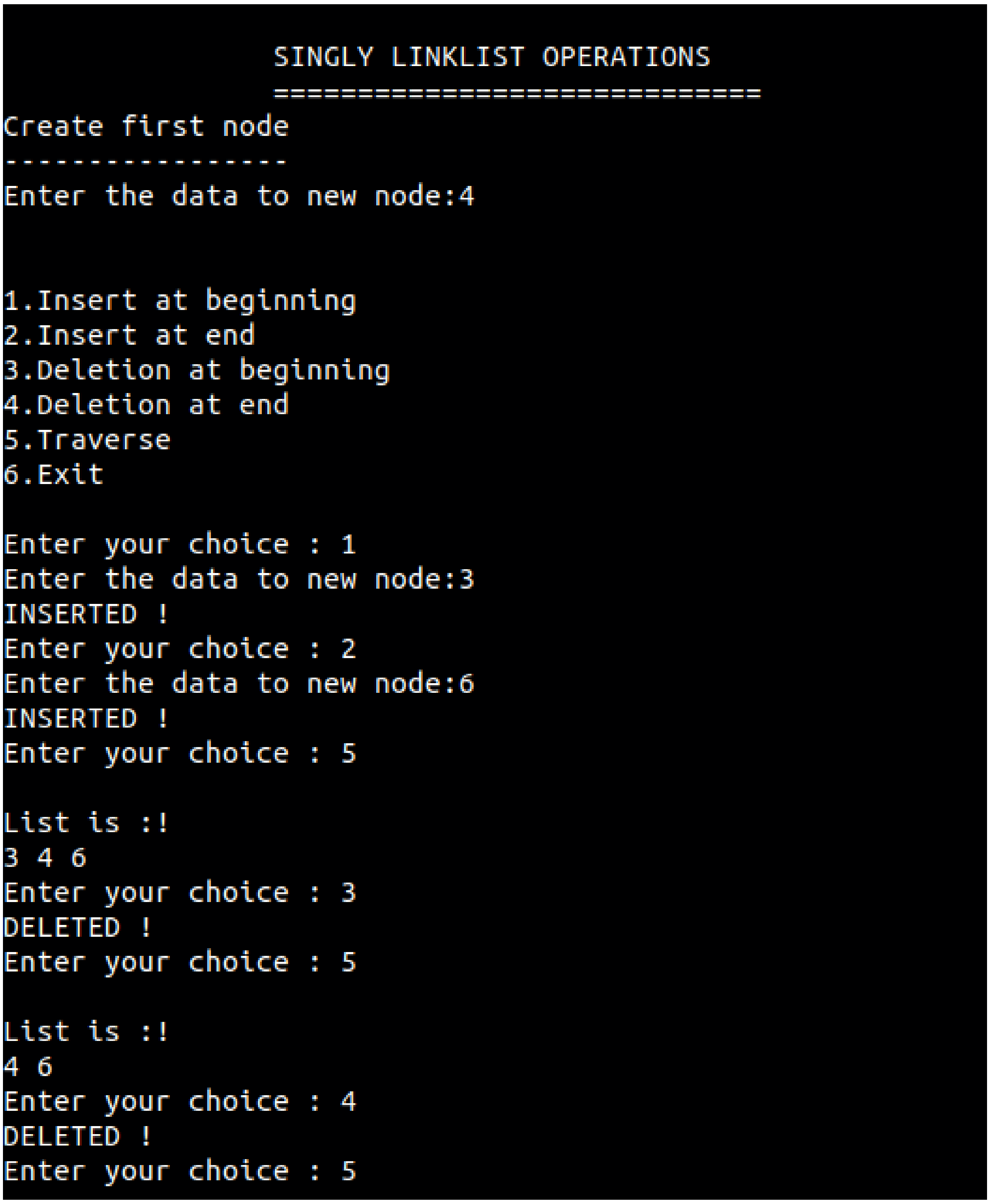
```
return;
}
printf("Queue elements :\n");
if( front_pos <= rear_pos )
while(front_pos <= rear_pos)
{
printf("%d\n",cqueue_arr[front_pos]);
front_pos++;
}
else
{
while(front_pos <= MAX-1)
{
printf("%d ",cqueue_arr[front_pos]);
front_pos++;
}
front_pos = 0;
while(front_pos <= rear_pos)
{
printf("%d\n",cqueue_arr[front_pos]);
front_pos++;
}
}
printf("\n");
}
int main()
{
int choice,item;
do
```

```
{
printf("1.Insert\n");
printf("2.Delete\n");
printf("3.Display\n");
printf("4.Quit\n");
printf("Enter your choice : ");
scanf("%d",&choice);
switch(choice)
{
case 1 :printf("Enter the element to be inserted : ");
            scanf("%d", &item);
            insert(item);
           break;
case 2 :deletion();
             break;
case 3:display();
            break;
case 4: break;
default: printf("Wrong choice\n");
}
}while(choice!=4);
return 0;
}
```

RESULT:

Program executed successfully and output verified.

OUTPUT:



```
                SINGLY LINKLIST OPERATIONS
                =============================
Create first node
-----------------
Enter the data to new node:4


1.Insert at beginning
2.Insert at end
3.Deletion at beginning
4.Deletion at end
5.Traverse
6.Exit

Enter your choice : 1
Enter the data to new node:3
INSERTED !
Enter your choice : 2
Enter the data to new node:6
INSERTED !
Enter your choice : 5

List is :!
3 4 6
Enter your choice : 3
DELETED !
Enter your choice : 5

List is :!
4 6
Enter your choice : 4
DELETED !
Enter your choice : 5
```

INSERTION & DELETION ON SINGLY LINKED LIST

AIM: Write a menu driven program to create a singly linked list and to perform insertion at the beginning, at the end of the linked list, deletion of a node from the beginning and end of the list and to traverse the list.

PROGRAM CODE:

```
#include<stdio.h>
#include<stdlib.h>
typedef struct linklist
{int data;
struct linklist *next;
}node;
void create_node();
void traverse();
void insert_begi();
void insert_end();
void dele_begi();
void dele_end();
node *ptr,*t,*ptr1;
void main()
{int choice;
printf("\n\n\t\tSINGLY LINKLIST OPERATIONS\n");
printf("\t\t==============================\n");
printf("Create first node\n");
printf("-----------------\n");
create_node();
```

```
printf("\n\n1.Insert at beginning\n");
printf("2.Insert at end  \n");
printf("3.Deletion at beginning\n");
printf("4.Deletion at end\n");
printf("5.Traverse\n");
printf("6.Exit\n");
do
{printf("\nEnter your choice : ");
scanf("%d",&choice);
switch(choice)
{case 1 :insert_begi();
            break;
case 2 :insert_end();
            break;
case 3 :dele_begi();
            break;
case 4 :dele_end();
            break;
case 5 :traverse();
             break;
case 6:exit(0);
           break;
default: printf("Wrong choice\n");
}
}while(choice!=7 || choice==0);
}
void create_node()
```

```
{ptr=(node*)malloc(sizeof(node));
t=ptr;
printf("Enter the data to new node:");
scanf("%d",&ptr->data);
ptr->next=NULL;
}
void traverse()
{ptr=t;
if(ptr==NULL)
{printf("\nList is empty !\n");
}
else
{printf("\nList is :!\n");
while(ptr!=NULL)
{printf("%d ",ptr->data);
ptr=ptr->next;
}
}
}
void insert_begi()
{ptr1=(node*)malloc(sizeof(node));
printf("Enter the data to new node:");
scanf("%d",&ptr1->data);
ptr1->next=t;
t=ptr1;
printf("INSERTED !");
}
```

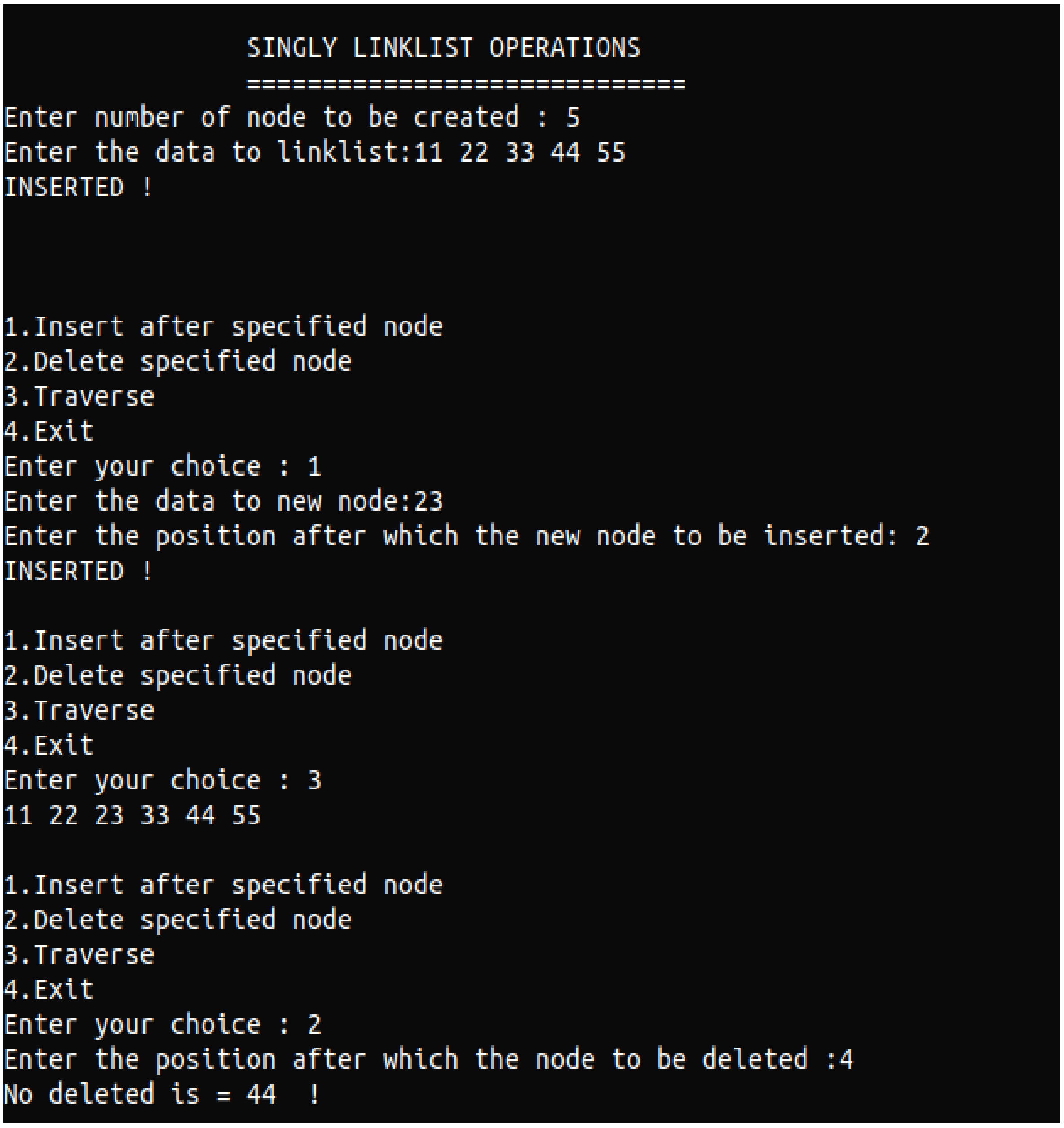
```
void insert_end()
{ptr1=(node*)malloc(sizeof(node));
printf("Enter the data to new node:");
scanf("%d",&ptr1->data);
ptr=t;
while(ptr->next!=NULL)
{ptr=ptr->next;
}
ptr->next=ptr1;
ptr1->next=NULL;
printf("INSERTED !");
}
void dele_begi()
{ptr=t;
if(ptr==NULL)
{printf("\nList is empty !\n");
}
else
{t=ptr->next;
free(ptr);
}
printf("DELETED !");
}
void dele_end()
{ptr=t;
if(ptr==NULL)
{printf("\nList is empty !\n");
```

```
}
else
{
while(ptr->next!=NULL)
{
ptr1=ptr;
ptr=ptr->next;
}
free(ptr);
ptr1->next=NULL;
}
printf("DELETED !");
}
```

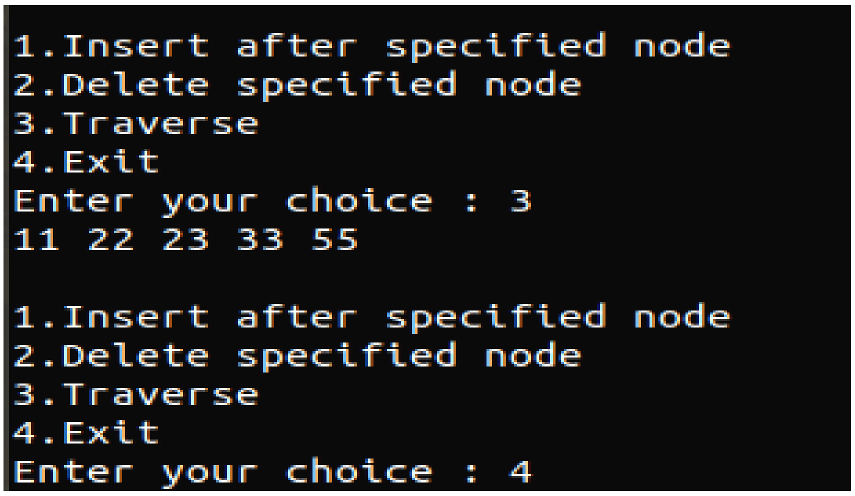
RESULT:

Program executed successfully and output verified.

