# ROLL NO: 15013570033

# NAME : PRANAV GURDITTA

# COURSE :BSC(HONS) COMPUTER SCIENCE

DATA STRUCTURES LAB PRACTICALS

# 1. Write a program to search an element from a list. Give user the option to perform Linear or Binary search. Use Template functions.

#include<iostream.h>

#include<conio.h>

template<class T>

class search

{

T arr[100];

int n;

public:

int linear(T n);

int binary(T n);

void enter();

};

template<class T>

void search<T>::enter()

{

cout<<"Enter the number of elements you want to store"<<endl;

cin>>n;

cout<<"Enter the elements"<<endl;

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

cin>>arr[i];

}

template<class T>

int search<T>::linear(T no)

{

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

{

if(arr[i]==no)

return (i+1);

}

return 0;

}

template<class T>

int search<T>::binary(T no)

{

int mid=0;

int lb=0;

int ub=n-1;

while(lb<=ub)

{

mid=(lb+ub)/2;

if(arr[mid]==no)

return (mid+1);

else if(arr[mid]>no)

ub=mid-1;

else

lb=mid+1;

}

return 0;

}

void main()

{

clrscr();

search<int> o;

int l;

do

{

cout<<"Enter 1 to do linear search and 2 to do binary search"<<endl;

cin>>l;

int k,q;

switch(l)

{

case 1:

o.enter();

cout<<"Enter the element to be searched"<<endl;

cin>>k;

q=o.linear(k);

if(q!=0)

cout<<"Element found at position "<<q<<endl;

else

cout<<"Element not found"<<endl;

break;

case 2:

cout<<"The elements have to be inserted in ascending order only"<<endl;

o.enter();

cout<<"Enter the element to be searched"<<endl;

cin>>k;

q=o.binary(k);

if(q!=0)

cout<<"Element found at position "<<q<<endl;

else

cout<<"Element not found"<<endl;

}

cout<<"Press 0 to continue"<<endl;

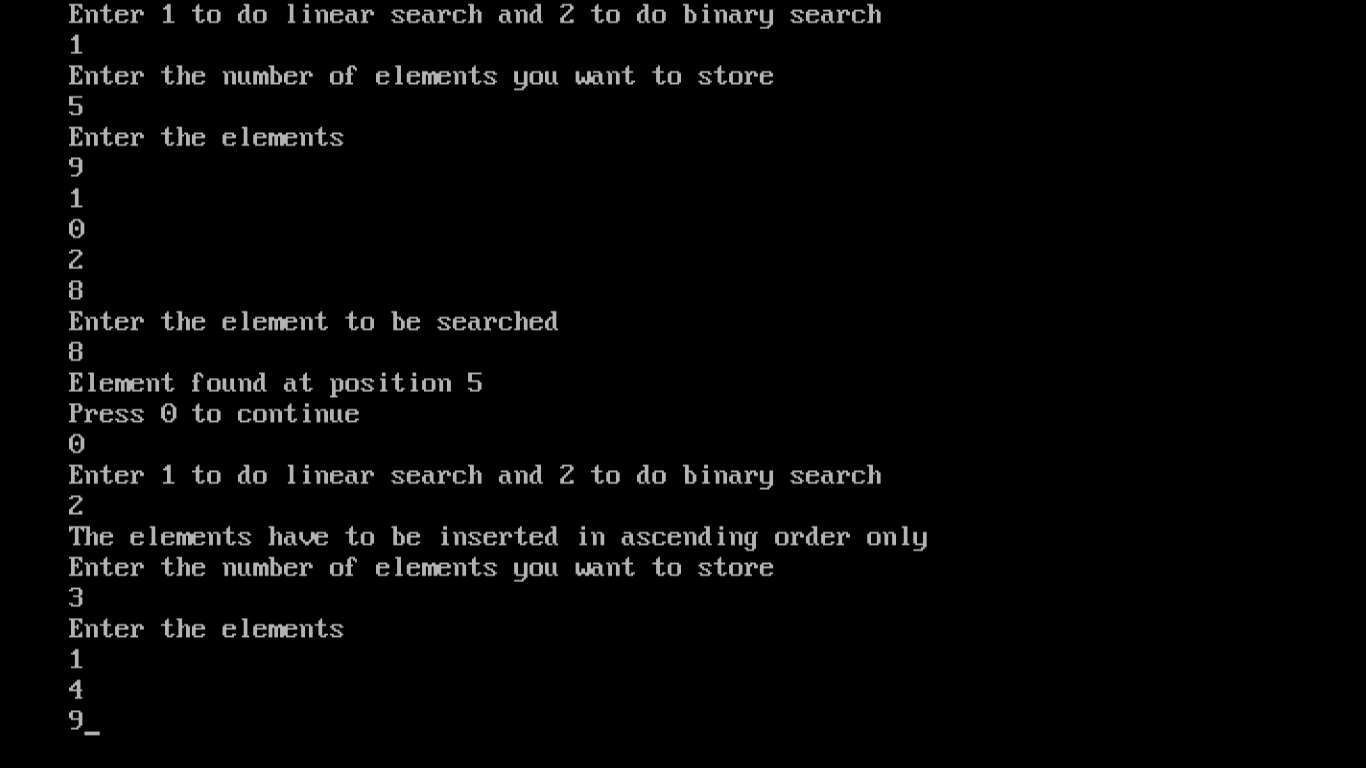
cin>>l;

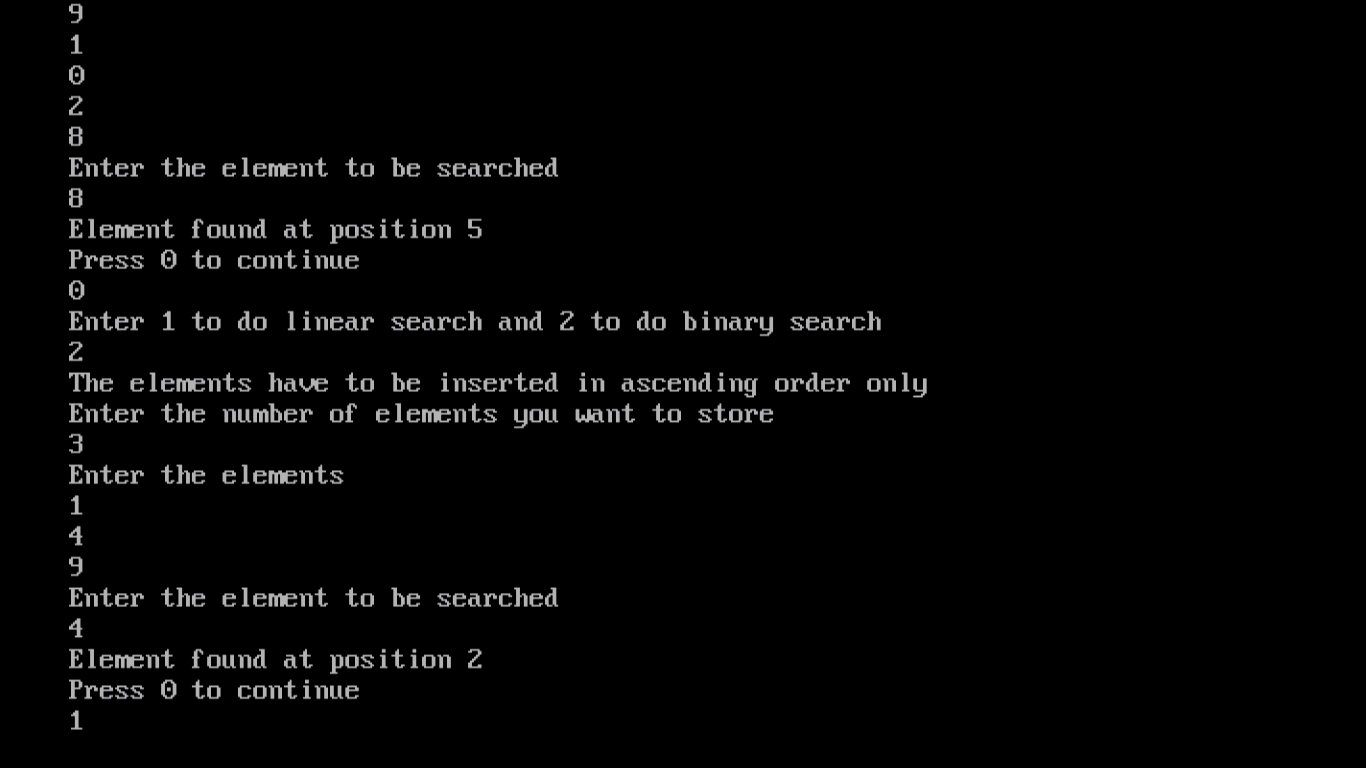
}while(l==0) ;

getch();

}

# OUTPUT





# 2. WAP using templates to sort a list of elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.

#include<iostream.h>

#include<conio.h>

template<class T>

class sort

{

T\* l; int n;

public:

void enter();

void display(int t);

T\* sort<T>:: bubblesort(T lis[],int n);

T\* selectionsort(T lis[],int n);

T\* insertionsort(T lis[],int n);

};

template<class T>

T\* sort<T>:: selectionsort(T lis[],int n)

{

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

{

T min=lis[i];

int k=i;

for(int j=i+1;j<n;j++)

{

if(lis[j]<min)

{

min=lis[j];

k=j;

}

}

lis[k]=lis[i];

lis[i]=min;

}

return lis;

}

template<class T>

T\* sort<T>:: insertionsort(T list[],int n){

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

{

int temp=list[i];

int j=i-1;

while(j>=0 && list[j]>temp)

{

list[j+1]=list[j];

j--;

}

list[j+1]=temp;

}

return(list);

}

template<class T>

T\* sort<T>:: bubblesort(T lis[],int n)

{

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

{

for(int j=0;j<n-i-1;j++)

{

if(lis[j]>lis[j+1])

{

int t=lis[j];

lis[j]=lis[j+1];

lis[j+1]=t;

}

}

}

return lis;

}

template<class T>

void sort<T>:: enter()

{

clrscr();

cout<<"Enter the no of elements to store:"<<endl;

cin>>n;

cout<<"Enter the elements"<<endl;

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

cin>>l[i];

}

template<class T>

void sort<T>::display(int k)

{

cout<<"The elements in the desired sorted order are"<<endl;

switch(k)

{

case 1:

l=bubblesort(l,n);

break;

case 2:

l=selectionsort(l,n);

break;

case 3:

l=insertionsort(l,n);

break;

}

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

cout<<l[i]<<endl;

}

void main()

{

int t;

sort<int> o;

o. enter();

cout<<"Enter 1 to do bubble sort , 2 to do selection sort"<<endl<<" and 3 to do insertion sort"<<endl;

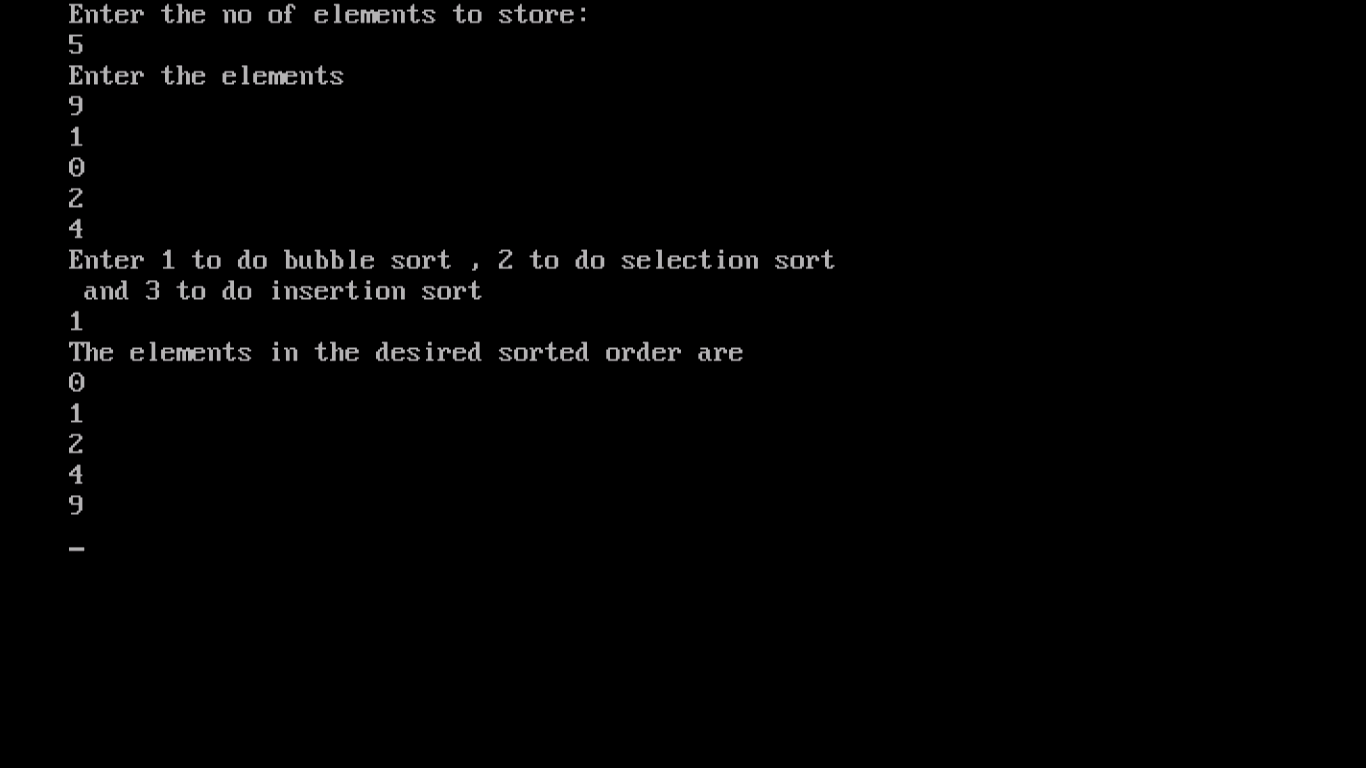
cin>>t;

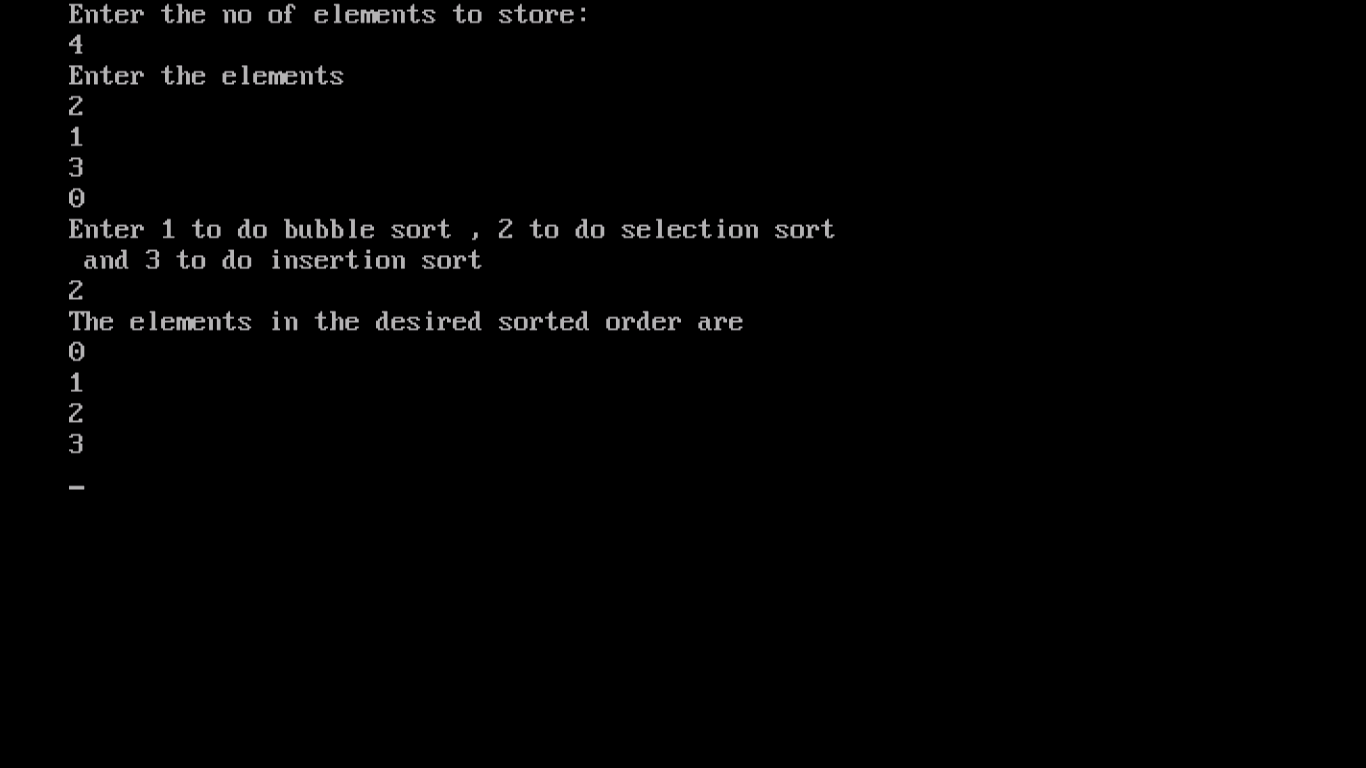
o. display(t);

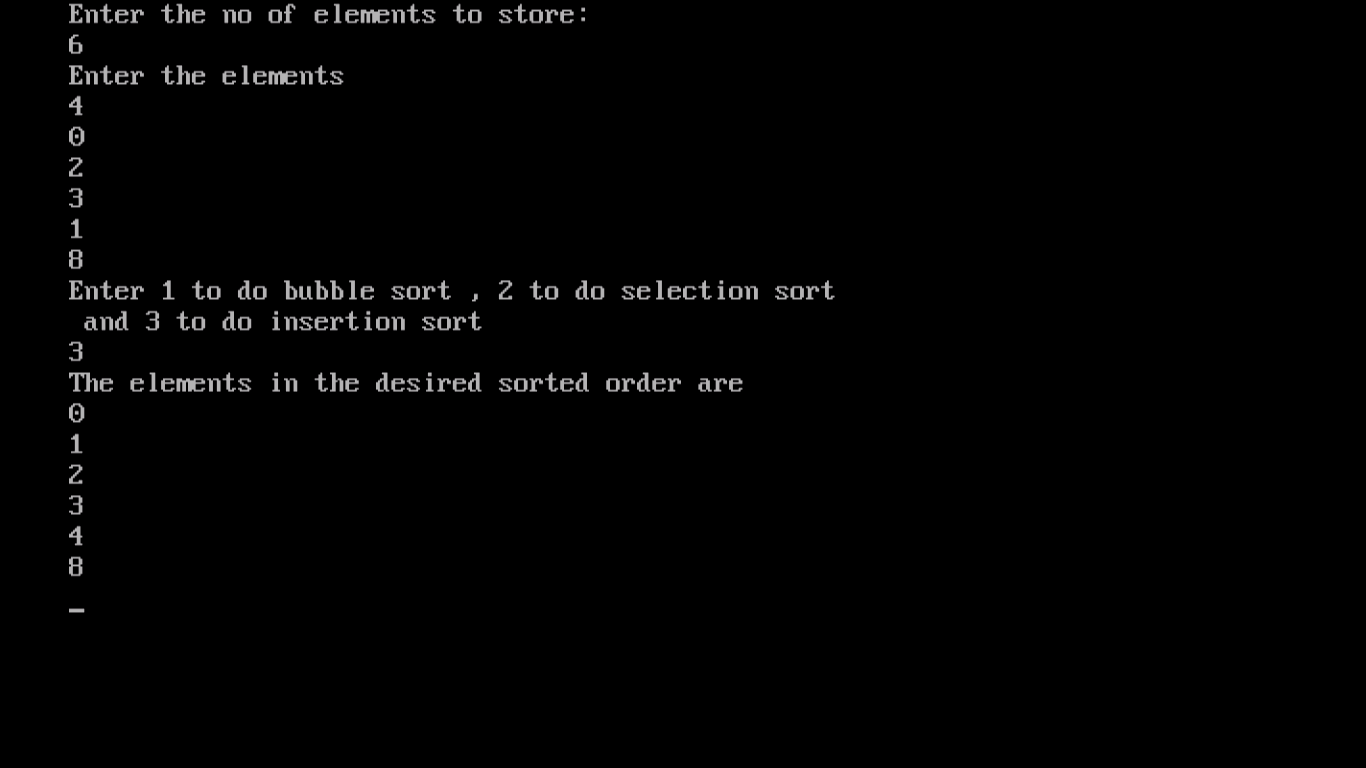
getch();

}

# OUTPUT







# 3. Implement Linked List using templates. Include functions for insertion, deletion and search of a number, reverse the list and concatenate two linked lists (include a function and also overload operator +).

