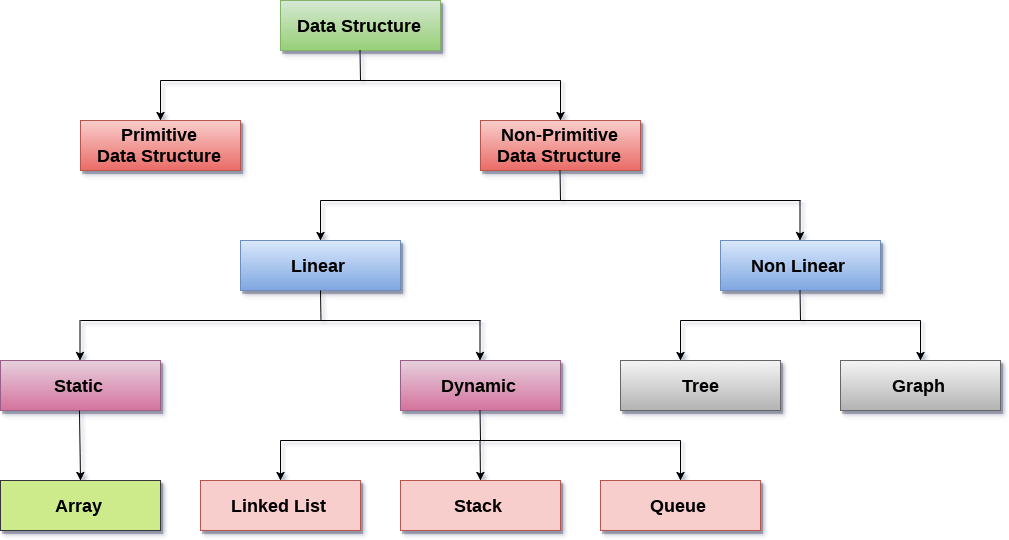
**11TH DEC,2021**



**DATA STRUCTURE PRACTICAL FILE**

**Name: Neha kumari**

**Roll no:2020331**

**Submitted to: Tina ma’am**

**DATA STRUCTURE PRACTICAL LIST QUESTIONS**

**Q1. Given a list of N elements, which follows no particular arrangement, you are required to search an element x in the list. The list is stored using array data structure. If the search is successful, the output should be the index at which the element occurs, otherwise returns -1 to indicate that the element is not present in the list. Assume that the elements of the list are all distinct. Write a program to perform the desired task.**

**Solution:**

**#include <iostream>**

**using namespace std;**

**template <class T>**

**class MyArray**

**{**

**T a[20];**

**int SIZE;**

**public:**

**void input();**

**int linearSearch(T);**

**void display();**

**};**

**template <class T>**

**void MyArray<T>::input()**

**{**

**cout<<"ENTER THE NUMBER OF ELEMENTS (MAX=20)"<<endl;**

**cin>>SIZE;**

**if(SIZE>20)**

**SIZE=20;**

**cout<<"ENTER THE ELEMENTS IN THE ARRAY"<<endl;**

**for(int i=0;i<SIZE;i++)**

**cin>>a[i];**

**}**

**template <class T>**

**int MyArray<T>::linearSearch(T key)**

**{**

**for(int i=0;i<SIZE;i++)**

**{**

**if(key==a[i])**

**return i;**

**}**

**return -1;**

**}**

**template <class T>**

**void MyArray<T>::display()**

**{**

**cout<<"THE ARRAY IS : ";**

**for(int i=0;i<SIZE;i++)**

**cout<<a[i]<<"\t";**

**cout<<endl;**

**}**

**int main()**

**{**

**MyArray<int> obj1;**

**int k,choice,isfound;**

**char ch='y';**

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

**{**

**cout<<"MAIN MENU"<<endl;**

**cout<<"1.INPUT"<<endl;**

**cout<<"2.SEARCH AN ELEMENT USING LINEAR SERACH"<<endl;**

**cout<<"3.DISPLAY THE ARRAY"<<endl;**

**cout<<"ENTER YOUR CHOICE:-"<<endl;**

**cin>>choice;**

**switch(choice)**

**{**

**case 1:cout<<"---------------------------------------------------\n";**

**obj1.input();**

**cout<<"------------------------------------------------------\n";**

**break;**

**case 2:cout<<"----------------------------------------------------------\n";**

**cout<<"ENTER THE ELEMENT TO BE SEARCHED"<<endl;**

**cin>>k;**

**isfound=obj1.linearSearch(k);**

**if(isfound==-1)**

**cout<<"ELEMENT IS NOT FOUND"<<endl;**

**else**

**cout<<"SEARCH SUCCESSFUL ELEMENT FOUND AT INDEX "<<isfound<<endl;**

**cout<<"------------------------------------------------------\n";**

**break;**

**case 3:cout<<"-----------------------------------------------------------\n";**

**obj1.display();**

**cout<<"--------------------------------------------------------------\n";**

**break;**

**default:**

**cout<<"ERROR IN INPUT"<<endl;**

**}**

**cout<<"DO YOU WANT TO CONTINUE?"<<endl;**

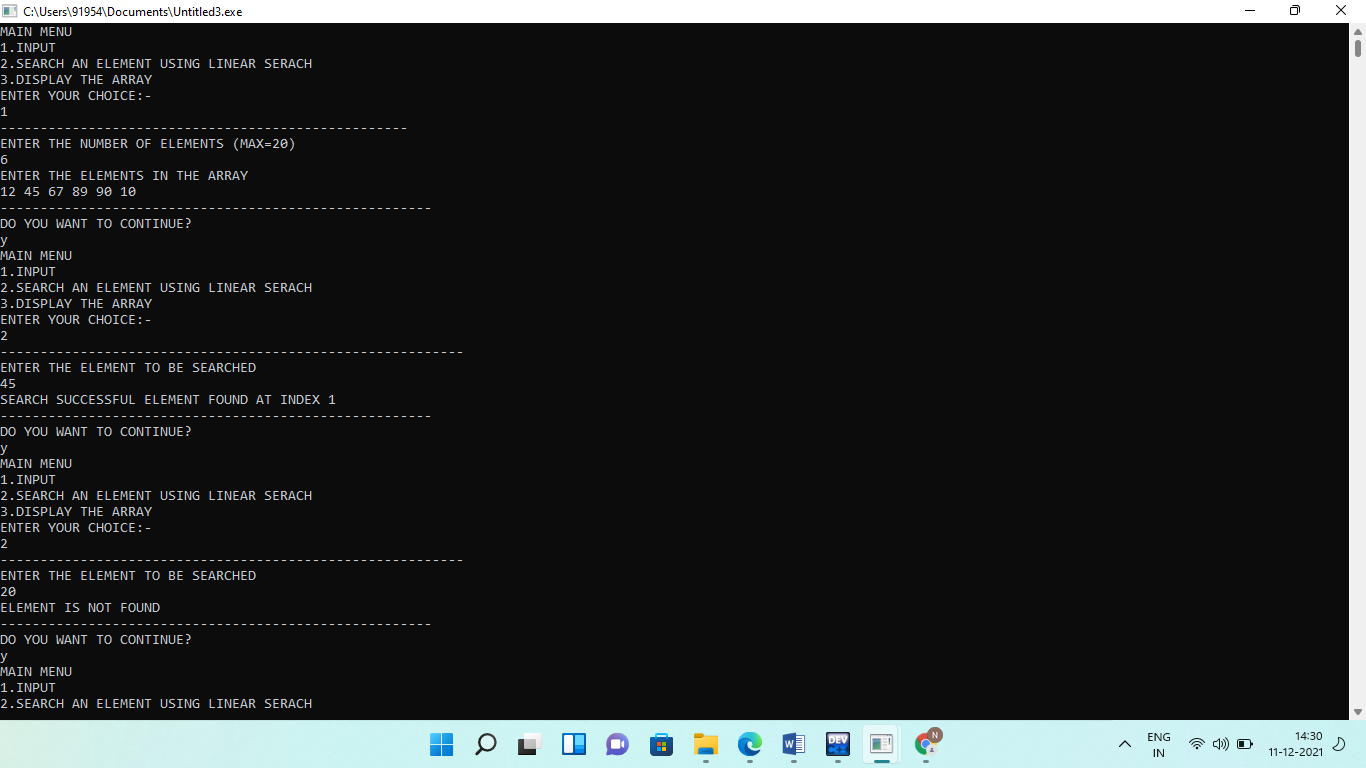
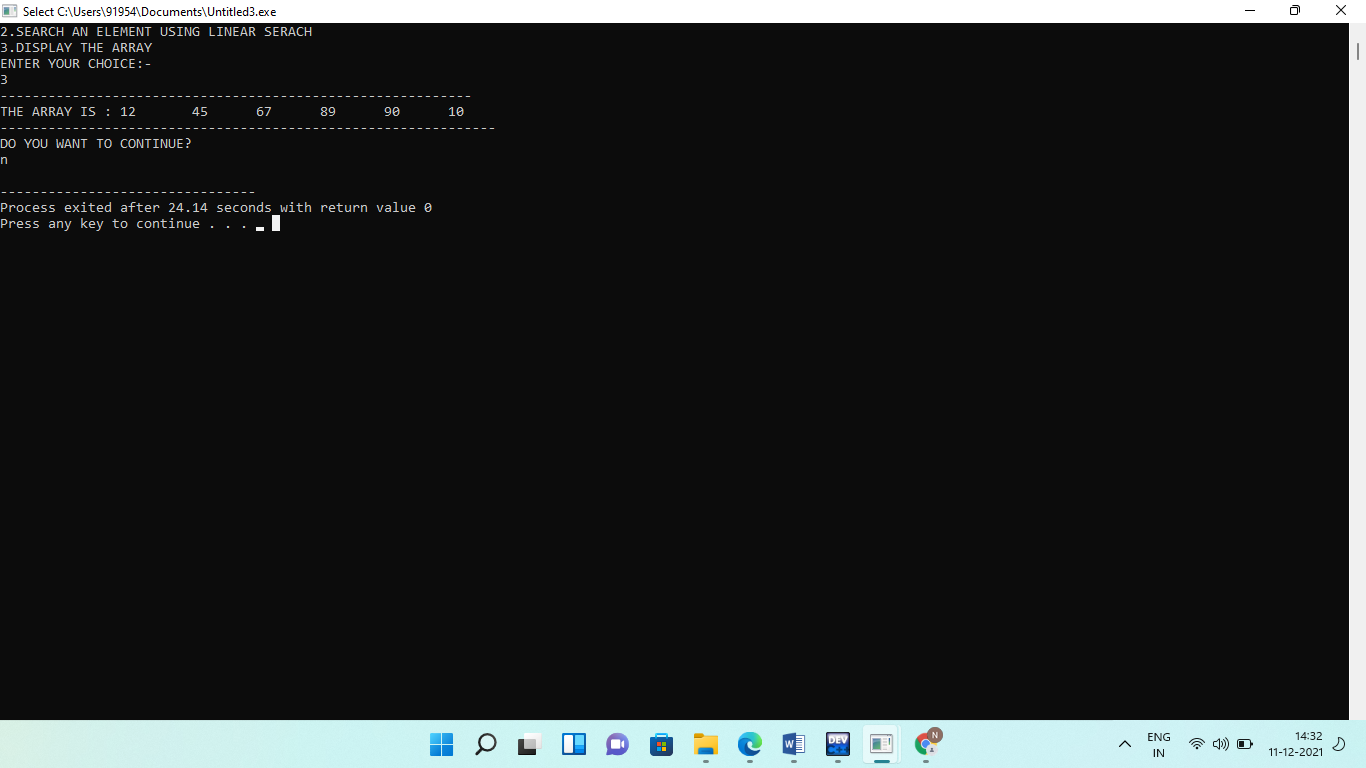
**cin>>ch;**

**}**

**return 0;**

**}**

**Output:**



**Q2.Given a list of N elements, which is sorted in ascending order, you are required to search an element x in the list. The list is stored using array data structure. If the search is successful, the output should be the index at which the element occurs, otherwise returns -1 to indicate that the element is not present in the list. Assume that the elements of the list are all distinct. Write a program to perform the desired task.**

**Solution:**

**#include <iostream>**

**using namespace std;**

**template <class T>**

**class MyArray**

**{**

**T a[20];**

**int SIZE;**

**public:**

**void input();**

**int binarySearch(T);**

**void display();**

**};**

**template <class T>**

**void MyArray<T>::input()**

**{**

**cout<<"ENTER THE NUMBER OF ELEMENTS (MAX=20)"<<endl;**

**cin>>SIZE;**

**if(SIZE>20)**

**SIZE=20;**

**cout<<"ENTER THE ELEMENTS IN THE ARRAY"<<endl;**

**for(int i=0;i<SIZE;i++)**

**cin>>a[i];**

**}**

**template <class T>**

**int MyArray<T>::binarySearch(T key)**

**{**

**int minimum=0;**

**int maximum=SIZE-1;**

**int mid;**

**while(minimum<=maximum)**

**{**

**mid=(minimum+maximum)/2;**

**if(key==a[mid])**

**return mid;**

**else if(key>a[mid])**

**minimum=mid+1;**

**else if(key<a[mid])**

**maximum=mid-1;**

**}**

**return -1;**

**}**

**template <class T>**

**void MyArray<T>::display()**

**{**

**cout<<"THE ARRAY IS : ";**

**for(int i=0;i<SIZE;i++)**

**cout<<a[i]<<"\t";**

**cout<<endl;**

**}**

**int main()**

**{**

**MyArray<int> obj1;**

**int k,choice,isfound;**

**char ch='y';**

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

**{**

**cout<<"MAIN MENU"<<endl;**

**cout<<"1.INPUT"<<endl;**

**cout<<"2.SEARCH AN ELEMENT USING BINARY SERACH"<<endl;**

**cout<<"3.DISPLAY THE ARRAY"<<endl;**

**cout<<"ENTER YOUR CHOICE"<<endl;**

**cin>>choice;**

**switch(choice)**

**{**

**case 1:cout<<"-----------------------------------------------------------\n";**

**obj1.input();**

**cout<<"-----------------------------------------------------------\n";**

**break;**

**case 2:cout<<"-----------------------------------------------------------\n";**

**cout<<"ENTER THE ELEMENT TO BE SEARCHED"<<endl;**

**cin>>k;**

**isfound=obj1.binarySearch(k);**

**if(isfound==-1)**

**cout<<"ELEMENT IS NOT FOUND"<<endl;**

**else**

**cout<<"SEARCH SUCCESSFUL ELEMENT FOUND AT INDEX "<<isfound<<endl;**

**cout<<"-----------------------------------------------------------\n";**

**break;**

**case 3:cout<<"-----------------------------------------------------------\n";**

**obj1.display();**

**cout<<"-----------------------------------------------------------\n";**

**break;**

**default:**

**cout<<"ERROR IN INPUT"<<endl;**

**}**

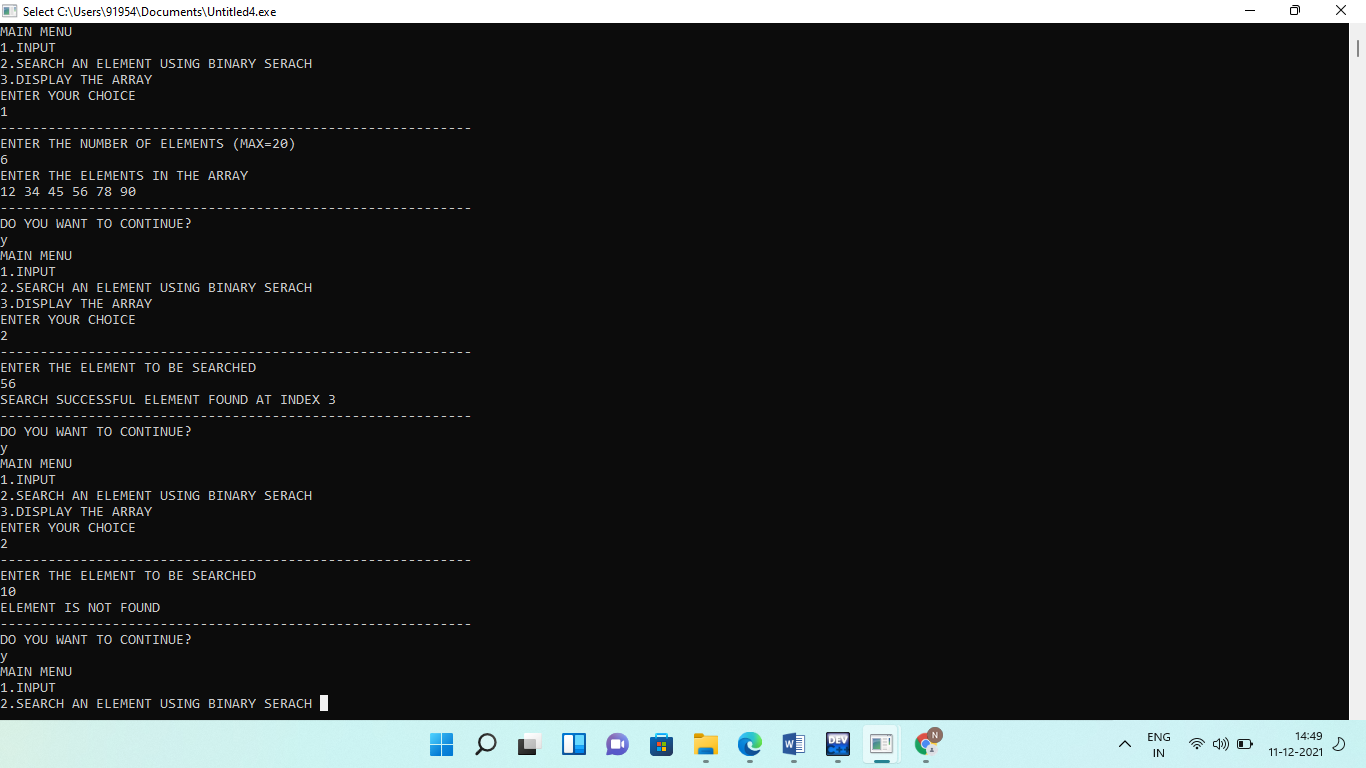
**cout<<"DO YOU WANT TO CONTINUE?"<<endl;**

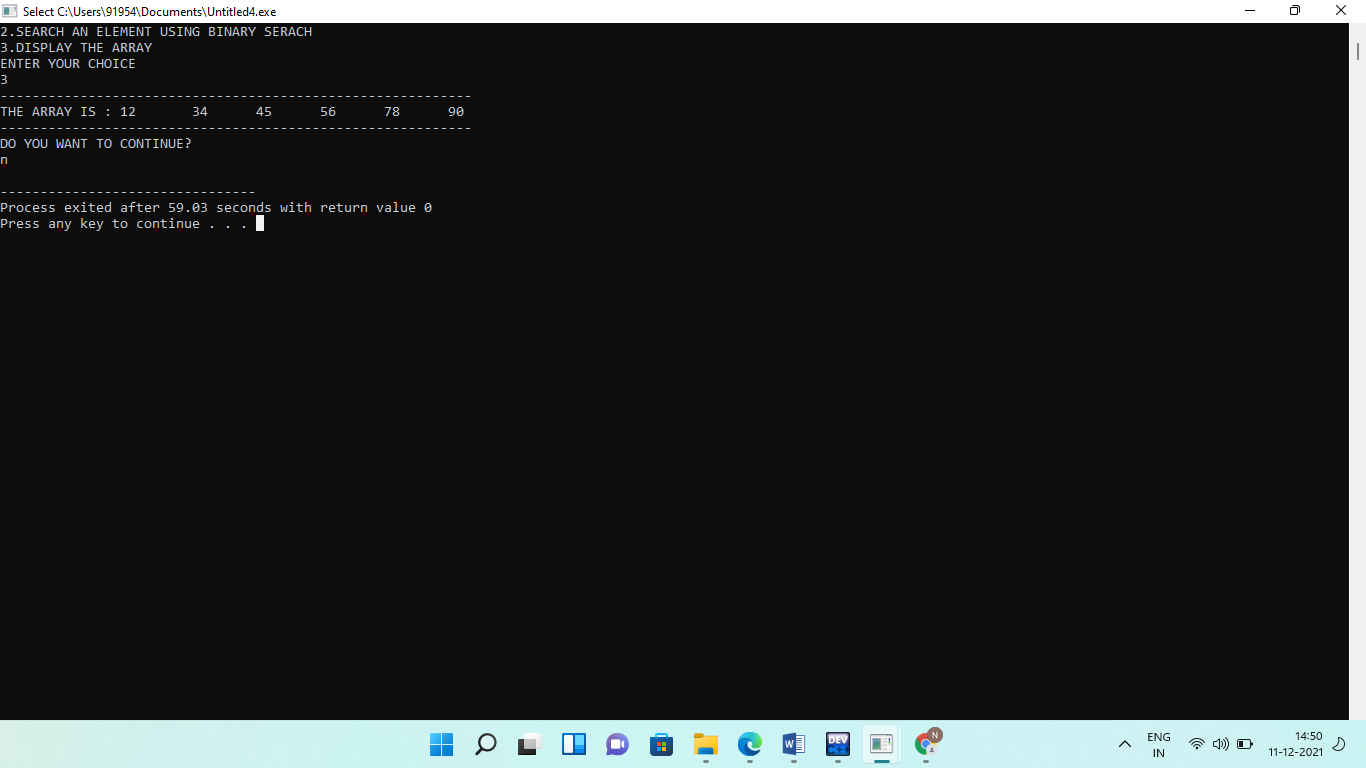
**cin>>ch;**

**}**

**return 0;**

**}**

**Output:**



**Q3.Write a program to implement singly linked list which supports the following operations:**

1. **Insert an element x at the beginning of the singly linked list.**
2. **Insert an element x at position in the singly linked list.**
3. **Remove an element from the beginning of the singly linked list.**
4. **Remove an element from position in the singly linked list.**
5. **Search for an element x in the singly linked list and return its pointer.**
6. **Concatenate two singly linked lists.**

