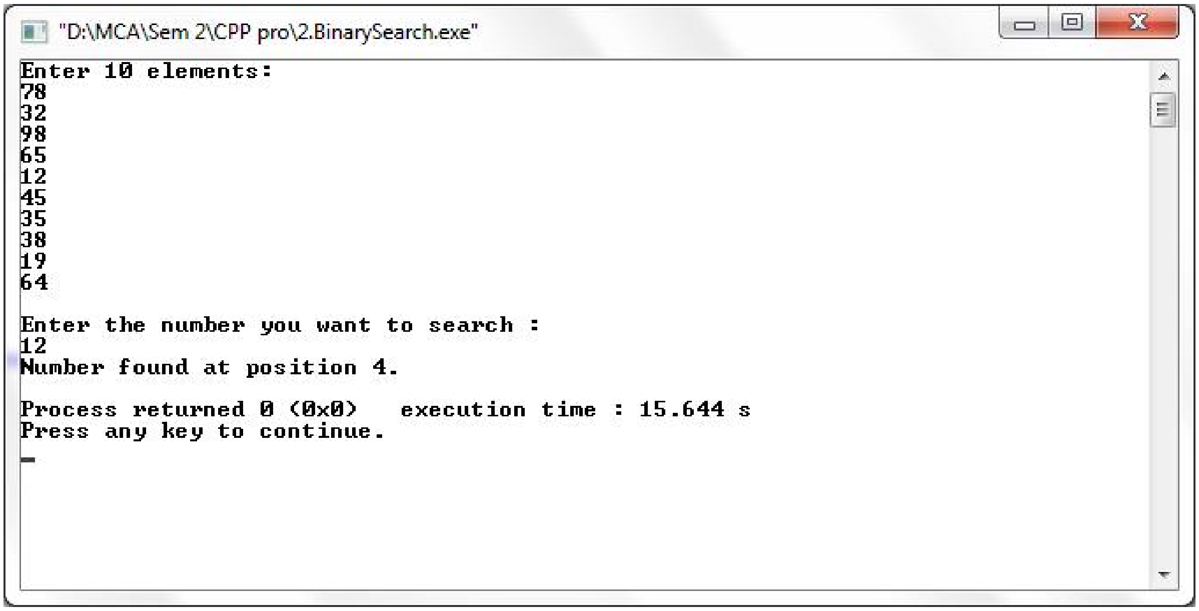
A] Linear Search

```
#include <iostream>
using namespace std;
int linearSearch(int a[], int n, int val) {
for (int i = 0; i < n; i++)
{
if (a[i] == val)
return i+1;
}
return -1;
}
int main() {
int a[] = {69, 39, 29, 10, 56, 40, 24, 13, 51};
int val = 56;
int n = sizeof(a) / sizeof(a[0]);
int res = linearSearch(a, n, val);
cout<<"The elements of the array are - ";
for (int i = 0; i < n; i++)
cout<<a[i]<<" ";
cout<<"\nElement to be searched is - "<<val;
if (res == -1)
cout<<"\nElement is not present in the array";
else
cout<<"\nElement is present at "<<res<<" position of array";
return 0;
}
```

B] Binary Search

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

using namespace std;

int main()
{
     int i,n,a[10],st=0,ed=9,mid;

    cout<<"Enter 10 elements:\n";
    for(i=0;i<10;i++)
    {
        cin>>a[i];
    }
    cout<<"\nEnter the number you want to search :\n";
    cin>>n;

    mid=(st+ed)/2;

while(n!=a[mid]&&st<=ed)
    {
        if(n>a[mid])
            st=mid+1;
        else
            ed=mid-1;
        mid=(st+ed)/2;
    }
    if(n==a[mid])
        cout<<"Element found at position "<<mid<<".\n";
    if(st>ed)
        cout<<"Element not found.\n";
}
```

Output:



```
Enter 10 elements:
78
32
98
65
12
45
35
38
19
64

Enter the number you want to search :
12
Number found at position 4.

Process returned 0 (0x0)   execution time : 15.644 s
Press any key to continue.
```

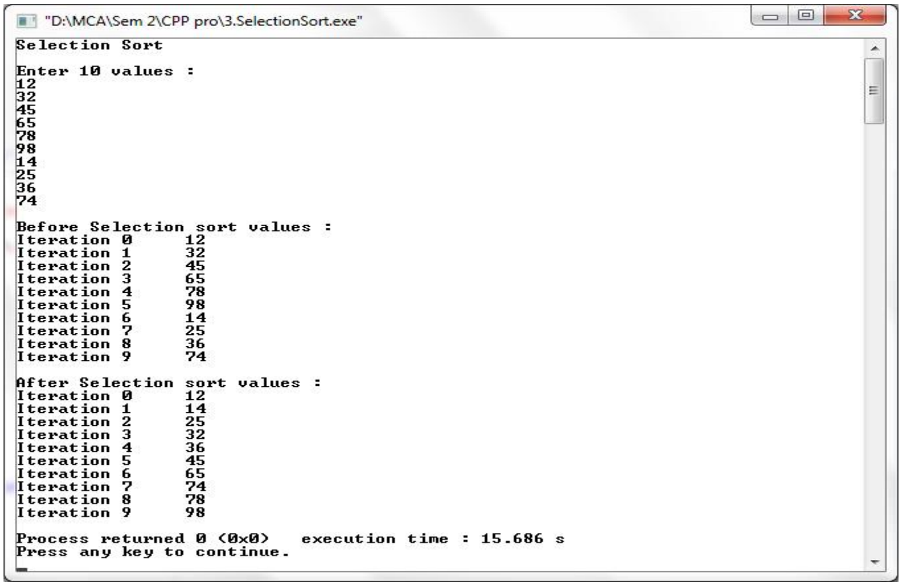
PRACTICAL 2

AIM: Implementation of different sorting techniques.

A] Selection Sort.

```
#include<iostream>
using namespace std;
int main()
{
    int a[10], i, j, n, temp;
    cout<<"Selection Sort\n\n";
    cout<<"Enter 10 values :\n";
    for(n=0;n<10;n++)
    {F
        cin>>a[n];
    }
    cout<<"\nBefore Selection sort values :\n";
    for(n=0;n<10;n++)
    {
        cout<<"Iteration "<<n<<"\t"<<a[n]<<"\n";
    }
    cout<<"\nAfter Selection sort values :\n";

    for(i=0;i<=n-1;i++)
    {
        for(j=i+1;j<=n;j++)
        {
            if(a[i]>a[j])
            {
                temp=a[i];
                a[i]=a[j];
                a[j]=temp;
            }
        }
        cout<<"Iteration "<<i<<"\t"<<a[i]<<"\n";
    }
    return 0;
}
```

Output:-

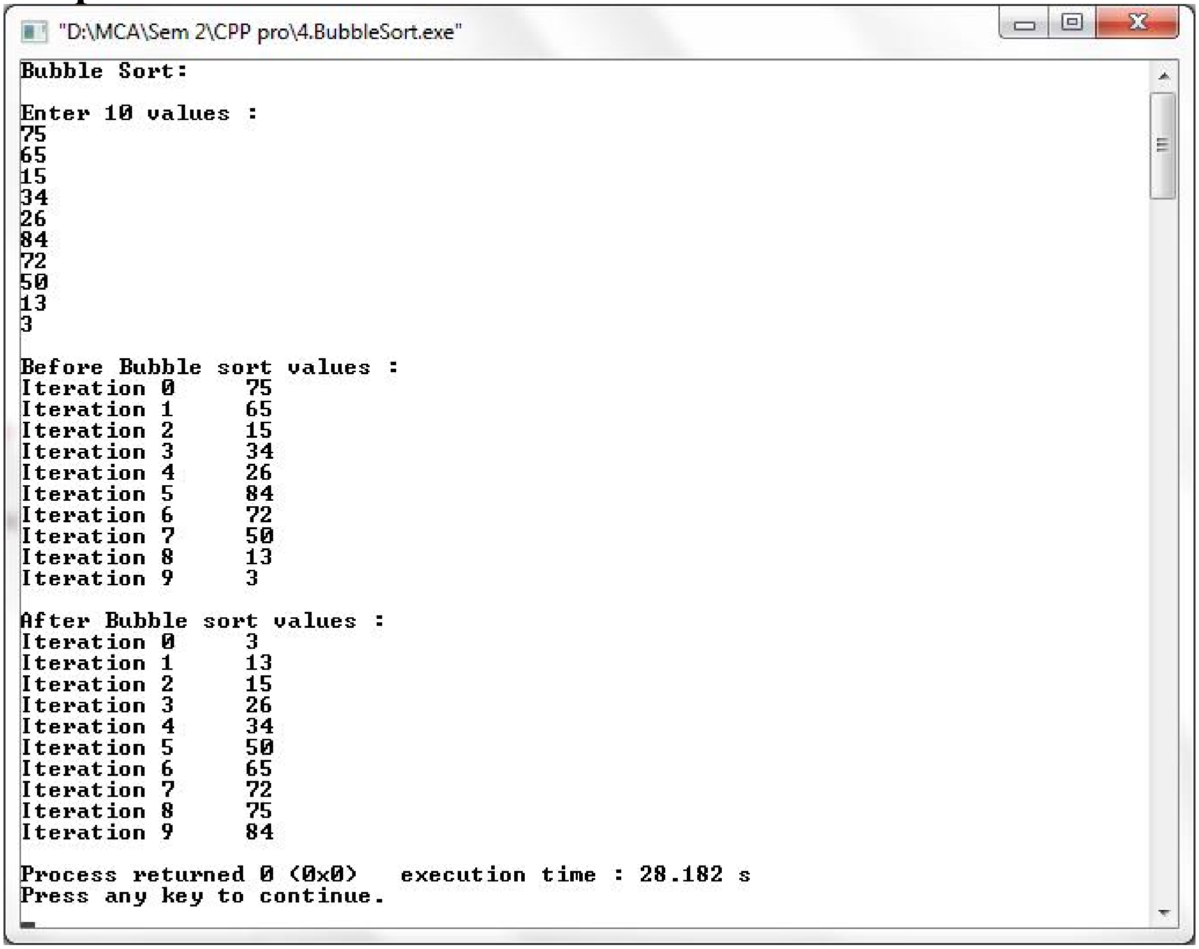


```
Selection Sort

Enter 10 values :
12
32
45
65
78
98
14
25
36
74

Before Selection sort values :
Iteration 0     12
Iteration 1     32
Iteration 2     45
Iteration 3     65
Iteration 4     78
Iteration 5     98
Iteration 6     14
Iteration 7     25
Iteration 8     36
Iteration 9     74

After Selection sort values :
Iteration 0     12
Iteration 1     14
Iteration 2     25
Iteration 3     32
Iteration 4     36
Iteration 5     45
Iteration 6     65
Iteration 7     74
Iteration 8     78
Iteration 9     98

Process returned 0 (0x0)   execution time : 15.686 s
Press any key to continue.
_
```

B] Bubble Sort.

```
#include<iostream>
using namespace std;
int main()
{
    int a[10], i, j, n, temp;
    cout<<"Bubble Sort:\n\n";
    cout<<"Enter 10 values :\n";
    for(n=0;n<10;n++)
    {
        cin >> a[n];
    }

    cout<<"\nBefore Bubble sort values :\n";
    for(n=0;n<10;n++)
    {
         cout<<"Iteration "<<n<<"\t"<<a[n]<<"\n";
    }

    cout<<"\nAfter Bubble sort values :\n";
    for(i=0;i<=n-1;i++)
    {
        for(j=0;j<=n-i-1;j++)
        {
            if(a[j]>a[j+1])

      {
                temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
    for(i=0; i<n;i++)
    {
         cout<<"Iteration "<<i<<"\t"<<a[i]<<"\n";
    }
    return 0;
}
```

Output:

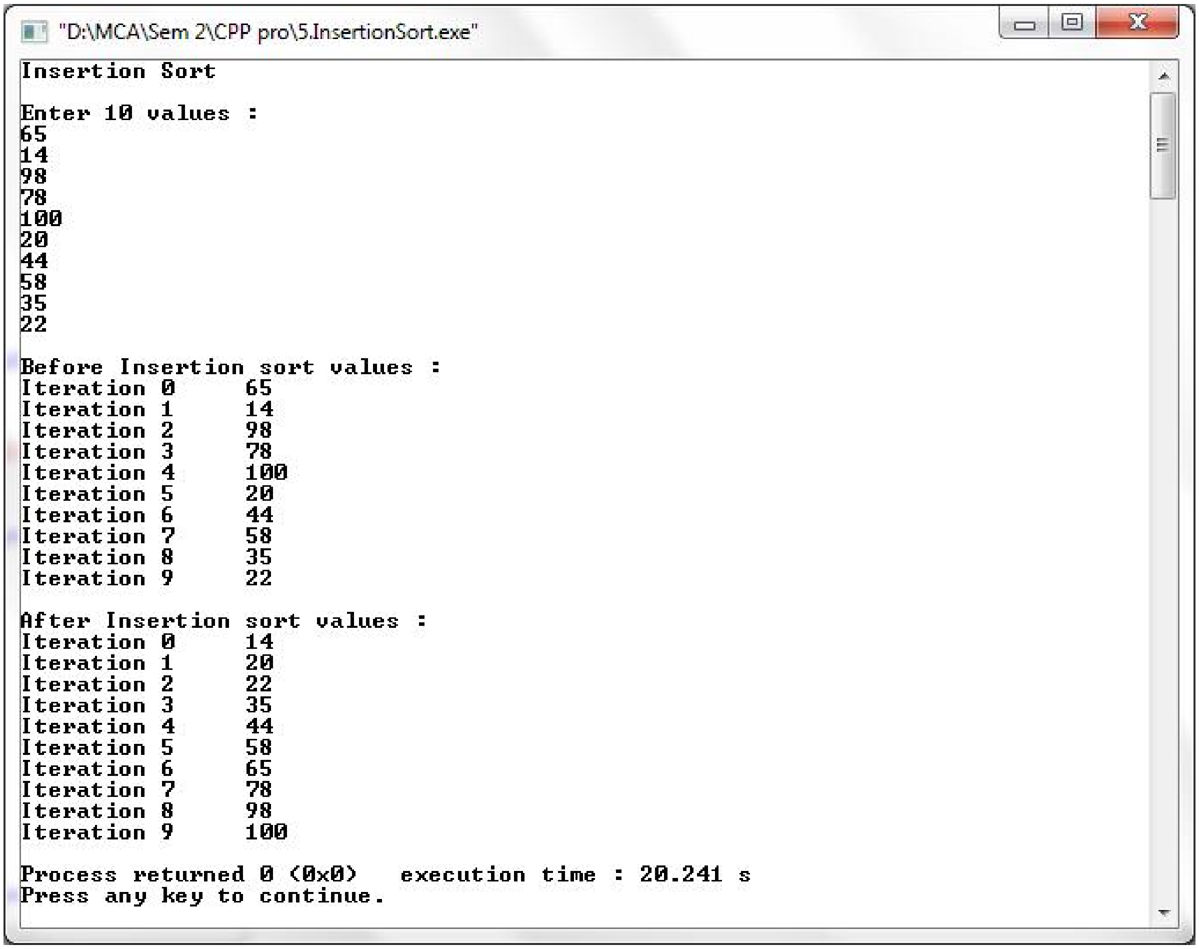


```
Bubble Sort:

Enter 10 values :
75
65
15
34
26
84
72
50
13
3

Before Bubble sort values :
Iteration 0     75
Iteration 1     65
Iteration 2     15
Iteration 3     34
Iteration 4     26
Iteration 5     84
Iteration 6     72
Iteration 7     50
Iteration 8     13
Iteration 9     3

After Bubble sort values :
Iteration 0     3
Iteration 1     13
Iteration 2     15
Iteration 3     26
Iteration 4     34
Iteration 5     50
Iteration 6     65
Iteration 7     72
Iteration 8     75
Iteration 9     84

Process returned 0 (0x0)   execution time : 28.182 s
Press any key to continue.
```

C] Insertion Sort

```cpp
#include<iostream>
using namespace std;
int main()
{
    int a[10], i, j, x;
    cout<<"Insertion Sort\n\n";
    cout<<"Enter 10 values :\n";
    for(i=0;i<10;i++)
    {
        cin>>a[i];
    }

    cout<<"\nBefore Insertion sort values :\n";
    for(i=0;i<10;i++)
    {
        cout<<"Iteration "<<i<<"\t"<<a[i]<<"\n";
    }

    cout<<"\nAfter Insertion sort values :\n";
    for(j=1;j<10;j++)
    {
        x=a[j];
        for(i=j-1;i>=0&&x<a[i];i--)
        {
            a[i+1]=a[i];
        }
        a[i+1]=x;
    }
    for(i=0;i<10;i++)
    {
         cout<<"Iteration "<<i<<"\t"<<a[i]<<"\n";
    }
    return 0;
}
```

Output:



```
Insertion Sort

Enter 10 values :
65
14
98
78
100
20
44
58
35
22

Before Insertion sort values :
Iteration 0      65
Iteration 1      14
Iteration 2      98
Iteration 3      78
Iteration 4      100
Iteration 5      20
Iteration 6      44
Iteration 7      58
Iteration 8      35
Iteration 9      22

After Insertion sort values :
Iteration 0      14
Iteration 1      20
Iteration 2      22
Iteration 3      35
Iteration 4      44
Iteration 5      58
Iteration 6      65
Iteration 7      78
Iteration 8      98
Iteration 9      100

Process returned 0 (0x0)   execution time : 20.241 s
Press any key to continue.
```

PRACTICAL 3

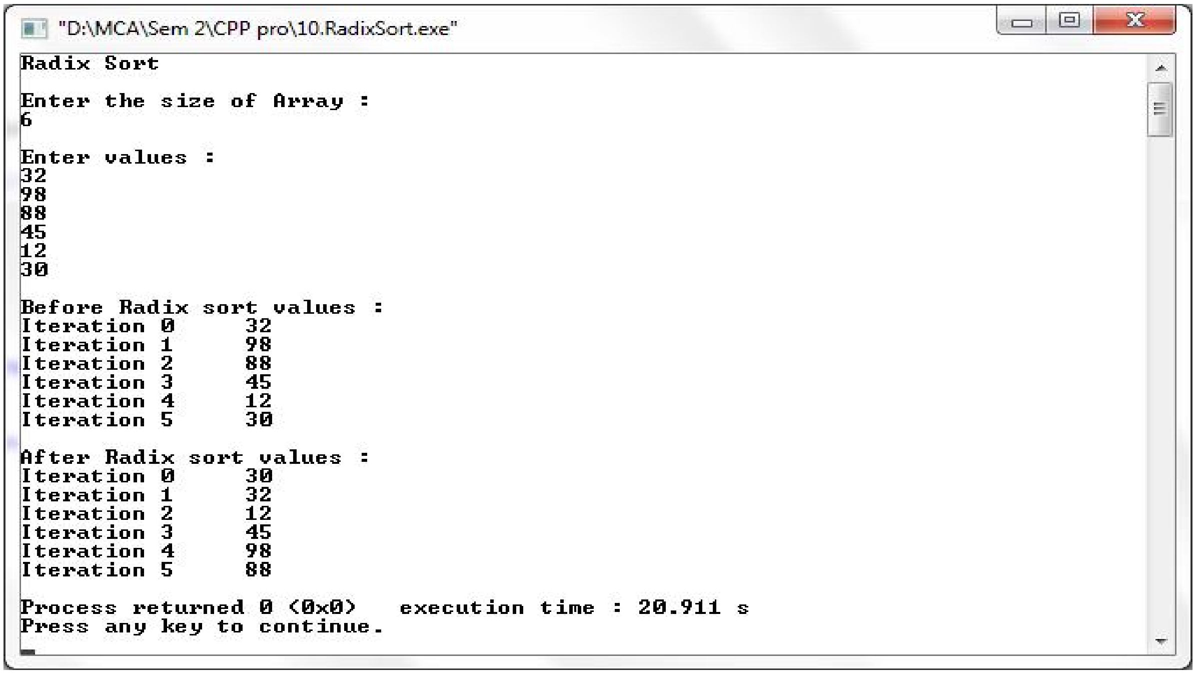
AIM: Implementation of different sorting techniques.

A] Radix Sort

```
#include<iostream>
#include<conio.h>
using namespace std;
class radix
{
    public:
    void sort()
    {
        int
arr[6],i,j,k,large,noofpasses=0,temp[10][10],divisor=1,arrcount[10],n;
        cout<<"Radix Sort\n\n";
        cout<<"Enter the size of Array :\n";
        cin>>n;
        cout<<"\nEnter values :\n";
        for(i=0;i<n;i++)
        {
            cin>>arr[i];
        }
        cout<<"\nBefore Radix sort values :\n";
        for(i=0;i<n;i++)
        {
            cout<<"Iteration "<<i<<"\t"<<arr[i]<<"\n";
        }
        large=arr[0];
        for(i=0;i<n;i++)
        {
            if(arr[i]>large)
            large=arr[i];
        }
        while(large>0)
        {
            noofpasses++;
            large/=10;
        }
        for(i=0;i<noofpasses;i++)
        {
            for(j=0;j<10;j++)
                arrcount[j]=0;

            for(j=0;j<n;j++)
            {
                k=(arr[j]/divisor)%10;
```

```
                    temp[k][arrcount[k]++]=arr[j];
                }
                int u=0;
                for(int p=0;p<10;p++)
                {
                    for(j=0;j<arrcount[p];j++)
                    {
                        arr[u++]=temp[p][j];

divisor*=10;
                    }
                }
            }
            cout<<"\nAfter Radix sort values :\n";
            for(i=0;i<n;i++)
            {
                cout<<"Iteration "<<i<<"\t"<<arr[i]<<"\n";
            }
        }
};
int main()
{
    radix r;
    r.sort();
    return 0;
}
```

Output:

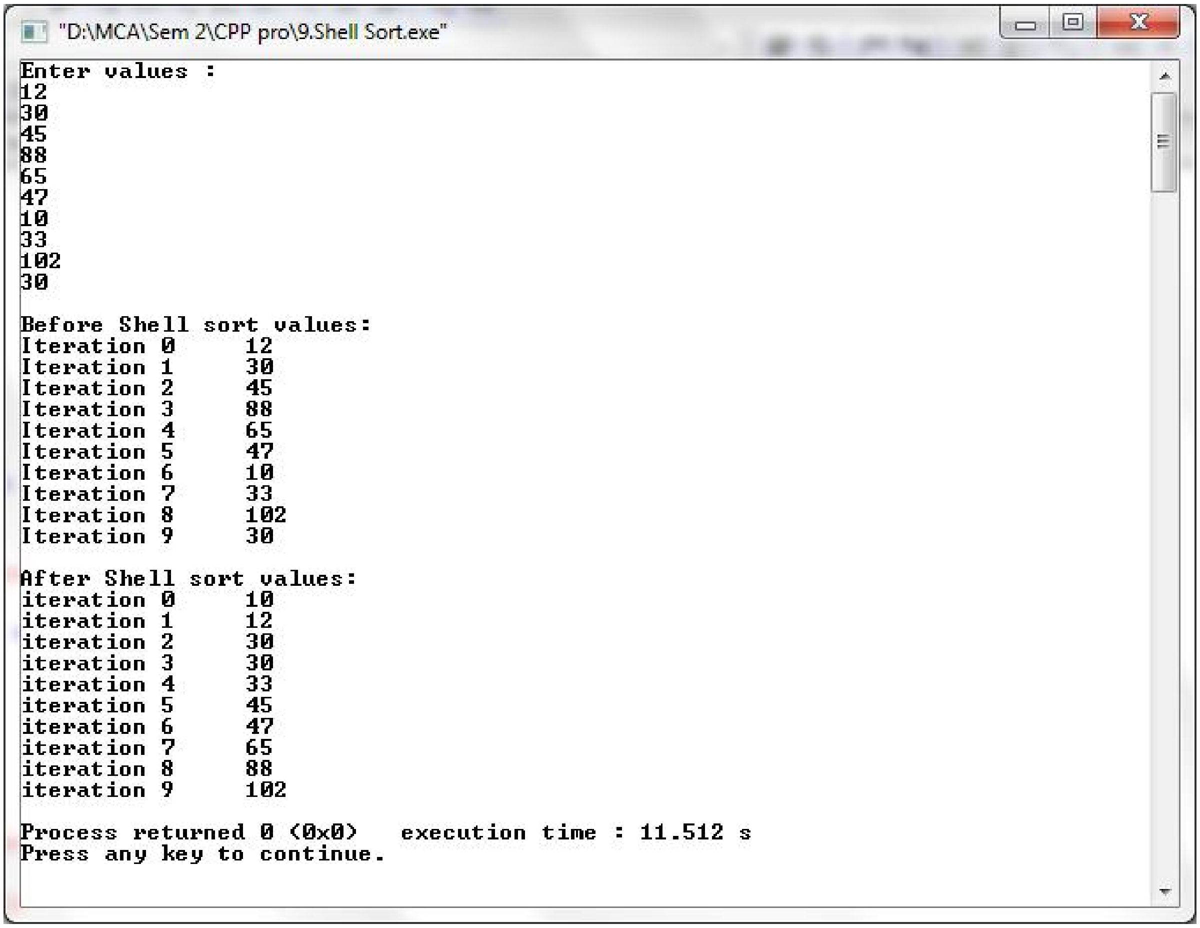


```
Radix Sort

Enter the size of Array :
6

Enter values :
32
98
88
45
12
30

Before Radix sort values :
Iteration 0     32
Iteration 1     98
Iteration 2     88
Iteration 3     45
Iteration 4     12
Iteration 5     30

After Radix sort values :
Iteration 0     30
Iteration 1     32
Iteration 2     12
Iteration 3     45
Iteration 4     98
Iteration 5     88

Process returned 0 (0x0)   execution time : 20.911 s
Press any key to continue.
```

B] Shell Sort

```
#include<iostream>
using namespace std;
int main()
{
     int a[10], i, j, n,temp;
    cout<<"Enter values :\n";
    for(i=0;i<10;i++)
    {
        cin>>a[i];
    }
    cout<<"\nBefore Shell sort values: \n";
    for(i=0; i<10; i++)
    {
        cout<<"Iteration "<<i<<"\t"<<a[i]<<"\n";
    }
    cout<<"\nAfter Shell sort values: \n";
    for(i=10/2;i>0;i/=2)
    {
        int flag=1;
        while(flag==1)
        {
            flag=0;
            for(j=0;j<10-i;j++)
            {
                if(a[j]>a[j+i])
                {
                    temp=a[j];
                    a[j]=a[j+i];
                    a[j+i]=temp;
                    flag=1;
                }
            }
        }

    }
    for(i=0;i<10;i++)
    {
         cout<<"iteration "<<i<<"\t"<<a[i]<<"\n";
    }
    return 0;
}
```

Output:



```
Enter values :
12
30
45
88
65
47
10
33
102
30

Before Shell sort values:
Iteration 0     12
Iteration 1     30
Iteration 2     45
Iteration 3     88
Iteration 4     65
Iteration 5     47
Iteration 6     10
Iteration 7     33
Iteration 8     102
Iteration 9     30

After Shell sort values:
iteration 0     10
iteration 1     12
iteration 2     30
iteration 3     30
iteration 4     33
iteration 5     45
iteration 6     47
iteration 7     65
iteration 8     88
iteration 9     102

Process returned 0 (0x0)   execution time : 11.512 s
Press any key to continue.
```

PRACTICAL 4

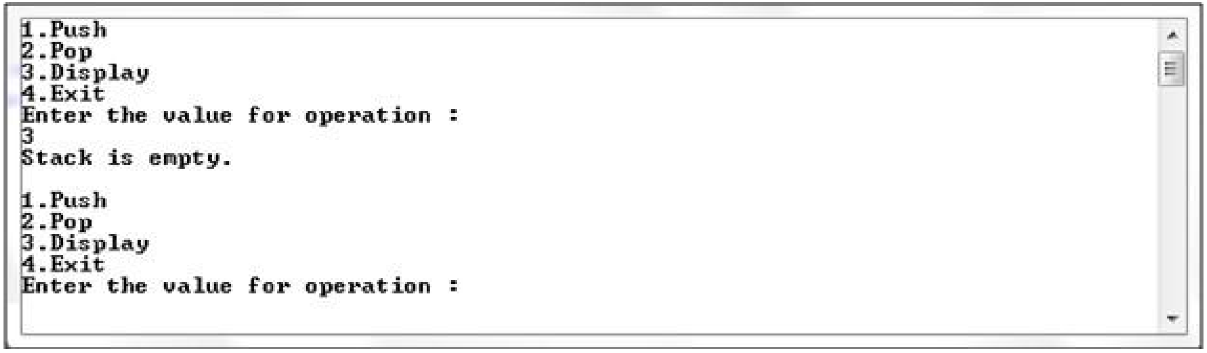
Aim: Implementation of stack operation using arrays and linked list.

A] Arrays

```
#include<iostream>
#define MAX 5
using namespace std;

class stack
{
    public:
    int top=-1;
    int x, stk[5],i;
    int push(int x)
    {
        if(top == MAX - 1)
        {
            cout<<"Stack Overflow\n\n";
        }
        else
        {
            cout<<"Enter the number to push to the stack :\n";
            cin>>x;

            stk[++top]=x;
            stk[top]=x;
        }
    }

    int pop()
    {
        if(top==-1)
        {
            cout<<"Stack Underflow\n\n";
        }
        cout<<"Popped value : ";
        x=stk[top];
        top--;
        cout<<x<<"\n";
    }

    void display()
    {
        if(top==-1)
        {
            cout<<"Stack is empty.\n\n";
        }
        else
        cout<<"Stack :\n";
        for(i=top;i>=0;i--)
        {
            cout<<stk[i]<<"\n";
        }
    }
};

int main()
{
    stack s;
    int ch,x;

    while(ch!=4)
    {
        cout<<"1.Push\n2.Pop\n3.Display\n4.Exit\n";
        cout<<"Enter the value for operation :\n";
        cin>>ch;

        switch(ch)
        {
            case 1:
                s.push(x);
                break;

            case 2:
                s.pop();
                break;

            case 3:
                s.display();
                break;

            case 4:
                break;
            default:
            cout<<"\nWrong choice.\n\n";
        }

