**Q. Write a C++ program to perform Sequential Search**.

#include<iostream>

#include<conio.h>

using namespace std;

int main()

{

int a[10],n,i,x;

int flag=0;

cout<<"Enter 10 elements:\n";

for(i=0;i<10;i++)

{

cin>>a[i];

}

cout<<"\nWhich element you want to search -\n";

cin>>x;

for(i=0;i<10;i++)

{

if(x==a[i])

{

flag=1;

break;

}

}

if(flag==1)

{

cout<<"Number is found.\n";

}

else

{

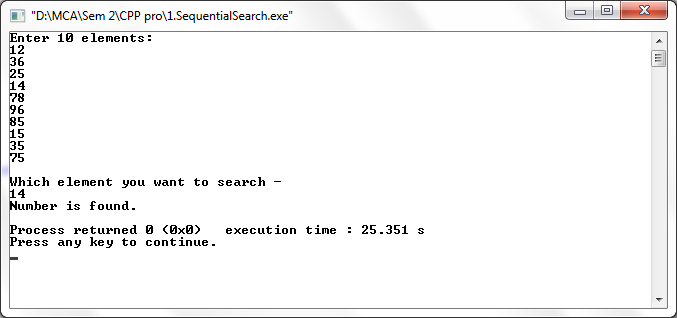
cout<<"The number is not in this array.\n";

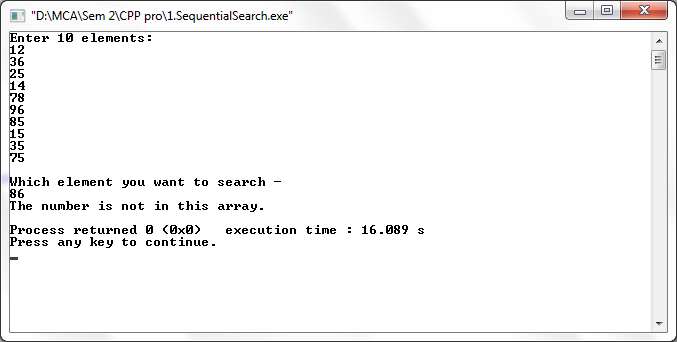
}

return 0;

}

**Output –**





**Q. Write a C++ program to perform 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])

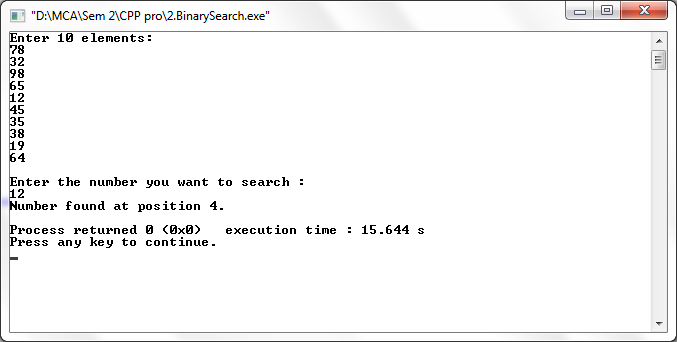
cout<<"Element found at position "<<mid<<".\n";

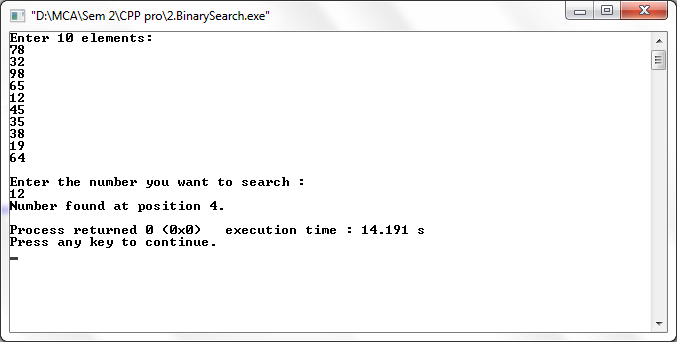
if(st>ed)

cout<<"Element not found.\n";

}

**Output –**





**Q. Write a C++ program to perform 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++)

{

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;

}

}

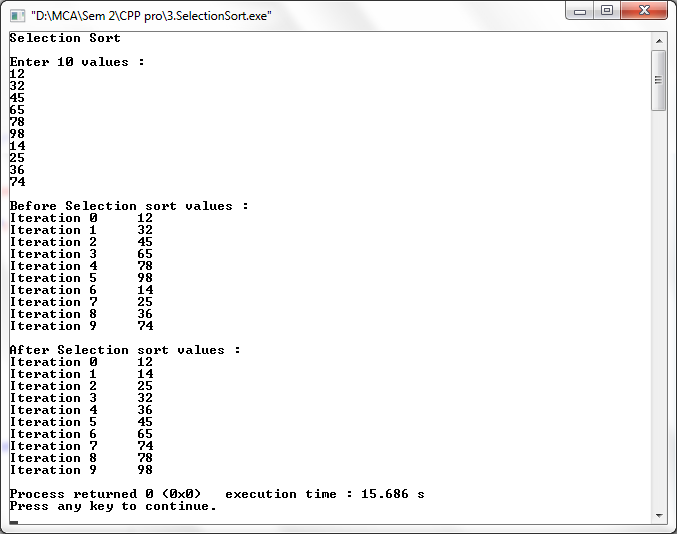
cout<<"Iteration "<<i<<"\t"<<a[i]<<"\n";

}

return 0;

}

**Output –**



**Q. Write a C++ program to perform 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++)

{

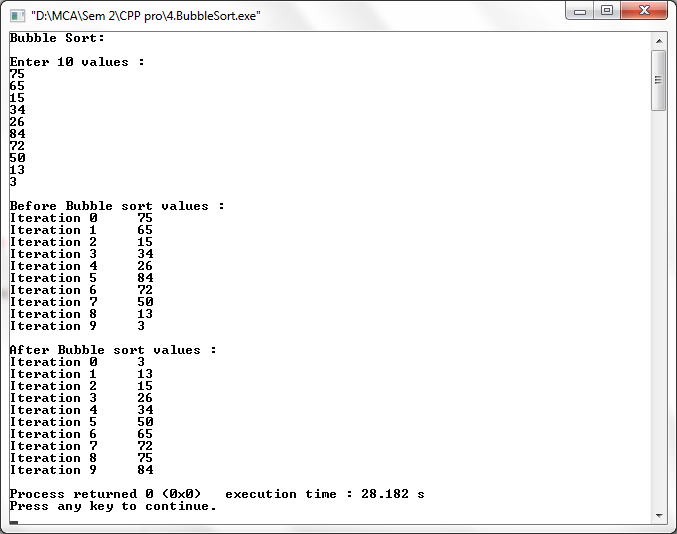
cout<<"Iteration "<<i<<"\t"<<a[i]<<"\n";

}

return 0;