OUTPUT:



```
                SINGLY LINKLIST OPERATIONS
                =============================
Enter number of node to be created : 5
Enter the data to linklist:11 22 33 44 55
INSERTED !



1.Insert after specified node
2.Delete specified node
3.Traverse
4.Exit
Enter your choice : 1
Enter the data to new node:23
Enter the position after which the new node to be inserted: 2
INSERTED !

1.Insert after specified node
2.Delete specified node
3.Traverse
4.Exit
Enter your choice : 3
11 22 23 33 44 55

1.Insert after specified node
2.Delete specified node
3.Traverse
4.Exit
Enter your choice : 2
Enter the position after which the node to be deleted :4
No deleted is = 44  !
```



```
1.Insert after specified node
2.Delete specified node
3.Traverse
4.Exit
Enter your choice : 3
11 22 23 33 55

1.Insert after specified node
2.Delete specified node
3.Traverse
4.Exit
Enter your choice : 4
```

INSERTION & DELETION AFTER SPECIFIED NODE ON SINGLY LINKED LIST

AIM: Write a menu driven program to create a singly linked list and insert and delete a node after the specified node.

PROGRAM CODE:

```
#include<stdio.h>
#include<stdlib.h>
typedef struct linklist
{
int data;
struct linklist *next;
}node;
int n;
node *ptr,*t,*ptr1;
void traverse();
void insert_speci();
void delete_speci();
void main()
{
int choice,i;
printf("\n\n\t\tSINGLY LINKLIST OPERATIONS\n");
printf("\t\t=============================\n");
printf("Enter number of node to be created : ");
scanf("%d",&n);
ptr=(node*)malloc(sizeof(node));
t=ptr;
```

```
printf("Enter the data to linklist:");
scanf("%d",&ptr->data);
for(i=1;i<n;i++)
{ptr->next=(node*)malloc(sizeof(node));
ptr= ptr->next;
scanf("%d",&ptr->data);
}
ptr->next=NULL;
printf("INSERTED !\n\n");
do
{printf("\n\n1.Insert after specified node\n");
printf("2.Delete specified node\n");
printf("3.Traverse\n");
printf("4.Exit\n");
printf("Enter your choice : ");
scanf("%d",&choice);
switch(choice)
{case 1 :insert_speci();
             break;
case 2 :delete_speci();
            break;
case 3 :traverse();
            break;
case 4:exit(0);
           break;
default:printf("Wrong choice\n");
}
}while(choice!=7 || choice==0);
}
```

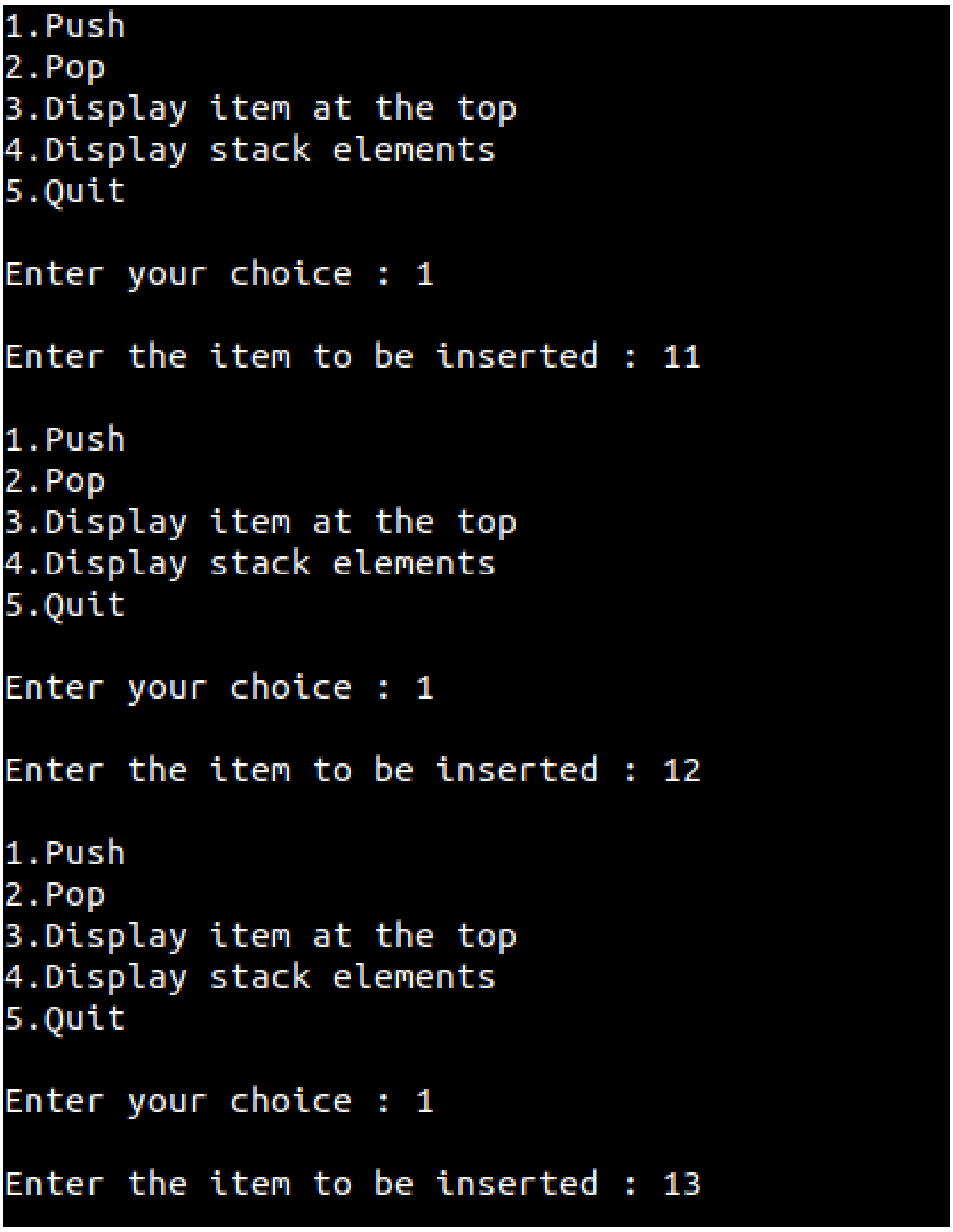
```
void traverse()
{ptr=t;
if(ptr==NULL)
{printf("\nList is empty !\n");
}
else
{while(ptr!=NULL)
{printf("%d ",ptr->data);
ptr=ptr->next;
}
}
}
void insert_speci()
{
int speci_pos,i;
ptr1=(node*)malloc(sizeof(node));
printf("Enter the data to new node:");
scanf("%d",&ptr1->data);
printf("Enter the position after which the new node to be inserted: ");
scanf("%d",&speci_pos);
ptr=t;
if(speci_pos<=n)
{
for(i=1;i<speci_pos;i++)
{ptr=ptr->next;
}
ptr1->next=ptr->next;
ptr->next=ptr1;
printf("INSERTED !");
```

```
}
}
void delete_speci()
{
int dele_node,i;
printf("Enter the position after which the node to be deleted :");
scanf("%d",&dele_node);
if(dele_node<=n)
{
ptr=t;
for(i=1;i<=dele_node;i++)
{
ptr1=ptr;
 ptr=ptr->next;
}
printf("No deleted is = %d  !",ptr->data);
ptr1->next=ptr->next;
free(ptr);
}
else
{
 printf("key not found");
}
}
```

RESULT:

Program executed successfully and output verified.

OUTPUT:


```
1.Push
2.Pop
3.Display item at the top
4.Display stack elements
5.Quit

Enter your choice : 1

Enter the item to be inserted : 11

1.Push
2.Pop
3.Display item at the top
4.Display stack elements
5.Quit

Enter your choice : 1

Enter the item to be inserted : 12

1.Push
2.Pop
3.Display item at the top
4.Display stack elements
5.Quit

Enter your choice : 1

Enter the item to be inserted : 13
```

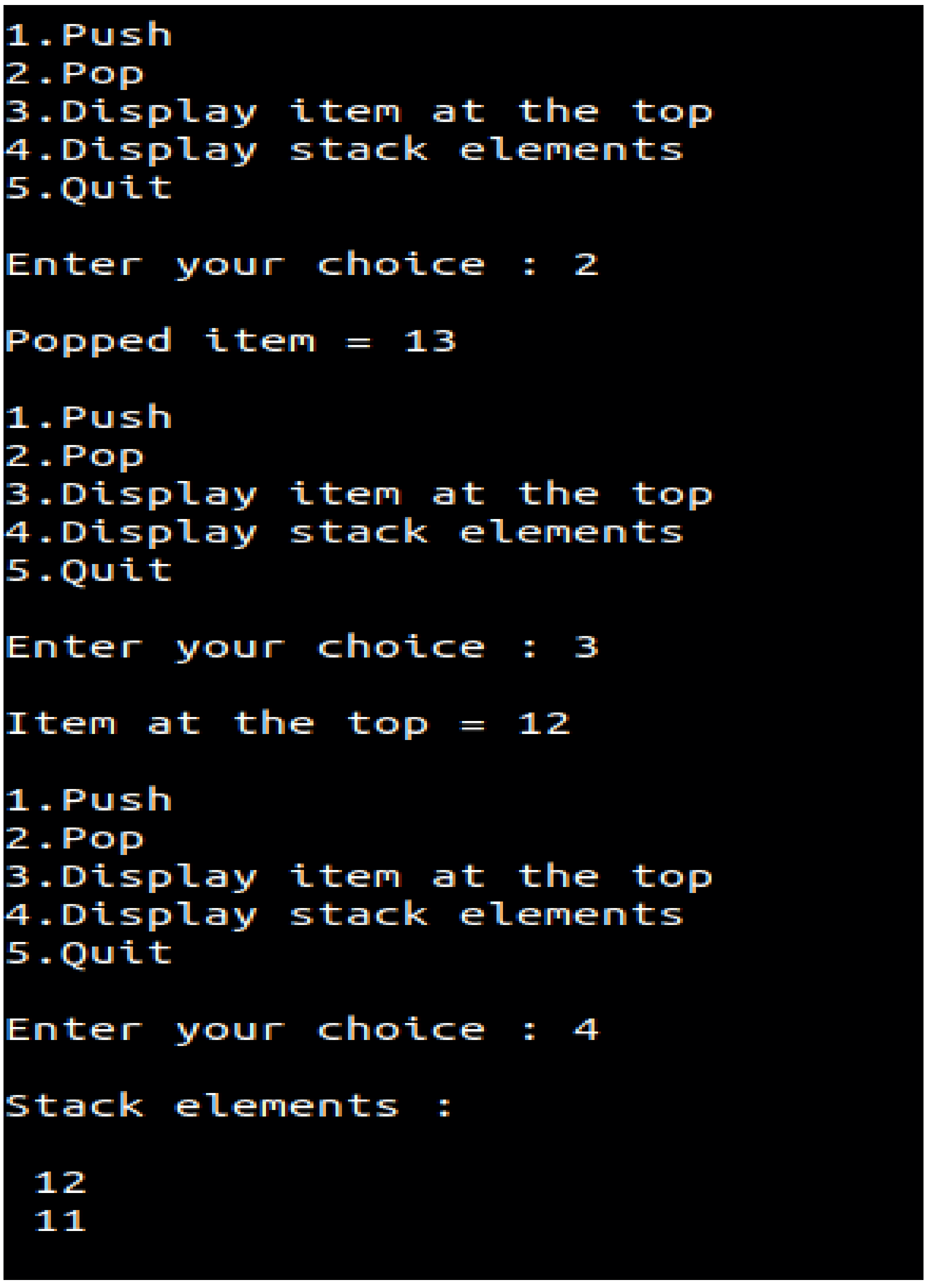
LINKED STACK OPERATIONS

AIM: Write a menu driven program to perform linked stack operations.