    }
    return 0;
}
```

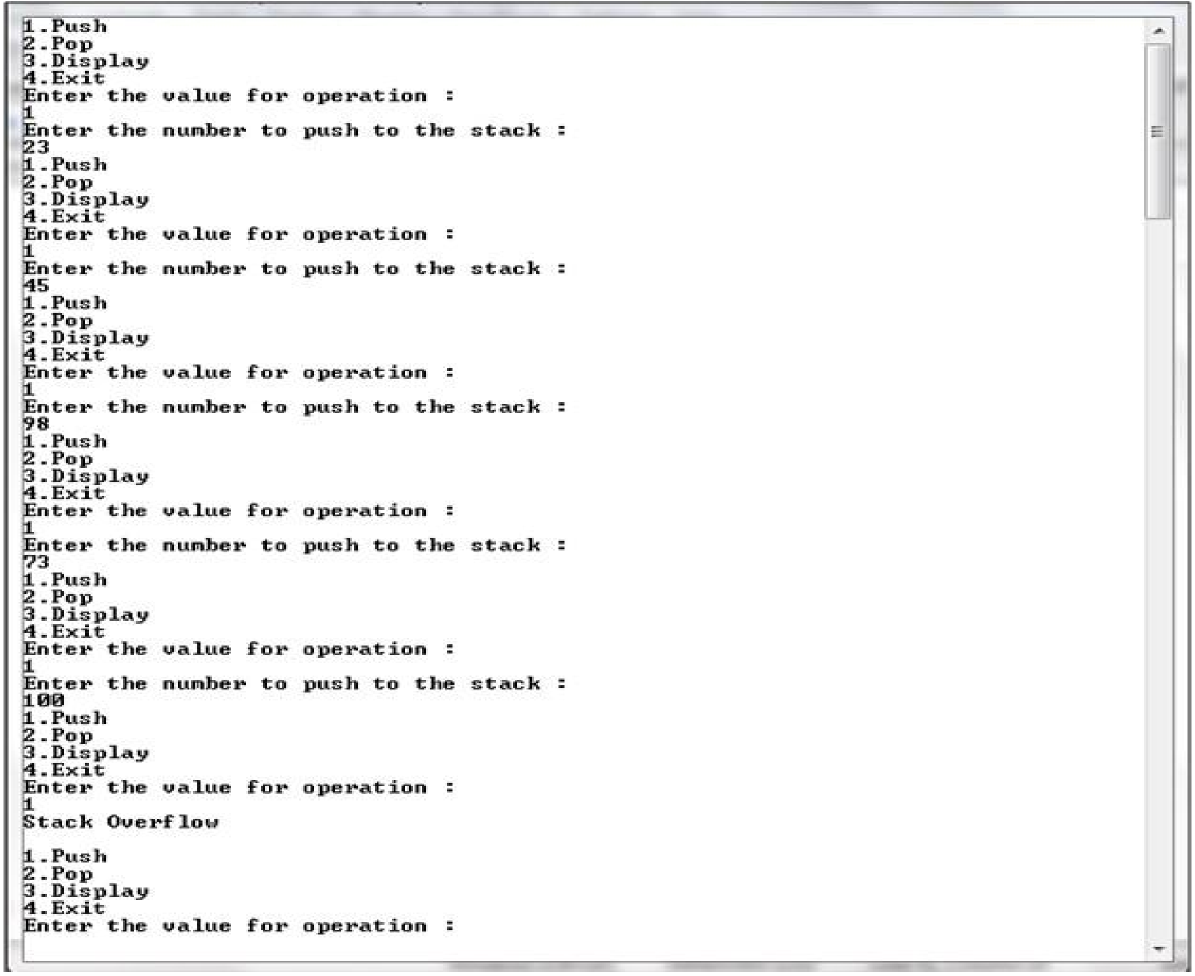
Output:

Stack Empty:

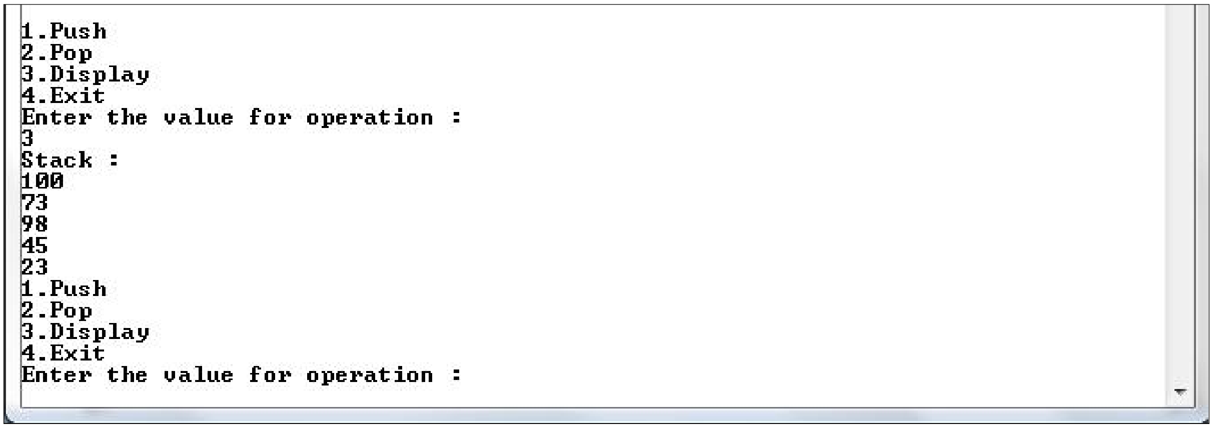


```
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
3
Stack is empty.

1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
```

Push Operation:

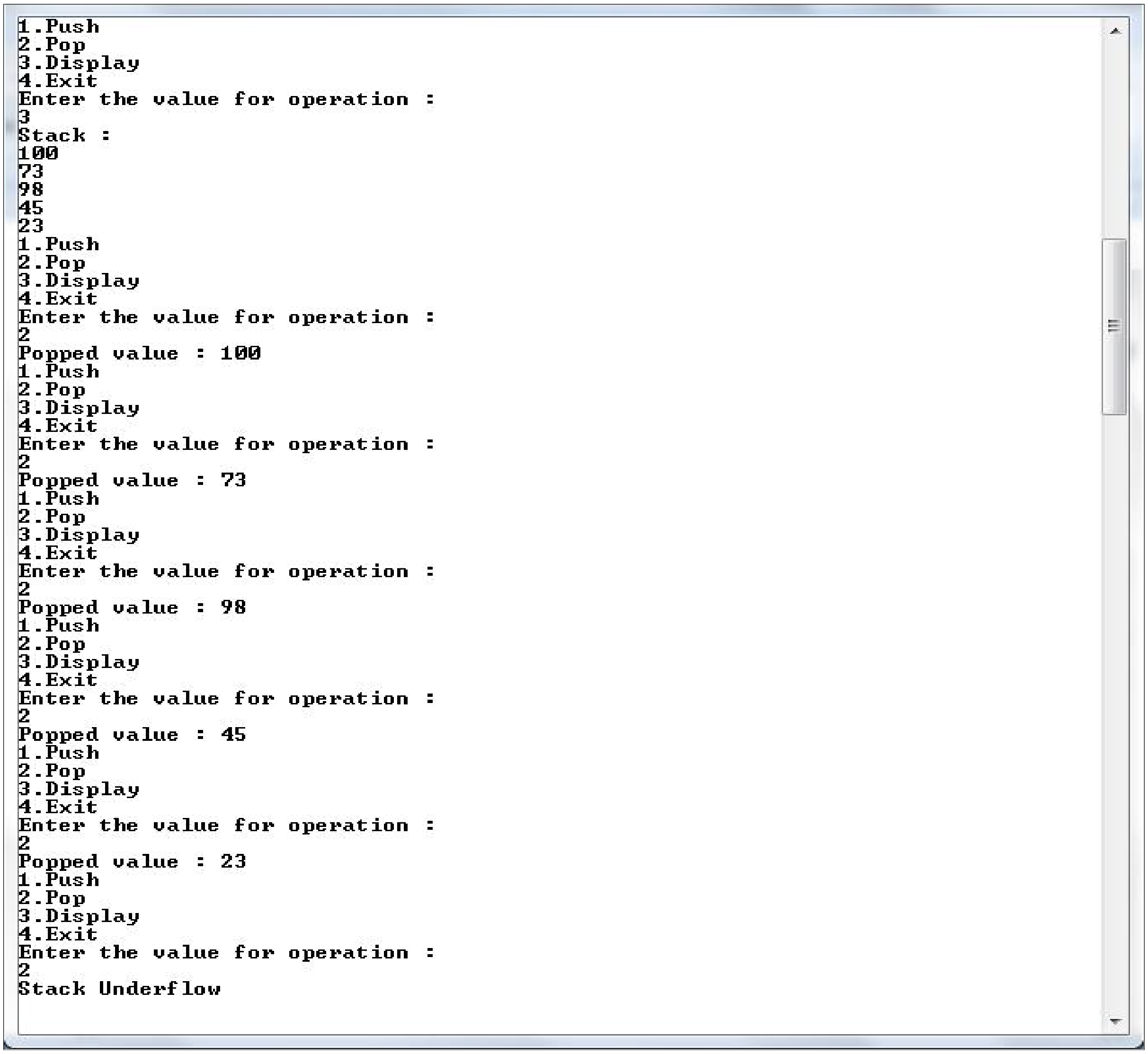


```
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
1
Enter the number to push to the stack :
23
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
1
Enter the number to push to the stack :
45
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
1
Enter the number to push to the stack :
98
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
1
Enter the number to push to the stack :
73
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
1
Enter the number to push to the stack :
100
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
1
Stack Overflow

1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
```



```
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
3
Stack :
100
73
98
45
23
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
```

Pop Operation:



```
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
3
Stack :
100
73
98
45
23
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
2
Popped value : 100
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
2
Popped value : 73
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
2
Popped value : 98
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
2
Popped value : 45
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
2
Popped value : 23
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
2
Stack Underflow
```

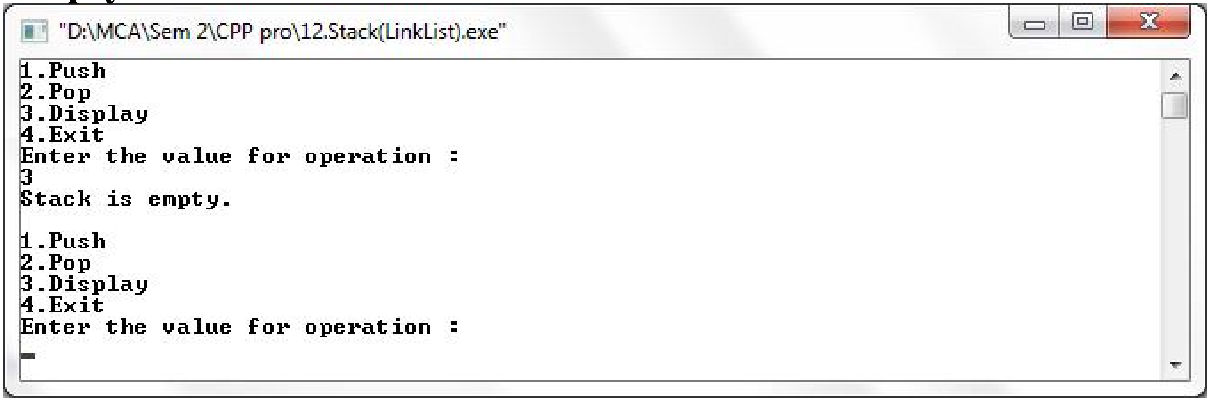
B] Linked list

```
#include<iostream>
using namespace std;
//Creating a NODE Structure
struct node
{
    int data;
    struct node *next;
};
//Creating a class STACK
class stack
{
    public:
        struct node *top=NULL;

        int push(int x) //to insert an element
        {
            struct node *ptr;
            ptr=new node;
            ptr->data=x;
            ptr->next=NULL;
            if(top!=NULL)
            ptr->next=top;
            top=ptr;
        }
        void pop() //to delete an element
        {
            struct node *temp;
            if(top==NULL)
            {
                cout<<"\nStack is empty.\n";
            }
            else
            {
            temp=top;
            top=top->next;
            cout<<"Popped value : "<<temp->data<<"\n";
            delete temp;
            }
        }
        void display() //to show the stack
        {
            struct node *ptr1=top;
            if(top==NULL)
            {
                cout<<"Stack is empty.\n\n";
            }
            else
            {
                cout<<"Stack :\n";
                while(ptr1!=NULL)
                {
```

```
cout<<ptr1->data<<"\n";
                    ptr1=ptr1->next;
                }
            }
        }
};

int main()
{
    stack s;
    int ch,x;
    while(ch!=4)
    {
        cout<<"1.Push\n2.Pop\n3.Display\n4.Exit\n";
        cout<<"Enter the value for operation :\n";
        cin>>ch;
        switch(ch)
        {
            case 1:
                cout<<"Enter the value :\n";
                cin>>x;
                s.push(x);
                break;
            case 2:
                s.pop();
                break;
            case 3:
                s.display();
                break;
            case 4:
                break;
            default:
            cout<<"\nWrong choice.\n\n";

        }
    }
    return 0;
}
```

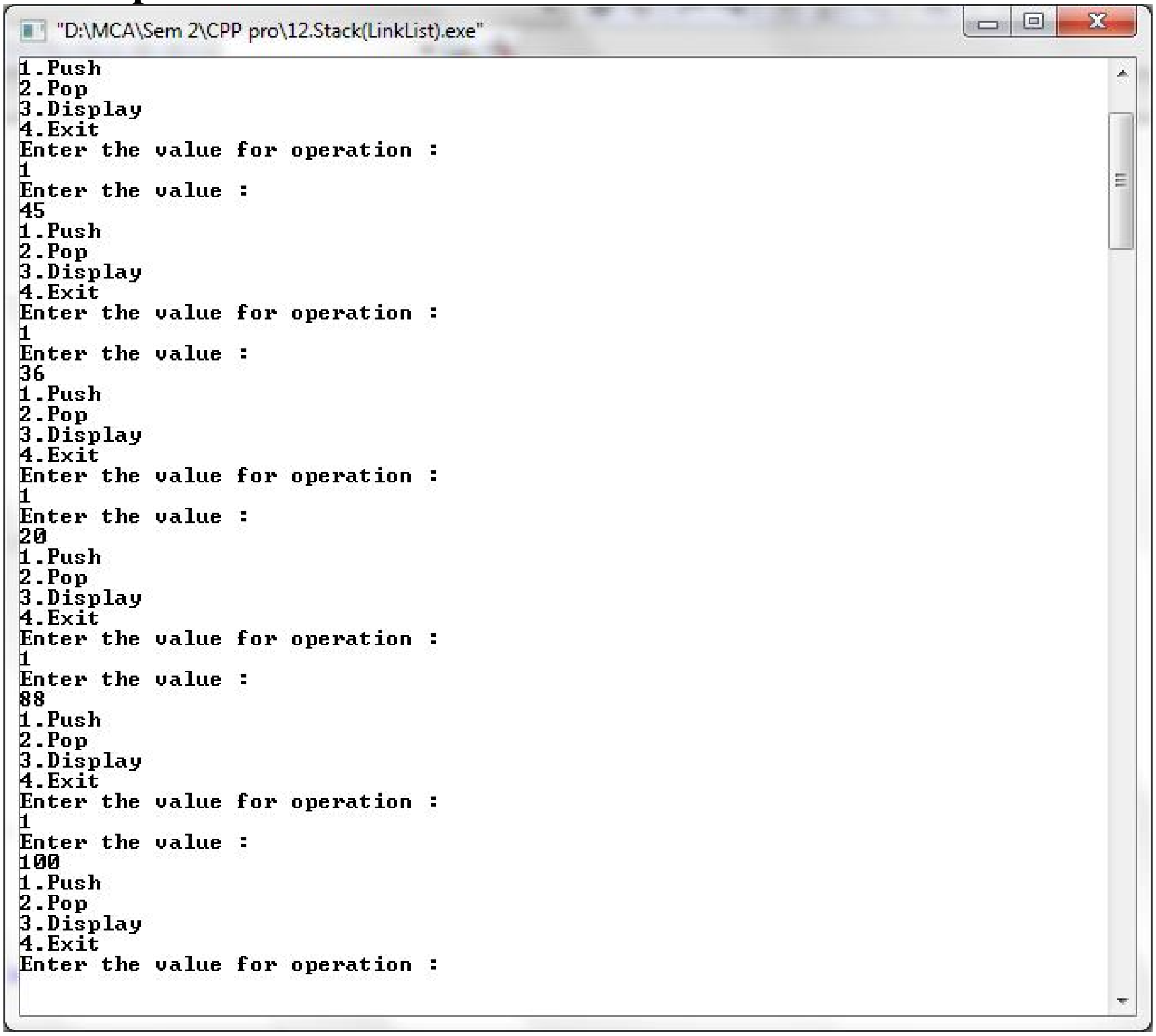
Output:

Empty stack:

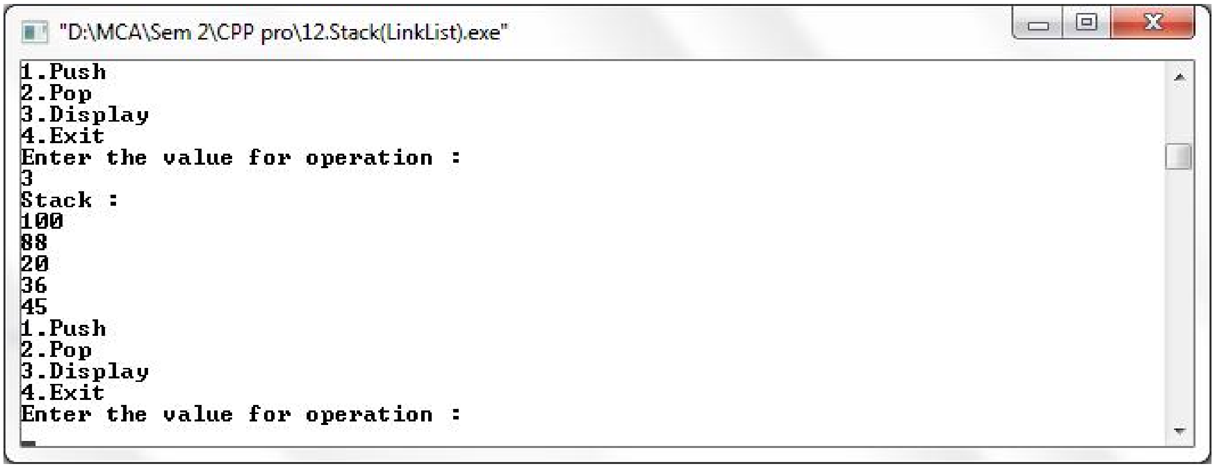


```
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
3
Stack is empty.

1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
_
```

Push Operation:

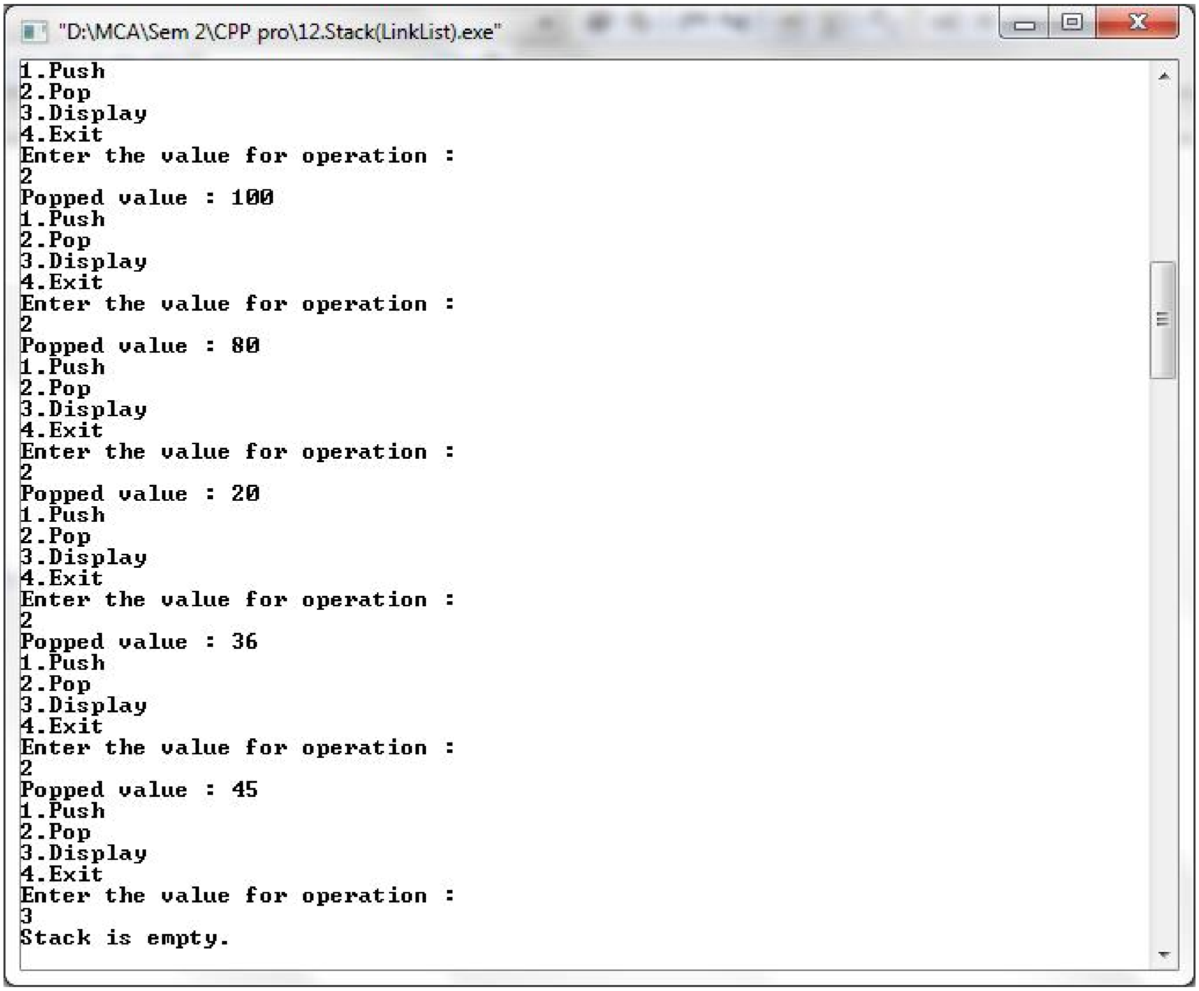


```
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
1
Enter the value :
45
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
1
Enter the value :
36
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
1
Enter the value :
20
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
1
Enter the value :
88
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
1
Enter the value :
100
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
```



```
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
3
Stack :
100
80
20
36
45
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
_
```

Pop Operation:



```
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
2
Popped value : 100
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
2
Popped value : 80
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
2
Popped value : 20
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
2
Popped value : 36
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
2
Popped value : 45
1.Push
2.Pop
3.Display
4.Exit
Enter the value for operation :
3
Stack is empty.
```

PRACTICAL 5

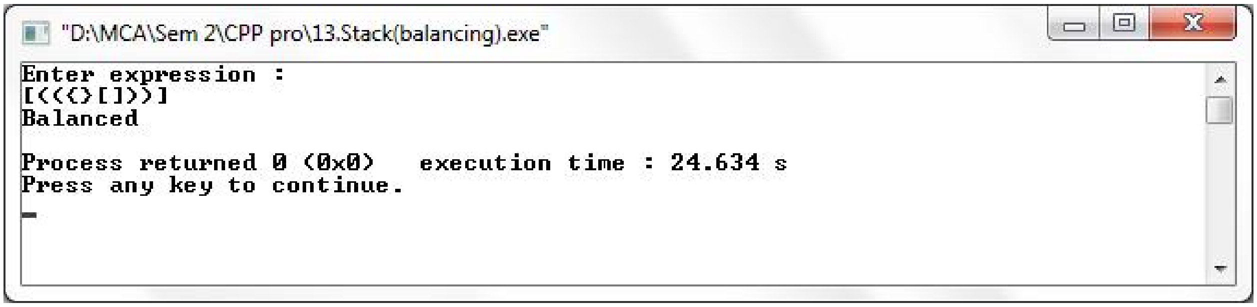
Aim: Implementation of Stack Application like Balancing of Parenthesis

```
#include<iostream>
#include<string.h>
#include<conio.h>
using namespace std;

struct node
    {
        int data;
        struct node *next;
    };
    struct node *tmp=NULL;
    struct node *tmp1=NULL;
    struct node *top=NULL;
    struct node *ptr=NULL;
    int push(char x)
    {
        tmp = new node;
        tmp->data=x;
        tmp->next=NULL;
        if(top == NULL)
        {
            top=tmp;
        }
        else
        {
            tmp1=top;
            top=tmp;
            tmp->next=tmp1;
        }
    }

    char pop()
    {
        if(top==NULL)
        {
            cout<<"Stack is empty.\n";
        }
        else
        {
            ptr=top;
            top=top->next;
            return(ptr->data);
            delete(ptr);
        }
      }
```

```
int main()
{

int len,i;
    char c,d,e;
    char a[30];
    cout<<"Enter expression :\n";
    cin>>a;
    len=strlen(a);

    for(i=0;i<len;i++)
    {
        if(a[i]=='{' || a[i]=='[' || a[i]=='(')
            {
                push(a[i]);
            }
    else
    {
            switch(a[i])
            {
                case ')':
                c=pop();
                if(c=='{' || c=='[')
                    {
                        cout<<"Invalid";
                        getch();
                    }
                    break;

                case ']':
                d=pop();
                if(d=='{' || d=='(')
                    {
                        cout<<"Invalid";
                        getch();
                    }
                    break;

                case '}':
                e=pop();
                if(e=='(' || e=='[')
                    {
                        cout<<"Invalid";
                        getch();
                    }
                    break;
                default:
                    cout<<"Enter the correct choice";
                    getch();
            }
      }
    }
    if(top==NULL)
        cout<<"Balanced\n";
    else
        cout<<"Unbalanced\n";
        getch();
        return 0;
}
```

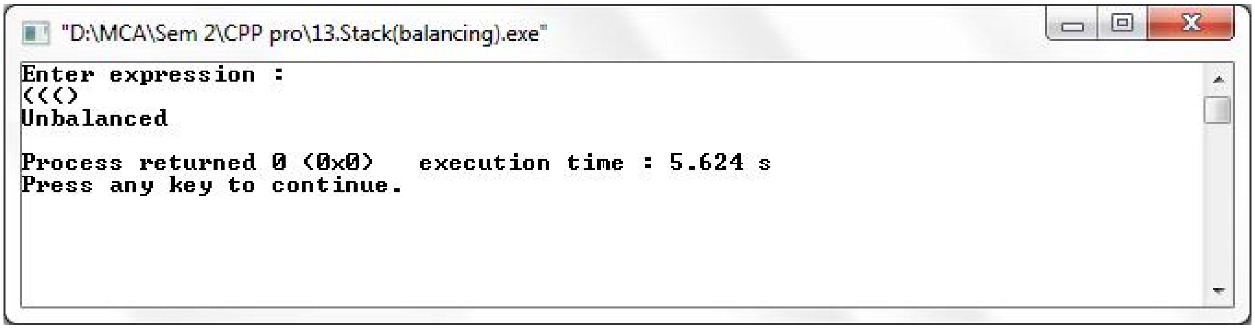
Output :

Balanced



```
Enter expression :
[((()[]))]
Balanced

Process returned 0 (0x0)   execution time : 24.634 s
Press any key to continue.
_
```

Unbalanced



```
Enter expression :
((()
Unbalanced

Process returned 0 (0x0)   execution time : 5.624 s
Press any key to continue.
```

PRACTICAL 6

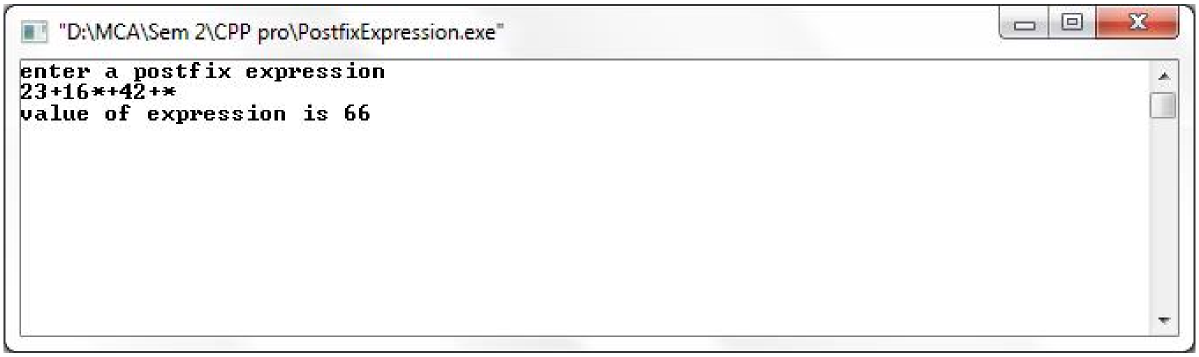
Aim: Implementation of Stack Application like Postfix Evaluation

```
#include<iostream>
#include<conio.h>
#include<string.h>
#include<math.h>
using namespace std;
class postfix
{
public:
int top;
char p[50];
long int A[50];
postfix()
{
top=-1;
}
void input();
void push(long int s);
long int pop();
int full();
int empty();
long int eval_post();
};
void postfix::input()
{
cout<<"enter a postfix expression\n";
cin>>p;
}
int postfix::full()
{
if(top==49)
return 1;
else
return 0;
}
void postfix::push(long int s)
{
if(full())
cout<<"overflow\n";
else
{
top=top+1;
A[top]=s;
}
}
int postfix::empty()
{
```

