**Experiment No: 8 Date:- 01-12-2020**

**AIM: To study the Standard Template Library**

**THEORY:**

The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc. It is a library of container classes, algorithms, and iterators. It is a generalized library and so, its components are parameterized. A working knowledge of template classes is a prerequisite for working with STL.

**STL has four components**

* Algorithms
* Containers
* Functions
* Iterators

**Containers**

* Containers are used to manage collections of objects of a certain kind. There are several different types of containers like deque, list, vector, map etc

**Algorithms**

* Algorithms act on containers. They provide the means by which you will perform initialization, sorting, searching, and transforming of the contents of containers.

**Iterators**

* Iterators are used to step through the elements of collections of objects. These collections may be containers or subsets of containers.

**Functions**

* The STL includes classes that overload the function call operator. Instances of such classes are called function objects or functors. Functors allow the working of the associated function to be customized with the help of parameters to be passed.

**Vector**

Vectors are same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container. Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators. In vectors, data is inserted at the end. Inserting at the end takes differential time, as sometimes there may be a need of extending the array. Removing the last element takes only constant time because no resizing happens. Inserting and erasing at the beginning or in the middle is linear in time.

. **List**

List is a sequence container which takes constant time in inserting and removing elements. List in STL is implemented as Doubly Link List.

The elements from List cannot be directly accessed. For example, to access element of a particular position, you have to iterate from a known position to that particular position.

**Dequeue**

Double ended queues are sequence containers with the feature of expansion and contraction on both the ends.

They are similar to vectors but are more efficient in case of insertion and deletion of elements. Unlike vectors, contiguous storage allocation may not be guaranteed.

Double Ended Queues are basically an implementation of the data structure double ended queue. A queue data structure allows insertion only at the end and deletion from the front. This is like a queue in real life, wherein people are removed from the front and added at the back. Double ended queues are a special case of queues where insertion and deletion operations are possible at both the ends.

**Stacks:**

Stack is a container which follows the **LIFO (Last In First Out)** order and the elements are inserted and deleted from one end of the container. The element which is inserted last will be extracted first.

1. **Write a C++ program to implement standard library vector sequence container.**

#include<iostream>

#include<vector>

using namespace std;

int main(){

vector<int>v;

while(1){

cout<<"\n\n1.Create\n2.Display\n3.Size\n4.Add at end\n5.Delete last\n6.Delete range of elements\n7.Exit\n";

int ch;

cin>>ch;

switch(ch){

case 1:{

int n,x;

cout<<"\nEnter number of elements: ";

cin>>n;

cout<<"\nEnter elements to be inserted:\n";

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

cin>>x;

v.push\_back(x);

}

break;

}

case 2:

for(int i=0;i<v.size();i++){

cout<<" "<<v[i];

}

break;

case 3:

cout<<"\nSize of vector: "<<v.size();

break;

case 4:{

int e;

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

cin>>e;

v.push\_back(e);

break;

}

case 5:

cout<<"\nElement deleted";

v.pop\_back();

break;

case 6: {

//deleting elements (m,n)<->(n-m)

int p1,p2;

cout<<"\nEnter range of position to be deleted:\n";

cin>>p1>>p2;

v.erase(v.begin()+p1-1,v.begin()+p2); //v.begin()=1

break;

}

case 7:

exit(1);

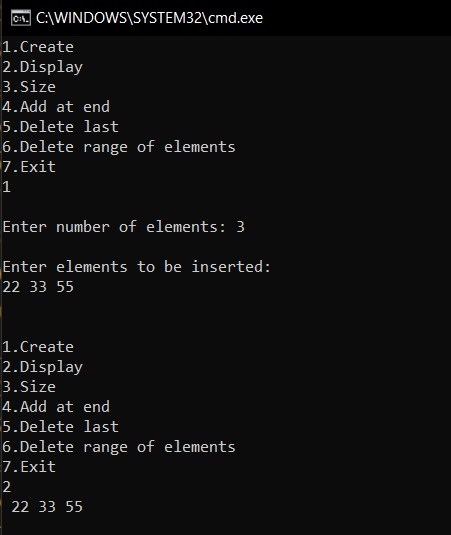
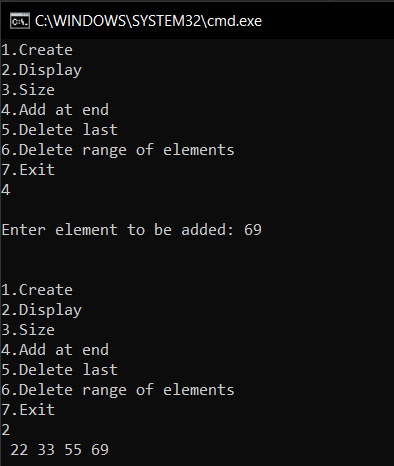
default:

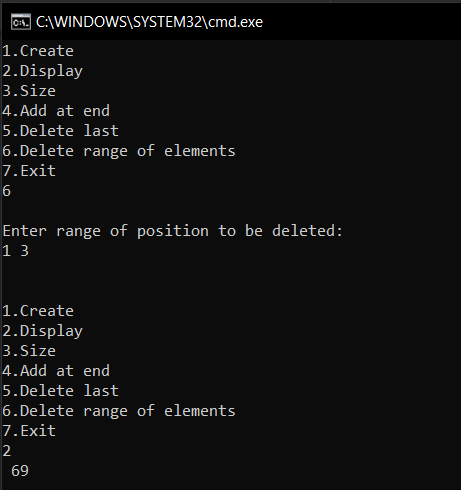
cout<<"\nInvalid choice..... try again";

}

}

}

****

**B] Write a C++ program to implement standard library list sequence container**

//STL list header file

#include <iostream>

#include <list> //list header file

#include<cstdlib>

using namespace std;

void display(list <int> & lst)

{

list <int> :: iterator p;

for(p=lst.begin(); p!=lst.end(); ++p)

cout<< \*p <<" "; //\*p is value to print

cout<<"\n";

}

int main()

{

list <int> list1; //empty list

list <int> list2; //empty list

int n;

cout<<"\nEnter 3 elements for list1:\n";

for(int i=0;i<3;i++)

{

cin>>n;

list1.push\_back(n);

}

cout<<"\nList 1 contents : ";

display(list1);

cout<<"\nEnter 3 elements for list2:\n";

for(int i=0;i<3;i++)

{

cin>>n;

list2.push\_back(n);

}

cout<<"\nList 2 contents : ";

display(list2);

list <int> lista, listb;

lista = list1;

listb = list2;

//Merging 2 list

list1.merge(list2); //merge function

cout<<"\nMerged unsorted List 1 contents with list2 : ";

display(list1);

//sorting lists

lista.sort();

listb.sort();

lista.merge(listb);

cout<<"\nMerged sorted list: ";

display(lista);

//Reversing list

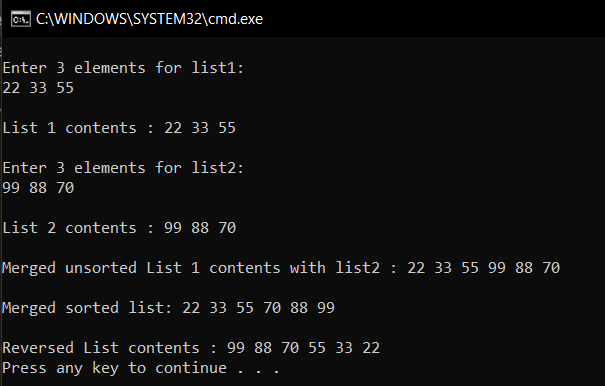
lista.reverse();

cout<<"\nReversed List contents : ";

display(lista);

return 0;

}

****

**C] Write a C++ program to implement standard library deque sequence container**

//STL Deque Program

#include <iostream>

#include <deque>

using namespace std;

void showdq(deque <int> g)

{

deque <int> :: iterator it;

for (it = g.begin(); it != g.end(); ++it)

cout << '\t' << \*it;

cout << '\n';

}

int main()

{

deque <int> deck;

int x;

cout<<"\nDEQUE";

cout<<"\nEnter 5 elements:\n";

for(int i=0;i<5;i++){

cin>>x;

deck.push\_back(x);

}

cout << "The deque deck is : ";

showdq(deck);

cout << "\ndeck.size() : " << deck.size();

cout << "\ndeck.max\_size() : " << deck.max\_size()

cout << "\ndeck.at(2) : " << deck.at(2);

cout << "\ndeck.front() : " << deck.front();

cout << "\ndeck.back() : " << deck.back()

cout << "\ndeck.pop\_front() : ";

deck.pop\_front();

showdq(deck);

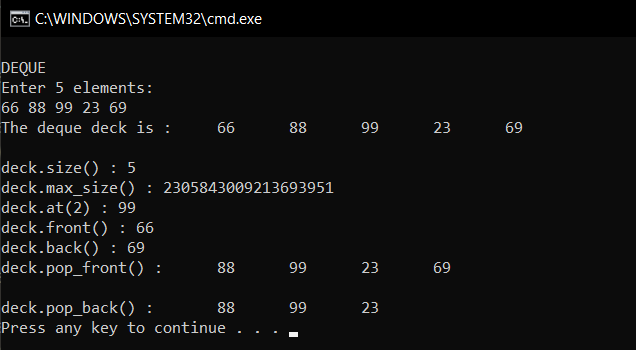
cout << "\ndeck.pop\_back() : ";

deck.pop\_back();

showdq(deck);

return 0;