**Solution:**

**# include <iostream>**

**using namespace std;**

**template<class t>**

**struct node**

**{**

**int info;**

**node<t> \*next;**

**};**

**template<class t>**

**class linkedtype{**

**node<t>\*start;**

**public :**

**linkedtype(){**

**start =NULL;**

**}**

**node<t> \*createnewnode(int);**

**void createnewlist(node<t>\*);**

**void addatend(node<t>\*);**

**void addafter(node<t>\*);**

**void addbefore(node<t>\*);**

**void add\_beg(node<t>\*);**

**void deletion\_at\_beg();**

**void deletion\_at\_end();**

**void deletion();**

**int count();**

**void search();**

**void display();**

**void add\_ith\_pos(node<t>\*,int);**

**void delete\_ith\_pos(int);**

**void reverse();**

**void concatenate(linkedtype<t> l1);**

**};**

**template<class t>**

**node<t>\*linkedtype<t>:: createnewnode(int x)**

**{**

**node<t>\*newptr;**

**newptr=new node<t>;**

**newptr->info=x;**

**newptr->next=NULL;**

**return(newptr);**

**}**

**template<class t>**

**void linkedtype<t>::createnewlist(node<t> \*newptr)**

**{**

**node<t>\* temp;**

**temp=start;**

**start=newptr;**

**newptr->next=temp;**

**}**

**template<class t>**

**void linkedtype<t>::add\_beg(node<t>\* nptr)**

**{**

**node<t> \*temp;**

**temp=start;**

**start=nptr;**

**nptr->next=temp;**

**}**

**template<class t>**

**void linkedtype<t>:: addafter(node<t>\* nptr)**

**{**

**int after;**

**node<t>\* temp;**

**cout<<"Enter the info part of the node after which you want to add ";**

**cin>>after;**

**for(temp=start; temp!=NULL; temp=temp->next)**

**{**

**if(temp->info==after)**

**{**

**nptr->next=temp->next;**

**temp->next=nptr;**

**}**

**}**

**}**

**template<class t>**

**void linkedtype<t>:: addbefore(node<t>\* nptr)**

**{**

**node<t>\* temp;**

**int before;**

**cout<<"Enter the info part of the node before which you want to add ";**

**cin>>before;**

**if(start->info==before)**

**{**

**nptr->next=start;**

**start=nptr;**

**return;**

**}**

**else{**

**for(temp=start; temp->next!=NULL; temp=temp->next)**

**{**

**if(temp->next->info==before)**

**{**

**nptr->next=temp->next;**

**temp->next=nptr;**

**break;**

**}**

**}**

**}**

**}**

**template<class t>**

**void linkedtype<t>:: addatend(node<t>\* nptr)**

**{**

**node<t> \*temp;**

**if(start==NULL)**

**{**

**start=nptr;**

**return;**

**}**

**for(temp=start; temp->next!=NULL; temp=temp->next)**

**{**

**}**

**temp->next=nptr;**

**}**

**template<class t>**

**void linkedtype<t>::add\_ith\_pos(node<t>\*nptr,int n)**

**{**

**node<t>\* temp;**

**if(n==1)**

**{**

**nptr->next=start;**

**start=nptr;**

**return;**

**}**

**else**

**{**

**temp=start;**

**for(int i=0;i<n-2;i++)**

**{**

**temp=temp->next;**

**}**

**nptr->next=temp->next;**

**temp->next=nptr;**

**}**

**}**

**template<class t>**

**void linkedtype<t>::delete\_ith\_pos(int n)**

**{**

**node<t>\*temp,\*ptr;**

**temp=start;**

**if(n==1)**

**{**

**start=temp->next;**

**delete temp;**

**return;**

**}**

**else**

**{**

**for(int i=0;i<n-2;i++)**

**temp=temp->next;**

**ptr=temp->next;**

**temp->next=ptr->next;**

**delete(ptr);**

**}**

**}**

**template<class t>**

**void linkedtype<t>::deletion\_at\_end()**

**{**

**node<t>\*temp,\*ptr;**

**temp = start;**

**while(temp->next != NULL)**

**{**

**ptr = temp;**

**temp = temp ->next;**

**}**

**ptr->next = NULL;**

**delete(temp);**

**}**

**template<class t>**

**void linkedtype<t>::deletion\_at\_beg()**

**{**

**node<t>\* temp,\*ptr;**

**temp=start;**

**ptr=temp->next;**

**start=ptr;**

**delete(temp);**

**}**

**template<class t>**

**void linkedtype<t>:: deletion()**

**{**

**int del;**

**cout<<"Enter the info part of the node to be deleted ";**

**cin>>del;**

**node<t>\* temp;**

**for(temp=start; temp->next!=NULL; temp=temp->next)**

**{**

**if(temp->next->info==del)**

**{**

**node<t>\* ptr=temp->next;**

**temp->next=temp->next->next;**

**delete(ptr);**

**break;**

**}**

**}**

**}**

**template<class t>**

**int linkedtype<t>:: count()**

**{**

**node<t>\* temp;**

**int c1=0;**

**for(temp=start; temp!=NULL; temp=temp->next)**

**{**

**c1++;**

**}**

**return c1;**

**}**

**template<class t>**

**void linkedtype<t>:: display()**

**{**

**if(start==NULL)**

**{**

**cout<<"Linked List is empty ";**

**return;**

**}**

**node<t>\* temp;**

**cout<<"Contents of the linked list are :"<<endl;**

**for(temp=start; temp!=NULL; temp=temp->next)**

**{**

**cout<<temp->info<<" ";**

**}**

**cout<<endl;**

**}**

**template<class t>**

**void linkedtype<t>::search()**

**{**

**node<t> \*ptr;**

**int item,i=0,flag;**

**ptr = start;**

**if(ptr == NULL)**

**{**

**cout<<"\nEmpty List";**

**}**

**else**

**{**

**cout<<"\nEnter item which you want to search?";**

**cin>>item;**

**while (ptr!=NULL)**

**{**

**if(ptr->info == item)**

**{**

**cout<<"\nitem found at location "<<i+1;**

**flag=0;**

**}**

**else**

**{**

**flag=1;**

**}**

**i++;**

**ptr = ptr -> next;**

**}**

**if(flag==1)**

**{**

**cout<<"\n Item not found";**

**}**

**}**

**}**

**template<class t>**

**void linkedtype<t>::reverse()**

**{**

**node<t> \*current,\*prev,\*Next;**

**current=start;**

**prev=NULL;**

**while(current!=NULL)**

**{**

**Next=current->next;**

**current->next=prev;**

**prev=current;**

**current=Next;**

**}**

**start=prev;**

**}**

**template<class t>**

**void linkedtype<t>::concatenate(linkedtype<t> l1)**

**{**

**node <t>\*tmp = new node<t>;**

**tmp=(\*this).start;**

**while(tmp->next!=NULL)**

**tmp=tmp->next;**

**tmp->next=l1.start;**

**}**

**int main()**

**{**

**int choice,in,n,c1;**

**node<int>\* nptr,\*ptr;**

**linkedtype<int> l,l1;**

**char c='y';**

**cout<<"Creating new Linked list. Enter no of nodes ";**

**cin>>n;**

**for(int i=n-1; i>=0; i--)**

**{**

**cout<<"Enter info for node numbered "<<i+1<<endl;**

**cin>>in;**

**nptr=l.createnewnode(in);**

**if(nptr==NULL)**

**{**

**cout<<"Compiler out of memory "<<endl;**

**exit(0);**

**}**

**else**

**l.createnewlist(nptr);**

**}**

**while(c=='y' || c=='Y')**

**{**

**cout<<"Main menu"<<endl;**

**cout<<"\n1. Add a node before ";**

**cout<<"\n2. Add a node after ";**

**cout<<"\n3. Add a node at last" ;**

**cout<<"\n4. Delete a node ";**

**cout<<"\n5. Count the total no of nodes ";**

**cout<<"\n6. Display the linked list";**

**cout<<"\n 7. Add a node at begining:";**

**cout<<"\n 8. delete at beg:";**

**cout<<"\n 9. delete at end:";**

**cout<<"\n 10.search:";**

**cout<<"\n 11.add at ith:";**

**cout<<"\n 12.delete at ith";**

**cout<<"\n 13.reverse";**

**cout<<"\n 14.concatenate"<<endl;;**

**cout<<"Enter your choice(1-14):- ";**

**cin>>choice;**

**switch(choice)**

**{**

**case 1:cout<<"===========================================================================\n";**

**cout<<"Enter the info part of the node to be added ";**

**cin>>in;**

**nptr=l.createnewnode(in);**

**if(nptr==NULL){**

**cout<<"Compiler out of memory";**

**exit(0);**

**}**

**else**

**{**

**l.addbefore(nptr);**

**l.display();**

**}**

**cout<<"=====================================================================================\n";**

**break;**

**case 2: cout<<"=============================================================================\n";**

**cout<<"Enter the info part";**

**cin>>in;**

**nptr=l.createnewnode(in);**

**if(nptr==NULL)**

**{**

**cout<<"Compiler out of memory "<<endl;**

**exit(0);**

**}**

**else**

**{**

**l.addafter(nptr);**

**l.display();**

**}**

**cout<<"==================================================================\n";**

**break;**

**case 3: cout<<"==================================================================\n";**

**cout<<"Enter the info part";**

**cin>>in;**

**nptr=l.createnewnode(in);**

**if(nptr==NULL)**

**{**

**cout<<"Compiler out of memory "<<endl;**

**exit(0);**

**}**

**else**

**{**

**l.addatend(nptr);**

**l.display();**

**}**

**cout<<"==================================================================\n";**

**break;**

**case 4: cout<<"==================================================================\n";**

**l.deletion();**

**cout<<"After delete a node:\n";**

**l.display();**

**cout<<"==================================================================\n";**

**break;**

**case 5:cout<<"==================================================================\n";**

**c1=l.count();**

**cout<<"Total no of nodes in a linkedlist are : "<<c1<<endl;**

**cout<<"==================================================================\n";**

**break;**

**case 6: cout<<"==================================================================\n";**

**l.display();**

**cout<<"==================================================================\n";**

**break;**

**case 7: cout<<"==================================================================\n";**

**cout<<"Enter the info part of the node to be added ";**

**cin>>in;**

**nptr=l.createnewnode(in);**

**if(nptr==NULL){**

**cout<<"Compiler out of memory";**

**exit(0);**

**}**

**else**

**{**

**l.add\_beg(nptr);**

**l.display();**

**cout<<"==================================================================\n";**

**}**

**break;**

**case 8:cout<<"==================================================================\n";**

**l.deletion\_at\_beg();**

**l.display();**

**cout<<"==================================================================\n";**

**break;**

**case 9:cout<<"==================================================================\n";**

**l.deletion\_at\_end();**

**l.display();**

**cout<<"==================================================================\n";**

**break;**

**case 10: cout<<"==================================================================\n";**

**l.search();**

**cout<<"==================================================================\n";**

**break;**

**case 11: cout<<"==================================================================\n";**

**cout<<"Enter the info part of the node to be added ";**

**cin>>in;**

**nptr=l.createnewnode(in);**

**cout<<"\nenter the position:";**

**cin>>n;**

**if(nptr==NULL){**

**cout<<"Compiler out of memory";**

**exit(0);**

**}**

**else**

**{**

**l.add\_ith\_pos(nptr,n);**

**l.display();**

**cout<<"==================================================================\n";**

**}**

**break;**

**case 12: cout<<"==================================================================\n";**

**cout<<"\nenter the position:";**

**cin>>n;**

**l.delete\_ith\_pos(n);**

**l.display();**

**cout<<"==================================================================\n";**

**break;**

**case 13:cout<<"==================================================================\n";**

**l.reverse();**

**l.display();**

**cout<<"==================================================================\n";**

**break;**

**case 14:**

**cout<<"==================================================================\n";**

**cout<<"Enter another list to concatenate";**

**int i;**

**cout<<"Creating new Linked list. Enter no of nodes ";**

**cin>>n;**

**for(int i=n-1; i>=0; i--)**

**{**

**cout<<"Enter info for node numbered "<<i+1<<endl;**

**cin>>in;**

**nptr=l1.createnewnode(in);**

**if(nptr==NULL)**

**{**

**cout<<"Compiler out of memory "<<endl;**

**exit(0);**

**}**

**else**

**l1.createnewlist(nptr);**

**}**

**l.concatenate(l1);**

**l.display();**

**cout<<"==================================================================\n";**

**break;**

**default: cout<<"Error in input"<<endl;**

**}**

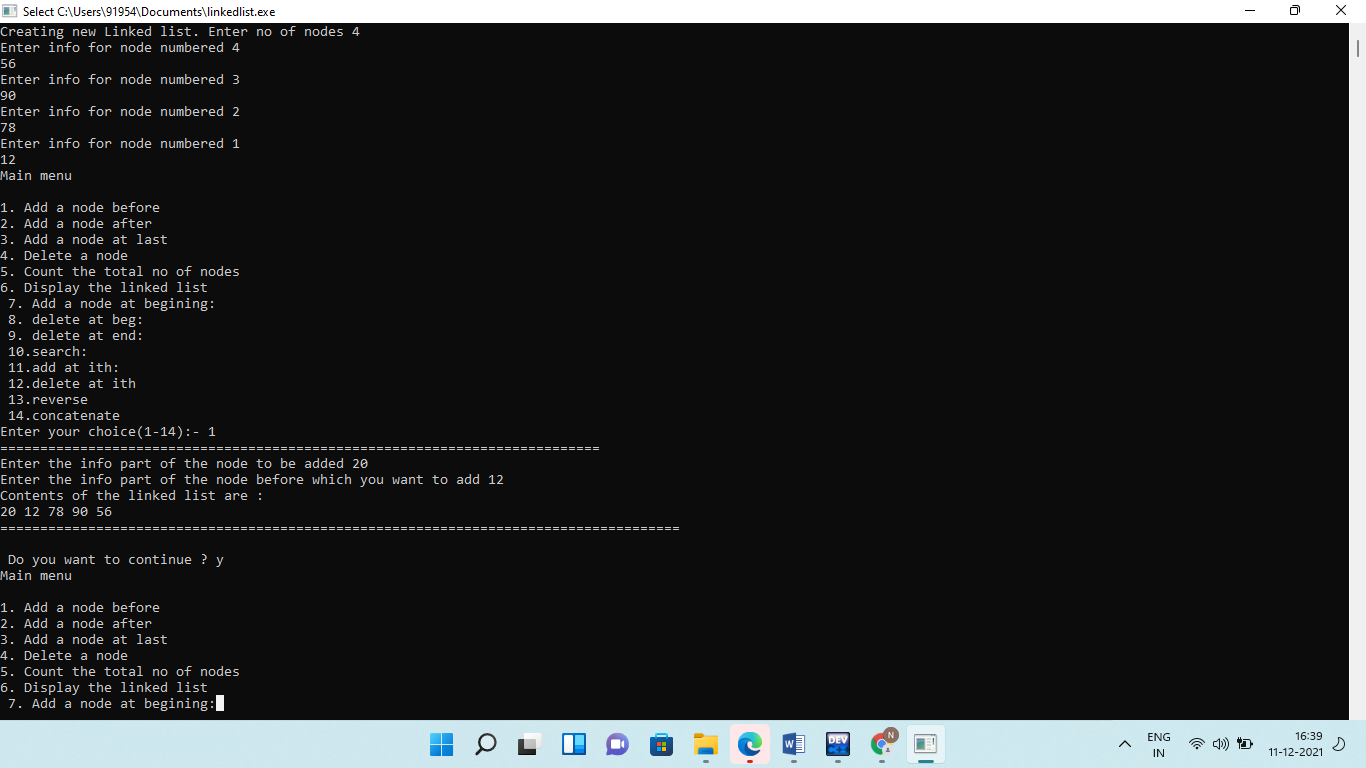
**cout<<"\n Do you want to continue ? ";**

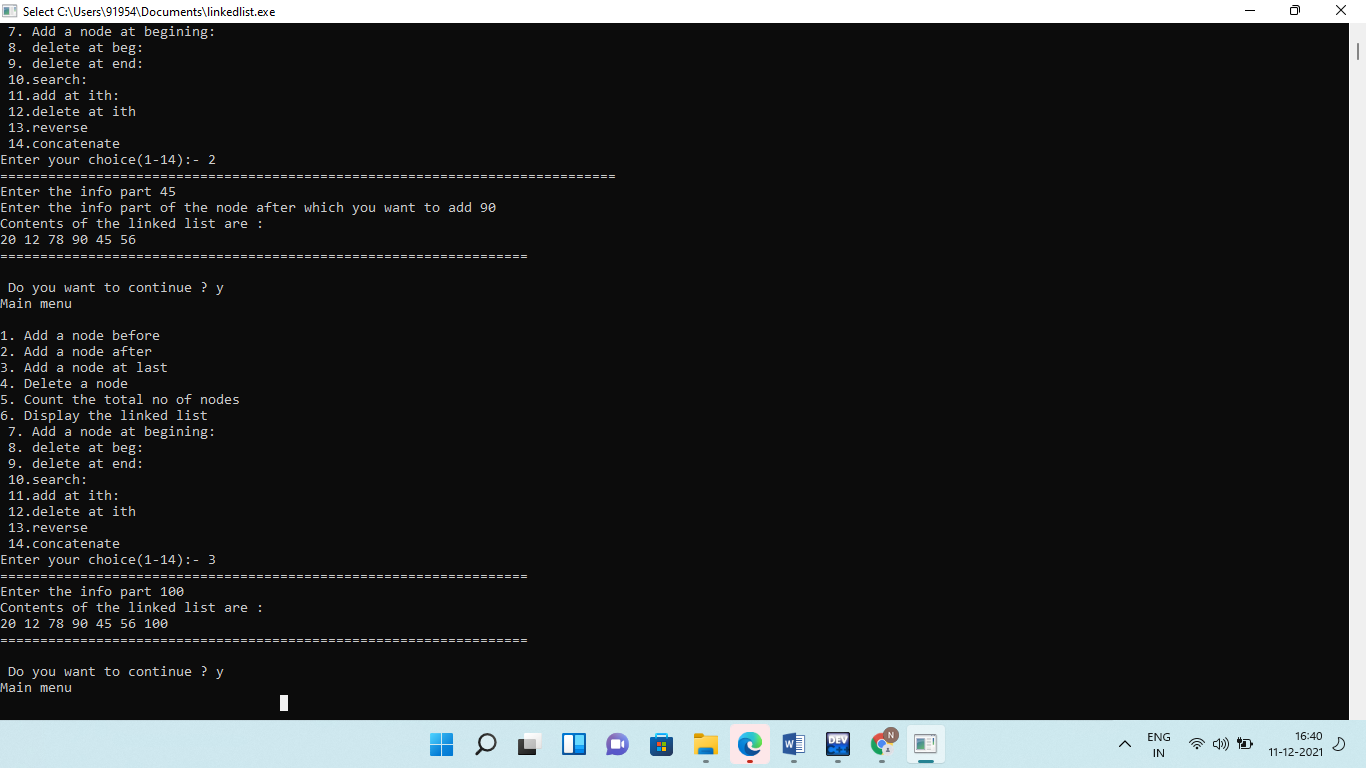
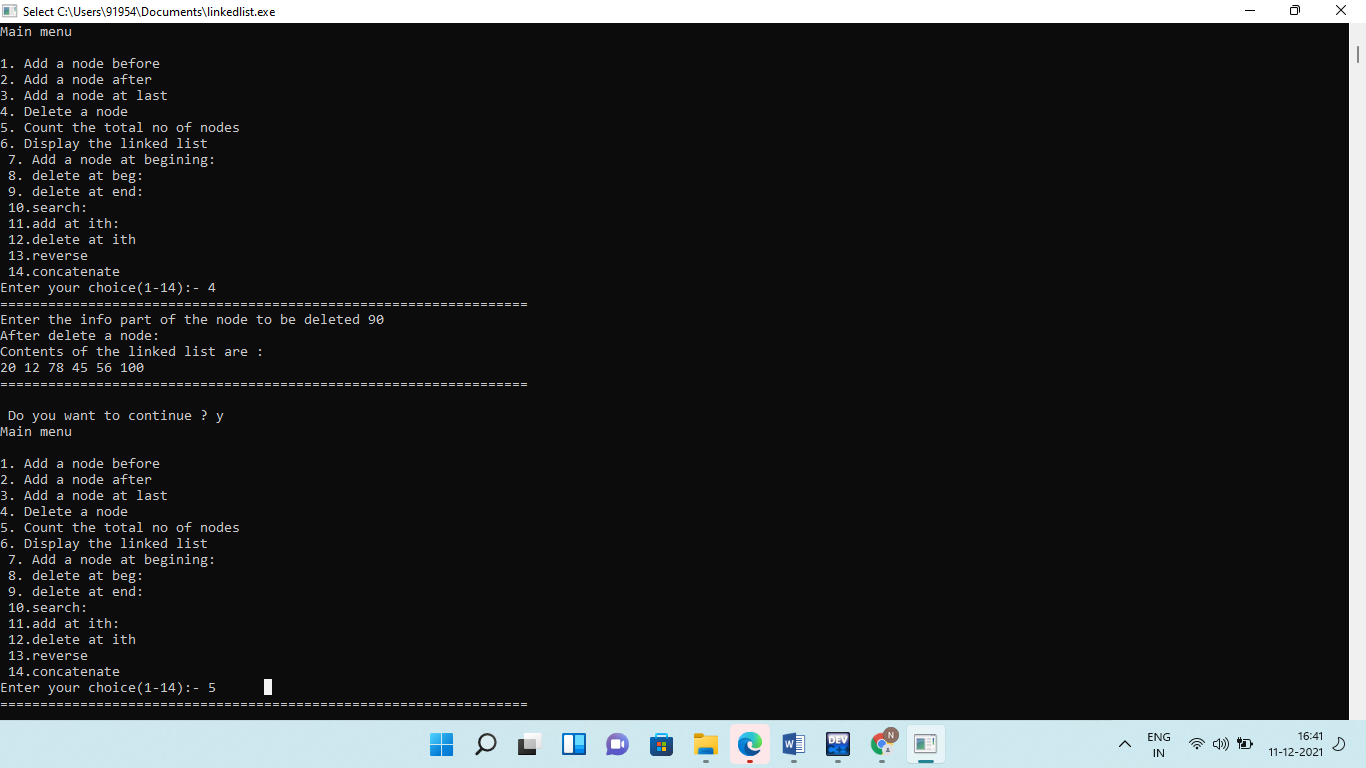
**cin>>c;**

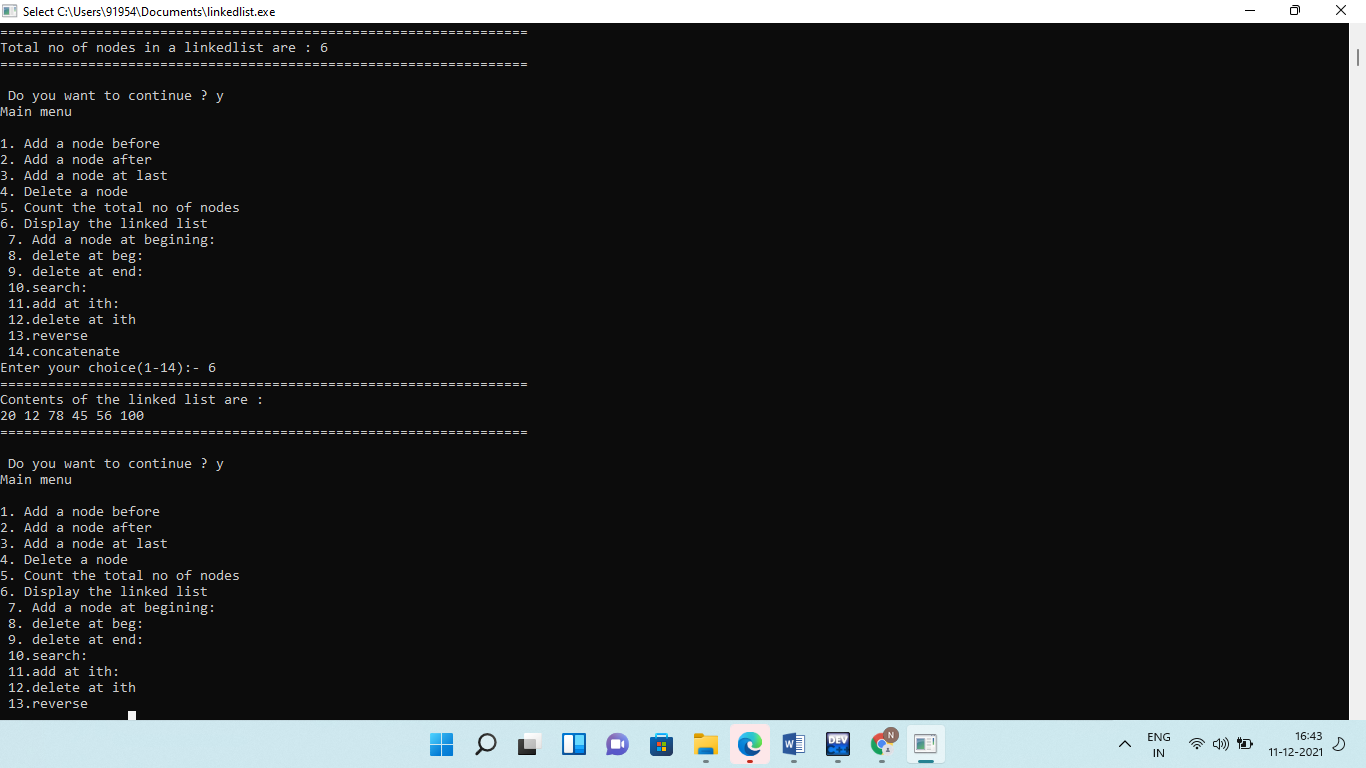
**}**

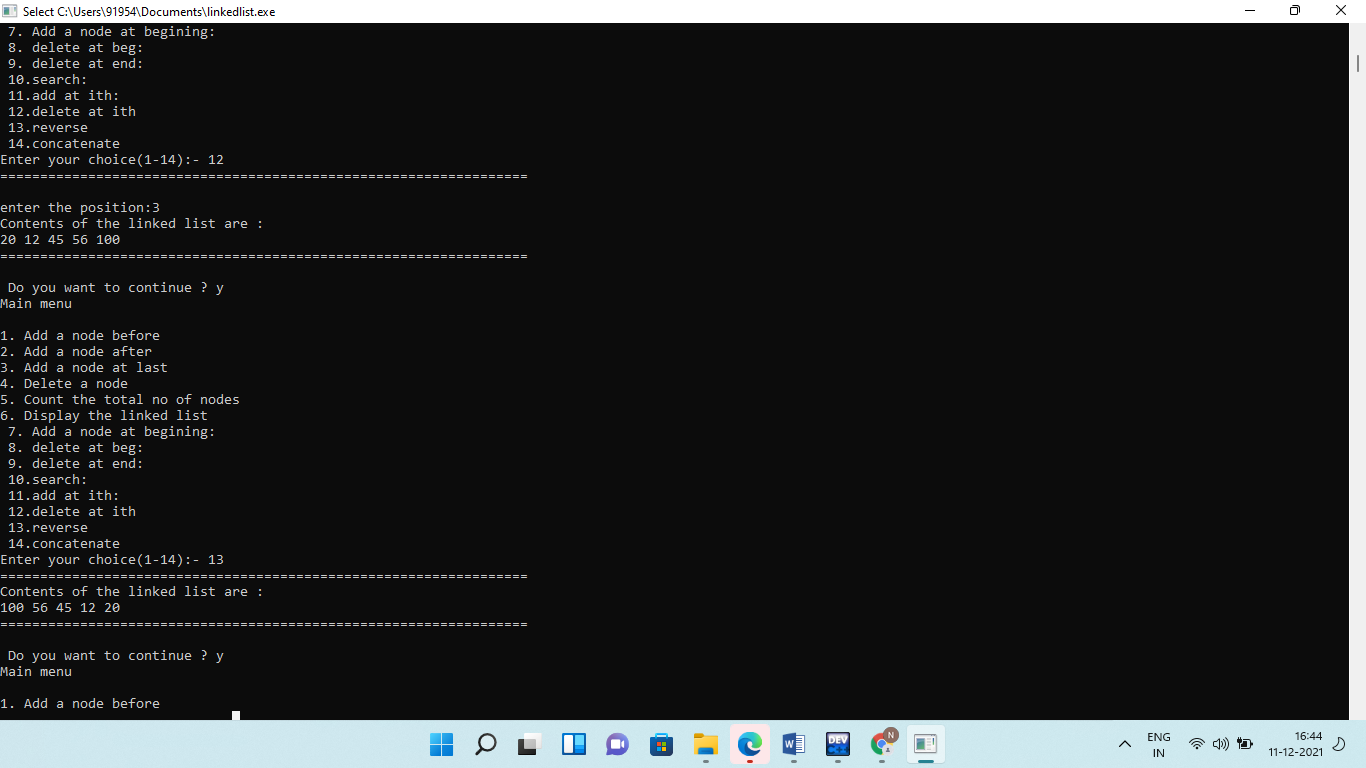
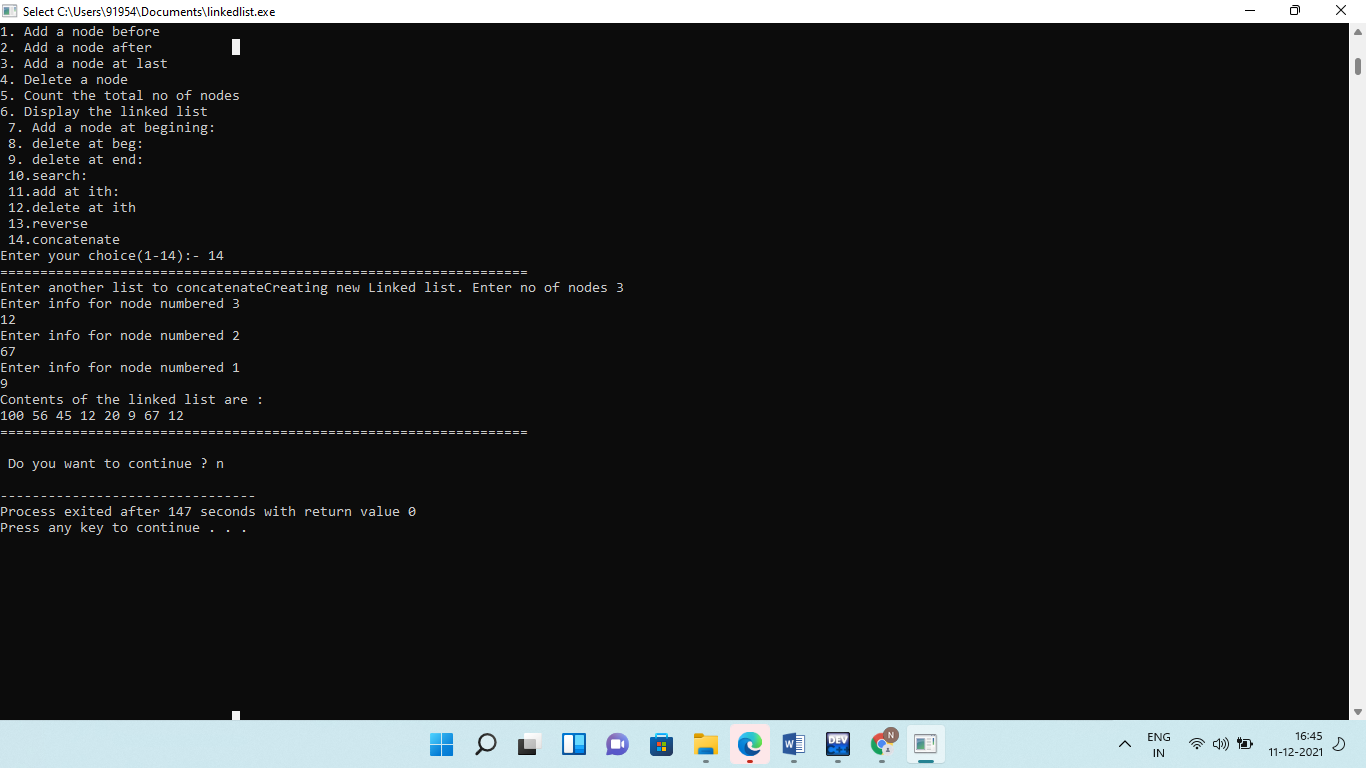
**return 0;**