```
if(top==-1)
return 1;
else
return 0;
}
long int postfix::pop()
{
if(empty())
cout<<"underflow\n";
else
return(A[top--]);
}
long int postfix::eval_post()
{
long int a,b,temp,result,len;
int i;
len=strlen(p);
p[len]='#';
for(i=0;p[i]!='#';i++)
{
if(p[i]<='9'&&p[i]>='0')
push(p[i]-48);
else
{
a=pop();
b=pop();
switch(p[i])
{
case '+':
temp=b+a;
break;
case '-':
temp=b-a;
break;
case '*':
temp=b*a;
break;
case '/':
temp=b/1;
break;
case '%':
temp=b%a;
break;
case '^':
temp=pow(b,a);
}
push(temp);
}
}
result=pop();
return result;
```

```
}
main()
{
long int value;
postfix f;
f.input();

value=f.eval_post();
cout<<"value of expression is "<<value;
getch();
}
```

Output:



```
enter a postfix expression
23+16**+42+*
value of expression is 66
```

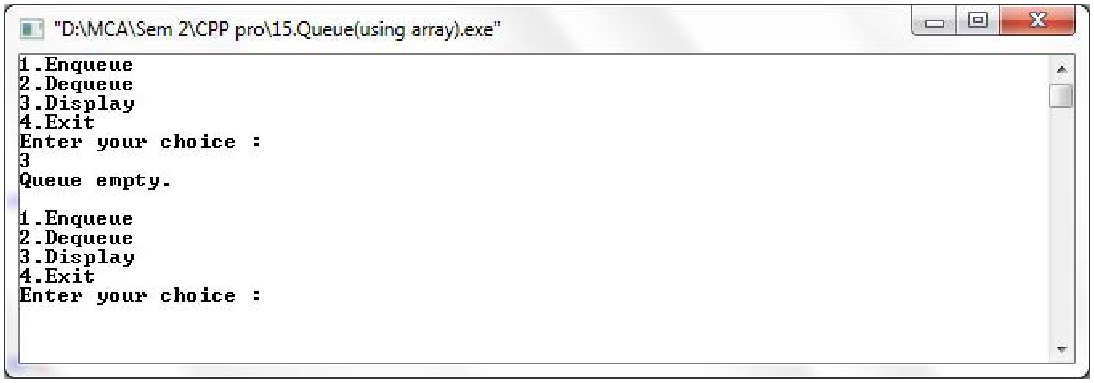
PRACTICAL 7

Aim: Write a program for Queue using Array.

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
class queue
{
    public:
    int q[4],x,result;
    int front=-1;
    int rear=-1;
    int maxsize=4;
    void enqueue(int x)
    {
        if(rear==maxsize-1)
        {
            cout<<"Queue full.\n\n";
        }
        else
        {
            rear++;
            q[rear]=x;
        }
    }
    void dequeue()
    {
        if(rear==-1)
        {
            cout<<"Queue empty.\n\n";
        }
        else
        {
            cout<<"Deleted.\n";
            if(front==rear)
            {
                front=-1;
                rear=-1;
            }
            else
            {
               x=q[front];
               front++;
            }
        }
    }
    void display()
    {
        if(rear==-1)
        {
            cout<<"Queue empty.\n\n";
        }
        else
        {

cout<<"Queue :\n";
            for(int i=front+1;i<=rear;i++)
            {
                cout<<q[i]<<"\n";
            }
        }
    }
};

int main()
{
    int ch,x;
    queue q;
    do
    {
        cout<<"1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n";
        cout<<"Enter your choice :\n";
        cin>>ch;
        switch(ch)
        {
            case 1:
                cout<<"Enter the value :\n";
                cin>>x;
                q.enqueue(x);

                break;
            case 2:
                q.dequeue();
                break;
            case 3:
                q.display();
                break;
            case 4:
                break;
            default:
            cout<<"\nInvalid choice!!\n";
        }
    }
  while(ch<4);
return 0;
}
```

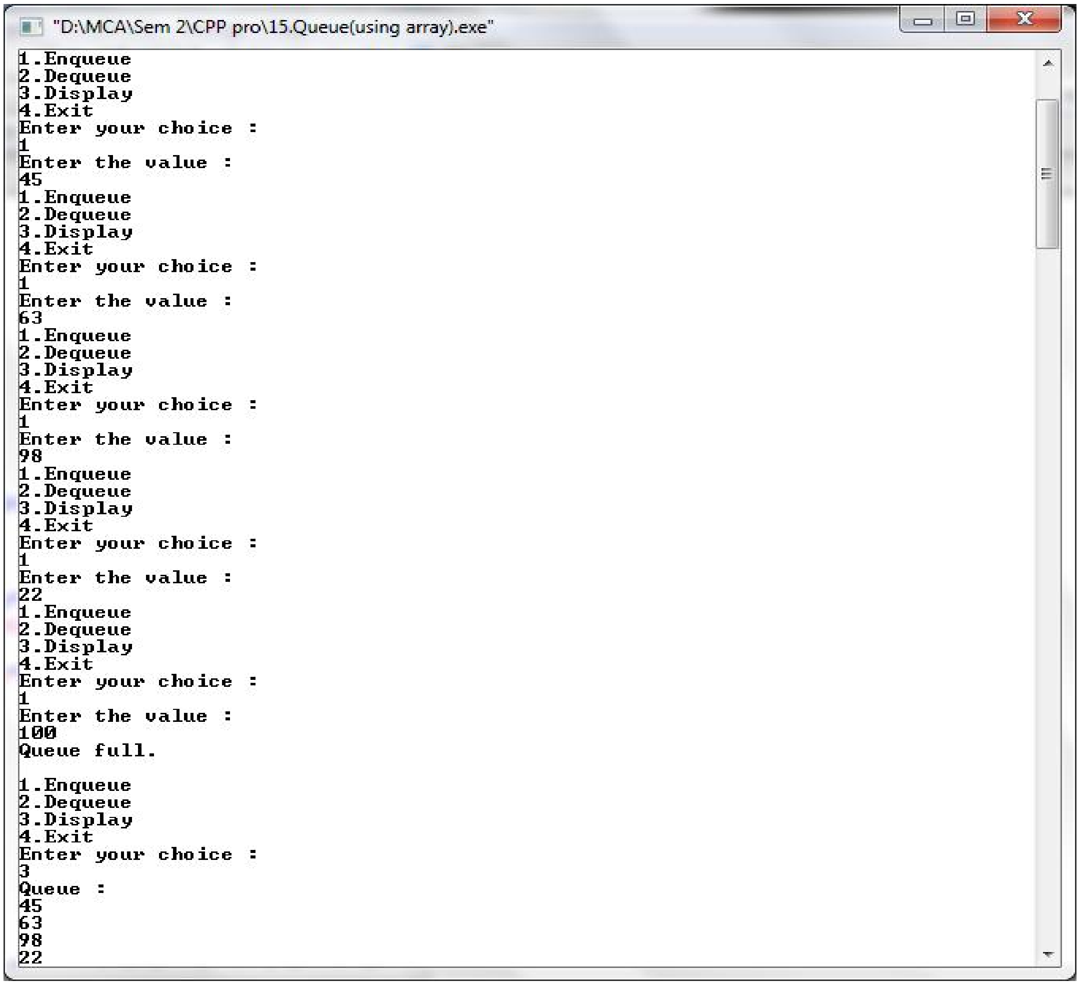
Output:

Queue Empty



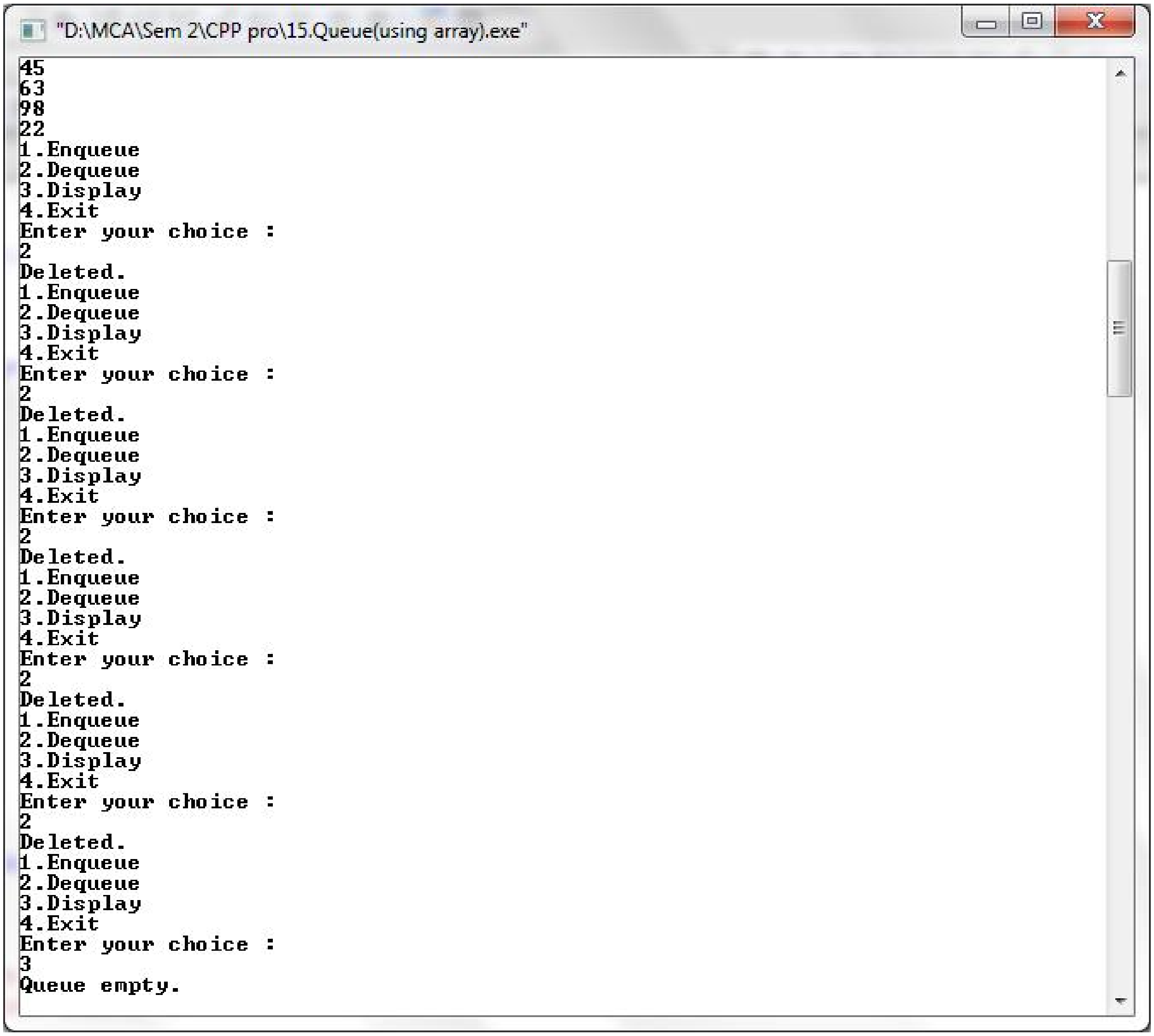
```
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice :
3
Queue empty.

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice :
```

Insert



```
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice :
1
Enter the value :
45
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice :
1
Enter the value :
63
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice :
1
Enter the value :
98
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice :
1
Enter the value :
22
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice :
1
Enter the value :
100
Queue full.

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice :
3
Queue :
45
63
98
22
```

Delete



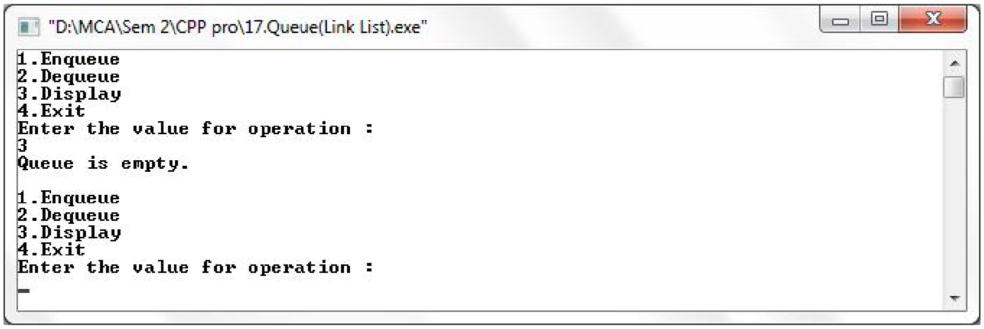
```
45
63
98
22
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice :
2
Deleted.
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice :
2
Deleted.
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice :
2
Deleted.
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice :
2
Deleted.
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice :
2
Deleted.
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice :
3
Queue empty.
```

PRACTICAL 8

Aim: Write a program for Queue using Linked List.

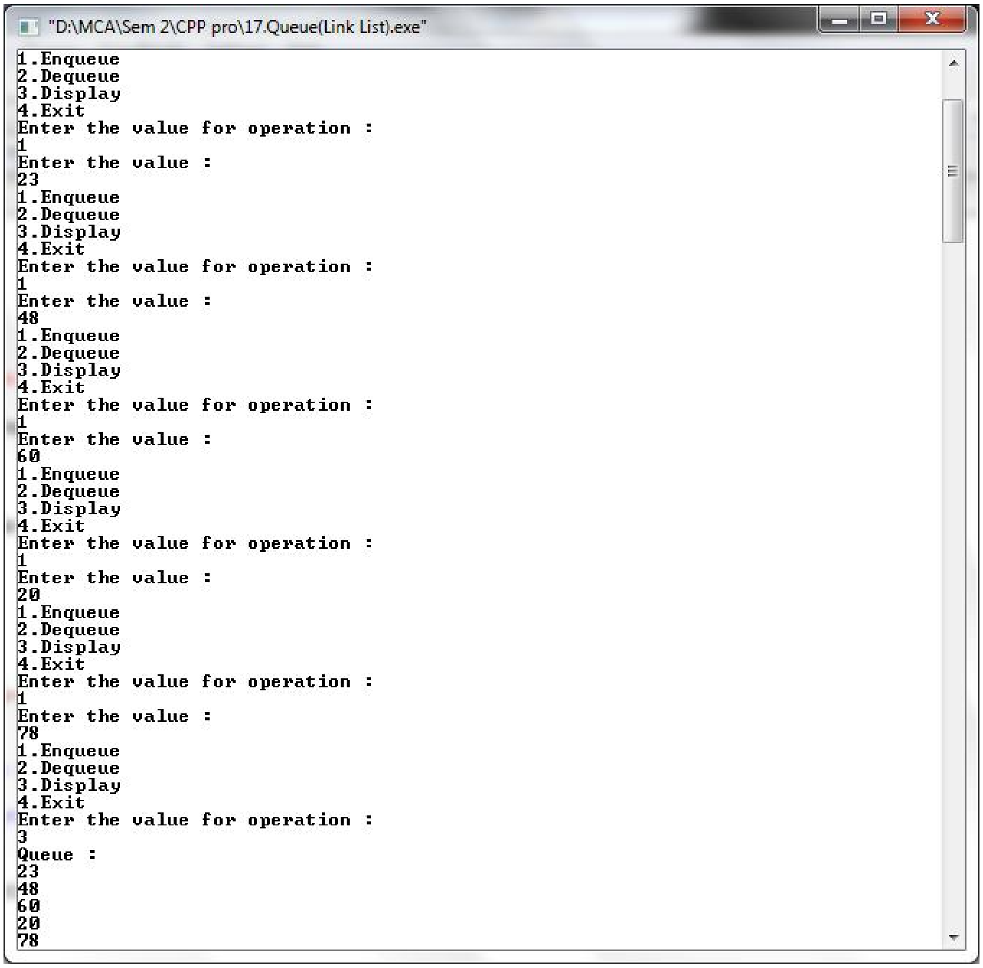
```
#include<iostream>
using namespace std;
class linkqueue
{
public:
 struct node
    {
        int data;
        struct node *next;
    };
    node *front = NULL;
    node *rear = NULL;
    void enqueue(int x)
    {
        node *tmp, *q;
        tmp=new node;
        tmp->data=x;
        tmp->next=NULL;
        if(front==NULL && rear==NULL)
        {
            front=rear=tmp;
        }
        else
        {
           rear->next=tmp;
           rear=tmp;

        }
    }
    void dequeue()
    {
        struct node *tmp=front;
        if(front==NULL)
        {
            cout<<"Queue is empty\n\n";
        }
        cout<<"Deleted\n";
        if(front==rear)
        {
            front=rear=NULL;
        }
        else
        {
            front=front->next;
        }
    }
    void display()
    {
        if(front==NULL)

        {

            cout<<"Queue is empty.\n\n";
        }
        else
        {
            node *ptr;
            ptr=front;
            cout<<"Queue :\n";
            while(ptr!=NULL)
            {
                cout<<ptr->data<<"\t";
                cout<<"\n";
                ptr=ptr->next;
            }
        }
    }
};

int main()
{
    linkqueue q;
    int ch,x,n;
    do
    {
        cout<<"1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n";
        cout<<"Enter the value for operation :\n";
        cin>>ch;
        switch(ch)
        {
            case 1:
                cout<<"Enter the value :\n";
                cin>>x;
                q.enqueue(x);
                break;
            case 2:
                q.dequeue();
                break;
            case 3:
                q.display();
                break;
           case 4:
                break;
            default:
            cout<<"\nWrong choice.\n\n";
        }
    }
    while(n!=4);
  return 0;}
```

Output :
Queue Empty



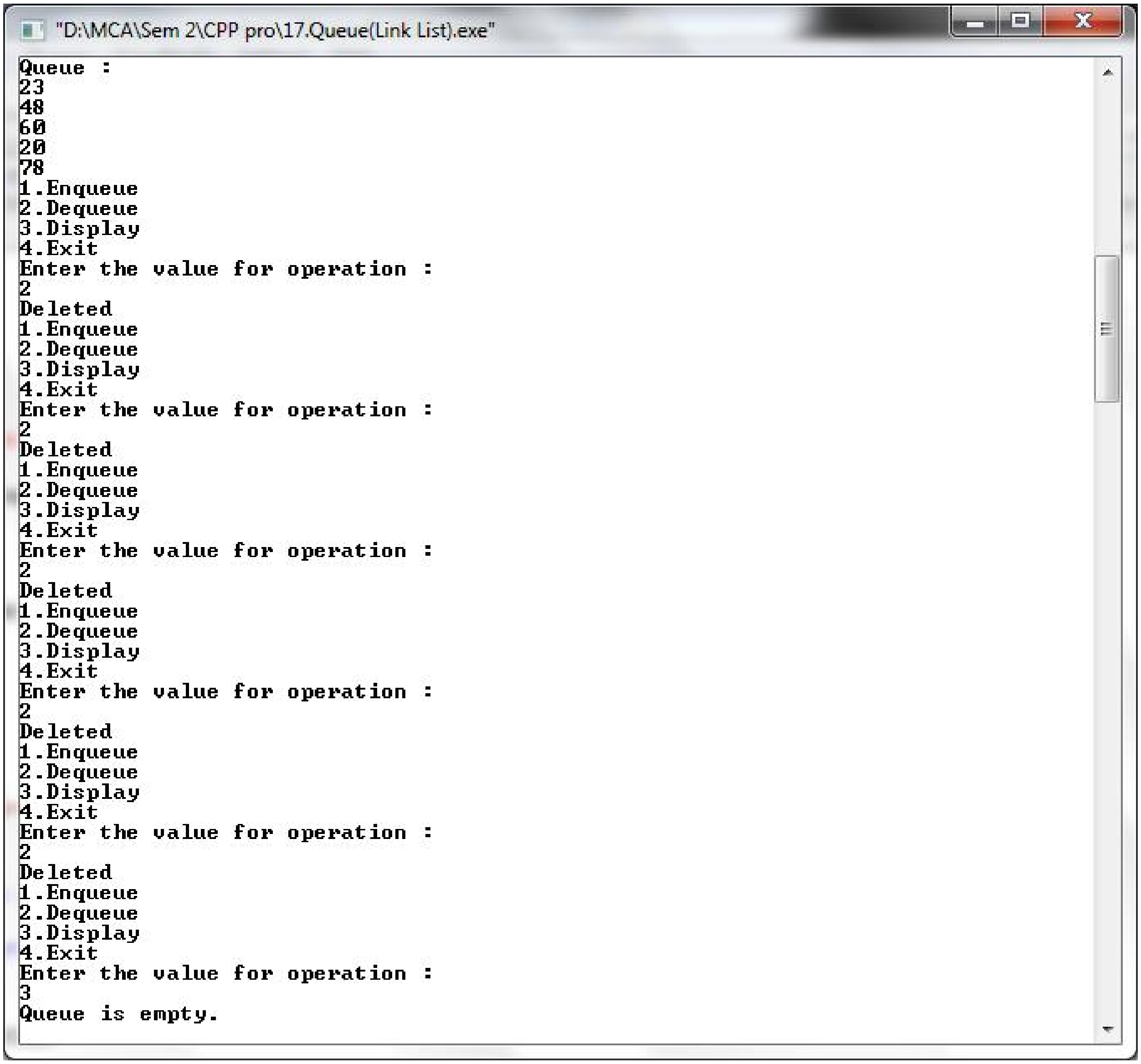
```
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter the value for operation :
3
Queue is empty.

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter the value for operation :
_
```

Insert



```
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter the value for operation :
1
Enter the value :
23
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter the value for operation :
1
Enter the value :
48
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter the value for operation :
1
Enter the value :
60
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter the value for operation :
1
Enter the value :
20
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter the value for operation :
1
Enter the value :
78
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter the value for operation :
3
Queue :
23
48
60
20
78
```

Delete



```
Queue :
23
48
60
20
78
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter the value for operation :
2
Deleted
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter the value for operation :
2
Deleted
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter the value for operation :
2
Deleted
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter the value for operation :
2
Deleted
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter the value for operation :
2
Deleted
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter the value for operation :
3
Queue is empty.
```

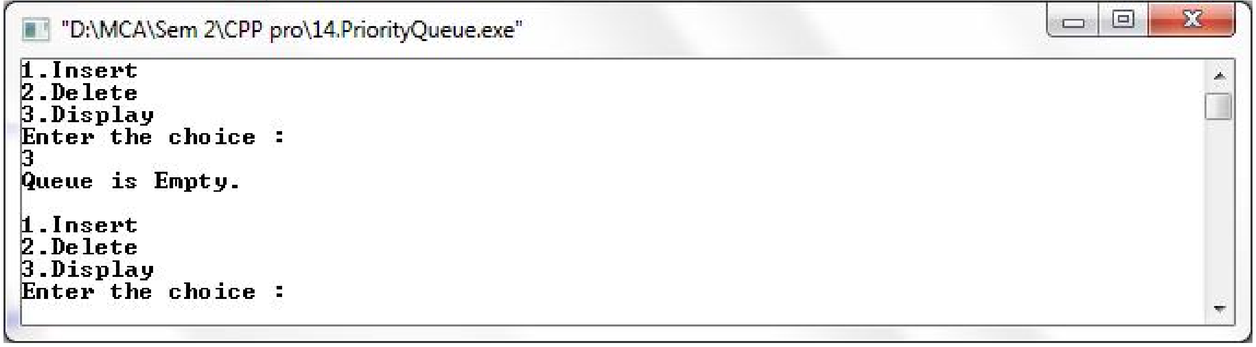
PRACTICAL 9

Aim: Demonstrate Application of Priority Queue.

```
#include<iostream>
using namespace std;

class priority
{
    public:
    struct node
    {
        int pr;
        int data;
        struct node *next;
    };
    node *front=NULL;

    void insert(int item, int pr)
    {
        node *tmp, *q;
        tmp=new node;
        tmp->data=item;
        tmp->pr=pr;
        if(front==NULL || pr < front->pr)
        {
            tmp->next=front;
            front=tmp;
        }
        else
        {
            q=front;
            while(q->next!=NULL && q->next->pr<=pr)
                q=q->next;

            tmp->next = q-> next;
            q->next=tmp;
        }
    }

    void del()
    {
        node *tmp,*q;
        if(front==NULL)
            cout<<"Queue is Empty";
        else
        {
            tmp=front;
            cout<<tmp->data<<" is deleted \n";
            front=front->next;
        }
    }

    void display()
    {

if(front==NULL)
        {
            cout<<"Queue is Empty.\n\n";
        }
        else
        {
            node *ptr;
            ptr=front;
            cout<<"Item\t"<<"Priority\n";
            while(ptr!=NULL)
            {
                cout<<ptr->data<<"\t";
                cout<<ptr->pr<<"\t";
                cout<<"\n";
                ptr=ptr->next;
            }
        }
    }
};

int main()
{
    priority p;
    int ch,x,y;

    while(ch!=0)
    {
        cout<<"1.Insert\n2.Delete\n3.Display\n";
        cout<<"Enter the choice :\n";
        cin>>ch;
        switch(ch)
        {
        case 1:
            cout<<"Enter the Item :\n";
            cin>>x;
            cout<<"Enter the Priority :\n";
            cin>>y;
            p.insert(x,y);
            break;
        case 2:
            p.del();
            break;
        case 3:
            p.display();
            break;
        default:

cout<<"Enter the correct choice";
            break;
        }

    }
    return 0;
}
```

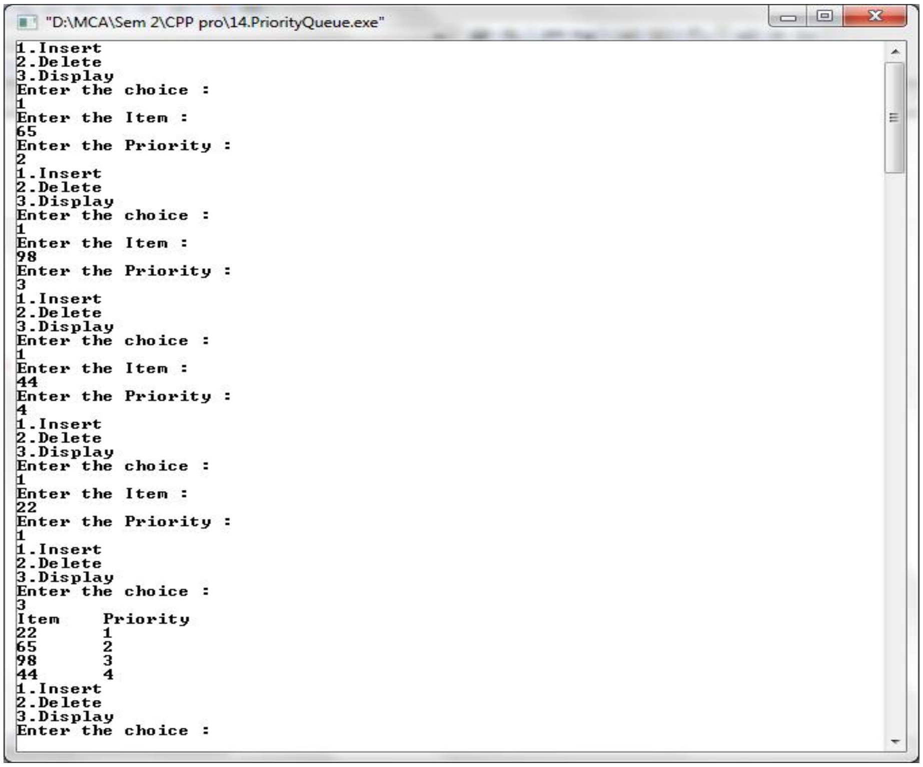
Output :

Queue is empty



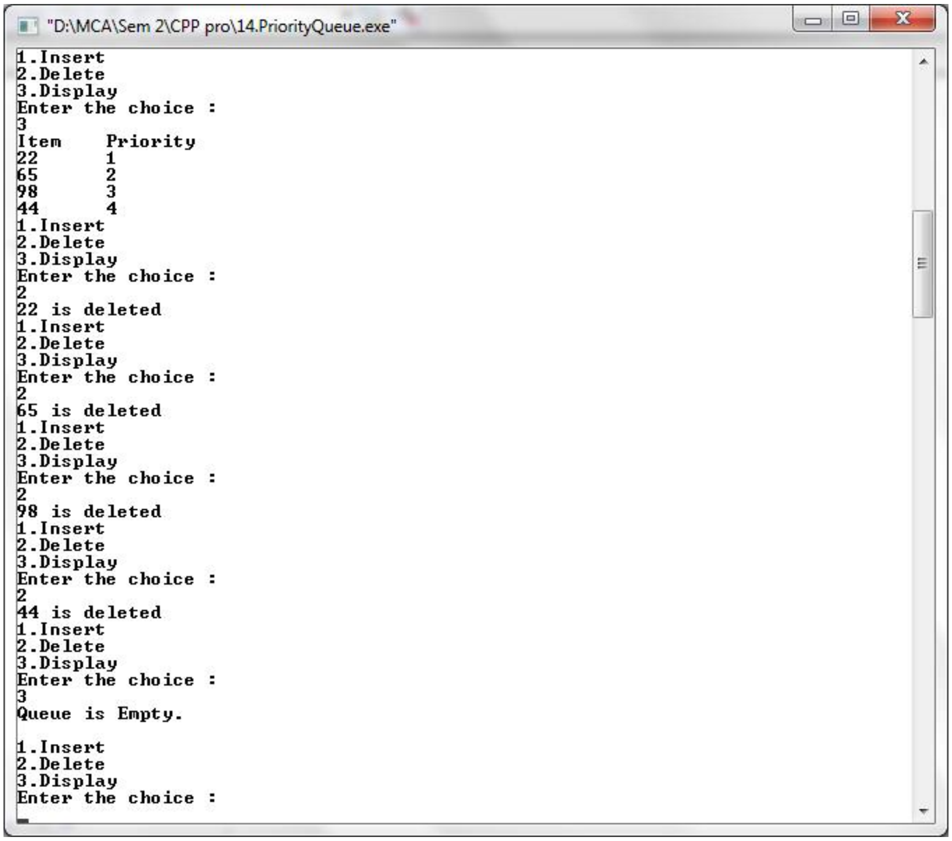
```
1.Insert
2.Delete
3.Display
Enter the choice :
3
Queue is Empty.

1.Insert
2.Delete
3.Display
Enter the choice :
```

Insert



```
1.Insert
2.Delete
3.Display
Enter the choice :
1
Enter the Item :
65
Enter the Priority :
2
1.Insert
2.Delete
3.Display
Enter the choice :
1
Enter the Item :
98
Enter the Priority :
3
1.Insert
2.Delete
3.Display
Enter the choice :
1
Enter the Item :
44
Enter the Priority :
4
1.Insert
2.Delete
3.Display
Enter the choice :
1
Enter the Item :
22
Enter the Priority :
1
1.Insert
2.Delete
3.Display
Enter the choice :
3
Item    Priority
22      1
65      2
98      3
44      4
1.Insert
2.Delete
3.Display
Enter the choice :
```

Delete



```
1.Insert
2.Delete
3.Display
Enter the choice :
3
Item    Priority
22      1
65      2
98      3
44      4
1.Insert
2.Delete
3.Display
Enter the choice :
2
22 is deleted
1.Insert
2.Delete
3.Display
Enter the choice :
2
65 is deleted
1.Insert
2.Delete
3.Display
Enter the choice :
2
98 is deleted
1.Insert
2.Delete
3.Display
Enter the choice :
2
44 is deleted
1.Insert
2.Delete
3.Display
Enter the choice :
3
Queue is Empty.

1.Insert
2.Delete
3.Display
Enter the choice :
_
```

PRACTICAL 10

Aim: Implementation of Singly Linked Lists .