#include<iostream.h>

#include<conio.h>

template<class T>

class node

{

public:

node \*next;

T data;

};

template<class T>

class sll

{

public:

node<T>\* h;

node<T>\* end;

sll()

{

end=h=NULL;

}

node<T>\* insert();

node<T> \* deleted();

void display(node<T> \*);

node<T>\* reverse();

int search(node<T> \*,T d);

void concat(sll<T> o);

sll operator+(sll<T> o);

};

template<class T>

sll<T> sll<T>:: operator+(sll<T> p)

{

int n1=0;

cout<<"Enter the second list to concatenate"<<endl;

while(n1==0)

{

cout<<"1 to insert, 2 to delete, 3 to display"<<endl;

cin>>n1;

cout<<endl;

switch(n1)

{

case 1:

p.h=p.insert();

break;

case 2:

p.deleted();

break;

case 3:

p.display(p.h);

break;

case 4:

int f;

cout<<"Enter the element to be searched "<<endl;

cin>>f;

if(p.search(p.h,f)==1)

cout<<"Element is there in the list."<<endl;

else

cout<<"Element is not there in the list."<<endl;

break;

case 5:

p.h=p.reverse();

break;

}

cout<<"enter 0 to continue"<<endl;

cin>>n1;

}

return p;

}

template<class T>

node<T>\* sll<T>::insert()

{

int tail=0;

int k1;

T d;

cout<<"Enter 1 to insert at front , 2 to insert at end , 3 to insert before a specific element"<<endl;

cin>>k1;

cout<<"Enter the item to be inserted"<<endl;

cin>>d;

node<T> \*item=NULL;

item=new node<T>;

item->next=NULL;

item->data=d;

if(h==NULL)

{

h=end=item;

tail++;

}

else

{

switch(k1)

{

case 1:

item->next=h;

h=item;

break;

case 2:

end->next=item;

end=item;

break;

case 3:

T t;

cout<<"Enter the element before which insertion has to take place"<<endl;

cin>>t;

if(search(h,t)==1)

{

{

if(t==h->data)

{

item->next=h;

h=item;

}

else

{

node<T> \*r;

node<T> \* k=h;

while ((k->data!=t) && (k!=NULL))

{

r=k;

k=k->next;

}

r->next=item;

item->next=k;

}

}

}

else

cout<<"Element before which insertion is to be done not found"<<endl;

break;

}

}

return h;

}

template<class T>

node<T>\* sll<T>::deleted()

{

T k;

if(h==NULL)

{

cout<<"Cannot delete as list is empty"<<endl;

return NULL;

}

else

{

cout<<"Enter the element to be deleted"<<endl;

cin>>k;

if(search(h,k)==1)

{

node<T> \*kh=NULL;

if(h->data==k)

{

if(h==end)

{

h=end=NULL;

return NULL;

}

else

{

h->data=h->next->data;

kh=h->next;

h->next=h->next->next;

delete kh;

}

return h;

}

else

{

node<T> \*r=NULL;

node<T> \*kh=NULL;

for(kh=h;(kh!=NULL);kh=kh->next)

{

if(kh->data==k)

{

break;

}

r=kh;

}

r->next=kh->next;

delete kh;

}

}

else

{

cout<<"NO such element exists in the list to be deleted"<<endl;

}

}

return h;

}

template<class T>

void sll<T>::display(node<T> \* h)

{

if(h!=NULL)

{

cout<<"DATA IS "<<endl;

while(h!=NULL)

{

cout<<h->data<<endl;

h=h->next;

}

}

}

template<class T>

int sll<T>::search(node<T>\* h,T g)

{

while(h!=NULL)

{

if(h->data==g)

{

return 1;

}

h=h->next;

}

return 0;

}

template<class T>

node<T>\* sll<T>::reverse()

{

node<T>\* current=h;

node<T>\* prev=NULL;

node<T>\* coming=NULL;

while(current!=NULL)

{

coming=current->next;

current->next=prev;

prev=current;

current=coming;

}

h=prev;

return h;

}

template<class T>

void sll<T>::concat(sll<T> o)

{

sll<T> p;

node<T> \*c=NULL;

int n1=0;

cout<<"Enter the second list to concatenate"<<endl;

while(n1==0)

{

cout<<"1 to insert, 2 to delete, 3 to display, 4 to search, 5 to reverse"<<endl;

cin>>n1;

cout<<endl;

switch(n1)

{

case 1:

c=p.insert();

break;

case 2:

p.deleted();

break;

case 3:

p.display(c);

break;

case 4:

int f;

cout<<"Enter the element to be searched "<<endl;

cin>>f;

if(p.search(c,f)==1)

cout<<"Element is there in the list."<<endl;

else

cout<<"Element is not there in the list."<<endl;

break;

case 5:

c=p.reverse();

break;

}

cout<<"enter 0 to continue"<<endl;

cin>>n1;

}

node<T> \*thirdhead=o.h;

node<T> \*third=thirdhead;

while(third->next!=NULL)

{

third=third->next;

}

third->next=p.h;

while(third->next!=NULL)

{

third=third->next;

}

cout<<"The concatenated list is "<<endl;

while(thirdhead!=NULL)

{

cout<<thirdhead->data<<endl;

thirdhead=thirdhead->next;

}

cout<<"Exiting the concatenated and second list"<<endl;

}

void main()

{

sll<int> p;

node<int> \*c=NULL;

sll<int> o;

clrscr();

int n=0;

node<int> \* l=NULL;

while(n==0)

{

cout<<"1 to insert, 2 to delete, 3 to display"<<endl;

cout<<"4 to search , 5 to reverse, 6 to concat normally"<<endl;

cout<<"7 to concat using operator overloading"<<endl;

cin>>n;

cout<<endl;

switch(n)

{

case 1:

l=o.insert();

break;

case 2:

o.deleted();

break;

case 3:

o.display(l);

break;

case 4:

int f;

cout<<"Enter the element to be searched "<<endl;

cin>>f;

if(o.search(l,f)==1)

cout<<"Element is there in the list."<<endl;

else

cout<<"Element is not there in the list."<<endl;

break;

case 5:

l=o.reverse();

break;

case 6:

o.concat(o);

break;

case 7:

sll<int> k;

k=o+p;

node<int>\* thirdhead=o.h;

node<int>\* third=thirdhead;

while(third->next!=NULL)

{

third=third->next;

}

third->next=k.h;

while(third->next!=NULL)

{

third=third->next;

}

cout<<"The concatenated list is "<<endl;

while(thirdhead!=NULL)

{

cout<<thirdhead->data<<endl;

thirdhead=thirdhead->next;

}

cout<<"Exiting the concatenated and second list"<<endl;

break;

default:

cout<<"Wrong option entered"<<endl;

}

o.end->next=NULL;

cout<<"Enter 0 to continue"<<endl;

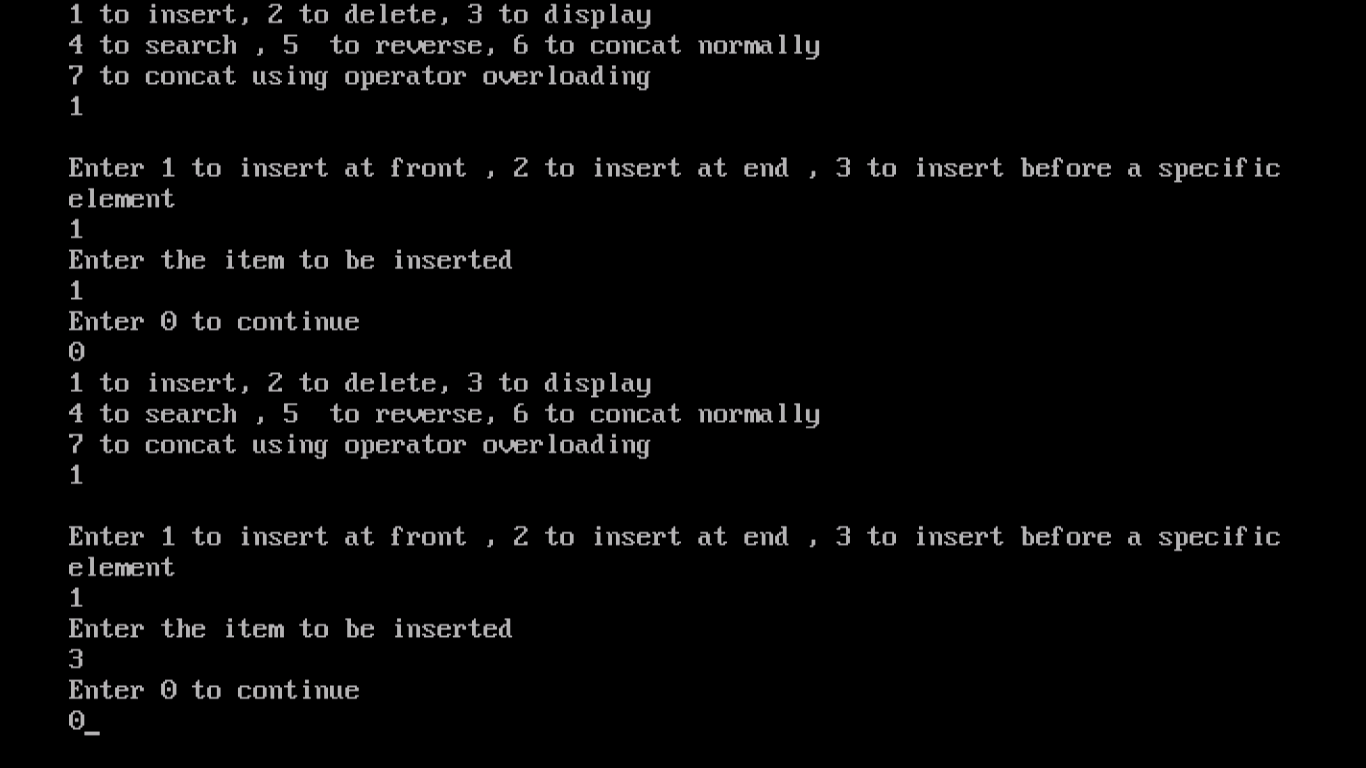
cin>>n;

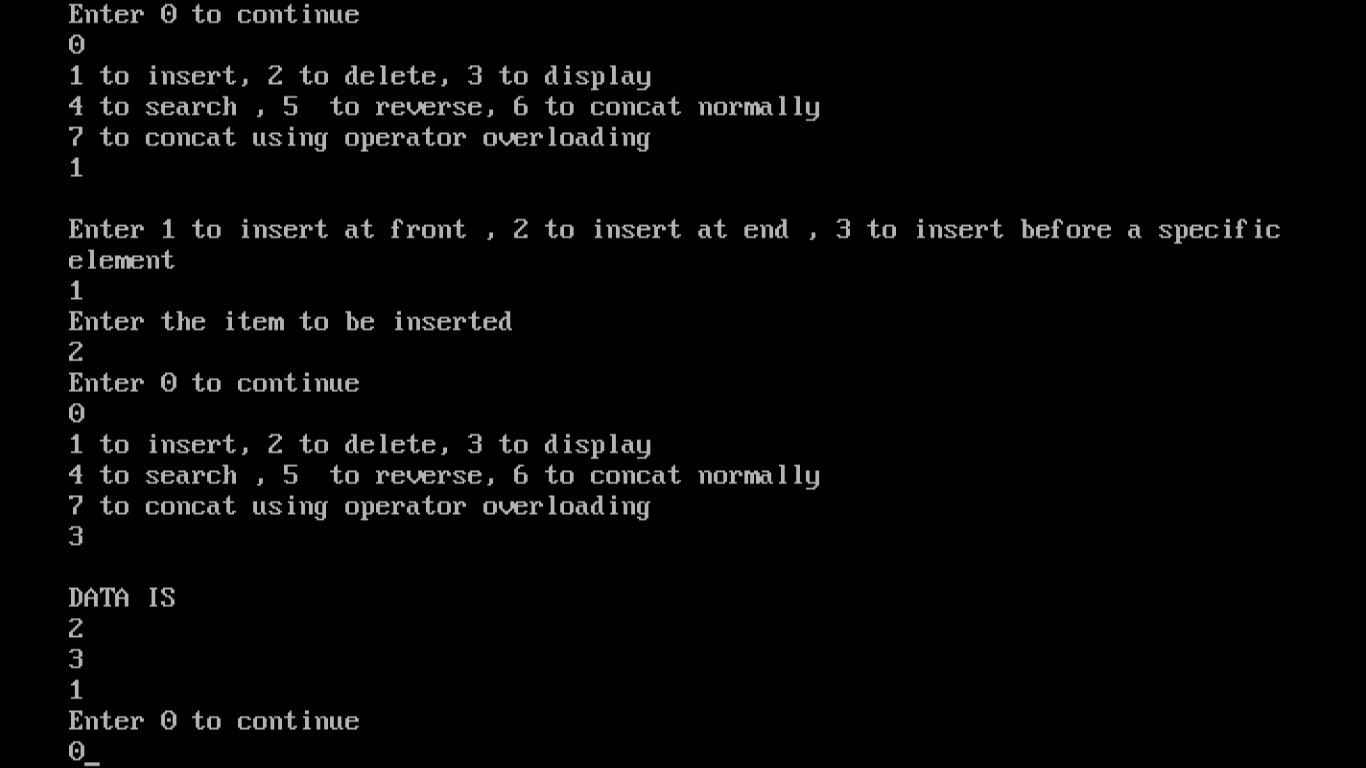
}

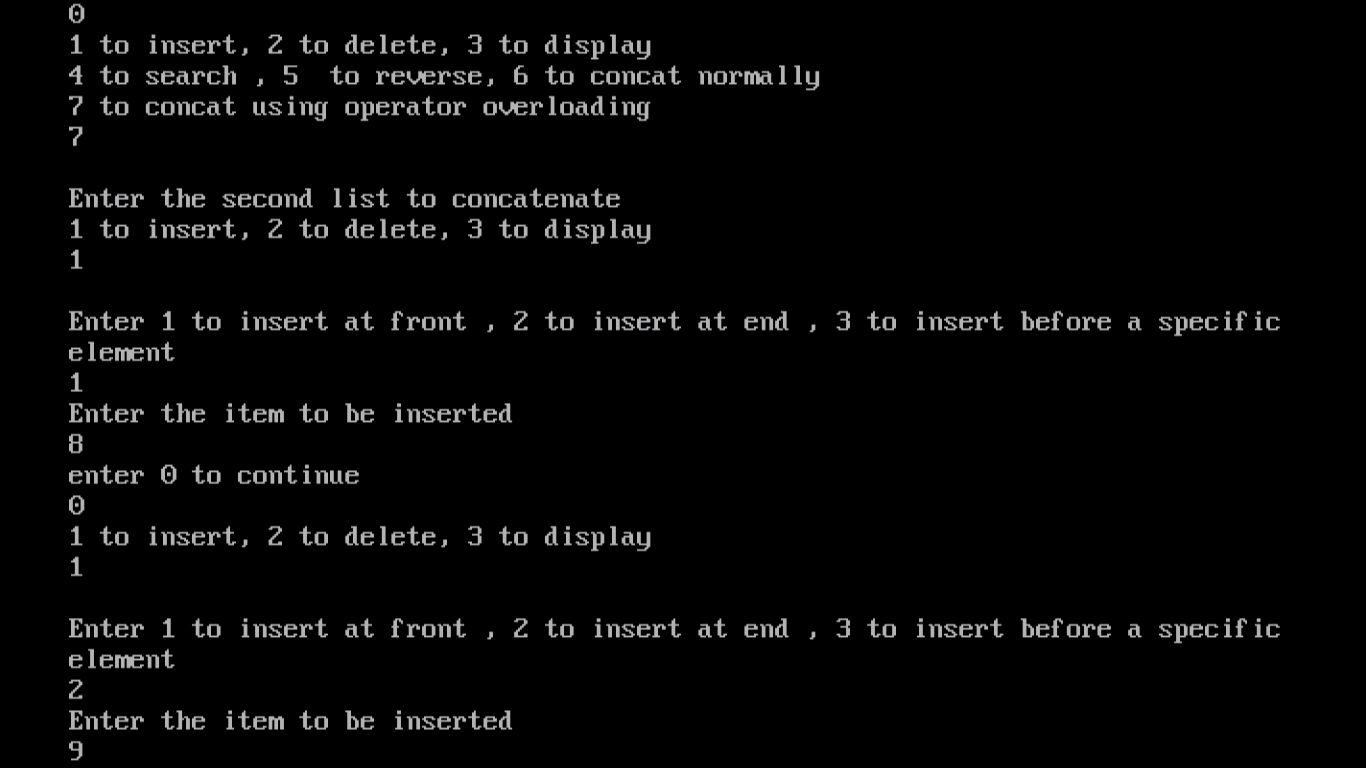
getch();

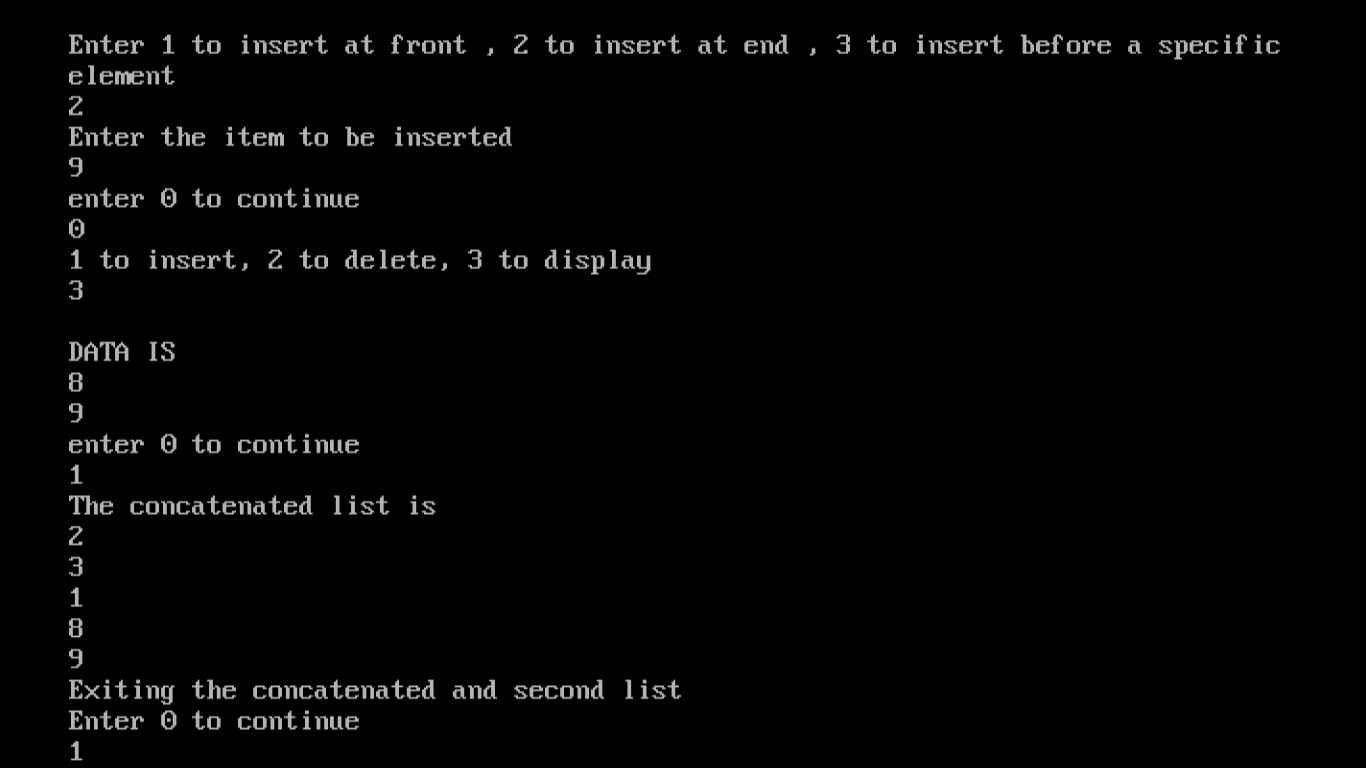
}

# OUTPUT









# 4. Implement Doubly Linked List using templates. Include functions for insertion, deletion and search of a number, reverse the list.