**}**

**Output:**







**Q4.** **Write a program to implement doubly linked list which supports the following operations:**

**(i) Insert an element x at the beginning of the doubly linked list**

**(ii) Insert an element x at ith position in the doubly linked list**

**(iii)Insert an element x at the end of the doubly linked list**

**(iv) Remove an element from the beginning of the doubly linked list**

**(v) Remove an element from ith position in the doubly linked list.**

**(vi) Remove an element from the end of the doubly linked list**

**(vii) Search for an element x in the doubly linked list and return its pointer**

**(viii) Concatenate two doubly linked lists**

**Solution:**

**#include <iostream>**

**using namespace std;**

**template<class T>class node**

**{**

**public:**

**T info;**

**node<T> \*next,\*prev;**

**node()**

**{**

**prev = 0;**

**next = 0;**

**}**

**node(T x, node<T> \*n = 0, node<T> \*p = 0)**

**{**

**info = x;**

**next = n;**

**prev = p;**

**}**

**};**

**template<class T>class dllist**

**{**

**private:**

**node<T> \*head;**

**node<T> \*tail;**

**public:**

**dllist()**

**{**

**head = tail = 0;**

**}**

**int isempty();**

**void addtohead(T x);**

**void addtotail(T x);**

**void add\_at\_i(int i, T x);**

**T delete\_fromhead();**

**T delete\_fromtail();**

**void deletenode(T x);**

**void length();**

**void delete\_from\_i(int i);**

**T isinlist(int x);**

**void concat(dllist<T> L1);**

**void reverse(dllist<T> L1);**

**dllist operator + (dllist<T> L1);**

**T input();**

**void display();**

**};**

**template<class T>int dllist<T> :: isempty()**

**{**

**if(head == 0 && tail == 0)**

**return 1;**

**else**

**return 0;**

**}**

**template<class T>void dllist<T> :: addtohead(T x)**

**{**

**node<T> \*temp = new node<T>(x);**

**if(isempty())**

**{**

**head = tail = temp;**

**}**

**else**

**{**

**temp -> next = head;**

**head -> prev = temp;**

**head = temp;**

**}**

**}**

**template<class T>void dllist<T> :: addtotail(T x)**

**{**

**node<T> \*temp = new node<T>(x);**

**if(isempty())**

**{**

**head = tail = temp;**

**}**

**else**

**{**

**tail -> next = temp;**

**temp -> prev = tail;**

**tail = temp;**

**}**

**}**

**template<class T>void dllist<T> :: add\_at\_i(int i,T x)**

**{**

**if(i < 0)**

**{**

**cout<<"Out of bound !"<<endl;**

**return;**

**}**

**node<T> \*nptr = new node<T>[1];**

**nptr -> info = x;**

**int count = 0;**

**node<T> \*temp = head;**

**while(temp != NULL && count < i)**

**{**

**if(count == i-1)**

**{**

**if(temp -> next != NULL)**

**{**

**nptr -> next = temp -> next;**

**}**

**temp -> next = nptr;**

**nptr -> prev = temp;**

**cout<<"New Node added at ith position "<<i<<" !"<<endl;**

**break;**

**}**

**count++;**

**temp = temp -> next;**

**}**

**}**

**template<class T>void dllist<T> :: length()**

**{**

**int len = 0;**

**node<int> \*temp = head;**

**while(temp != NULL)**

**{**

**len++;**

**temp = temp -> next;**

**}**

**return;**

**}**

**template<class T>T dllist<T> :: delete\_fromhead()**

**{**

**T x = head -> info;**

**if(head == tail)**

**{**

**delete head;**

**head = tail = 0;**

**}**

**else**

**{**

**head = head -> next;**

**delete head -> prev;**

**head -> prev = 0;**

**}**

**return x;**

**}**

**template<class T>T dllist<T> :: delete\_fromtail()**

**{**

**T x = tail -> info;**

**if(head == tail)**

**{**

**delete head;**

**head = tail = 0;**

**}**

**else**

**{**

**tail = tail -> prev;**

**delete tail -> next;**

**tail -> next = 0;**

**}**

**return x;**

**}**

**template<class T>void dllist<T> :: delete\_from\_i(int i)**

**{**

**if(head == NULL)**

**{**

**cout<<"Linked List is Empty !"<<endl;**

**return;**

**}**

**if(i < 0)**

**{**

**cout<<"Out of bound !"<<endl;**

**return;**

**}**

**if(i == 0)**

**{**

**delete\_fromhead();**

**cout<<"Node removed from ith position "<<i<<endl;**

**return;**

**}**

**int count = 0;**

**node<T> \*temp = head;**

**while(temp != NULL)**

**{**

**if(count == i-1)**

**{**

**temp -> next = temp -> next -> next;**

**cout<<"Node removed from ith position"<<i<<endl;**

**break;**

**}**

**count++;**

**temp = temp -> next;**

**}**

**}**

**template<class T>void dllist<T> :: display()**

**{**

**node<T> \*temp = head;**

**cout<<endl<<"\_Doubly Linked List is ";**

**while(temp != NULL)**

**{**

**cout<<temp -> info<<" ";**

**temp = temp -> next;**

**}**

**}**

**template<class T>T dllist<T> :: input()**

**{**

**T n;**

**cout<<"\*\*\*Enter the Number\*\*\* : ";**

**cin>>n;**

**return n;**

**}**

**template<class T>void dllist<T> :: deletenode(T x)**

**{**

**node<T> \*temp,\*prev;**

**if((head == tail) && (head -> info == x))**

**{**

**delete head;**

**head = tail = 0;**

**}**

**else if(head -> info == x)**

**{**

**head = head -> next;**

**delete head -> prev;**

**head -> prev = NULL;**

**cout<<"node"<<x<<"has been deleted from your list";**

**}**

**else**

**{**

**prev = head;**

**temp = head -> next;**

**while((temp != 0) && (temp -> info != x))**

**{**

**prev = temp;**

**temp = temp -> next;**

**}**

**if(temp != 0)**

**{**

**prev -> next = temp -> next;**

**if(temp == tail)**

**tail = prev;**

**delete temp;**

**}**

**else**

**{**

**cout<<"Elements not found "<<endl;**

**}**

**}**

**}**

**template<class T>T dllist<T> :: isinlist(int x)**

**{**

**node<T> \*temp;**

**if(head -> info == x)**

**{**

**return 1;**

**}**

**else if(tail -> info == x)**

**return 1;**

**else**

**{**

**temp = head;**

**while((temp != 0) && (temp -> info != x))**

**{**

**temp = temp -> next;**

**}**

**if(temp != 0)**

**return 1;**

**else**

**return 0;**

**}**

**return 0;**

**}**

**template<class T>void dllist<T> :: reverse(dllist<T> L1)**

**{**

**node<T> \*temp;**

**temp = L1.head;**

**while(temp != 0)**

**{**

**(\*this).addtohead(temp -> info);**

**temp = temp -> next;**

**}**

**display();**

**}**

**template<class T>void dllist<T> :: concat(dllist<T> L1)**

**{**

**node<T> \*temp;**

**temp = L1.head;**

**while(temp != 0)**

**{**

**(\*this).addtotail(temp -> info);**

**temp = temp -> next;**

**}**

**(\*this).display();**

**}**

**template<class T>dllist<T>dllist<T> :: operator + (dllist<T> L1)**

**{**

**node<T> \*temp;**

**dllist L3;**

**if(!(\*this).isempty());**

**{**

**temp = (\*this).head;**

**while(temp != 0)**

**{**

**L3.addtotail(temp -> info);**

**temp = temp -> next;**

**}**

**if(!L1.isempty())**

**{**

**temp = L1.head;**

**while(temp!=0)**

**{**

**L3.addtotail(temp -> info);**

**temp = temp -> next;**

**}**

**}**

**return L3;**

**}**

**}**

**int main()**

**{**

**dllist<int> L1,L2,L3;**

**char c = 'y';**

**int choice, x, s, num , a;**

**while(c=='y'||c=='Y')**

**{**

**cout<<"\*\*\*Have a look at the Main Menu\*\*\*"<<endl;**

**cout<<" 1. Add To Head "<<endl;**

**cout<<" 2. Add To Tail "<<endl;**

**cout<<" 3. Add at ith position "<<endl;**

**cout<<" 4. Delete From Head "<<endl;**

**cout<<" 5. Delete From Tail "<<endl;**

**cout<<" 6. Delete from ith position "<<endl;**

**cout<<" 7. Delete The Node From Doubly Linked List "<<endl;**

**cout<<" 8. Find The Node in Doubly Linked List "<<endl;**

**cout<<" 9. To Reverse The Element "<<endl;**

**cout<<" 10. To concatenate the Elements in Doubly Linked List "<<endl;**

**cout<<" 11. To Display The Doubly Linked List "<<endl;**

**cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;**

**cout<<"Enter your Choice : ";**

**cin>>choice;**

**switch(choice)**

**{**

**case 1: x= L1.input();**

**L1.addtohead(x);**

**L1.display();**

**break;**

**case 2: x= L1.input();**

**L1.addtotail(x);**

**L1.display();**

**break;**

**case 3: x= L1.input();**

**cout<<"Enter the ith position where the new node to be inserted :";**

**cin>>a;**

**cout<<"Enter the No. which you inserted at the very starting : ";**

**cin>>num;**

**L1.add\_at\_i(a, num);**

**L1.display();**

**break;**

**case 4: if(!L1.isempty())**

**{**

**s = L1.delete\_fromhead();**

**cout<<"Deleted Node is : "<<s;**

**L1.display();**

**}**

**else**

**{**

**cout<<"The List is Empty !";**

**}**

**break;**

**case 5: if(!L1.isempty())**

**{**

**s = L1.delete\_fromtail();**

**cout<<"Deleted node is : "<<s;**

**L1.display();**

**}**

**else**

**{**

**cout<<"The List is Empty !";**

**}**

**break;**

**case 6: if(!L1.isempty())**

**{**

**cout<<"Enter the index : ";**

**cin>>a;**

**L1.delete\_from\_i(a);**

**L1.display();**

**}**

**else**

**{**

**cout<<"List is empty!";**

**}**

**break;**

**case 7: if(!L1.isempty())**

**{**

**cout<<"Enter the value which you want to delete : ";**

**cin>>x;**

**L1.deletenode(x);**

**L1.display();**

**}**

**else**

**{**

**cout<<"List is empty!";**

**}**

**break;**

**case 8: if(!L1.isempty())**

**{**

**cout<<"Enter the Value which you want to Search : ";**

**cin>>x;**

**s = L1.isinlist(x);**

**if(s == 1)**

**cout<<" We have found the Node you Searched For !!! ";**

**else**

**cout<<" Sorry ! The Node you searched for is not found ";**

**}**

**else**

**{**

**cout<<"The List is Empty ";**

**}**

**break;**

**case 9: if(!L1.isempty())**

**{**

**L2.reverse(L1);**

**}**

**else**

**{**

**cout<<"List is empty ";**

**}**

**break;**

**case 10: if((L1.isempty()) && (L2.isempty()))**

**{**

**cout<<"The List is Empty";**

**}**

**else**

**{**

**L2.concat(L1);**

**cout<<"Concatenated List is : ";**

**L2.display();**

**}**

**break;**

**case 11: L3 = L1 + L2;**

**L3.display();**

**break;**

**default: cout<<"Error in input"<<endl;**

**}**

**cout<<endl;**

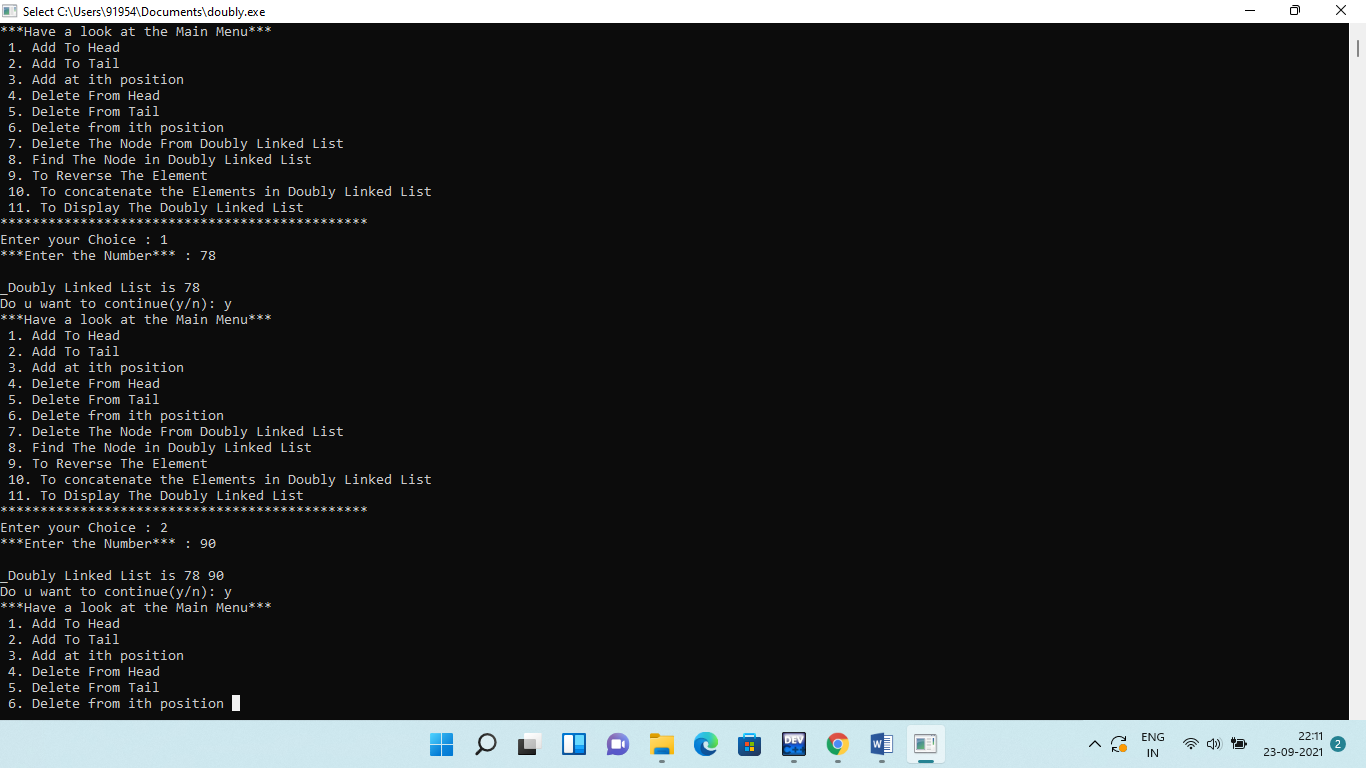
**cout<<"Do u want to continue(y/n): ";**

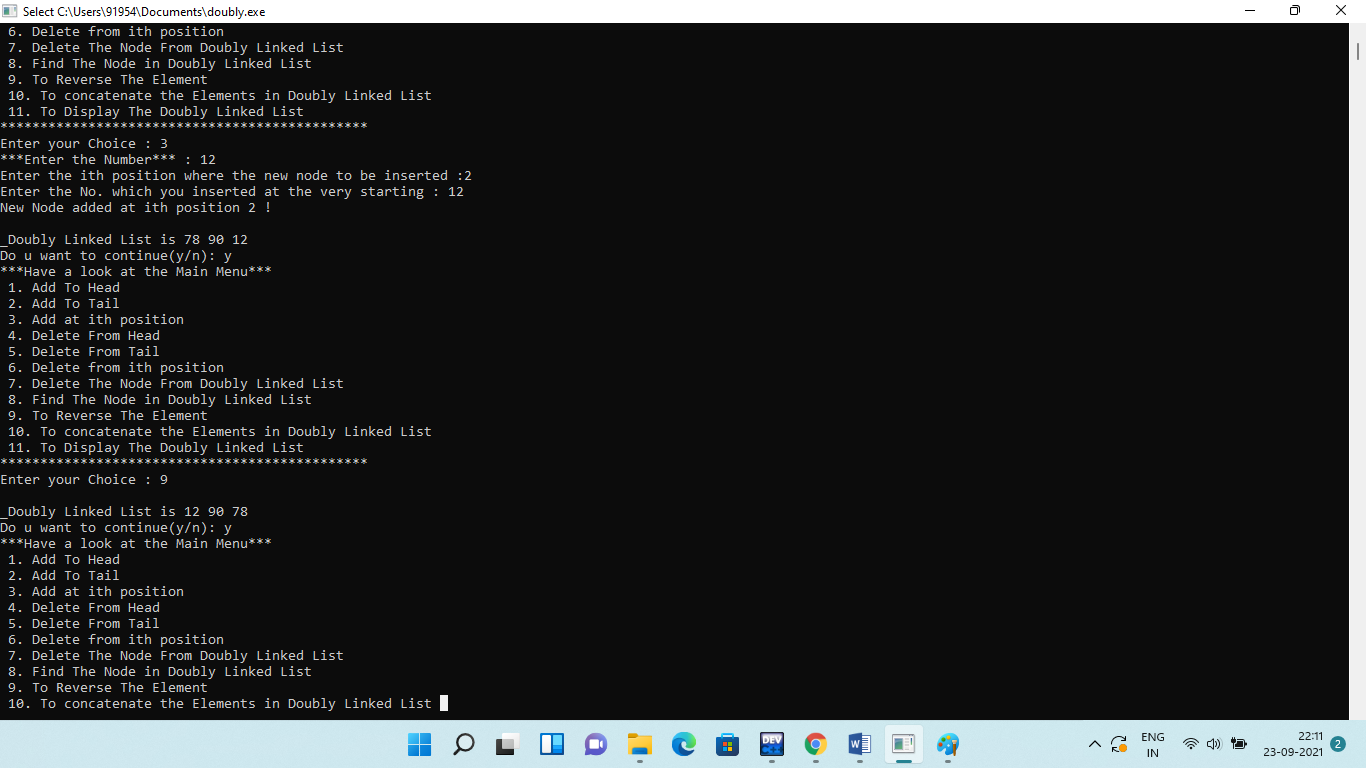
**cin>>c;**

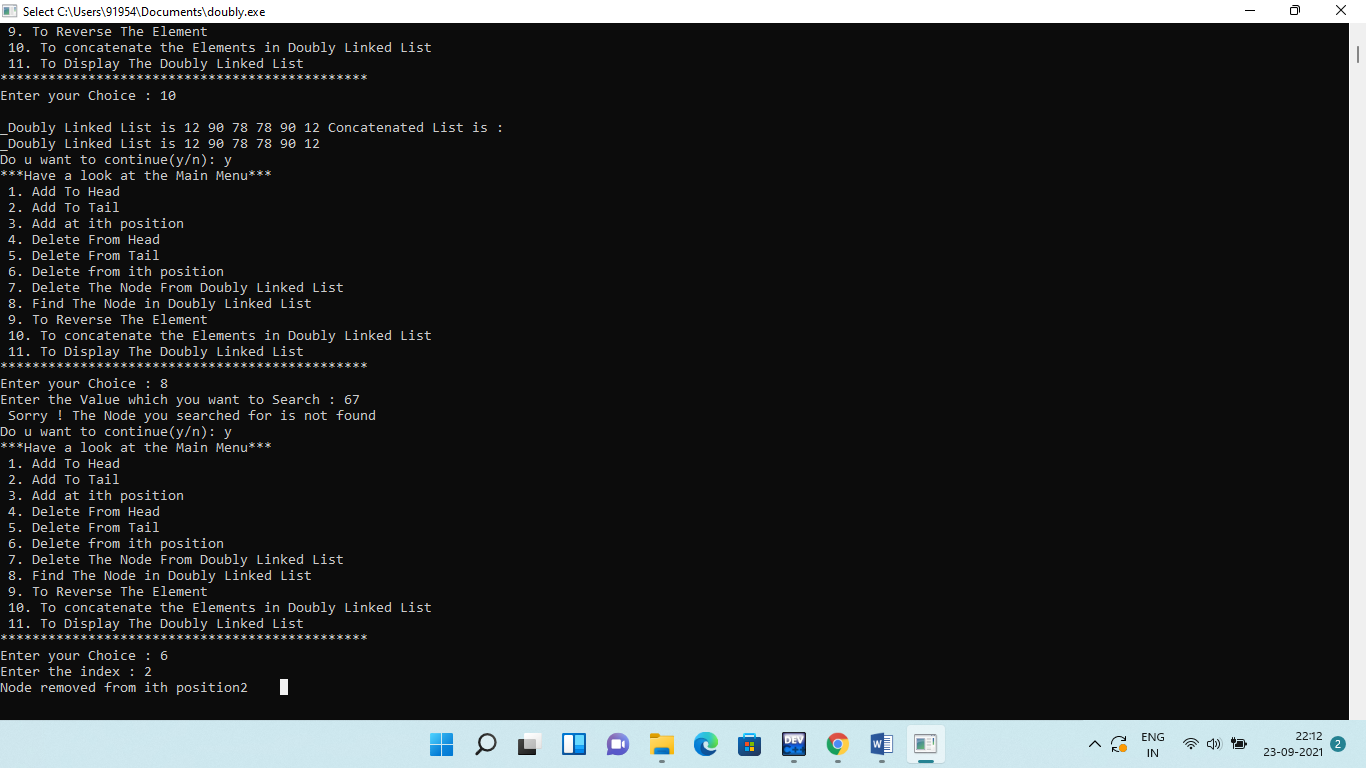
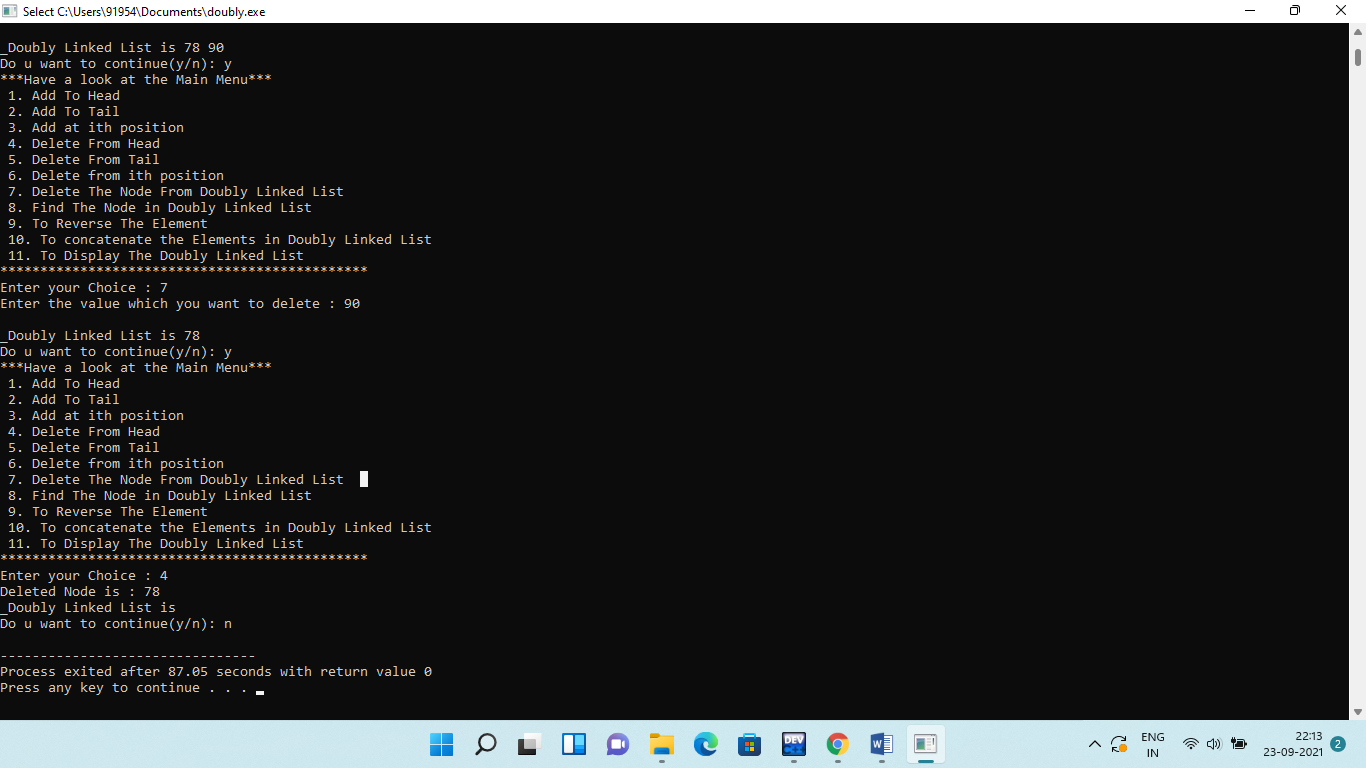
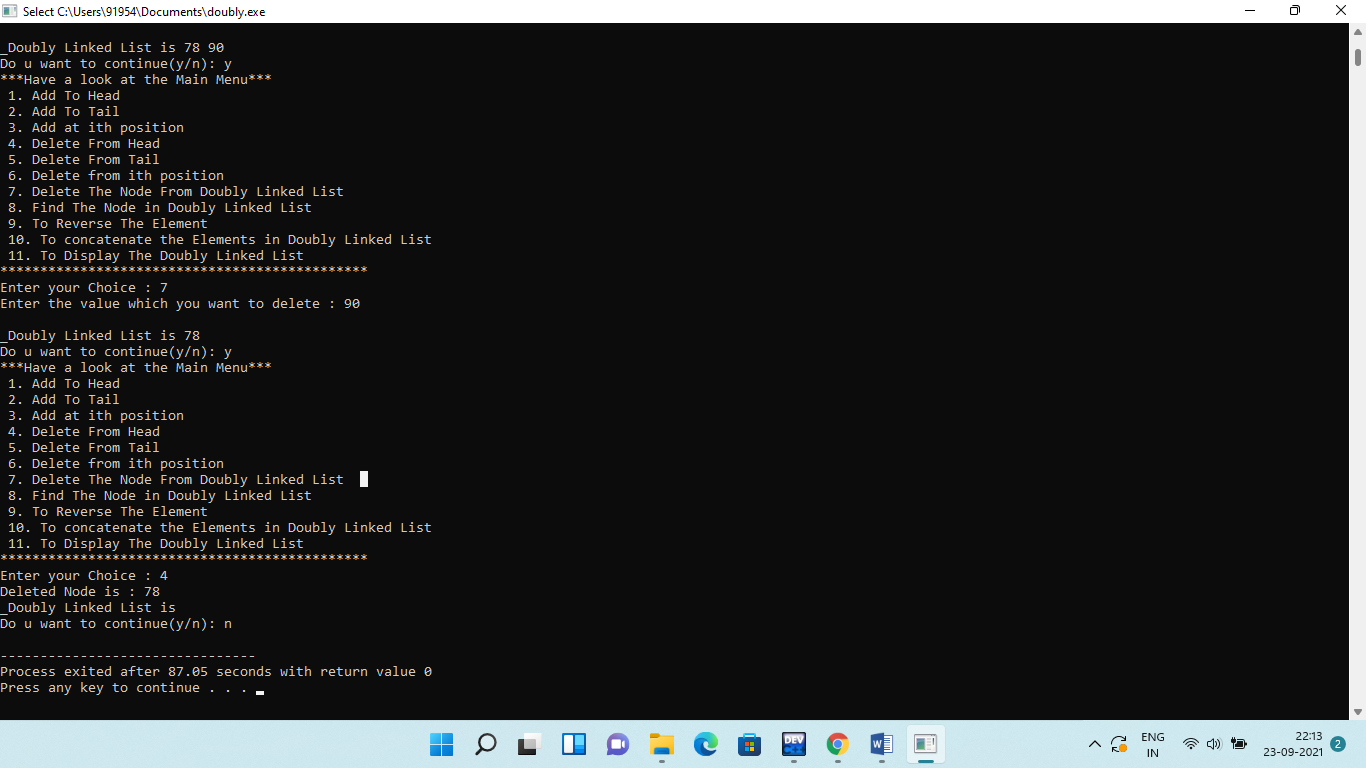
**}**