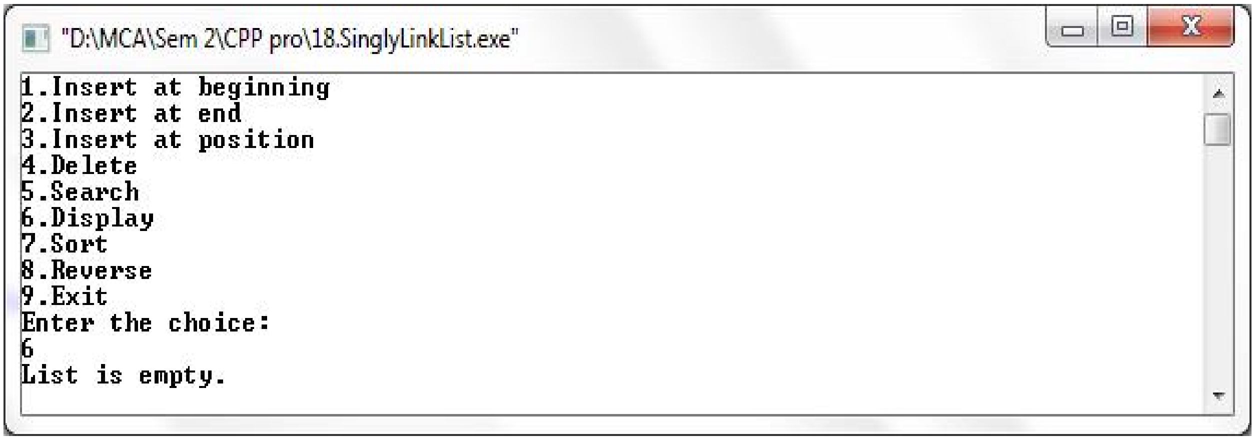
```
#include<iostream>
using namespace std;
class linklist
{
    public:
        int flag=true;
        int pos, i, value, count=0;
        struct node
        {
            int data;
            struct node *next;
        };
        struct node *tmp=NULL;
        struct node *tmp1=NULL;
        struct node *start=NULL;
        struct node *p=NULL;
        struct node *ptr=NULL;
        struct node *ptr1=NULL;
        struct node *ptr2=NULL;
        struct node *ptr3=NULL;

        int insert_at_beg(int x)
        {
            tmp = new node;
            tmp->data=x;
            tmp->next=NULL;
            if(start == NULL)
            {
                start=tmp;
                start->next=NULL;
            }
            else
            {
                tmp1=start;
                start=tmp;
                start->next=tmp1;
            }
        }

        int insert_at_end(int x)
        {
            tmp = new node;
            tmp->data=x;
            tmp->next=NULL;

   p=start;
            while(p->next!=NULL)
            {
                p=p->next;
            }
            p->next=tmp;
            tmp->next=NULL;

}

        int insert_at_pos(int x)
        {
            cout<<"Insert the position :\n";
            cin>>pos;
            tmp = new node;
            tmp->data=x;
            tmp->next=NULL;
            p=start;
            while(p!=NULL)
            {
                p=p->next;
                count++;
            }
            if(pos==1)
            {
                start=tmp;
                start->next=NULL;
            }
            else if(pos > 1 && pos<count)
            {
                p=start;
                for(i=1;i<pos;i++)
                {
                    ptr=p;
                    p=p->next;
                }

                ptr->next=tmp;
                tmp->next=p;
            }
            else
            {
                cout<<"Invalid position.\n";
            }
        }

        int search_pos()
        {
            cout<<"Insert the value:\n";
            cin>>value;

         count=0;
            flag=false;
            if(start==NULL)
```

```
    {
        cout<<"List is empty\n";
    }
    else
    {
        p=start;
        while(p!=NULL)
        {
            count++;
            if(p->data == value)
            {
                flag==true;
                cout<<"Value found at "<<count<<"
                       position.\n";
            }
            p=p->next;

        }
    }
}

void del()
{
    cout<<"Delete the position:\n";
    cin>>pos;
    if(start==NULL)
    {
        cout<<"List is empty.\n";
    }
    else
    {
    if(pos==1)
    {
        tmp=start;
        start=start->next;
        delete (tmp);
    }
    else
    {
        while(p!=NULL)
        {
            count++;
            p=p->next;
        }
    }
    if(pos>1 && pos<count)
    {
        p=start;
        for(i=1;i<pos;i++)
        {
            ptr=p;
            p=p->next;
        }
        ptr->next=p->next;
    }
    }
}
int sort()
{
    int x;
    if(start==NULL)
    {
        cout<<"List is empty.\n";
    }
    ptr=start;
    while(ptr!=NULL)
    {
        for(p=ptr->next;p!=NULL;p=p->next)
        {
            if(ptr->data>p->data)
            {
                x=ptr->data;
                ptr->data=p->data;
                p->data=x;
            }
        }
        ptr=ptr->next;
    }
}

int rev()
{
    if(start==NULL)
    {
        cout<<"List is empty.\n";
    }
    if(start->next==NULL)
    {
        cout<<"only one.\n";
    }
    ptr1=start;
    ptr2=ptr1->next;
    ptr3=ptr2->next;
    ptr1->next=NULL;
    ptr2->next=ptr1;
    while(ptr3!=NULL)
    {
        ptr1=ptr2;
        ptr2=ptr3;
        ptr3=ptr3->next;
        ptr2->next=ptr1;
    }
    start=ptr2;
}
```

```
        void display()
        {
            if(start==NULL)
            {
                cout<<"List is empty.\n";
            }
            else
            {
                p=start;
                cout<<"\nSingly Linked List :\n";
                while(p!=NULL)
                {
                    cout<< p->data<<" -> ";
                    p=p->next;
                }
                cout<<"\n";
            }
        }
};
int main()
{
    linklist l;
    int ch,x;

    while(ch!=0)
    {
        cout<<"\n1.Insert at beginning\n2.Insert at end\n3.Insert at 
position\n4.Delete\n5.Search\n6.Display\n7.Sort\n8.Reverse\n9.Exit";
        cout<<"\nEnter the choice:\n";
        cin>>ch;
    switch(ch)
    {
    case 1:
        cout<<"Enter the value\n";
        cin>>x; l.insert_at_beg(x);
        l.display();
        break;

    case 2:
        cout<<"Enter the value\n";
        cin>>x; l.insert_at_end(x);
        l.display();
        break;

    case 3:
        cout<<"Enter the value\n";
        cin>>x; l.insert_at_pos(x);
        l.display();
        break;
    case 4:
        l.del();
        l.display();
        break;

    case 5:
        l.search_pos();
        l.display();
        break;

    case 6:
        l.display();
        break;

    case 7:
        l.sort();
        l.display();
        break;

    case 8:
        l.rev();
        l.display();
        break;

    case 9:
        break;
           default:
           cout<<"Wrong choice\n";
    }
  }
 return 0;
}
```

Output :

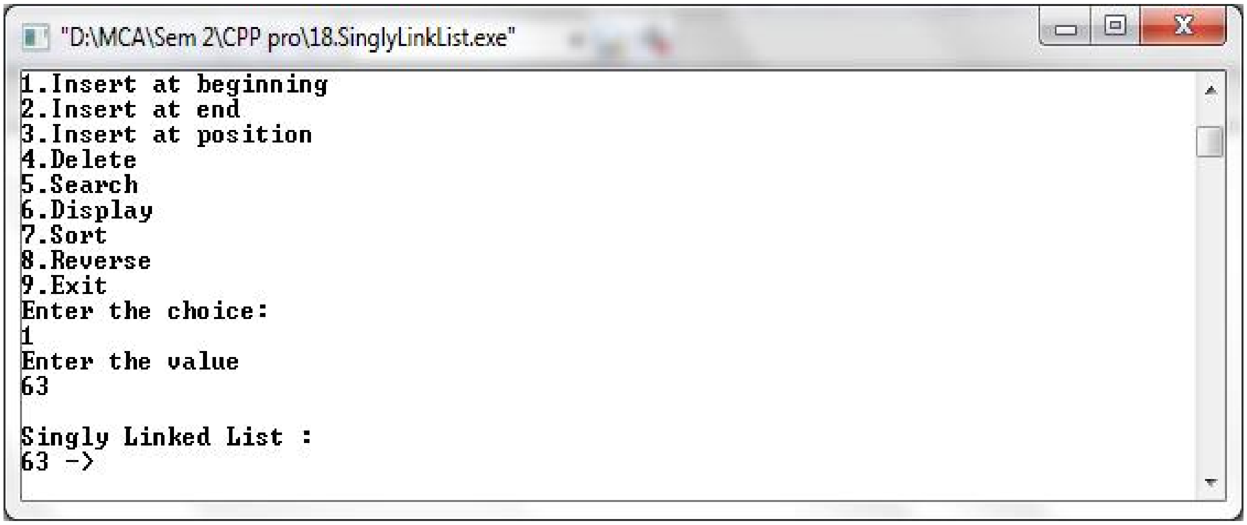
List Empty



"D:\MCA\Sem 2\CPP pro\18.SinglyLinkList.exe"

```
1.Insert at beginning
2.Insert at end
3.Insert at position
4.Delete
5.Search
6.Display
7.Sort
8.Reverse
9.Exit
Enter the choice:
6
List is empty.
```

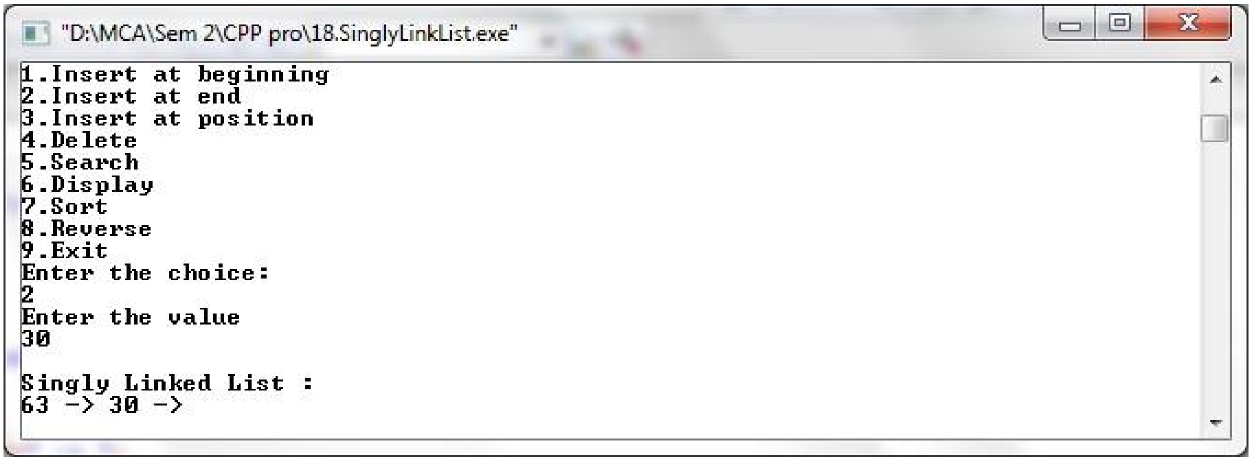
Insert at beginning



"D:\MCA\Sem 2\CPP pro\18.SinglyLinkList.exe"

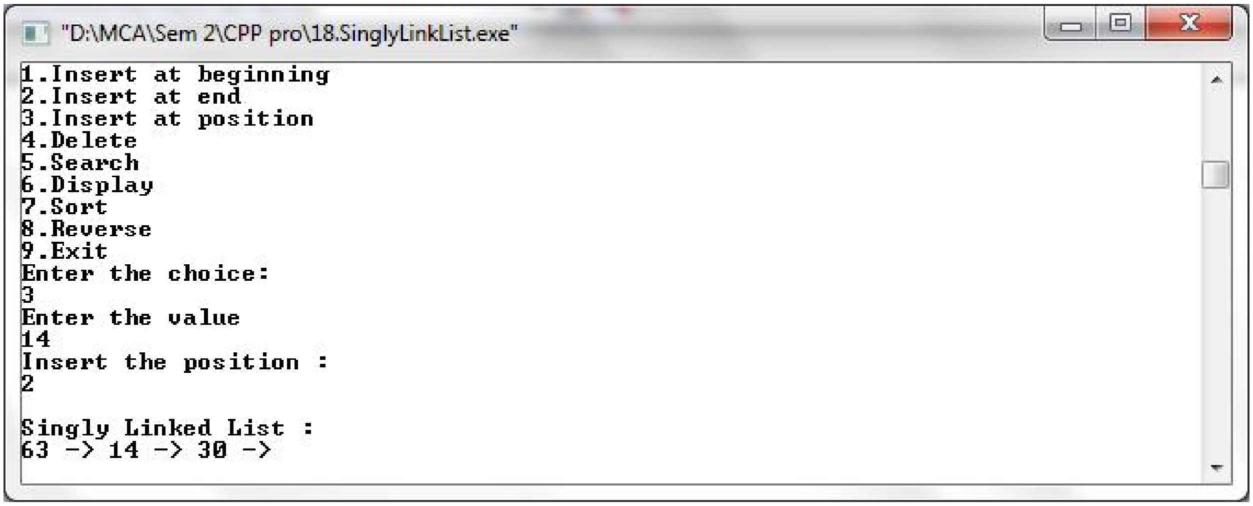
```
1.Insert at beginning
2.Insert at end
3.Insert at position
4.Delete
5.Search
6.Display
7.Sort
8.Reverse
9.Exit
Enter the choice:
1
Enter the value
63

Singly Linked List :
63 ->
```

Insert at end



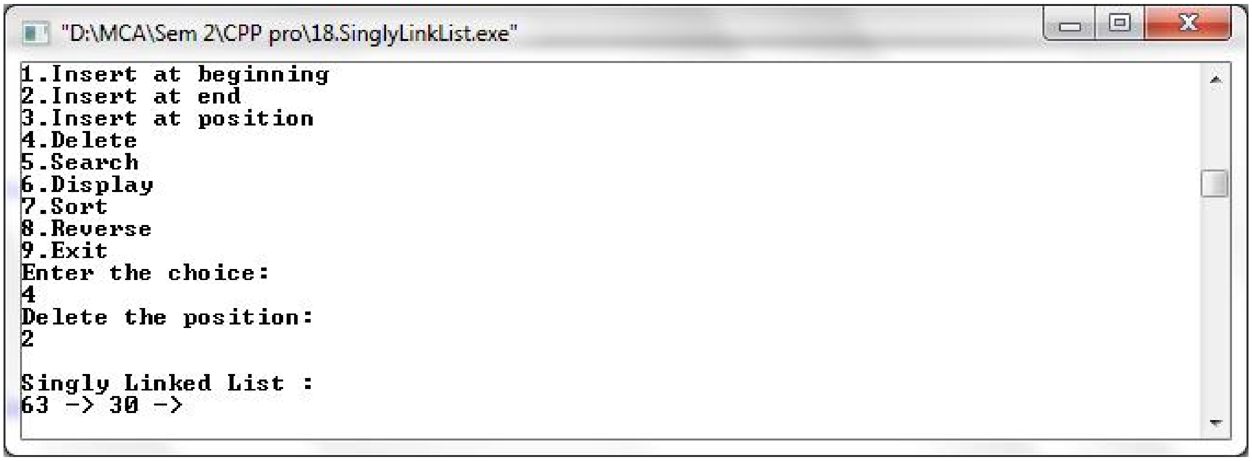
```
1.Insert at beginning
2.Insert at end
3.Insert at position
4.Delete
5.Search
6.Display
7.Sort
8.Reverse
9.Exit
Enter the choice:
2
Enter the value
30

Singly Linked List :
63 -> 30 ->
```

Insert at position



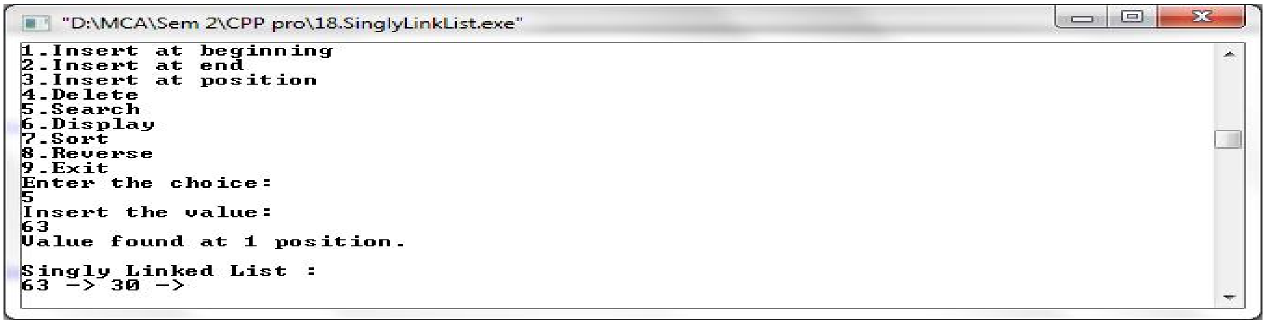
```
1.Insert at beginning
2.Insert at end
3.Insert at position
4.Delete
5.Search
6.Display
7.Sort
8.Reverse
9.Exit
Enter the choice:
3
Enter the value
14
Insert the position :
2

Singly Linked List :
63 -> 14 -> 30 ->
```

Delete



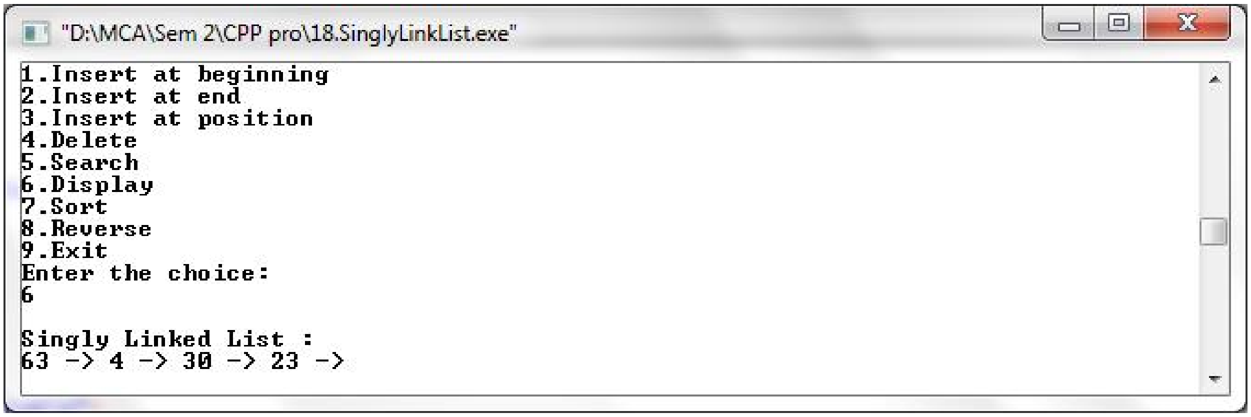
```
1.Insert at beginning
2.Insert at end
3.Insert at position
4.Delete
5.Search
6.Display
7.Sort
8.Reverse
9.Exit
Enter the choice:
4
Delete the position:
2

Singly Linked List :
63 -> 30 ->
```

Search



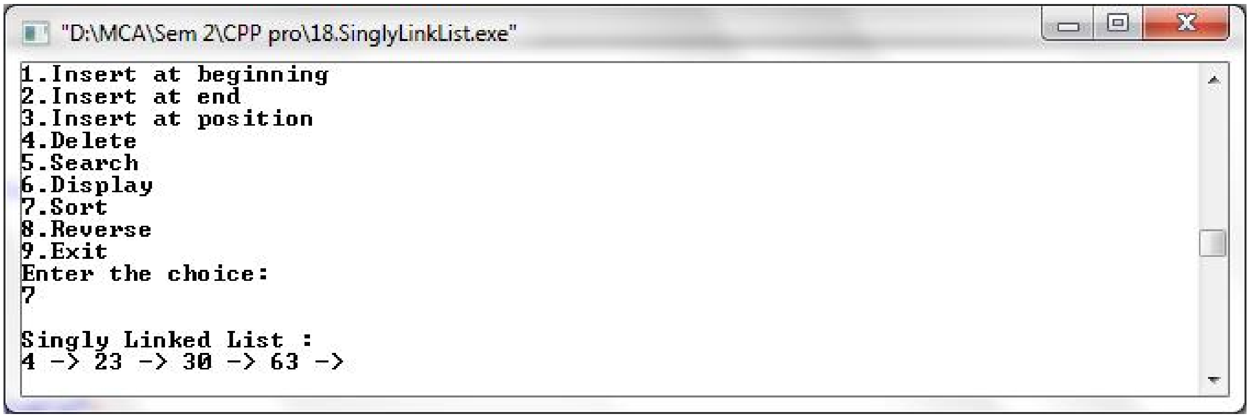
```
1.Insert at beginning
2.Insert at end
3.Insert at position
4.Delete
5.Search
6.Display
7.Sort
8.Reverse
9.Exit
Enter the choice:
5
Insert the value:
63
Value found at 1 position.

Singly Linked List :
63 -> 30 ->
```

Display



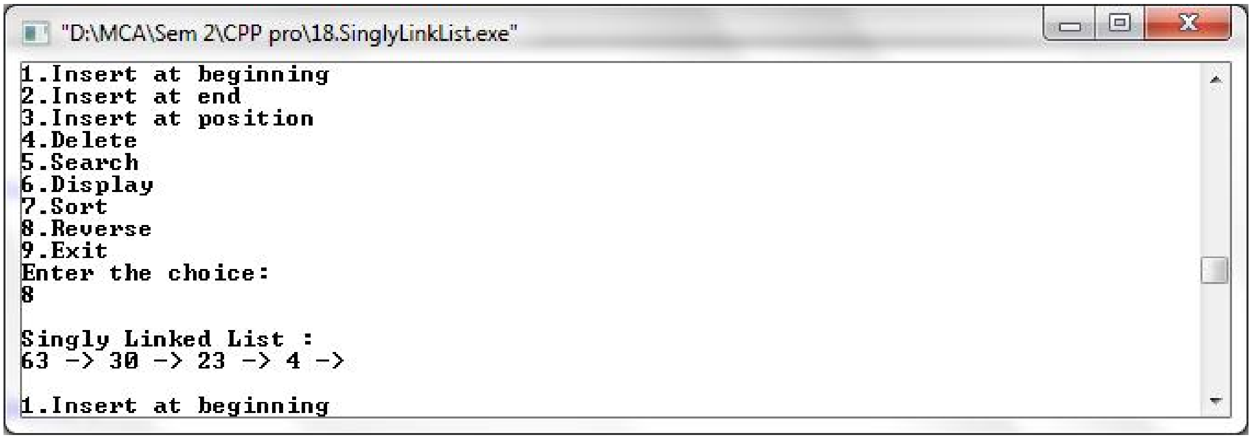
```
1.Insert at beginning
2.Insert at end
3.Insert at position
4.Delete
5.Search
6.Display
7.Sort
8.Reverse
9.Exit
Enter the choice:
6

Singly Linked List :
63 -> 4 -> 30 -> 23 ->
```

Sort



```
1.Insert at beginning
2.Insert at end
3.Insert at position
4.Delete
5.Search
6.Display
7.Sort
8.Reverse
9.Exit
Enter the choice:
7

Singly Linked List :
4 -> 23 -> 30 -> 63 ->
```

Reverse



```
1.Insert at beginning
2.Insert at end
3.Insert at position
4.Delete
5.Search
6.Display
7.Sort
8.Reverse
9.Exit
Enter the choice:
8

Singly Linked List :
63 -> 30 -> 23 -> 4 ->

1.Insert at beginning
```

PRACTICAL 11

Aim: Implementation of Doubly Linked Lists.

```
#include<iostream>
using namespace std;
class doubly
{
   public:


      struct node
      {
         int data;
         struct node *next;
         struct node *prev;
      };
      struct node *tmp=NULL;
      struct node *ptr=NULL;
      struct node *start=NULL;
      struct node *p=NULL;
      struct node *p1=NULL;
      struct node *p2=NULL;
      void create(int x)
      {
         tmp=new node;
         tmp->data=x;
         tmp->next=NULL;
         if(start==NULL)
         {
            tmp->prev=NULL;
            start=tmp;
         }
         else
         {
            p=start;
            while(p->next!=NULL)
            {
               p=p->next;
            }
            p->next=tmp;
            tmp->prev=p;
         }
      }
      void add_atbegin(int x)
```

```
{
   if(start==NULL)
   {
      cout<<"List is empty.\n\n";
   }
   tmp=new node;
   tmp->data=x;
   tmp->next=start;
   start->prev=tmp;


   start=tmp;
}
void add_after(int x,int pos)
{
   if(start==NULL)
   {
      cout<<"List is empty.\n\n";
   }
   p=start;
   for(int i=0;i<pos-1;i++)
   {
      p=p->next;
      if(p==NULL)
      {
         cout<<"Position does not exist.\n\n";
      }
   }
   tmp=new node;
   tmp->data=x;
   if(p->next==NULL)
   {
      p->next=tmp;
      tmp->next=NULL;
      tmp->prev=p;
   }
   else
   {
      tmp->next=p->next;
      tmp->next->prev=tmp;
      p->next=tmp;
      tmp->prev=p;
   }
}
void del(int x)
{
    if(start->data==x) //deleting first element
    {
        tmp=start;
        start=start->next;
        start->prev=NULL;
        delete(tmp);
    }
    p=start;
    while(p->next->next!=NULL) //deleting element in between
    {
        if(p->next->data==x)
        {
            tmp=p->next;
            p->next=tmp->next;
            tmp->next->prev=p;
            delete(tmp);
        }
        p=p->next;
    }
    if(p->next->data==x) //last element deleted
    {
        tmp=p->next;
        delete(tmp);
        p->next=NULL;
    }
}
void reverse()


{
    p1=start;
    p2=p1->next;
    p1->next=NULL;
    p1->prev=p2;
    while(p2!=NULL)
    {
        p2->prev=p2->next;
        p2->next=p1;
        p1=p2;
        p2=p2->prev;
    }
    start=p1;
    cout<<"List reversed.\n";
}
void count()
{
    p=start;
    int cnt=0;
    while(p!=NULL)
    {
        p=p->next;
        cnt++;
    }
    cout<<"Number of element are "<<cnt<<".\n";
}
void search()
{
    int count=0,value;
    int flag=0;
    cout<<"Enter the element to be searched :\n";
    cin>>value;
    if(start==NULL)
    {
        cout<<"List is empty.\n\n";
    }
    else
    {
        p=start;
        while(p!=NULL)
        {
            count++;
            if(p->data==value)
            {
               flag=1;
               cout<<"Element found at position "<<count<<".\n";
            }
            p=p->next;
        }
    }
}
void sort()
{
    if(start==NULL)
    {
        cout<<"list is empty.\n\n";
    }
    ptr=start;
    while(ptr!=NULL)
    {
        for(p=ptr->next;p!=NULL;p=p->next)
```