#include<iostream.h>

#include<conio.h>

struct node

{

int data;

node\* prev;

node\* next;

};

class dll

{

node\* head;

node\* tail;

public:

dll()

{

head=NULL;

tail=NULL;

}

node\* insertatend();

node\* insertatbeg();

node\* insertanywhere();

void display();

node\* deleteatbeg();

node\* deleteatend();

node\* deleteatmiddle();

node\* searching(int n, node\*x );

void reverse();

};

void dll::reverse()

{

node\* r;

r=tail;

while (r!=NULL)

{

cout<<r->data<<" , ";

r=r->prev;

}

cout<<endl;

}

node\* dll::insertanywhere()

{

int l;

node \*item=new node;

cout<<"Enter the element to be inserted"<<endl;

cin>>l;

int nl;

cout<<"Enter the data before which insertion is to be performed"<<endl;

cin>>nl;

if(head==NULL )

{

cout<<"List is empty hence inserting at front"<<endl;

head=tail=item;

head->prev=NULL;

tail->next=NULL;

}

else

{

node\* t2=NULL;

node \*t1=head;

while((t1->data!=nl)&&(t1!=NULL))

{

t2=t1;

t1=t1->next;

}

if(t1==NULL)

{

cout<<"Cannot insert as element before which insertion is to be done not found"<<endl;

return head;

}

else

{

item=new node;

item->data=l;

item->next=t1;

t1->prev=item;

t2->next=item;

item->prev=t2;

}

}

return head;

}

node\* dll::deleteatmiddle()

{

int l;

cout<<"Enter the element to be deleted"<<endl;

cin>>l;

if(head==NULL )

{

cout<<"List is empty hence cannot delete"<<endl;

}

else

{

node\* t2=NULL;

node \*t1=head;

while((t1->data!=l)&&(t1!=NULL))

{

t2=t1;

t1=t1->next;

}

if(t1==NULL)

{

cout<<"Cannot delete as element not found"<<endl;

return head;

}

else

{

t2->next=t1->next;

t1->next->prev=t2;

}

}

return head;

}

node\* dll::insertatbeg()

{

node \*item=new node;

cout<<"Enter the elements to be inserted"<<endl;

cin>>item->data;

if(head==NULL)

{

head=tail=item;

head->prev=NULL;

tail->next=NULL;

}

else

{

item->next=head;

head->prev=item;

head=item;

}

return head;

}

node\* dll::insertatend()

{

node \*item=new node;

cout<<"Enter the elements to be inserted"<<endl;

cin>>item->data;

if(head==NULL)

{

head=tail=item;

head->prev=NULL;

tail->next=NULL;

}

else

{

tail->next=item;

item->prev=tail;

tail=item;

}

return head;

}

node\* dll::searching(int n,node\* h)

{

node\* k=new node;

k=h;

while(k!=NULL)

{

if(k->data==n)

return k;

k=k->next;

}

return k;

}

void dll::display()

{

node\* y=NULL;

y=head;

cout<<"The list is "<<endl;

while(y!=NULL)

{

cout<<y->data<<endl;

y=y->next;

}

}

node\* dll::deleteatend()

{

if(head==NULL)

{

cout<<"The list is empty and there is nothing to delete"<<endl;

return NULL;

}

else

{

node\* t=tail;

tail=tail->prev;

tail->next=NULL;

delete t;

}

return head;

}

node\* dll::deleteatbeg()

{

if(head==NULL)

{

cout<<"The list is empty and there is nothing to delete"<<endl;

return NULL;

}

else

{

node\* t=head;

head=head->next;

head->next->prev=NULL;

delete t;

}

return head;

}

void main()

{

clrscr();

int k;

dll o;

node\* p=NULL;

do

{

int n ;

cout<<"Enter 1 to insert, 2 to display , 3 to search , 4 to delete , 5 to reverse"<<endl;

cin>>n;

switch(n)

{

case 1:

int t;

cout<<"Enter 1 to insert at beginning , 2 to insert anywhere , 3 to insert at end"<<endl;

cin>>t;

switch( t)

{

case 1:

p=o.insertatbeg();

break;

case 2:

p=o.insertanywhere();

break;

case 3:

p=o.insertatend();

break;

default:

cout<<"wrong choice"<<endl;

}

break;

case 2:

o.display();

break;

case 3:

cout<<"Enter no to search"<<endl;

node\* ok=new node;

cin>>ok->data;

ok=o.searching(ok->data,p);

if(ok!=NULL)

cout<<"FOUND"<<endl;

else

cout<<"NOT FOUND"<<endl;

break;

case 4:

cout<<"Enter 1 to delete at beginning , 2 to delete from middle , 3 to delete from end"<<endl;

cin>>k;

switch(k)

{

case 1:

p=o.deleteatbeg();

break;

case 2:

p=o.deleteatmiddle();

break;

case 3:

p=o.deleteatend();

break;

default:

cout<<"wrong choice"<<endl;

}

break;

case 5:

cout<<"The reversed list is "<<endl;

o.reverse();

break;

default:

cout<<"wrong choice"<<endl;

}

cout<<endl<<"enter 0 to continue";

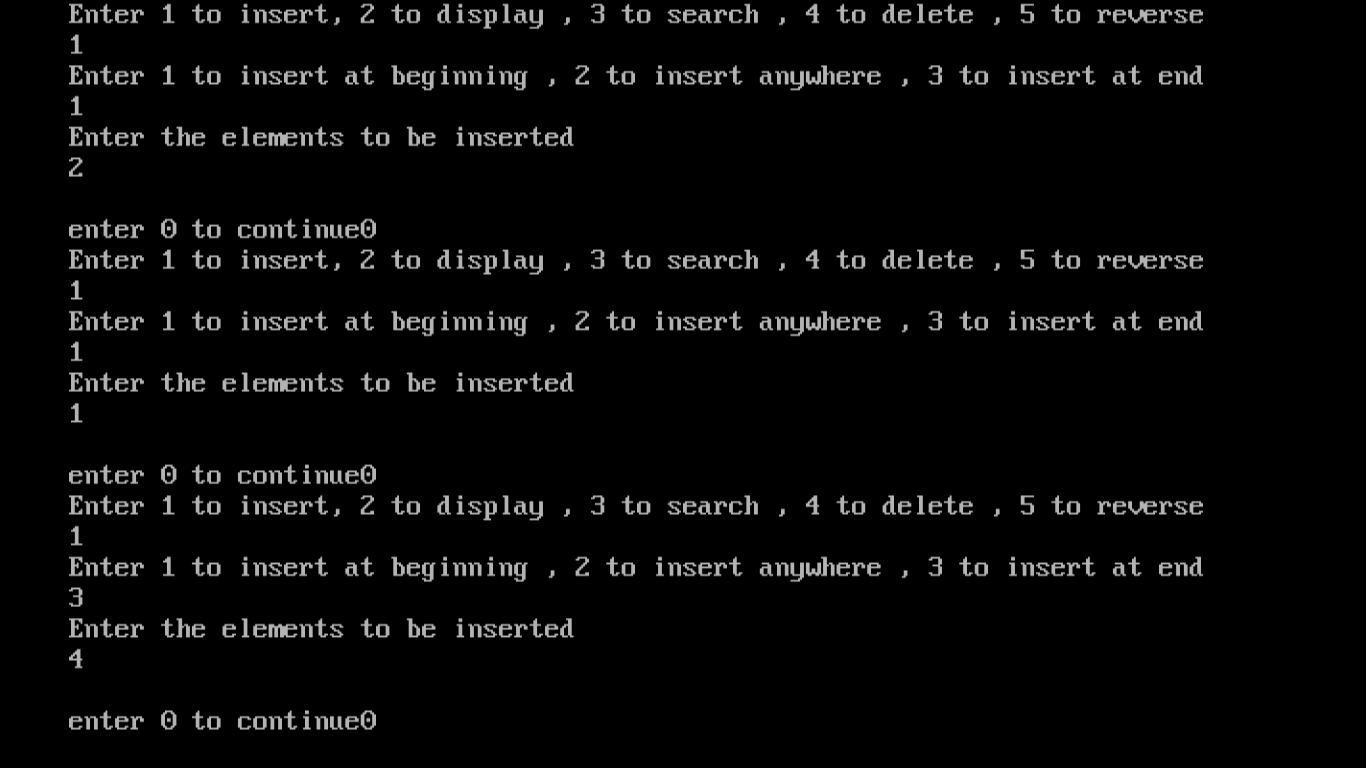
cin>>k;

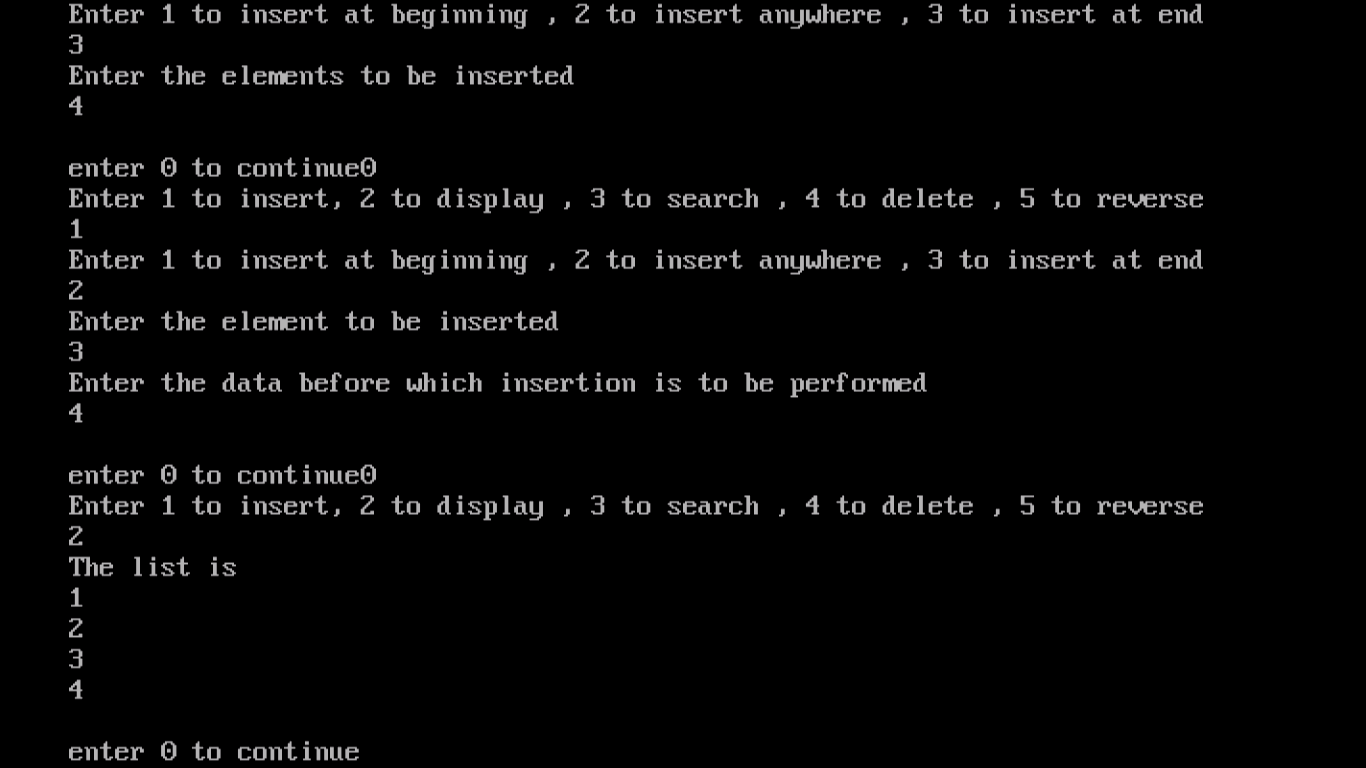
}while(k==0);

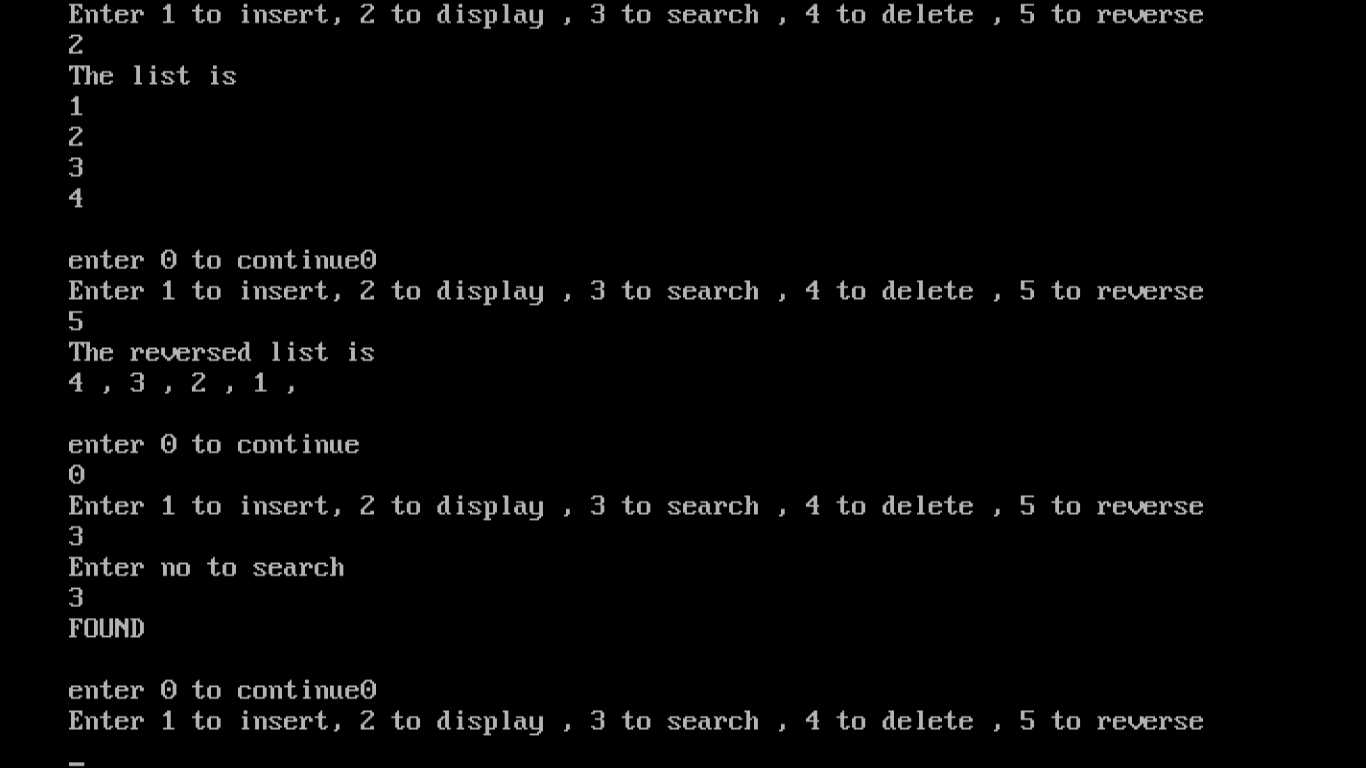
getch();

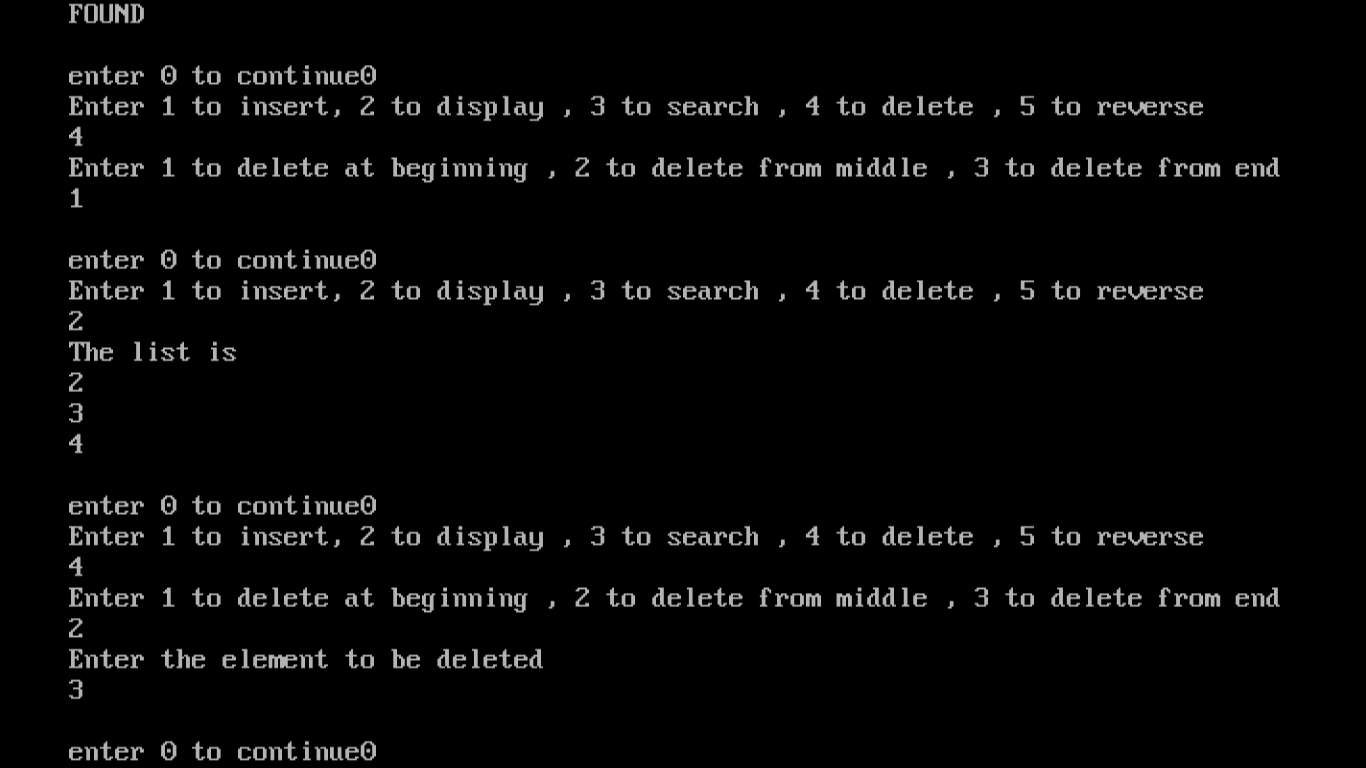
}

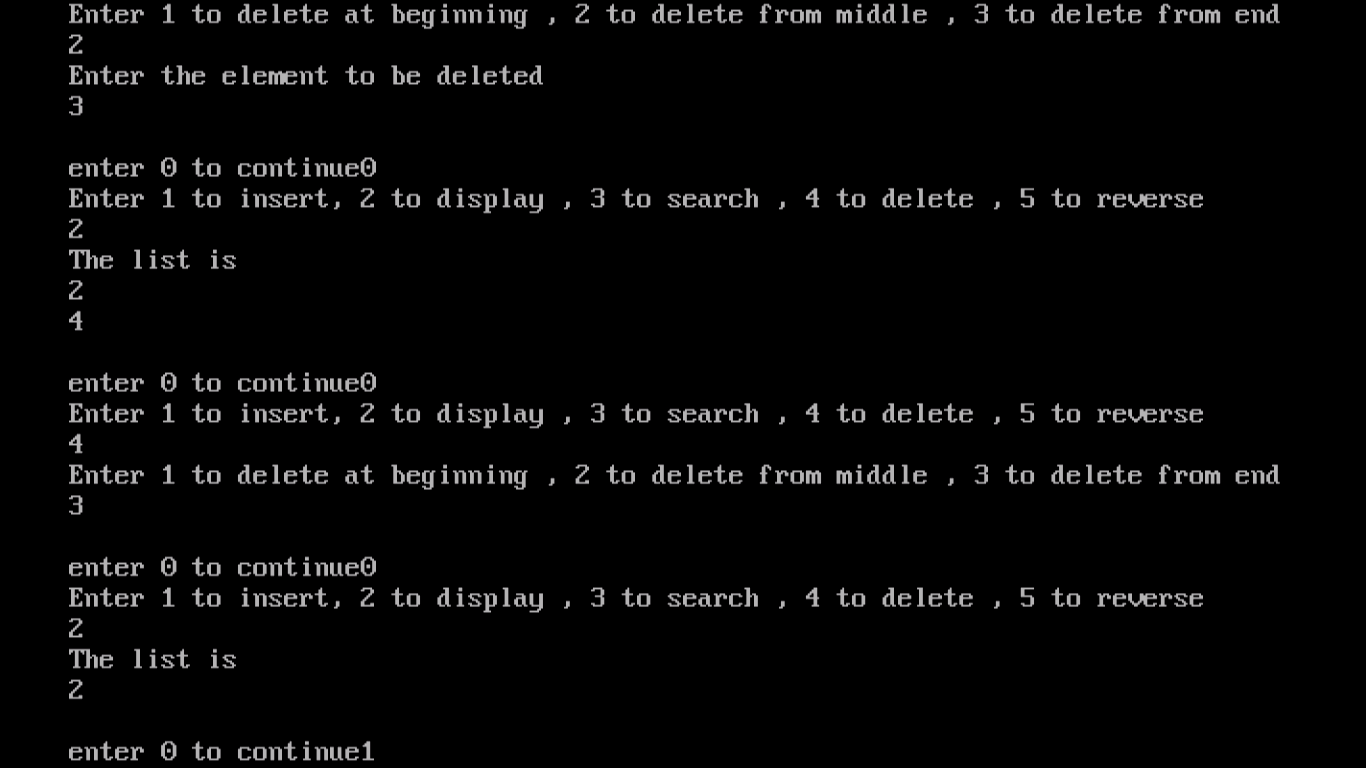
# OUTPUT











## 5. Implement Circular Linked List using templates. Include functions for insertion, deletion and search of a number, reverse the list.