PROGRAM CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{int info;
struct node *link;
}*top=NULL;
void push(int item);
int pop();
int peek();
int isEmpty();
void display();
int main()
{int choice,item;
while(1)
{
printf("\n1.Push\n");
printf("2.Pop\n");
printf("3.Display item at the top\n");
printf("4.Display stack elements\n");
printf("5.Quit\n");
printf("\nEnter your choice : ") ;
scanf("%d", &choice);
switch(choice)
```

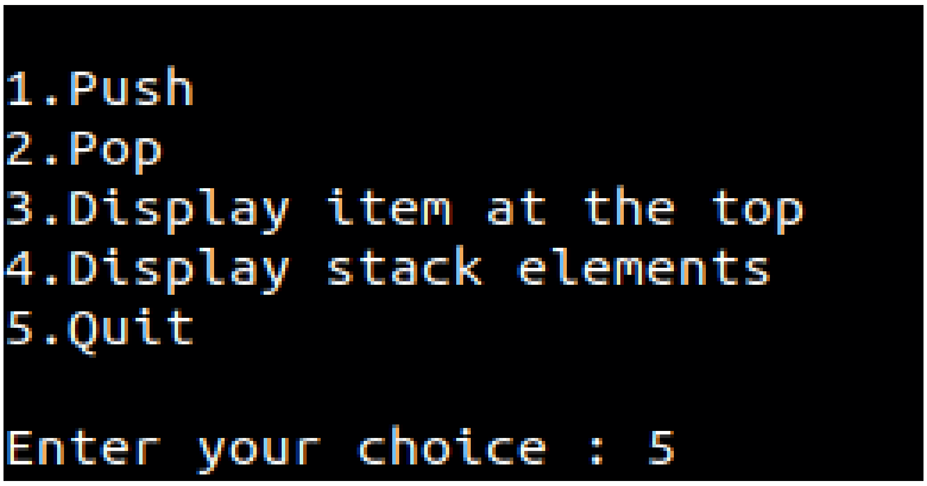
OUTPUT:



```
1.Push
2.Pop
3.Display item at the top
4.Display stack elements
5.Quit

Enter your choice : 2

Popped item = 13

1.Push
2.Pop
3.Display item at the top
4.Display stack elements
5.Quit

Enter your choice : 3

Item at the top = 12

1.Push
2.Pop
3.Display item at the top
4.Display stack elements
5.Quit

Enter your choice : 4

Stack elements :

 12
 11
```

```
{case 1: printf("\nEnter the item to be inserted : ");
              scanf("%d",&item);
               push(item);
               break;
case 2: item=pop();
             printf("\nPopped item = %d\n",item);
             break;
case 3: printf("\nItem at the top = %d\n",peek());
            break;
case 4: display();
            break;
case 5: exit(1);
default : printf("\nWrong choice\n");
}
}
}
void push(int item)
{struct node *tmp;
tmp=(struct node *)malloc(sizeof(struct node));
if(tmp==NULL)
{printf("\nStack Overflow\n");
return;
}
tmp->info=item;
tmp->link=top;
top=tmp;
}
int pop()
{struct node *tmp;
```

OUTPUT:



```
1.Push
2.Pop
3.Display item at the top
4.Display stack elements
5.Quit

Enter your choice : 5
```

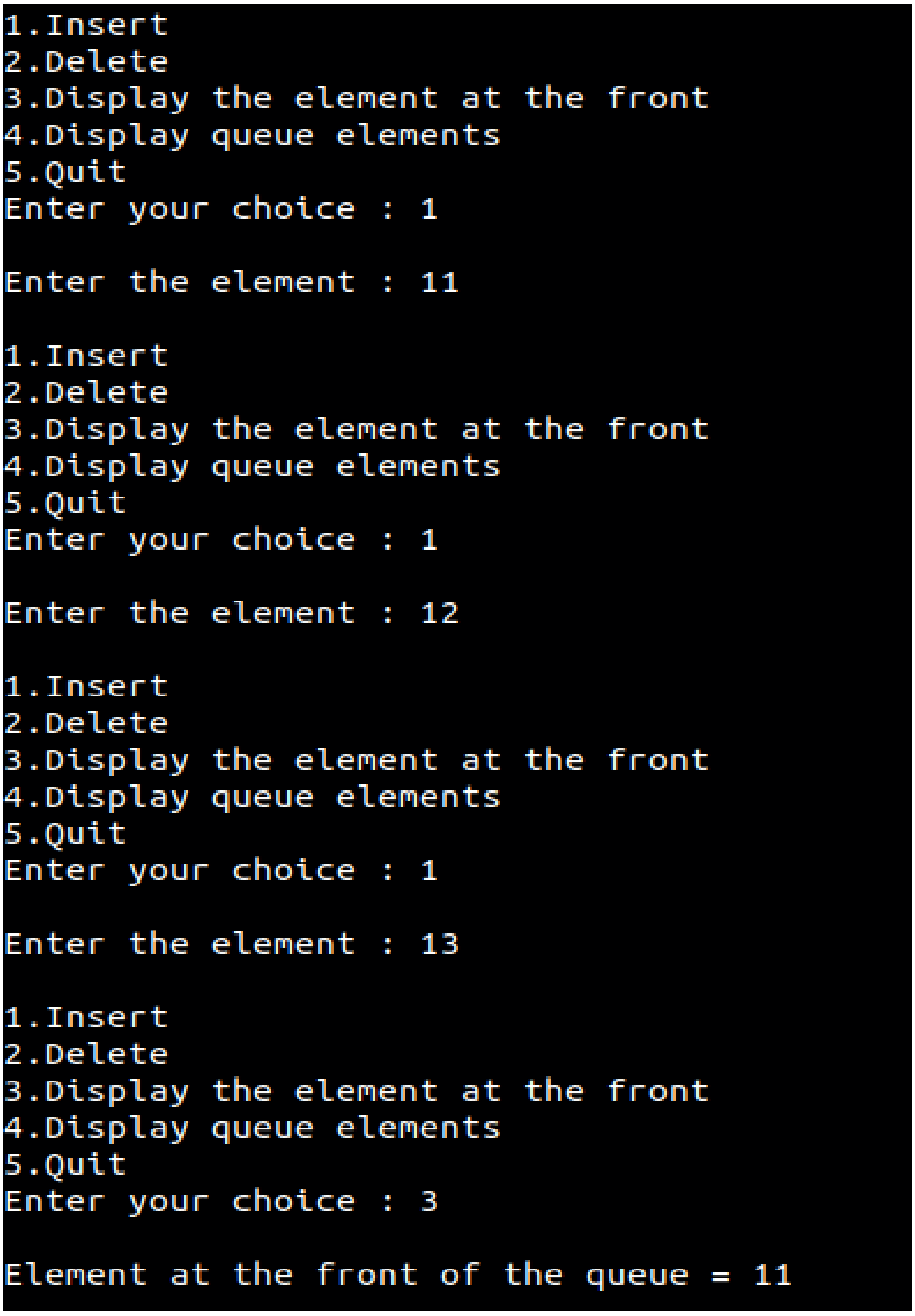
```
int item;
if( isEmpty() )
{printf("\nStack Underflow\n");
 exit(1);
}
tmp=top;
item=tmp->info;
top=top->link;
free(tmp);
return item;
}
int peek()
{if( isEmpty() )
{printf("\nStack Underflow\n");
 exit(1);
}
return top->info;
}
int isEmpty()
{if(top == NULL)
return 1;
else
return 0;
}
void display()
{struct node *ptr;
ptr=top;
if(isEmpty())
{printf("\nStack is empty\n");
```

```
return;
}
printf("\nStack elements :\n\n");
while(ptr!=NULL)
{printf(" %d\n",ptr->info);
ptr=ptr->link;
}
printf("\n");
}
```

RESULT:

Program executed successfully and output verified.

OUTPUT:



```
1.Insert
2.Delete
3.Display the element at the front
4.Display queue elements
5.Quit
Enter your choice : 1

Enter the element : 11

1.Insert
2.Delete
3.Display the element at the front
4.Display queue elements
5.Quit
Enter your choice : 1

Enter the element : 12

1.Insert
2.Delete
3.Display the element at the front
4.Display queue elements
5.Quit
Enter your choice : 1

Enter the element : 13

1.Insert
2.Delete
3.Display the element at the front
4.Display queue elements
5.Quit
Enter your choice : 3

Element at the front of the queue = 11
```

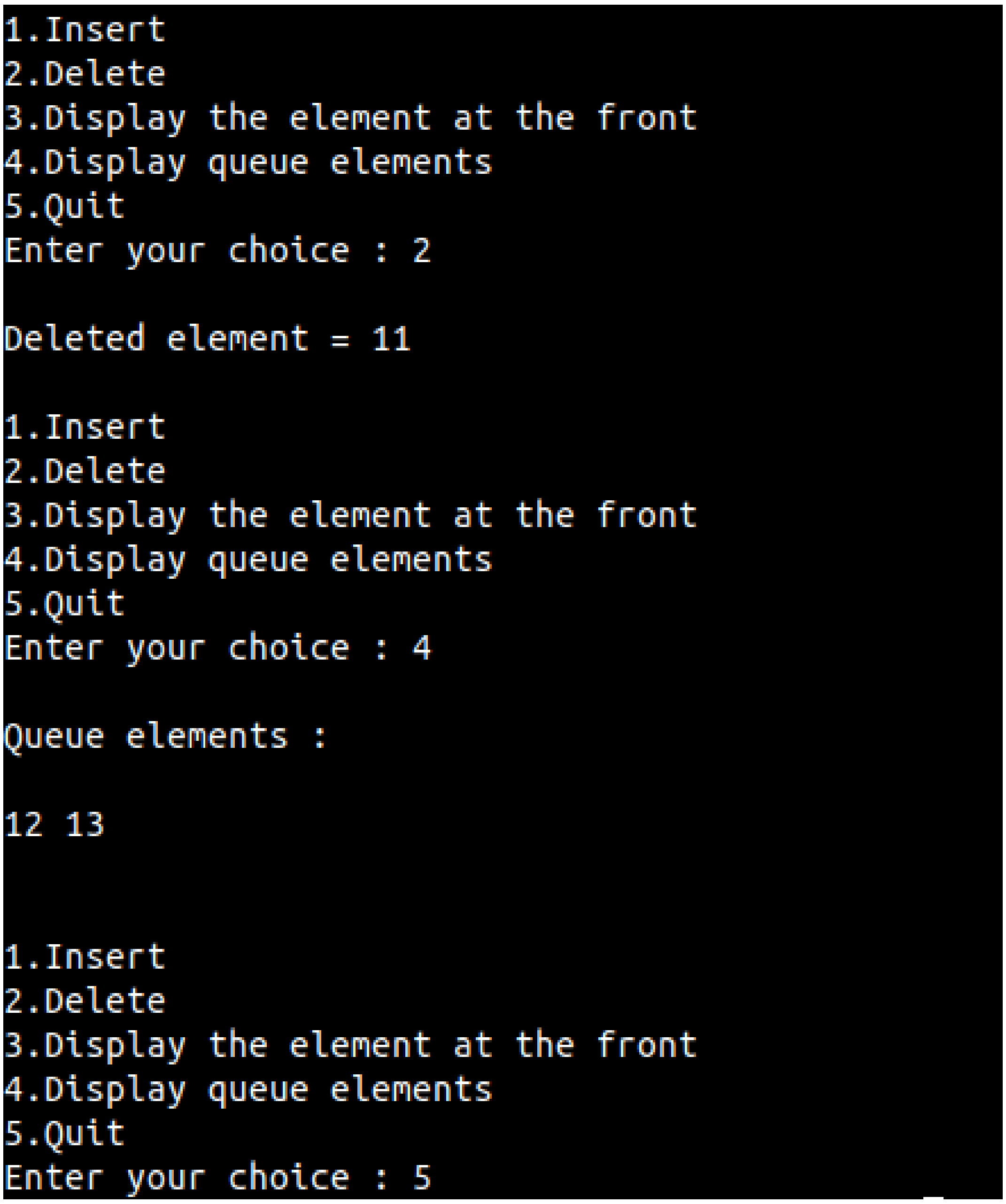
LINKED QUEUE OPERATIONS

AIM: Write a menu driven program to perform linked queue operations

PROGRAM CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int info;
struct node *link;
}*front=NULL,*rear=NULL;
void insert(int item);
int del();
int peek();
int isEmpty();
void display();
int main()
{
int choice,item;
while(1)
{
printf("\n1.Insert");
printf("\n2.Delete");
printf("\n3.Display the element at the front");
printf("\n4.Display queue elements");
printf("\n5.Quit");
printf("\nEnter your choice : ");
scanf("%d", &choice);
```

OUTPUT:



```
1.Insert
2.Delete
3.Display the element at the front
4.Display queue elements
5.Quit
Enter your choice : 2

Deleted element = 11

1.Insert
2.Delete
3.Display the element at the front
4.Display queue elements
5.Quit
Enter your choice : 4

Queue elements :

12 13


1.Insert
2.Delete
3.Display the element at the front
4.Display queue elements
5.Quit
Enter your choice : 5
```

```
switch(choice)
{case 1: printf("\nEnter the element : ");
             scanf("%d",&item);
             insert(item);
             break;
case 2: printf("\nDeleted element = %d\n",del());
            break;
case 3: printf("\nElement at the front of the queue = %d\n", peek() );
            break;
case 4: display();
            break;
case 5: exit(1);
default : printf("\nWrong choice\n");
}
}
}
void insert(int item)
{struct node *tmp;
tmp=(struct node *)malloc(sizeof(struct node));
if(tmp==NULL)
{printf("\nMemory not available\n");
 return;
}
tmp->info = item;
tmp->link=NULL;
if(front==NULL)
front=tmp;
else
rear->link=tmp;
```