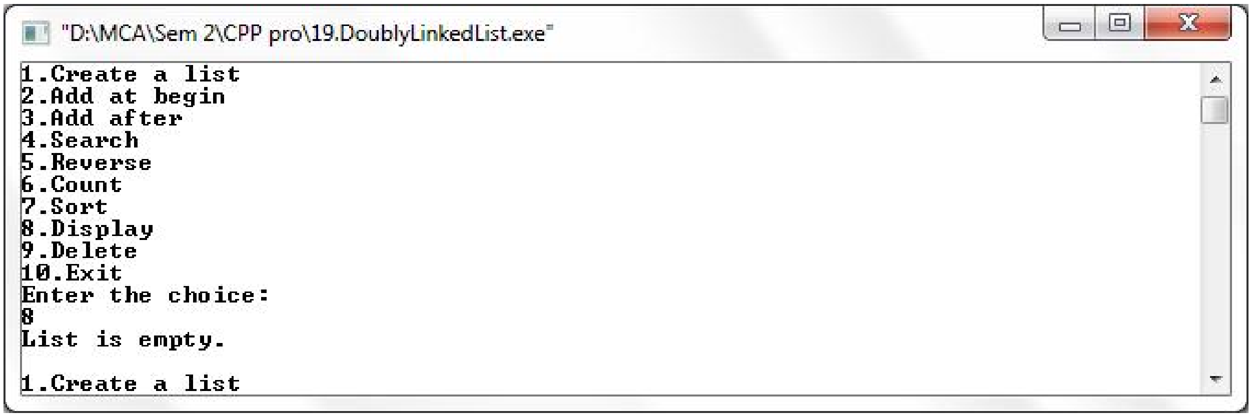
```
            {
               if(ptr->data > p->data)
               {
                  int x=ptr->data;
                  ptr->data = p->data;
                  p->data = x;
               }
            }
            ptr=ptr->next;
         }
      }
      void display()
      {
         if(start==NULL)
         {
            cout<<"List is empty.\n\n";
            return;
         }
         else
         {
            p=start;
            cout<<"\nDoubly Linked List :\n";
            while(p!=NULL)
            {
               cout<<p->data<<" -> ";
               p=p->next;
            }
            cout<<"\n\n";
         }
      }
};
int main()
{
   doubly d;
   int x,ch;
   int pos;


   while(ch!=0)
   {
      cout<<"1.Create a list\n2.Add at begin\n3.Add
after\n4.Search\n5.Reverse\n6.Count\n7.Sort\n";
      cout<<"8.Display\n9.Delete\n10.Exit";
      cout<<"\nEnter the choice:\n";
      cin>>ch;
```

```
switch(ch)
{
case 1:
   cout<<"Enter the value :\n";
   cin>>x;
   d.create(x);
   d.display();
   break;


case 2:
   cout<<"Enter the value :\n";
   cin>>x;
   d.add_atbegin(x);
   d.display(); break;


case 3:


   cout<<"Enter the position :\n";
   cin>>pos;
   cout<<"Enter the value :\n";
   cin>>x;
   d.add_after(x,pos);
   d.display();
   break;


case 4:
   d.search();
   d.display();
   break;


case 5:
   d.reverse();
   d.display();
   break;


case 6:
   d.count();
   d.display();
```

```
            break;


        case 7:
            cout<<"Before sorting -";
            d.display();
            d.sort();
            cout<<"After sorting -";
            d.display();
            break;


        case 8 :
            d.display();
            break;


        case 9:
            cout<<"Enter the element to be delete :\n";
            cin>>x;
            d.del(x);
            d.display();
            break;


        case 10:
            break;


        default:
            cout<<"Wrong choice.\n";
        }
    return 0;
```

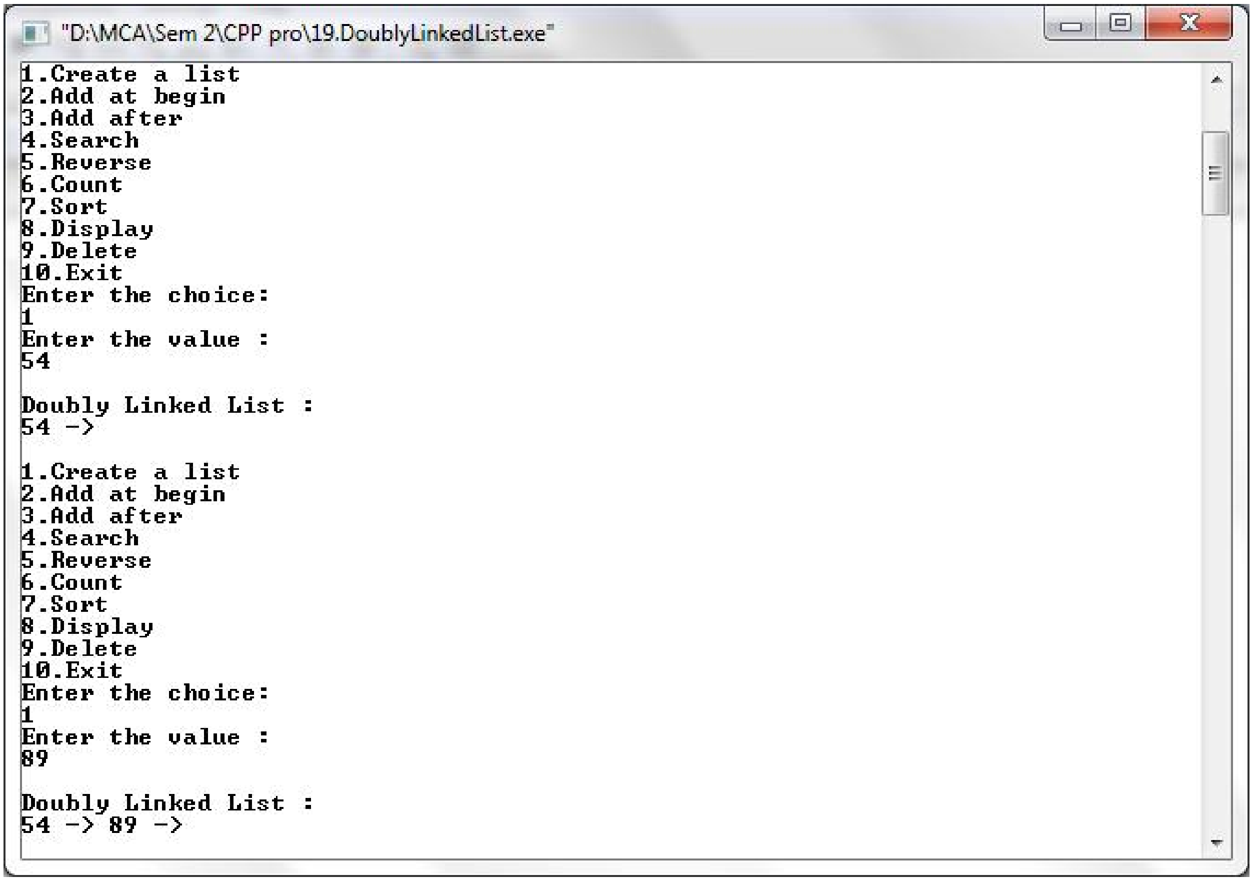
Output :

List is empty



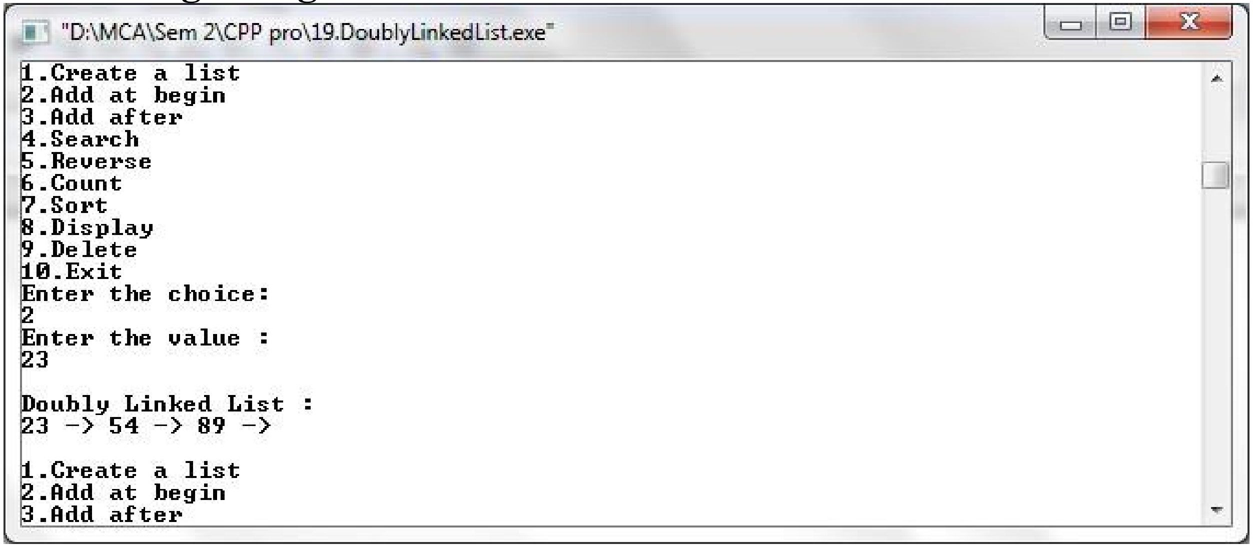
```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Reverse
6.Count
7.Sort
8.Display
9.Delete
10.Exit
Enter the choice:
8
List is empty.

1.Create a list
```

Create List



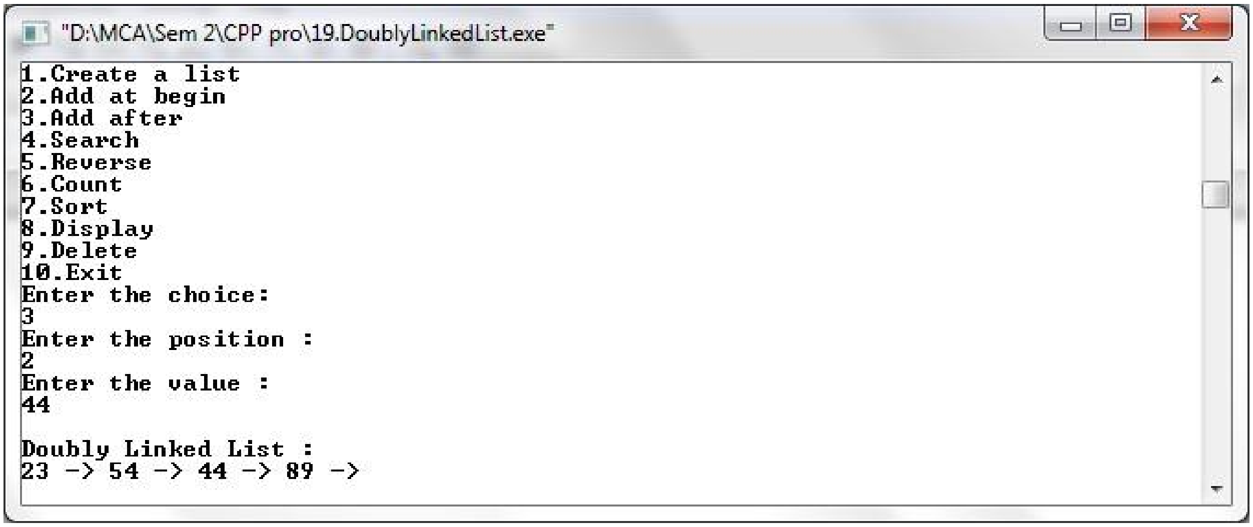
```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Reverse
6.Count
7.Sort
8.Display
9.Delete
10.Exit
Enter the choice:
1
Enter the value :
54

Doubly Linked List :
54 ->

1.Create a list
2.Add at begin
3.Add after
4.Search
5.Reverse
6.Count
7.Sort
8.Display
9.Delete
10.Exit
Enter the choice:
1
Enter the value :
89

Doubly Linked List :
54 -> 89 ->
```

Add at beginning



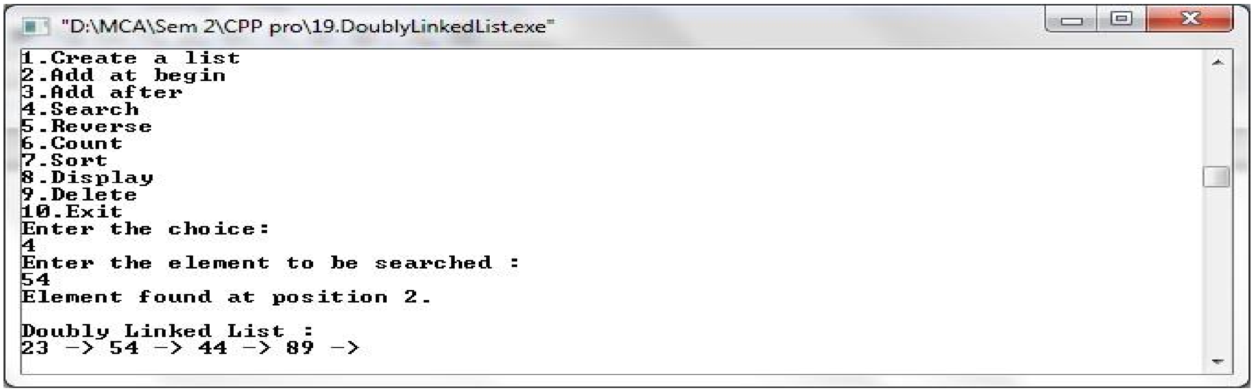
```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Reverse
6.Count
7.Sort
8.Display
9.Delete
10.Exit
Enter the choice:
2
Enter the value :
23

Doubly Linked List :
23 -> 54 -> 89 ->

1.Create a list
2.Add at begin
3.Add after
```

Add after



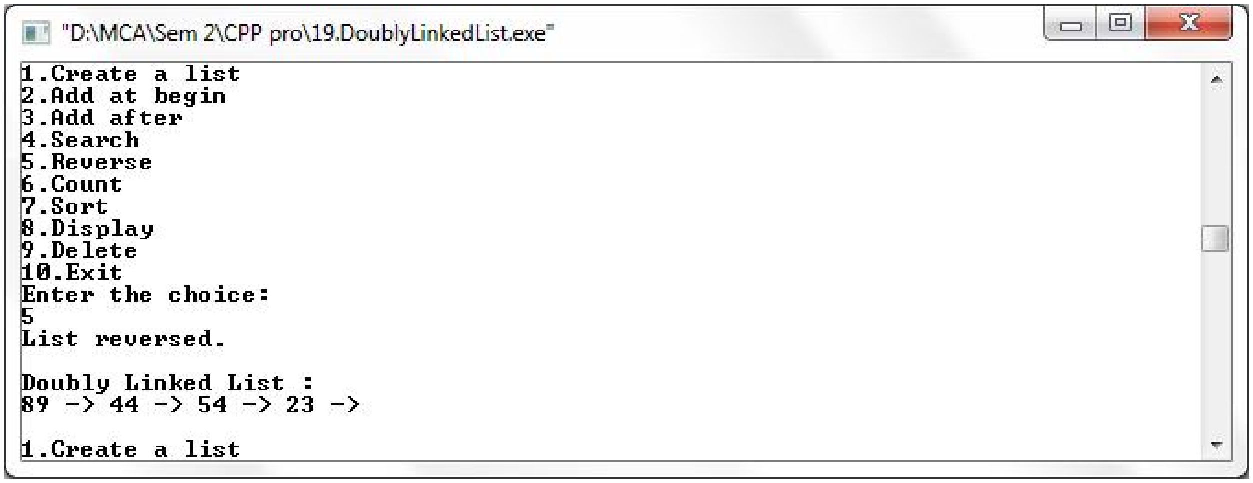
```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Reverse
6.Count
7.Sort
8.Display
9.Delete
10.Exit
Enter the choice:
3
Enter the position :
2
Enter the value :
44

Doubly Linked List :
23 -> 54 -> 44 -> 89 ->
```

Search



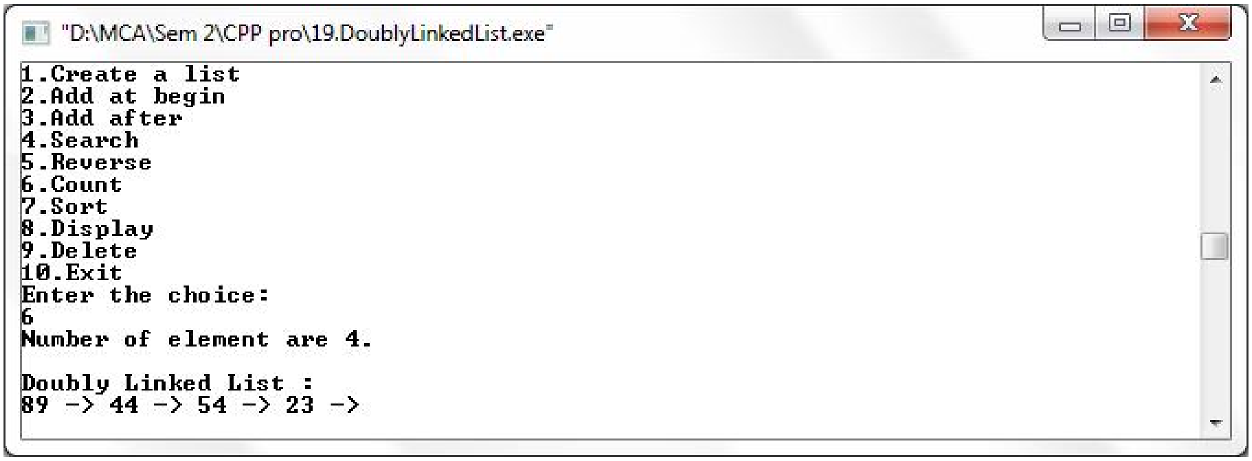
```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Reverse
6.Count
7.Sort
8.Display
9.Delete
10.Exit
Enter the choice:
4
Enter the element to be searched :
54
Element found at position 2.

Doubly Linked List :
23 -> 54 -> 44 -> 89 ->
```

Reverse



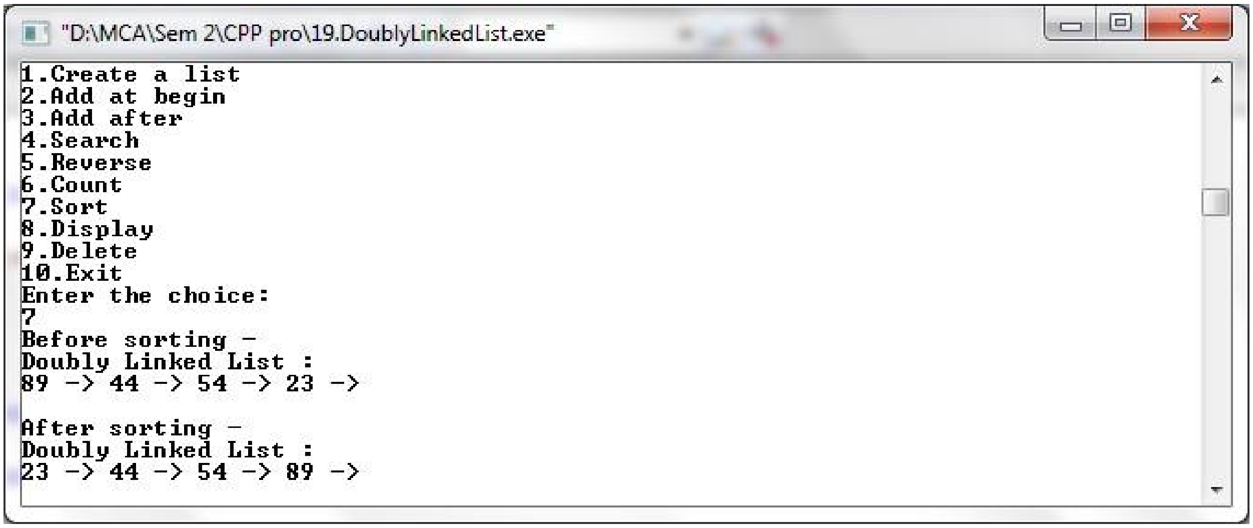
```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Reverse
6.Count
7.Sort
8.Display
9.Delete
10.Exit
Enter the choice:
5
List reversed.

Doubly Linked List :
89 -> 44 -> 54 -> 23 ->

1.Create a list
```

Count



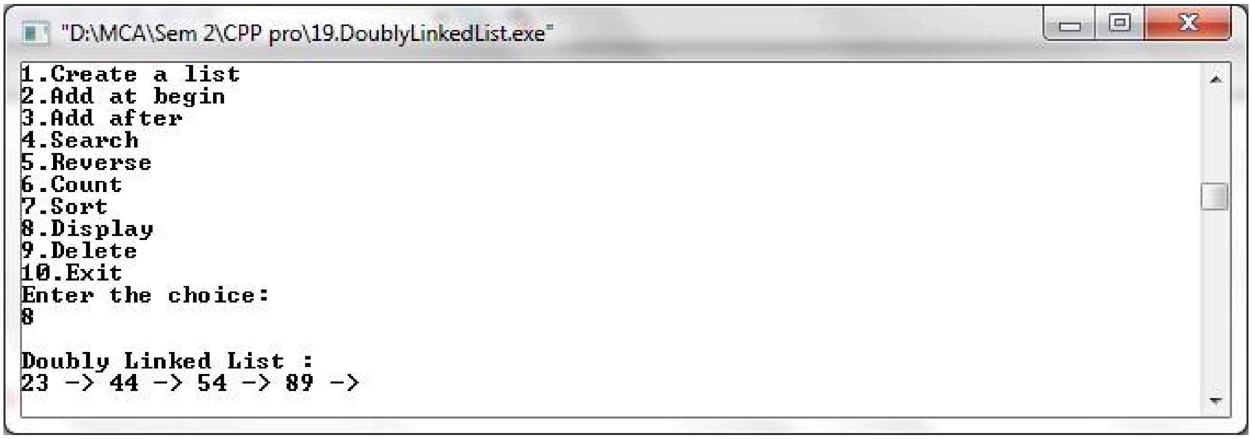
```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Reverse
6.Count
7.Sort
8.Display
9.Delete
10.Exit
Enter the choice:
6
Number of element are 4.

Doubly Linked List :
89 -> 44 -> 54 -> 23 ->
```

Sort



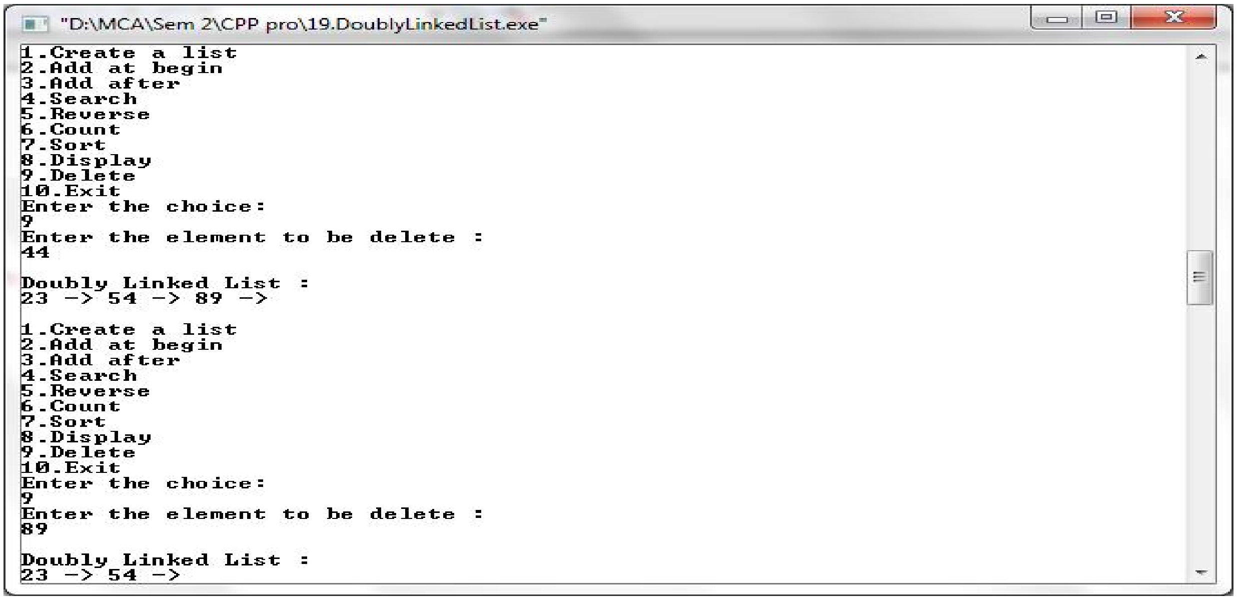
```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Reverse
6.Count
7.Sort
8.Display
9.Delete
10.Exit
Enter the choice:
7
Before sorting -
Doubly Linked List :
89 -> 44 -> 54 -> 23 ->

After sorting -
Doubly Linked List :
23 -> 44 -> 54 -> 89 ->
```

Display



```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Reverse
6.Count
7.Sort
8.Display
9.Delete
10.Exit
Enter the choice:
8

Doubly Linked List :
23 -> 44 -> 54 -> 89 ->
```

Delete



```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Reverse
6.Count
7.Sort
8.Display
9.Delete
10.Exit
Enter the choice:
9
Enter the element to be delete :
44

Doubly Linked List :
23 -> 54 -> 89 ->

1.Create a list
2.Add at begin
3.Add after
4.Search
5.Reverse
6.Count
7.Sort
8.Display
9.Delete
10.Exit
Enter the choice:
9
Enter the element to be delete :
89

Doubly Linked List :
23 -> 54 ->
```

PRACTICAL 12

Aim: Implementation of Singly Circular Linked Lists.

```
#include<iostream>
using namespace std;

class singly_circular
{
    public:
        int flag=true;
        int pos, i, value, count=0;
        struct node
        {
            int data;
            struct node *next;
            struct node *prev;
        };
        struct node *tmp=NULL;
        struct node *start=NULL;
        struct node *last=NULL;
        struct node *p=NULL;
        struct node *ptr=NULL;

        void create(int x)
        {
            tmp=new node;
            tmp->data=x;
            if(last==NULL)
            {
                last=tmp;
                tmp->next=last;
            }
            else
            {
                tmp->next=last->next;
                last->next=tmp;
                last=tmp;
            }
        }
        void add_atbegin(int x)
        {
            if(last==NULL)
            {
                cout<<"List is empty.\n";
            }
            tmp=new node;
            tmp->data=x;
            tmp->next=last->next;
            last->next=tmp;
        }

void add_after(int x,int pos)
        {
            if(last==NULL)
            {
                cout<<"List is empty.\n";
            }
            p=last->next;
            for(int i=0;i<pos-1;i++)
            {
                p=p->next;
                if(p==last->next)
                {
                    cout<<"Position does not exist.\n";
                    //break;
                }
            }
            tmp=new node;
            tmp->next=p->next;
            tmp->data=x;
            p->next=tmp;
            if(p==last)
            {
                last=tmp;
            }
        }
        void del(int x)
        {
            //p=last->next;
            if(last->next==last && last->data==x) // for only one node
            {
                tmp=last;
                last=NULL;
                delete(tmp);
                return;
            }
            p=last->next;
            if(p->data==x) //first element deleted
            {
```

```
        tmp=p;
        last->next=p->next;
        delete(tmp); return;
    }
    while(p->next!=last)
    {
        if(p->next->data==x)
        {
            tmp=p->next;
            p->next=tmp->next;
            delete(tmp);
            //cout<<"Deleted item "<<x;
            return;
        }//delete element in between
        p=p->next;
    }
    if(p->next->data==x)
    {
        tmp=p->next;
        p->next=last->next;
        delete(tmp); last=p;

        return;
    }//last element deleted
    cout<<"Element not found.\n";
}
void search1(int x)
{
    int pos=1;
    while(p->next!=last)
    {
        if(p->data==x)
        {
            cout<<"Element found at position "<<pos-1<<".\n";
        }
        p=p->next;
        pos++;
    }
    if(p==NULL)
        cout<<"Item not found.\n";
}

void sort()
```

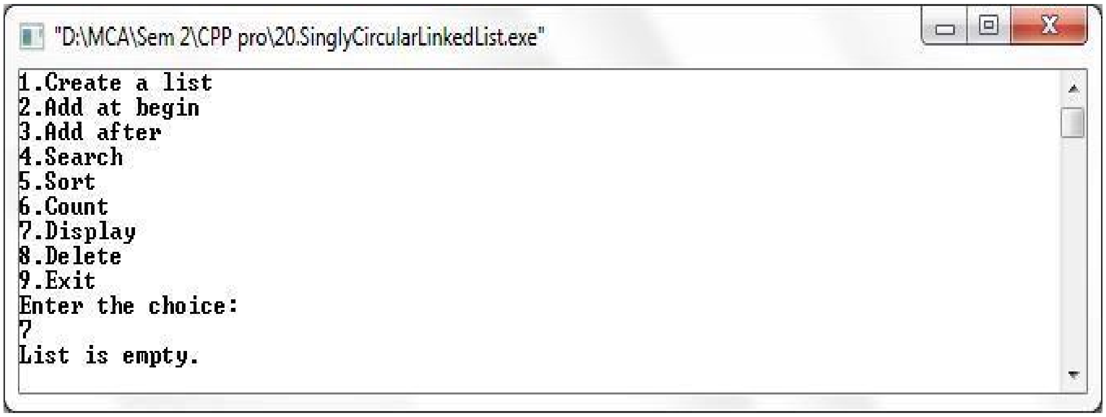
```
{
    int x;
    if(last==NULL)
    {
        cout<<"List is empty.\n\n";
    }
    p=last->next;

    while(p!=last)
    {
        ptr=p->next;
        while(ptr!=last->next)
        {
            if(ptr!=last->next)
            {
                if(p->data>ptr->data)
                {
                    x=p->data;
                    p->data=ptr->data;
                    ptr->data=x;
                }
            }
            ptr=ptr->next;
        }
        p=p->next;
    }
}
int count1()
{
    if(last==NULL)
    {
        cout<<"List is empty.\n\n";
    }
    else
    {
        p=last->next;

        while(p!=last)
        {
            count++;

            p=p->next;
        }
        count++;
        cout<<"Number of element are "<<count<<"\n";
    }
```

```
        }

        void display()
        {
            if(last==NULL)
            {
                cout<<"List is empty.\n\n";
                return;
            }
            p=last->next;
            cout<<"\nSingly Circular Linked List :\n";
            while(p!=last)
            {

                cout<<p->data<<" -> ";
                p=p->next;
            }
            cout<<last->data<<"\n\n";
        }

};
int main()
{
    singly_circular d;
    int x,ch;
    int pos;

    while(ch!=9)
    {
        cout<<"1.Create a list\n2.Add at begin\n3.Add after\n4.Search\n";
        cout<<"5.Sort\n6.Count\n7.Display\n8.Delete\n9.Exit\n";
        cout<<"Enter the choice:\n";
        cin>>ch;

        switch(ch)
        {
        case 1:
            cout<<"Enter the value :\n";
            cin>>x;
            d.create(x);
            d.display();
            break;

        case 2:
            cout<<"Enter the value :\n";
```

```
                cin>>x;
                d.add_atbegin(x);
                d.display();
                break;

        case 3:
                cout<<"Enter the position :\n";
                cin>>pos;
                cout<<"Enter the value :\n";
                cin>>x;
                d.add_after(x,pos);
                d.display();

                break;

        case 4:
                cout<<"Enter element to be
                searched:\n"; cin>>x;
```