#include<iostream.h>

#include<conio.h>

template<class T>

class node

{

public:

node \*next;

T data;

};

template<class T>

class csll

{

public:

node<T>\* h;

node<T>\* end;

csll()

{

end=h=NULL;

}

node<T>\* insert();

node<T> \* deleted();

void display(node<T> \*);

node<T>\* reverse();

int search(node<T> \*,T d);

};

template<class T>

node<T>\* csll<T>::insert()

{

int tail=0;

int k1;

T d;

cout<<"Enter 1 to insert at front , 2 to insert at end , 3 to insert before a specific element"<<endl;

cin>>k1;

cout<<"Enter the item to be inserted"<<endl;

cin>>d;

node<T> \*item=NULL;

item=new node<T>;

item->next=NULL;

item->data=d;

if(h==NULL)

{

h=end=item;

tail++;

}

else

{

switch(k1)

{

case 1:

item->next=h;

h=item;

end->next=h;

break;

case 2:

end->next=NULL;

end->next=item;

end=item;

item->next=h;

break;

case 3:

T t;

cout<<"Enter the element before which insertion has to take place"<<endl;

cin>>t;

if(search(h,t)==1)

{

{

if(t==h->data)

{

item->next=h;

h=item;

end->next=h;

}

else

{

node<T> \*r;

node<T> \* k=h;

while ((k->data!=t) && (k->next!=h))

{

r=k;

k=k->next;

}

r->next=item;

item->next=k;

}

}

}

else

cout<<"Element before which insertion is to be done not found"<<endl;

break;

}

}

return h;

}

template<class T>

node<T>\* csll<T>::deleted()

{

T k;

if(h==NULL)

{

cout<<"Cannot delete as list is empty"<<endl;

return NULL;

}

else

{

cout<<"Enter the element to be deleted"<<endl;

cin>>k;

if(search(h,k)==1)

{

node<T> \*kh=NULL;

if(h->data==k)

{

if(h==end)

{

h=end=NULL;

return NULL;

}

else

{

h->data=h->next->data;

kh=h->next;

h->next=h->next->next;

end->next=h;

delete kh;

}

return h;

}

else

{

node<T> \*r=NULL;

node<T> \*kh=NULL;

for(kh=h;(kh->next!=h);kh=kh->next)

{

if(kh->data==k)

{

break;

}

r=kh;

}

r->next=kh->next;

delete kh;

}

}

else

{

cout<<"NO such element exists in the list to be deleted"<<endl;

}

}

return h;

}

template<class T>

void csll<T>::display(node<T> \* h)

{

node<T> \*t=NULL;

t=h;

if(t)

{

cout<<"DATA IS "<<endl;

while(t)

{

cout<<t->data<<endl;

t=t->next;

if(t==h)

return;

}

}

}

template<class T>

int csll<T>::search(node<T>\* h,T g)

{

node<T> \*y=NULL;

y=h;

while(y->next!=h)

{

if(y->data==g)

{

return 1;

}

y=y->next;

}

return 0;

}

template<class T>

node<T>\* csll<T>::reverse()

{

node<T>\* current=h;

node<T>\* result=NULL;

node<T>\* next;

while(current!=NULL)

{

next=current->next;

current->next=result;

result=current;

current=next;

if(current==h)

break;

}

h->next=result;

h=result;

return h;

}

void main()

{

csll<int> o;

clrscr();

int n=0;

node<int> \* l=NULL;

while(n==0)

{

cout<<"1 to insert, 2 to delete, 3 to display,4 to search , 5 to reverse "<<endl;

cin>>n;

cout<<endl;

switch(n)

{

case 1:

l=o.insert();

break;

case 2:

o.deleted();

break;

case 3:

o.display(l);

break;

case 4:

int f;

cout<<"Enter the element to be searched "<<endl;

cin>>f;

if(o.search(l,f)==1)

cout<<"Element is there in the list."<<endl;

else

cout<<"Element is not there in the list."<<endl;

break;

case 5:

l=o.reverse();

break;

default:

cout<<"Wrong option entered"<<endl;

}

cout<<"Enter 0 to continue"<<endl;

cin>>n;

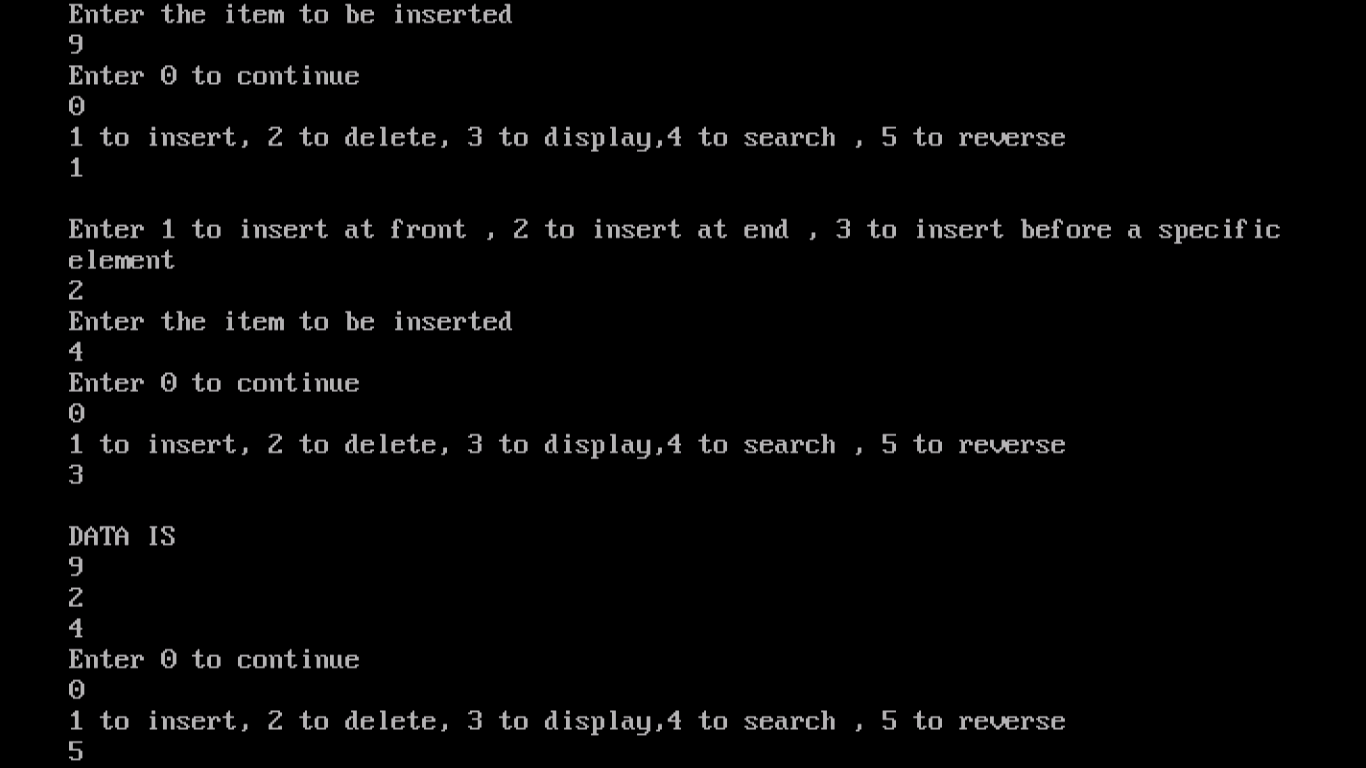
}

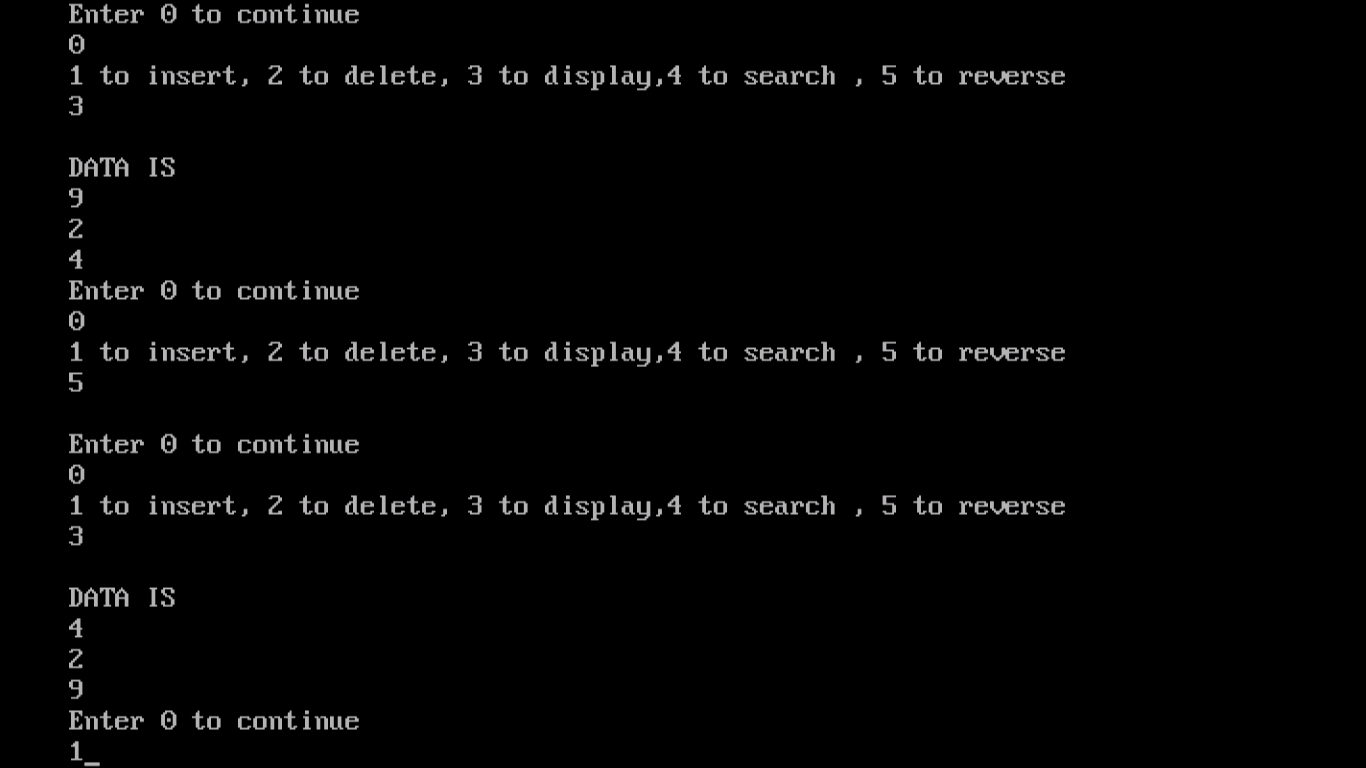
getch();

}

# OUTPUT

## 





## 6. Perform Stack operations using Linked List implementation.

#include<iostream.h>

#include<conio.h>

template<class T>

class node

{

public:

node\* next;

T data;

} ;

template<class T>

class stack

{

public:

node<T>\* top;

stack()

{

top=NULL;

}

void push();

T pop();

void display();

};

template<class T>

void stack<T>::push()

{

T d;

node<T>\* item=NULL;

item=new node<T>;

cout<<"Enter the element to insert"<<endl;

cin>>d;

item->data=d;

if(top==NULL)

{

top=item;

}

else

{

item->next=top;

top=item;

}

}

template<class T>

void stack<T>::display()

{

cout<<"The stack is "<<endl;

for(node<T>\* y=top;y!=NULL;y=y->next)

cout<<y->data<<endl;

}

template<class T>

T stack<T>:: pop()

{

if(top==NULL)

return NULL;

else

{

T i;

i=top->data;

top=top->next;

return i;

}

}

void main()

{int k=0;

clrscr();

stack<int> o;

while(k==0)

{

cout<<" 1 for push , 2 for pop, 3 to display"<<endl;

cin>>k;

switch(k)

{

case 1:

o.push();

break;

case 2:

cout<<"The popped out element is"<<o.pop()<<endl;

break;

case 3:

o.display();

break;

}

cout<<"Enter 0 to continue"<<endl;

cin>>k;

}

getch();

}

## OUTPUT

## 

## 7. Perform Stack operations using Array implementation. Use Templates.

#include<iostream.h>

#include<conio.h>

template<class T>

class stack

{

int top;

T arr[100];

public:

int n;

stack()

{

top=-1;

}

void push(int n);

T pop();

void display();

};

template<class T>

void stack<T>::push(int n)

{

T k;

cout<<"Enter the element to be pushed"<<endl;

cin>>k;

if(top==n-1)

cout<<"Stack Overflow"<<endl;

else

{

top++;

arr[top]=k;

}

}

template<class T>

T stack<T>::pop()

{

if(top==-1)

{

cout<<"stack underflow"<<endl;

return -9999;

}

else

{

T r=arr[top];

top--;

return r;

}

}

template<class T>

void stack<T>::display()

{

cout<<"The stack is "<<endl;

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

{

cout<<arr[i]<<endl;

}

}

void main()

{

clrscr();

stack<int> o;

int k=0;

cout<<"Enter the maximum no of elements to be stored in the stack"<<endl;

cin>>o.n;

while(k==0)

{

cout<<" 1 for push , 2 for pop, 3 to display"<<endl;

cin>>k;

switch(k)

{

case 1:

o.push(o.n);

break;

case 2:

cout<<"The popped out element is"<<o.pop()<<endl;

break;

case 3:

o.display();

break;

}

cout<<"Enter 0 to continue"<<endl;

cin>>k;

}

getch();

}

## OUTPUT

## 

## 8. Perform Queues operations using Circular Array implementation. Use Templates.

#include<iostream.h>

#include<conio.h>

template<class T>

class queue

{

T arr[100];

public:

queue()

{

f=r=-1;

}

int n;

int f;

int r;

void push();

T pop();

void display();

};

template<class T>

void queue<T>::push()

{

T ele;

cout<<"Enter the element to be entered"<<endl;

cin>>ele;

if(((f== 0)&&(r==n-1)) || (f==r+1))

{

cout<<"QUEUE OVERFLOW"<<endl;

}

else

{

if((f==-1) &&(r==-1))

f=r=0;

else if(r==n-1)

r=0;

else

r++;

arr[r]=ele;

}

}

template<class T>

T queue<T>::pop()

{

T val;

if((f==-1)&& (r==-1))

{

cout<<"QUEUE UNDERFLOW , no elements to be deleted"<<endl;

}

else

{

val=arr[f];

if(f==r)

{

f=r=-1;

}

else if(f==n-1)

{

f=0;

}

else

{

f=f+1; }

}

return (val);

}

template<class T>

void queue<T>::display()

{

cout<<"The elements of the queue are "<<endl;

if(f<=r)

{

for(int i=f;i<=r;i++)

{

cout<<arr[i]<<endl;

}

}

else

{

for(int i=f;i<n;i++)

cout<<arr[i]<<endl;

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

cout<<arr[i]<<endl;

}

}

void main()

{

clrscr();

queue<int> o;

cout<<"Enter the number of elements"<<endl;

cin>>o.n;

int k=0;

while(k==0)

{

cout<<" 1 for push , 2 for pop, 3 to display"<<endl;

cin>>k;

switch(k)

{

case 1:

o.push();

break;

case 2:

cout<<"The popped out element is"<<o.pop()<<endl;

break;

case 3:

o.display();

break;

}

cout<<"Enter 0 to continue"<<endl;

cin>>k;

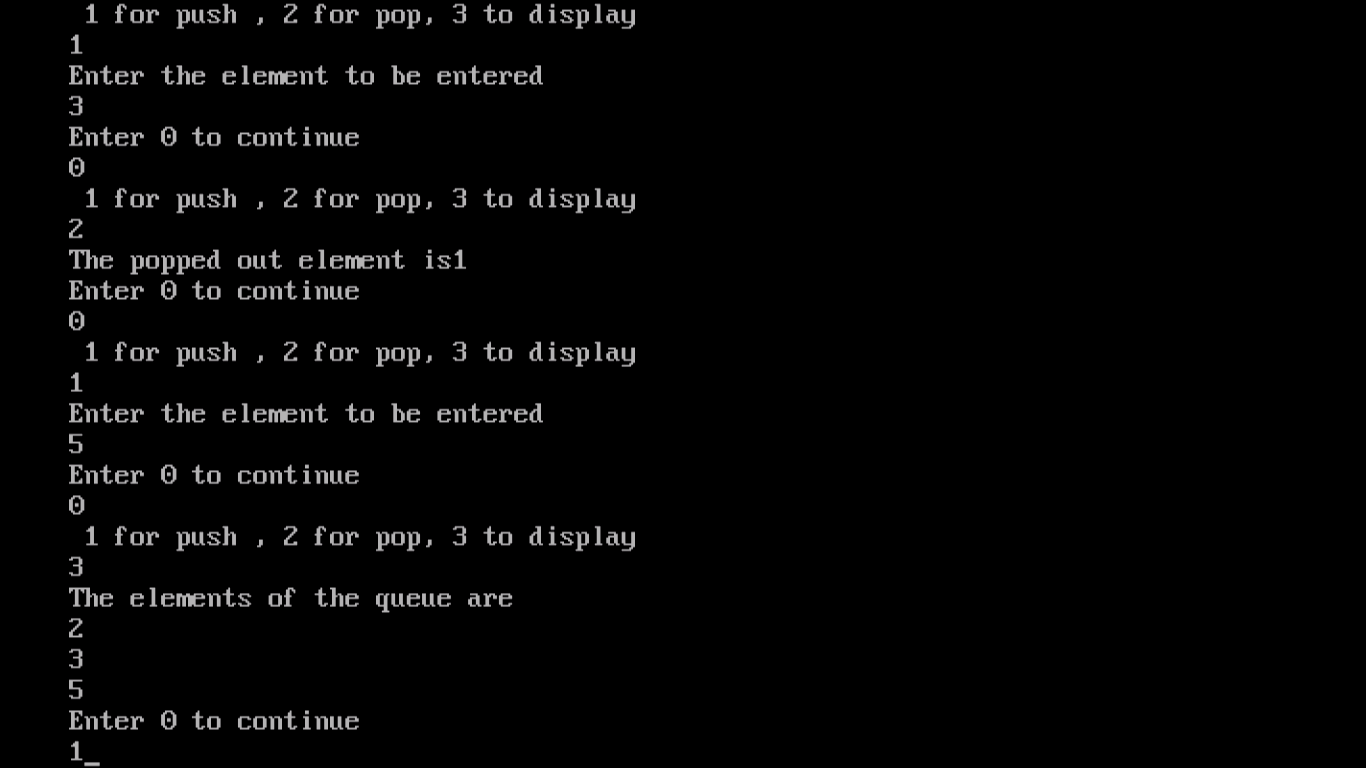
}

getch();

}

## OUTPUT

## 



## 9. Create and perform different operations on Double-ended Queues using Linked List implementation.

#include<iostream.h>

#include<conio.h>

class node

{

public:

int data;

node\* next;

node\* prev;

};

class deque

{

node\* start;

node\* end;

public:

deque()

{

start=end=NULL;

}

int remove\_front();

void add\_front();

void add\_rear();

int remove\_rear();

void display();

int getfront();

int getrear();

} ;

void deque::add\_front()

{

int k;

cout<<"Enter the element to be added"<<endl;

cin>>k;

node\* item;

item=new node;

item->data=k;

item->prev=NULL;

item->next=NULL;

if(start==NULL)

{

start=end=item;

}

else

{

item->next=start;

start->prev=item;

start=item;

}

}

void deque::display()

{

node\* t=NULL;

t=start;

cout<<"The data is "<<endl;

while(t!=NULL)

{

cout<<t->data<<endl;

t=t->next;

}

}

void deque::add\_rear()

{

int k;

cout<<"Enter the element to be added"<<endl;

cin>>k;

node\* item;

item=new node;

item->data=k;

item->prev=NULL;

item->next=NULL;

if(start==NULL)

{

start=end=item;

}

else

{

end->next=item;

item->prev=end;

end=item;

}

}

int deque::remove\_front()

{ int u;

if(start==NULL)

cout<<"DEQUEUE EMPTY , hence cannot remove front."<<endl;

else if(start==end)

{ u=start->data;

start=end=NULL;

}

else

{ u=start->data;

node\* temp=start;

start=start->next;

start->prev=NULL;

delete temp;

}

return u;

}

int deque::remove\_rear()

{

int u;

if(start==NULL)

cout<<"DEQUEUE EMPTY , hence cannot remove rear."<<endl;

else if(start==end)

{

u=end->data;

start=end=NULL;