}

**Output –**



**Q. Write a C++ program to perform Insertion Sort.**

#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++)

{

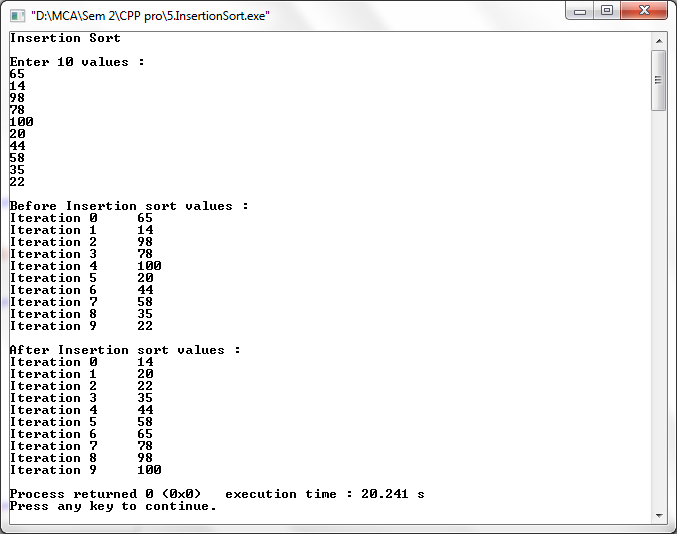
cout<<"Iteration "<<i<<"\t"<<a[i]<<"\n";

}

return 0;

}

**Output –**



**Q. Write a C++ program to perform Merge Sort.**

#include<iostream>

using namespace std;

int main()

{

int i;

int j;

int k;

int a[10], b[10],c[25], m, n,l;

cout<<"Merge Sort :\n\n";

cout<<"Enter the size of Array 1 :\n";

cin>>m;

cout<<"Enter values for Array 1 :\n";

for(i=0;i<m;i++)

{

cin>>a[i];

}

cout<<"\nEnter the size of Array 2 :\n";

cin>>n;

cout<<"Enter values for Array 2 :\n";

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

{

cin>>b[j];

}

cout<<"\nBefore Merge sort values :\n";

for(i=0;i<m;i++)

{

cout<<a[i]<<"\n";

}

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

{

cout<<b[j]<<"\n";

}

l=m+n;

i=j=k=0;

while((i<m) && (j<n))

{

if(a[i]<b[j])

{

c[k]=a[i];

i++;

}

else if(a[i]>b[j])

{

c[k]=b[j];

j++;

}

else

{

c[k]=a[i];

i++;

j++;

}

k++;

}

while(i<m)

{

c[k]=a[i];

i++;

k++;

}

while(j<n)

{

c[k]=b[j];

j++;

k++;

}

cout<<"\nAfter Merge sort values :\n";

for(k=0; k<l;k++)

{

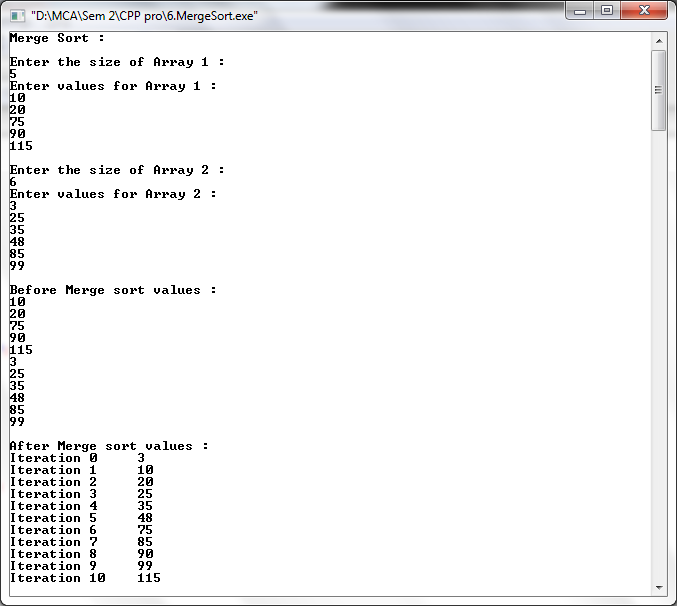
cout<<"Iteration "<<k<<"\t"<<c[k]<<"\n";

}

return 0;

}

**Output –**



**Q. Write a C++ program to perform Quick Sort.**

#include<iostream>

using namespace std;

int partition1(int a[], int p,int r);

void quicksort(int a[],int p,int r)

{

int q;

if(p<r)

{

q= partition1(a,p,r);

quicksort(a,p,q-1);

quicksort(a,q+1,r);

}

}

int partition1(int a[],int p, int r)

{

int t,j,k,y,i,x;

x=a[r];

i=p-1;

for(j=p;j<r;j++)

{

if(a[j]<=x)

{

i=i+1;

t=a[i];

a[i]=a[j];

a[j]= t;

}

}

k=a[i+1];

a[i+1]=a[r];

a[r]=k;

y = i+1;

return y;

}

int main()

{

int a[10],i,n;

cout<<"Quick Sort\n\n";

cout<<"Enter the size of Array :\n";

cin>>n;

cout<<"Enter the values :\n";

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

{

cin>>a[i];

}

cout<<"\nBefore Quick sort values :\n";

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

{

cout<<"Iteration "<<i<<"\t"<<a[i]<<"\n";

}

quicksort(a,0,n);

cout<<"\nAfter Quick sort values :\n";

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

{

cout<<"Iteration "<<i<<"\t"<<a[i]<<"\n";

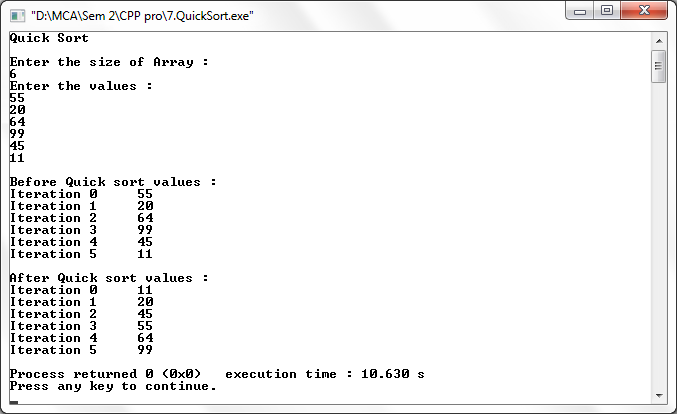
//cout<<"Iteration "<<i<<""<<a[i]<<"\n";

}

return 0;