**return 0;**

**}**

**Output:**

****



**Q5.** **Write a program to implement circularly linked list which supports the following operations:**

**(i) Insert an element x at the front of the circularly linked list**

**(ii) Insert an element x after an element y in the circularly linked list**

**(iii)Insert an element x at the back of the circularly linked list**

**(iv) Remove an element from the back of the circularly linked list**

**(v) Remove an element from the front of the circularly linked list**

**(vi) remove the element x from the circularly linked list**

**(vii)Search for an element x in the circularly linked list and return its pointer**

**(viii) Concatenate two circularly linked lists.**

**Solution:**

**#include <iostream>**

**using namespace std;**

**template<class T>class cnode**

**{**

**public:**

**T info;**

**cnode \*next;**

**cnode(T x, cnode \*n = 0)**

**{**

**info = x;**

**next = n;**

**}**

**};**

**template<class T>class cllist**

**{**

**cnode<T> \*cursor;**

**public:**

**cllist()**

**{**

**cursor = NULL;**

**}**

**void input();**

**int isempty();**

**void addtofront(T x);**

**void addtoback(T x);**

**void add\_at\_i(int i,T x);**

**T deletefromfront();**

**T deletefromback();**

**void deletenode(T x);**

**T isinlist(int x);**

**int length();**

**void display();**

**cllist<T> operator + (cllist<T> L1);**

**};**

**template<class T>void cllist<T> :: input()**

**{**

**int n;**

**T x;**

**cout<<"Enter no. of nodes in CLL : ";**

**cin>>n;**

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

**{**

**cout<<"Enter the info part : ";**

**cin>>x;**

**addtoback(x);**

**}**

**}**

**template<class T>int cllist<T> :: isempty()**

**{**

**if(cursor == NULL)**

**return 1;**

**else**

**return 0;**

**}**

**template<class T>void cllist<T> :: addtofront(T x)**

**{**

**cnode<T> \*temp = new cnode<T>(x);**

**if(isempty())**

**{**

**cursor = temp;**

**temp -> next = cursor;**

**}**

**else**

**{**

**temp -> next = cursor -> next;**

**cursor -> next = temp;**

**}**

**}**

**template<class T>void cllist<T> :: addtoback(T x)**

**{**

**cnode<T> \*temp = new cnode<T>(x);**

**if(isempty())**

**{**

**cursor = temp;**

**cursor -> next = cursor;**

**}**

**else**

**{**

**temp -> next = cursor -> next;**

**cursor -> next = temp;**

**cursor = temp;**

**}**

**}**

**template<class T>int cllist<T> :: length()**

**{**

**int len = 0;**

**cnode<T>\*temp ;**

**if(isempty())**

**{**

**len = 0;**

**}**

**else if(cursor -> next == cursor)**

**{**

**len = 1;**

**}**

**else**

**{**

**for(temp = cursor -> next ; temp != cursor ; temp = temp -> next)**

**len++;**

**}**

**if(temp==cursor)**

**{**

**len++;**

**}**

**return len;**

**}**

**template<class T>void cllist<T> :: add\_at\_i(int i,T x)**

**{**

**int num = length();**

**int len = 0;**

**cnode<T> \*temp;**

**if(i>num || i<0)**

**{**

**cout<<"Index Out of bound!! "<<endl;**

**return;**

**}**

**else**

**{**

**if(i==0 && num>=1)**

**addtofront(x);**

**else if(i==num +1)**

**addtoback(x);**

**else**

**{**

**cnode<T> \*n = new cnode<T>(x);**

**for(temp = cursor -> next ; temp != cursor ; temp = temp -> next)**

**{**

**len++;**

**if(len==i)**

**break;**

**}**

**n -> next = temp -> next;**

**temp -> next = n;**

**}**

**}**

**}**

**template<class T>T cllist<T> :: deletefromfront()**

**{**

**cnode<T> \*temp;**

**T x;**

**x = cursor -> next -> info;**

**if(cursor -> next == cursor)**

**{**

**delete cursor;**

**cursor = NULL;**

**}**

**else**

**{**

**temp = cursor -> next;**

**cursor -> next = temp -> next;**

**delete temp;**

**}**

**return x;**

**}**

**template<class T>T cllist<T> :: deletefromback()**

**{**

**cnode<T> \*temp;**

**T x = cursor -> info;**

**if(cursor -> next == cursor)**

**{**

**delete cursor;**

**cursor = NULL;**

**}**

**else**

**{**

**temp = cursor -> next;**

**while(temp -> next!= cursor)**

**{**

**temp = temp -> next;**

**}**

**temp -> next = cursor -> next;**

**delete cursor;**

**cursor = temp;**

**}**

**return x;**

**}**

**template<class T>void cllist<T> :: deletenode(T x)**

**{**

**cnode<T> \*prev, \*current;**

**if(cursor -> next == cursor && cursor -> info == x)**

**{**

**delete cursor;**

**cursor = NULL;**

**}**

**else if(cursor -> next -> info == x)**

**{**

**T z = deletefromfront();**

**}**

**else**

**{**

**prev = cursor -> next;**

**current = prev -> next;**

**while(current != cursor && current -> info != x)**

**{**

**prev = current;**

**current = current -> next;**

**}**

**if(current != cursor -> next)**

**{**

**prev -> next = current -> next;**

**if(current == cursor)**

**{**

**cursor = prev;**

**delete current;**

**}**

**}**

**else**

**{**

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

**}**

**}**

**}**

**template<class T>T cllist<T> :: isinlist(int x)**

**{**

**cnode<T> \*temp;**

**temp = cursor -> next;**

**do**

**{**

**if(temp -> info == x)**

**return 0;**

**temp = temp -> next;**

**}**

**while(temp != cursor -> next);**

**return 0;**

**}**

**template<class T> cllist<T> cllist<T> :: operator + (cllist<T> L1)**

**{**

**cllist<T> L3;**

**cnode<T> \*temp;**

**if(!(\*this).isempty())**

**{**

**temp = (\*this).cursor -> next;**

**do**

**{**

**L3.addtoback(temp -> info);**

**temp = temp -> next;**

**}**

**while(temp != (\*this).cursor -> next);**

**}**

**if(!L1.isempty());**

**{**

**temp = (L1.cursor -> next);**

**do**

**{**

**L3.addtoback(temp -> info);**

**temp = temp -> next;**

**}**

**while(temp != L1.cursor -> next);**

**}**

**return L3;**

**}**

**template<class T>void cllist<T> :: display()**

**{**

**cnode<T> \*temp;**

**int y;**

**cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";**

**if(isempty())**

**{**

**cout<<"The List is Empty !!\n";**

**}**

**else**

**{**

**cout<<"Contents of Circular Linked List :\n";**

**for(temp = cursor -> next; temp != cursor; temp = temp -> next)**

**{**

**cout<<temp->info<<" ";**

**}**

**if(temp == cursor)**

**{**

**cout<<temp -> info <<" ";**

**}**

**}**

**cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";**

**}**

**int main()**

**{**

**cllist<int> L1,L2,L3;**

**char c ='y';**

**int choice,num,a,b,d,e;**

**L1.input();**

**L1.display();**

**while(c=='y' || c=='Y')**

**{**

**cout<<"\*\*\*\*WELCOME TO MY MAIN MENU\*\*\*\*"<<endl;**

**cout<<"1. Add to front "<<endl;**

**cout<<"2. Add to Back "<<endl;**

**cout<<"3. Add at ith position "<<endl;**

**cout<<"4. Delete from front "<<endl;**

**cout<<"5. Delete from Back "<<endl;**

**cout<<"6. Delete the Node from Circular Linked List "<<endl;**

**cout<<"7. Find the Node in Circular Linked List "<<endl;**

**cout<<"8. Find The Length "<<endl;**

**cout<<"9. To Concatenate the Elements in Circular Linked List "<<endl;**

**cout<<"10. Display the Circular Linked List "<<endl;**

**cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;**

**cout<<"Please Enter your Choice (1-9) : ";**

**cin>>choice;**

**switch(choice)**

**{**

**case 1: cout<<" Enter your info part : ";**

**cin>>a;**

**L1.addtofront(a);**

**L1.display();**

**break;**

**case 2: cout<<" Enter the info part : ";**

**cin>>a;**

**L1.addtoback(a);**

**L1.display();**

**break;**

**case 3: cout<<"Enter the position where the new node to be inserted: ";**

**cin>>b;**

**cout<<"Enter the no. which u want to insert : ";**

**cin>>num;**

**L1.add\_at\_i(b,num);**

**L1.display();**

**break;**

**case 4: e = L1.isempty();**

**if(e==1)**

**{**

**cout<<"The List is Empty !!";**

**}**

**else**

**{**

**d = L1.deletefromfront();**

**cout<<"Deleted Node is : "<<d<<endl;**

**L1.display();**

**}**

**break;**

**case 5: e = L1.isempty();**

**if(e==1)**

**{**

**cout<<"The List is Empty ! ";**

**}**

**else**

**{**

**d = L1.deletefromback();**

**cout<<"Deleted Node is : "<<d<<endl;**

**L1.display();**

**}**

**break;**

**case 6: e = L1.isempty();**

**if(e==1)**

**{**

**cout<<"List is empty !";**

**}**

**else**

**{**

**cout<<"Enter the value which you want to delete : ";**

**cin>>a;**

**L1.deletenode(a);**

**L1.display();**

**}**

**break;**

**case 7: e = L1.isempty();**

**if(e==1)**

**{**

**cout<<"The List is Empty!!";**

**}**

**else**

**{**

**cout<<"Enter the Value which you want to Search : ";**

**cin>>a;**

**d = L1.isinlist(a);**

**if(d==0)**

**cout<<"We have found the Node you Searched For !!!";**

**else**

**cout<<"Opps! The Node you searched for is not found ";**

**}**

**break;**

**case 8: cout<<"No. of Nodes present in the List is : "<<L1.length();**

**break;**

**case 9: if((L1.isempty()) && (L2.isempty()))**

**{**

**cout<<"The List is Empty!!";**

**}**

**else**

**{**

**L2.input();**

**L3 = L1 + L2;**

**cout<<"Concatenated List is : ";**

**L3.display();**

**}**

**break;**

**case 10:cout<<"1st List :\n";**

**L1.display();**

**cout<<"2nd List :\n";**

**L2.display();**

**break;**

**default : cout<<"Error in input"<<endl;**

**}**

**cout<<endl;**

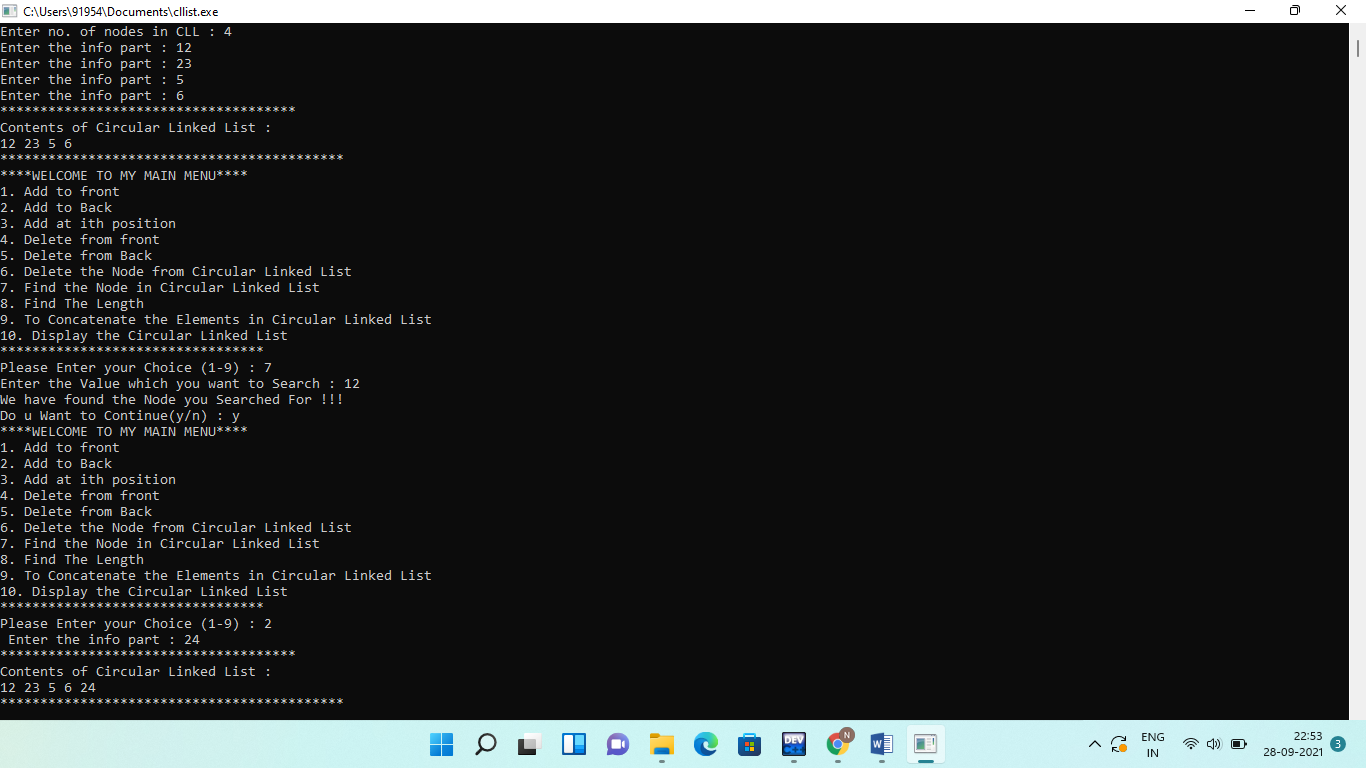
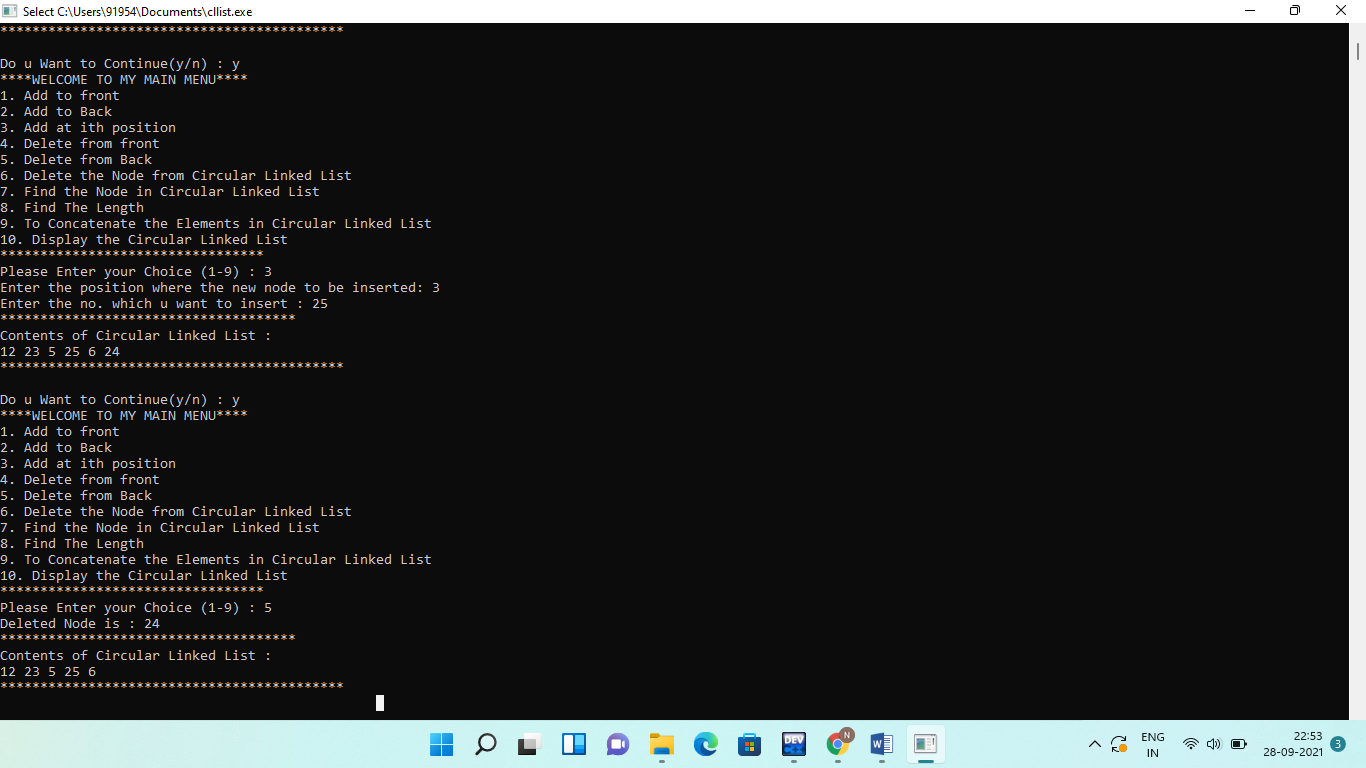
**cout<<"Do u Want to Continue(y/n) : ";**

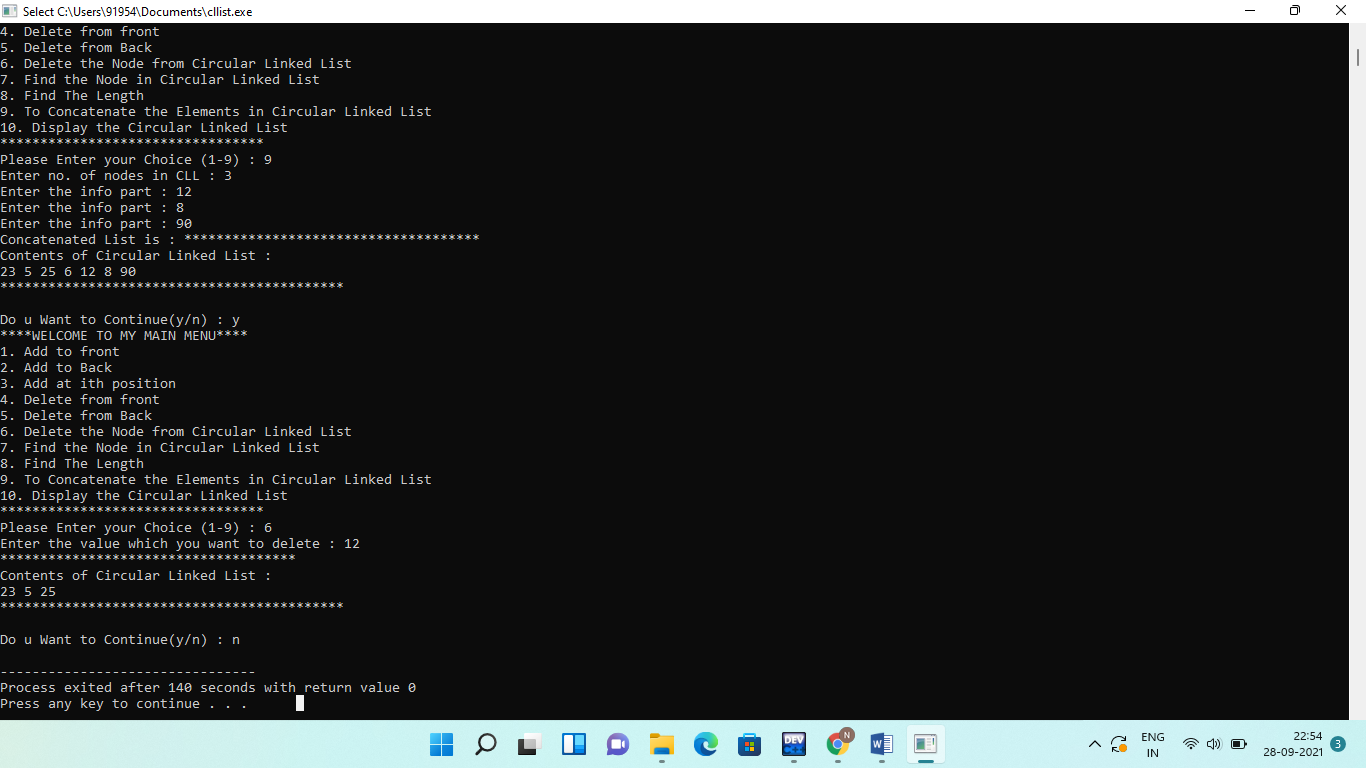
**cin>>c;**

**}**

**return 0;**

**}**

**Output:**

****

**Q6.** **Implement a stack using Array representation**.

**Solution:**

**#include<iostream>**

**using namespace std;**

**const int Max=7;**

**template<class T>**

**class Stacktype**

**{**

**int top;**

**int s[Max];**

**public:**

**void push(int);**

**int pop();**

**int isempty();**

**int isfull();**

**void display();**

**stacktype()**

**{**

**top=-1;**

**}**

**};**

**template<class T>**

**void Stacktype<T>::push(int p)**

**{**

**top++;**

**s[top]=p;**

**}**

**template<class T>**

**int Stacktype<T>::pop(){**

**int del;**

**del=s[top];**

**top--;**

**return del;**

**}**

**template<class T>**

**int Stacktype<T>::isempty(){**

**if(top==-1)**

**return 1;**

**else**

**return -1;**

**}**

**template<class T>**

**int Stacktype<T>::isfull()**

**{**

**if(top==Max-1)**

**return 1;**

**else**

**return -1;**

**}**

**template<class T>**

**void Stacktype<T>::display()**

**{**

**if(isempty()==-1)**

**{**

**cout<<"\n contents of stack are:";**

**for(int i=0;i<=top;i++)**

**{**

**cout<<s[i]<<" ";**

**}**

**cout<<endl;**

**}**

**else**

**{**

**cout<<"\n stack is empty";**

**}**

**}**

**int main()**

**{**

**Stacktype<int>stack;**

**int choice,f,e,d,a;**

**char c='y';**

**while(c=='y'|| c=='Y')**

**{**

**cout<<"\n Main Menu:";**

**cout<<"\n 1. To Perform Push Operation:";**

**cout<<"\n 2. To Perform Pop Operation:";**

**cout<<"\n 3. To Check if empty:";**

**cout<<"\n 4. To Check if full:";**

**cout<<"\n 5.Display:";**

**cout<<"\n enter your choice[1-5]"<<endl;**

**cin>>choice;**

**switch(choice)**

**{**

**case 1: f=stack.isfull();**

**if(f==1)**

**{**

**cout<<"\n Overflow! stack is full";**

**break;**

**}**

**else{**

**cout<<"\n Enter an integer value:";**

**cin>>a;**

**stack.push(a);**

**stack.display();**

**}**

**break;**

**case 2: e=stack.isempty();**

**if(e==1)**

**{**

**cout<<"\n Underflow! stack is empty";**

**break;**

**}**

**else{**

**d=stack.pop();**

**cout<<d<<" has been deleted"<<endl;**

**stack.display();**

**}**

**break;**

**case 3: e=stack.isempty();**

**if(e==1)**

**{**

**cout<<"\n stack is empty";**

**}**

**else{**

**cout<<"\n stack is not empty";**

**}**

**break;**

**case 4: f=stack.isfull();**

**if(f==1)**

**{**

**cout<<"\n stack is full";**

**}**

**else{**

**cout<<"\n stack is not full";**

**}**

**break;**

**case 5: stack.display();**

**break;**

**default: cout<<"\n Error in input";**

**}//closing block of switch**

**cout<<"\n Do you want to continue:";**

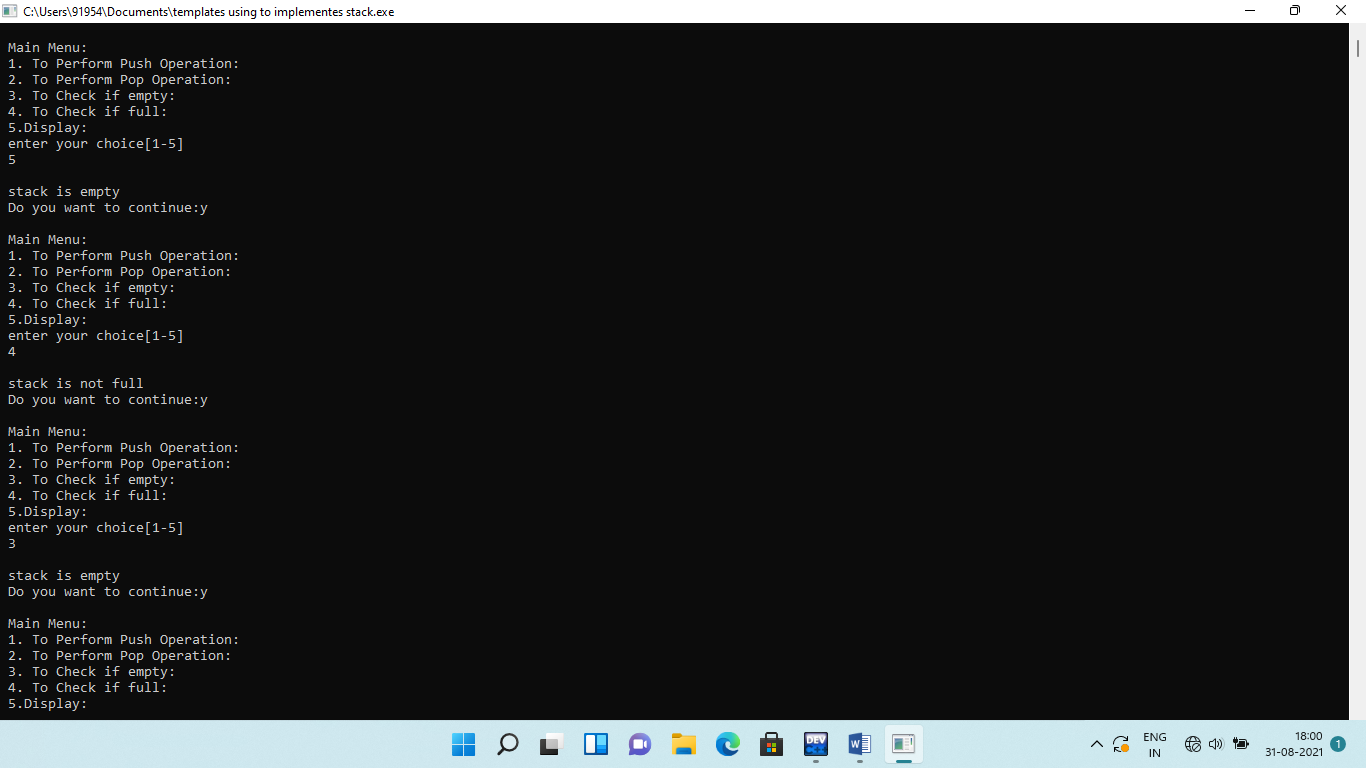
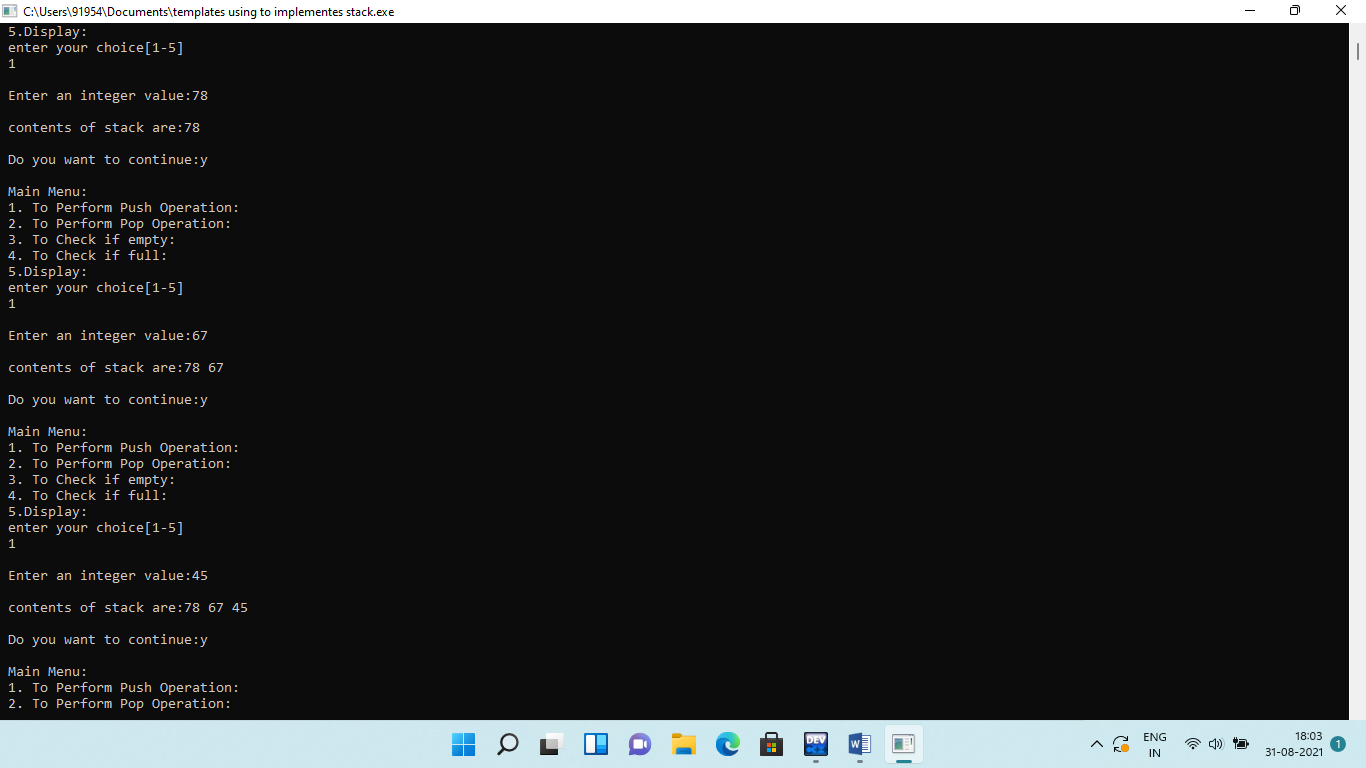
**cin>>c;**

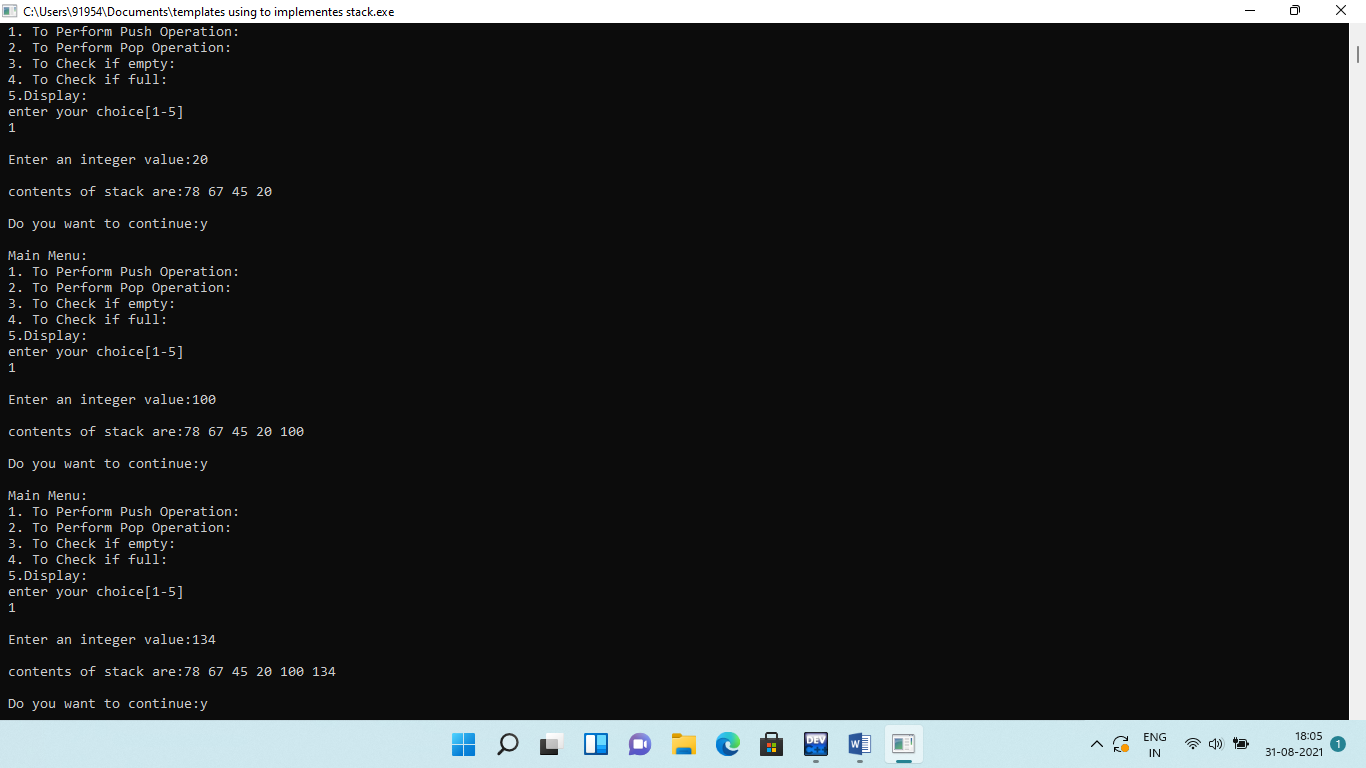
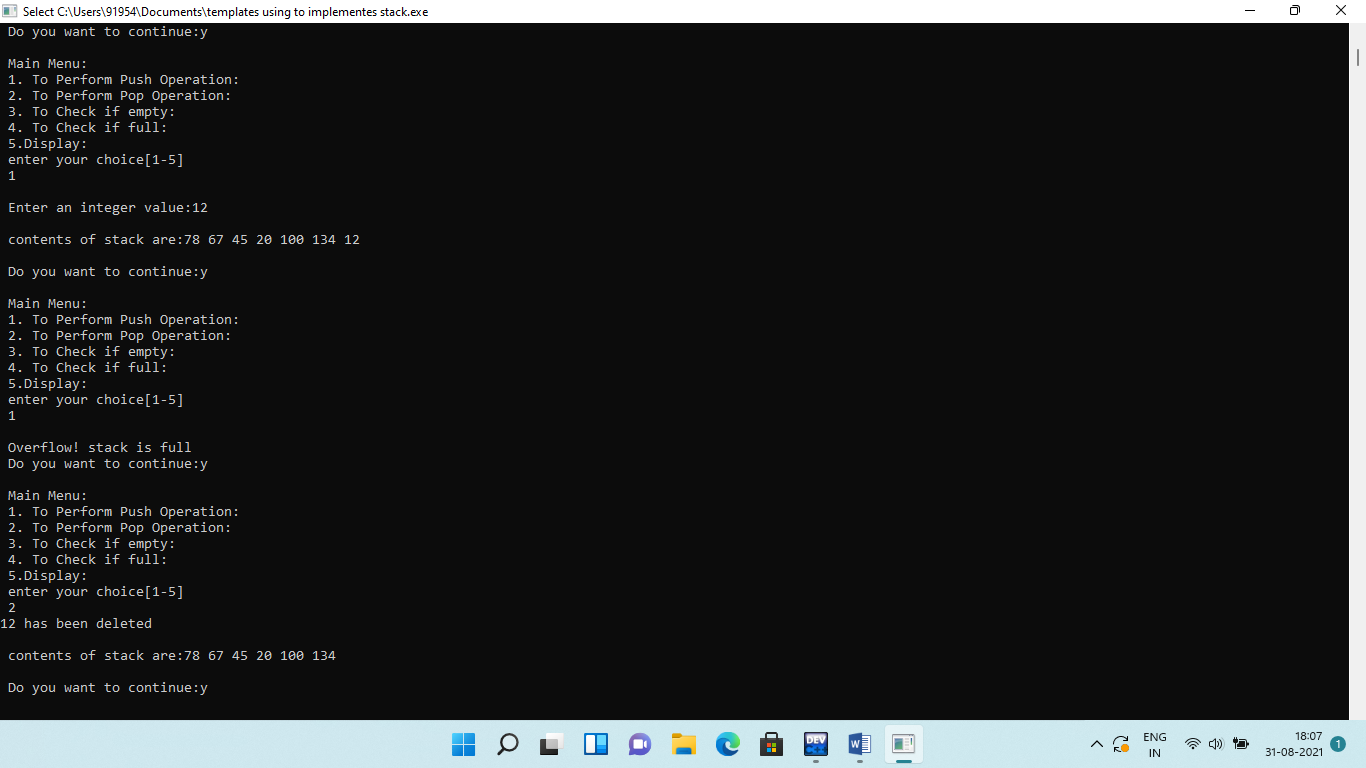
**}**