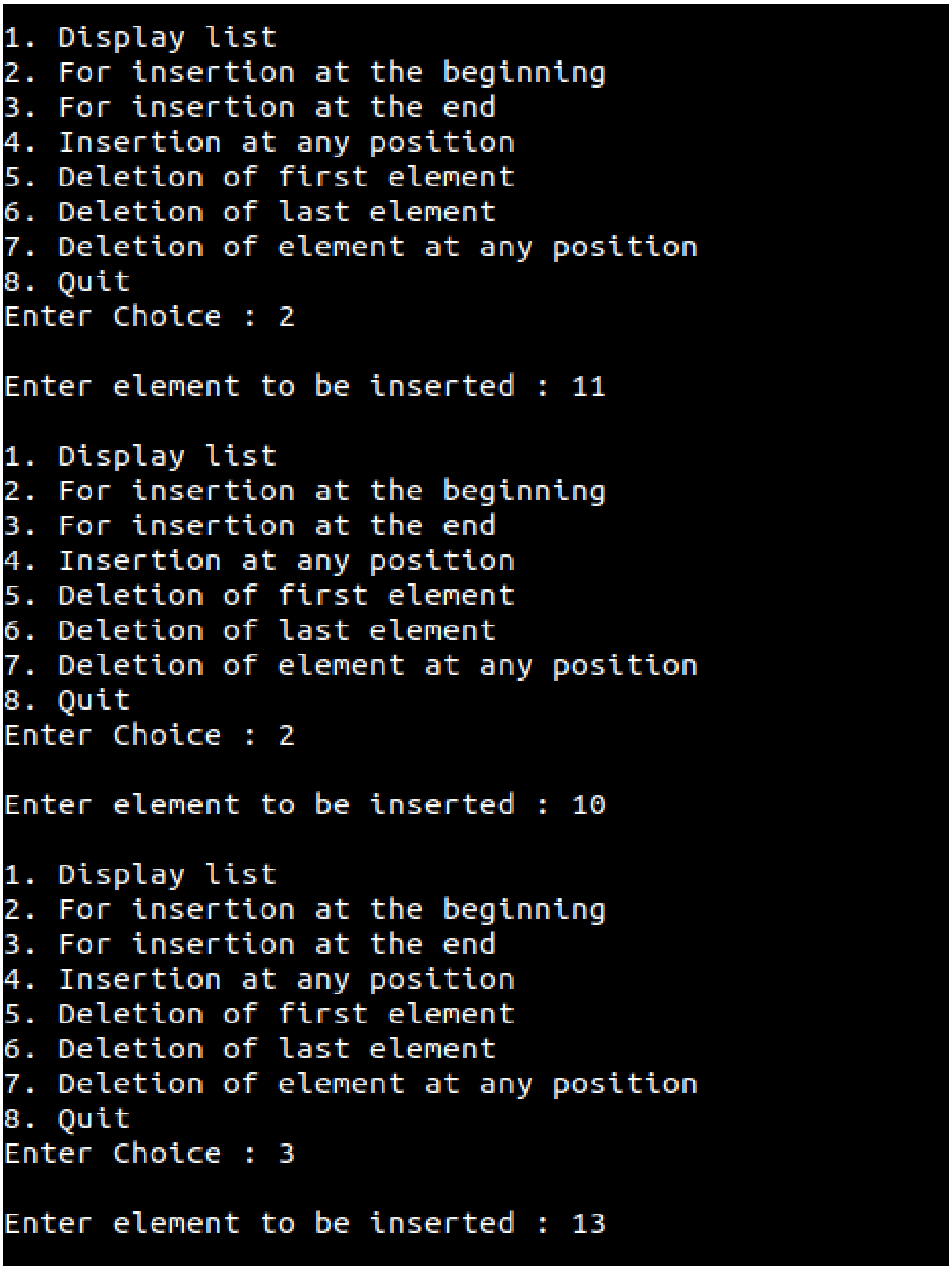
```
rear=tmp;
}
int del()
{struct node *tmp;
int item;
if( isEmpty( ) )
{
printf("\nQueue Underflow\n");
exit(1);
}
tmp=front;
item=tmp->info;
front=front->link;
free(tmp);
return item;
}
int peek()
{if( isEmpty( ) )
{
printf("\nQueue Underflow\n");
exit(1);
}
return front->info;
}
int isEmpty()
{if(front==NULL)
return 1;
else
return 0;
```

```
}
void display()
{
struct node *ptr;
ptr=front;
if(isEmpty()
{
printf("\nQueue is empty\n");
return;
}
printf("\nQueue elements :\n\n");
while(ptr!=NULL)
{
printf("%d ",ptr->info);
 ptr=ptr->link;
}
printf("\n\n");
}
```

RESULT:

Program executed successfully and output verified.

OUTPUT:



```
1. Display list
2. For insertion at the beginning
3. For insertion at the end
4. Insertion at any position
5. Deletion of first element
6. Deletion of last element
7. Deletion of element at any position
8. Quit
Enter Choice : 2

Enter element to be inserted : 11

1. Display list
2. For insertion at the beginning
3. For insertion at the end
4. Insertion at any position
5. Deletion of first element
6. Deletion of last element
7. Deletion of element at any position
8. Quit
Enter Choice : 2

Enter element to be inserted : 10

1. Display list
2. For insertion at the beginning
3. For insertion at the end
4. Insertion at any position
5. Deletion of first element
6. Deletion of last element
7. Deletion of element at any position
8. Quit
Enter Choice : 3

Enter element to be inserted : 13
```

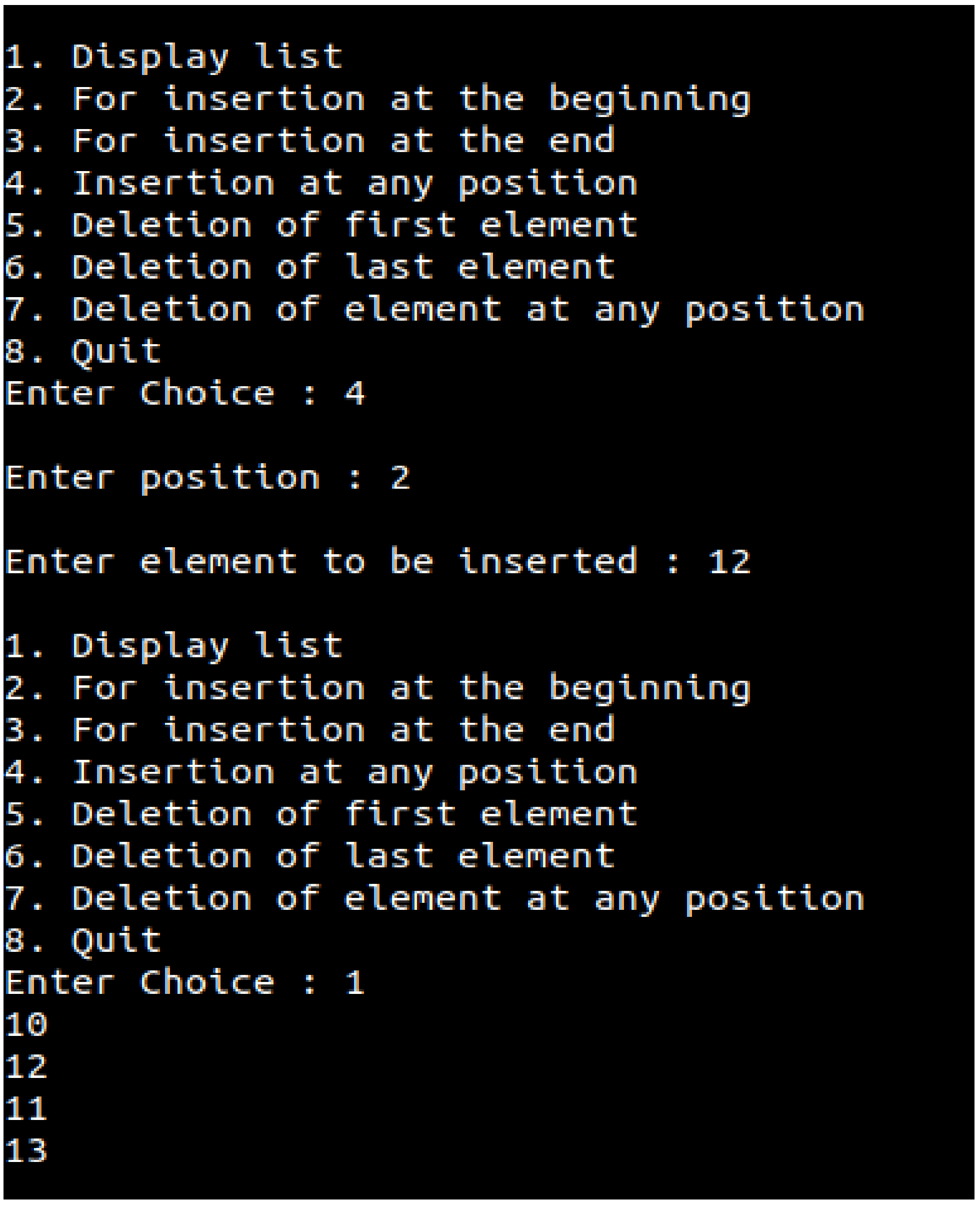
DOUBLY LINKED LIST MANIPULATIONS

AIM: Write a menu driven program to perform doubly linked list manipulations.

PROGRAM CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct node
{int info;
struct node *prev, *next;
};
struct node* start = NULL;
void traverse()
{if (start == NULL)
{printf("\nList is empty\n");
return;
}
struct node* temp;
temp = start;
while (temp != NULL)
{printf("%d\n", temp->info);
temp = temp->next;
}}
void insertAtFront()
{int data;
struct node* temp;
temp = (struct node*)malloc(sizeof(struct node));
printf("\nEnter element to be inserted : ");
```

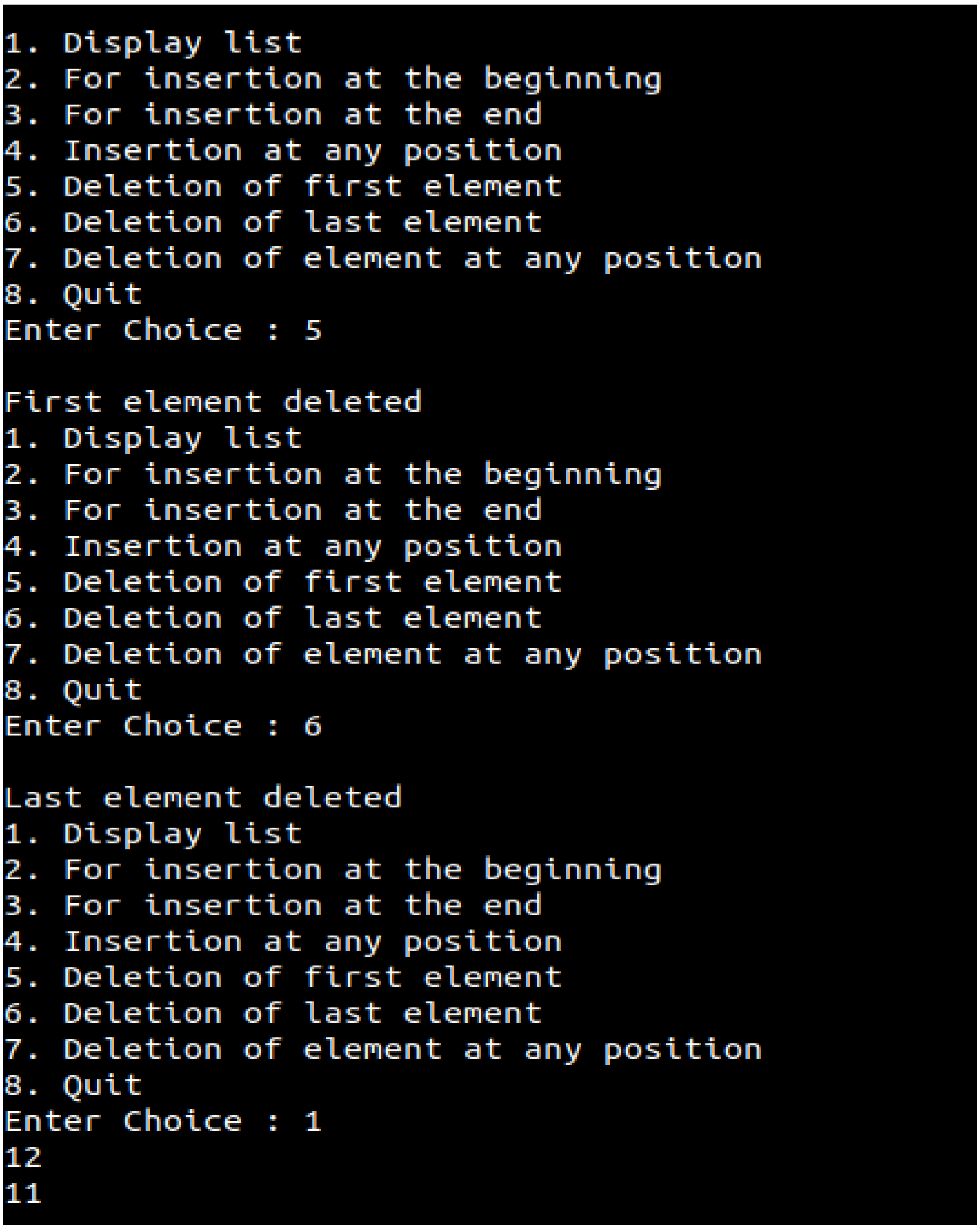
OUTPUT:



```
1. Display list
2. For insertion at the beginning
3. For insertion at the end
4. Insertion at any position
5. Deletion of first element
6. Deletion of last element
7. Deletion of element at any position
8. Quit
Enter Choice : 4

Enter position : 2

Enter element to be inserted : 12

1. Display list
2. For insertion at the beginning
3. For insertion at the end
4. Insertion at any position
5. Deletion of first element
6. Deletion of last element
7. Deletion of element at any position
8. Quit
Enter Choice : 1
10
12
11
13
```