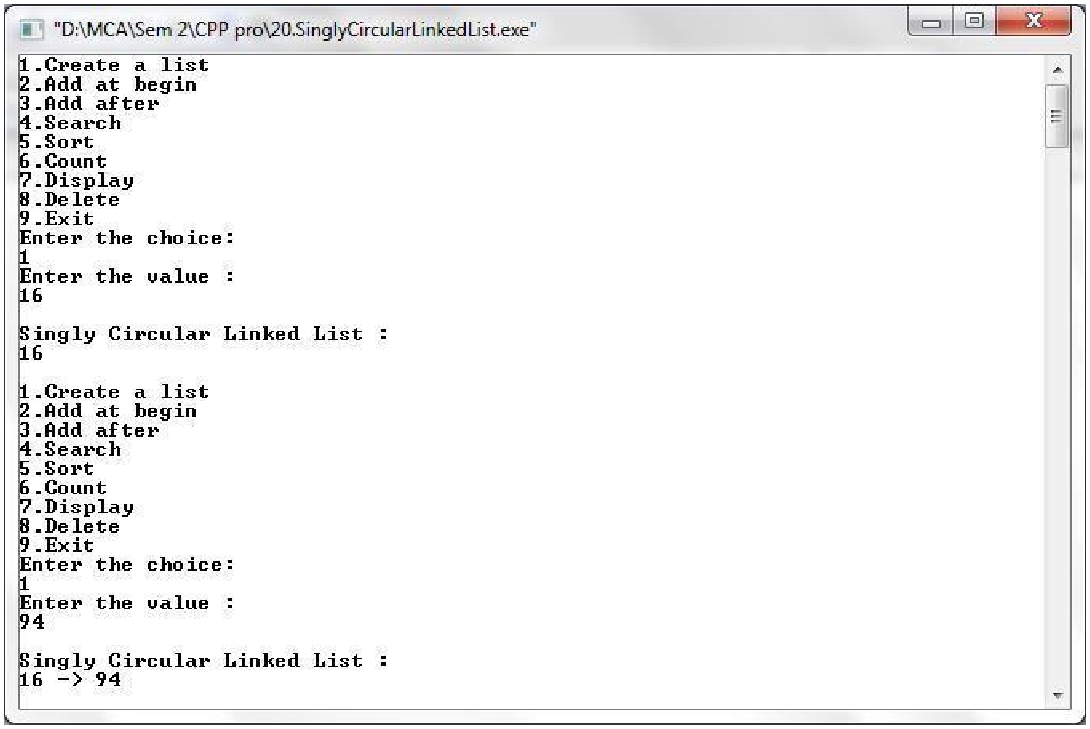
Output :

List is Empty



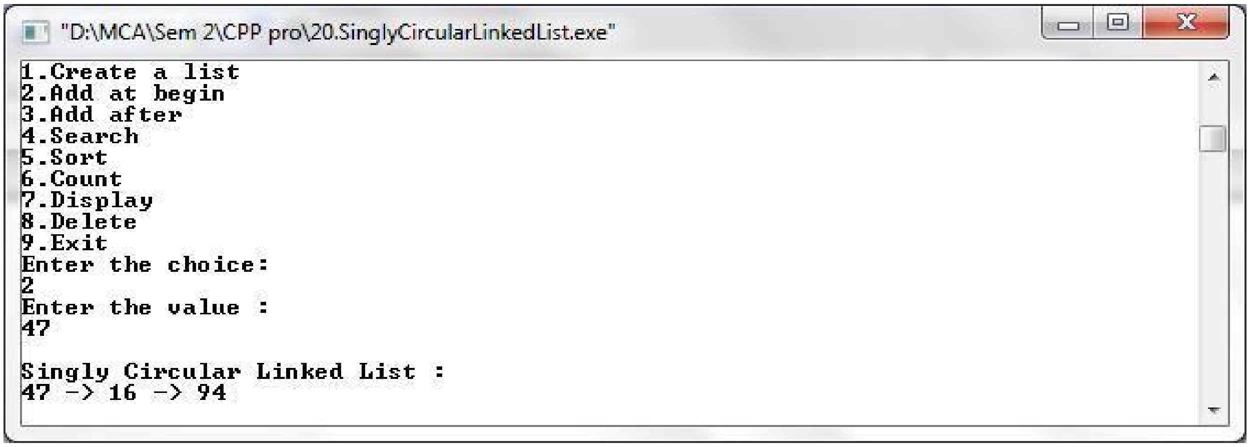
```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Sort
6.Count
7.Display
8.Delete
9.Exit
Enter the choice:
7
List is empty.
```

Create a list



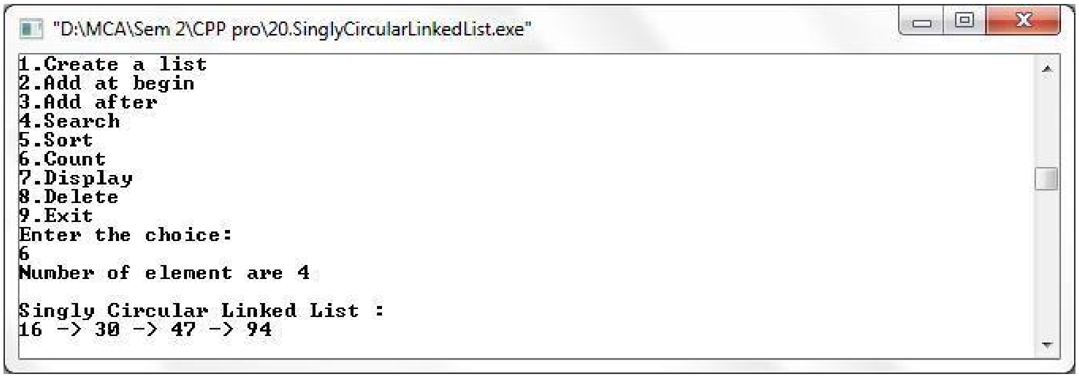
```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Sort
6.Count
7.Display
8.Delete
9.Exit
Enter the choice:
1
Enter the value :
16

Singly Circular Linked List :
16

1.Create a list
2.Add at begin
3.Add after
4.Search
5.Sort
6.Count
7.Display
8.Delete
9.Exit
Enter the choice:
1
Enter the value :
94

Singly Circular Linked List :
16 -> 94
```

Add at beginning



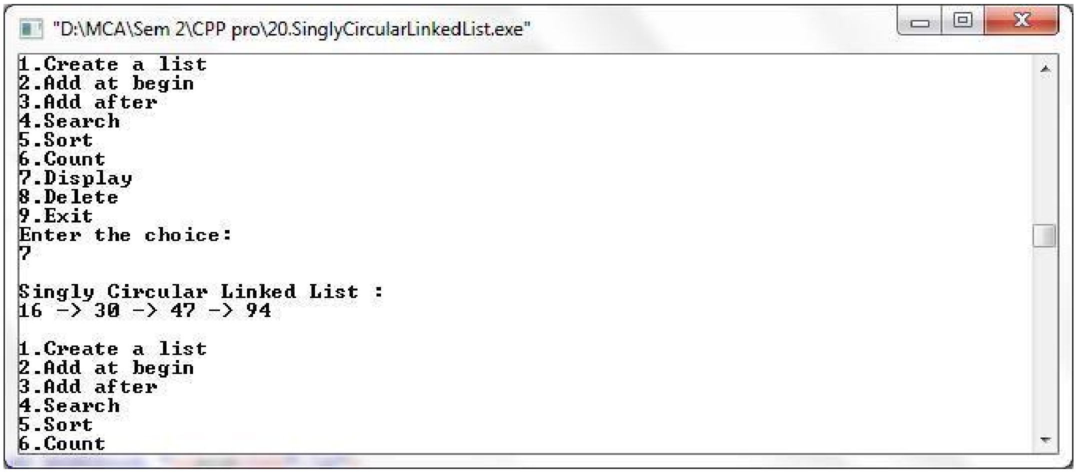
```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Sort
6.Count
7.Display
8.Delete
9.Exit
Enter the choice:
2
Enter the value :
47

Singly Circular Linked List :
47 -> 16 -> 94
```

Count



```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Sort
6.Count
7.Display
8.Delete
9.Exit
Enter the choice:
6
Number of element are 4

Singly Circular Linked List :
16 -> 30 -> 47 -> 94
```

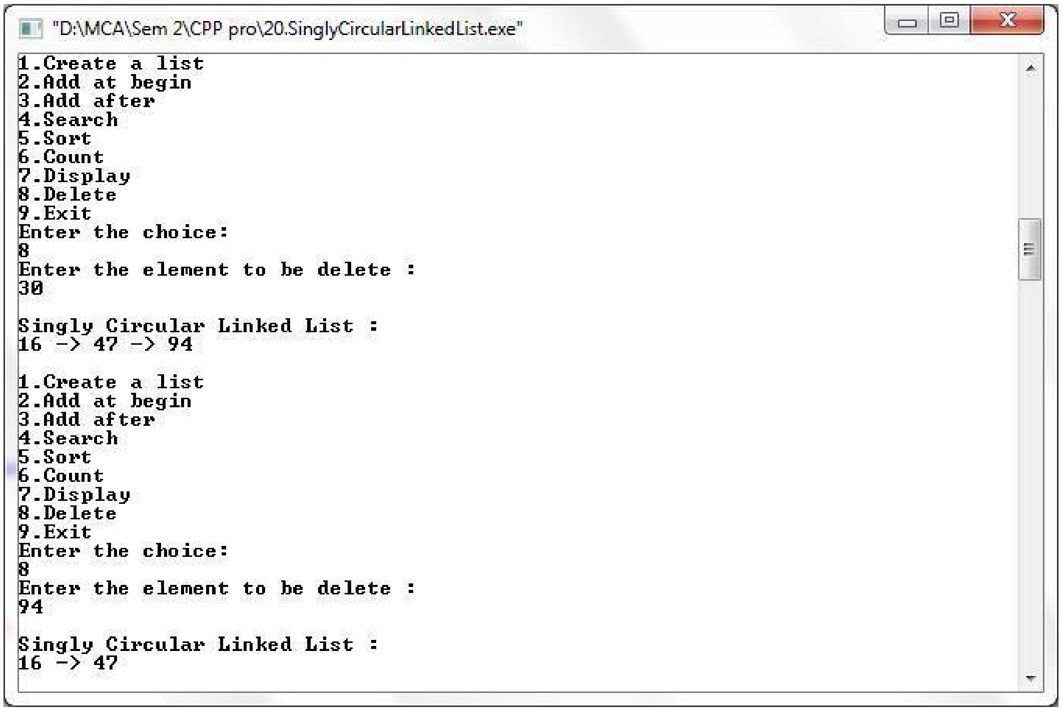
Display



"D:\MCA\Sem 2\CPP pro\20.SinglyCircularLinkedList.exe"

```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Sort
6.Count
7.Display
8.Delete
9.Exit
Enter the choice:
7

Singly Circular Linked List :
16 -> 30 -> 47 -> 94

1.Create a list
2.Add at begin
3.Add after
4.Search
5.Sort
6.Count
```

Delete



"D:\MCA\Sem 2\CPP pro\20.SinglyCircularLinkedList.exe"

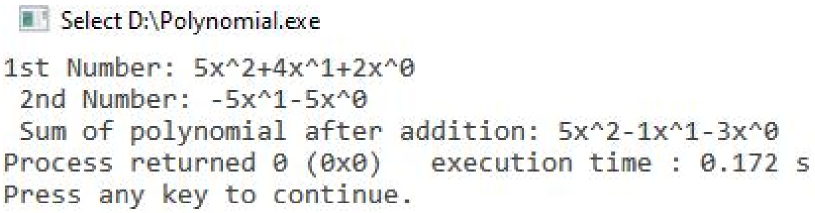
```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Sort
6.Count
7.Display
8.Delete
9.Exit
Enter the choice:
8
Enter the element to be delete :
30

Singly Circular Linked List :
16 -> 47 -> 94

1.Create a list
2.Add at begin
3.Add after
4.Search
5.Sort
6.Count
7.Display
8.Delete
9.Exit
Enter the choice:
8
Enter the element to be delete :
94

Singly Circular Linked List :
16 -> 47
```

PRACTICAL 13

Aim: Demonstrate Application of Linked List Like Polynomial.

```
     #include <bits/stdc++.h>
#include <iostream.h>
using namespace std;
struct Node
    {int coeff;
    int pow;
    struct Node* next;  };
void create_node(int x, int y, struct Node** temp)
{
    struct Node *r, *z;
    z = *temp;
    if (z == NULL) {
        r = (struct Node*)malloc(sizeof(struct Node));
        r->coeff = x;
        r->pow = y;
        *temp = r;
        r->next = (struct Node*)malloc(sizeof(struct Node));
        r = r->next;
        r->next = NULL;
    }
    else {
        r->coeff = x;
        r->pow = y;
        r->next = (struct Node*)malloc(sizeof(struct Node));
        r = r->next;
        r->next = NULL;
    }
}
void polyadd(struct Node* poly1, struct Node* poly2,
            struct Node* poly)
{
    while (poly1->next && poly2->next) {
        if (poly1->pow > poly2->pow)
            {poly->pow = poly1->pow;
            poly->coeff = poly1->coeff;
            poly1 = poly1->next;
        }
        else if (poly1->pow < poly2->pow)
            {poly->pow = poly2->pow;
            poly->coeff = poly2->coeff;
            poly2 = poly2->next;
        }
        else {
            poly->pow = poly1->pow;
            poly->coeff = poly1->coeff + poly2->coeff;
            poly1 = poly1->next;
            poly2 = poly2->next;
        }
                poly->next
            = (struct Node*)malloc(sizeof(struct Node));
        poly = poly->next;
        poly->next = NULL;
    }
    while (poly1->next || poly2->next) {
        if (poly1->next) {
            poly->pow = poly1->pow;
            poly->coeff = poly1->coeff;
            poly1 = poly1->next;
        }
        if (poly2->next) {
            poly->pow = poly2->pow;
            poly->coeff = poly2->coeff;
            poly2 = poly2->next;
        }
        poly->next
            = (struct Node*)malloc(sizeof(struct Node));
        poly = poly->next;
        poly->next = NULL;
    }
}
void show(struct Node* node)
{
    while (node->next != NULL)
        { printf("%dx^%d", node->coeff, node-
        >pow);node = node->next;
        if (node->coeff >= 0) {
            if (node->next != NULL)
                printf("+");
        }
    }
}
int main()
{
    struct Node *poly1 = NULL, *poly2 = NULL, *poly = NULL;
    create_node(5, 2, &poly1);
    create_node(4, 1, &poly1);
    create_node(2, 0, &poly1);
    create_node(-5, 1, &poly2);
    create_node(-5, 0, &poly2);
    printf("1st Number: ");
    show(poly1);
    printf("\n 2nd Number: ");
    show(poly2);
    poly = (struct Node*)malloc(sizeof(struct Node));
    polyadd(poly1, poly2, poly);
    printf("\n Sum of polynomial after addition: ");
    show(poly);
    return 0;
}
```

Output :



```
1st Number: 5x^2+4x^1+2x^0
 2nd Number: -5x^1-5x^0
 Sum of polynomial after addition: 5x^2-1x^1-3x^0
Process returned 0 (0x0)   execution time : 0.172 s
Press any key to continue.
```

PRACTICAL 14

Aim: Write a C++ Program to Implement Binary Search Tree.

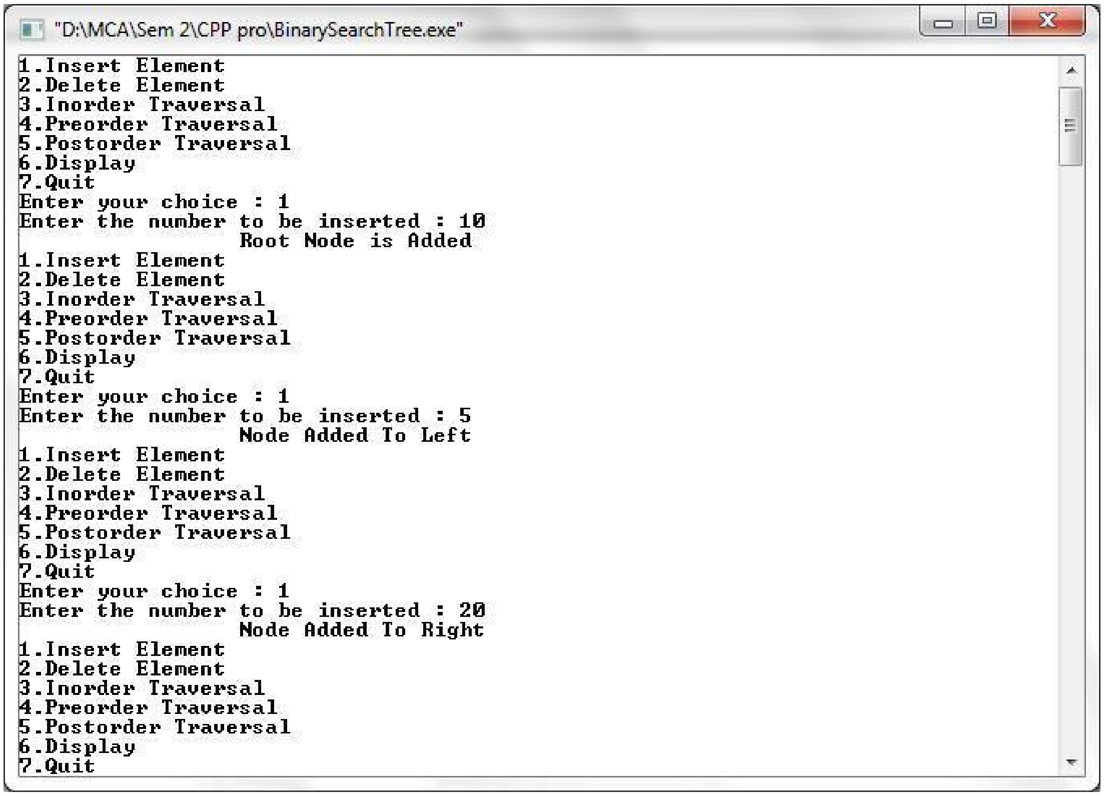
```
# include <iostream>
# include
<cstdlib>
using
namespace std;
/*Node
Declaration*/
struct node
{
    int data;
    struct node
    *left; struct
    node *right;
}*root;
/*Class
Declaration*/
class BST
{
    public:
        void find1(int, node **, node
        **); void insert1(node *, node
        *); void del(int);
        void case_a(node *,node
        *); void case_b(node
        *,node *); void
        case_c(node *,node *);
        void preorder(node *);
        void inorder(node *);
        void postorder(node *);
        void display(node *,
        int); BST()

        {
            root = NULL;
        }
};
/*Main Contains
Menu*/ int main()
{
  int choice,
  num; BST
  bst;
  node
  *temp;
  while
  (1)
  {

    cout<<"\n1.Insert Element \n";
    cout<<"2.Delete Element \n";
    cout<<"3.Inorder Traversal \n";
    cout<<"4.Preorder Traversal \n";
    cout<<"5.Postorder Traversal \n";
    cout<<"6.Display \n";
    cout<<"7.Quit \n";
cout<<"Enter your choice : ";
    cin>>choice;
    switch(choice)
    {
    case 1:
      temp = new node;
      cout<<"Enter the number to be inserted
    : "; cin>>temp->data;
      bst.insert1(root, temp);
      break;
    case 2:
      if (root == NULL)
      {
        cout<<"Tree is empty, nothing to delete
        \n"; continue;
      }
      cout<<"Enter the number to be deleted
      : "; cin>>num;
      bst.del(num);
      break;
    case 3:
      cout<<"Inorder Traversal:";
      bst.inorder(root);

      break;
  case 4:
      cout<<"Preorder Traversal:";
      bst.preorder(root);

      break;
```

```
        case 5:
            cout<<"Postorder Traversal:";
            bst.postorder(root);

            break;
        case 6:
            cout<<"Display BST:";
            bst.display(root,1);

            break;
        case 7:
            exit(1);
        default:
            cout<<"Wrong choice \n";
        }
    }
}
void BST::find1(int x, node **par, node **loc)
{
    node *ptr, *ptrsave;
    if (root == NULL)
    {
        *loc = NULL;
        *par = NULL;
        return;
    }

    if (x == root->data)
    {
        *loc = root;
        *par = NULL;
        return;
    }
    if (x < root->data)
        ptr = root->left;
    else
        ptr = root->right;
    ptrsave = root;
    while (ptr != NULL)
    {
        if (x == ptr->data)
        {
            *loc = ptr;
            *par = ptrsave;
            return;
        }
        ptrsave = ptr;
        if (x < ptr->data)
            ptr = ptr->left;
    else
        ptr = ptr->right;
    }
    *loc = NULL;
    *par = ptrsave;
}
/*Inserting Element into the Tree*/
void BST::insert1(node *tree, node *newnode)
{
    if (root == NULL)
    {
        root = new node;
        root->data = newnode->data;

        root->left = NULL;
        root->right = NULL;
        cout<<" \t\t Root Node is Added";
        return;
    }
    if (tree->data == newnode->data)
    {
        cout<<"Element already in the tree \n";
        return;
    }
    if (tree->data > newnode->data)
    {
        if (tree->left != NULL)
        {
            insert1(tree->left, newnode);
    }
    else
    {
            tree->left = newnode;
            (tree->left)->left = NULL;
            (tree->left)->right = NULL;

            cout<<" \t\t Node Added To Left";
            return;
        }
    }
    else
    {
        if (tree->right != NULL)
        {
            insert1(tree->right, newnode);
        }
        else
        {
            tree->right = newnode;
            (tree->right)->left = NULL;
            (tree->right)->right = NULL;
            cout<<"\t\t Node Added To Right";
            return;
        }
    }
}
/*Delete Element from the tree*/
void BST::del(int x)
{
    node *parent, *location;
    if (root == NULL)
    {
        cout<<"Tree empty";
        return;
    }
    find1(x, &parent, &location);
    if (location == NULL)
    {
        cout<<"x not present in tree";return;
    }
    if (location->left == NULL && location->right == NULL)
        case_a(parent, location);
    if (location->left != NULL && location->right == NULL)
        case_b(parent, location);
    if (location->left == NULL && location->right != NULL)
        case_b(parent, location);
    if (location->left != NULL && location->right != NULL)
        case_c(parent, location);
    free(location);
}
/*Case A*/
void BST::case_a(node *par, node *loc )
{
    if (par == NULL)
    {
        root = NULL;
    }
    else
    {
     if (loc == par->left)
        par->left  = NULL;
        else

            par->right = NULL;
    }
}
/*Case B*/
void BST::case_b(node *par, node *loc)
{
    node *child;
 if (loc->left != NULL)
    child  = loc->left;
    else
        child = loc->right;
    if (par == NULL)
    {
        root = child;
    }
    else
    {
        if (loc == par->left)
            par->left = child;
        else
            par->right = child;
    }
}
/*Case C*/
void BST::case_c(node *par, node *loc)
{
    node *ptr, *ptrsave, *leaf1, *parleaf1;
    ptrsave = loc;
    ptr = loc->right;
    while (ptr->left != NULL)
    {
        ptrsave = ptr;
        ptr = ptr->left;
    }
    leaf1 = ptr;
    parleaf1 = ptrsave;
    if (leaf1->left == NULL && leaf1->right == NULL)
        case_a(parleaf1, leaf1);
    else
```

```
        case_b(parleaf1, leaf1);
    if (par == NULL)
    {
        root = leaf1;
    }
    else
    {
        if (loc == par->left)
            par->left = leaf1;
        else
            par->right = leaf1;
    }
    leaf1->left = loc->left;
    leaf1->right = loc->right;
}
/*Pre Order Traversal*/
void BST::preorder(node *ptr)
{

    if (root == NULL)
    {
        cout<<"Tree is empty";
        return;
    }
    if (ptr != NULL)
    {
        cout<<ptr->data<<"  ";
        preorder(ptr->left);
        preorder(ptr->right);
    }
}

/*In Order Traversal*/
void BST::inorder(node *ptr)
{
    if (root == NULL)
    {
        cout<<"Tree is empty";
        return;
    }
    if (ptr != NULL)
    {
        inorder(ptr->left);
        cout<<ptr->data<<"  ";
        inorder(ptr->right);
    }
}
/*Postorder Traversal*/
void BST::postorder(node *ptr)
{
    if (root == NULL)
    {
        cout<<"Tree is empty";
        return;
    }
    if (ptr != NULL)
    {
        postorder(ptr->left);
        postorder(ptr->right);
        cout<<ptr->data<<"  ";
    }
}
void BST::display(node *ptr, int level)
{
    int i;
    if (ptr != NULL)
    {
        display(ptr->right, level+1);
        cout<<endl;
        if (ptr == root)
            cout<<"Root->:  ";
        else
        {
            for (i = 0;i < level;i++)
                cout<<"       ";
    }
        cout<<ptr->data;

        display(ptr->left, level+1);
    }
}
```

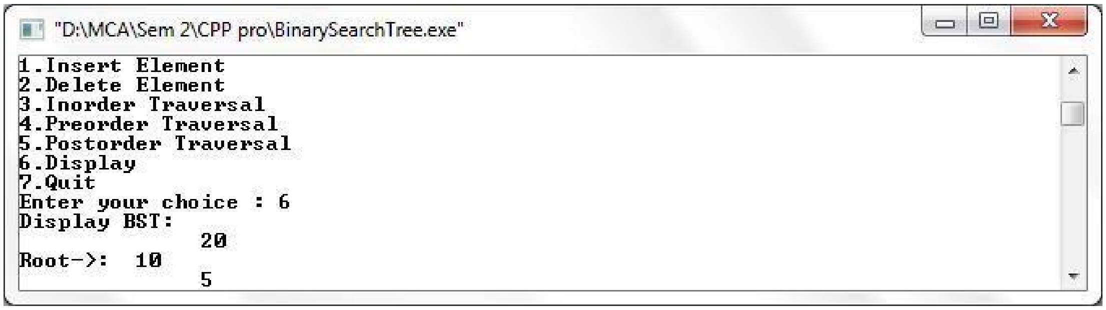
Output :

Insert



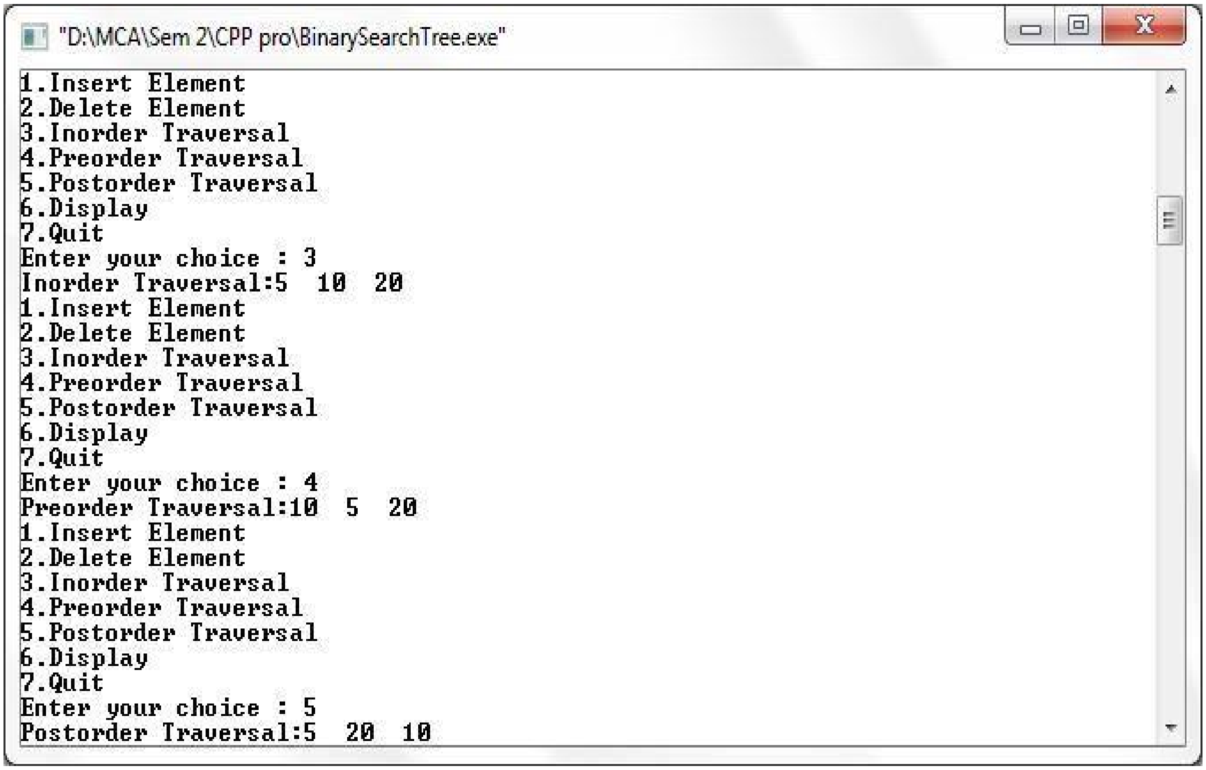
```
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 1
Enter the number to be inserted : 10
                 Root Node is Added
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 1
Enter the number to be inserted : 5
                 Node Added To Left
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 1
Enter the number to be inserted : 20
                 Node Added To Right
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
```

Display



```
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 6
Display BST:
              20
Root->:  10
              5
```

Inorder Traversal, Preorder traversal & Postorder Traversal:



```
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 3
Inorder Traversal:5  10  20
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 4
Preorder Traversal:10  5  20
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 5
Postorder Traversal:5  20  10
```

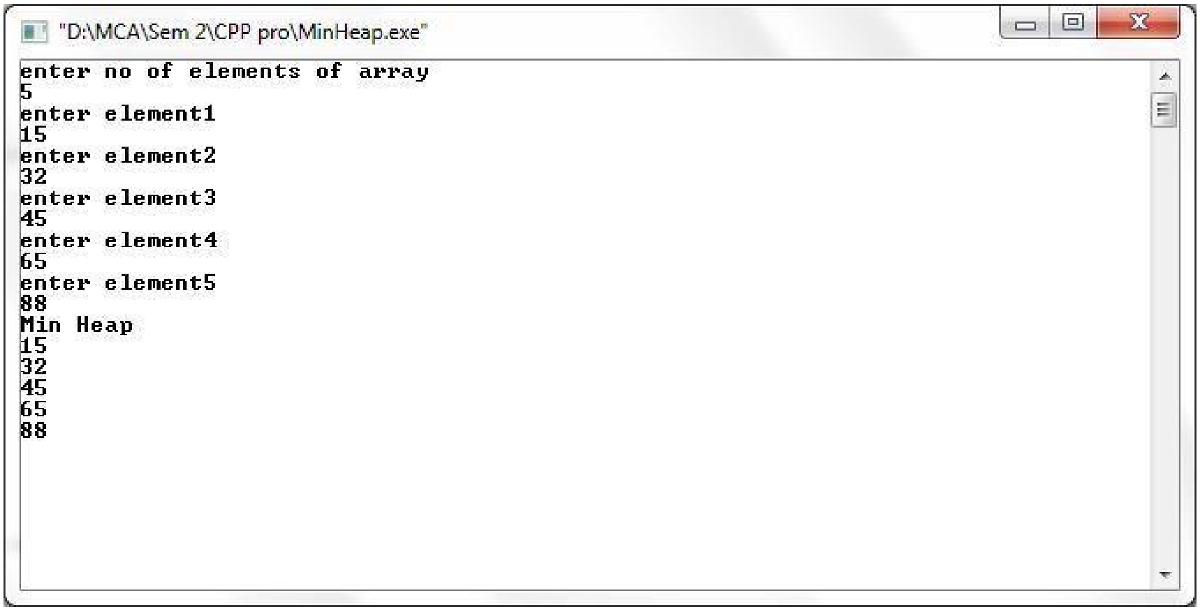
PRACTICAL 15

Aim: Implementation of Min and Max Heap.