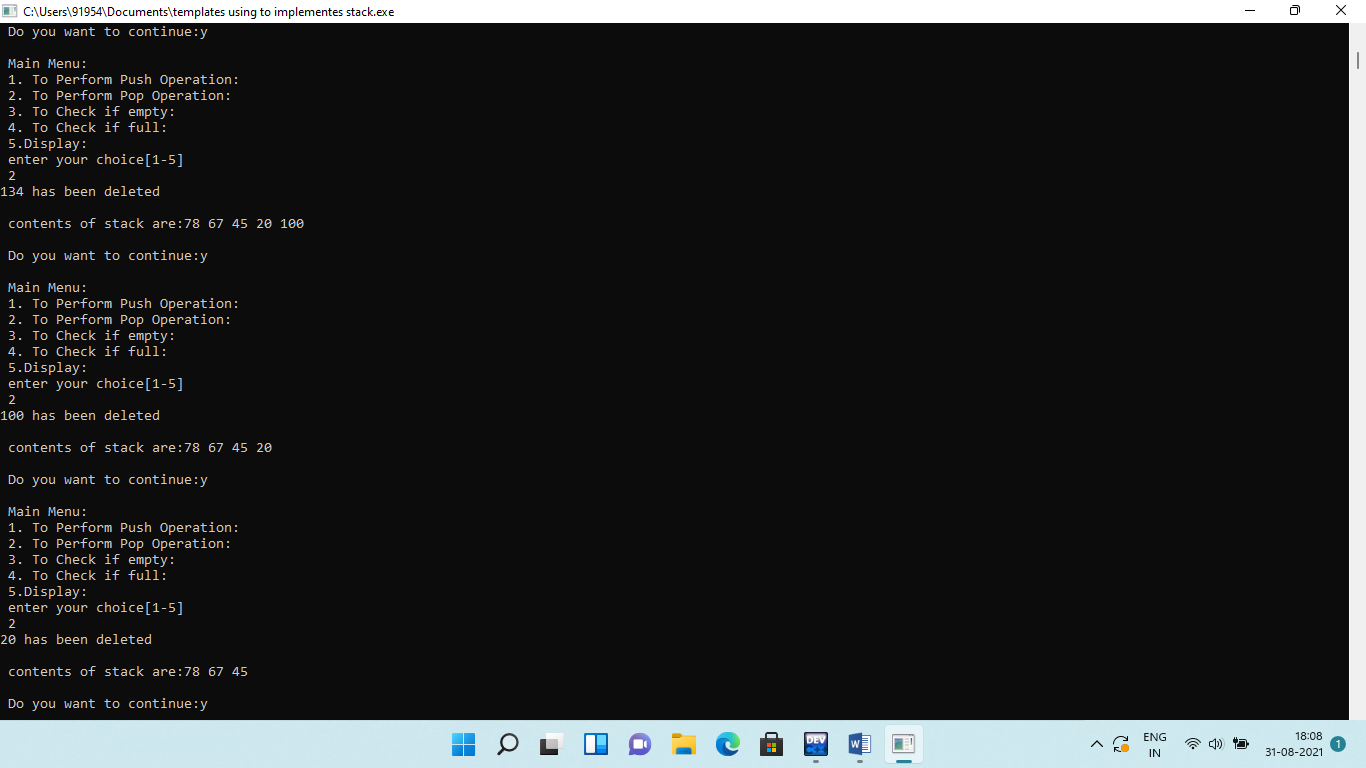
**return 0;**

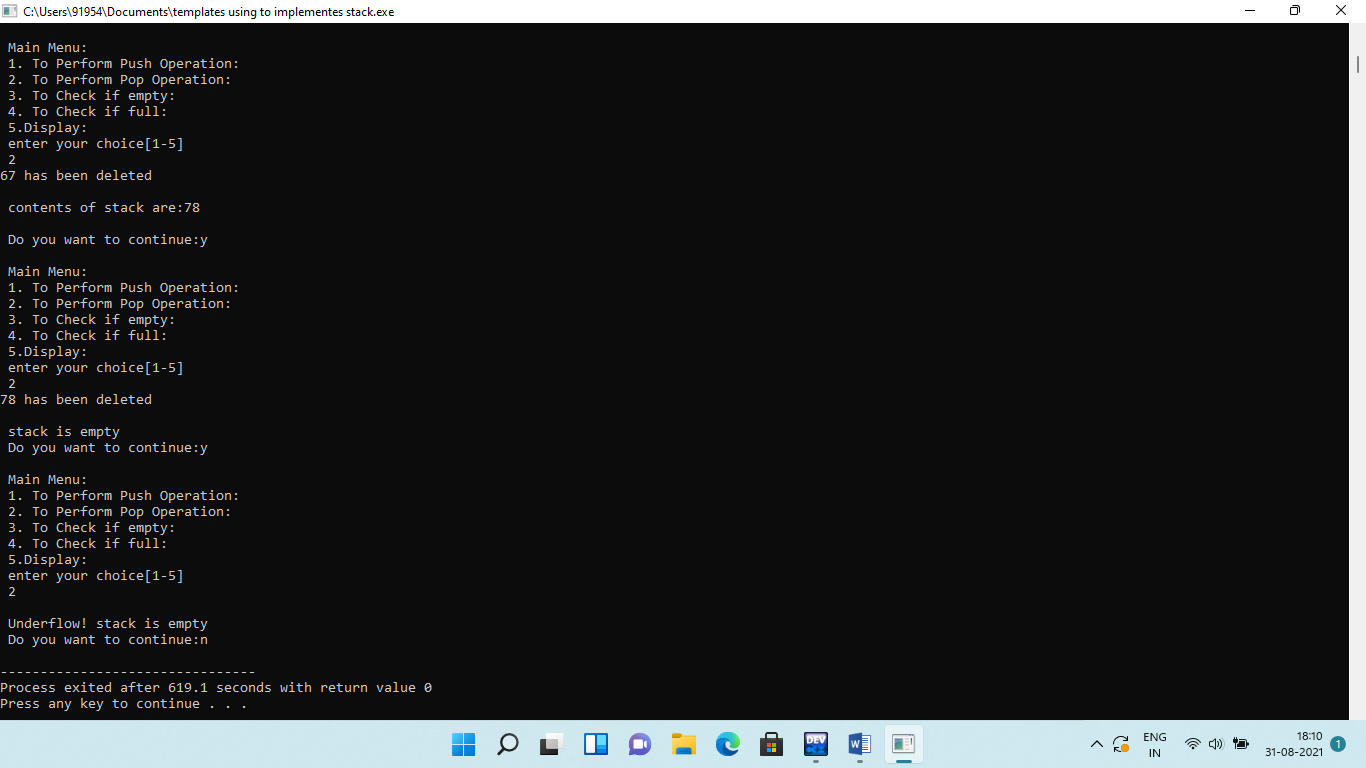
**}**

**Output:**

****

****

****

**Q7.** **Implement a stack using Linked representation.**

**Solution:**

**#include<iostream>**

**using namespace std;**

**class node**

**{**

**public:**

**int info;**

**node\* next;**

**node(int x,node\*n=NULL)**

**{**

**info=x;**

**next=n;**

**}**

**};**

**class Stacktype**

**{**

**node \*top;**

**public:**

**Stacktype()**

**{**

**top==NULL;**

**}**

**void push(node\*);**

**int pop();**

**int isempty();**

**void clear();**

**void topmost();**

**void display();**

**node\* createnewnode(int);**

**void count();**

**};**

**int Stacktype::isempty()**

**{**

**if(top==NULL)**

**return 1;**

**else**

**return -1;**

**}**

**void Stacktype::clear()**

**{**

**node\*p;**

**p=top;**

**top=top->next;**

**cout<<"\n memory retrieve:";**

**for(;p!=NULL;)**

**{**

**delete p;**

**p=top;**

**top=top->next;**

**}**

**}**

**int Stacktype::pop()**

**{**

**int d;**

**node \*p;**

**p=top;**

**top=top->next;**

**d=p->info;**

**delete p;**

**return d;**

**}**

**node \*Stacktype::createnewnode(int x)**

**{**

**node \*nptr;**

**nptr=new node(x,NULL);**

**nptr->info=x;**

**nptr->next=NULL;**

**return nptr;**

**}**

**void Stacktype::push(node \*nptr)**

**{**

**if(top==NULL)**

**{**

**top=nptr;**

**}**

**else**

**{**

**nptr->next=top;**

**top=nptr;**

**}**

**}**

**void Stacktype::display()**

**{**

**node \*nptr;**

**if(top==NULL)**

**{**

**cout<<"\n stack is empty";**

**return;**

**}**

**cout<<"\n contents of stack are:";**

**for(nptr=top;nptr!=NULL;nptr=nptr->next)**

**{**

**cout<<nptr->info<<" ";**

**}**

**}**

**void Stacktype::topmost()**

**{**

**cout<<" the topmost element/node in the stack is:";**

**cout<<top->info;**

**}**

**void Stacktype::count()**

**{**

**int c=0;**

**node \*nptr;**

**for(nptr=top;nptr!=NULL;nptr=nptr->next)**

**{**

**c++;**

**}**

**cout<<"\n no. of nodes are:"<<c;**

**}**

**int main()**

**{**

**Stacktype s;**

**int a,e,d,choice;**

**node \*temp=NULL;**

**char c='y';**

**while(c=='y'|| c=='Y')**

**{**

**cout<<"\n Main Menu:";**

**cout<<"\n 1. To Perform Push Operation:";**

**cout<<"\n 2. To Perform Pop Operation:";**

**cout<<"\n 3. To Check if empty:";**

**cout<<"\n 4. count:";**

**cout<<"\n 5. clear:";**

**cout<<"\n 6.Display:";**

**cout<<"\n 7. topmost:";**

**cout<<"\n enter your choice[1-5]"<<endl;**

**cin>>choice;**

**switch(choice)**

**{**

**case 1:cout<<"\n Enter an integer value:";**

**cin>>a;**

**temp=s.createnewnode(a);**

**s.push(temp);**

**s.display();**

**break;**

**case 2: e=s.isempty();**

**if(e==1)**

**{**

**cout<<"\n Underflow! stack is empty";**

**break;**

**}**

**else{**

**d=s.pop();**

**cout<<d<<" has been deleted"<<endl;**

**s.display();**

**}**

**break;**

**case 3: e=s.isempty();**

**if(e==1)**

**{**

**cout<<"\n stack is empty";**

**}**

**else{**

**cout<<"\n stack is not empty";**

**}**

**break;**

**case 4: s.count();**

**break;**

**case 5: s.clear();**

**break;**

**case 6: s.display();**

**break;**

**case 7: s.topmost();**

**break;**

**default: cout<<"\n Error in input";**

**}//closing block of switch**

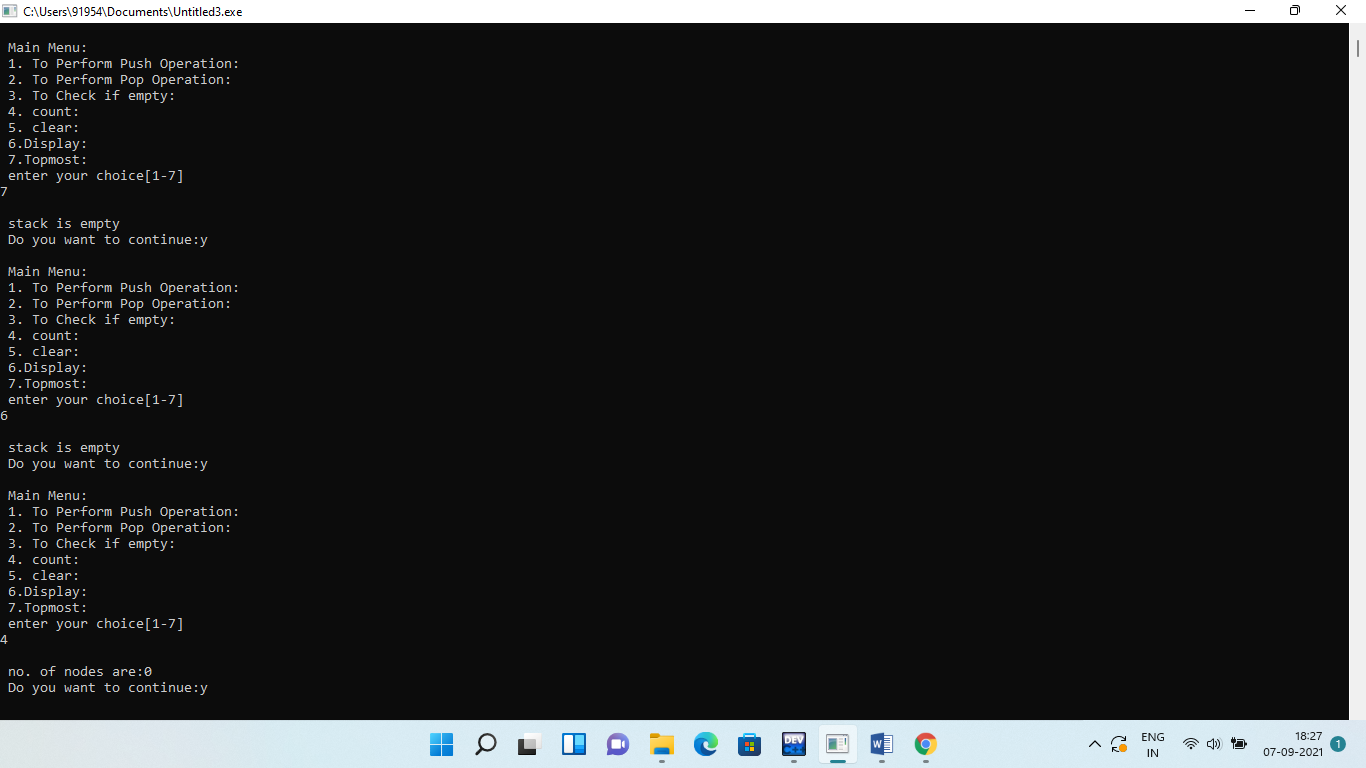
**cout<<"\n Do you want to continue:";**

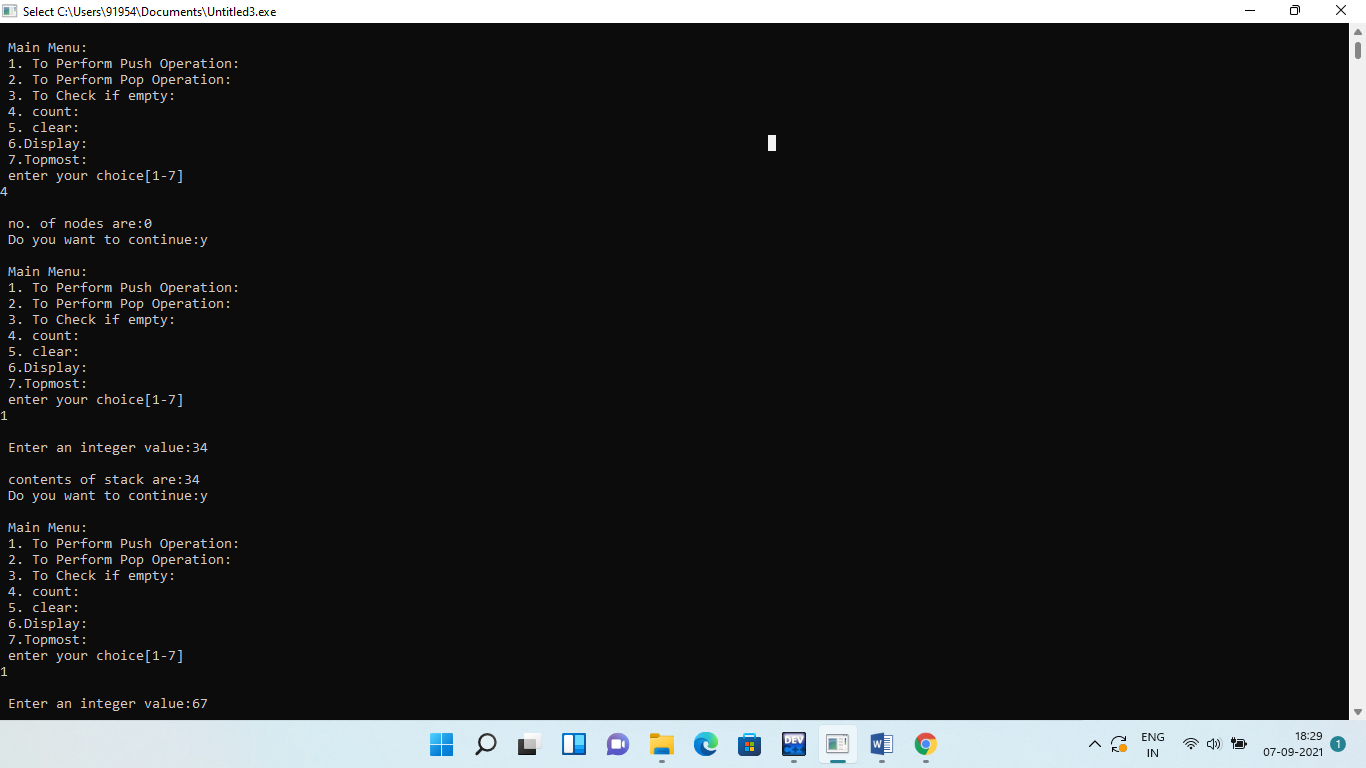
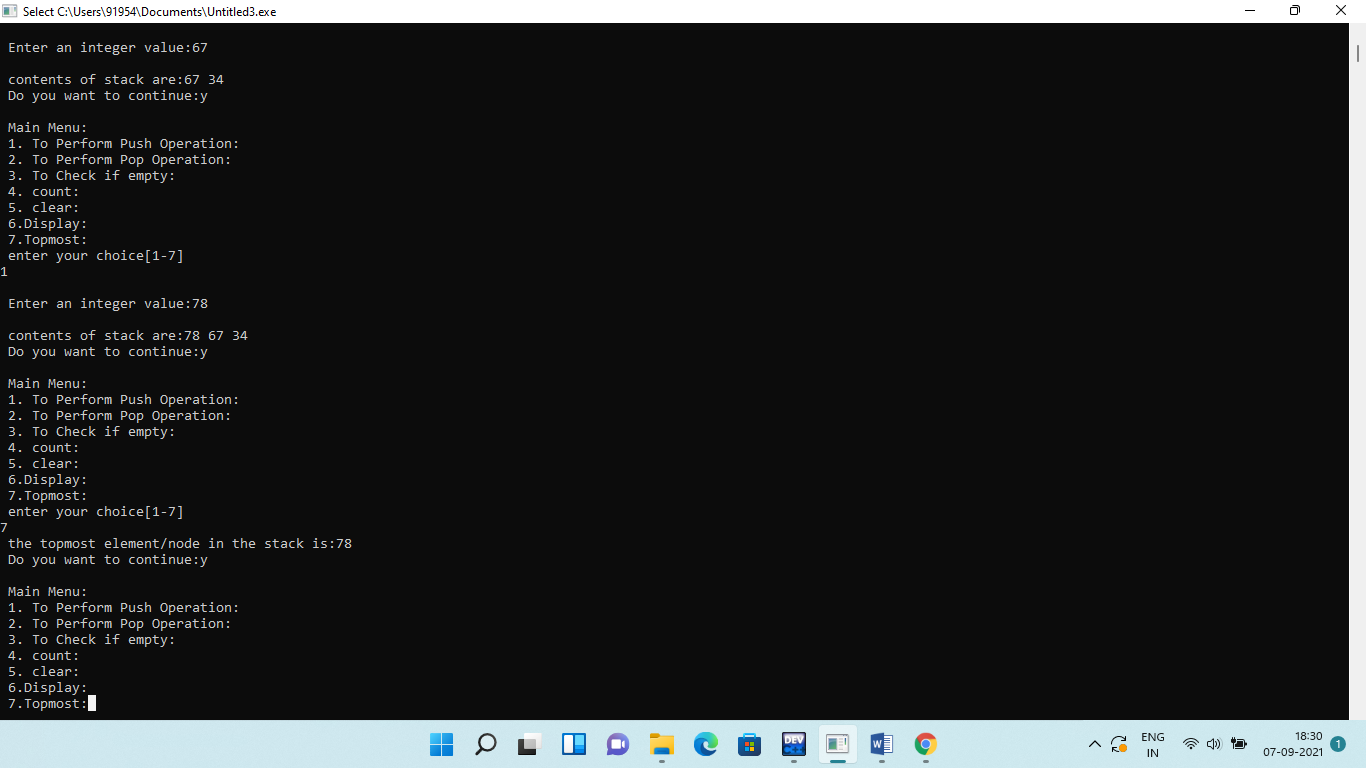
**cin>>c;**

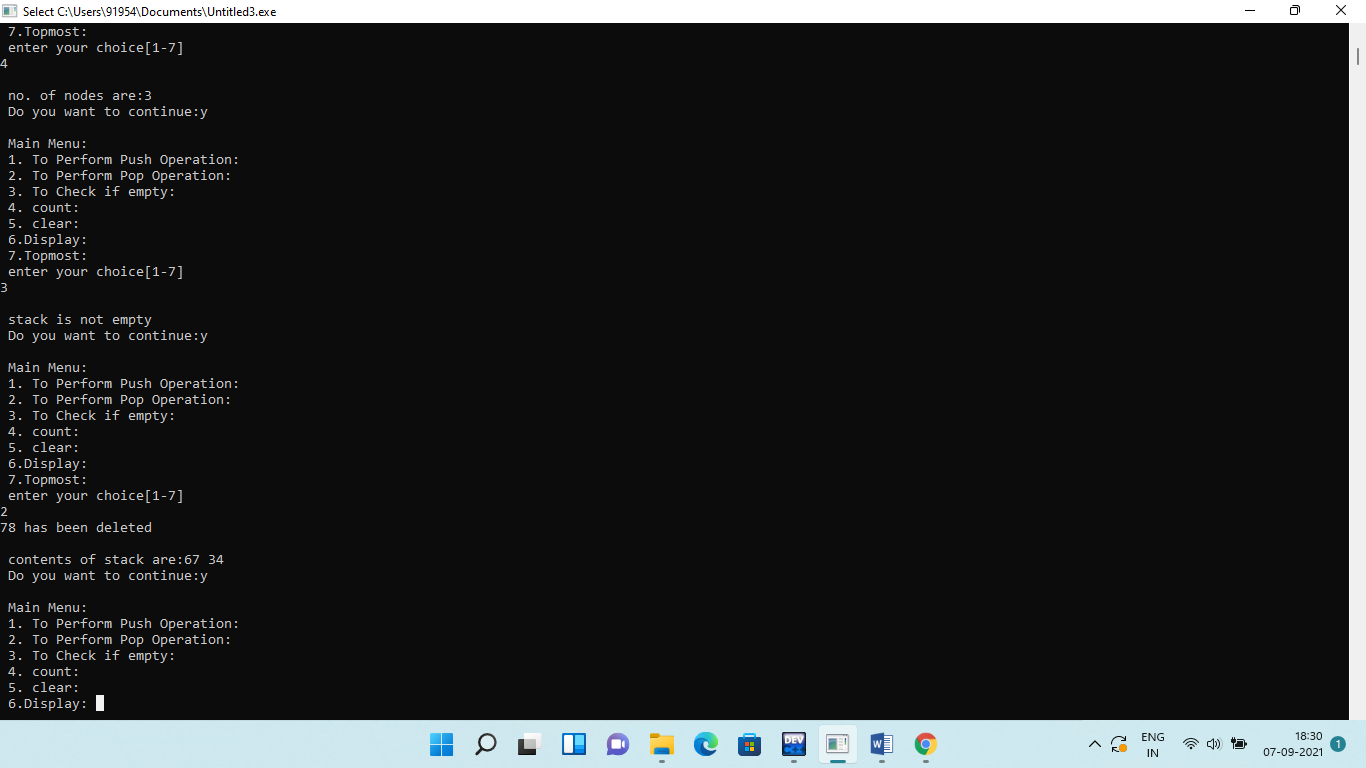
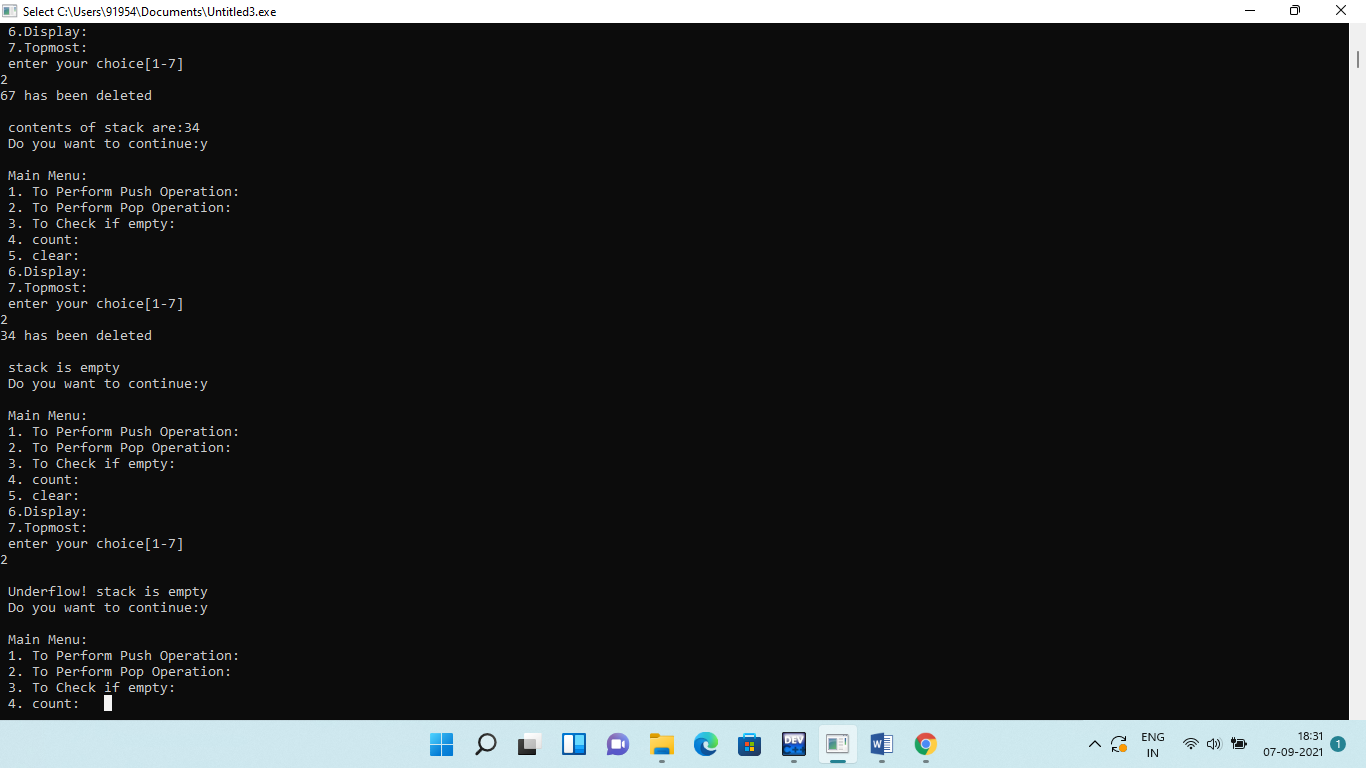
**}**