```
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 element 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;
```

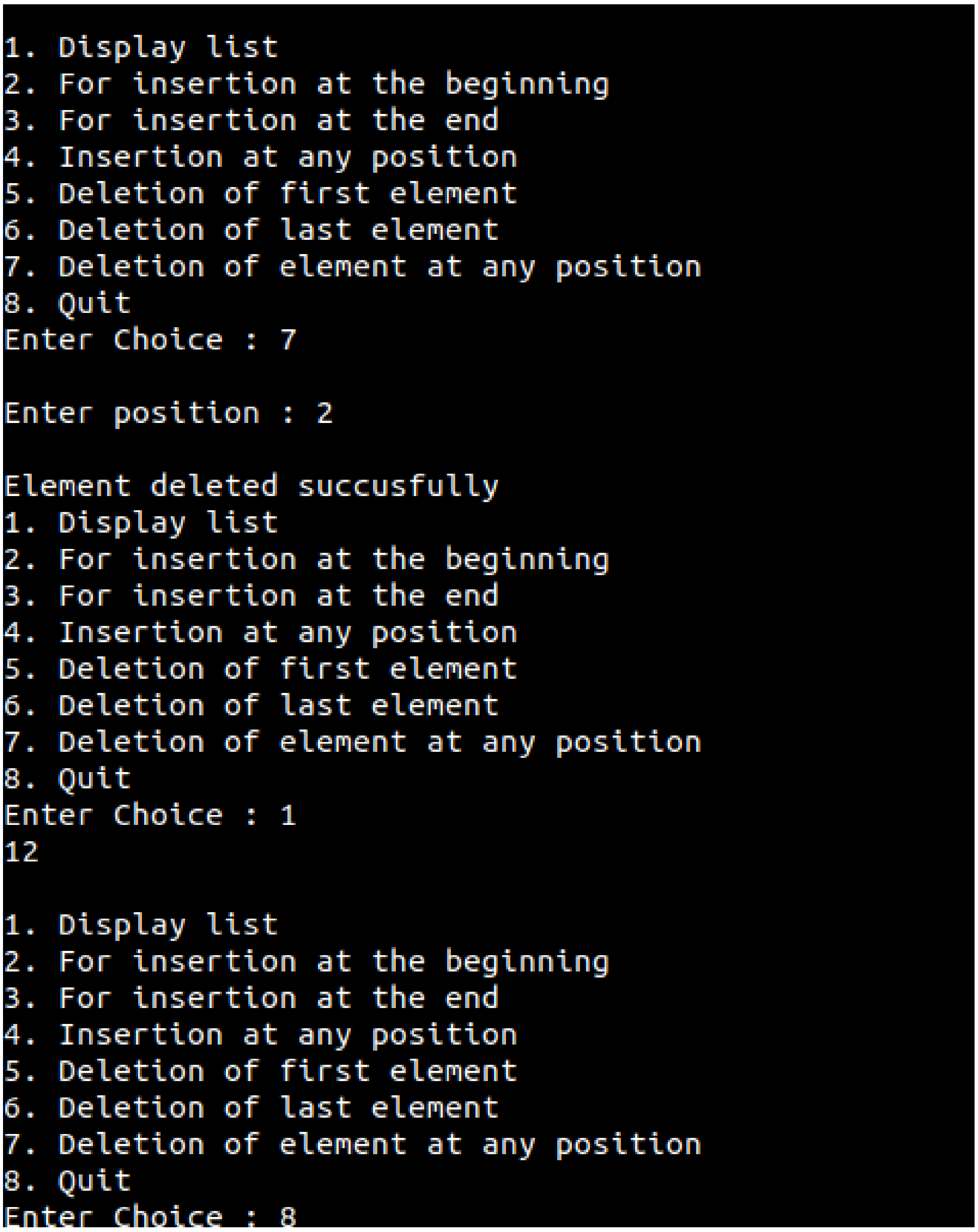
OUTPUT:



```
1. Display list
2. For insertion at the beginning
3. For insertion at the end
4. Insertion at any position
5. Deletion of first element
6. Deletion of last element
7. Deletion of element at any position
8. Quit
Enter Choice : 5

First element deleted
1. Display list
2. For insertion at the beginning
3. For insertion at the end
4. Insertion at any position
5. Deletion of first element
6. Deletion of last element
7. Deletion of element at any position
8. Quit
Enter Choice : 6

Last element deleted
1. Display list
2. For insertion at the beginning
3. For insertion at the end
4. Insertion at any position
5. Deletion of first element
6. Deletion of last element
7. Deletion of element at any position
8. Quit
Enter Choice : 1
12
11
```

```
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 element 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;
```

OUTPUT:



```
1. Display list
2. For insertion at the beginning
3. For insertion at the end
4. Insertion at any position
5. Deletion of first element
6. Deletion of last element
7. Deletion of element at any position
8. Quit
Enter Choice : 7

Enter position : 2

Element deleted succusfully
1. Display list
2. For insertion at the beginning
3. For insertion at the end
4. Insertion at any position
5. Deletion of first element
6. Deletion of last element
7. Deletion of element at any position
8. Quit
Enter Choice : 1
12

1. Display list
2. For insertion at the beginning
3. For insertion at the end
4. Insertion at any position
5. Deletion of first element
6. Deletion of last element
7. Deletion of element at any position
8. Quit
Enter Choice : 8
```

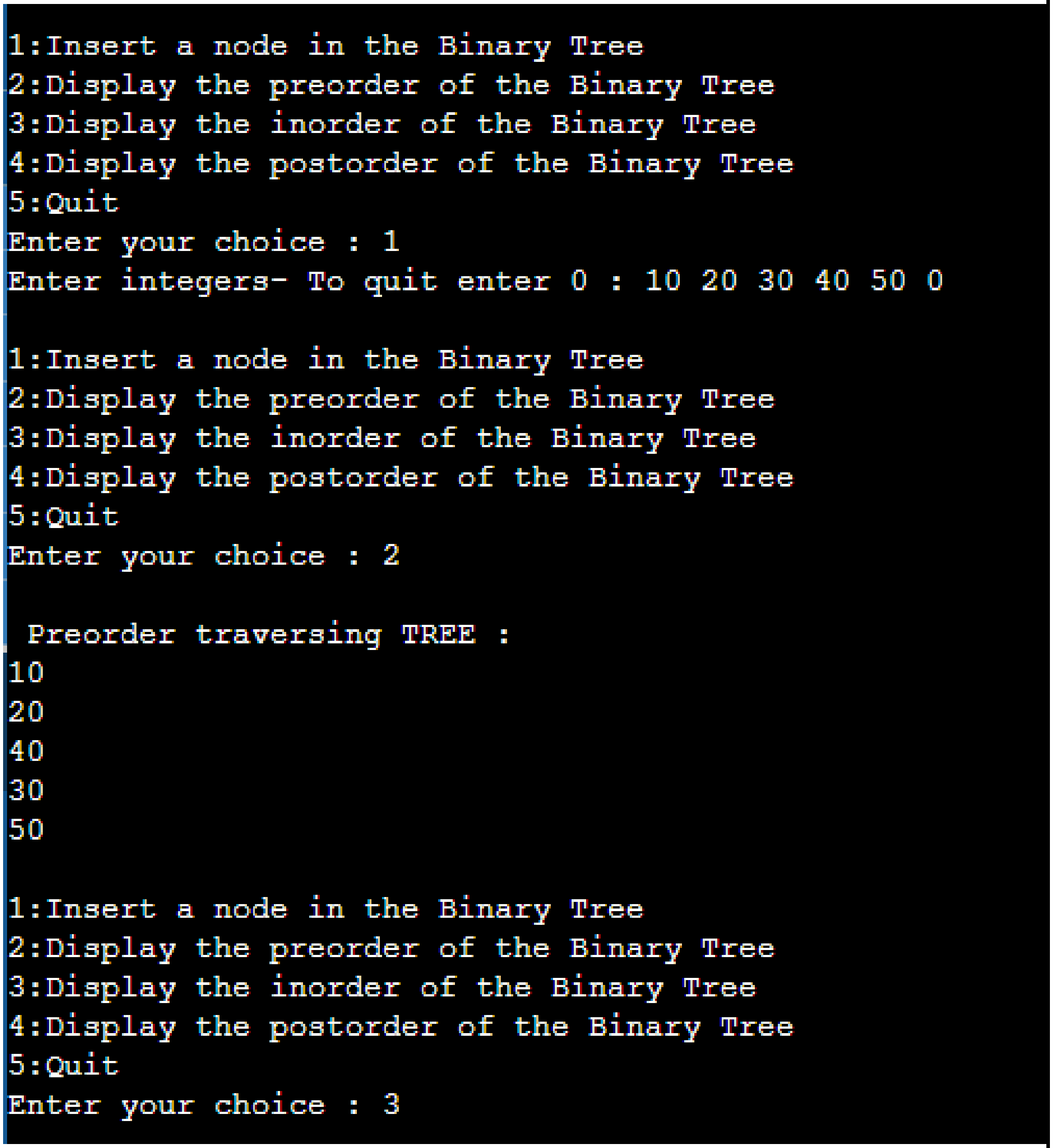
```
if (start == NULL)

printf("\nList is empty\n");

else

{temp = start;

start = start->next;

if (start != NULL)

start->prev = NULL;

printf("\nFirst element deleted");

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;

printf("\nLast element deleted");

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)
{if (start != NULL)
{start->prev = NULL;
}
deleteFirst();
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;
printf("\nElement deleted succusfully");
free(position);
}}
int main()
{int choice;
while (1)
{printf("\n1. Display list");
printf("\n2. For insertion at the beginning");
printf("\n3. For insertion at the end")
```

```
printf("\n4. Insertion at any position");
printf("\n5. Deletion of first element");
printf("\n6. Deletion of last element");
printf("\n7. Deletion of element at any position");
printf("\n8. Quit");
printf("\nEnter Choice : ");
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("\nIncorrect Choice\n");
        continue;
}}return 0;
}
```

RESULT: Program executed successfully and output verified.

OUTPUT:



```
1:Insert a node in the Binary Tree
2:Display the preorder of the Binary Tree
3:Display the inorder of the Binary Tree
4:Display the postorder of the Binary Tree
5:Quit
Enter your choice : 1
Enter integers- To quit enter 0 : 10 20 30 40 50 0

1:Insert a node in the Binary Tree
2:Display the preorder of the Binary Tree
3:Display the inorder of the Binary Tree
4:Display the postorder of the Binary Tree
5:Quit
Enter your choice : 2

 Preorder traversing TREE :
10
20
40
30
50

1:Insert a node in the Binary Tree
2:Display the preorder of the Binary Tree
3:Display the inorder of the Binary Tree
4:Display the postorder of the Binary Tree
5:Quit
Enter your choice : 3
```

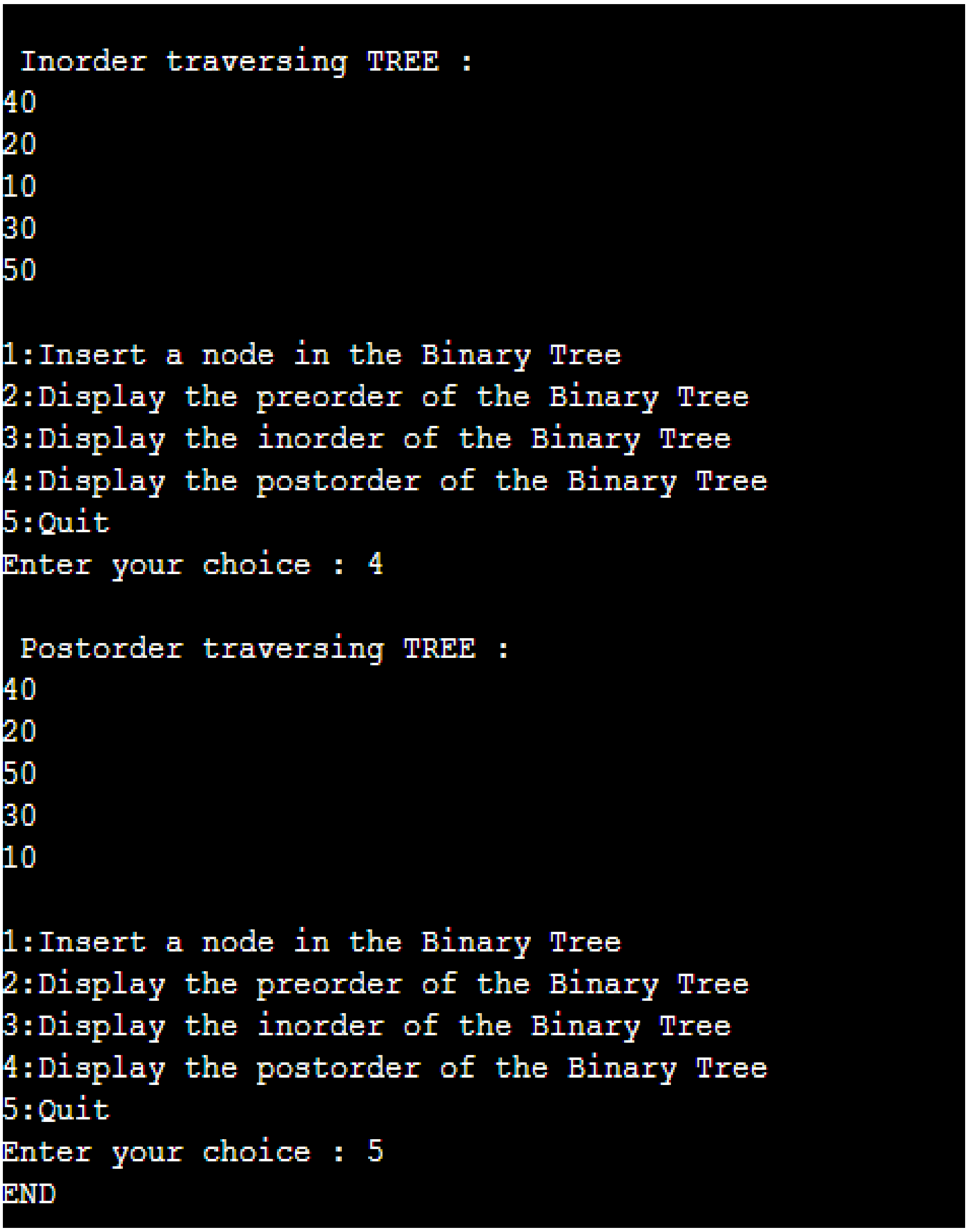
BINARY TREE TRAVERSALS

AIM: Write a menu driven program for implementing binary tree traversals.

PROGRAM CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{int num;
struct node *left;
struct node *right;
};
typedef struct node node;
node *root=NULL;
node *insert(node *tree,long num);
void preorder(node *tree);
void inorder(node *tree);
void postorder(node *tree);
int count = 1;
void main()
{long digit;
int s;
do
{printf ("\n1:Insert a node in the Binary Tree");
printf ("\n2: Display the preorder of the Binary Tree");
printf ("\n3: Display the inorder of the Binary Tree");
printf ("\n4: Display the postorder of the Binary Tree");
printf ("\n5: Quit");
printf ("\nEnter your choice : ");
```