}

**Output –**



**Q. Write a C++ program to perform 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()

{

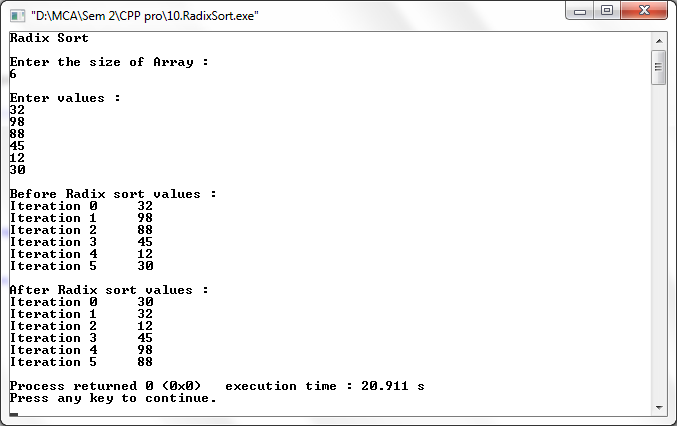
radix r;

r.sort();

return 0;

}

**Output –**



**Q. Write a C++ program to perform 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++)

{

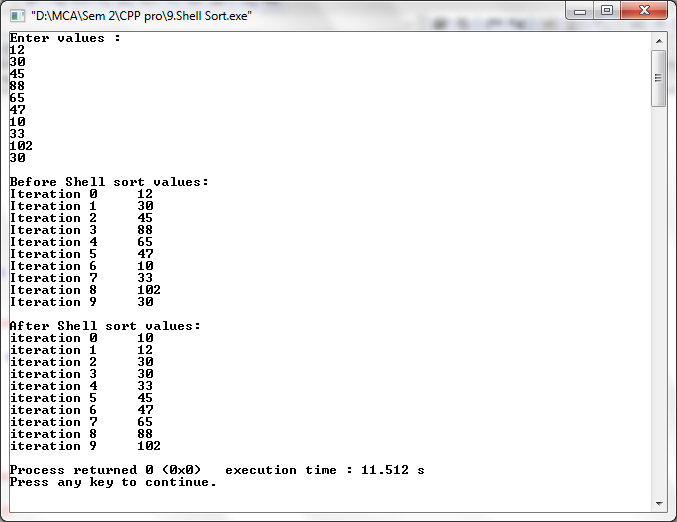
cout<<"iteration "<<i<<"\t"<<a[i]<<"\n";

}

return 0;

}

**Output –**



**Q. Write a C++ program to perform Stack operations using Array.**

#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";

}

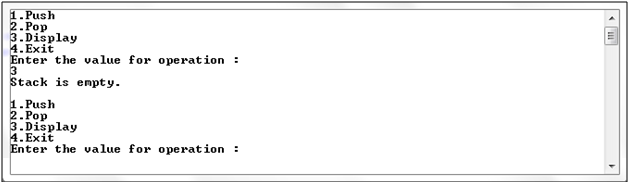
}

return 0;

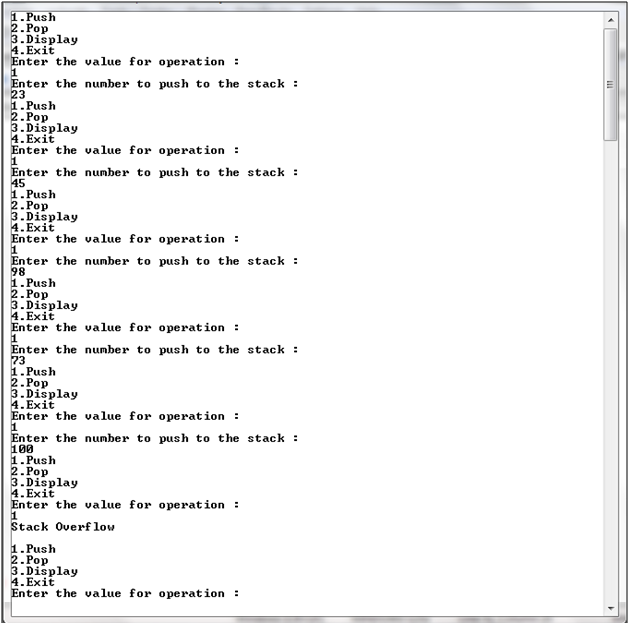
}

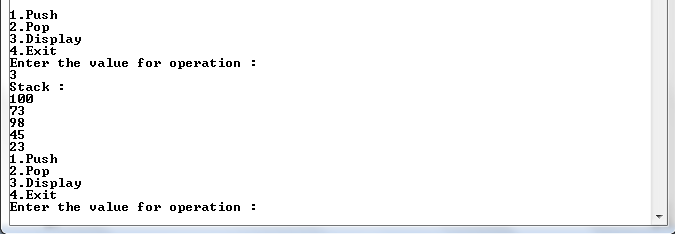
**Output –**

Stack Empty

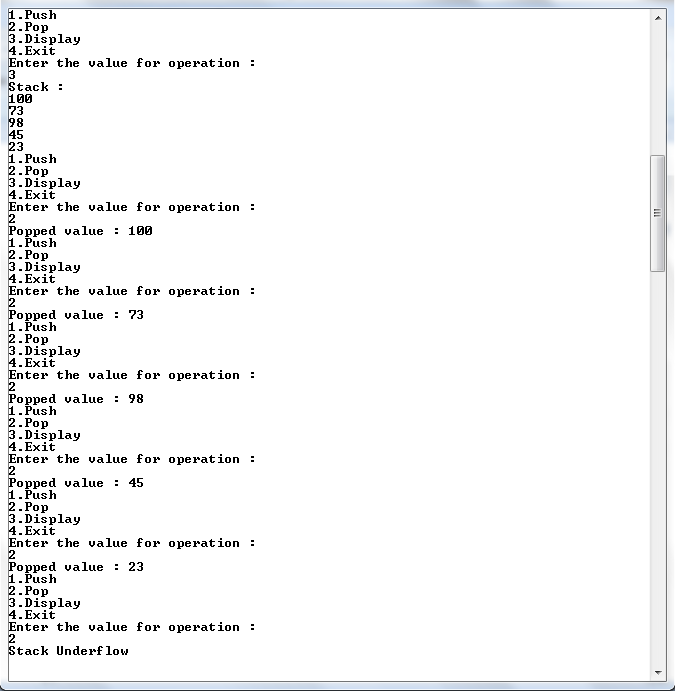


Push operation





Pop operation



**Q. Write a C++ program to perform Stack operations using Link 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";

}

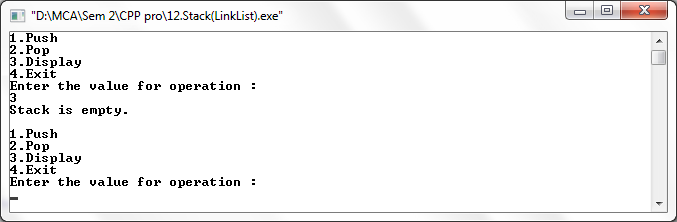
}

return 0;