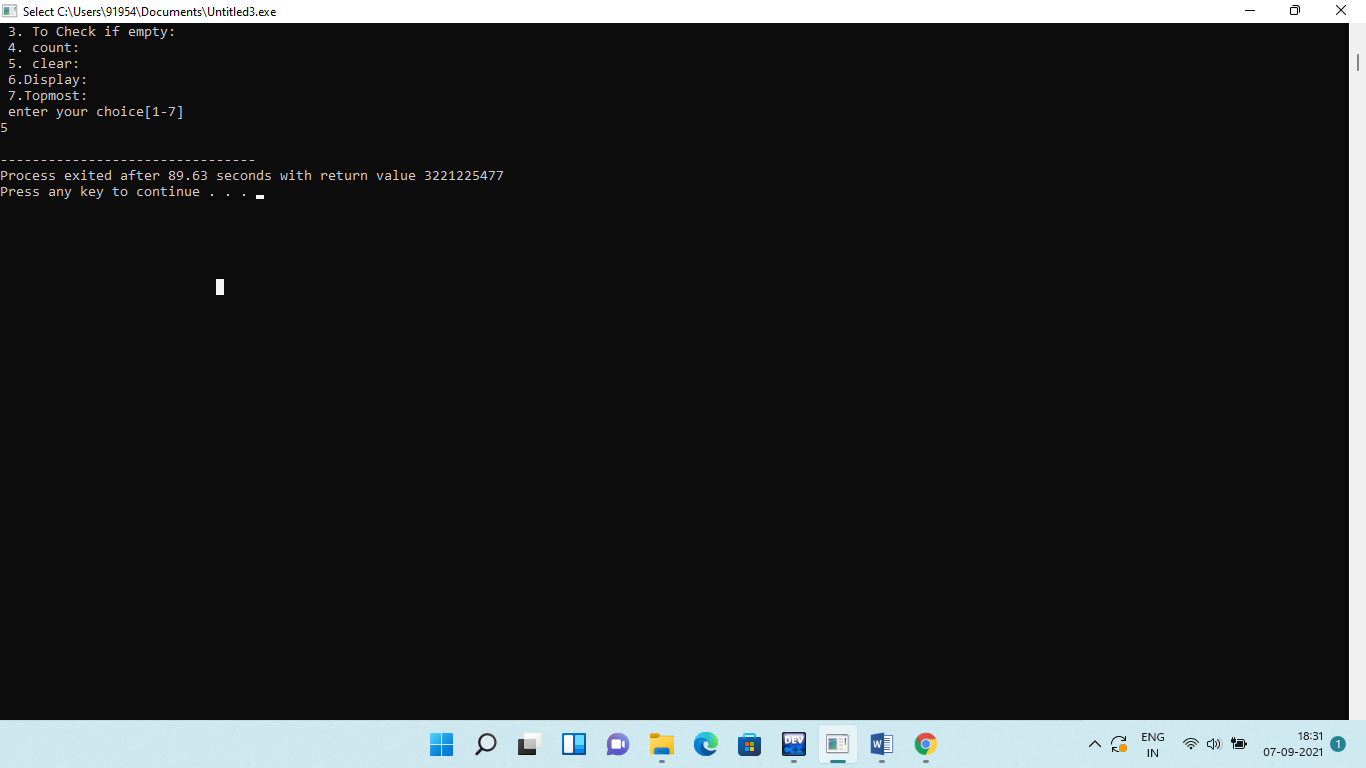
**return 0;**

**}**

**Output:**

****





**Q8.Implementation Queue using Circular Array representation.**

**Solution:**

**#include <iostream>**

**using namespace std;**

**const int SIZE = 5;**

**template<class T>class Cqueuetype**

**{**

**T cq[SIZE];**

**int f,r;**

**public:**

**void insert(T);**

**T remove();**

**int isempty();**

**int isfull();**

**void display();**

**void clear();**

**T frontmost();**

**T rearmost();**

**int count();**

**Cqueuetype()**

**{**

**f = r = -1;**

**}**

**};**

**template<class T>void Cqueuetype<T>::insert(T i)**

**{**

**if(r==-1)**

**{**

**f=r=0;**

**cq[r] = i;**

**}**

**else if(r==SIZE-1)**

**{**

**r=0;**

**cq[r]=i;**

**}**

**else**

**{**

**r++;**

**cq[r]=i;**

**}**

**}**

**template<class T>T Cqueuetype<T>::remove()**

**{**

**T rem;**

**if(f==r)**

**{**

**rem = cq[f];**

**f=r=-1;**

**}**

**else if(f==SIZE-1)**

**{**

**rem = cq[f];**

**f=0;**

**}**

**else**

**{**

**rem = cq[f];**

**f++;**

**}**

**return rem;**

**}**

**template<class T>int Cqueuetype<T>::isempty()**

**{**

**if(f==-1)**

**return 1;**

**else**

**return 0;**

**}**

**template<class T>int Cqueuetype<T>::isfull()**

**{**

**if(f==0 && r==SIZE-1)**

**return 1;**

**else if(f==r+1)**

**return 1;**

**else**

**return 0;**

**}**

**template<class T>void Cqueuetype<T>::display()**

**{**

**if(f==-1)**

**cout<<"Cqueue is empty"<<endl;**

**else if(f<=r)**

**{**

**cout<<"Contents of the Queue Starting from Front to Rear End are : ";**

**for(int j=0; j<f; j++)**

**cout<<"\_"<<"\t";**

**for(int j=f; j<=r; j++)**

**cout<<cq[j]<<"\t";**

**for(int j=r+1; j<SIZE; j++)**

**cout<<"\_"<<"\t";**

**}**

**else**

**{**

**for(int j=0; j<=r; j++)**

**cout<<cq[j]<<"\t";**

**for(int j=r+1; j<f; j++)**

**cout<<"\_"<<"\t";**

**for(int j=f; j<SIZE; j++)**

**cout<<cq[j]<<"\t";**

**}**

**cout<<endl;**

**}**

**template<class T>void Cqueuetype<T>::clear()**

**{**

**f=r=-1;**

**}**

**template<class T>T Cqueuetype<T>::frontmost()**

**{**

**T res;**

**res= cq[f];**

**return res;**

**}**

**template<class T>T Cqueuetype<T>::rearmost()**

**{**

**T res;**

**res= cq[r];**

**return res;**

**}**

**template<class T>int Cqueuetype<T>::count()**

**{**

**int c=0;**

**if(f==-1)**

**return 0;**

**else if(f<=r)**

**{**

**for(int j=f; j<=r; j++)**

**cout<<c++<<"\t";**

**}**

**else if(f>r)**

**{**

**for(int j=0; j<=r; j++)**

**cout<<c++<<"\t";**

**}**

**else**

**{**

**for(int j=f; j<SIZE; j++)**

**cout<<c++<<"\t";**

**}**

**return c;**

**}**

**int main()**

**{**

**Cqueuetype<int>cqueue;**

**char c = 'y';**

**int choice, i, rem, empty, full, count, f, r;**

**while(c=='y'||c=='Y')**

**{**

**cout<<" Have a look at the Main Menu "<<endl;**

**cout<<" 1. To insert a new element "<<endl;**

**cout<<" 2. To remove an element "<<endl;**

**cout<<" 3. To check if the Cqueue is empty "<<endl;**

**cout<<" 4. To check if the Cqueue is full "<<endl;**

**cout<<" 5. To Display the Contents of Cqueue "<<endl;**

**cout<<" 6. To Display the frontmost element "<<endl;**

**cout<<" 7. To Display the rearmost element "<<endl;**

**cout<<" 8. To count the elements in Cqueue "<<endl;**

**cout<<" 9. To Clear the Cqueue "<<endl;**

**cout<<"Enter your Choice : ";**

**cin>>choice;**

**switch(choice)**

**{**

**case 1: cout<<"--------------------------------------------------\n";**

**full = cqueue.isfull();**

**if(full == 1)**

**cout<<" overflow! Insertion not possible "<<endl;**

**else**

**{**

**cout<<"Enter the element to be inserted : ";**

**cin>>i;**

**cqueue.insert(i);**

**cqueue.display();**

**}**

**cout<<"--------------------------------------------------\n";**

**break;**

**case 2:cout<<"--------------------------------------------------\n";**

**empty = cqueue.isempty();**

**if(empty == 1)**

**cout<<"\n underflow! Deletion is not possible "<<endl;**

**else**

**{**

**rem = cqueue.remove();**

**cout<<rem<<"has been deleted"<<endl;**

**cqueue.display();**

**}**

**cout<<"--------------------------------------------------\n";**

**break;**

**case 3: cout<<"--------------------------------------------------\n";**

**empty = cqueue.isempty();**

**if(empty == 1)**

**cout<<"Cqueue is Empty "<<endl;**

**else**

**cout<<"Cqueue is not Empty"<<endl;**

**cout<<"--------------------------------------------------\n";**

**break;**

**case 4: cout<<"--------------------------------------------------\n";**

**full = cqueue.isfull();**

**if(full == 1)**

**cout<<"Cqueue is full"<<endl;**

**else**

**cout<<"Cqueue is not full"<<endl;**

**cout<<"--------------------------------------------------\n";**

**break;**

**case 5: cout<<"--------------------------------------------------\n";**

**cqueue.display();**

**cout<<"--------------------------------------------------\n";**

**break;**

**case 6: cout<<"--------------------------------------------------\n";**

**empty = cqueue.isempty();**

**if(empty!=1)**

**{**

**f = cqueue.frontmost();**

**cout<<"Frontmost Element in the Cqueue is : "<<f<<endl;**

**}**

**else**

**cout<<"\n Cqueue is empty";**

**cout<<"--------------------------------------------------\n";**

**break;**

**case 7: cout<<"--------------------------------------------------\n";**

**empty = cqueue.isempty();**

**if(empty!=1)**

**{**

**r=cqueue.rearmost();**

**cout<<"Rearmost Element in the Queue : "<<r<<endl;**

**}**

**else**

**cout<<"\n Cqueue is empty";**

**cout<<"--------------------------------------------------\n";**

**break;**

**case 8: cout<<"--------------------------------------------------\n";**

**count = cqueue.count();**

**cout<<"No. of Element in the List : "<<count<<endl;**

**cout<<"--------------------------------------------------\n";**

**break;**

**case 9: cqueue.clear();**

**break;**

**default: cout<<"Error in input"<<endl;**

**}**

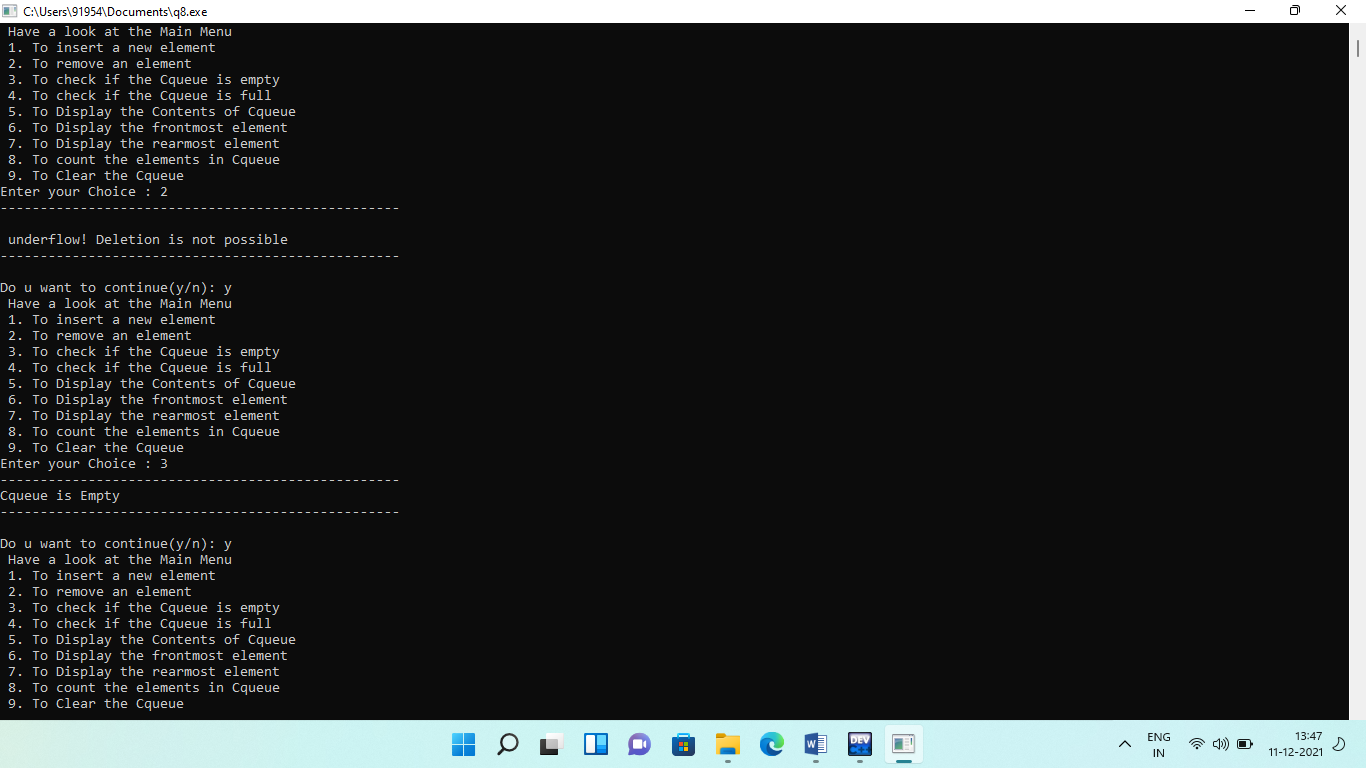
**cout<<"\nDo u want to continue(y/n): ";**

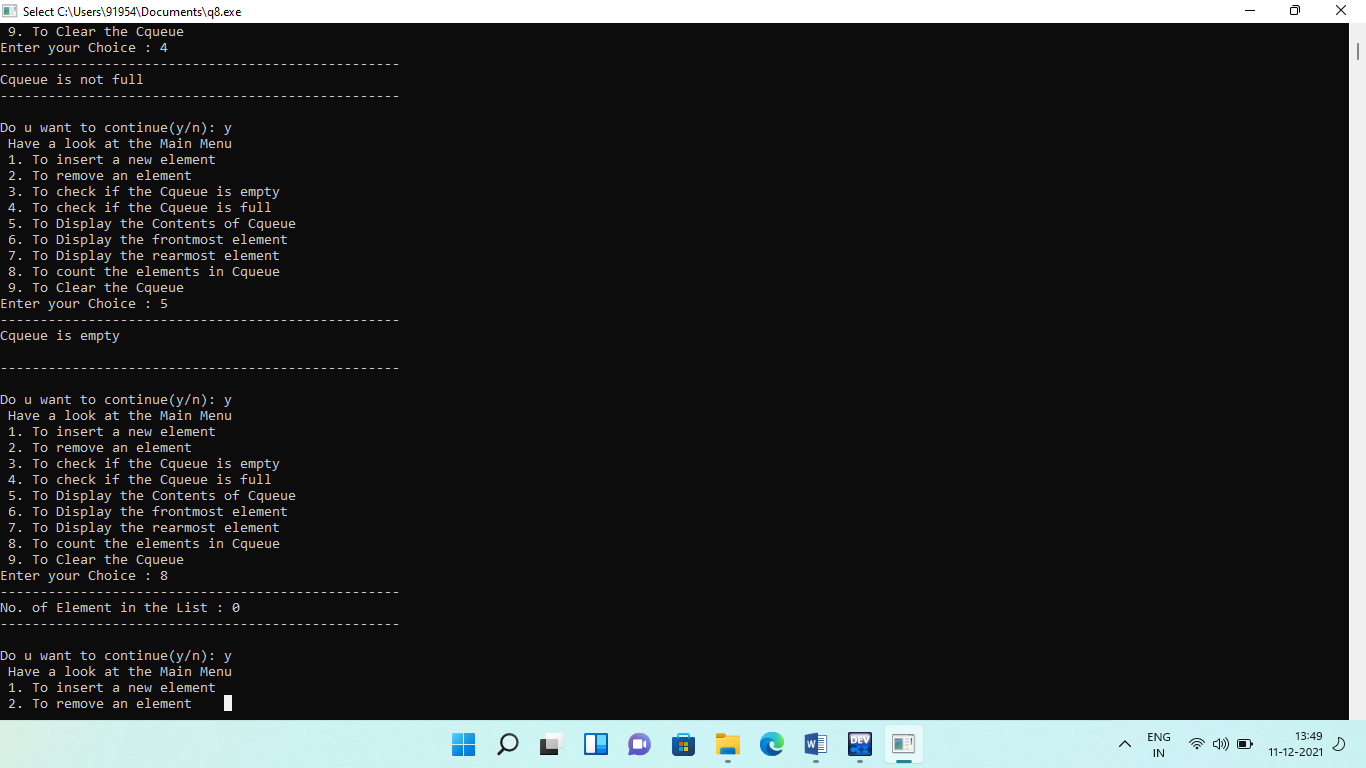
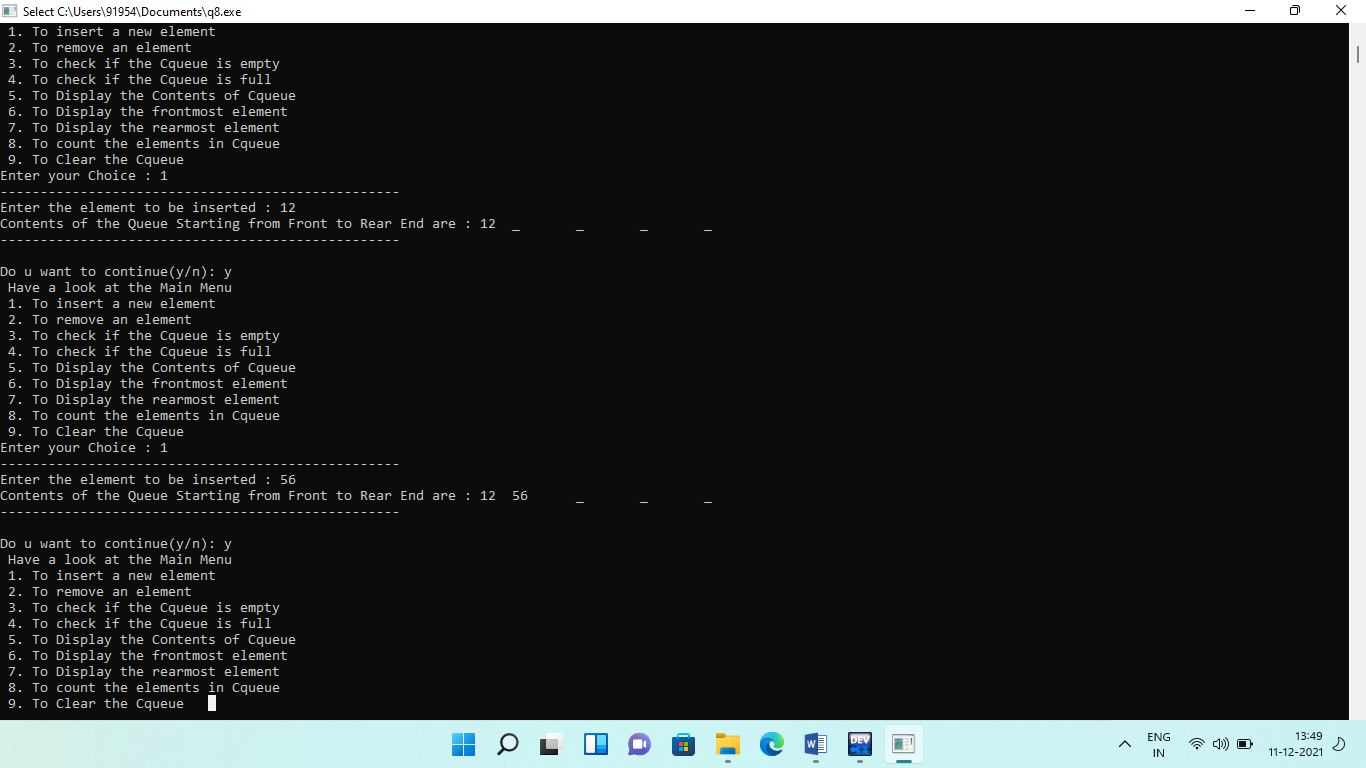
**cin>>c;**

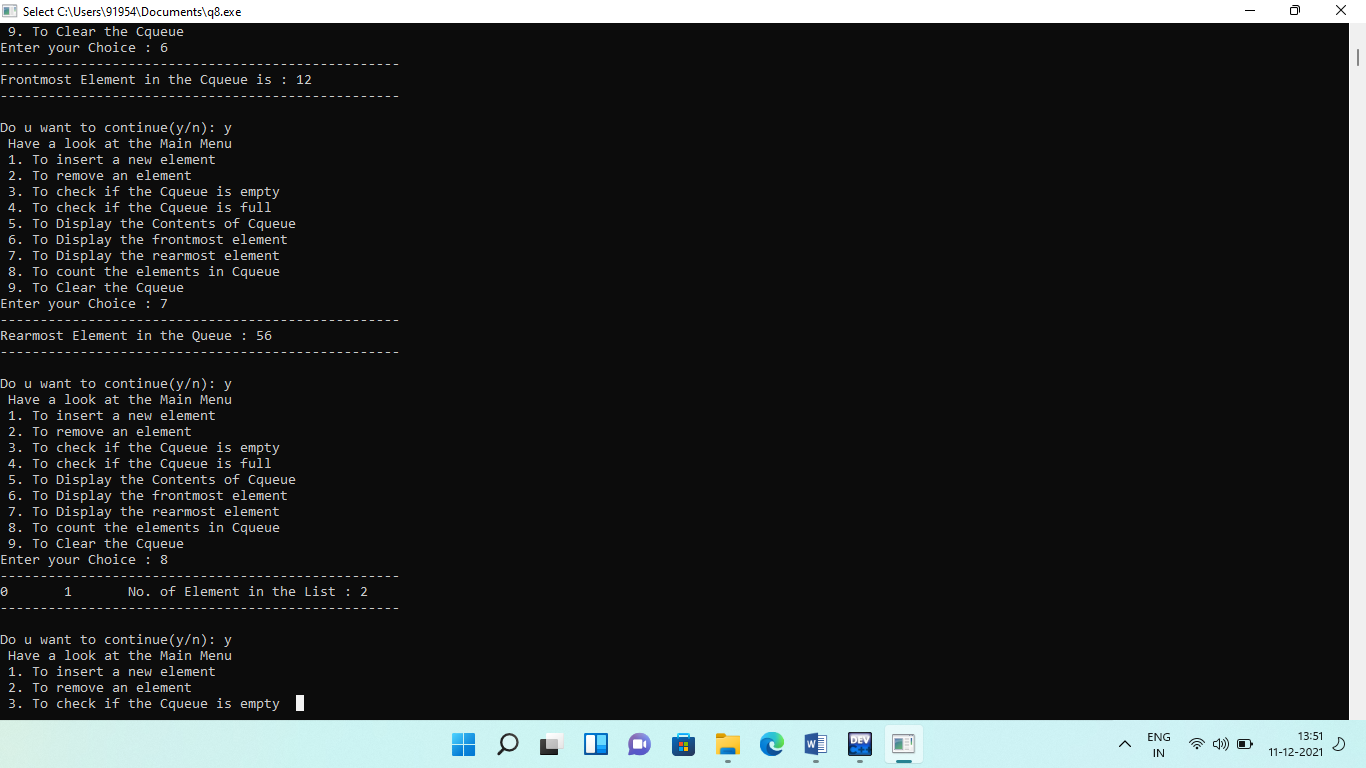
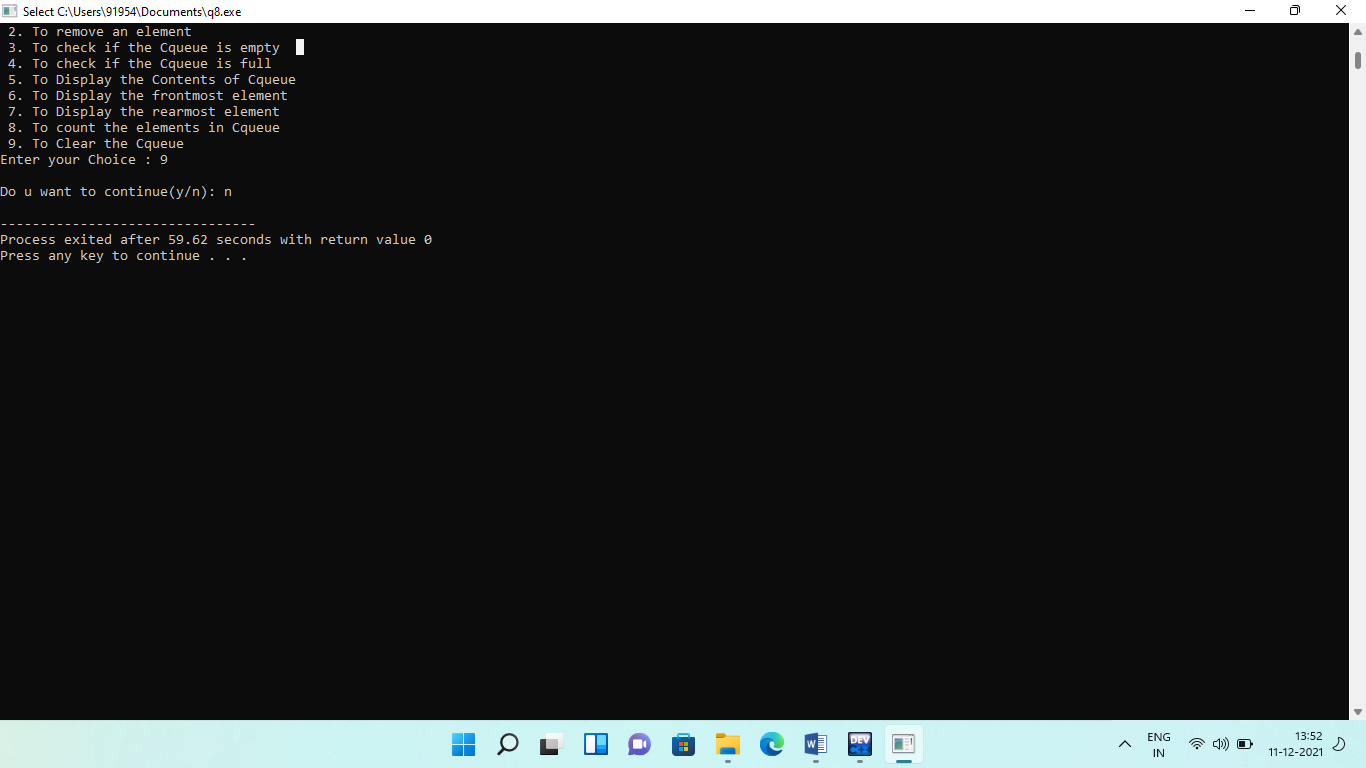
**}**