}

else

{

u=end->data;

node\* temp=end;

end=end->prev;

end->next=NULL;

delete temp;

}

return u;

}

int deque::getfront()

{

return(start->data);

}

int deque::getrear()

{

return(end->data);

}

void main()

{

deque o;

int k=0;

clrscr();

while(k==0)

{

cout<<" 1 to add at front , 2 to add at rear , 3 to display"<<endl;

cout<<"4 to remove from front , 5 to remove from rear end"<<endl;

cout<<"6 to know the front element , 7 to know the rear element"<<endl;

cin>>k;

switch(k)

{

case 1:

o.add\_front();

break;

case 2:

o.add\_rear();

break;

case 3:

o.display();

break;

case 4:

cout<<"The removed element is "<<o.remove\_front()<<endl;

break;

case 5:

cout<<"The removed element is "<<o.remove\_rear()<<endl;

break;

case 6:

cout<<"The element at the front is "<<o.getfront()<<endl;

break;

case 7:

cout<<"The element at the end is "<< o.getrear()<<endl;

break;

default:

cout<<"Wrong option entered"<<endl;

}

cout<<"Enter 0 to continue"<<endl;

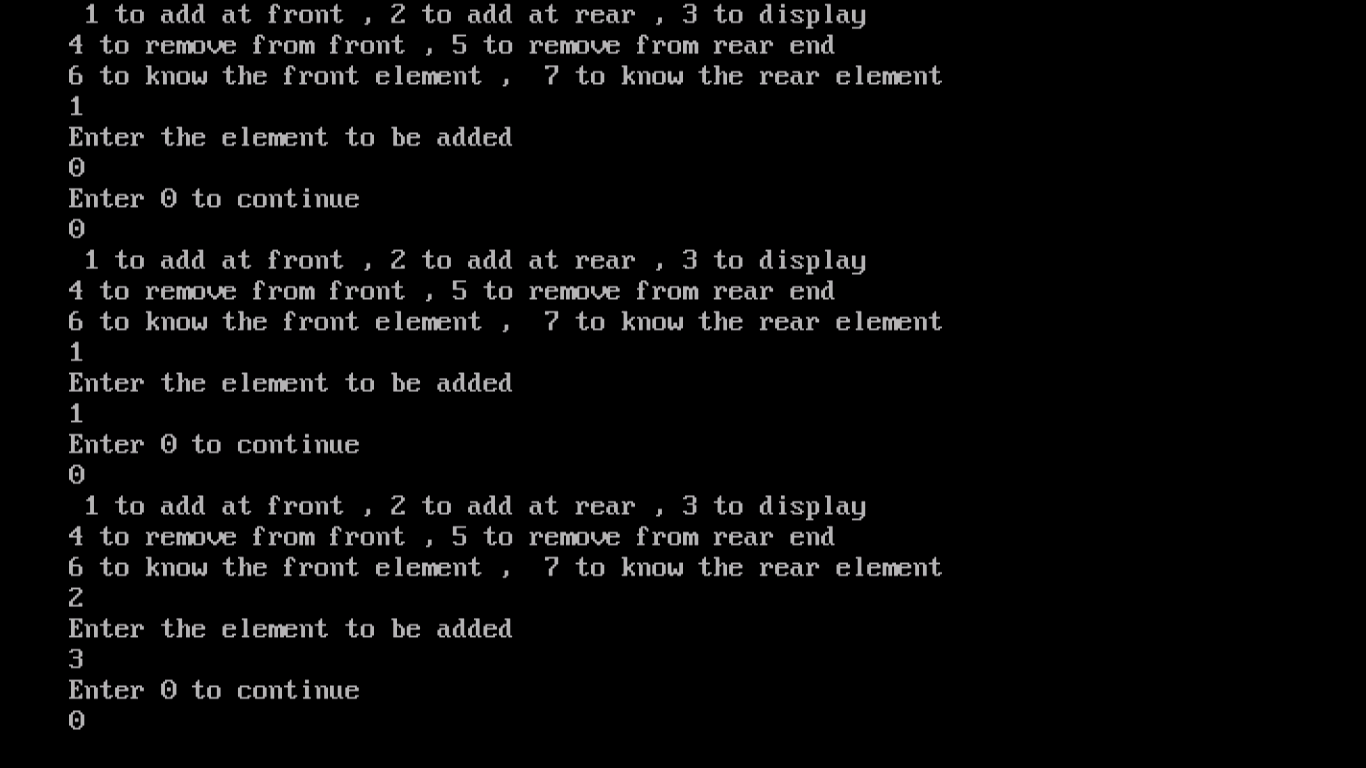
cin>>k;

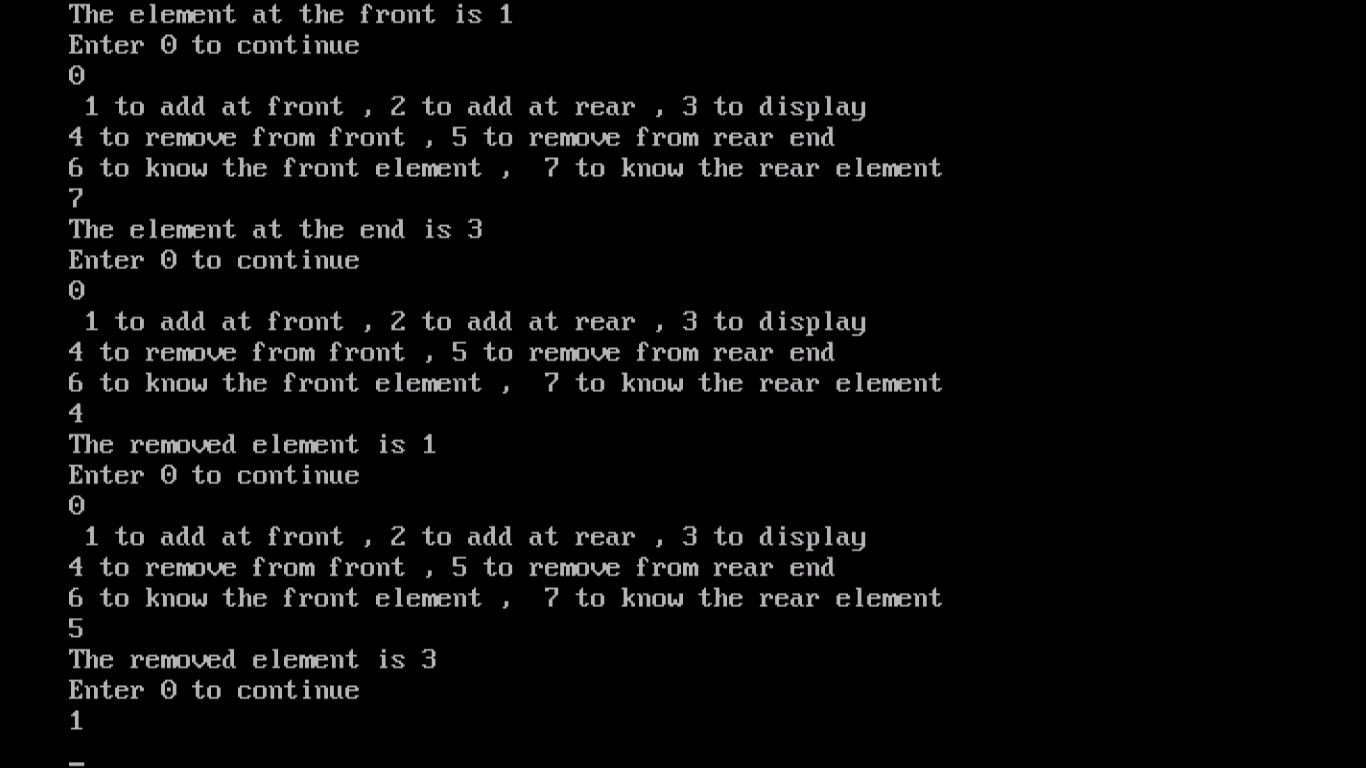
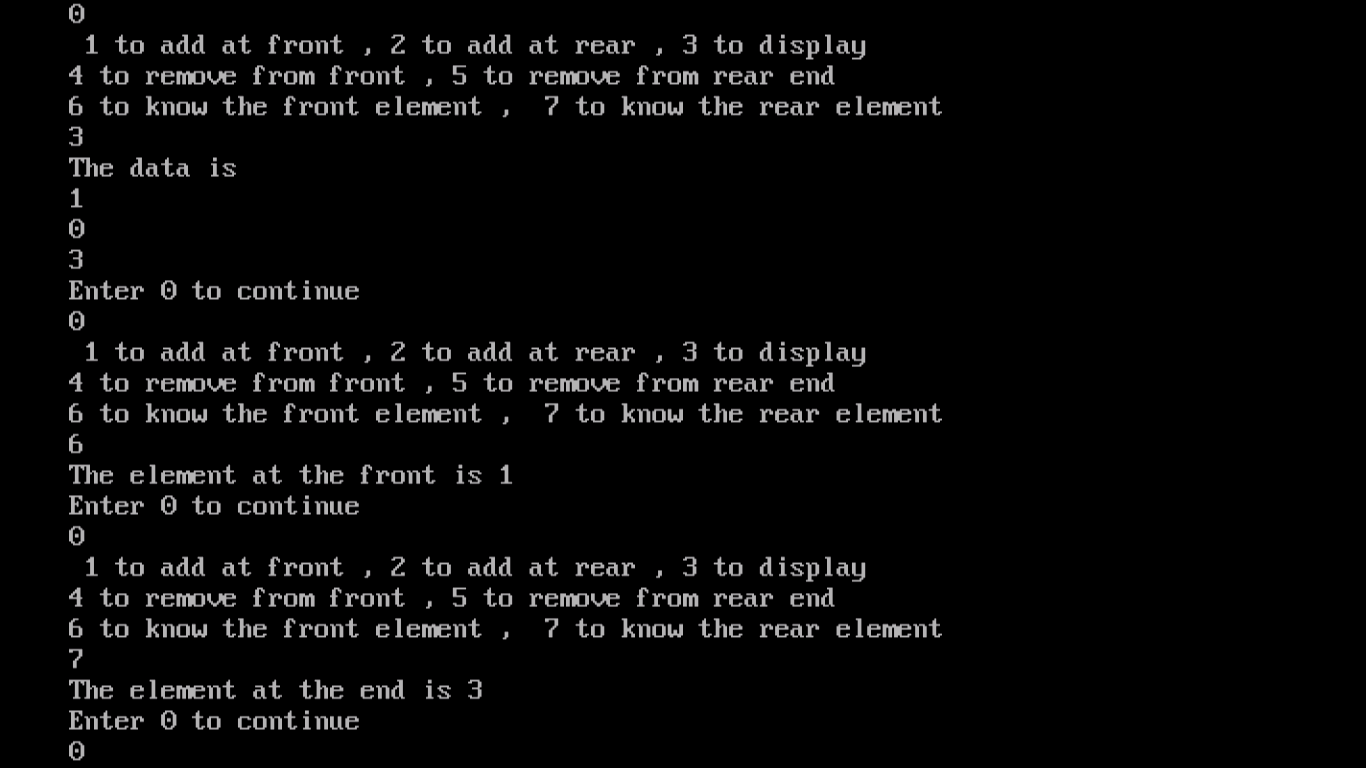
}

getch();

}

# OUTPUT





## 10. WAP to scan a polynomial using linked list and add two polynomial.

#include<iostream.h>

#include<conio.h>

#include<stdlib.h>

class poly

{

int exp;

int coeff;

poly \*next;

public:

void input1();

void input2();

void display();

void add();

};

poly \*head1,\*head2,\*head3;

void poly::input1() //Input for Polynomial 1

{

int degree;

cout<<endl<<"degree of d polynomial : ";

cin>>degree;

int c,e,i;

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

{

m:cout<<endl<<"exponent : ";

cin>>e;

if(e>degree)

{

cout<<endl<<"invalid exp!!! ";

goto m;

}

cout<<endl<<"coeff : ";

cin>>c;

poly \*temp1;

temp1=head1;

if(temp1==NULL)

{

temp1=new poly;

temp1->exp=e;

temp1->coeff=c;

temp1->next=NULL;

head1=temp1;

}

else

{

temp1=head1;

poly \*r;

r=new poly;

while(temp1->next!=NULL)

{

temp1=temp1->next;

}

r->exp=e;

r->coeff=c;

temp1->next=r;

r->next=NULL;

}

}

}

void poly::input2()//Input for Polynomial 2

{

int degree;

cout<<endl<<"degree of d polynomial : ";

cin>>degree;

int c,e,i;

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

{

m:cout<<endl<<"exponent : ";

cin>>e;

if(e>degree)

{

cout<<endl<<"invalid exp!!! ";

goto m;

}

cout<<endl<<"coeff : ";

cin>>c;

poly \*temp2;

temp2=head2;

if(temp2==NULL)

{

temp2=new poly;

temp2->exp=e;

temp2->coeff=c;

temp2->next=NULL;

head2=temp2;

}

else

{

temp2=head2;

poly \*r;

r=new poly;

while(temp2->next!=NULL)

{

temp2=temp2->next;

}

r->exp=e;

r->coeff=c;

temp2->next=r;

r->next=NULL;

}

}

}

void poly::add() //Addition of the polynomials

{

if(head1==NULL)

head3=head2;

if(head2==NULL)

head3=head1;

if((head1!=NULL)&&(head2!=NULL))

{

poly \*t1,\*t2;

t1=head1;

t2=head2;

int newcoeff,newexp;

while((t1!=NULL)&&(t2!=NULL))

{

if(t1->exp==t2->exp)

{

newcoeff=t1->coeff+t2->coeff;

newexp=t1->exp;

t1=t1->next;

t2=t2->next;

}

if(t1->exp<t2->exp)

{

newcoeff=t2->coeff;

newexp=t2->exp;

t2=t2->next;

}

if(t1->exp>t2->exp)

{

newcoeff=t1->coeff;

newexp=t1->exp;

t1=t1->next;

}

if(newcoeff!=0)

{

poly \*r,\*t3;

t3=head3;

r=new poly;

r->coeff=newcoeff;

r->exp=newexp;

r->next=NULL;

if(head3==NULL)

{

head3=r;

cout<<endl<<head3->coeff<<" "<<head3->exp<<endl;

}

else

{

t3=head3;

while(t3->next!=NULL)

{

t3=t3->next;

}

t3->next=r;

cout<<endl<<t3->coeff<<" "<<t3->exp<<endl;

}

}

}

}

}

void poly::display() //Display the results

{

if(head1!=NULL)

{

poly \*temp;

temp=head1;

cout<<endl<<"--poly1--"<<endl<<endl;

while(temp->next!=NULL)

{

cout<<temp->coeff<<"y^"<<temp->exp<<" + ";

temp=temp->next;

}

cout<<temp->coeff<<"y^"<<temp->exp;

}

if(head2!=NULL)

{

poly \*temp1;

temp1=head2;

cout<<endl<<endl<<"--poly2--"<<endl<<endl;

while(temp1->next!=NULL)

{

cout<<temp1->coeff<<"y^"<<temp1->exp<<" + ";

temp1=temp1->next;

}

cout<<temp1->coeff<<"y^"<<temp1->exp;

}

if(head3!=NULL)

{

poly \*temp3;

temp3=head3;

cout<<endl<<endl<<"--sum--"<<endl<<endl;

while(temp3->next!=NULL)

{

cout<<temp3->coeff<<"y^"<<temp3->exp<<" + ";

temp3=temp3->next;

}

cout<<temp3->coeff<<"y^"<<temp3->exp;

}

}

void main()

{

poly t;

char ch;

clrscr();

cout<<"do u want enter polynomial expression 1 (y/n) :";

cin>>ch;

if(ch=='y')

t.input1(); //function call

cout<<"do u want enter polynomial expression 2 (y/n) :";

cin>>ch;

if(ch=='y')

t.input2();

t.add();

t.display();

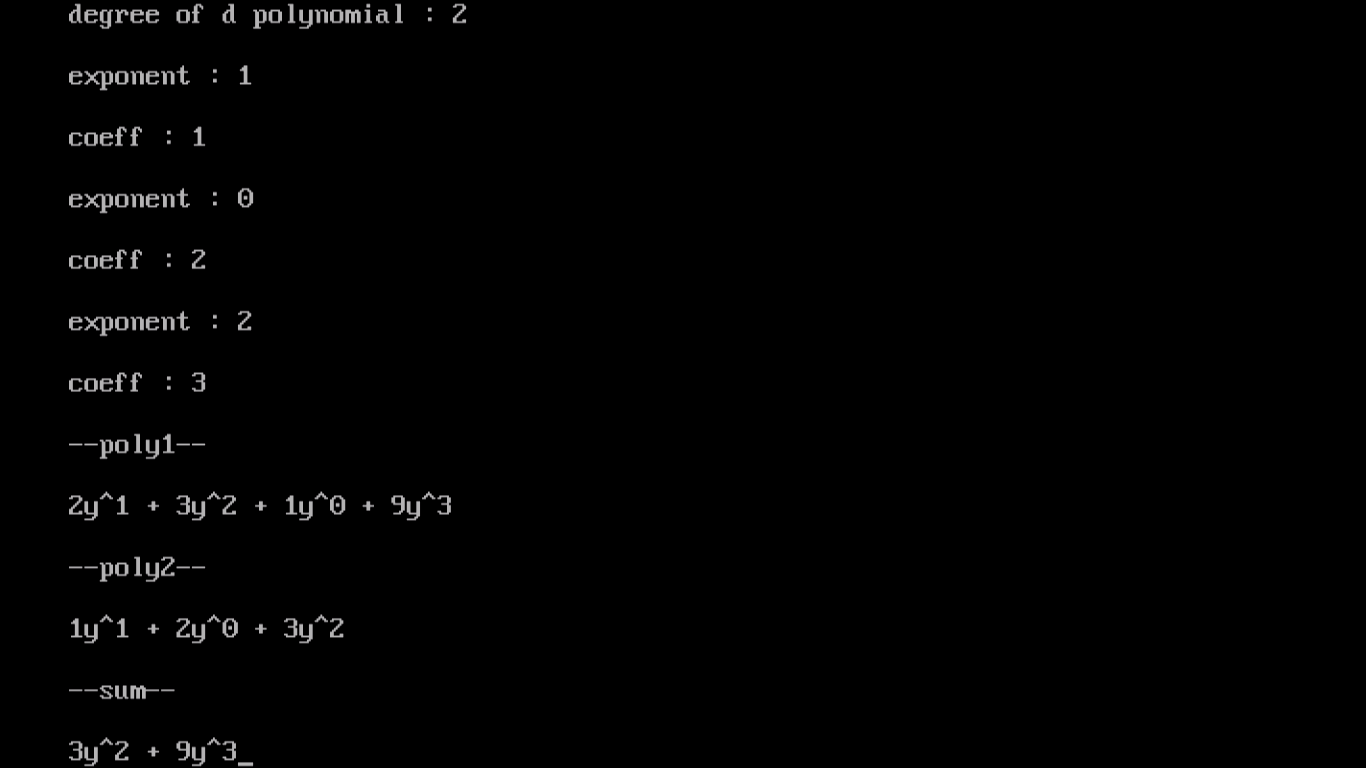
getch();

}

## 

## OUTPUT

## 



## 11. WAP to calculate factorial and to compute the factors of a given no. (i)using recursion, (ii)using iteration

#include<iostream.h>

#include<conio.h>

/\* Recursive Version \*/

int recur(int n)

{

return n>=1 ? n \* recur(n-1) : 1;

}

/\* Iterative Version \*/

int iteration(int n)

{

int f = 1;

int i;

for(i = 1; i <= n; i++)

{

f \*= i;

}

return f;

}

void main()

{

clrscr();

int k,n;

cout<<"Enter the value whose factorial is to be calculated"<<endl;

cin>>n;

cout<<"Enter 1 to find gcd by recursion and 2 to find gcd by iteration"<<endl;

cin>>k;

int p;

switch(k)

{

case 1:

p=recur(n);

break;

case 2:

p=iteration(n);

break;