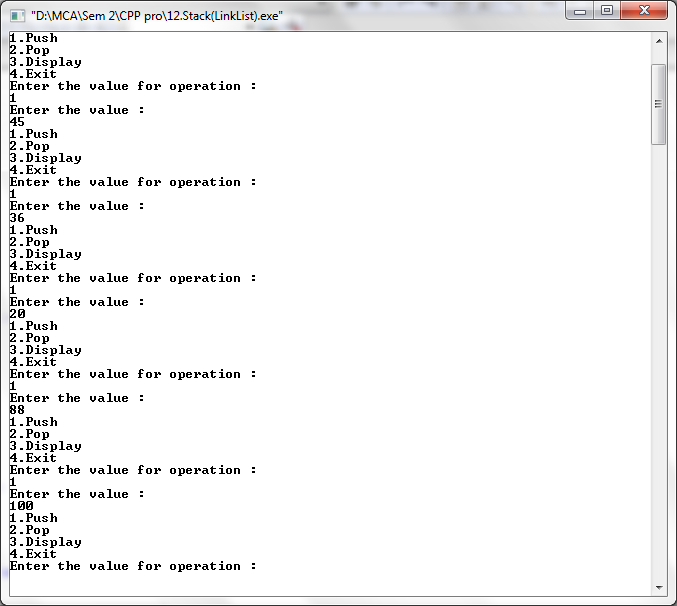
}

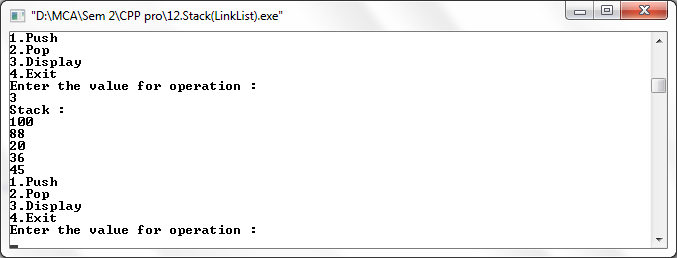
**Output –**

Stack Empty



Push operation





Pop operation



**Q. Write a C++ program to perform Stack operations for 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";

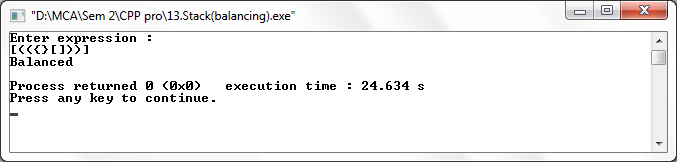
getch();

return 0;

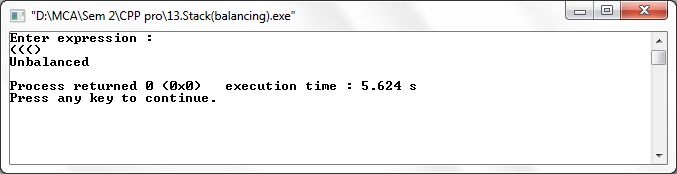
}

**Output –**

Balanced



Unbalanced



**Q. Write a C++ program to perform 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;

}

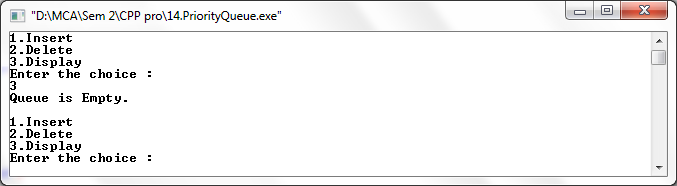
}

return 0;

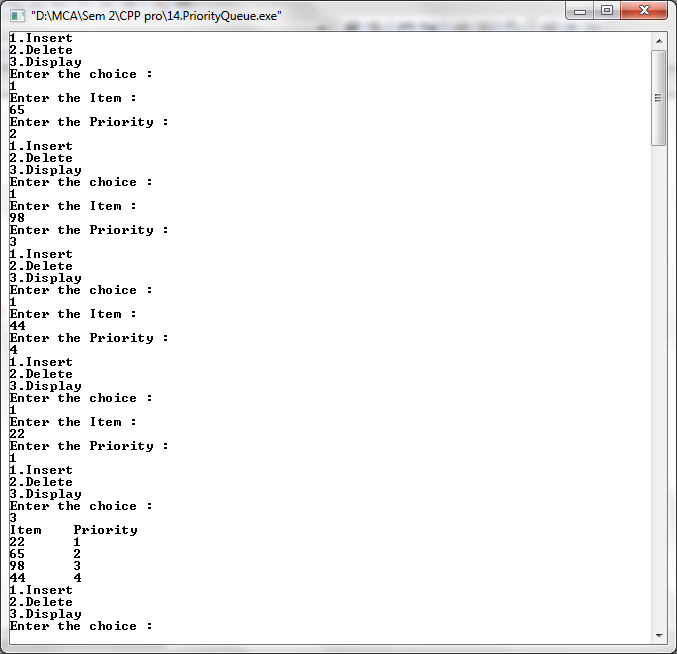
}

**Output –**

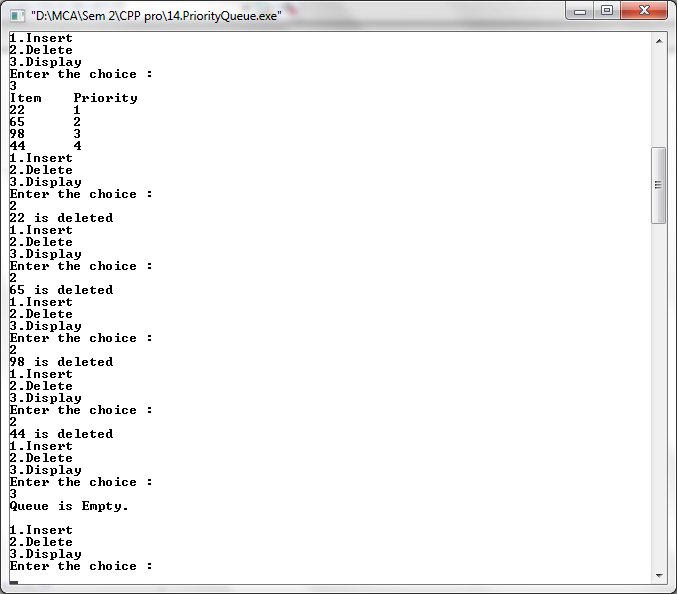
Queue is empty



Insert



Delete



**Q. Write a C++ program to perform 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";

}

}

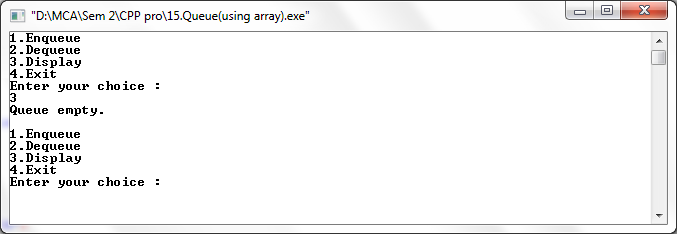
while(ch<4);

return 0;