}

****

**D] Write a C++ program to implement standard library stack adapter class**

#include <iostream>

#include <stack>

using namespace std;

int main()

{

stack <int> st;

while(1){

int ch;

cout<<"\n\n1.Push\n2.Pop\n3.Element on top\n4.Display all\n5.Exit\n";

cin>>ch;

switch(ch){

case 1:{

cout<<"\nEnter element: ";

int e;

cin>>e;

st.push(e);

break;

}

case 2:{

if(st.empty()) {

printf("\nEmpty");

}

else{

cout<<endl<<st.top()<<" is popped"<<endl;

st.pop();

}

break;

}

case 3:{

if(st.empty()){

printf("\nEmpty");

}else

cout<<"Element on top is : "<<st.top()<<endl;

break;

}

case 4:{

if(st.empty()){

printf("\nEmpty");

}else{

cout<<"\nStack is: ";

stack<int> s=st;

while(!s.empty()){

cout<<" "<<s.top();

s.pop(); }

break;

}

case 5: exit(1);

default:

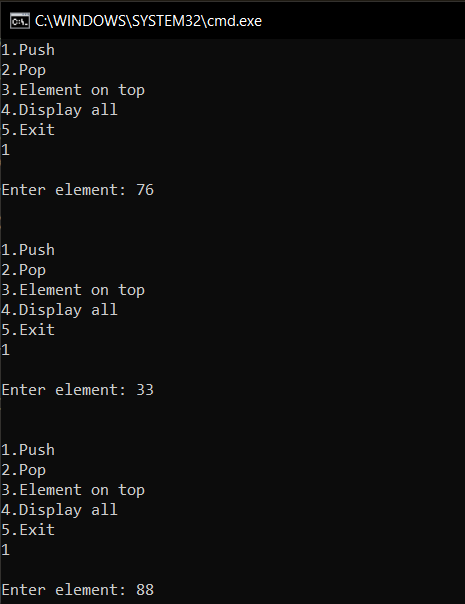
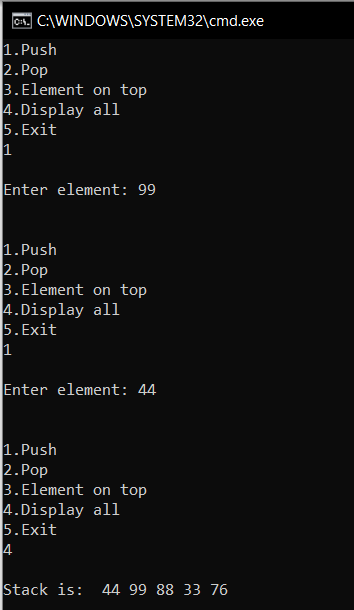
cout<<"\nWrong choice>>>>Try again";

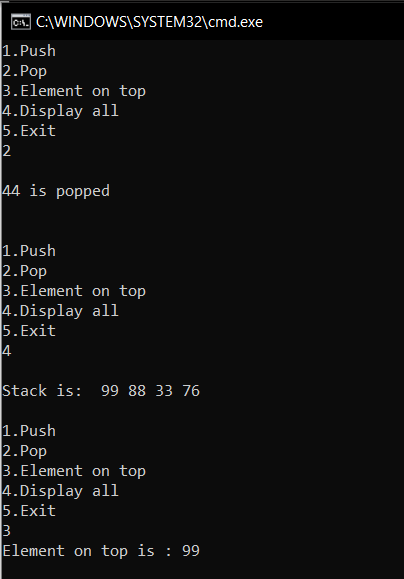
}

}

}

}

** **

****

**CONCLUSION:**

• STL consists of a collection of template classes, it is a generalized library that is independent of data types. Container adapters, on the other hand, are interfaces created by limiting functionality in a pre-existing container and providing a different set of functionalities

• Container adopters are sequential containers; however, they are implemented by providing a different interface. Thus, containers like a queue, deque, stack, and priority-queue are all classified as container adopters.

• STL Algorithms provide approximately 60 algorithm functions to perform the complex operations.

• Standard algorithms allow us to work with two different types of the container at the same time.

• Algorithms are not the member functions of a container, but they are the standalone template functions. Algorithms save a lot of time and effort.

• If we want to access the STL algorithms, we must include the header file in our program.