A] Min Heap :

```
#include <iostream>
#include <conio.h>
    using namespace std;
    void min_heapify(int *a,int i,int n)
    {
        int j, temp;
        temp = a[i];
        j = 2 * i;
        while (j <= n)
        {
            if (j < n && a[j+1] < a[j])
                j = j + 1;
            if (temp < a[j])
                break;
            else if (temp >= a[j])
            {
                a[j/2] = a[j];
                j = 2 * j;
            }
        }
        a[j/2] = temp;
        return;
    }
    void build_minheap(int *a, int n)
    {
        int i;
        for(i = n/2; i >= 1; i--)
        {
            min_heapify(a,i,n);
        }
    }
    int main()
    {
        int n, i, x;
        cout<<"enter no of elements of
        array\n"; cin>>n;
        int a[20];
        for (i = 1; i <= n; i++)
        {
            cout<<"enter element"<<(i)<<endl;
            cin>>a[i];
        }
        build_minheap(a, n);
        cout<<"Min Heap\n";
        for (i = 1; i <= n; i++)
        {
            cout<<a[i]<<endl;

}
    }
```

Output :

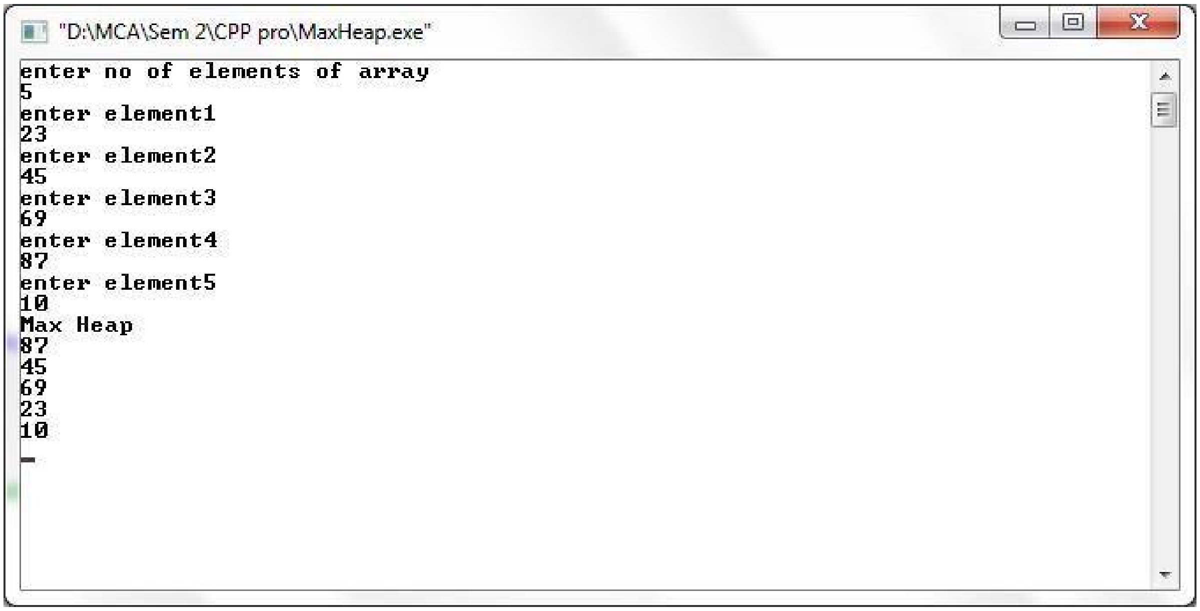


```
enter no of elements of array
5
enter element1
15
enter element2
32
enter element3
45
enter element4
65
enter element5
88
Min Heap
15
32
45
65
88
```

B] Max Heap :

```
#include <iostream>
#include <conio.h>
using namespace std;
void max_heapify(int *a, int i, int n)
    {
        int j, temp;
        temp = a[i];
        j = 2 * i;
        while (j <= n)
        {
            if (j < n && a[j+1] > a[j])
            j = j + 1;
            if (temp > a[j])
                break;
            else if (temp <= a[j])
            {
                a[j / 2] = a[j];
                j = 2 * j;
            }
        }
        a[j/2] = temp;
        return;
    }
    void build_maxheap(int *a,int n)
    {
        int i;
        for(i = n/2; i >= 1; i--)
        {
            max_heapify(a,i,n);
        }
    }
    int main()
    {
        int n, i, x;
        cout<<"enter no of elements of
        array\n"; cin>>n;
        int a[20];
        for (i = 1; i <= n; i++)
        {
            cout<<"enter element"<<(i)<<endl;
            cin>>a[i];
        }
        build_maxheap(a,n);
        cout<<"Max Heap\n";
        for (i = 1; i <= n; i++)
        {
            cout<<a[i]<<endl;

}
    }
```

Output :



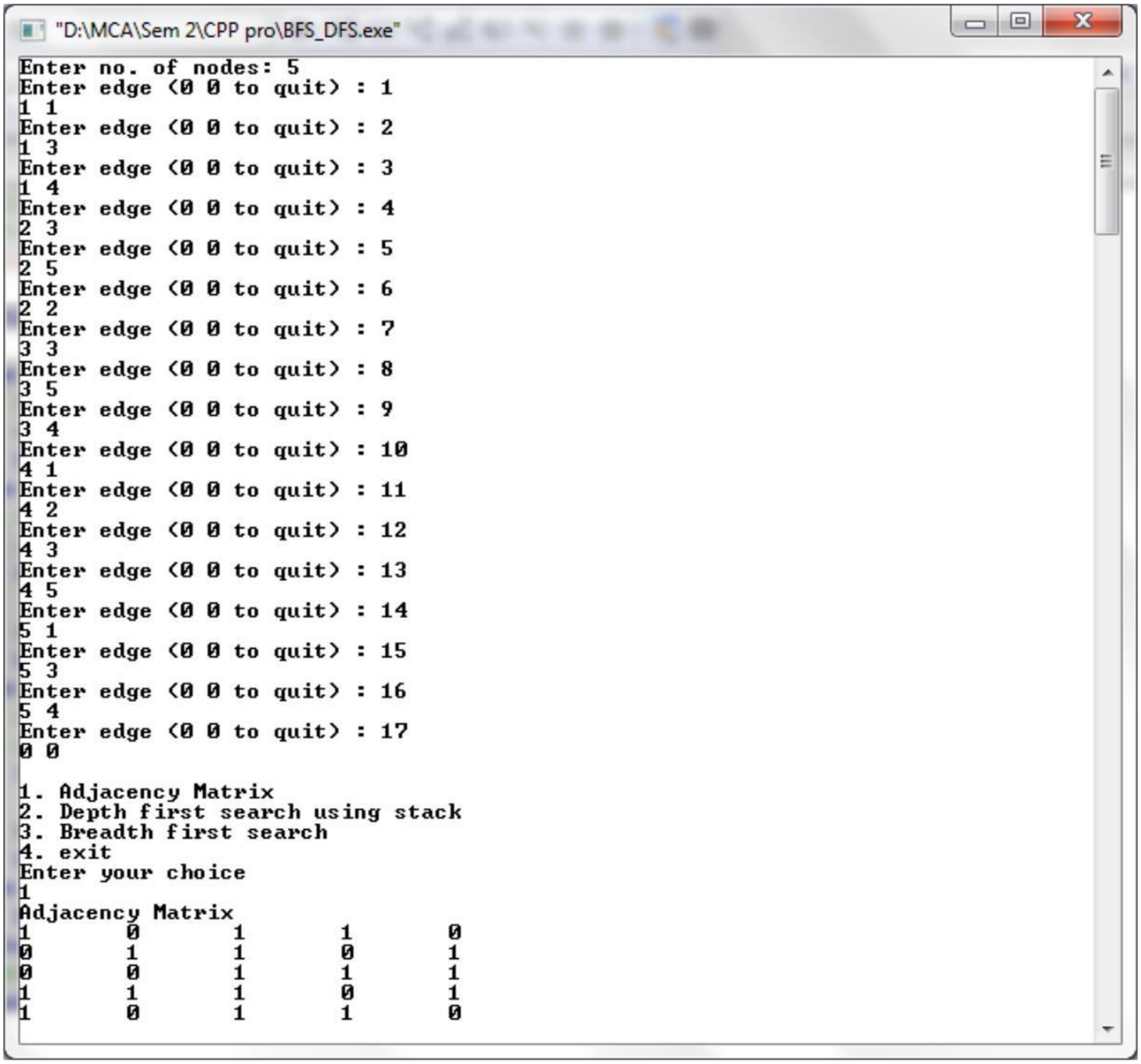
```
enter no of elements of array
5
enter element1
23
enter element2
45
enter element3
69
enter element4
87
enter element5
10
Max Heap
87
45
69
23
10
_
```

PRACTICAL 16

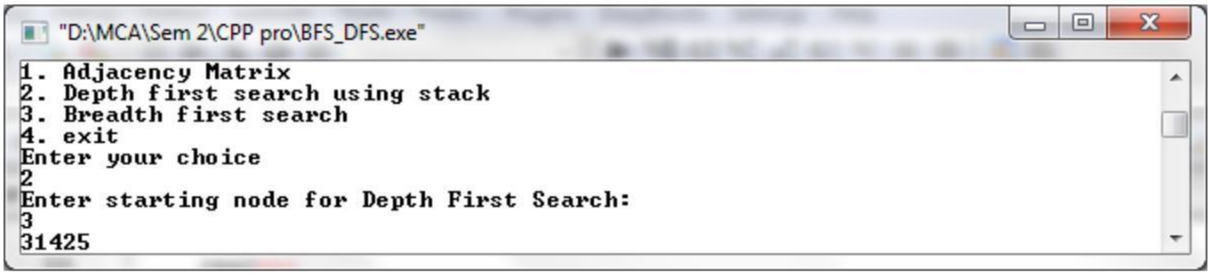
Aim: Implementation of Graph Traversal (DFS and BFS).

```
#include<iostream>
#include<stdio.h>
#define  max  20
using namespace std;
int adj[max][max];
bool visited[max];
int n;
int frnt;

void create_graph()
{
    int i, max_edges,origin,destin;

    cout<<"Enter no. of nodes: ";
    cin>>n;
    max_edges=n*(n-1);

    for(i=1;i<=max_edges;i++)
    {
        cout<<"Enter edge (0 0 to quit) :
        "<<i<<"\n"; cin>>origin>>destin;

        if(origin==0||destin==0)
            break;

        if(origin>n||destin>n||origin<=0||destin<=0)
        {
            cout<<"Invalid edge \n";
            i--;
        }
        else
        {
            adj[origin][destin]=1;
        }

    }
}
void display()
{
    int i, j;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            cout<<adj[i][j]<<"\t";
        }
        cout<<"\n";
    }
}

void dfs(int v)

{
    int i, stack[max], top=-1,pop_v, j, t;
    int c;

    top++;
    stack[top]=v;
    while(top>=0)
    {
        pop_v=stack[top];
        top--;
        if(visited[pop_v]==false)
        {
            cout<<pop_v;
            visited[pop_v]=true;
        }
        else
            continue;

        for(i=n;i>=1;i--)
        {
            if(adj[pop_v][i]==1 && visited[i]==false)
            {
                top++;
                stack[top]=i;
            }
        }
    }
}
void bfs(int v)
{
    int i, frnt, rear;
    int que[20];
    frnt=rear=-1;
    cout<<v;
    visited[v]=true;
    rear++;
    frnt++;
    que[rear]=v;
    while(frnt<=rear)
    {
        v=que[frnt];

        frnt++;
        for(i=1;i<=n;i++)
        {
            if(adj[v][i]==1&&visited[i]==false)
            {
                cout<<i<<"\t";
                visited[i]=true;
                rear++;
                que[rear]=i;
            }
        }
    }
}
void adj_nodes(int v)
{
    int i;

    for(i=1;i<=n;i++)
    {
        int i;
        for(i=1;i<=n;i++)
        {
            if(adj[v][i]==1)
            cout<<i;
            cout<<"\n";
        }
    }
}
int main()
{
    int i, v, ch;
    create_graph();
    while(1)
    {
        cout<<"\n";
        cout<<"1. Adjacency Matrix \n";
        cout<<"2. Depth first search using
        stack\n"; cout<<"3. Breadth first
        search\n"; cout<<"4. exit \n";
        cout<<"Enter your choice\n";
        cin>>ch;

        switch(ch)
        {
        case 1:
            cout<<"Adjacency Matrix \n";
            display();
            break;
        case 2:
            cout<<"Enter starting node for Depth First Search:
            \n"; cin>>v;
            for(i=1;i<=n;i++)
                visited[i]=false;
            dfs(v);
            break;
        case 3:
            cout<<"Enter starting node for Breadth First Search:
            \n"; cin>>v;
            for(i=1;i<=n;i++)
                visited[i]=false;
            bfs(v);
            break;
        case 4:
            break;
        default:
        cout<<"Wrong Choice";
        break;
        }
    }
    return 0;
}
```

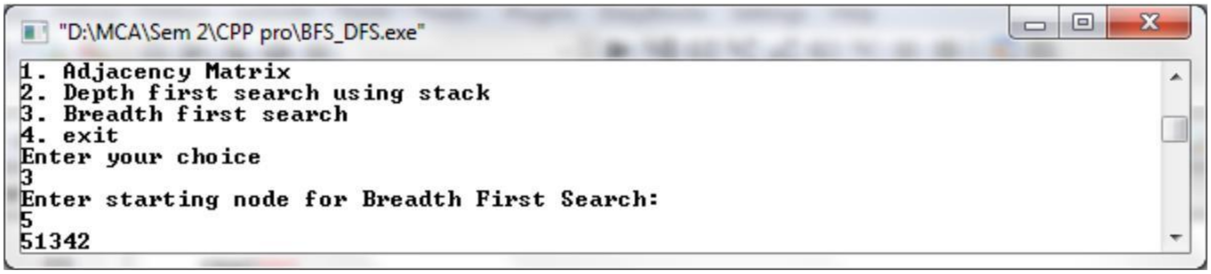
Ouput :



```
Enter no. of nodes: 5
Enter edge (0 0 to quit) : 1
1 1
Enter edge (0 0 to quit) : 2
1 3
Enter edge (0 0 to quit) : 3
1 4
Enter edge (0 0 to quit) : 4
2 3
Enter edge (0 0 to quit) : 5
2 5
Enter edge (0 0 to quit) : 6
2 2
Enter edge (0 0 to quit) : 7
3 3
Enter edge (0 0 to quit) : 8
3 5
Enter edge (0 0 to quit) : 9
3 4
Enter edge (0 0 to quit) : 10
4 1
Enter edge (0 0 to quit) : 11
4 2
Enter edge (0 0 to quit) : 12
4 3
Enter edge (0 0 to quit) : 13
4 5
Enter edge (0 0 to quit) : 14
5 1
Enter edge (0 0 to quit) : 15
5 3
Enter edge (0 0 to quit) : 16
5 4
Enter edge (0 0 to quit) : 17
0 0

1. Adjacency Matrix
2. Depth first search using stack
3. Breadth first search
4. exit
Enter your choice
1
Adjacency Matrix
1       0       1       1       0
0       1       1       0       1
0       0       1       1       1
1       1       1       0       1
1       0       1       1       0
```



```
1. Adjacency Matrix
2. Depth first search using stack
3. Breadth first search
4. exit
Enter your choice
2
Enter starting node for Depth First Search:
3
31425
```



```
1. Adjacency Matrix
2. Depth first search using stack
3. Breadth first search
4. exit
Enter your choice
3
Enter starting node for Breadth First Search:
5
51342
```

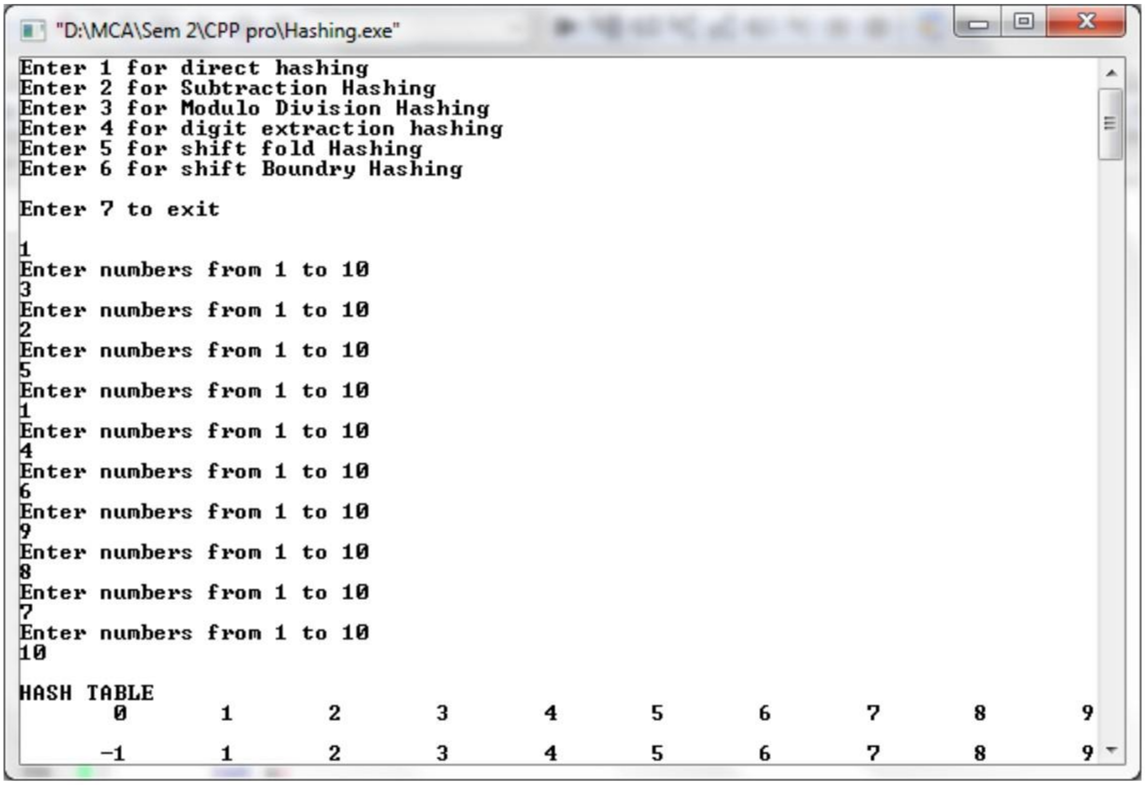
PRACTICAL 17

Aim: Perform Various Hashing Techniques with Linear Probe as Collision Resolution Scheme.

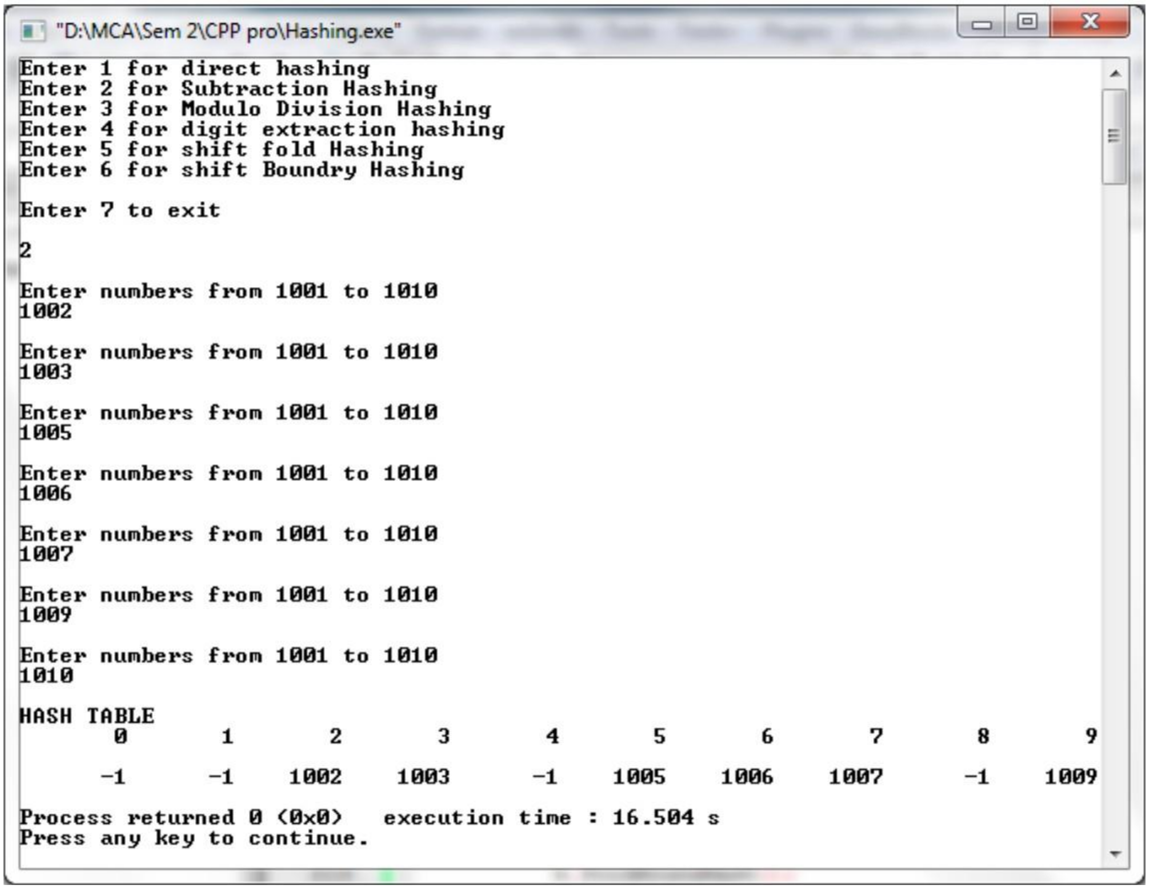
```
#include<iostream>
#include<conio.h>
#include<stdio.h>
#include<iomanip>
using namespace std;
const int SIZE=10;
static int coll;
class hash1
{
     long key;
     long index;
     long arr[10];
     public:
     void directHash();
     void subHash();
     void modDivision();
     void linProbe();
     void digitExHash();
     void foldShiftHash();
     void foldBoundHash();
     void display();
};
void hash1::modDivision()
{
   for(int i=0;i<10;i++)
arr[i]=-1;
for(int i=1;i<=7;i++)
{
    int x;
cout<<"\nEnter a number";
cin>>x;
index=x%10;
while(arr[index]!=-1)
index=(index+1)%10;
arr[index]=x;
}
}
void hash1::display()
{
    cout<<"\nHASH TABLE\n";
for(int i=0;i<10;i++)
```

```
cout<<setw(8)<<i;
cout<<"\n";
    for(int i=0;i<10;i++)

cout<<setw(8)<<arr [i];
}

void hash1::directHash()
{
     for(int i=0;i<10;i++)
       arr[i]=-1;
         for(int i=1;i<=10;i++)
{
       int x;
      cout<<"Enter numbers from 1 to 10\n";
      cin>>x;
          int index=x;
         arr[index]=x;
}
}
void hash1::subHash()
{
    for(int i=0;i<10;i++)
      arr[i]=-1;
         for(int i=1;i<=7;i++)
{
       int x;
       cout<<"Enter numbers from 1001 to1010\n";
       cin>>x;
        int index=x-1000;

        arr[index]=x;

}
}
void hash1::digitExHash()
{
       for(int i=0;i<10;i++)
         arr[i]=-1;
           for(int i=1;i<=10;i++)
{
        int x;
       cout<<"Enter a number of 6 digits\n";
        cin>>x;
      int index=0;
    long r,inc=100000,incr=1000;
    for(int i=1;i<=6;i++)
    {
        if(i==1 || i== 3 || i==5)
        {
           incr=incr/10;

r=(x/inc)%10;
           index=index+(r*incr);
        }
        inc=inc/10;
    }
    index=index%10;
     while(arr[index]!=-1)
     index=(index+1)%10;
       arr[index]=x;
}
}

void hash1::foldShiftHash()
{
   for(int i=0;i<10;i++)
    arr[i]=-1;
      for(int i=1;i<=10;i++)
  {
      int x;
      cout<<"Enter a number of 4 digits\n";
             cin>>x;
      index=0;
       long no,no1,no2,no3;

         no1=x/100;
         no3=no1*100;
         no2=x%no3;
         index=no1+no2;
index=index%10;
    if(index==-1)
    {
      arr[index]=x;
    }
       while(arr[index]!=-1)
       index=(index+1)%10;
       arr[index]=x;
    }
  }
  void hash1::foldBoundHash()
  {
      for(int i=0;i<10;i++)
    arr[i]=-1;
      for(int i=1;i<=10;i++)
  {
      int x;
      cout<<"Enter a number of 4 digits\n";
             cin>>x;
      index=0;
       long no,no1,no2,no3;

         no1=x/100;

       no3=no1*100;
      no2=x%no3;
      int tmp=0;
      while(no1>0)
        {

int rem=no1%10;
             tmp=(tmp*10)+rem;
             no1=no1/10;
        }

         int tmp1=0;
        while(no2>0)
        {
            int rem1=no2%10;
             tmp1=(tmp1*10)+rem1;
             no2=no2/10;
        }
         index=tmp+tmp1;
index=index%10;
    if(index==-1)
    {
      arr[index]=x;
    }
       while(arr[index]!=-1)
       index=(index+1)%10;
       arr[index]=x;

    }
  }
int main()
{
    hash1 h;
    int op;
    cout<<"Enter 1 for direct hashing\nEnter 2 for Subtraction 
Hashing\nEnter 3 for Modulo Division Hashing"<<endl;
    cout<<"Enter 4 for digit extraction hashing\nEnter 5 for shift 
fold Hashing\nEnter 6 for shift Boundry Hashing"<<endl;
    cout<<"\nEnter 7 to exit\n"<<endl;
    cin>>op;
    for(int i=0;i<SIZE;i++)
    {
    switch(op)
    {
    case 1:
           h.directHash();
           h.display();
           break;
    case 2:
           h.subHash();
           h.display();
           break;
    case 3:
           h.modDivision();
           h.display();
           break;
    case 4:
           h.digitExHash();
           h.display();
           break;
      case 5:
           h.foldShiftHash();
           h.display(); break;
      case 6:
          h.foldBoundHash();
          h.display(); break;
    }
    }
    return 0;
}
```