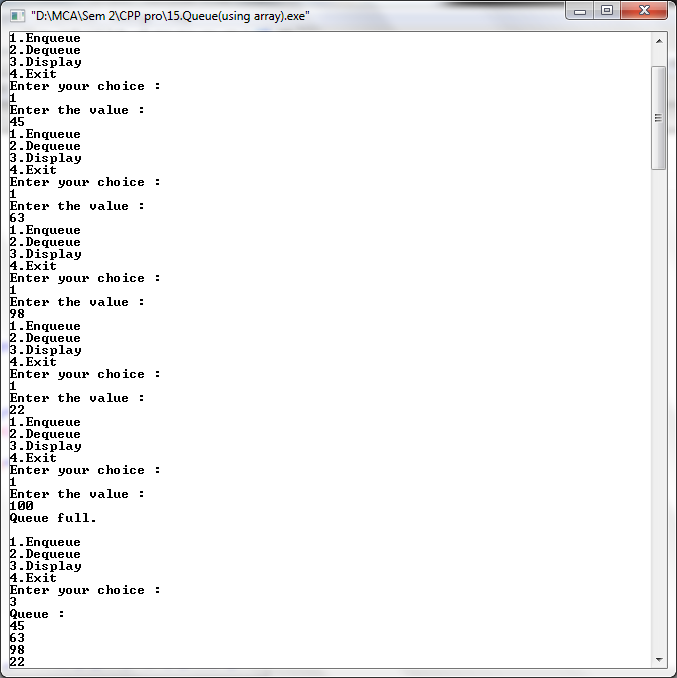
}

**Output –**

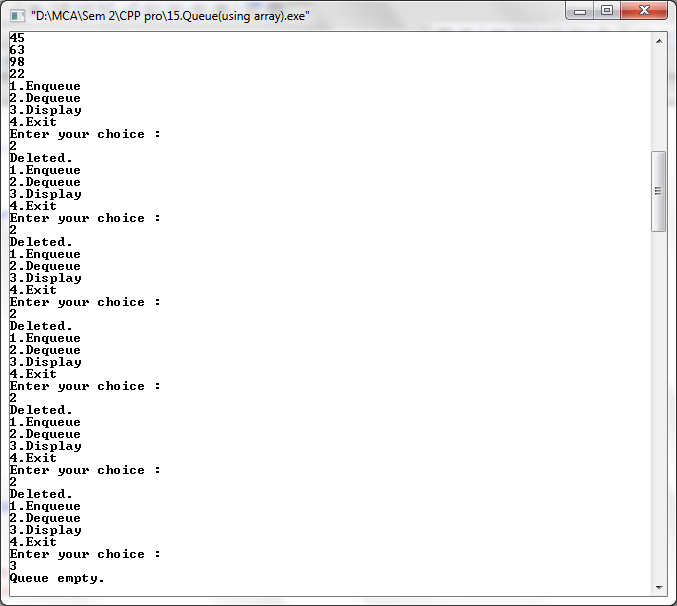
Queue Empty



Insert



Delete



**Q. Write a C++ program to perform Queue using Link 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";

}

}

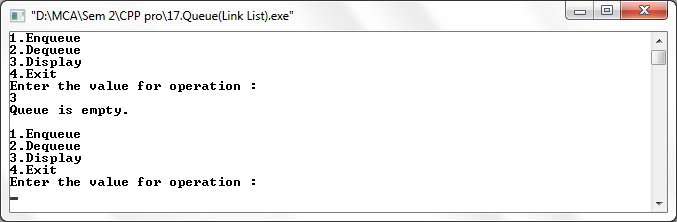
while(n!=4);

return 0;

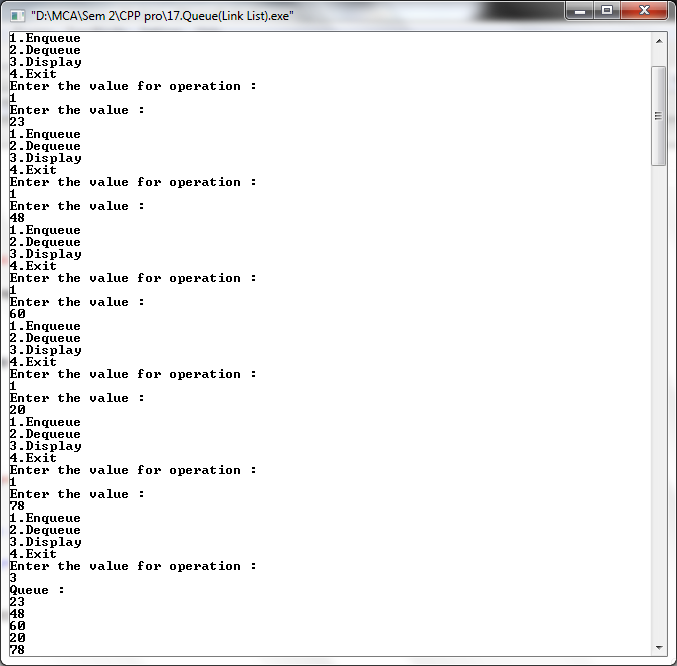
}

**Output –**

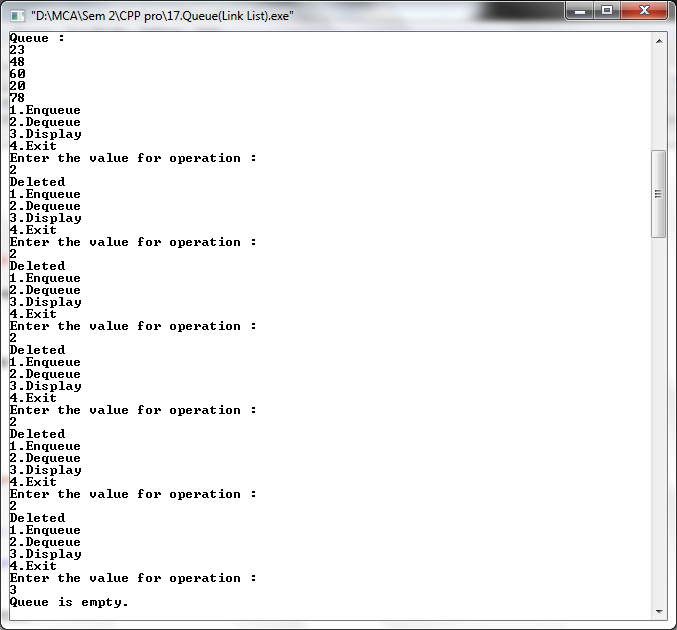
Queue Empty



Insert



Delete



**Q. Write a C++ program to perform Singly Linked List.**

#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";