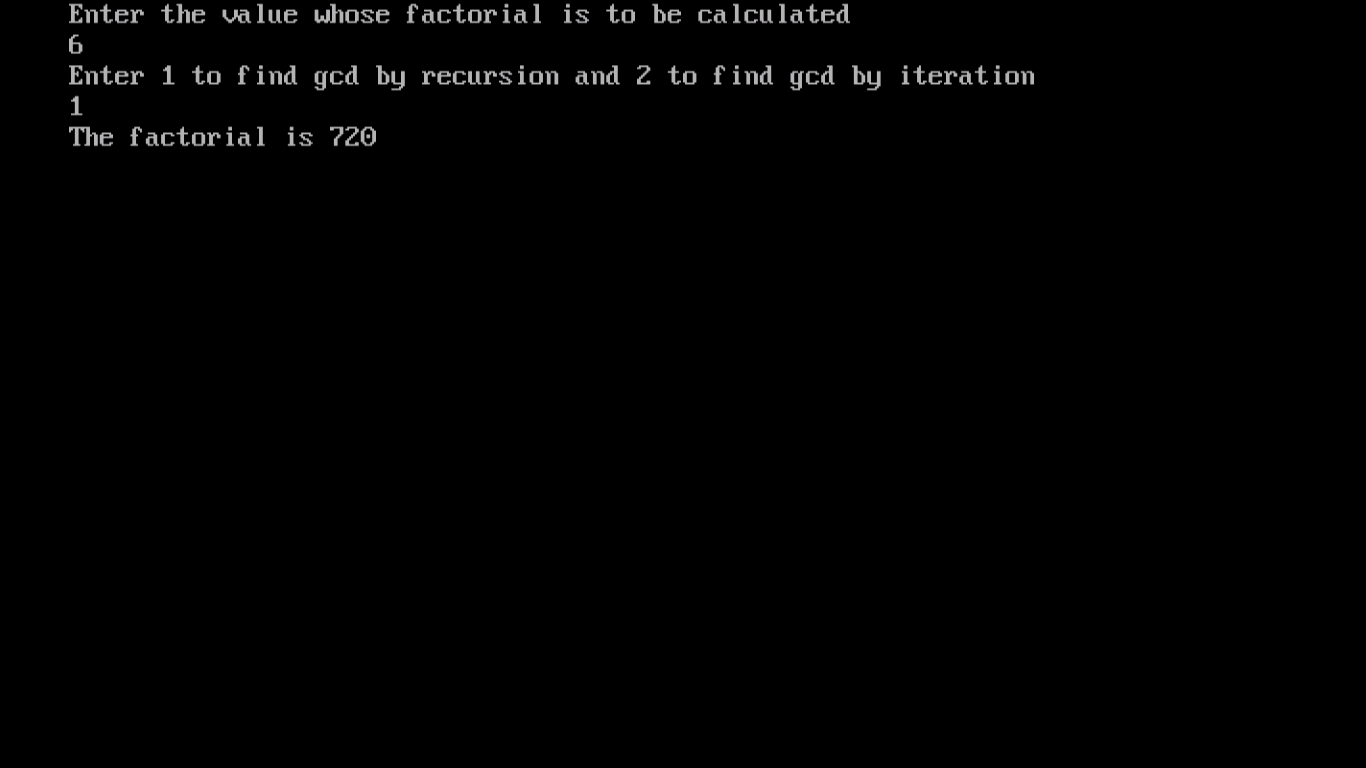
}

cout<<"The factorial is "<<p<<endl;

getch();

}

## OUTPUT



## 12. WAP to display fibonacci series (i)using recursion, (ii) using iteration

#include<conio.h>

#include <iostream.h>

/\* Fibonacci: recursive version \*/

int Fibonacci\_R(int n) {

if (n <= 0) return 0;

else if (n == 1) return 1;

else return Fibonacci\_R(n - 1) + Fibonacci\_R(n - 2);

}

// iterative version

int Fibonacci\_I(int n) {

int fib[] = {

0,

1,

1

};

for (int i = 2; i <= n; i++) {

fib[i % 3] = fib[(i - 1) % 3] + fib[(i - 2) % 3];

cout << fib[i % 3] << endl;

}

return fib[n % 3];

}

int main(void) {

clrscr();

int a;

cout << "Enter the number of terms >=2 you want of the serioes. ";

cin >> a;

// calculate the fib(i) from scratch for each i <= a using your recursive function

cout << endl << "From recursive function" << endl;

cout<<"0" <<endl<<"1"<<endl;

for (int i = 2; i <= a; ++i)

cout << Fibonacci\_R(i) << endl;

cout << endl;

// or calculate fib(a) once and output the intermediate results from the looping version

cout << "From iterative function" << endl;

cout<<"0" <<endl<<"1"<<endl;

Fibonacci\_I(a);

cout << endl;

getch();

return 0;

## OUTPUT



## 13. WAP to calculate GCD of 2 number (i) with recursion (ii) without recursion

#include<iostream.h>

#include<conio.h>

int gcditeration(int n,int n1)

{

int min=n1>n?n:n1;

for(int i=min;i>1;i--)

{

if((n%i==0)&& (n1%i==0))

return i;

}

return 1;

}

int gcdrecur(int p,int q)

{

if(q!=0)

return gcdrecur(q,p%q);

else

return p;

}

void main()

{

clrscr();

int p;

int n;

int n1;

cout<<"Enter the elements whose gcd is to be found out"<<endl;

cin>>n;

cin>>n1;

int k;

cout<<"Enter 1 to find gcd by recursion and 2 to find gcd by iteration"<<endl;

cin>>k;

switch(k)

{

case 1:

p=gcdrecur(n,n1);

break;

case 2:

p=gcditeration(n,n1);

break;

}

cout<<"The gcd is "<<p<<endl;

getch();

## OUTPUT





## 14. WAP to create a Binary Search Tree and include following operations in tree:

## (a) Insertion (Recursive and Iterative Implementation)

## (b) Deletion by copying

## (c) Deletion by Merging

## (d) Search a no. in BST

## (e) Display its preorder, postorder and inorder traversals Recursively

## (f) Display its preorder, postorder and inorder traversals Iteratively

## (g) Display its level-by-level traversals

## (h) Count the non-leaf nodes and leaf nodes

## (i) Display height of tree

## (j) Create a mirror image of tree

## (k) Check whether two BSTs are equal or not

#include<iostream.h>

#include<stdlib.h>

#include<conio.h>

struct treeNode

{

int data;

treeNode \*left;

treeNode \*right;

};

treeNode\* FindMin(treeNode \*node)

{

if(node==NULL)

{

/\* There is no element in the tree \*/

return NULL;

}

if(node->left) /\* Go to the left sub tree to find the min element \*/

return FindMin(node->left);

else

return node;

}

treeNode\* FindMax(treeNode \*node)

{

if(node==NULL)

{

/\* There is no element in the tree \*/

return NULL;

}

if(node->right) /\* Go to the left sub tree to find the min element \*/

return(FindMax(node->right));

else

return node;

}

treeNode \*Insert(treeNode \*node,int data)

{

if(node==NULL)

{

treeNode \*temp;

temp=new treeNode;

//temp = (treeNode \*)malloc(sizeof(treeNode));

temp -> data = data;

temp -> left = temp -> right = NULL;

return temp;

}

if(data >(node->data))

{

node->right = Insert(node->right,data);

}

else if(data < (node->data))

{

node->left = Insert(node->left,data);

}

/\* Else there is nothing to do as the data is already in the tree. \*/

return node;

}

treeNode \* Delet(treeNode \*node, int data)

{

treeNode \*temp;

if(node==NULL)

{

cout<<"Element Not Found";

}

else if(data < node->data)

{

node->left = Delet(node->left, data);

}

else if(data > node->data)

{

node->right = Delet(node->right, data);

}

else

{

/\* Now We can delete this node and replace with either minimum element

in the right sub tree or maximum element in the left subtree \*/

if(node->right && node->left)

{

/\* Here we will replace with minimum element in the right sub tree \*/

temp = FindMin(node->right);

node -> data = temp->data;

/\* As we replaced it with some other node, we have to delete that node \*/

node -> right = Delet(node->right,temp->data);

}

else

{

/\* If there is only one or zero children then we can directly

remove it from the tree and connect its parent to its child \*/

temp = node;

if(node->left == NULL)

node = node->right;

else if(node->right == NULL)

node = node->left;

free(temp); /\* temp is longer required \*/

}

}

return node;

}

treeNode \* Find(treeNode \*node, int data)

{

if(node==NULL)

{

/\* Element is not found \*/

return NULL;

}

if(data > node->data)

{

/\* Search in the right sub tree. \*/

return Find(node->right,data);

}

else if(data < node->data)

{

/\* Search in the left sub tree. \*/

return Find(node->left,data);

}

else

{

/\* Element Found \*/

return node;

}

}

void Inorder(treeNode \*node)

{

if(node==NULL)

{

return;

}

Inorder(node->left);

cout<<node->data<<" ";

Inorder(node->right);

}

void Preorder(treeNode \*node)

{

if(node==NULL)

{

return;

}

cout<<node->data<<" ";

Preorder(node->left);

Preorder(node->right);

}

void Postorder(treeNode \*node)

{

if(node==NULL)

{

return;

}

Postorder(node->left);

Postorder(node->right);

cout<<node->data<<" ";

}

int main()

{

treeNode \*root = NULL,\*temp;

int ch;

//clrscr();

while(1)

{

cout<<"\n1.Insert\n2.Delete\n3.Inorder\n4.Preorder\n5.Postorder\n6.FindMin\n7.FindMax\n8.Search\n9.Exit\n";

cout<<"Enter ur choice:";

cin>>ch;

switch(ch)

{

case 1:

cout<<"\nEnter element to be insert:";

cin>>ch;

root = Insert(root, ch);

cout<<"\nElements in BST are:";

Inorder(root);

break;

case 2:

cout<<"\nEnter element to be deleted:";

cin>>ch;

root = Delet(root,ch);

cout<<"\nAfter deletion elements in BST are:";

Inorder(root);

break;

case 3:

cout<<"\nInorder Travesals is:";

Inorder(root);

break;

case 4:

cout<<"\nPreorder Traversals is:";

Preorder(root);

break;

case 5:

cout<<"\nPostorder Traversals is:";

Postorder(root);

break;

case 6:

temp = FindMin(root);

cout<<"\nMinimum element is :"<<temp->data;

break;

case 7:

temp = FindMax(root);

cout<<"\nMaximum element is :"<<temp->data;

break;

case 8:

cout<<"\nEnter element to be searched:";

cin>>ch;

temp = Find(root,ch);

if(temp==NULL)

{

cout<<"Element is not foundn";

}

else

{

cout<<"Element "<<temp->data<<" is Found\n";

}

break;

case 9:

exit(0);

break;

default:

cout<<"\nEnter correct choice:";

break;

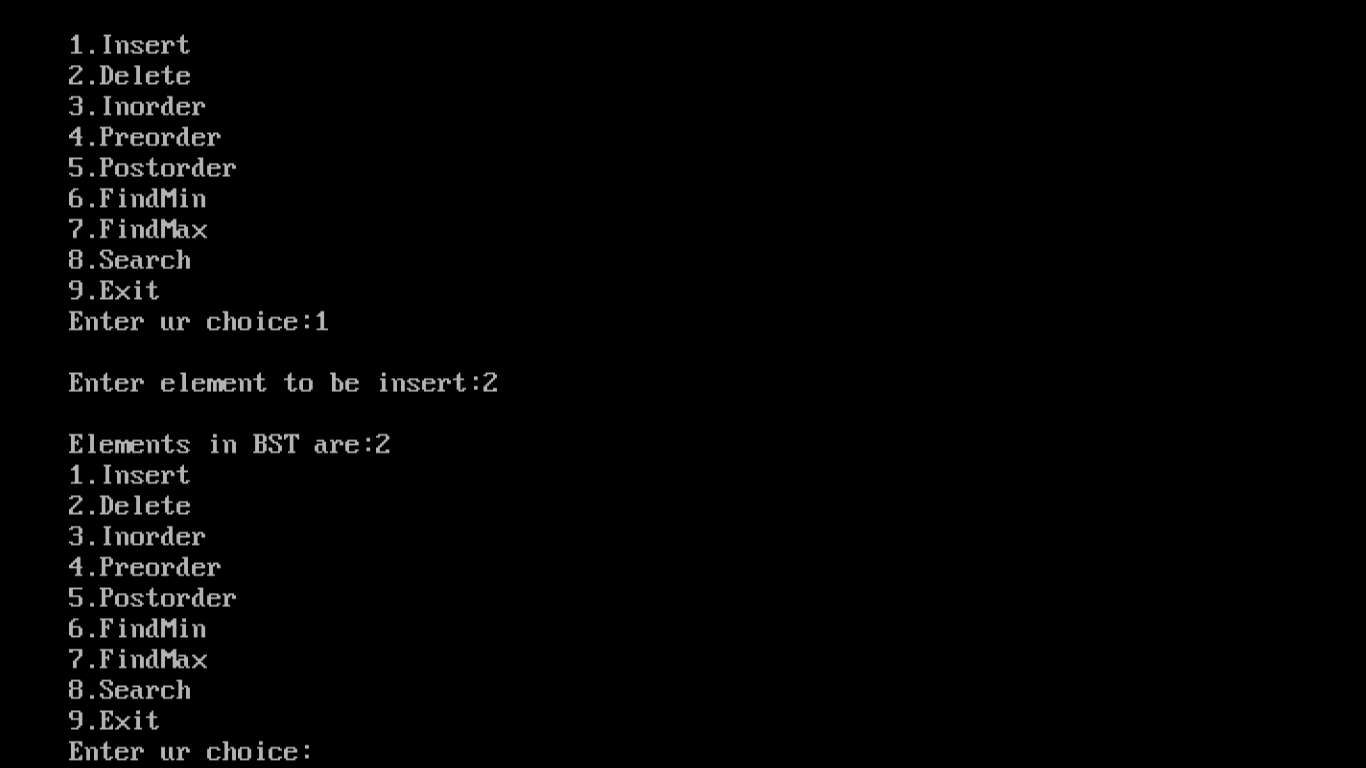
}

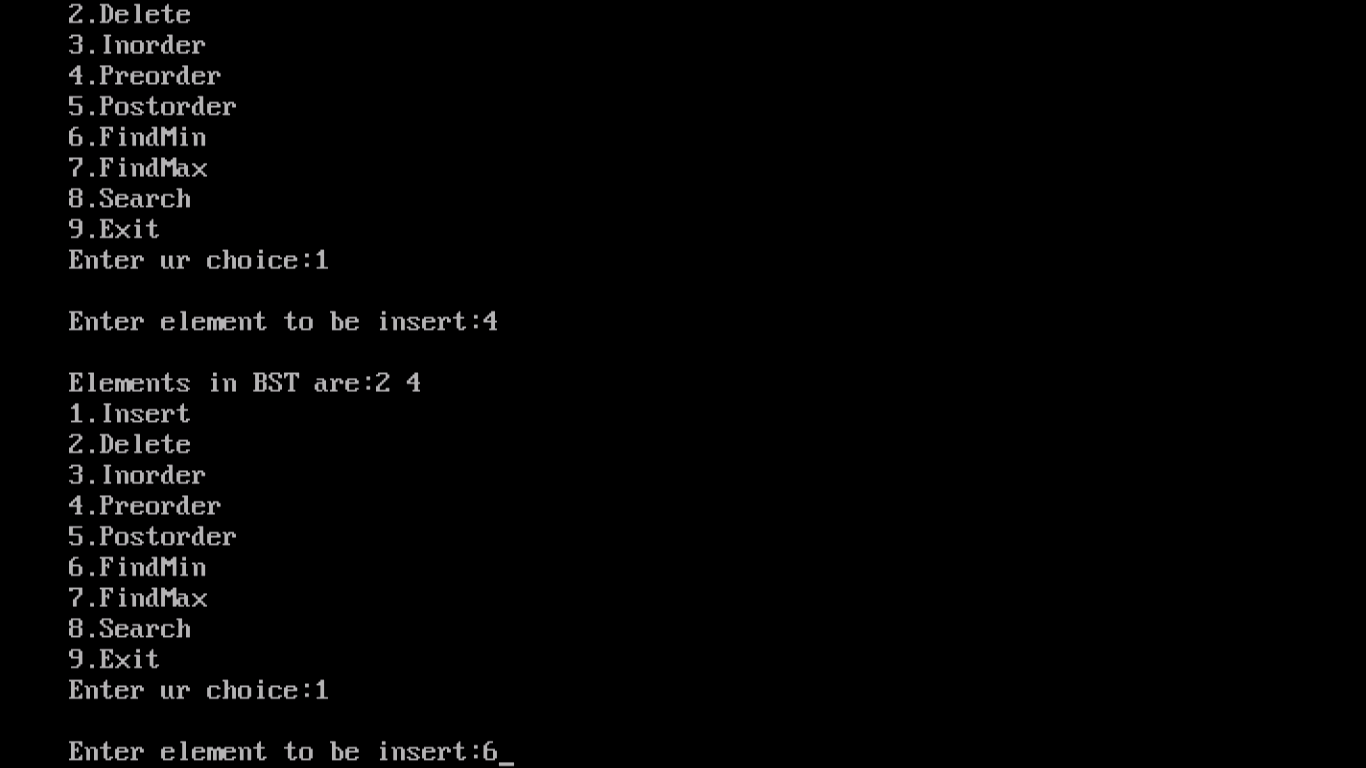
}

return 0;

}

## OUTPUT







## 15.WAP to convert the Sparse matrix into non zero form .

#include<iostream.h>

#include<conio.h>

class term

{

public:

int value;

int r,c;

term \* next;

term()

{

next=NULL;

}

};

void main()

{

clrscr();

int row,column;

int a[10][10];

cout<<"Enter the number of rows and columns "<<endl;

cin>>row;

cin>>column;

int d=0;

cout<<"Enter the elements"<<endl;

for(int i=0;i<row;i++)

{

for(int j=0;j<column;j++)

{

cin>>a[i][j];

}

}

cout<<"The matrix is "<<endl;

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

{

for(int j=0;j<column;j++)

{

cout<<a[i][j]<<" ";

if(a[i][j]!=0)

d++;

}

cout<<endl;

}

term arr[100];

int co=0;

cout<<"The non zero form is "<<endl;

cout<<"ROW " << "\tCOLUMN " <<"\tVALUE"<<endl;

for(i=0;i<d;)

{

for(int j=0;j<row;j++)

{

for(int k=0;k<column;k++)

{

if(a[j][k]!=0)

{

arr[i].value=a[j][k];

arr[i].r=j;

arr[i].c=k;

cout<<arr[i].r<<"\t"<<arr[i].c<<"\t"<<arr[i].value<<"\t"<<endl;

i++; }

}

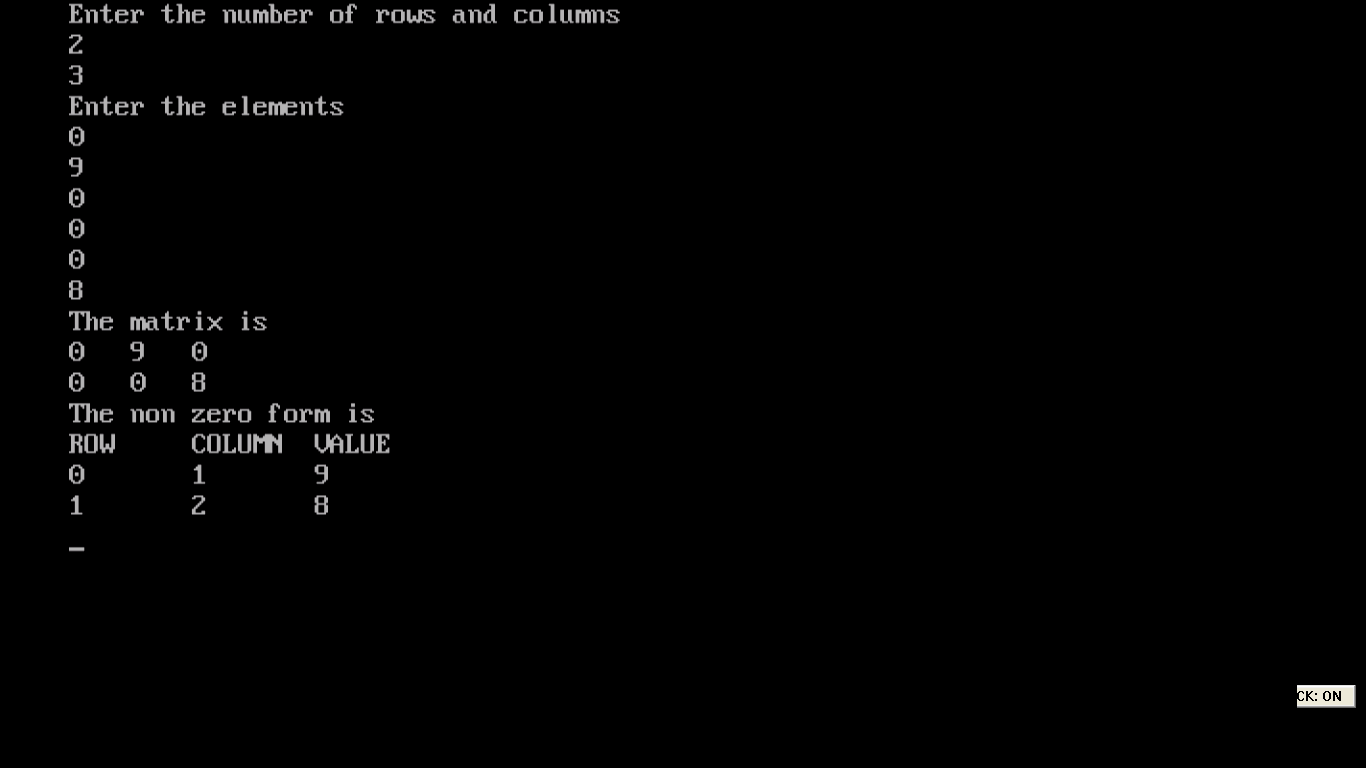
}

}

getch();