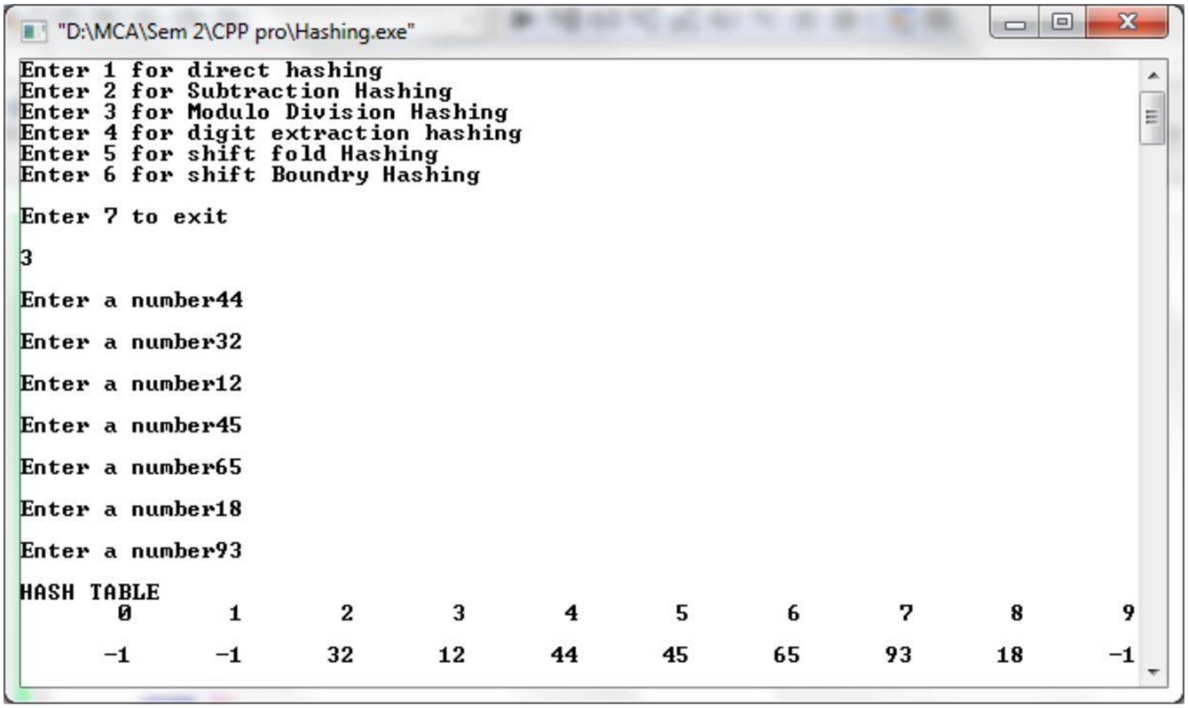
Output :



```
Enter 1 for direct hashing
Enter 2 for Subtraction Hashing
Enter 3 for Modulo Division Hashing
Enter 4 for digit extraction hashing
Enter 5 for shift fold Hashing
Enter 6 for shift Boundry Hashing

Enter 7 to exit

1
Enter numbers from 1 to 10
3
Enter numbers from 1 to 10
2
Enter numbers from 1 to 10
5
Enter numbers from 1 to 10
1
Enter numbers from 1 to 10
4
Enter numbers from 1 to 10
6
Enter numbers from 1 to 10
9
Enter numbers from 1 to 10
8
Enter numbers from 1 to 10
7
Enter numbers from 1 to 10
10

HASH TABLE
       0       1       2       3       4       5       6       7       8       9

      -1       1       2       3       4       5       6       7       8       9
```

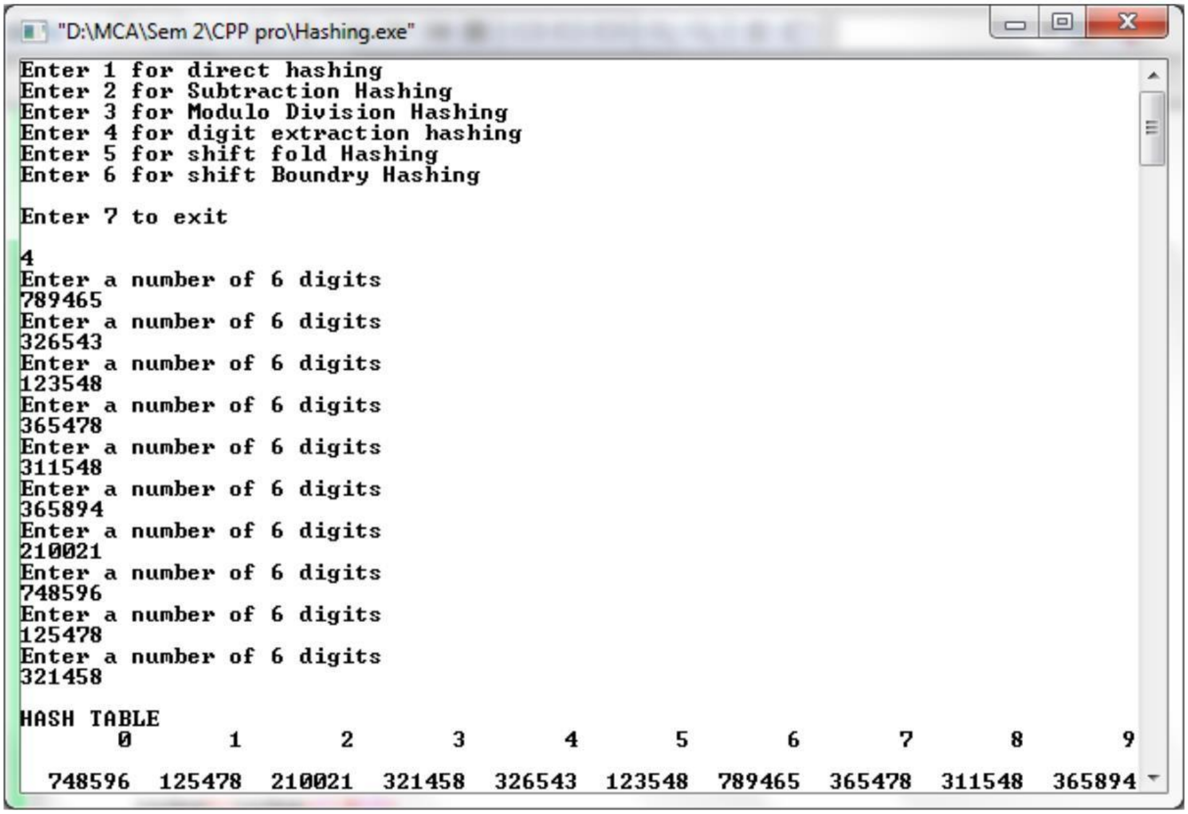


```
Enter 1 for direct hashing
Enter 2 for Subtraction Hashing
Enter 3 for Modulo Division Hashing
Enter 4 for digit extraction hashing
Enter 5 for shift fold Hashing
Enter 6 for shift Boundry Hashing

Enter 7 to exit

2

Enter numbers from 1001 to 1010
1002

Enter numbers from 1001 to 1010
1003

Enter numbers from 1001 to 1010
1005

Enter numbers from 1001 to 1010
1006

Enter numbers from 1001 to 1010
1007

Enter numbers from 1001 to 1010
1009

Enter numbers from 1001 to 1010
1010

HASH TABLE
       0       1       2       3       4       5       6       7       8       9

      -1      -1    1002    1003      -1    1005    1006    1007      -1    1009

Process returned 0 (0x0)   execution time : 16.504 s
Press any key to continue.
```



"D:\MCA\Sem 2\CPP pro\Hashing.exe"

```
Enter 1 for direct hashing
Enter 2 for Subtraction Hashing
Enter 3 for Modulo Division Hashing
Enter 4 for digit extraction hashing
Enter 5 for shift fold Hashing
Enter 6 for shift Boundry Hashing

Enter 7 to exit

3

Enter a number44

Enter a number32

Enter a number12

Enter a number45

Enter a number65

Enter a number18

Enter a number93

HASH TABLE
         0         1         2         3         4         5         6         7         8         9

        -1        -1        32        12        44        45        65        93        18        -1
```



"D:\MCA\Sem 2\CPP pro\Hashing.exe"

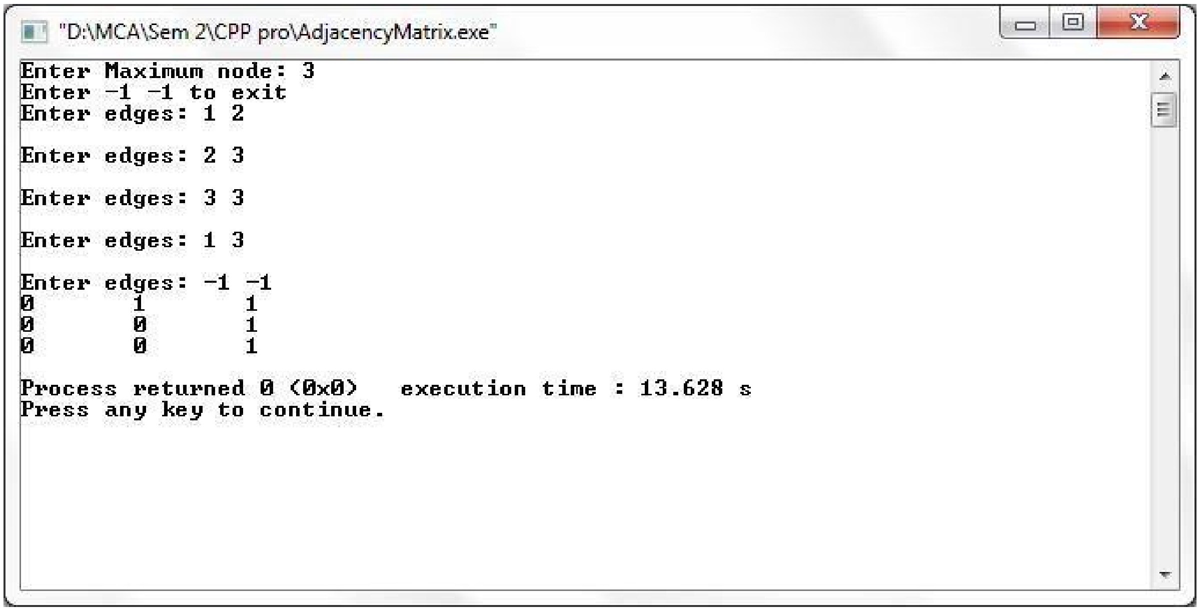
```
Enter 1 for direct hashing
Enter 2 for Subtraction Hashing
Enter 3 for Modulo Division Hashing
Enter 4 for digit extraction hashing
Enter 5 for shift fold Hashing
Enter 6 for shift Boundry Hashing

Enter 7 to exit

4
Enter a number of 6 digits
789465
Enter a number of 6 digits
326543
Enter a number of 6 digits
123548
Enter a number of 6 digits
365478
Enter a number of 6 digits
311548
Enter a number of 6 digits
365894
Enter a number of 6 digits
210021
Enter a number of 6 digits
748596
Enter a number of 6 digits
125478
Enter a number of 6 digits
321458

HASH TABLE
         0         1         2         3         4         5         6         7         8         9

    748596    125478    210021    321458    326543    123548    789465    365478    311548    365894
```

PRACTICAL 18

Aim: Create a Graph Storage Structure (eg. Adjacency Matrix).

```
#include<iostream>
using namespace std;

class adjMatrix
{
    int **adj;
    bool *visited;
    int n,i,j;
public:
    adjMatrix(int n)
    {
        this->n=n;
        visited=new bool[n];
        adj=new int *[n];
        for(i=1;i<=n;i++)
        {
            adj[i]=new int [n];
            for(j=1;j<=n;j++)
            {
                adj[i][j]=0;
            }
        }
    }
    int add_edge(int origin, int dest)
    {
        if(origin>n||dest>n||origin<0||dest<0)
        {
            cout<<"Wrong nodes";
        }
        else
        {
            adj[origin][dest]=1;
        }
    }
    int display()
    {
         for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                cout<<adj[i][j]<<"\t";
            }

            cout<<"\n";
        }
    }

};
int main()
{
    int nodes, Max_edges,i, origin, dest;
    cout<<"Enter Maximum node: ";
    cin>>nodes;
    adjMatrix am(nodes);
    Max_edges=nodes*(nodes-1);
    cout<<"Enter -1 -1 to exit";
    for(i=0;i<Max_edges;i++)
     {
         cout<<"\nEnter edges: ";
         cin>>origin>>dest;
         if((origin==-1)&&(dest==-1))
            break;
         else
            am.add_edge(origin,dest);
     }
     am.display();
     return 0;
}
```

Output :



```
Enter Maximum node: 3
Enter -1 -1 to exit
Enter edges: 1 2

Enter edges: 2 3

Enter edges: 3 3

Enter edges: 1 3

Enter edges: -1 -1
0       1       1
0       0       1
0       0       1

Process returned 0 (0x0)   execution time : 13.628 s
Press any key to continue.
```

PRACTICAL 19

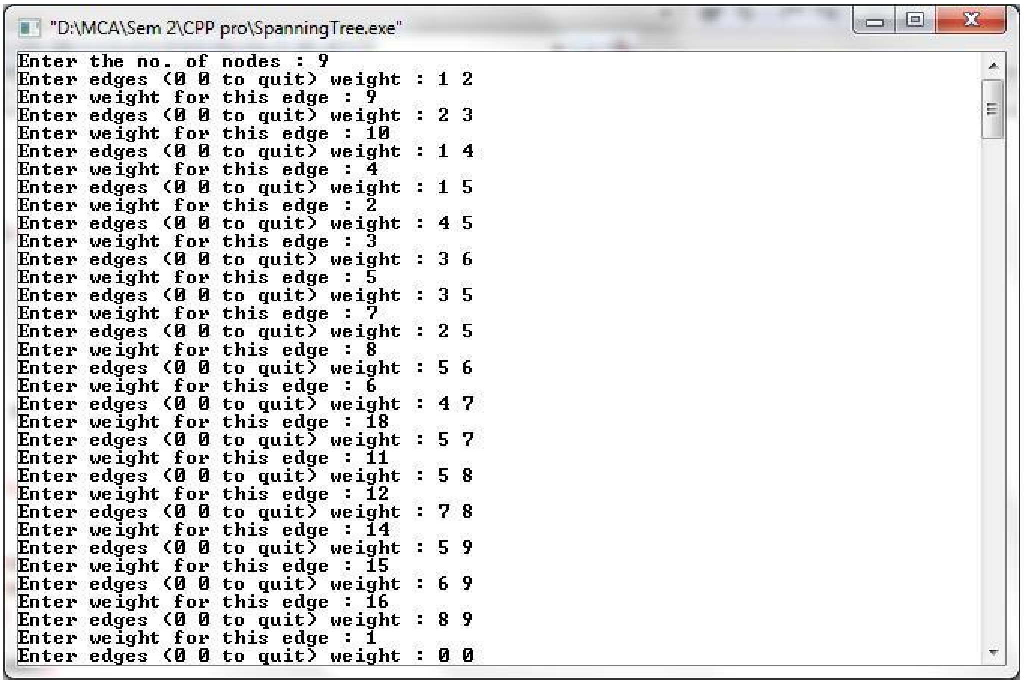
Aim: Create a Minimum Spanning Tree Using Any Method Kruskal's Algorithm.

```
#include<iostream>
#include<stdlib.h>
#define  max  30
using namespace std;

    struct edge
    {
        int weight;
        int u;
        int v;
        struct edge *link;
    };
    struct edge *frnt=NULL;
    struct edge *tmp;

     int  i,j,wt;
    int father[max];
    struct edge tree[max];
    int wt_tree;
    int cnt=0;

    void make_tree();
    void insert_tree(int i, int j, int wt);
    void insert_pque(int i, int j, int wt);
    struct edge *del_pque();

    void create_graph()
    {
        int i, n, max_edges, origin, destin;
        cout<<"Enter the no. of nodes : ";
        cin>>n;
        max_edges=n*(n-1)/2;
        for(i=1;i<max_edges;i++)
        {
            cout<<"Enter edges (0 0 to quit) weight : ";
            cin>>origin;
            cin>>destin;
            if((origin==0)&&(destin==0))
```

```
                break;
            cout<<"Enter weight for this edge : ";
               cin>>wt;
            if(origin>n || destin>n || origin<=0 || destin<=0)

            {
                cout<<"Invalid edge \n";
                i--;

}
            else
                insert_pque(origin,destin,wt);
        }
            if(i<n-1)
            {
                cout<<"Spanning tree is not possible
                \n"; exit(1);
            }
    }
    void insert_pque(int i,int j,int wt)
    {
        struct edge *tmp, *q;
        tmp = (struct edge *) malloc(sizeof(struct edge));
        tmp->u=i;
        tmp->v=j;
        tmp->weight=wt;
        if(frnt==NULL || tmp->weight<frnt->weight)
        {
            tmp->link=frnt;
            frnt=tmp;
        }
        else
        {
            q=frnt;
        while(q->link!=NULL&&q->link->weight<=tmp->weight)
          q=q->link;
        tmp->link=q->link;
        q->link=tmp;
        if(q->link==NULL)
            tmp->link=NULL;
        }
    }
    void make_tree()
    {
        edge *tmp;
        int node1, node2, root_n1, root_n2, wt_root=0,n,cnt=0;
        while(cnt<n-1)
        {
            tmp=del_pque();
            node1=tmp->u;
            node2=tmp->v;
            cout<<"N1 ="<<node1;
            cout<<"N2 ="<<node2;
            while(node1>0)
            {
                root_n1=node1;
                node1=father[node1];
            }
            while(node2>0)
            {
                root_n2=node2;
                node2=father[node2];
            }
            cout<<"root N1= "<<root_n1;
            cout<<"root N2= "<<root_n2;
            if(root_n1!=root_n2)

{
                insert_tree(tmp->u,tmp->v,tmp->weight);
                wt_tree=wt_tree+tmp->weight;
                father[root_n2]=root_n1;
            }
        }

    }

    void insert_tree(int i, int j, int wt)
    {
        cout<<"This edge inserted in the spanning tree
        \n"; cnt++;
        tree[cnt].u=i;
        tree[cnt].v=j;
        tree[cnt].weight=wt;
    }

   struct edge *del_pque()
    {
        struct edge * tmp;
        tmp=frnt;
        cout<<"Edge processed \n"<<tmp->u;
        cout<<"Edge processed \n"<<tmp->v;
```

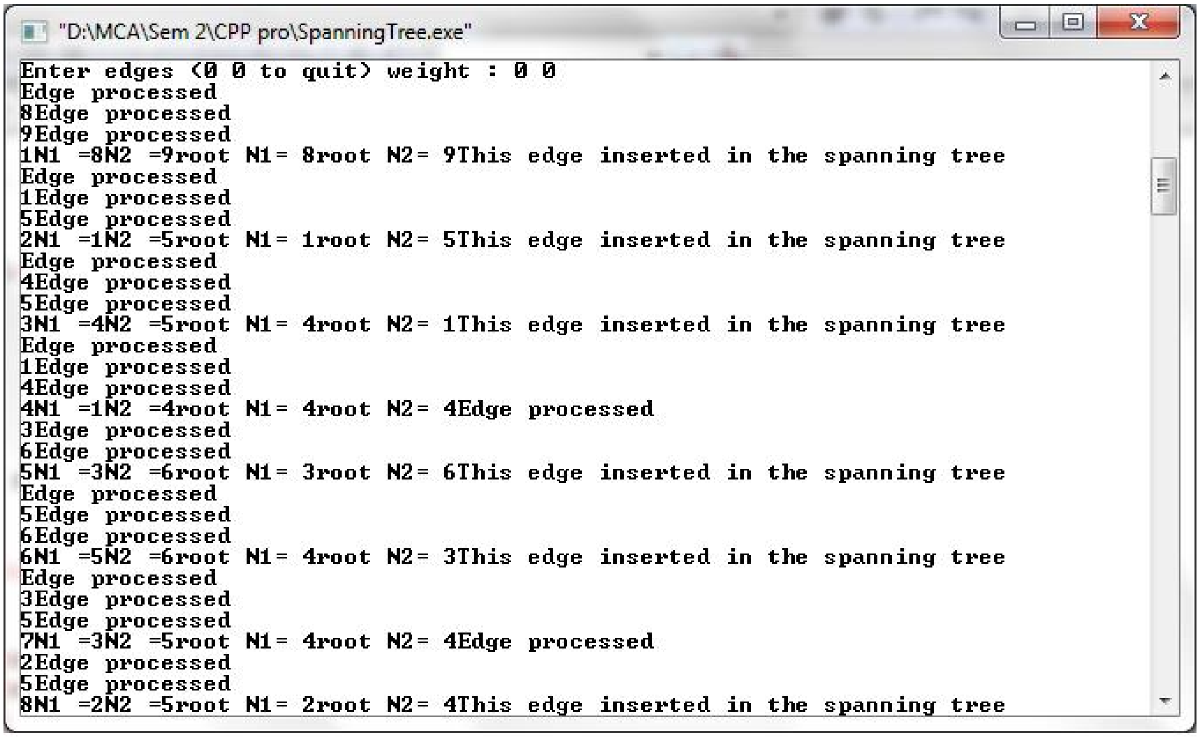
```
        cout<<"Edge processed \n"<<tmp->weight;
        frnt=frnt->link;
        return tmp;
    }

  int main()
{
    int i,j,wt_tree,cnt=0;
    struct edge tree[max];
    create_graph();
    make_tree();
    cout<<"Edges to be included in spanning tree \n";
    for(i=1;i<=cnt;i++)
    {
        cout<<tree[i].u;
        cout<<tree[j].v;
    }
    cout<<"Weight of this spanning tree is
    :"<<wt_tree; return 0;
}
```

Output :

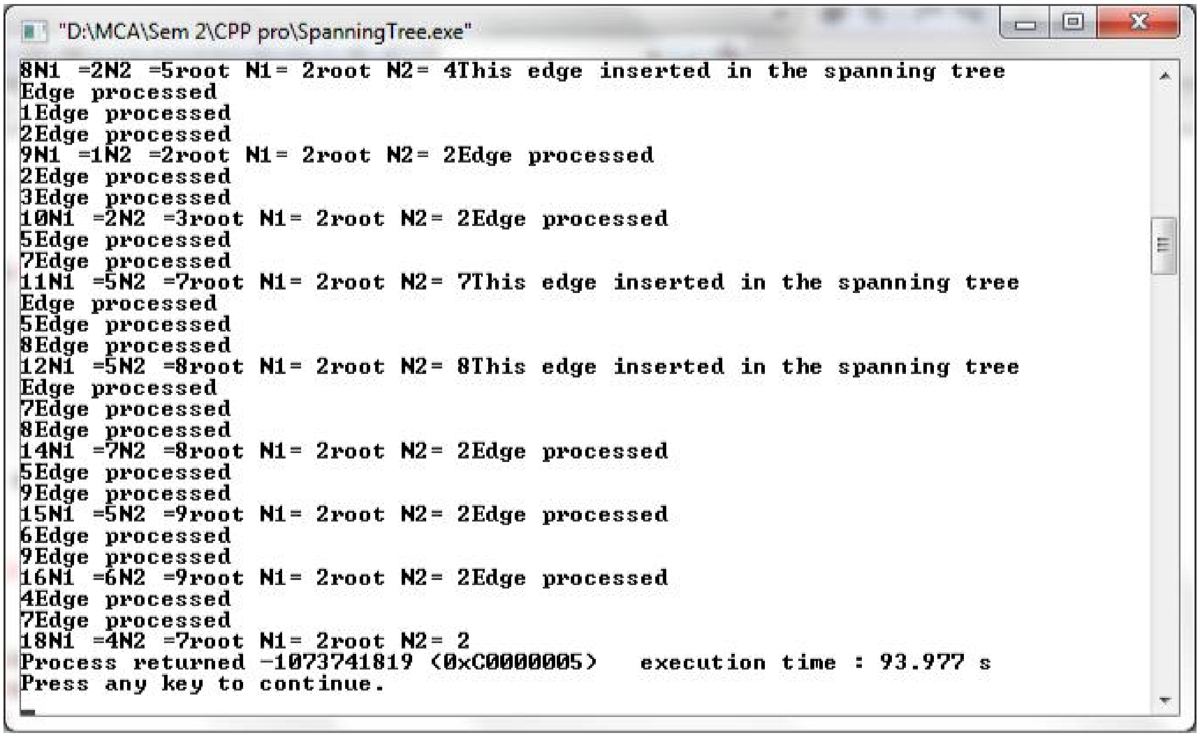


"D:\MCA\Sem 2\CPP pro\SpanningTree.exe"

```
Enter the no. of nodes : 9
Enter edges (0 0 to quit) weight : 1 2
Enter weight for this edge : 9
Enter edges (0 0 to quit) weight : 2 3
Enter weight for this edge : 10
Enter edges (0 0 to quit) weight : 1 4
Enter weight for this edge : 4
Enter edges (0 0 to quit) weight : 1 5
Enter weight for this edge : 2
Enter edges (0 0 to quit) weight : 4 5
Enter weight for this edge : 3
Enter edges (0 0 to quit) weight : 3 6
Enter weight for this edge : 5
Enter edges (0 0 to quit) weight : 3 5
Enter weight for this edge : 7
Enter edges (0 0 to quit) weight : 2 5
Enter weight for this edge : 8
Enter edges (0 0 to quit) weight : 5 6
Enter weight for this edge : 6
Enter edges (0 0 to quit) weight : 4 7
Enter weight for this edge : 18
Enter edges (0 0 to quit) weight : 5 7
Enter weight for this edge : 11
Enter edges (0 0 to quit) weight : 5 8
Enter weight for this edge : 12
Enter edges (0 0 to quit) weight : 7 8
Enter weight for this edge : 14
Enter edges (0 0 to quit) weight : 5 9
Enter weight for this edge : 15
Enter edges (0 0 to quit) weight : 6 9
Enter weight for this edge : 16
Enter edges (0 0 to quit) weight : 8 9
Enter weight for this edge : 1
Enter edges (0 0 to quit) weight : 0 0
```



```
Enter edges (0 0 to quit) weight : 0 0
Edge processed
8Edge processed
9Edge processed
1N1 =8N2 =9root N1= 8root N2= 9This edge inserted in the spanning tree
Edge processed
1Edge processed
5Edge processed
2N1 =1N2 =5root N1= 1root N2= 5This edge inserted in the spanning tree
Edge processed
4Edge processed
5Edge processed
3N1 =4N2 =5root N1= 4root N2= 1This edge inserted in the spanning tree
Edge processed
1Edge processed
4Edge processed
4N1 =1N2 =4root N1= 4root N2= 4Edge processed
3Edge processed
6Edge processed
5N1 =3N2 =6root N1= 3root N2= 6This edge inserted in the spanning tree
Edge processed
5Edge processed
6Edge processed
6N1 =5N2 =6root N1= 4root N2= 3This edge inserted in the spanning tree
Edge processed
3Edge processed
5Edge processed
7N1 =3N2 =5root N1= 4root N2= 4Edge processed
2Edge processed
5Edge processed
8N1 =2N2 =5root N1= 2root N2= 4This edge inserted in the spanning tree
```



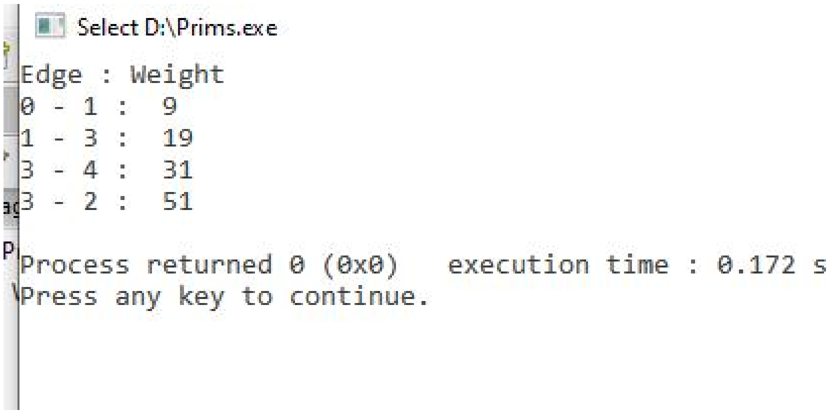
```
8N1 =2N2 =5root N1= 2root N2= 4This edge inserted in the spanning tree
Edge processed
1Edge processed
2Edge processed
9N1 =1N2 =2root N1= 2root N2= 2Edge processed
2Edge processed
3Edge processed
10N1 =2N2 =3root N1= 2root N2= 2Edge processed
5Edge processed
7Edge processed
11N1 =5N2 =7root N1= 2root N2= 7This edge inserted in the spanning tree
Edge processed
5Edge processed
8Edge processed
12N1 =5N2 =8root N1= 2root N2= 8This edge inserted in the spanning tree
Edge processed
7Edge processed
8Edge processed
14N1 =7N2 =8root N1= 2root N2= 2Edge processed
5Edge processed
9Edge processed
15N1 =5N2 =9root N1= 2root N2= 2Edge processed
6Edge processed
9Edge processed
16N1 =6N2 =9root N1= 2root N2= 2Edge processed
4Edge processed
7Edge processed
18N1 =4N2 =7root N1= 2root N2= 2
Process returned -1073741819 (0xC0000005)   execution time : 93.977 s
Press any key to continue.
```

PRACTICAL 20

Aim: Create a Minimum Spanning Tree Using Any Method Prim’s Algorithm.

```
#include <cstring>
#include <iostream>
using namespace std;

#define INF 9999999


#define V 5


int G[V][V] = {
  {0, 9, 75, 0, 0},
  {9, 0, 95, 19, 42},
  {75, 95, 0, 51, 66},
  {0, 19, 51, 0, 31},
  {0, 42, 66, 31, 0}};

int main() {
  int no_edge;  // number of edge


  int selected[V];

  memset(selected, false, sizeof(selected));

  no_edge = 0;

  selected[0] = true;

  int x;  //  row number
  int y;  //  col number

  cout << "Edge"
     << " : "
     << "Weight";
  cout << endl;
  while (no_edge < V - 1) {


    int min = INF;
    x = 0;
    y = 0;

    for (int i = 0; i < V; i++)
      {if (selected[i]) {
        for (int j = 0; j < V; j++) {
          if (!selected[j] && G[i][j]) {  // not in selected and there is
an edge
            if (min > G[i][j]) {
              min = G[i][j];
              x = i;
              y = j;
            }
          }
        }
      }
    }
    cout << x << " - " << y << " :  " << G[x][y];
    cout << endl;
    selected[y] = true;
    no_edge++;
  }

  return 0;
}
```

Output :



```
Edge : Weight
0 - 1 :  9
1 - 3 :  19
3 - 4 :  31
3 - 2 :  51

Process returned 0 (0x0)   execution time : 0.172 s
Press any key to continue.
```