Lab 12

Standard Template Library

Objective

- To learn about STL components(Algorithms,containers and iterators)

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.

STL Components

- Algorithms
- Containers
- Iterators

Sr.No	Component & Description
1	**Containers** Containers are used to manage collections of objects of a certain kind. There are several different types of containers like arrays,deque, list, vector etc.
2	**Algorithms** Algorithms act on containers.They provide the means by which you will perform initialization, sorting, searching, and transforming of the contents of containers.
3	**Iterators** Iterators are used to step through the elements of collections of objects. These collections may be containers or subsets of containers.

An Introduction to `std::array`

`std::array` is a container that encapsulates fixed size arrays. It provides fixed array functionality that won't decay when passed into a function. `std::array` is defined in the array header, inside the std

namespace.The length of an std::array must be known at the time of compilation.Some of the built-in functions of arrays are as follows.

Functions	**Functionalities**
`back()`	Returns the last element of the array
`front()`	Returns the first element of the array
`size()`	Returns number of elements of the array
`begin()`	Returns an iterator to the beginning
`end()`	Returns an iterator to the end
`empty()`	Is used to check whether elements exists in array or not

Syntax of declaring an array is as follows

Syntax

```
std::array<int, 10> my_array;
```

Example 12.1

```
//Implementing std::array
#include<array>
#include<iostream>
int main()
{

    //To declare an integer array with length 3
    std::array<int, 3> a1{1, 2, 3};

    //To print all the array elements using for loop
    for(int i=0; i<3 ; i++)
    {

        std::cout << a1.at(i) << std::endl ;

    }
```

```
    //To display size of array
    std::cout << "Size of array :" << a1.size() << "\n";

    //To access value at first index of array
    std::cout << "Value at first index :"<< a1.at(1) << "\n";

    //To access value at 0th index of array
    std::cout << "First element of array"<< a1.front() << "\n";

    //To access value at last index of array
    std::cout << "Last element of array:" << a1.back() << "\n";

    return 0;
}
```

Output

```
1
2
3
Size of array :3
Value at first index :2
First element of array1
Last element of array:3
```

Example 12.2

```
//Implementing std::array
#include <iostream>
#include <array>
int main()
{
    std::array<int, 4> array1 = {10, 20, 30, 40};
    std::array<int, 4> array2 = {50, 60, 70, 80};
    //This function will be used to swap values of two vectors
    array1.swap(array2);

    //printing the values of the elements of array1
    std::cout << "Elements of array1" << std::endl;
    for (int j = 0; j < array1.size(); j++ )
    {
       std::cout << array1.at(j) << std::endl;
    }

    //printing the values of the elements of a2
    std::cout << "Elements of array2" << std::endl;
    for (int j = 0; j < array2.size(); j++ )
    {
```

```
        std::cout << array2.at(j) << std::endl;
    }
    return 0;
}
```

Output

```
Elements of array1
50
60
70
80
Elements of array2
10
20
30
40
```

Vectors in C++

Vectors are the 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. Certain functions associated with the vector are:

Functions	**Functionalities**
`size()`	Returns the number of elements in the vector
`max_size()`	Returns the maximum number of elements that the vector can hold
`capacity()`	Returns the size of the storage space currently allocated to the vector expressed as number of elements.
`resize()`	Resizes the container so that it contains ‘g’ elements.
`empty()`	Returns whether the container is empty

`shrink_to_fit()`	Reduces the capacity of the container to fit its size and destroys all elements beyond the capacity.

Example 12.3

```
//Implementing vector using built-in functions
#include <iostream>
#include <vector>
int main()
{
    std::vector<int> vec;

    for (int i = 1; i <= 5; i++)
    {
     //This function will be used to insert a value at the end of vector
        vec.push_back(i);
    }

    std::cout << "Size : " << vec.size();
    std::cout << "\nCapacity : " << vec.capacity();
    std::cout << "\nMax_Size : " << vec.max_size();

    // resizes the vector size to 4
    vec.resize(4);

    // prints the vector size after resize()
    std::cout << "\nSize : " << vec.size();

    // checks if the vector is empty or not
    if (vec.empty() == false)
        std::cout << "\nVector is not empty";
    else
        std::cout << "\nVector is empty";
    return 0;
}
```