}

### OUTPUT



## 16. WAP to reverse the order of the elements in the stack using additional stack.

#include<iostream.h>

#include <conio.h>

#define max 10

class mystack

{

int st[max];

int st2[max];

int top;

public:

mystack()

{

top=-1;

}

void push()

{

int x;

cout<<"enter element x\n";

cin>>x;

if(top==max-1)

{

cout<<"can't insert\n";

}

else

{

top++;

st[top]=x;

}

}

int pop()

{

int t;

if(top==-1)

{

cout<<"stack is empty\n";

return 0;

}

else

{

t=st[top];

top--;

return t;

}

}

void clear()

{

while(top!=-1)

{

pop();

}

cout<<"empty stack\n";

}

void display()

{

cout<<"Stack is:"<<endl;

for(int i=top;i>=0;i--) {

cout<<st[i]<<" ";

}

}

void reverse()

{

int toptop=top;

for(int i=0;i<=toptop;i++)

{

st2[i]=pop();

}

cout<<"reversed stack: "<<endl;

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

{

cout<<st2[i]<<" ";

}

}

};

int main()

{

clrscr();

char ch;

int choice;

mystack s1;

cout<<endl<<"Operations on stack"<<endl<<endl;

cout<<"1.pushing a element\n";

cout<<"2.popping the element\n";

cout<<"3.clear the stack\n";

cout<<"4.reverse stack\n";

do{

cout<<"enter the choice: ";

cin>>choice;

switch(choice)

{ case 1:

s1.push();

s1.display();

break;

case 2:

s1.pop();

break;

case 3:

s1.clear();

s1.display();

break;

case 4:

s1.reverse();

break;

default:

cout<<"wrong choice";

}

cout<<"\ndo you want to enter more operations\n";

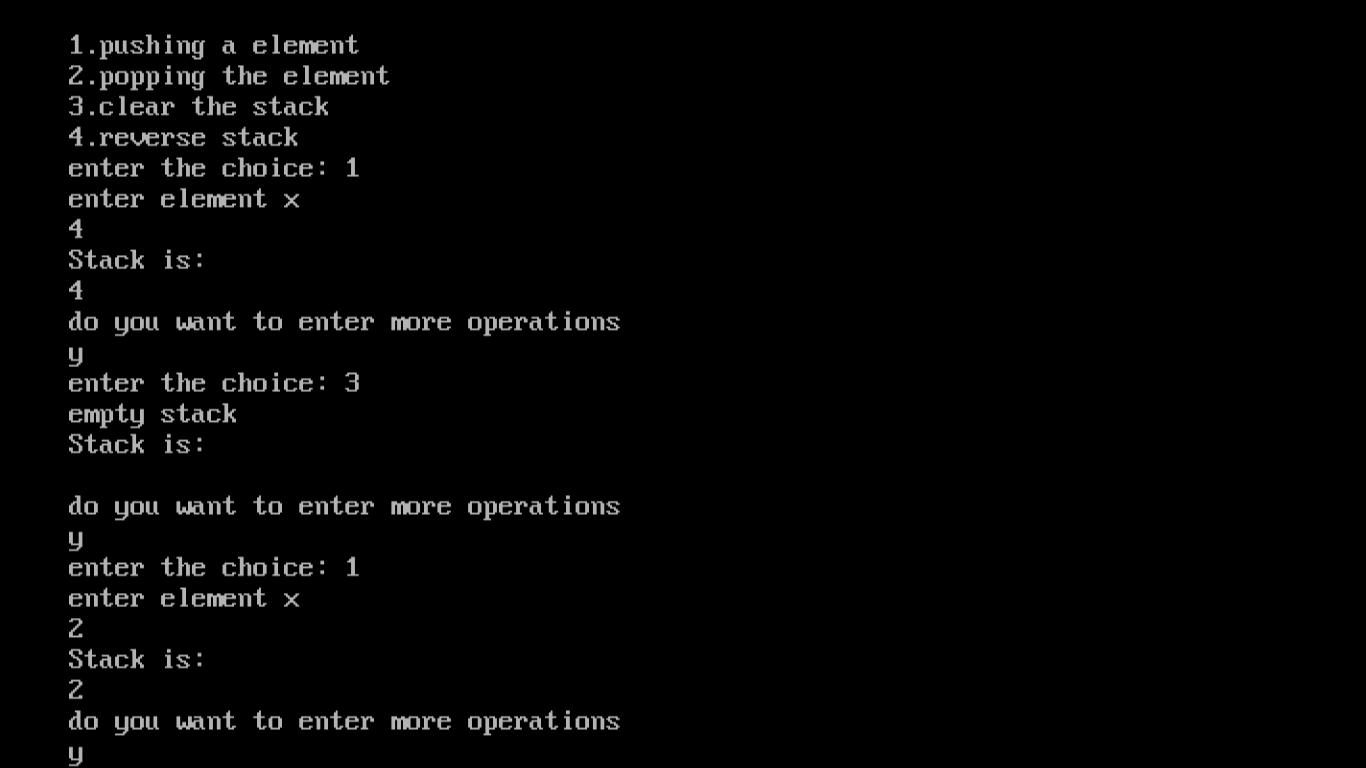
cin>>ch;

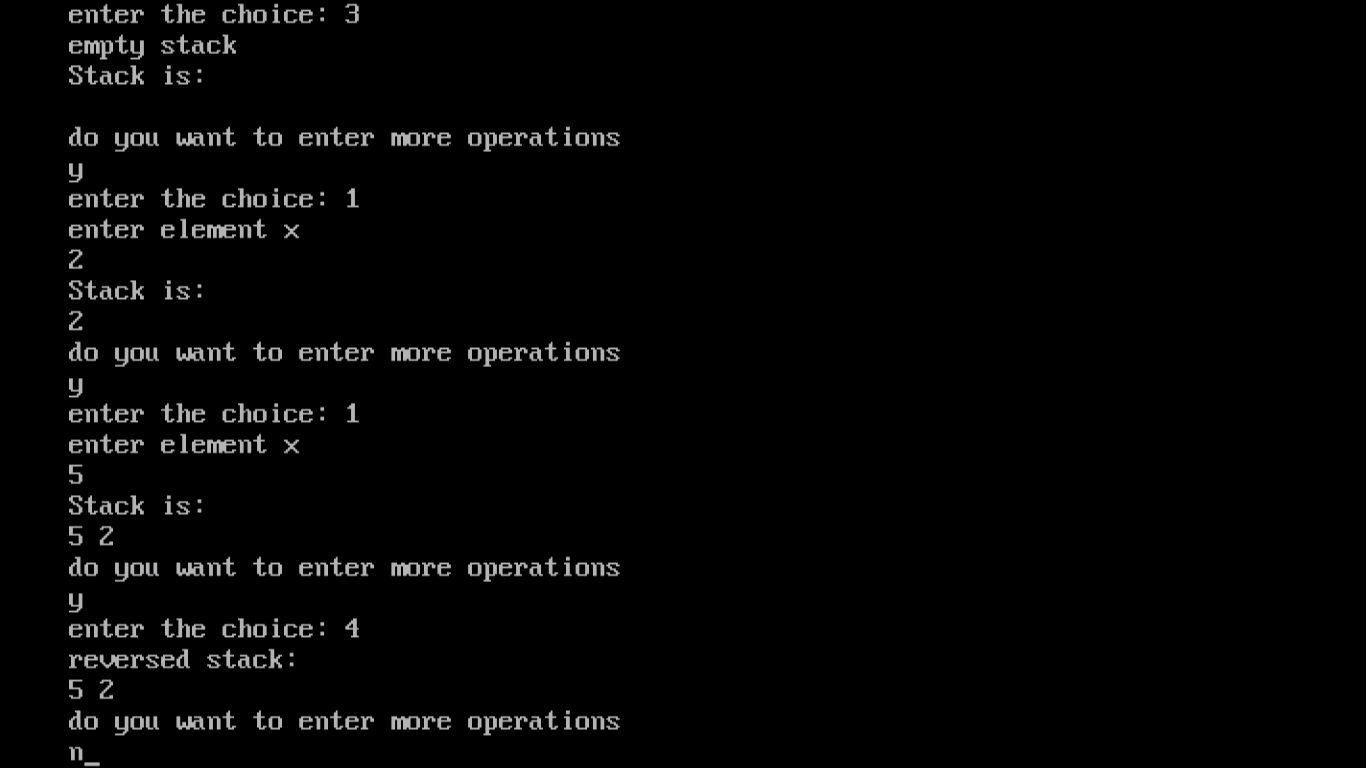
}while(ch=='y'||ch=='Y');

return 0;

}

## OUTPUT





## 17. WAP to reverse the order of the elements in the stack using additional Queue.

#include<conio.h>

#include<stdio.h>

#include<iostream.h>

#define MAX 20

void show(int stack[], int size, int top);

void show(int stack[],int size,int top)

{

int i;

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

{

cout<<"\nValue at "<<top<<" is "<<stack[top]<<endl;

top=top-1;

}

}

void reverse(int stack[],int qu[],int \*t,int \*r,int \*f)

{

\*f=0;

while(\*t>-1)

{

\*r=\*r+1;

qu[\*r]=stack[\*t];

\*t=\*t-1;

}

while(\*f<=\*r)

{

\*t=\*t+1;

stack[\*t]=qu[\*f];

\*f=\*f+1;

}

}

void main()

{

int size;

int item,t,i,stack[MAX],quee[MAX];

int top=-1,front=-1,rear=-1;

clrscr();

cout<<"Enter size of stack: ";

cin>>size;

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

{

top=top+1;

cout<<"Enter value of for position "<<top<<": ";

cin>>item;

stack[top]=item;

}

show(stack,size,top);

reverse(stack,quee,&top,&rear,&front);

cout<<"\nAfter reverse, using additional queue: ";

show(stack,size,top);

getch();

}

# OUTPUT



## 18. WAP to implement Diagonal Matrix using one-dimensional array.

#include <iostream.h>

#include<conio.h>

class DiagonalMatrix

{

int \*arr;

int size;

public:

DiagonalMatrix(int size=0)

{

if(size>0)

{

this->size=size;

arr=new int [size];

}

else

{

cout<<"Enter a non-zero positive size "<<endl;

}

}

void set\_values()

{

cout<<"Enter the diagonal elements "<<endl;

for(int i=0; i<size; i++)

cin>>arr[i];

}

int get\_values(int i,int j)

{ if(i==j)

return arr[i];

else

return 0;

}

};

int main()

{

clrscr();

int row,col;

char choice='y';

while(choice=='y'||choice=='Y')

{

cout<<"Enter the number of rows ";

cin>>row;

cout<<"Enter the number of columns ";

cin>>col;

if(row!=col)

cout<<"Enter a square matrix "<<endl;

DiagonalMatrix D(row); //no of elements for lower triangular matrix

D.set\_values(); //enter the elements

cout<<"Elements of matrix are"<<endl;

for(int i=0;i<row;i++)

{

for(int j=0;j<col;j++)

{

cout<<D.get\_values(i,j)<<" "; //display the matrix

}

cout<<endl;

}

cout<<"Enter y or Y to continue with another number ";

cin>>choice;

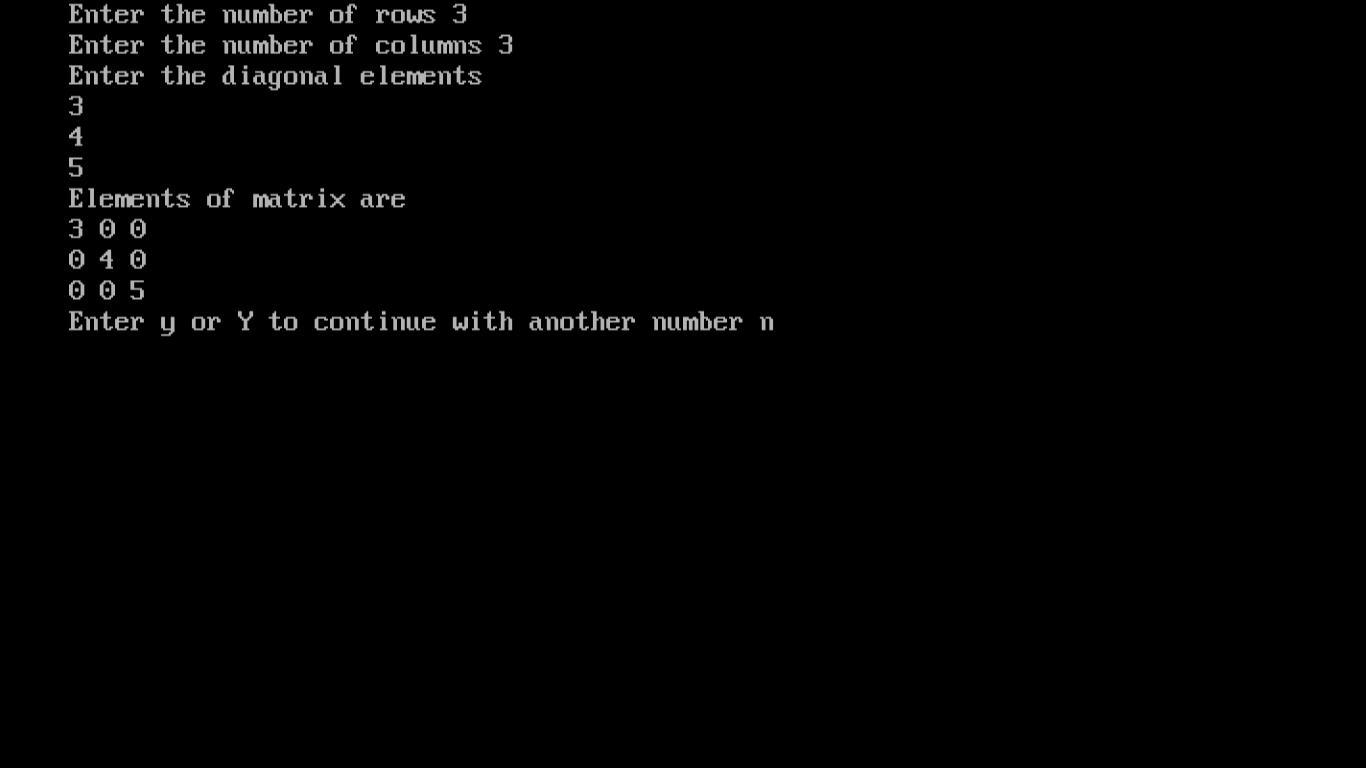
}

getch();

return 0;

}

## OUTPUT



## 19. WAP to implement Lower Triangular Matrix using one-dimensional array.

#include <iostream.h>

#include <conio.h>

class LowerTriangularMatrix

{

int \*arr;

int size;

public:

LowerTriangularMatrix(int size=0)

{

if(size>0)

{

this->size=size;

arr=new int [size];

}

else

{

cout<<"Enter a non-zero positive size "<<endl;

}

}

void set\_values()

{

cout<<"Enter lower triangle elements "<<endl;

for(int i=0; i<size; i++)

cin>>arr[i];

}

int get\_values(int i,int j)

{ if(i<=j)

return 0;

else

return arr[i\*(i-1)/2+j];

}

};

int main()

{

clrscr();

int row,col;

char choice='y';

while(choice=='y'||choice=='Y')

{

cout<<"Enter the number of rows ";

cin>>row;

cout<<"Enter the number of columns ";

cin>>col;

if(row!=col)

cout<<"Enter a square matrix "<<endl;

LowerTriangularMatrix L(row\*(row-1)/2); //no of elements for lower triangular matrix

L.set\_values(); //enter the elements

cout<<"Elements of matrix are"<<endl;

for(int i=0;i<row;i++)

{

for(int j=0;j<col;j++)

{

cout<<L.get\_values(i,j)<<" "; //display the matrix

}

cout<<endl;

}

cout<<"Enter y or Y to continue with another number ";

cin>>choice;

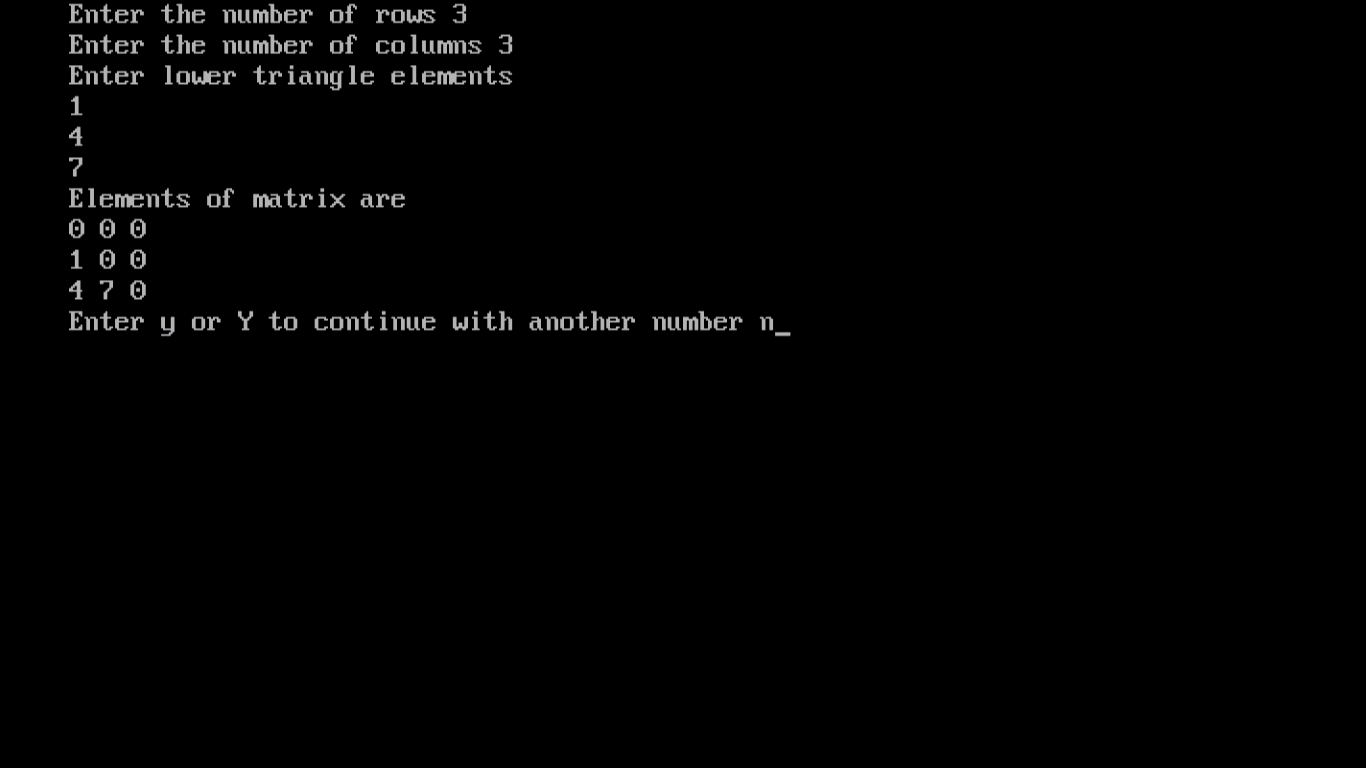
}

getch();

return 0;

}

## OUTPUT



## 20. WAP to implement Upper Triangular Matrix using one-dimensional array.

#include <iostream.h>

#include <conio.h>

class UpperTriangularMatrix

{

int \*arr;

int size;

public:

UpperTriangularMatrix(int size=0)

{

if(size>0)

{

this->size=size;

arr=new int [size];

}

else

{

cout<<"Enter a non-zero positive size "<<endl;

}

}

void set\_values()

{

cout<<"Enter upper triangle elements "<<endl;

for(int i=0; i<size; i++)

cin>>arr[i];

}

int get\_values(int i,int j)

{ if(i>=j)

return 0;

else

return arr[j\*(j-1)/2+i];

}

};

int main()

{

clrscr();

int row,col;

char choice='y';

while(choice=='y'||choice=='Y')

{

cout<<"Enter the number of rows ";

cin>>row;

cout<<"Enter the number of columns ";

cin>>col;

if(row!=col)

cout<<"Enter a square matrix "<<endl;

UpperTriangularMatrix U(row\*(row-1)/2); //no of elements for lower triangular matrix

U.set\_values(); //enter the elements

cout<<"Elements of matrix are"<<endl;

for(int i=0;i<row;i++)

{

for(int j=0;j<col;j++)

{

cout<<U.get\_values(i,j)<<" "; //display the matrix

}

cout<<endl;

}

cout<<"Enter y or Y to continue with another number ";

cin>>choice;