OUTPUT:



```
 Inorder traversing TREE :
40
20
10
30
50

1:Insert a node in the Binary Tree
2:Display the preorder of the Binary Tree
3:Display the inorder of the Binary Tree
4:Display the postorder of the Binary Tree
5:Quit
Enter your choice : 4

 Postorder traversing TREE :
40
20
50
30
10

1:Insert a node in the Binary Tree
2:Display the preorder of the Binary Tree
3:Display the inorder of the Binary Tree
4:Display the postorder of the Binary Tree
5:Quit
Enter your choice : 5
END
```

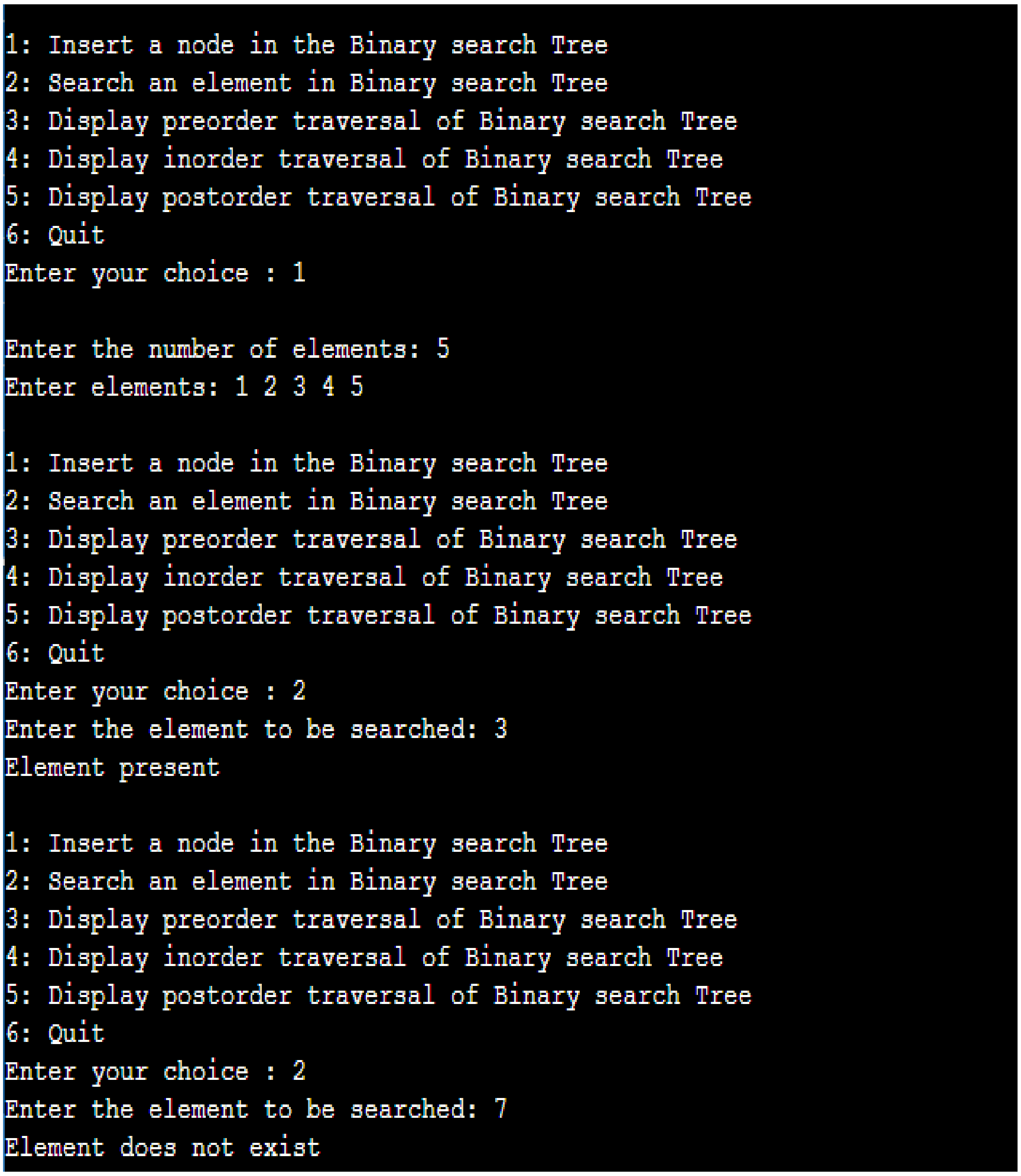
```
scanf ("%d", &s);
switch (s)
{case 1: printf ("Enter integers- To quit enter 0 : ");
            scanf("%ld", &digit);
            while (digit != 0)
            {root = insert (root, digit);
             scanf ("%ld", &digit);}
            continue;
case 2:printf ("\n Preorder traversing TREE :");
          preorder (root);
          continue ;
case 3:printf ("\n Inorder traversing TREE : ");
          inorder (root);
          continue;
          printf ("\n Inorder traversing TREE : ");
          inorder (root);
          continue;
case 4:printf ("\n Postorder traversing TREE : ");
          postorder (root);
          continue;
case 5:printf ("END");
          exit(0);
}
}while(s != 5);}
node *insert (node *p, long digit)
{if (p == NULL)
{p= (node *) malloc (sizeof (node));
p-> left =p->right = NULL;
p-> num = digit;
```

```
count++;
}
else
if (count %2 == 0)
p->left = insert (p->left, digit) ;
else
p-> right = insert (p-> right, digit) ;
return (p);
}
void preorder (node *p)
{if (p != NULL)
{printf ("%d\n", p->num) ;
preorder (p-> left);
preorder (p->right);
}}
void inorder (node *p)
{if (p != NULL)
{inorder (p->left);
printf ("%d\n", p->num);
inorder (p->right);
}}
void postorder (node *p)
{if (p != NULL)
{postorder (p->left);
postorder (p->right);
printf ("%d\n", p->num) ;
}}
```

RESULT:

Program executed successfully and output verified.

OUTPUT:



```
1: Insert a node in the Binary search Tree
2: Search an element in Binary search Tree
3: Display preorder traversal of Binary search Tree
4: Display inorder traversal of Binary search Tree
5: Display postorder traversal of Binary search Tree
6: Quit
Enter your choice : 1

Enter the number of elements: 5
Enter elements: 1 2 3 4 5

1: Insert a node in the Binary search Tree
2: Search an element in Binary search Tree
3: Display preorder traversal of Binary search Tree
4: Display inorder traversal of Binary search Tree
5: Display postorder traversal of Binary search Tree
6: Quit
Enter your choice : 2
Enter the element to be searched: 3
Element present

1: Insert a node in the Binary search Tree
2: Search an element in Binary search Tree
3: Display preorder traversal of Binary search Tree
4: Display inorder traversal of Binary search Tree
5: Display postorder traversal of Binary search Tree
6: Quit
Enter your choice : 2
Enter the element to be searched: 7
Element does not exist
```

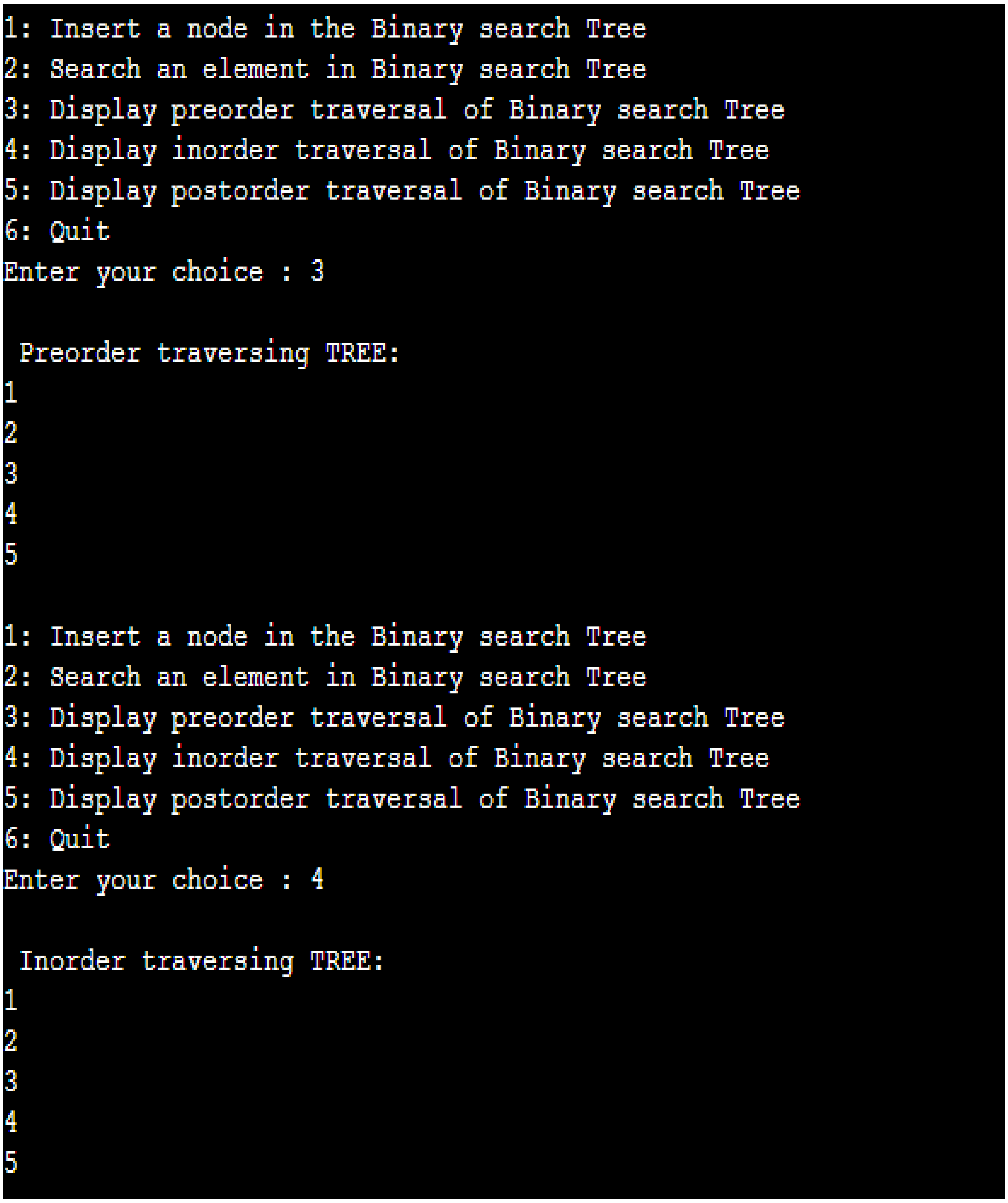
BINARY SEARCH TREE & TRAVERSALS

AIM: Write a menu driven program for creation of binary search tree and traversals.