}

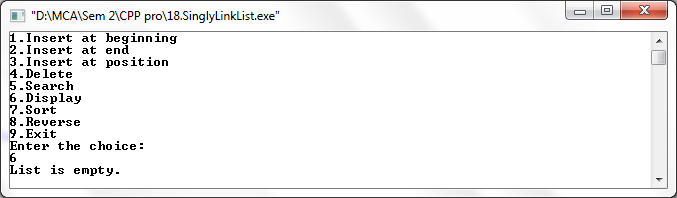
}

return 0;

}

**Output –**

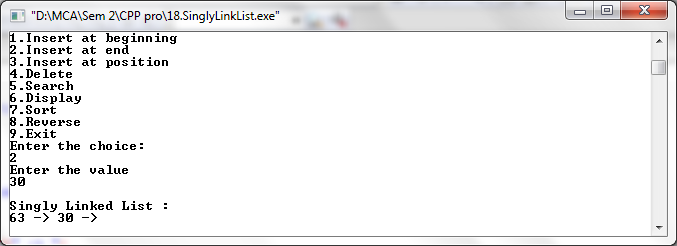
List Empty



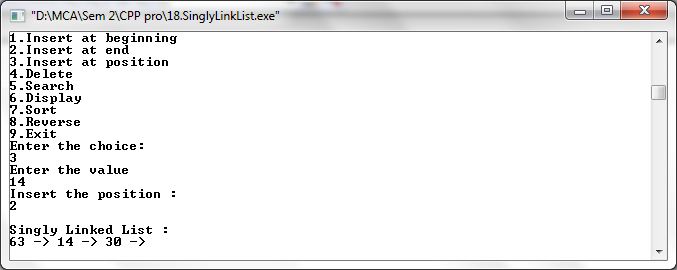
Insert at beginning



Insert at end



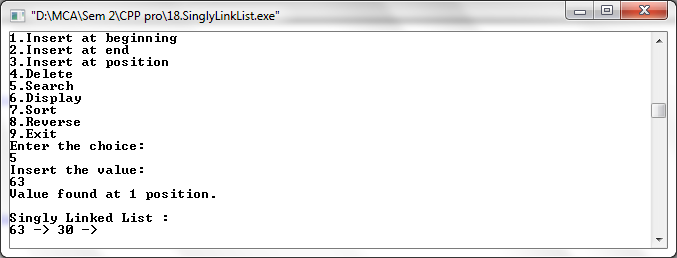
Insert at position



Delete



Search



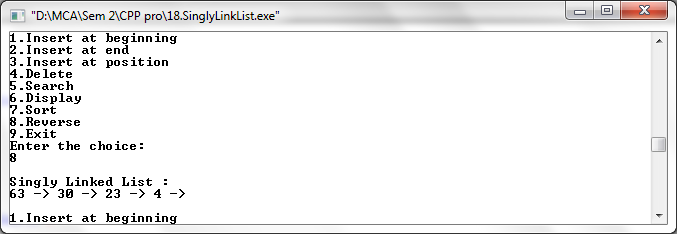
Display



Sort



Reverse



**Q. Write a C++ program to perform Doubly Linked List.**

#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";

}

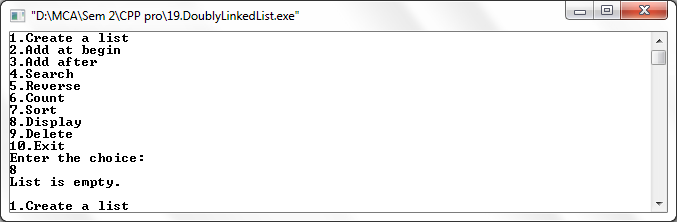
}

return 0;