Output

```
Size : 5
Capacity : 8
Max_Size : 4611686018427387903
Size : 4
Vector is not empty
```

Example 12.4

```
//Implementing vectors using swap functions
#include <iostream>
#include <vector>
int main()
{
   //first vector
   std::vector<int> v1;
   //second vector
   std::vector<int> v2 = {1, 2, 3, 4, 5};

   //This function will be used to swap values of two vectors
   v1.swap(v2);

   //printing vector in screen
   std::cout << "Vector v1 contains" << std::endl;
   for (int i = 0; i < v1.size(); ++i)
   {
      std::cout << v1[i] << std::endl;
   }

   return 0;
}
```

Output

```
Vector v1 contains
1
2
3
4
5
```

Iterators

Iterators are used to hold the address of STL containers. We can access elements of container by using iterator to iterate through elements of a container.Elements can be dynamically added or deleted from container easily using iterators.One advantage of using iterator is code reusability that will be obtain by

declaring one object of object of iterator class and using it throughout the program.Some major functions of iterator are following

functions of iterator are following

Functions	**Functionalities**
`begin()`	Returns a pointer to the first element of the container
`end()`	Returns a pointer to the last element of the container
`advance()`	Used to increment the position of the pointer to point to next location
`inserter()`	Used to insert elements at particular position of a container
`next()`	Returns a new iterator that points to the next location of the specified argument
`previous()`	Returns a new iterator that points to the previous location of the specified argument

Example 12.5

```
//Implementing Iterators using built-in functions
#include<iostream>
#include<vector>
int main()
{
	//creating a vector v
	std::vector<int>  v;

	//assigning values to vector
	for (int j = 0; j <10; j++)
	{
	//This function will be used to insert a value at the end of vector
	 v.push_back(j);
	}
```

```
        //defining an iterator
        std::vector<int>::iterator i = v.begin();

        ///accessing values of vector using iterator i
        while( i != v.end())
        {
           std::cout << "value of v = " <<*i<< std::endl;
           i++;
        }

}
```

Output

```
value of v = 0
value of v = 1
value of v = 2
value of v = 3
value of v = 4
value of v = 5
value of v = 6
value of v = 7
value of v = 8
value of v = 9
```

Example 12.6

```
// C++ code to demonstrate the working of
// inserter()
#include<iostream>
#include<iterator> // for iterators
#include<vector> // for vectors
using namespace std;
int main()
{
     std::vector<int>V1 = { 1, 2, 3, 4, 5 };
     std::vector<int>V2 = {10, 20, 30};

    // Declaring iterator to a vector
     std::vector<int>::iterator ptr =V1.begin();
```

```
    // Using advance to set position
    advance(ptr, 3);

    // copying 1 vector elements in other using inserter()
    // inserts V2 after 3rd position in V1
     copy(V2.begin(), V2.end(), inserter(V1,ptr));

    // Displaying new vector elements
    cout << "The new vector after inserting elements is : ";
    for (int &x :V1)
        cout << x << " ";

    return 0;
}
```

Output

```
The new vector after inserting elements is : 1 2 3 10 20 30 4 5
```

DHA SUFFA UNIVERSITY
Department of Computer Science
Object Oriented Programming
CS-1002L
Spring 2020

Lab Tasks

1. Create an array of type `int` named roll_numbers using `std::array`, pass this array to a function that should sort these roll numbers. Assume any logic for sorting.
2. Implement your own functions to work as `front()`,`back()`,`fill()` for an array.
3. Create a vector, define iterators to point first and last element of the vector. Calculate distance between or number of elements between these iterators.
4. Create two vectors, pass these vectors to a function that should merge these vectors into one.
5. Create two vectors, pass these vectors to a function that should swap these vectors.

Home Assignment

Write a program, making use of arrays,vectors, and iterators demonstrating at least three functions of each.

Submission Instructions

1. Number your solution folders as question number e.g. Q1, Q2, etc. (Q is in upper case)
2. Every folder should contain three files (one header, one implementation and one driver)
3. Create a new folder named cs152abc where abc is your 3 digit roll #. e.g. cs152111.
4. Copy all the project folders into this folder.
5. Now make sure a zip file named cs152abc.zip is created e.g. cs152111.zip
6. Upload the assignment solution on LMS under the assignment named Lab 06 Assignment – XX, where XX is your section name