PROGRAM CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{int num;
struct node *left;
struct node *right;
};
typedef struct node node;
node *p=NULL;
node *delnum(int digit, node *root);
node *delte_node(int digit,node *root);
node *insert(node *tree,long num);
void search(node *p,int num);
void preorder(node *tree);
void inorder(node *tree);
void postorder(node *tree);
void main()
{int digit,dig;
int s,n,i;
do
{ printf ("\n1: Insert a node in the Binary search Tree");
printf ("\n2: Search an element in Binary search Tree");
printf ("\n3: Display preorder traversal of Binary search Tree");
```

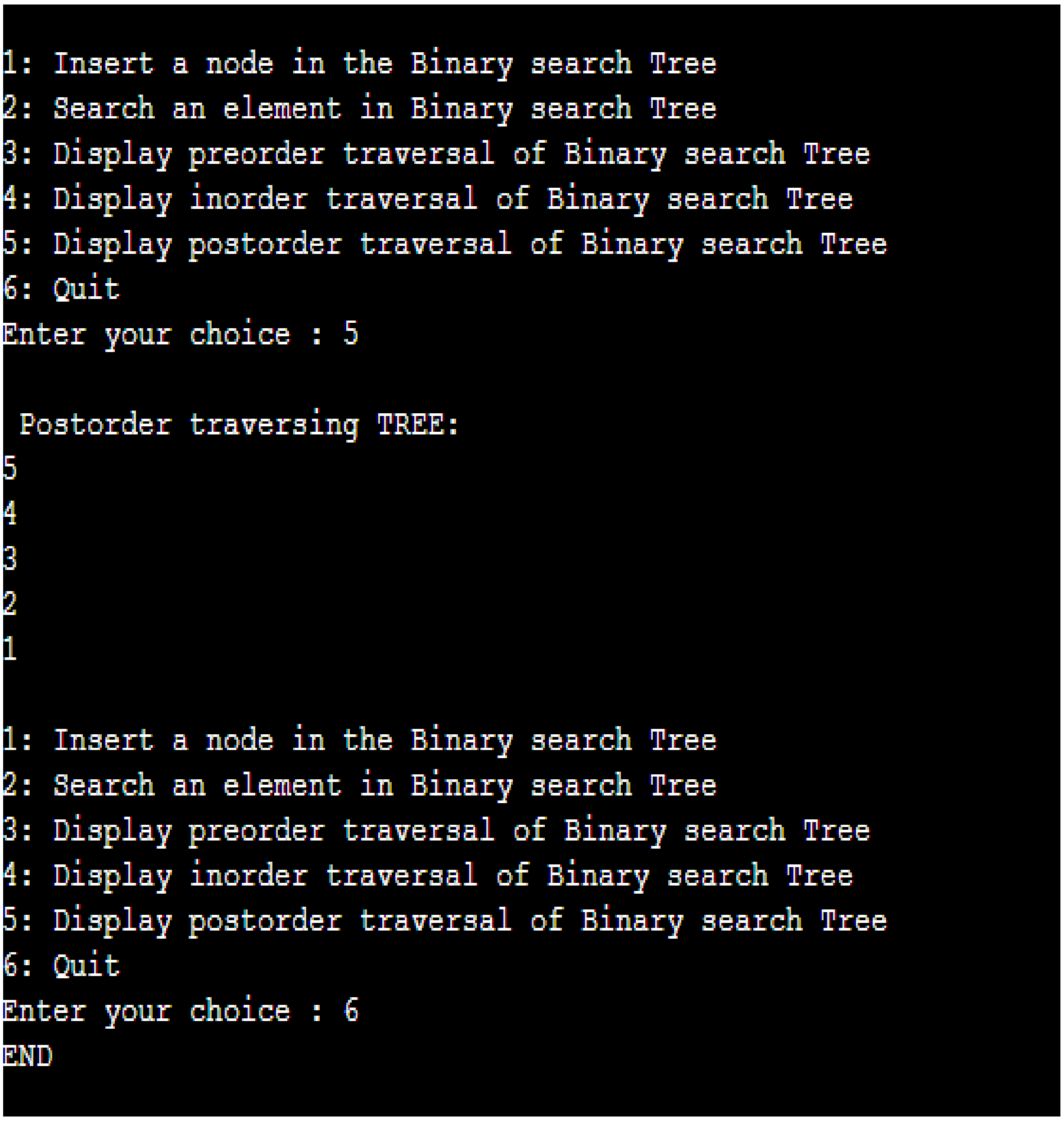
OUTPUT:



```
1: Insert a node in the Binary search Tree
2: Search an element in Binary search Tree
3: Display preorder traversal of Binary search Tree
4: Display inorder traversal of Binary search Tree
5: Display postorder traversal of Binary search Tree
6: Quit
Enter your choice : 3

 Preorder traversing TREE:
1
2
3
4
5

1: Insert a node in the Binary search Tree
2: Search an element in Binary search Tree
3: Display preorder traversal of Binary search Tree
4: Display inorder traversal of Binary search Tree
5: Display postorder traversal of Binary search Tree
6: Quit
Enter your choice : 4

 Inorder traversing TREE:
1
2
3
4
5
```

```
printf ("\n4: Display inorder traversal of Binary search Tree");
printf ("\n5: Display postorder traversal of Binary search Tree");
printf ("\n6: Quit");
printf ("\nEnter your choice : ");
scanf ("%d", &s);
switch (s)
{case 1: printf("\nEnter the number of elements: ");
              scanf("%d",&n);
              printf ("Enter elements: ");
              for(i=0;i<n;i++)
             {scanf("%d", &digit);
              p = insert (p, digit);
              }
             continue;
case 2:printf("Enter the element to be searched: ");
            scanf("%d",&dig);
            search(p,dig);
            continue;
case 3:printf ("\n Preorder traversing TREE: \n");
            preorder (p);
            continue ;
case 4:printf ("\n Inorder traversing TREE:\n");
            inorder (p);
            continue;
            printf ("\n Inorder traversing TREE");
            inorder (p);
            continue;
case 5:printf ("\n Postorder traversing TREE:\n");
           postorder (p);
```

OUTPUT:



```
1: Insert a node in the Binary search Tree
2: Search an element in Binary search Tree
3: Display preorder traversal of Binary search Tree
4: Display inorder traversal of Binary search Tree
5: Display postorder traversal of Binary search Tree
6: Quit
Enter your choice : 5

 Postorder traversing TREE:
5
4
3
2
1

1: Insert a node in the Binary search Tree
2: Search an element in Binary search Tree
3: Display preorder traversal of Binary search Tree
4: Display inorder traversal of Binary search Tree
5: Display postorder traversal of Binary search Tree
6: Quit
Enter your choice : 6
END
```

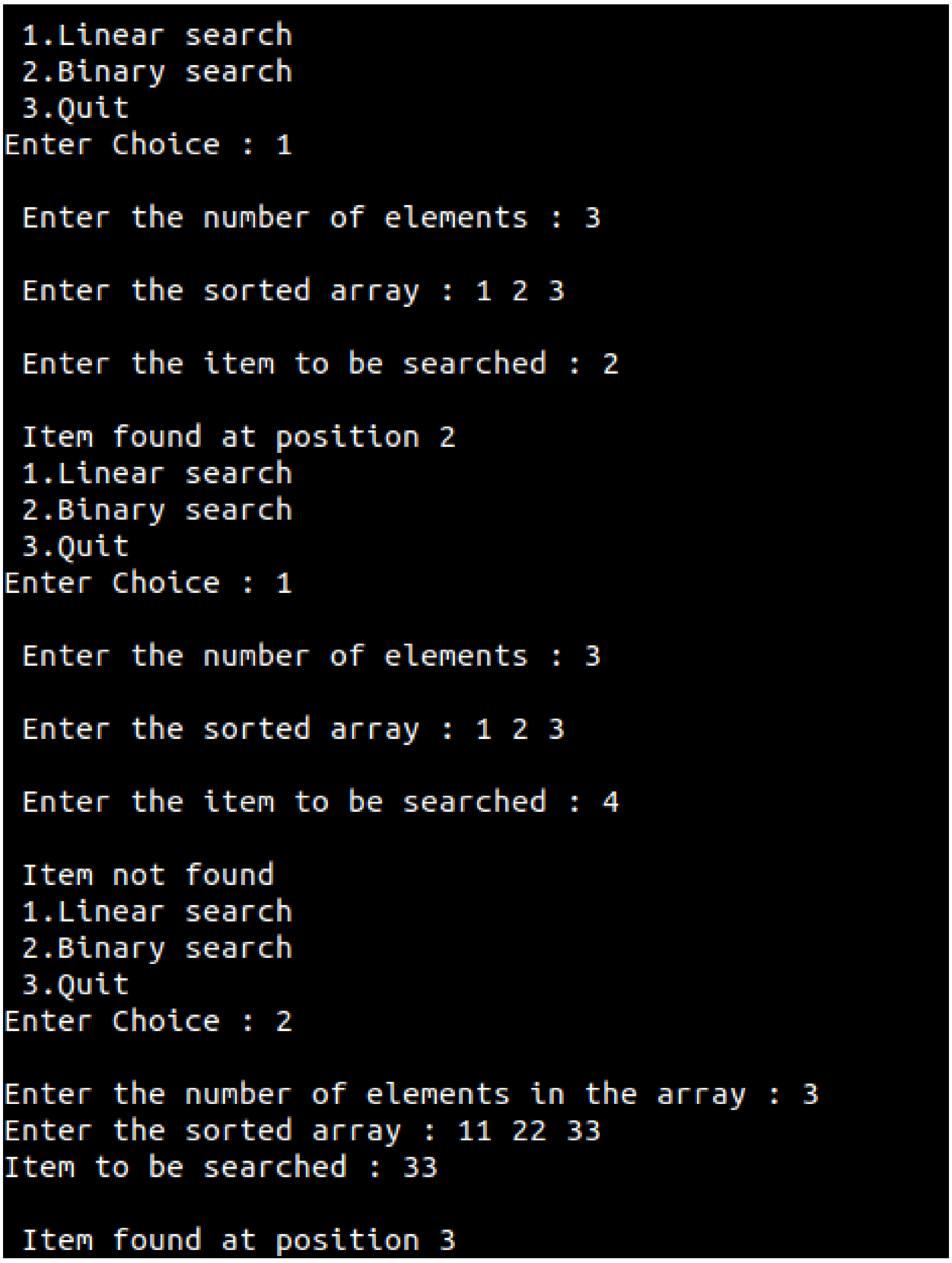
```
        continue;
case 6:printf ("END");
        exit(0);
}
}while(s != 6);
}
node *insert (node *p, long digit)
{if (p == NULL)
{p= (node *) malloc (sizeof (node));
p-> left =p->right = NULL;
p-> num = digit;
}
else
if (digit<p->num)
p->left = insert (p->left, digit) ;
else
if (digit>p->num)
p-> right = insert (p-> right, digit) ;
else
if(digit == p->num)
{printf("duplicate key");
}
return (p);
}
void search( node *p,int digit)
{if(p==NULL)
printf("Element does not exist");
else
 if (digit == p->num)
```

```
printf("Element present\n");
else
 if(digit<p->num)
search(p->left,digit);
else
search(p->right,digit);
}
void preorder (node *p)
{if (p != NULL)
{printf ("%d\n", p->num) ;
preorder (p-> left);
preorder (p->right);
}
}
void inorder (node *p)
{if (p != NULL)
{inorder (p->left);
printf ("%d\n", p->num);
inorder (p->right);
}}
void postorder (node *p)
{if (p != NULL)
{postorder (p->left);
postorder (p->right);
printf ("%d\n", p->num) ;
}
}
```

RESULT:

Program executed successfully and output verified

OUTPUT:



```
 1.Linear search
 2.Binary search
 3.Quit
Enter Choice : 1

 Enter the number of elements : 3

 Enter the sorted array : 1 2 3

 Enter the item to be searched : 2

 Item found at position 2
 1.Linear search
 2.Binary search
 3.Quit
Enter Choice : 1

 Enter the number of elements : 3

 Enter the sorted array : 1 2 3

 Enter the item to be searched : 4

 Item not found
 1.Linear search
 2.Binary search
 3.Quit
Enter Choice : 2

Enter the number of elements in the array : 3
Enter the sorted array : 11 22 33
Item to be searched : 33

 Item found at position 3
```

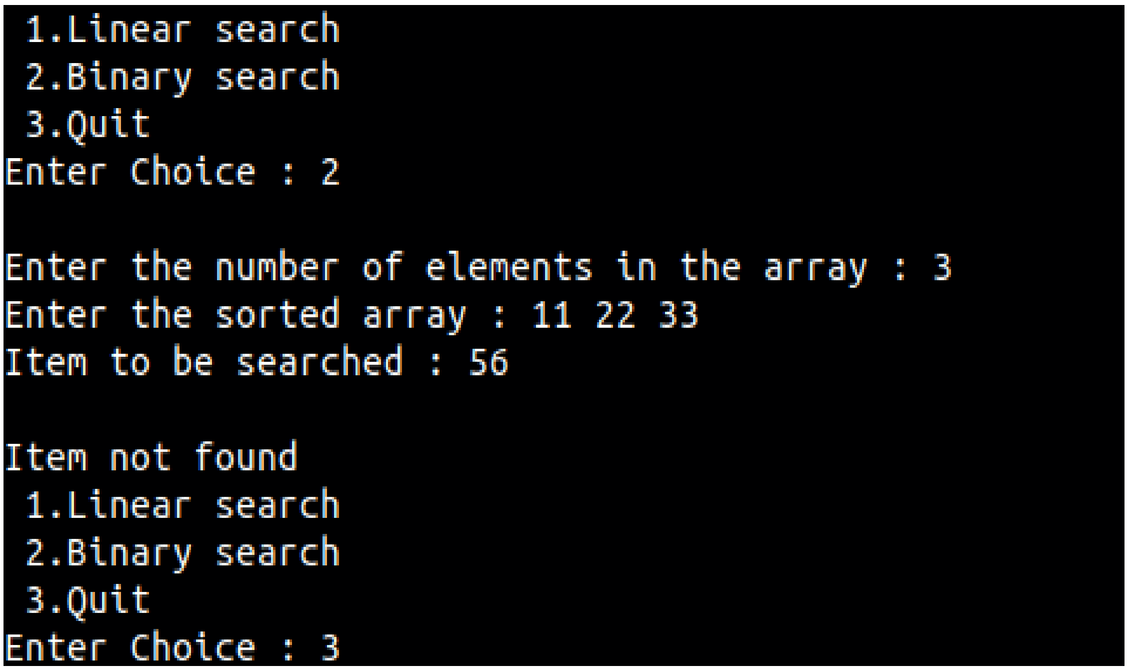
LINEAR SEARCH & BINARY SEARCH

AIM: Write a menu driven program to perform linear search and binary search.

PROGRAM CODE:

```
#include<stdio.h>
#include<stdlib.h>
void linear();
void binary();
int main()
{
int choice;
while (1)
{printf("\n 1.Linear search");
printf("\n 2.Binary search");
printf("\n 3.Quit");
printf("\nEnter Choice : ");
scanf("%d", &choice);
 switch (choice)
{
case 1:linear();
              break;
case 2:binary();
            break;
case 3:exit(0);
default:printf("\nIncorrect Choice\n");
             continue;
}
}
```

OUTPUT:



```
 1.Linear search
 2.Binary search
 3.Quit
Enter Choice : 2

Enter the number of elements in the array : 3
Enter the sorted array : 11 22 33
Item to be searched : 56

Item not found
 1.Linear search
 2.Binary search
 3.Quit
Enter Choice : 3
```

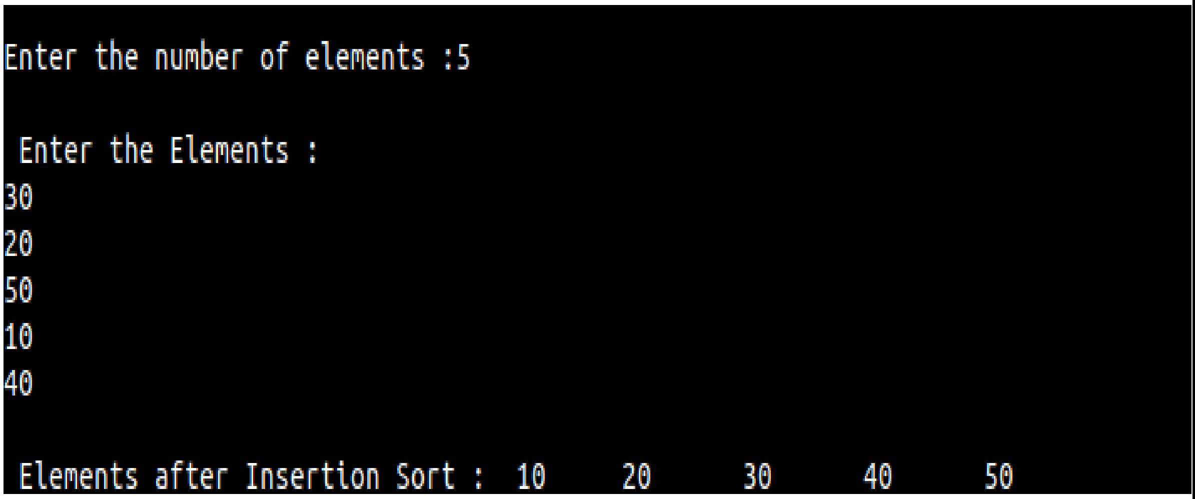
```
return 0;
}
void linear()
{
int a[25],item,n,i;
printf("\n Enter the number of elements : ");
scanf("%d",&n);
printf("\n Enter the sorted array : ");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("\n Enter the item to be searched : ");
scanf("%d",&item);
for(i=0;i<n;i++)
{
if(a[i]==item)
{
printf("\n Item found at position %d",i+1);
break;
}
}
if(i==n)
printf("\n Item not found");
}
void binary()
{
int a[25],beg,item,last,n,num,i,ch,mid,f=0,flag=0;
printf("\nEnter the number of elements in the array : ");
scanf("%d",&n);
printf("Enter the sorted array : ");
```

```
for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("Item to be searched : ");
scanf("%d",&item);
last=n-1;
mid=(beg+last)/2;
while(beg<=last)
{
if(item==a[mid])
{
printf("\n Item found at position %d",mid+1);
flag=flag+1;
break;
}
else if(a[mid]>item)
last=mid-1;
else
beg=mid+1;
mid=(beg+last)/2;
}
if(flag==0)
printf("\nItem not found");
}
```

<u>RESULT:</u>

Program executed successfully and output verified.

OUTPUT:



```
Enter the number of elements :5

 Enter the Elements :
30
20
50
10
40

 Elements after Insertion Sort :  10     20      30      40      50
```

INSERTION SORT

AIM: Write a program to perform insertion sort.

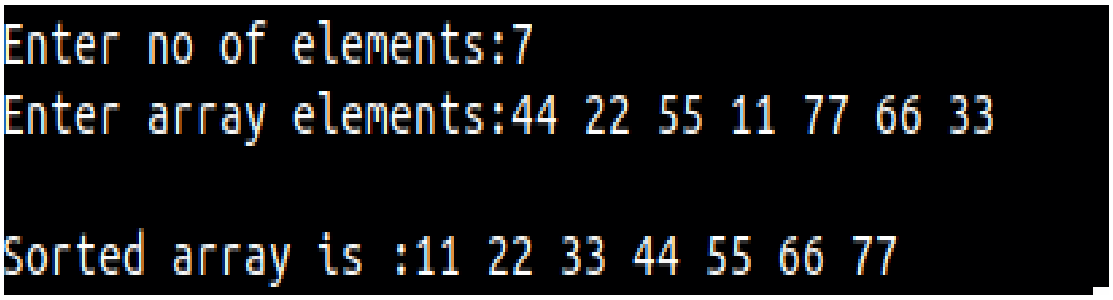
PROGRAM CODE:

```
#include <stdio.h>
void main()
{int a[100], number, i, j, temp;
printf("\nEnter the number of elements :");
scanf("%d", &number);
printf("\n Enter the Elements :\n");
for(i = 0; i < number; i++)
scanf("%d", &a[i]);
for(i = 1; i <= number - 1; i++)
{for(j = i; j > 0 && a[j - 1] > a[j]; j--)
{temp = a[j];
a[j] = a[j - 1];
a[j - 1] = temp;
}
}
printf("\n Elements after Insertion Sort : ");
for(i = 0; i < number; i++)
{printf(" %d \t", a[i]);
}
printf("\n");
}
```

RESULT:

Program executed successfully and output verified.

OUTPUT:



```
Enter no of elements:7
Enter array elements:44 22 55 11 77 66 33

Sorted array is :11 22 33 44 55 66 77
```

MERGE SORT

AIM: Write a program to perform merge sort.

PROGRAM CODE:

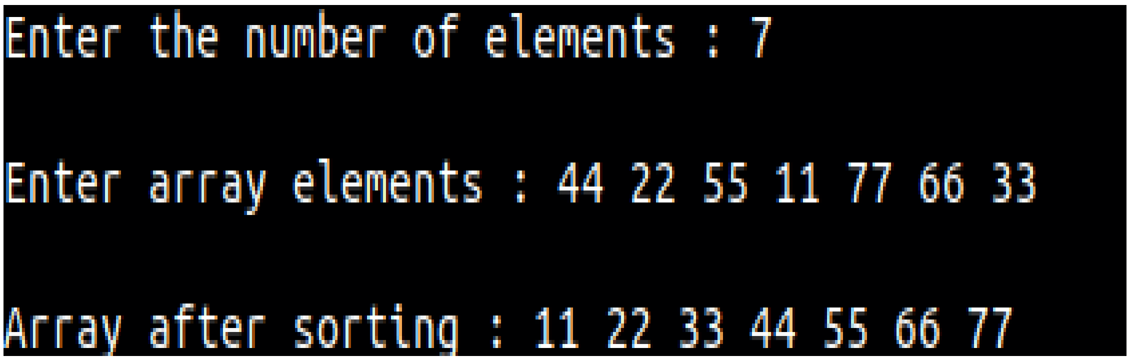
```
#include<stdio.h>
void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);
int main()
{
int a[30],n,i;
printf("Enter no of elements:");
scanf("%d",&n);
printf("Enter array elements:");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
mergesort(a,0,n-1);
printf("\nSorted array is : ");
for(i=0;i<n;i++)
printf("%d ",a[i]);
printf("\n");
return 0;
}
void mergesort(int a[],int i,int j)
{
int mid;
if(i<j)
{
```

```
mid=(i+j)/2;
mergesort(a,i,mid);
mergesort(a,mid+1,j);
merge(a,i,mid,mid+1,j);
}
}
void merge(int a[],int i1,int j1,int i2,int j2)
{
int temp[50];
int i,j,k;
i=i1;
j=i2;
k=0;
while(i<=j1 && j<=j2)
{
if(a[i]<a[j])
temp[k++]=a[i++];
else
temp[k++]=a[j++];
}
while(i<=j1)
temp[k++]=a[i++];
while(j<=j2)
temp[k++]=a[j++];
for(i=i1,j=0;i<=j2;i++,j++)
a[i]=temp[j];
}
```

RESULT:

Program executed successfully and output verified.

OUTPUT:



```
Enter the number of elements : 7

Enter array elements : 44 22 55 11 77 66 33

Array after sorting : 11 22 33 44 55 66 77
```

QUICK SORT

AIM: Write a program to perform quick sort.

PROGRAM CODE:

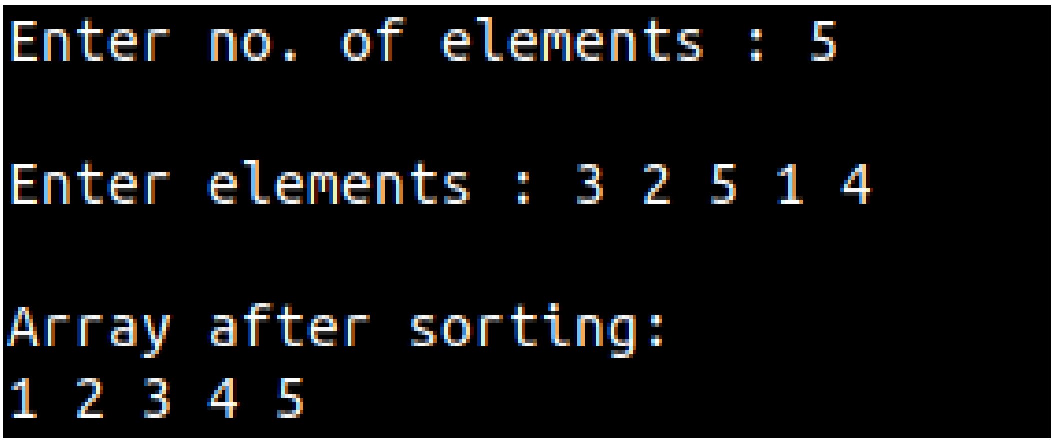
```
#include <stdio.h>
void quick_sort(int[],int,int);
int partition(int[],int,int);
int main()
{
int a[50],n,i;
printf("Enter the number of elements : ");
scanf("%d",&n);
printf("\nEnter array elements : ");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
quick_sort(a,0,n-1);
printf("\nArray after sorting : ");
for(i=0;i<n;i++)
printf("%d ",a[i]);
printf("\n");
return 0;
}
void quick_sort(int a[],int l,int u)
{
int j;
if(l<u)
{
j=partition(a,l,u);
```

```
quick_sort(a,l,j-1);
quick_sort(a,j+1,u);
}
}
int partition(int a[],int l,int u)
{int v,i,j,temp;
v=a[l];
i=l;
j=u+1;
do
{do
i++;
while(a[i]<v&&i<=u);
do
j--;
while(v<a[j]);
if(i<j)
{temp=a[i];
a[i]=a[j];
a[j]=temp;
}
}
while(i<j);
a[l]=a[j];
a[j]=v;
return(j);
}
```

RESULT:

Program executed successfully and output verified.

OUTPUT:



```
Enter no. of elements : 5

Enter elements : 3 2 5 1 4

Array after sorting:
1 2 3 4 5
```

HEAP SORT

AIM: Write a program to perform heap sort.

PROGRAM CODE:

```
#include<stdio.h>
void create(int []);
void down_adjust(int [],int);
void main()
{int heap[30],n,i,last,temp;
printf("Enter no. of elements : ");
scanf("%d",&n);
printf("\nEnter elements : ");
for(i=1;i<=n;i++)
scanf("%d",&heap[i]);
heap[0]=n;
create(heap);
while(heap[0] > 1)
{last=heap[0];
temp=heap[1];
heap[1]=heap[last];
heap[last]=temp;
heap[0]--;
down_adjust(heap,1);
}
printf("\nArray after sorting:\n");
for(i=1;i<=n;i++)
printf("%d ",heap[i]);
printf("\n");
```

```
}
void create(int heap[])
{int i,n;
n=heap[0];
for(i=n/2;i>=1;i--)
down_adjust(heap,i);
}
void down_adjust(int heap[],int i)
{int j,temp,n,flag=1;
n=heap[0];
while(2*i<=n && flag==1)
{j=2*i;
if(j+1<=n && heap[j+1] > heap[j])
j=j+1;
if(heap[i] > heap[j])
flag=0;
else
{
temp=heap[i];
heap[i]=heap[j];
heap[j]=temp;
i=j;
}
}
}
```

RESULT:

Program executed successfully and output verified.