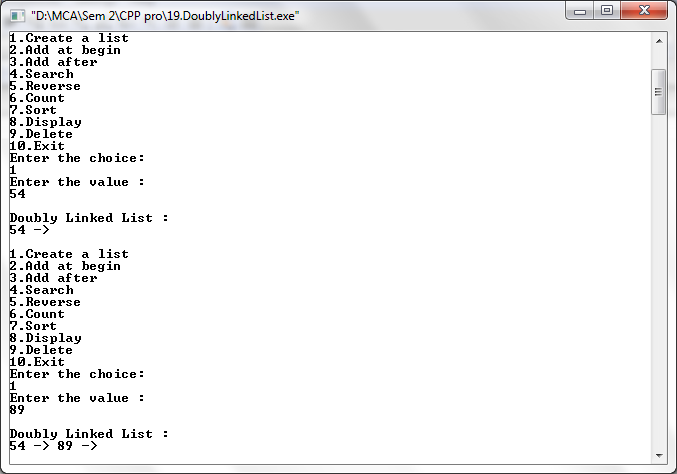
}

**Output –**

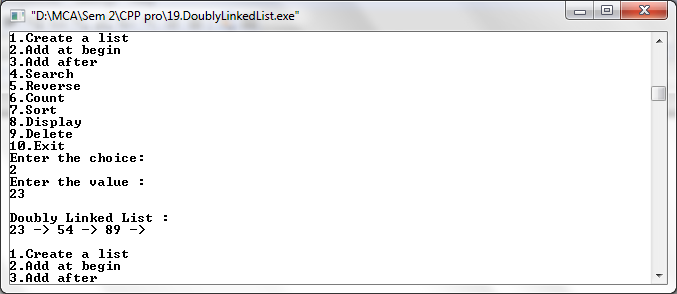
List is empty



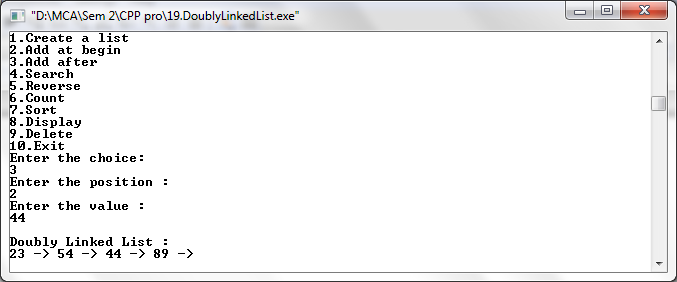
Create List



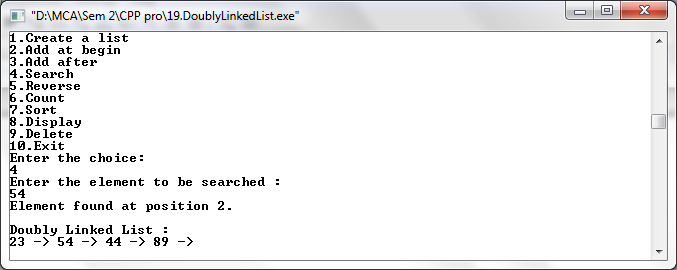
Add at beginning



Add after



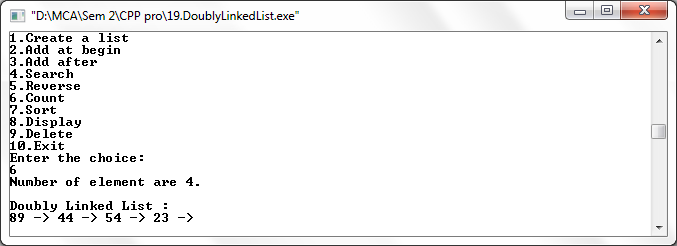
Search



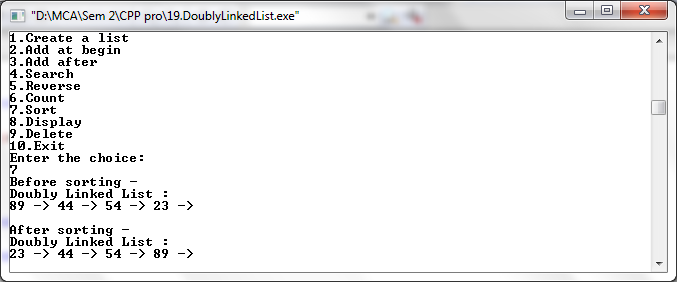
Reverse



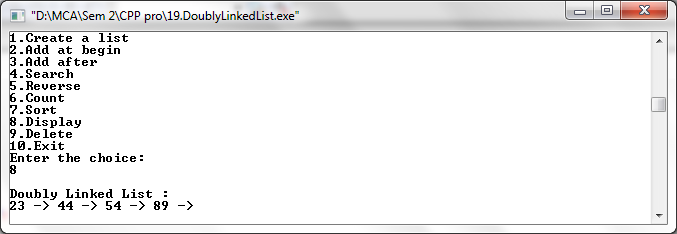
Count



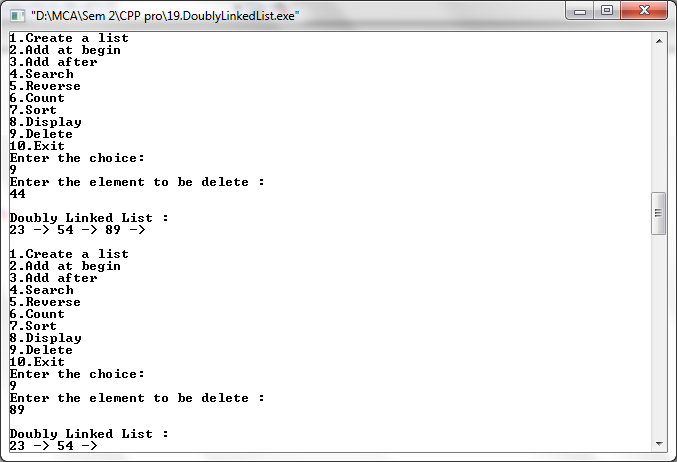
Sort



Display



Delete



**Q. Write a C++ program to perform Singly Circular Linked List.**

#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;

d.search1(x);

d.display();

break;

case 5:

cout<<"Before sorting -";

d.display();

d.sort();

cout<<"After sorting -";

d.display();

break;

case 6:

d.count1();

d.display();

break;

case 7:

d.display();

break;

case 8:

cout<<"Enter the element to be delete :\n";

cin>>x;

d.del(x);

d.display();

break;

case 9:

break;

default:

cout<<"Wrong choice.\n";

}

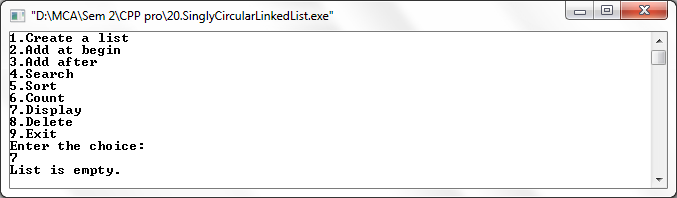
}

return 0;