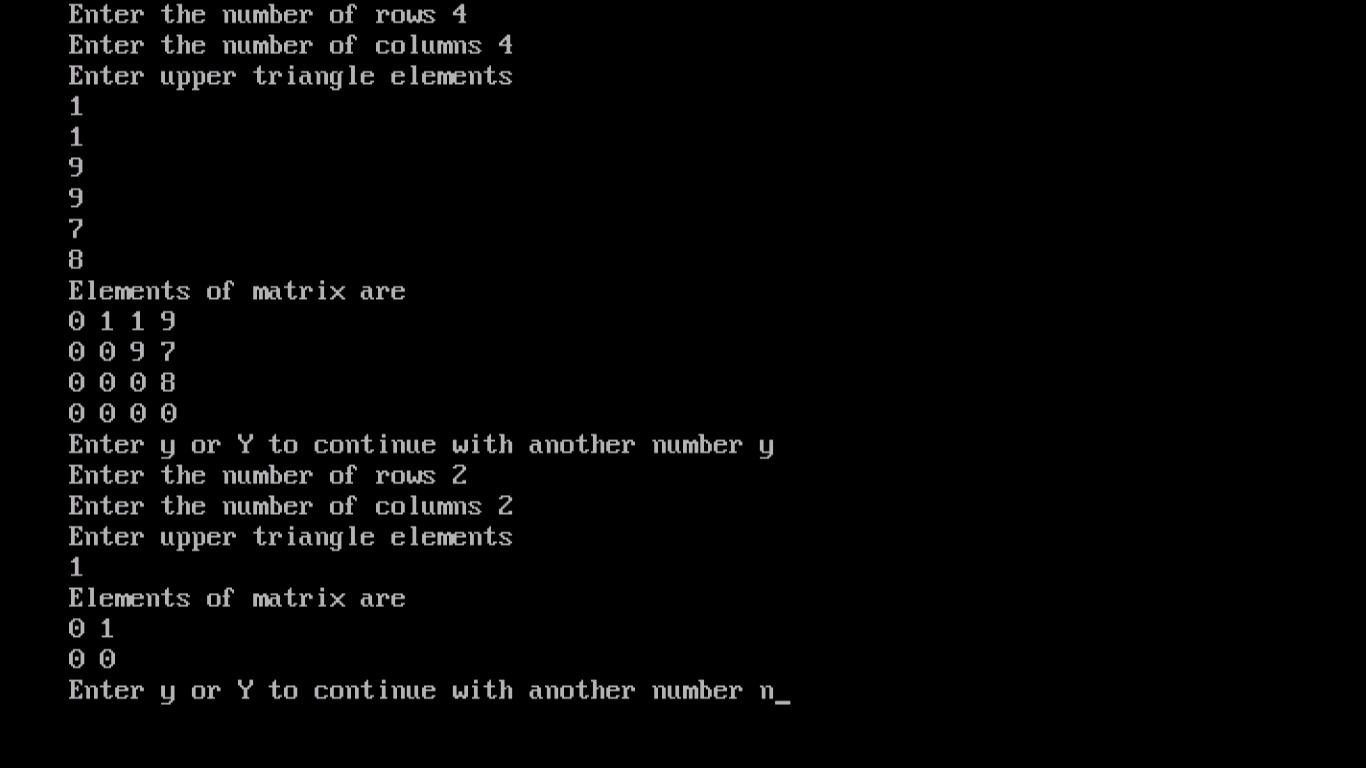
}

getch();

return 0;

}

## OUTPUT



## 21. WAP to implement Symmetric Matrix using one-dimensional array.

#include <iostream.h>

#include <conio.h>

class Transpose

{

int \*a;

int size;

public:

Transpose(int size=0)

{

if(size>0)

{

this->size=size;

a=new int [size];

}

else

{

cout<<"Enter a non-zero positive size ";

}

}

void set\_values(int rows)

{

cout<<"Enter lower triangle elements "<<endl;

for(int i=0; i<size-rows; i++)

cin>>a[i];

cout<<"Enter diagonal elements"<<endl;

for( i=size-rows;i<size;i++)

cin>>a[i];

}

int get\_values(int i,int j,int rows)

{

if(i<0 || i>=size || j<0 ||j>=size)

cout<<"index out of bounds";

else if(i>j)

return a[i\*(i-1)/2+j];

else if(i<j)

return a[j\*(j-1)/2+i];

else

return a[size-rows+i];

}

};

int main()

{

int row,col;

char choice='y';

while(choice=='y'||choice=='Y')

{

cout<<"Enter the number of rows ";

cin>>row;

cout<<"Enter the number of columns ";

cin>>col;

if(row!=col)

cout<<"Should be a square matrix ";

Transpose T((row\*(row-1)/2)+row);

T.set\_values(row);

cout<<"Elements of matrix are"<<endl;

for(int i=0;i<row;i++)

{

for(int j=0;j<col;j++)

{

cout<<T.get\_values(i,j,row)<<" ";

}

cout<<endl;

}

cout<<"Enter y or Y to continue with another number ";

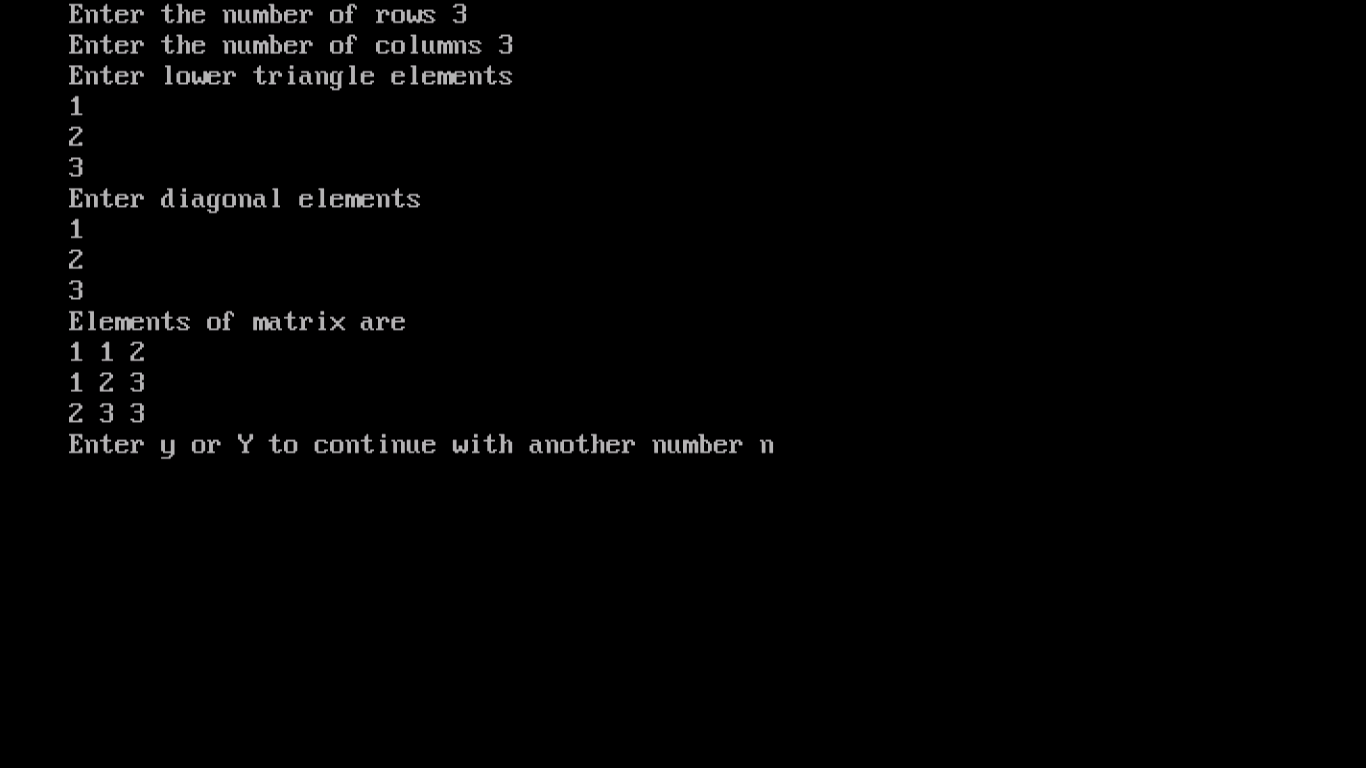
cin>>choice;

}

return 0;

}

## OUTPUT



## 22. WAP to create a Threaded Binary Tree as per inorder traversal, and implement operations like finding the successor / predecessor of an element, insert an element, inorder traversal.

## 

#include <iostream.h>

#include <conio.h>

#define MAX\_VALUE 65536

#define true 1

#define false 0

/\* Class Node \*/

class Node

{

public:

int key;

Node \*left, \*right;

int leftThread,rightThread;

};

/\* Class ThreadedBinarySearchTree \*/

class ThreadedBinarySearchTree

{

private:

Node \*root;

public:

/\* Constructor \*/

ThreadedBinarySearchTree()

{

root = new Node();

root->right = root->left = root;

root->leftThread = true;

root->key = MAX\_VALUE;

}

/\* Function to clear tree \*/

void makeEmpty()

{

root = new Node();

root->right = root->left = root;

root->leftThread = true;

root->key = MAX\_VALUE;

}

/\* Function to insert a key \*/

void insert(int key)

{

Node \*p = root;

for (;;)

{

if (p->key < key)

{

if (p->rightThread)

break;

p = p->right;

}

else if (p->key > key)

{

if (p->leftThread)

break;

p = p->left;

}

else

{

/\* redundant key \*/

return;

}

}

Node \*tmp = new Node();

tmp->key = key;

tmp->rightThread = tmp->leftThread = true;

if (p->key < key)

{

/\* insert to right side \*/

tmp->right = p->right;

tmp->left = p;

p->right = tmp;

p->rightThread = false;

}

else

{

tmp->right = p;

tmp->left = p->left;

p->left = tmp;

p->leftThread = false;

}

}

/\* Function to search for an element \*/

int search(int key)

{

Node \*tmp = root->left;

for (;;)

{

if (tmp->key < key)

{

if (tmp->rightThread)

return false;

tmp = tmp->right;

}

else if (tmp->key > key)

{

if (tmp->leftThread)

return false;

tmp = tmp->left;

}

else

{

return 1;

}

}

}

/\* Fuction to delete an element \*/

void Delete(int key)

{

Node \*dest = root->left, \*p = root;

for (;;)

{

if (dest->key < key)

{

/\* not found \*/

if (dest->rightThread)

return;

p = dest;

dest = dest->right;

}

else if (dest->key > key)

{

/\* not found \*/

if (dest->leftThread)

return;

p = dest;

dest = dest->left;

}

else

{

/\* found \*/

break;

}

}

Node \*target = dest;

if (!dest->rightThread && !dest->leftThread)

{

/\* dest has two children\*/

p = dest;

/\* find largest node at left child \*/

target = dest->left;

while (!target->rightThread)

{

p = target;

target = target->right;

}

/\* using replace mode\*/

dest->key = target->key;

}

if (p->key >= target->key)

{

if (target->rightThread && target->leftThread)

{

p->left = target->left;

p->leftThread = true;

}

else if (target->rightThread)

{

Node \*largest = target->left;

while (!largest->rightThread)

{

largest = largest->right;

}

largest->right = p;

p->left = target->left;

}

else

{

Node \*smallest = target->right;

while (!smallest->leftThread)

{

smallest = smallest->left;

}

smallest->left = target->left;

p->left = target->right;

}

}

else

{

if (target->rightThread && target->leftThread)

{

p->right = target->right;

p->rightThread = true;

}

else if (target->rightThread)

{

Node \*largest = target->left;

while (!largest->rightThread)

{

largest = largest->right;

}

largest->right = target->right;

p->right = target->left;

}

else

{

Node \*smallest = target->right;

while (!smallest->leftThread)

{

smallest = smallest->left;

}

smallest->left = p;

p->right = target->right;

}

}

}

/\* Function to print tree \*/

void printTree()

{

Node \*tmp = root, \*p;

for (;;)

{

p = tmp;

tmp = tmp->right;

if (!p->rightThread)

{

while (!tmp->leftThread)

{

tmp = tmp->left;

}

}

if (tmp == root)

break;

cout<<tmp->key<<" ";

}

cout<<endl;

}

};

/\* Main Contains Menu \*/

int main()

{

ThreadedBinarySearchTree tbst;

cout<<"ThreadedBinarySearchTree Test\n";

char ch;

int choice, val;

/\* Perform tree operations \*/

do

{

cout<<"\nThreadedBinarySearchTree Operations\n";

cout<<"1. Insert "<<endl;

cout<<"2. Delete"<<endl;

cout<<"3. Search"<<endl;

cout<<"4. Clear"<<endl;

cout<<"Enter Your Choice: ";

cin>>choice;

switch (choice)

{

case 1 :

cout<<"Enter integer element to insert: ";

cin>>val;

tbst.insert(val);

break;

case 2 :

cout<<"Enter integer element to delete: ";

cin>>val;

tbst.Delete(val);

break;

case 3 :

cout<<"Enter integer element to search: ";

cin>>val;

if (tbst.search(val) == 1)

cout<<"Element "<<val<<" found in the tree"<<endl;

else

cout<<"Element "<<val<<" not found in the tree"<<endl;

break;

case 4 :

cout<<"\nTree Cleared\n";

tbst.makeEmpty();

break;

default :

cout<<"Wrong Entry \n ";

break;

}

/\* Display tree \*/

cout<<"\nTree = ";

tbst.printTree();

cout<<"\nDo you want to continue (Type y or n): ";

cin>>ch;

}

while (ch == 'Y'|| ch == 'y');

return 0;

}

## OUTPUT

## 

## 23. WAP to implement various operations on AVL Tree.

#include<iostream.h>

#include<stdlib.h>

#include<conio.h>

#define TRUE 1

#define FALSE 0

#define NULL 0

class AVL;

class AVLNODE

{

friend class AVL;

private:

int data;

AVLNODE \*left,\*right;

int bf;

};

class AVL

{

private:

AVLNODE \*root;

public:

AVLNODE \*loc,\*par;

AVL()

{

root=NULL;

}

int insert(int);

void displayitem();

void display(AVLNODE \*);

void removeitem(int);

void remove1(AVLNODE \*,AVLNODE \*,int);

void remove2(AVLNODE \*,AVLNODE \*,int);

void search(int x);

void search1(AVLNODE \*,int);

};

int AVL::insert(int x)

{

AVLNODE \*a,\*b,\*c,\*f,\*p,\*q,\*y,\*clchild,\*crchild;

int found,unbalanced;

int d;

if(!root) //special case empty tree

{ y=new AVLNODE;

y->data=x;

root=y;

root->bf=0;

root->left=root->right=NULL;

return TRUE; }

//phase 1:locate insertion point for x.a keeps track of the most

// recent node with balance factor +/-1,and f is the parent of a

// q follows p through the tree.

f=NULL;

a=p=root;

q=NULL;

found=FALSE;

while(p&&!found)

{ //search for insertion point for x

if(p->bf)

{

a=p;

f=q;

}

if(x<p->data) //take left branch

{

q=p;

p=p->left;

}

else if(x>p->data)

{

q=p;

p=p->right;

}

else

{

y=p;

found=TRUE;

}

} //end while

//phase 2:insert and rebalance.x is not in the tree and

// may be inserted as the appropriate child of q.

if(!found)

{

y = new AVLNODE;

y->data=x;

y->left=y->right=NULL;

y->bf=0;

if(x<q->data) //insert as left child

q->left=y;

else

q->right=y; //insert as right child

//adjust balance factors of nodes on path from a to q

//note that by the definition of a,all nodes on this

//path must have balance factors of 0 and so will change

//to +/- d=+1 implies that x is inserted in the left

// subtree of a d=-1 implies

//to that x inserted in the right subtree of a.

if(x>a->data)

{

p=a->right;

b=p;

d=-1;

}

else

{

p=a->left;

b=p;

d=1;

}

while(p!=y)

if(x>p->data) //height of right increases by 1

{

p->bf=-1;

p=p->right;

}

else //height of left increases by 1

{

p->bf=1;

p=p->left;

}

//is tree unbalanced

unbalanced=TRUE;

if(!(a->bf)||!(a->bf+d))

{ //tree still balanced

a->bf+=d;

unbalanced=FALSE;

}

if(unbalanced) //tree unbalanced,determine rotation type

{

if(d==1)

{ //left imbalance

if(b->bf==1) //rotation type LL

{

a->left=b->right;

b->right=a;

a->bf=0;

b->bf=0;

}

else //rotation type LR

{

c=b->right;

b->right=c->left;

a->left=c->right;

c->left=b;

c->right=a;

switch(c->bf)

{

case 1: a->bf=-1; //LR(b)

b->bf=0;

break;

case -1:b->bf=1; //LR(c)

a->bf=0;

break;

case 0: b->bf=0; //LR(a)

a->bf=0;

break;

}

c->bf=0;

b=c; //b is the new root

} //end of LR

} //end of left imbalance

else //right imbalance

{

if(b->bf==-1) //rotation type RR

{

a->right=b->left;

b->left=a;

a->bf=0;

b->bf=0;

}

else //rotation type LR

{

c=b->right;

b->right=c->left;

a->right=c->left;

c->right=b;

c->left=a;

switch(c->bf)

{

case 1: a->bf=-1; //LR(b)

b->bf=0;

break;

case -1:b->bf=1; //LR(c)

a->bf=0;

break;

case 0: b->bf=0; //LR(a)

a->bf=0;

break;

}

c->bf=0;

b=c; //b is the new root

} //end of LR

}

//subtree with root b has been rebalanced and is the new subtree

if(!f)

root=b;

else if(a==f->left)

f->left=b;

else if(a==f->right)

f->right=b;

} //end of if unbalanced

return TRUE;

} //end of if(!found)

return FALSE;

} //end of AVL INSERTION

void AVL::displayitem()

{

display(root);

}

void AVL::display(AVLNODE \*temp)

{

if(temp==NULL)

return;

cout<<temp->data<<" ";

display(temp->left);

display(temp->right);

}

void AVL::removeitem(int x)

{

search(x);

if(loc==NULL)

{

cout<<"\nitem is not in tree";

return;

}

if(loc->right!=NULL&&loc->left!=NULL)

remove1(loc,par,x);

else

remove2(loc,par,x);

}

void AVL::remove1(AVLNODE \*l,AVLNODE \*p,int x)

{

AVLNODE \*ptr,\*save,\*suc,\*psuc;

ptr=l->right;

save=l;

while(ptr->left!=NULL)

{

save=ptr;

ptr=ptr->left;

}

suc=ptr;

psuc=save;

remove2(suc,psuc,x);

if(p!=NULL)

if(l==p->left)

p->left=suc;

else

p->right=suc;

else

root=l;

suc->left=l->left;

suc->right=l->right;

return;

}

void AVL::remove2(AVLNODE \*s,AVLNODE \*p,int x)

{

AVLNODE \*child;

if(s->left==NULL && s->right==NULL)

child=NULL;

else if(s->left!=NULL)

child=s->left;

else

child=s->right;

if(p!=NULL)

if(s==p->left)

p->left=child;

else

p->right=child;

else

root=child;

}

void AVL::search(int x)

{

search1(root,x);

}

void AVL::search1(AVLNODE \*temp,int x)

{

AVLNODE \*ptr,\*save;

int flag;

if(temp==NULL)

{

cout<<"\nthe tree is empty";

return;

}

if(temp->data==x)

{

cout<<"\nthe item is root and is found";

par=NULL;

loc=temp;

par->left=NULL;

par->right=NULL;

return; }

if( x < temp->data)

{

ptr=temp->left;

save=temp;

}

else

{

ptr=temp->right;

save=temp;

}

while(ptr!=NULL)

{

if(x==ptr->data)

{ flag=1;

cout<<"\nitemfound";

loc=ptr;

par=save;

}

if(x<ptr->data)

ptr=ptr->left;

else

ptr=ptr->right;

}

if(flag!=1)

{

cout<<"item is not there in tree";

loc=NULL;

par=NULL;

cout<<loc;

cout<<par;

}

}

main()

{

clrscr();

AVL a;

int x,y,c;

char ch;

do

{

cout<<"\n1.insert";

cout<<"\n2.display";

cout<<"\n3.delete";

cout<<"\n4.search";

cout<<"\n5.exit";

cout<<"\nEnter u r choice to perform on AVL tree";

cin>>c;

switch(c)

{

case 1:cout<<"\nEnter an element to insert into tree";

cin>>x;

a.insert(x);

break;

case 2:a.displayitem(); break;

case 3:cout<<"\nEnter an item to deletion";

cin>>y;

a.removeitem(y);

break;

case 4:cout<<"\nEnter an element to search";

cin>>c;

a.search(c);

break;

case 5:exit(0); break;

default :cout<<"\nInvalid option try again";

}

cout<<"\ndo u want to continue";

cin>>ch;

}

while(ch=='y'||ch=='Y');

}

# OUTPUT