**return 0;**

**}**

**Output:**





**Q11. Write a program to implement Binary Search Tree which supports the following operations:**

**(i) Insert an element x**

**(ii) Delete an element x**

**(iii) Search for an element x in the BST and change its value to y and then place the node with**

**value y at its appropriate position in the BST**

**(iv) Display the elements of the BST in preorder, inorder, and postorder traversal**

**(v) Display the elements of the BST in level-by-level traversal**

**(vi) Display the height of the BST**

**Solution:**

**# include <iostream>**

**# include <cstdlib>**

**using namespace std;**

**struct node //node declaration**

**{**

**int info;**

**node \*l;**

**node \*r;**

**}\*r;**

**template<class T>**

**class BST**

**{**

**public://functions declaration**

**void find(T, node \*\*, node \*\*);**

**void search(node \*, T);**

**void insert(node \*, node \*);**

**void del(T);**

**void casea(node \*,node \*);**

**void caseb(node \*,node \*);**

**void casec(node \*,node \*);**

**void preorder(node \*);**

**void inorder(node \*);**

**void postorder(node \*);**

**void display(node \*, T);**

**T height(node\*);**

**BST()**

**{**

**r = NULL;**

**}**

**};**

**template<class T>**

**void BST<T>::find(T i, node \*\*par, node \*\*loc)//find the position of the item**

**{**

**node \*ptr, \*ptrsave;**

**if (r == NULL)**

**{**

**\*loc = NULL;**

**\*par = NULL;**

**return;**

**}**

**if (i == r->info)**

**{**

**\*loc = r;**

**\*par = NULL;**

**return;**

**}**

**if (i < r->info)**

**ptr = r->l;**

**else**

**ptr = r->r;**

**ptrsave = r;**

**while (ptr != NULL)**

**{**

**if (i == ptr->info)**

**{**

**\*loc = ptr;**

**\*par = ptrsave;**

**return;**

**}**

**ptrsave = ptr;**

**if (i < ptr->info)**

**ptr = ptr->l;**

**else**

**ptr = ptr->r;**

**}**

**\*loc = NULL;**

**\*par = ptrsave;**

**}**

**template<class T>**

**T BST<T>::height(node\* node)**

**{**

**if(node==NULL)**

**return 0;**

**else**

**{**

**int Lheight=height(node->l);**

**int Rheight=height(node->r);**

**if(Lheight>Rheight)**

**return (Lheight+1);**

**else**

**return (Rheight +1);**

**}**

**}**

**template<class T>**

**void BST<T>::search(node \*root, T data) //searching**

**{**

**T y;**

**T depth = 0;**

**node \*temp = new node;**

**temp = root;**

**while(temp != NULL)**

**{**

**depth++;**

**if(temp->info == data)**

**{**

**cout<<"\nData found at depth: "<<depth<<endl;**

**cout<<"change the value of x to y:";**

**cin>>y;**

**temp->info=y;**

**return;**

**}**

**else if(temp->info > data)**

**temp = temp->l;**

**else**

**temp = temp->r;**

**}**

**cout<<"\n Data not found"<<endl;**

**return;**

**}**

**template<class T>**

**void BST<T>::insert(node \*tree, node \*newnode)**

**{**

**if (r == NULL)**

**{**

**r = new node;**

**r->info = newnode->info;**

**r->l= NULL;**

**r->r= NULL;**

**cout<<"Root Node is Added"<<endl;**

**return;**

**}**

**if (tree->info == newnode->info)**

**{**

**cout<<"Element already in the tree"<<endl;**

**return;**

**}**

**if (tree->info > newnode->info)**

**{**

**if (tree->l != NULL)**

**{**

**insert(tree->l, newnode);**

**}**

**else**

**{**

**tree->l= newnode;**

**(tree->l)->l = NULL;**

**(tree->l)->r= NULL;**

**cout<<"Node Added To Left"<<endl;**

**return;**

**}**

**}**

**else**

**{**

**if (tree->r != NULL)**

**{**

**insert(tree->r, newnode);**

**}**

**else**

**{**

**tree->r = newnode;**

**(tree->r)->l= NULL;**

**(tree->r)->r = NULL;**

**cout<<"Node Added To Right"<<endl;**

**return;**

**}**

**}**

**}**

**template<class T>**

**void BST<T>::del(T i)**

**{**

**node \*par, \*loc;**

**if (r == NULL)**

**{**

**cout<<"Tree empty"<<endl;**

**return;**

**}**

**find(i, &par, &loc);**

**if (loc == NULL)**

**{**

**cout<<"Item not present in tree"<<endl;**

**return;**

**}**

**if (loc->l == NULL && loc->r == NULL)**

**{**

**casea(par, loc);**

**cout<<"item deleted"<<endl;**

**}**

**if (loc->l!= NULL && loc->r == NULL)**

**{**

**caseb(par, loc);**

**cout<<"item deleted"<<endl;**

**}**

**if (loc->l== NULL && loc->r != NULL)**

**{**

**caseb(par, loc);**

**cout<<"item deleted"<<endl;**

**}**

**if (loc->l != NULL && loc->r != NULL)**

**{**

**casec(par, loc);**

**cout<<"item deleted"<<endl;**

**}**

**free(loc);**

**}**

**template<class T>**

**void BST<T>::casea(node \*par, node \*loc )**

**{**

**if (par == NULL)**

**{**

**r= NULL;**

**}**

**else**

**{**

**if (loc == par->l)**

**par->l = NULL;**

**else**

**par->r = NULL;**

**}**

**}**

**template<class T>**

**void BST<T>::caseb(node \*par, node \*loc)**

**{**

**node \*child;**

**if (loc->l!= NULL)**

**child = loc->l;**

**else**

**child = loc->r;**

**if (par == NULL)**

**{**

**r = child;**

**}**

**else**

**{**

**if (loc == par->l)**

**par->l = child;**

**else**

**par->r = child;**

**}**

**}**

**template<class T>**

**void BST<T>::casec(node \*par, node \*loc)**

**{**

**node \*ptr, \*ptrsave, \*suc, \*parsuc;**

**ptrsave = loc;**

**ptr = loc->r;**

**while (ptr->l!= NULL)**

**{**

**ptrsave = ptr;**

**ptr = ptr->l;**

**}**

**suc = ptr;**

**parsuc = ptrsave;**

**if (suc->l == NULL && suc->r == NULL)**

**casea(parsuc, suc);**

**else**

**caseb(parsuc, suc);**

**if (par == NULL)**

**{**

**r = suc;**

**}**

**else**

**{**

**if (loc == par->l)**

**par->l = suc;**

**else**

**par->r= suc;**

**}**

**suc->l = loc->l;**

**suc->r= loc->r;**

**}**

**template<class T>**

**void BST<T>::preorder(node \*ptr)**

**{**

**if (r == NULL)**

**{**

**cout<<"Tree is empty"<<endl;**

**return;**

**}**

**if (ptr != NULL)**

**{**

**cout<<ptr->info<<" ";**

**preorder(ptr->l);**

**preorder(ptr->r);**

**}**

**}**

**template<class T>**

**void BST<T>::inorder(node \*ptr)//inorder traversal**

**{**

**if (r == NULL)**

**{**

**cout<<"Tree is empty"<<endl;**

**return;**

**}**

**if (ptr != NULL)**

**{**

**inorder(ptr->l);**

**cout<<ptr->info<<" ";**

**inorder(ptr->r);**

**}**

**}**

**template<class T>**

**void BST<T>::postorder(node \*ptr)//postorder traversal**

**{**

**if (r == NULL)**

**{**

**cout<<"Tree is empty"<<endl;**

**return;**

**}**

**if (ptr != NULL)**

**{**

**postorder(ptr->l);**

**postorder(ptr->r);**

**cout<<ptr->info<<" ";**

**}**

**}**

**template<class T>**

**void BST<T>::display(node \*ptr, T level)//print the tree**

**{**

**int i;**

**if (ptr != NULL)**

**{**

**display(ptr->l, level+1);**

**cout<<endl;**

**if (ptr == r)**

**cout<<"Root:-";**

**else**

**{**

**for (i = 0;i <level;i++)**

**cout<<"";**

**}**

**cout<<ptr->info<<" ";**

**display(ptr->r, level+1);**

**}**

**}**

**int main()**

**{**

**int c, n,item,h;**

**BST<int> bst;**

**node \*t=NULL;**

**while (1)**

**{**

**cout<<"\n MAIN MENU:-"<<endl;**

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

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

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

**cout<<"4.Inorder Traversal"<<endl;**

**cout<<"5.Preorder Traversal"<<endl;**

**cout<<"6.Postorder Traversal"<<endl;**

**cout<<"7.Display the tree"<<endl;**

**cout<<"8.Height of BST"<<endl;**

**cout<<"9.Exit"<<endl;**

**cout<<"Enter your choice : ";**

**cin>>c;**

**switch(c)**

**{**

**case 1:**

**cout<<"==========================================================================\n";**

**t = new node;**

**cout<<"Enter the number to be inserted : ";**

**cin>>t->info;**

**bst.insert(r, t);**

**cout<<"==========================================================================\n";**

**break;**

**case 2:**

**cout<<"==========================================================================\n";**

**if (r == NULL)**

**{**

**cout<<"Tree is empty, nothing to delete"<<endl;**

**continue;**

**}**

**cout<<"Enter the number to be deleted : ";**

**cin>>n;**

**bst.del(n);**

**cout<<"==========================================================================\n";**

**break;**

**case3: cout<<"==========================================================================\n";**

**cout<<"Search:"<<endl;**

**cin>>item;**

**bst.search(r,item);**

**cout<<"==========================================================================\n";**

**break;**

**case4: cout<<"==========================================================================\n";**

**cout<<"Inorder Traversal of BST:"<<endl;**

**bst.inorder(r);**

**cout<<endl;**

**cout<<"==========================================================================\n";**

**break;**

**case5:**

**cout<<"==========================================================================\n";**

**cout<<"Preorder Traversal of BST:"<<endl;**

**bst.preorder(r);**

**cout<<endl;**

**cout<<"==========================================================================\n";**

**break;**

**case6: cout<<"==========================================================================\n";**

**cout<<"Postorder Traversal of BST:"<<endl;**

**bst.postorder(r);**

**cout<<endl;**

**cout<<"==========================================================================\n";**

**break;**

**case7: cout<<"==========================================================================\n";**

**cout<<"Display BST:"<<endl;**

**bst.display(r,1);**

**cout<<endl;**

**cout<<"==========================================================================\n";**

**break;**

**case8: cout<<"==========================================================================\n";**

**h=bst.height(t);**

**cout<<"Height of BST:"<<h<<endl;**

**cout<<"==========================================================================\n";**

**break;**

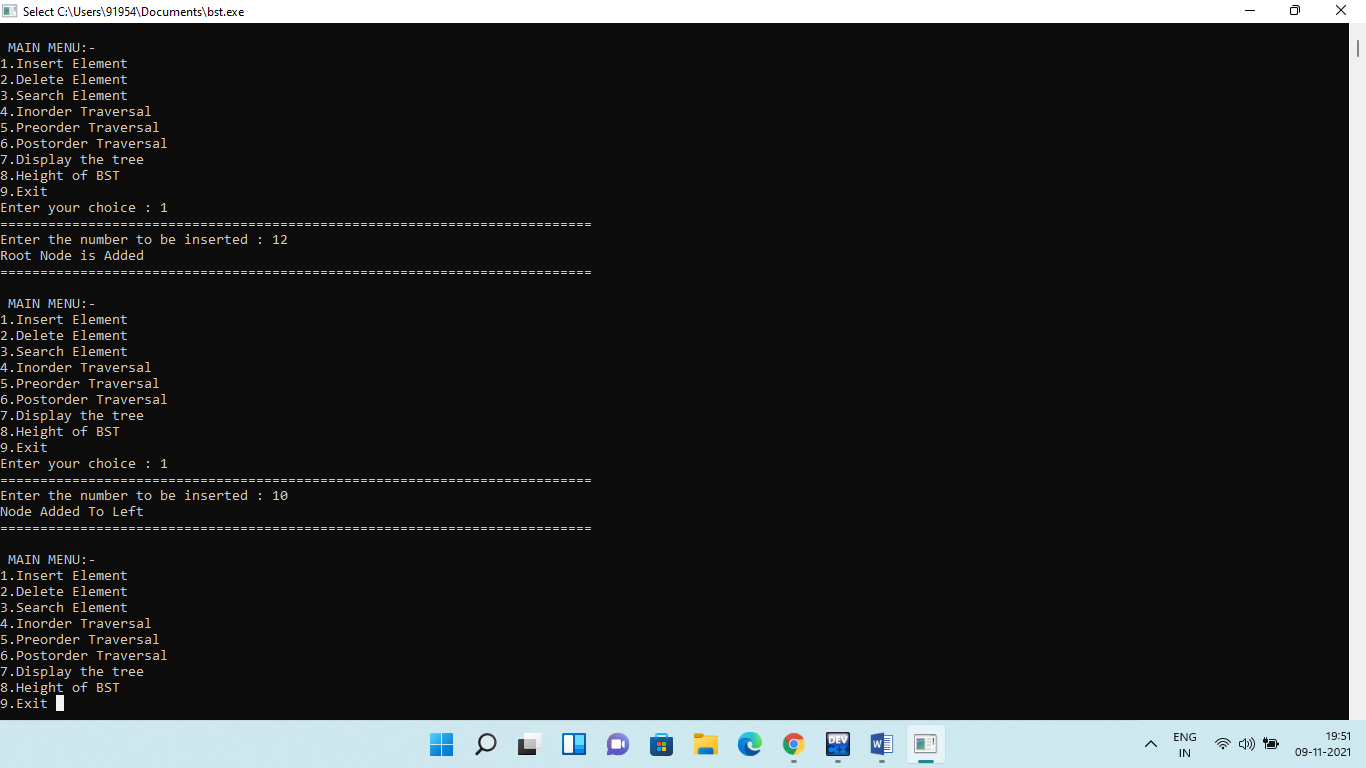
**case 9: exit(1);**

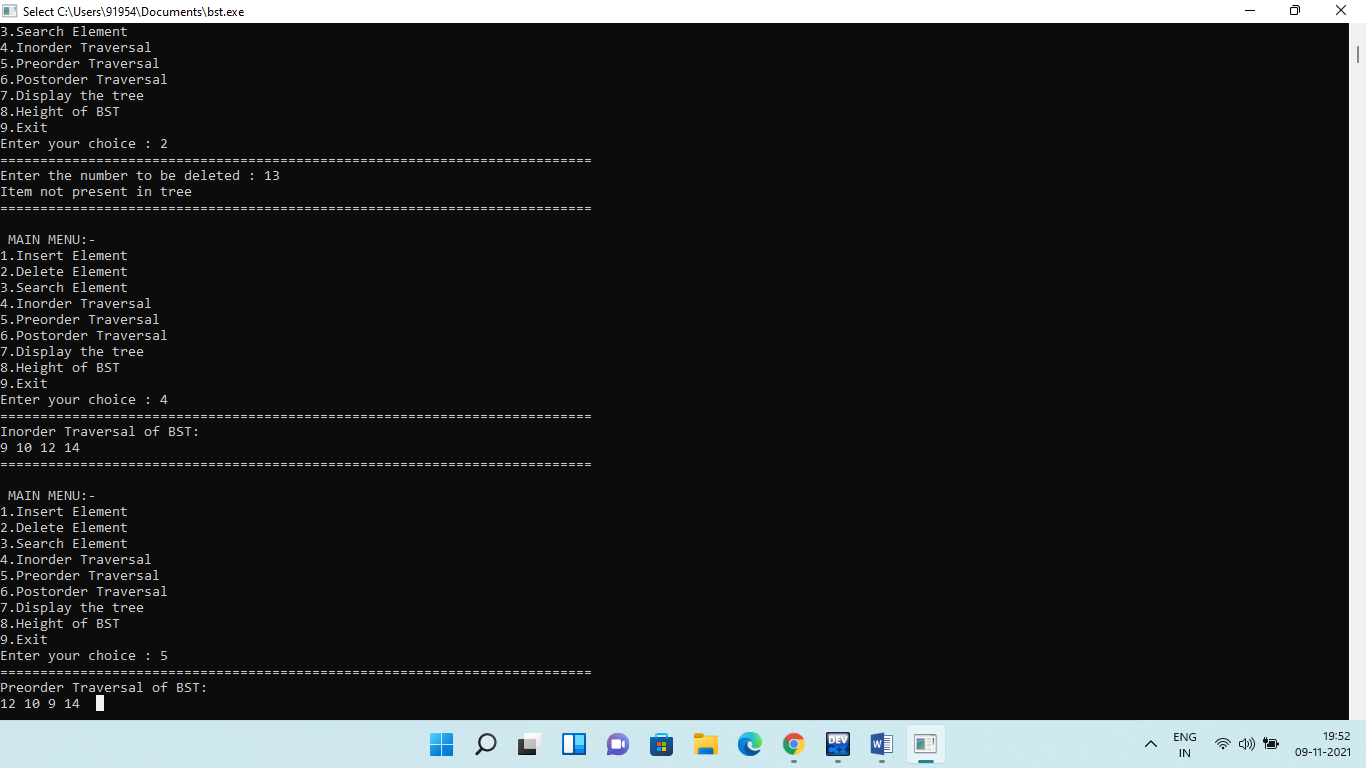
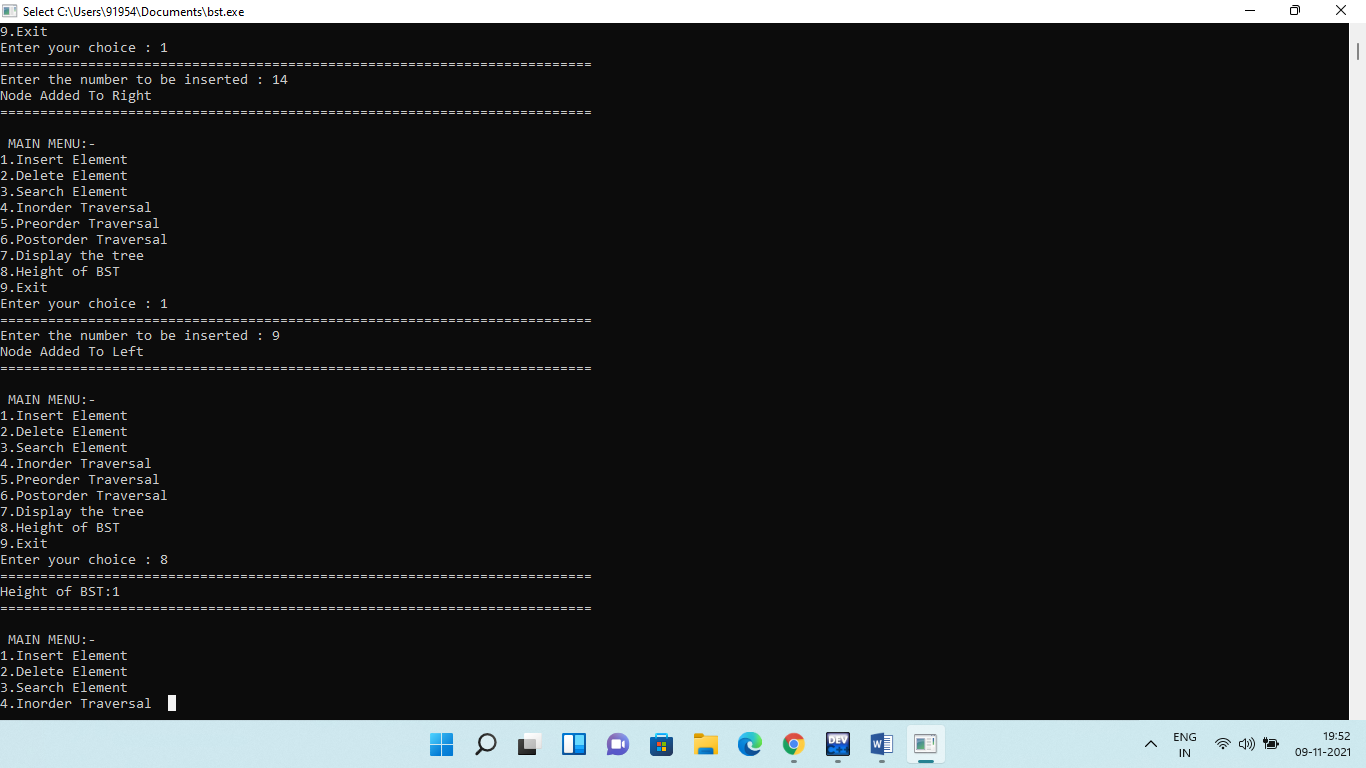
**default:**

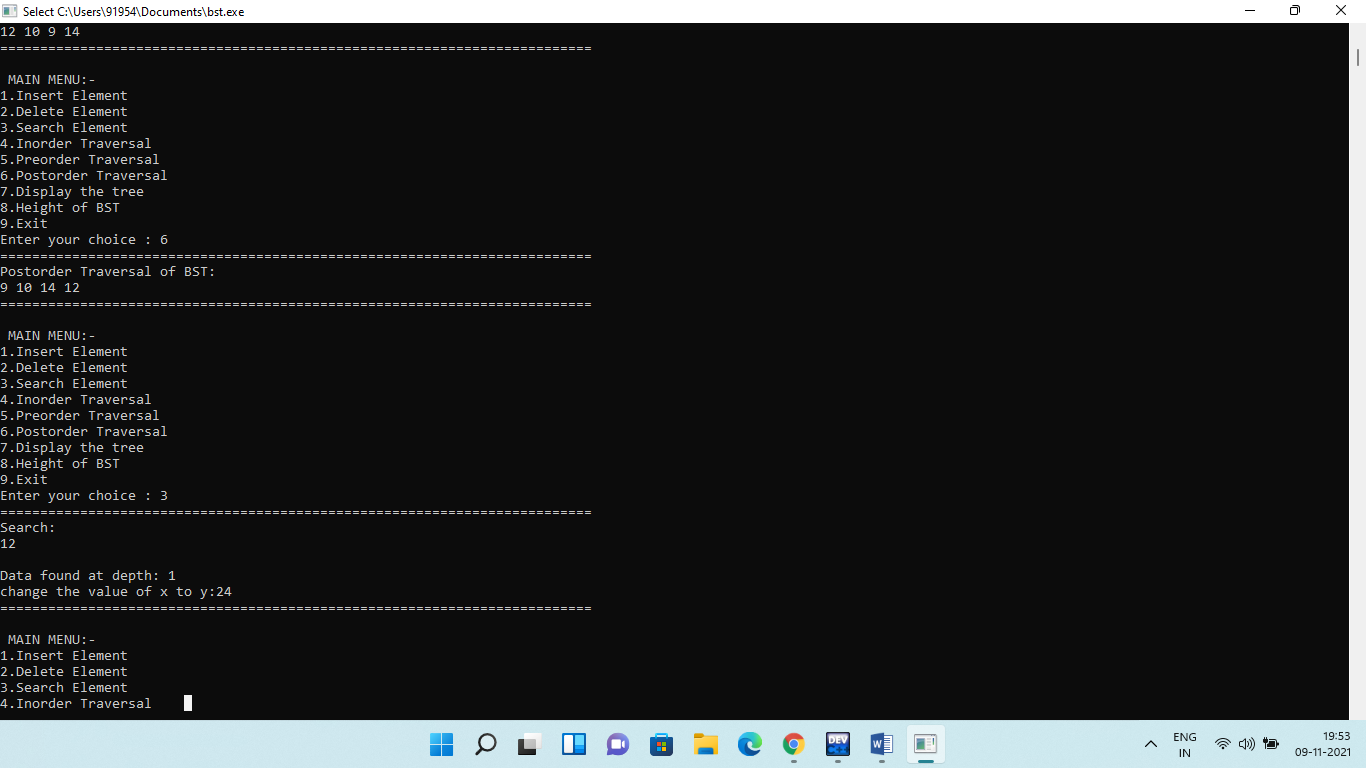
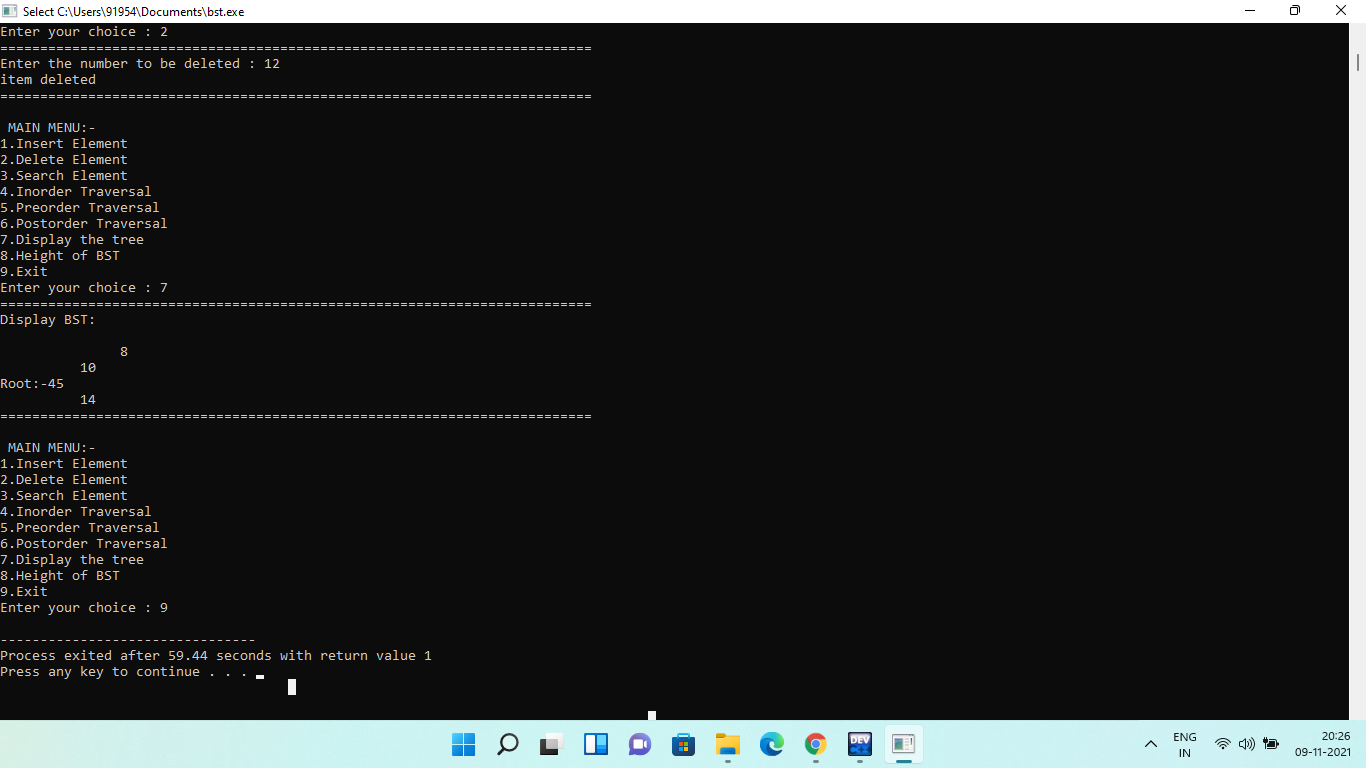
**cout<<"Wrong choice"<<endl;**

**}**

**}**

**Output:**

****

****

**Q12.** **Write a program, using templates, to sort a list of n elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.**

**Solution:**

**#include<iostream>**

**using namespace std;**

**template<class T>**

**void accept(T Arr[], T s)**

**{**

**for(int i=0;i<s;i++)**

**{**

**cout<<"Enter element "<<i+1<<":";**

**cin>>Arr[i];**

**}**

**}**

**template<class T>**

**void display(T Arr[],T s)**

**{**

**cout<<"The elements of the array are:\n";**

**for(int i=0;i<s;i++)**

**cout<<Arr[i]<<" ";**

**}**

**template<class T>**

**void isort(T Arr[], T s)**

**{**

**int i,j,Temp;**

**for(i=1;i<s;i++)**

**{**

**Temp=Arr[i];**

**j=i-1;**

**while((Temp<Arr[j]) && (j>=0))**

**{**

**Arr[j+1]=Arr[j];**

**j--;**

**}**

**Arr[j+1]=Temp;**

**}**

**}**

**template<class T>**

**void ssort(T Arr[], T s)**

**{**

**int i,j,Temp,Small;**

**for(i=0;i<s-1;i++)**

**{**

**Small=i;**

**for(j=i+1;j<s;j++) //finding the smallest element**

**if(Arr[j]<Arr[Small])**

**Small=j;**

**if(Small!=i)**

**{**

**Temp=Arr[i]; //Swapping**

**Arr[i]=Arr[Small];**

**Arr[Small]=Temp;**

**}**

**}**

**}**

**template<class T>**

**void bsort(T Arr[],T s)**

**{**

**int i,j,Temp;**

**for(i=0;i<s-1;i++)**

**{**

**for(j=0;j<(s-1-i);j++)**

**if(Arr[j]>Arr[j+1])**

**{**

**Temp=Arr[j]; //swapping**

**Arr[j]=Arr[j+1];**

**Arr[j+1]=Temp;**

**}**

**}**

**}**

**int main()**

**{**

**int Arr[100],n,choice;**

**cout<<"Enter Size of Array ";**

**cin>>n;**

**do**

**{**

**cout<<"\n\n MAIN MENU";**

**cout<<"\n1. Accept elements of array";**

**cout<<"\n2. Display elements of array";**

**cout<<"\n3. Sort the array using insertion sort method";**

**cout<<"\n4. Sort the array using selection sort method";**

**cout<<"\n5. Sort the array using bubble sort method";**

**cout<<"\n6. Exit";**

**cout<<"\n\nEnter your choice 1-5 :";**

**cin>>choice;**

**switch(choice)**

**{**

**case 1: accept(Arr,n);**

**break;**

**case 2: display(Arr,n);**

**break;**

**case 3: isort(Arr,n);**

**cout<<"\nAfter insertation sort :-\n ";**

**display(Arr,n);**

**break;**

**case 4: ssort(Arr,n);**

**cout<<"\nAfter Selection sort :- \n";**

**display(Arr,n);**

**break;**

**case 5: bsort(Arr,n);**

**cout<<"\nAfter Bubble sort :- \n";**

**display(Arr,n);**

**break;**

**default: cout<<"\nInvalid choice";**

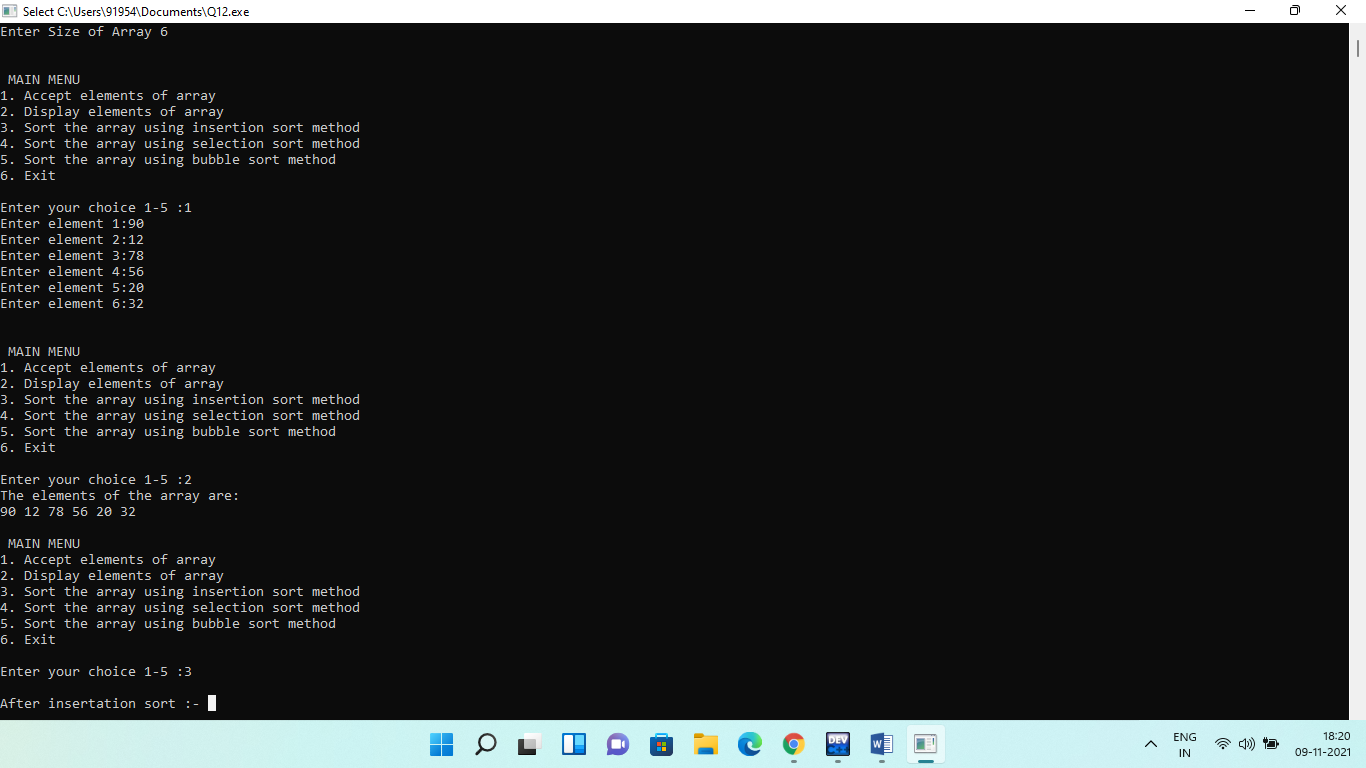
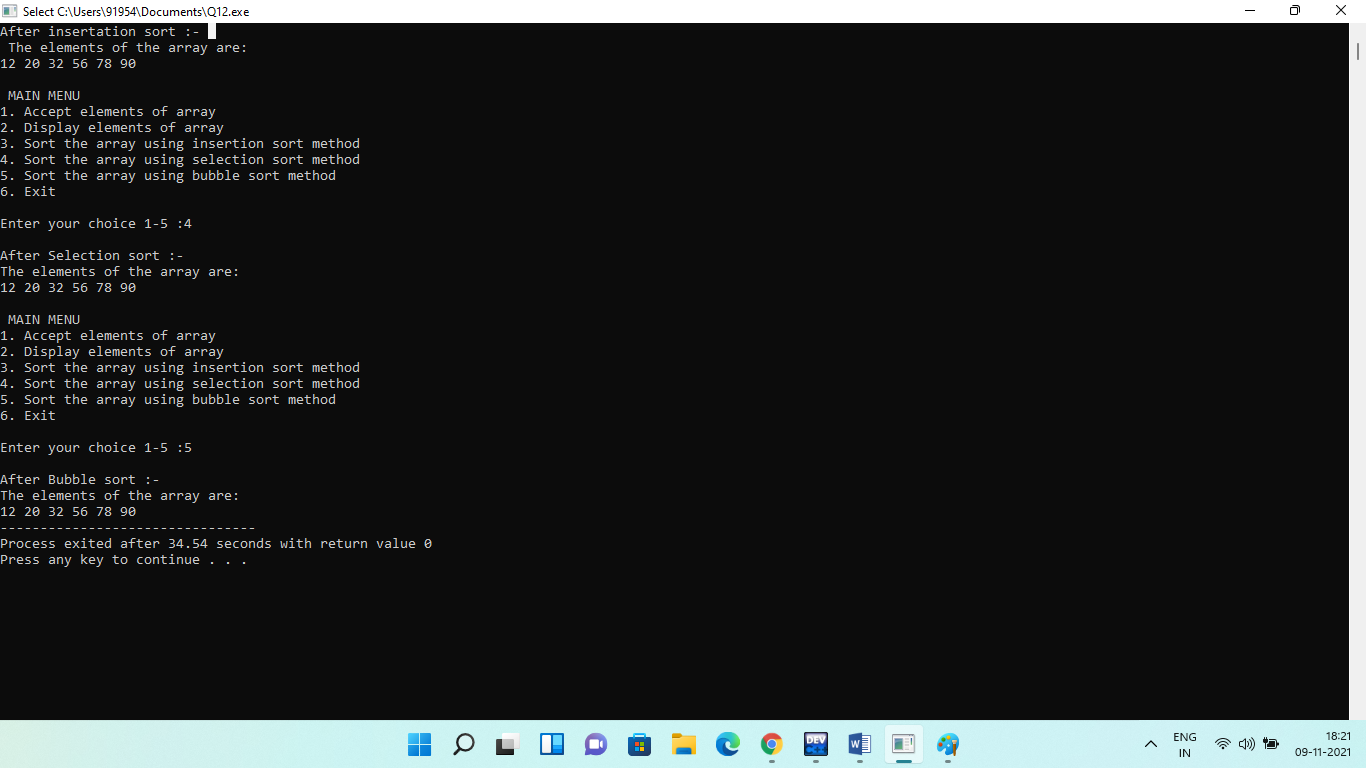
**}**