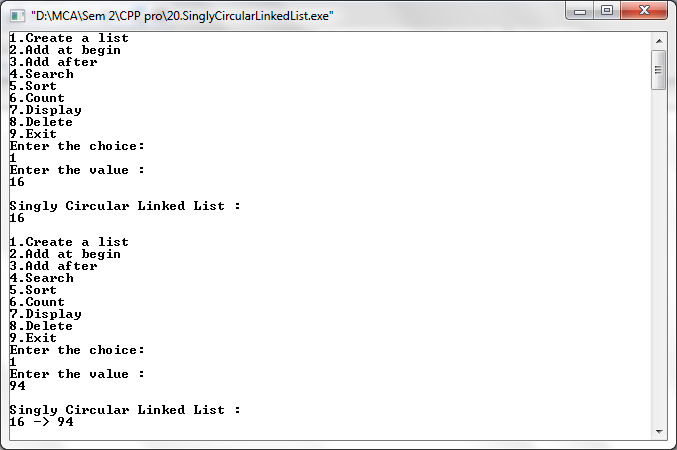
}

**Output –**

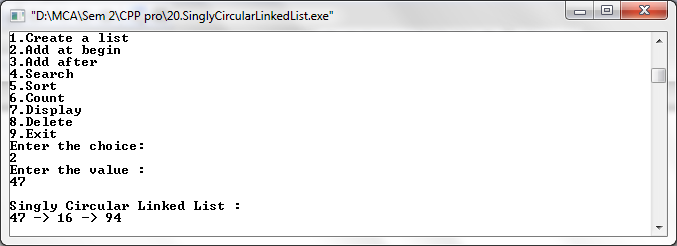
List is Empty



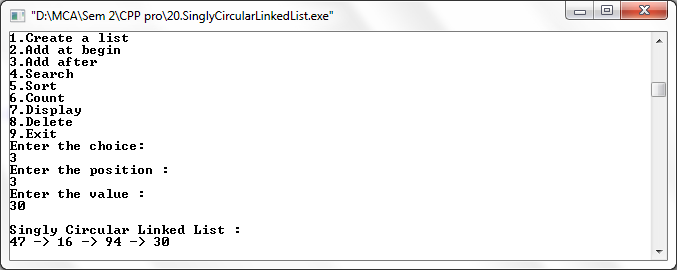
Create a list



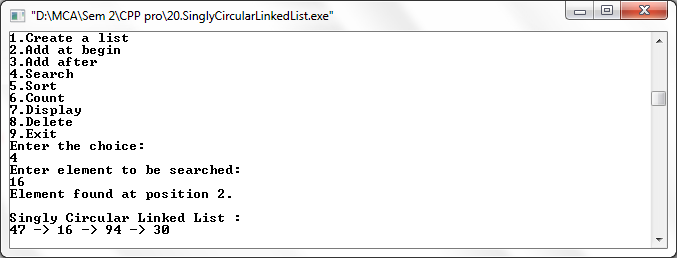
Add at beginning



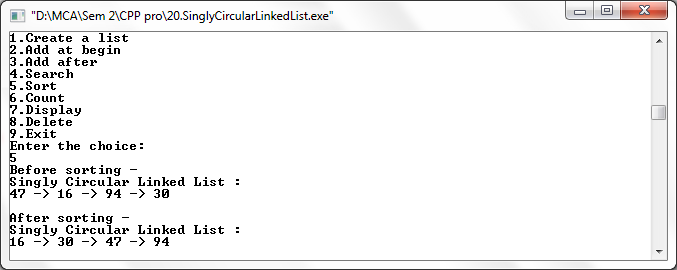
Add after



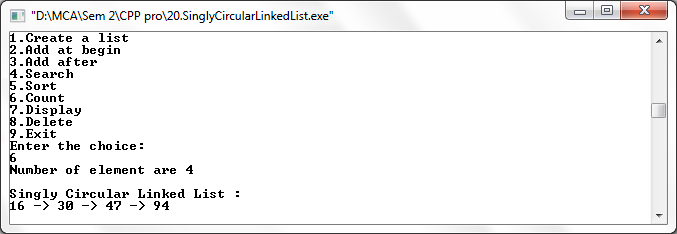
Search



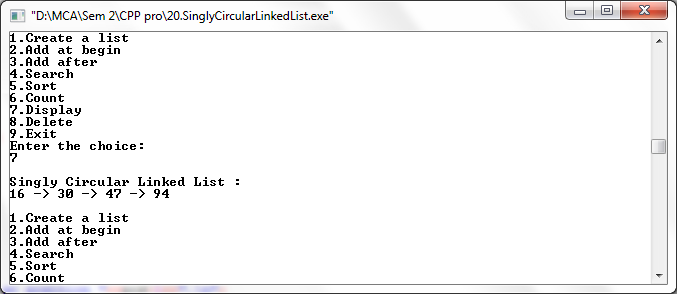
Sort



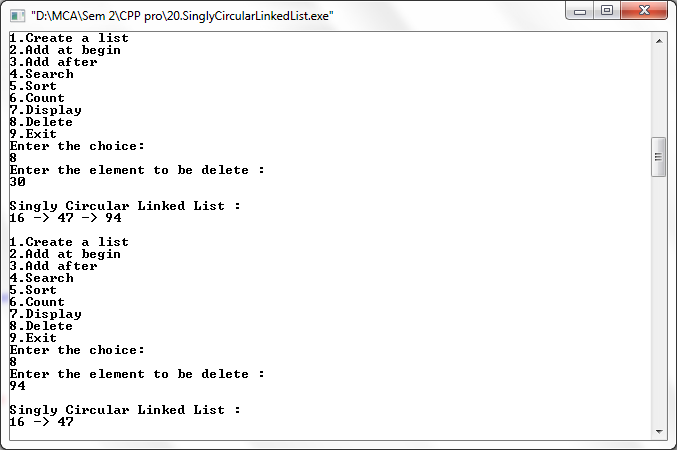
Count



Display



Delete



**Q. Write a C++ program to perform Circular Queue using Array.**

#include<iostream>

#include<stdio.h>

#include<conio.h>

using namespace std;

class queue

{

public:

int n=4;

int que[4],front=-1,rear=-1,ch,x;

int enqueue(int x)

{

if(rear==n-1)

{

cout<<"Queue full.\n";

}

else

if(front==-1 && rear==-1)

front=rear=0;

else

{

rear=(rear+1)%n;

}

que[rear]=x;

cout<<"Element is inserted.\n";

}

int dequeue()

{

if(front==rear)

{

cout<<"Queue Empty.\n\n";

}

else

{

x=que[front];

if(front==rear)

{

front=-1; rear=-1;

}

else

front=(front+1) % n;

cout<<x<<" is deleted from queue.\n";

}

}

int display()

{

if(front==rear)

{

cout<<"Queue Empty.\n\n";

}

else

{

cout<<"Circular Queue :";

for(int i=rear;i>=front;i--)

cout<<"\n"<<que[i];

cout<<"\n";

}

}

};

int main()

{

queue q;

int a[5],front=-1,rear=-1,ch,x;

while(ch!=0)

{

cout<<"\n1.Enqueue\n2.Dequeue\n3.Display\n";

cout<<"Enter your choice\n";

cin>>ch;

switch(ch)

{

case 1:

cout<<"Enter an element\n";

cin>>x;

q.enqueue(x);

break;

case 2:

q.dequeue();

break;

case 3:

q.display();

break;

default:

cout<<"You entered a wrong choice\n";

break;

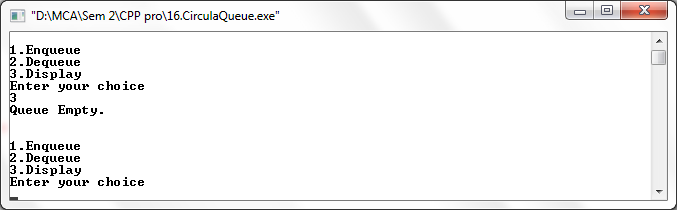
}

}

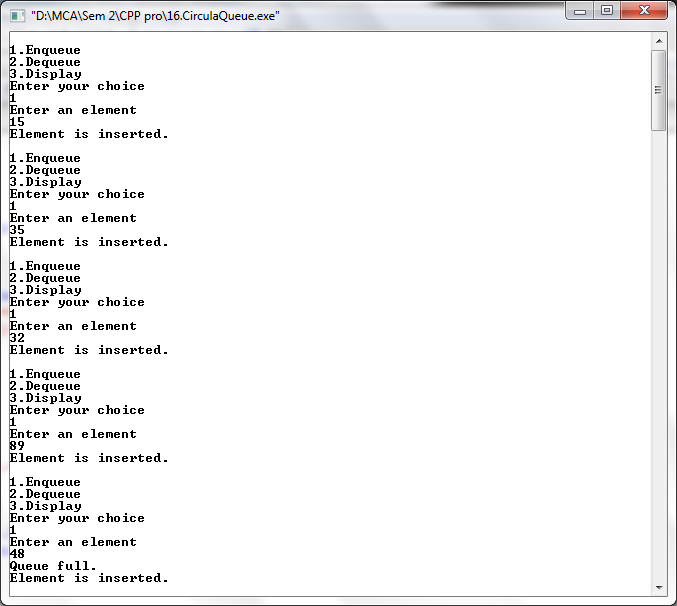
}

**Output –**

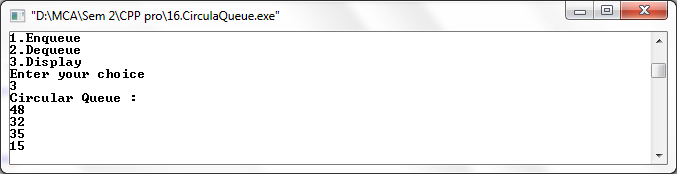
Queue empty



Enqueue



Display



Dequeue