**}while(choice!=5);**

**return 0;**

**}**

**Output:**

****

**Q13.** **Write a program to implement:**

**(i) Diagonal Matrix using one-dimensional array.**

**(ii) Lower Triangular Matrix using one-dimensional array**

**(iii) Upper Triangular Matrix using one-dimensional array**

**(iv) Symmetric Matrix using one-dimensional array**

**Solution:**

**#include<iostream>**

**using namespace std;**

**template<class t>**

**class Diagonal**

**{**

**t\*a;**

**int size;**

**public:**

**Diagonal(int s1)**

**{**

**size=s1;**

**a=new t[size];**

**}**

**void Store (int i,int j,t x);**

**t Retrieve(int i ,int j);**

**};**

**template<class t>**

**void Diagonal<t>::Store(int i,int j,t x)**

**{**

**if((i<1) || (j<1) || (i>size)||(j>size))**

**{**

**cout<<"INVALID INDEX......!!!!!!!"<<endl;**

**return;**

**}**

**else if(i==j)**

**a[i-1]=x;**

**else if(x!=0)**

**{**

**cout<<"X MUST BE ZERO....!!!!!!!"<<endl;**

**return;**

**}**

**}**

**template<class t>**

**t Diagonal<t>::Retrieve(int i,int j)**

**{**

**if((i<1) || (j<1) || (i>size)||(j>size))**

**{**

**cout<<"INVALID INDEX......!!!!!!!"<<endl;**

**exit(1);**

**}**

**else if(i==j)**

**return a[i-1];**

**else**

**return 0;**

**}**

**template<class t>**

**class Lowtri**

**{**

**t\*a;**

**int size;**

**public:**

**Lowtri(int s1)**

**{**

**size=s1\*(s1+1)/2;**

**a=new t[size];**

**}**

**void Store (int i,int j,t x);**

**t Retrieve(int i ,int j);**

**};**

**template<class t>**

**void Lowtri<t>::Store(int i,int j,t x)**

**{**

**if((i<1) || (j<1) || (i>size)||(j>size))**

**{**

**cout<<"INVALID INDEX......!!!!!!!"<<endl;**

**return;**

**}**

**else if(i>=j)**

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

**else if(x!=0)**

**{**

**cout<<"X MUST BE ZERO....!!!!!!!"<<endl;**

**return;**

**}**

**}**

**template<class t>**

**t Lowtri<t>::Retrieve(int i,int j)**

**{**

**if((i<1) || (j<1) || (i>size)||(j>size))**

**{**

**cout<<"INVALID INDEX......!!!!!!!"<<endl;**

**exit(1);**

**}**

**else if(i>=j)**

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

**else**

**return 0;**

**}**

**template<class t>**

**class Uptri**

**{**

**t\*a;**

**int size;**

**public:**

**Uptri(int s1)**

**{**

**size=s1\*(s1+1)/2;**

**a=new t[size];**

**}**

**void Store (int i,int j,t x);**

**t Retrieve(int i ,int j);**

**};**

**template<class t>**

**void Uptri<t>::Store(int i,int j,t x)**

**{**

**if((i<1) || (j<1) || (i>size)||(j>size))**

**{**

**cout<<"INVALID INDEX......!!!!!!!"<<endl;**

**return;**

**}**

**else if(j>=i)**

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

**else if(x!=0)**

**{**

**cout<<"X MUST BE ZERO....!!!!!!!"<<endl;**

**return;**

**}**

**}**

**template<class t>**

**t Uptri<t>::Retrieve(int i,int j)**

**{**

**if((i<1) || (j<1) || (i>size)||(j>size))**

**{**

**cout<<"INVALID INDEX......!!!!!!!"<<endl;**

**exit(1);**

**}**

**else if(j>=i)**

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

**else**

**return 0;**

**}**

**template<class t>**

**class Symmetric**

**{**

**t\*a;**

**int size;**

**public:**

**Symmetric(int s1)**

**{**

**size=s1\*(s1+1)/2;**

**a=new t[size];**

**}**

**void Store (int i,int j,t x);**

**t Retrieve(int i ,int j);**

**};**

**template<class t>**

**void Symmetric<t>::Store(int i,int j,t x)**

**{**

**if((i<1) || (j<1) || (i>size)||(j>size))**

**{**

**cout<<"INVALID INDEX......!!!!!!!"<<endl;**

**return;**

**}**

**else if(i>=j)**

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

**else if(x!=0)**

**{**

**cout<<"X MUST BE ZERO....!!!!!!!"<<endl;**

**return;**

**}**

**}**

**template<class t>**

**t Symmetric<t>::Retrieve(int i,int j)**

**{**

**if((i<1) || (j<1) || (i>size)||(j>size))**

**{**

**cout<<"INVALID INDEX......!!!!!!!"<<endl;**

**exit(1);**

**}**

**else if(i>=j)**

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

**else**

**return 0;**

**}**

**int main()**

**{**

**int x,size,choice;**

**char ch;**

**cout<<"\n ENTER THE SIZE OF MATRIX.............."<<endl;**

**cin>>size;**

**Diagonal<int> d(size);**

**Lowtri<int> l(size);**

**Uptri<int> u(size);**

**Symmetric<int>s(size);**

**do**

**{**

**cout<<"\n ----------------MAIN MENU-----------------------"<<endl;**

**cout<<"1.DIAGONAL MATRIX"<<endl;**

**cout<<"2.LOWERTRIANGULAR MATRIX"<<endl;**

**cout<<"3.UPPERTRIANGULAR MATRIX"<<endl;**

**cout<<"4.SYMMETRIC MATRIX"<<endl;**

**cout<<"------------------------------------------------------"<<endl;**

**cout<<"ENTER YOUR CHOICE :: "<<endl;**

**cin>>choice;**

**switch(choice)**

**{**

**case 1: cout<<"\n ENTER THE ELEMENTS ::"<<endl;**

**for(int i=1;i<=size;i++)**

**for(int j=1;j<=size;j++)**

**{**

**cout<<"\n";**

**cin>>x;**

**d.Store(i,j,x);**

**}**

**cout<<"THE DIAGONAL MATRIX IS ::"<<endl;**

**for(int i=1;i<=size;i++)**

**{**

**for(int j=1;j<=size;j++)**

**{**

**cout<<d.Retrieve(i,j)<<"\t";**

**}**

**cout<<"\n";**

**}**

**break;**

**case 2: cout<<"\n ENTER THE ELEMENTS ::"<<endl;**

**for(int i=1;i<=size;i++)**

**for(int j=1;j<=size;j++)**

**{**

**cout<<"\n";**

**cin>>x;**

**l.Store(i,j,x);**

**}**

**cout<<"THE LOWER-TRIANGULAR MATRIX IS ::"<<endl;**

**for(int i=1;i<=size;i++)**

**{**

**for(int j=1;j<=size;j++)**

**{**

**cout<<l.Retrieve(i,j)<<"\t";**

**}**

**cout<<"\n";**

**}**

**break;**

**case 3: cout<<"\n ENTER THE ELEMENTS ::"<<endl;**

**for(int i=1;i<=size;i++)**

**for(int j=1;j<=size;j++)**

**{**

**cout<<"\n";**

**cin>>x;**

**u.Store(i,j,x);**

**}**

**cout<<"THE UPPER-TRIANGULAR MATRIX IS ::"<<endl;**

**for(int i=1;i<=size;i++)**

**{**

**for(int j=1;j<=size;j++)**

**{**

**cout<<u.Retrieve(i,j)<<"\t";**

**}**

**cout<<"\n";**

**}**

**break;**

**case 4: cout<<"\n ENTER THE ELEMENTS ::"<<endl;**

**for(int i=1;i<=size;i++)**

**for(int j=1;j<=size;j++)**

**{**

**cout<<"\n";**

**cin>>x;**

**s.Store(i,j,x);**

**}**

**cout<<"THE SYMMERTIC MATRIX IS ::"<<endl;**

**for(int i=1;i<=size;i++)**

**{**

**for(int j=1;j<=size;j++)**

**{**

**cout<<s.Retrieve(i,j)<<"\t";**

**}**

**cout<<"\n";**

**}**

**break;**

**default: cout<<"INVALID CHOICE....!!!!! PLEASE RETRY"<<endl;**

**}**

**cout<<"DO YOU WANT TO CONTINUE"<<endl;**

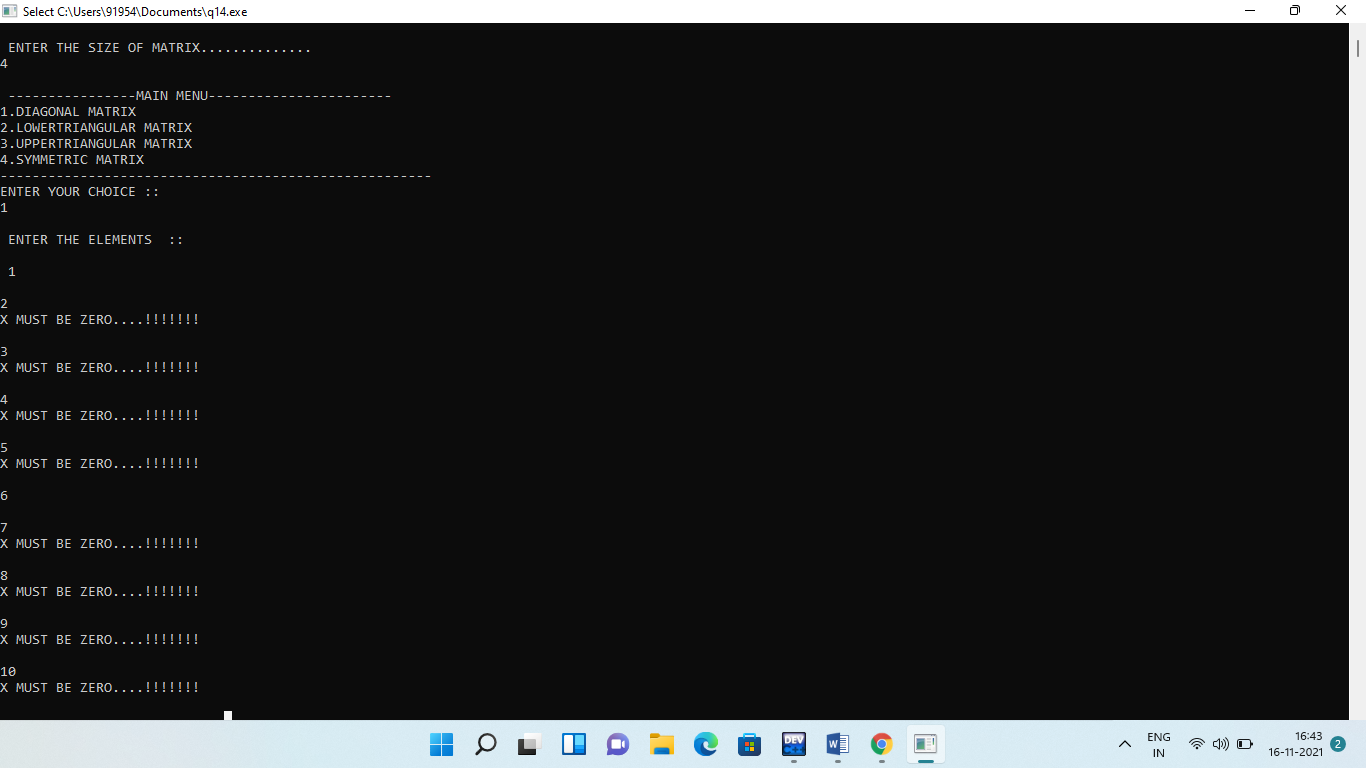
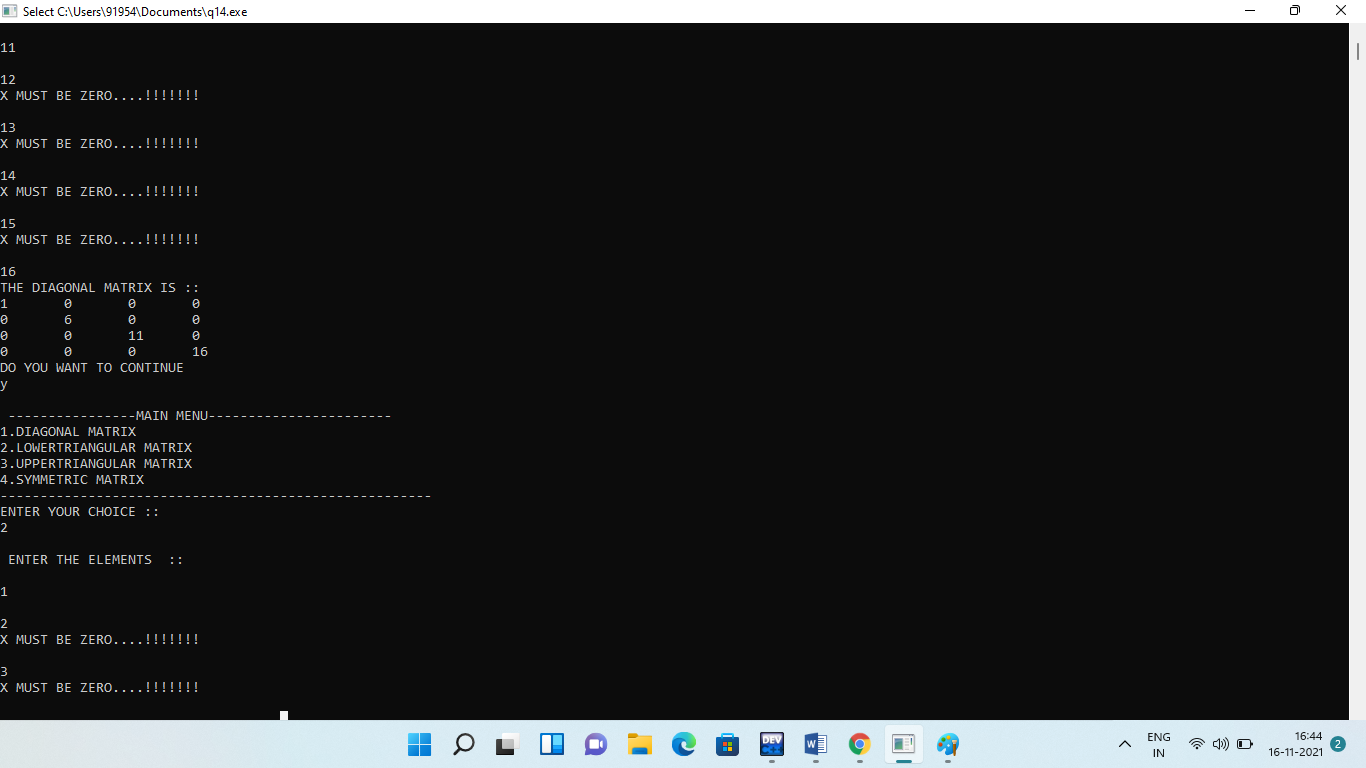
**cin>>ch;**

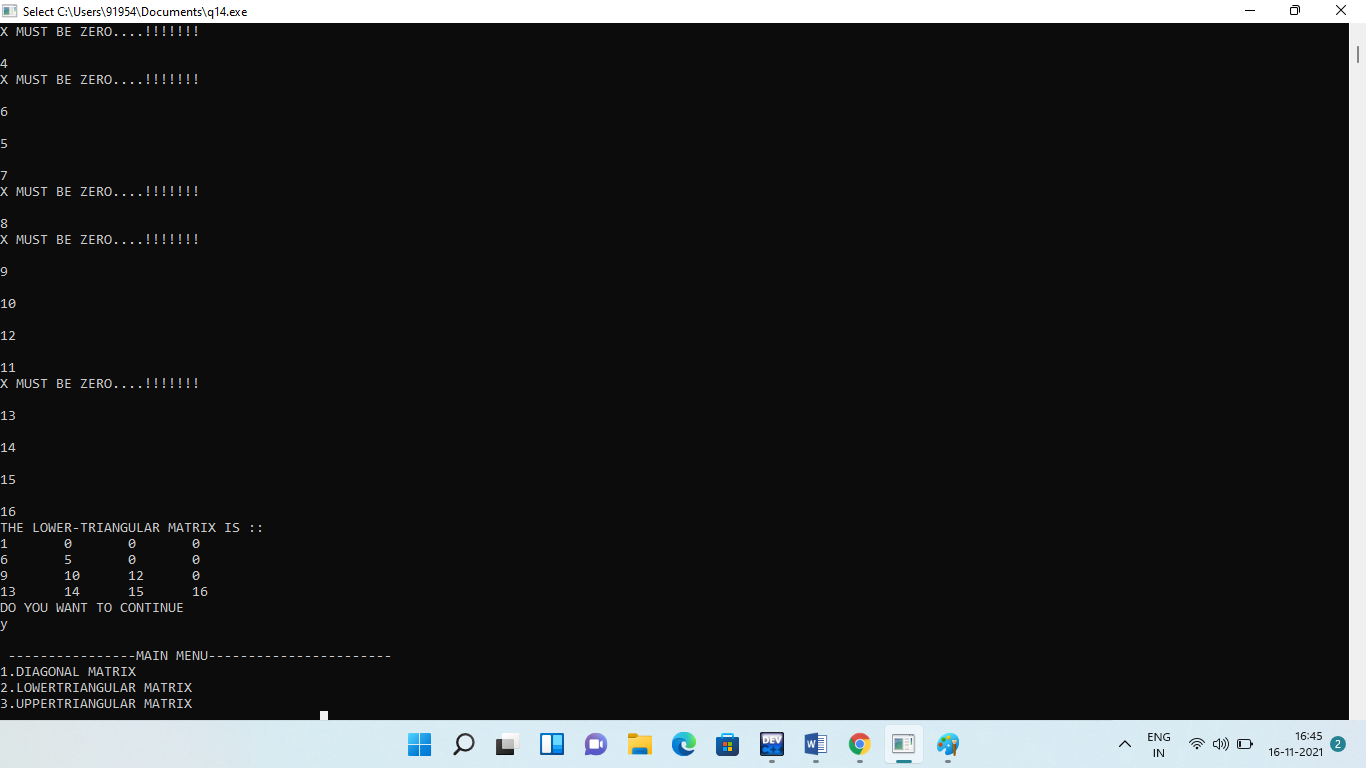
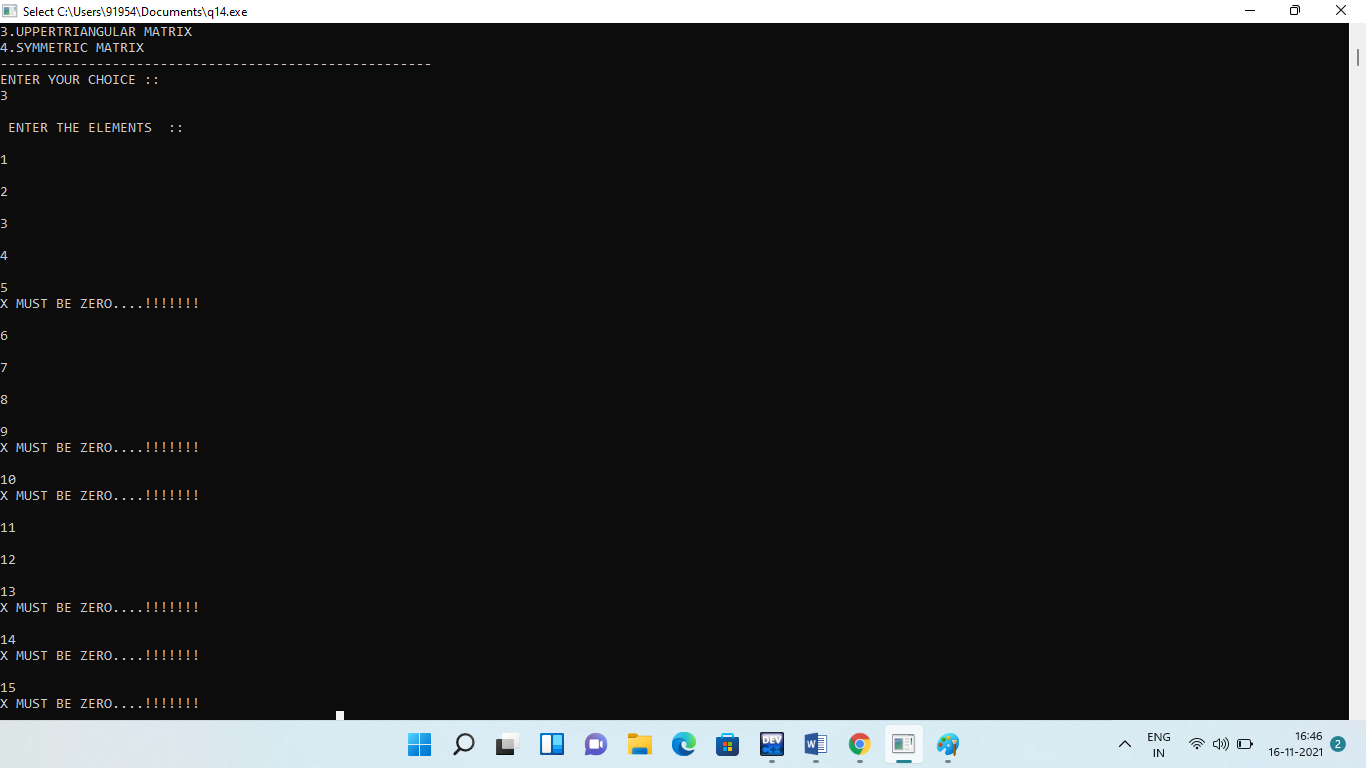
**}while(ch=='y'|| ch=='Y');**

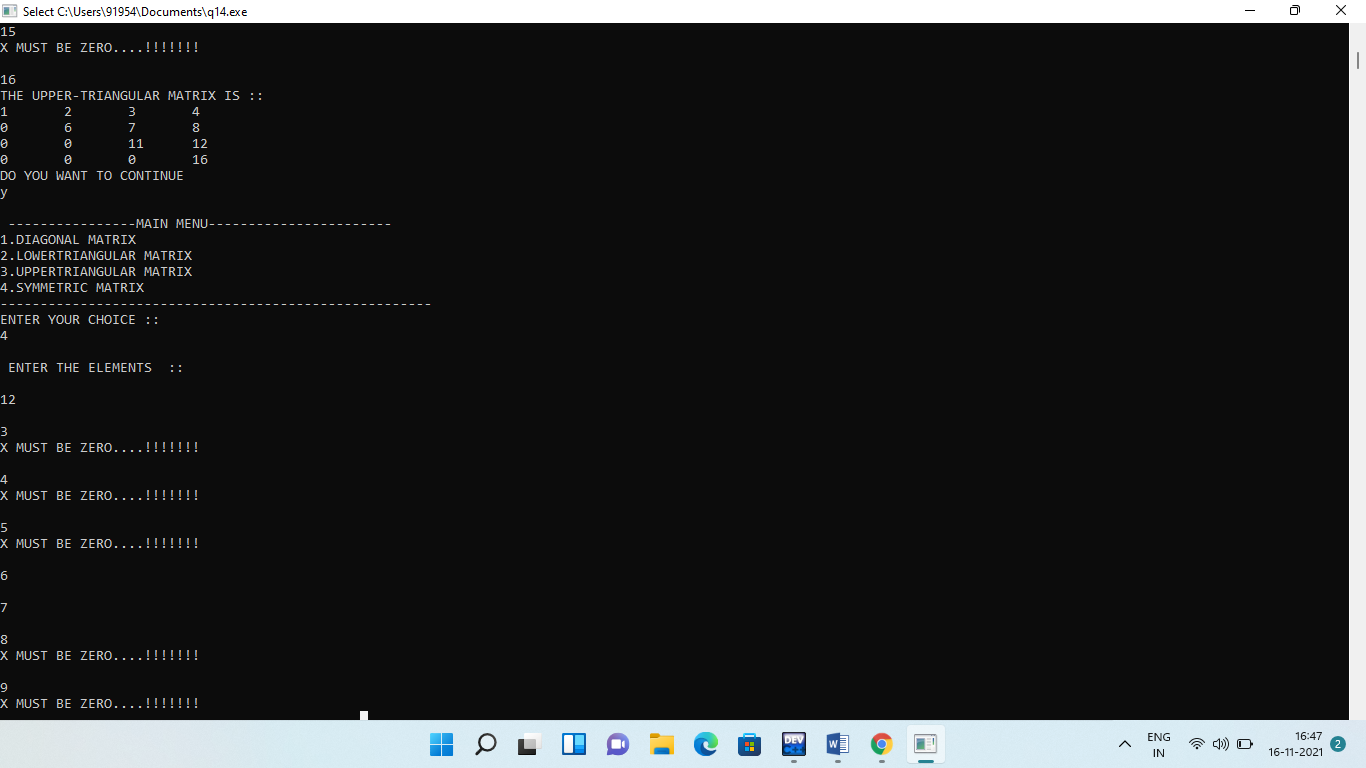
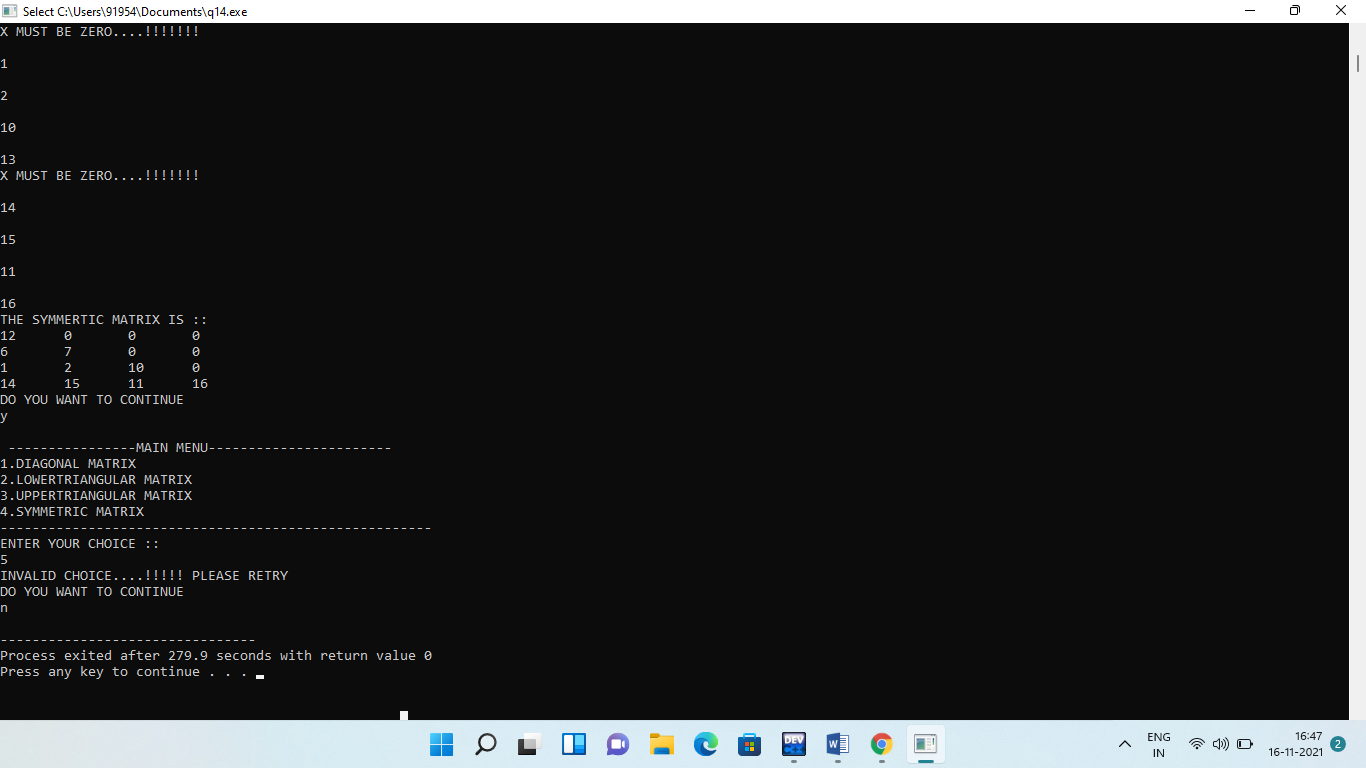
**return 0;**

**}**

**Output:**

****

****

**j**

**Q16.** **Write a program to evaluate a prefix/postfix expression using stacks.**

**Solution:**

**#include<iostream>**

**#include<stack>**

**#include<math.h>**

**using namespace std;**

**int postfixEvaluation(string s)**

**{**

**stack<int> st;**

**for(int i=0;i<s.length();i++)**

**{**

**if(s[i]>='0' && s[i]<='9')**

**{**

**st.push(s[i]-'0');**

**}**

**else**

**{**

**int op2=st.top();**

**st.pop();**

**int op1=st.top();**

**st.pop();**

**switch(s[i])**

**{**

**case'+':st.push(op1+op2);**

**break;**

**case'-':st.push(op1-op2);**

**break;**

**case'\*':st.push(op1\*op2);**

**break;**

**case'/':st.push(op1/op2);**

**break;**

**case'^':st.push(pow(op1,op2));**

**break;**

**default: cout<<"!!please enter a valid operation!!"<<endl;**

**break;**

**}**

**}**

**}**

**return st.top();**

**}**

**int main()**

**{**

**string s=" ";**

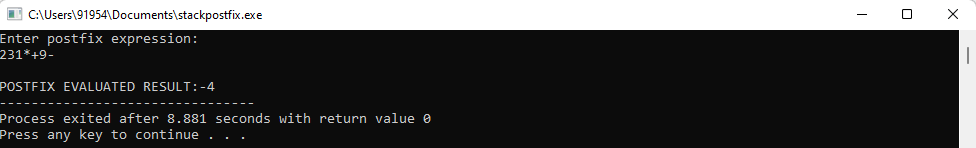
**cout<<"Enter postfix expression:"<<endl;**

**cin>>s;**

**cout<<"\nPOSTFIX EVALUATED RESULT:"<<postfixEvaluation(s);;**

**return 0;**

**}**

**Output:**