National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Computing

CS212: Object Oriented Programming

Class: BSCS-5

Lab 12: Exploring Standard Template Library (STL) in C++

Date: 23-5-2016

Time: 09:00 am – 12:00 pm / 2pm to 5pm

Instructor: Ms. Hirra Anwar

Lab 12: Standard Template Library (STL) in OOP C++

Introduction

A container is a holder object that stores a collection of other objects (its elements). They are implemented as class templates, which allow a great flexibility in the types supported as elements. The container manages the storage space for its elements and provides member functions to access them, either directly or through **iterators** (reference objects with similar properties to pointers).Containers replicate structures very commonly used in programming: dynamic arrays (vector), queues (queue), stacks (stack), heaps (priority_queue), linked lists (list), trees (set), associative arrays (map).

Objectives

- STL in OOP C++
- Introduction to Vectors C++
- Vectors with Objects
- Deque

Tools/Software Requirement

You will need Visual Studio.

Description

We will discuss about all the three C++ STL components in next chapter while discussing C++ Standard Library. For now, keep in mind that all the three components have a rich set of pre-defined functions which help us in doing complicated tasks in very easy fashion. Follow all the content given below to understand the concept of the given topics. Run all the given examples to get a better understanding.
Your focus on this lab should be examples in the manual and other examples related to containers available online. Get an understanding of their usage through various examples.

Lab Task 1

Let us take the following program demonstrates the vector container (a C++ Standard Template) which is similar to an array with an exception that it automatically handles its own storage requirements in case it grows:

Practice 1

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
  // create a vector to store int
  vector<int> vec;
  int i;

  // display the original size of vec
  cout << "vector size = " << vec.size() << endl;

  // push 5 values into the vector
  for(i = 0; i < 5; i++){
    vec.push_back(i);
  }

  // display extended size of vec
  cout << "extended vector size = " << vec.size() << endl;

  // access 5 values from the vector
  for(i = 0; i < 5; i++){
    cout << "value of vec [" << i << "] = " << vec[i] << endl;
  }

  // use iterator to access the values
  vector<int>::iterator v = vec.begin();
  while( v != vec.end()) {
    cout << "value of v = " << *v << endl;
    v++;
  }

  return 0;
}
```

Here are following points to be noted related to various functions we used in the above example:

- The push_back( ) member function inserts value at the end of the vector, expanding its size as needed.
- The size( ) function displays the size of the vector.
- The function begin( ) returns an iterator to the start of the vector.

- The function end( ) returns an iterator to the end of the vector.

Practice 2

```
#include <iostream>

#include <forward_list>


using namespace std;


void print(forward_list<int> list)

{

for(forward_list<int>::iterator itr = list.begin(); itr != list.end(); itr++)

{

cout << *itr << endl;

}

}


int main()

{

forward_list<int> list;

list.push_front(2);

list.push_front(1);


print(list);


return 0;
```

```
}
```

Vector with objects

Practice 3

```
#include <iostream>
#include <vector>
#include <string>
#include <cstdlib>
#include <sstream>
   using namespace std;
   class alfa{
   private:
   int x;
   public:
   alfa(){
   cout<<"Constructing"<<endl;
   x=rand()%10;
   }
   ~alfa(){
   cout<<"Destructing"<<endl;
   }
   void get_x(){
   cout<<"This is get func: "<<x<<endl;
   }
   };

   int main(){
   alfa* alfadi;
   int no;
   cout<<"Hello World"<<endl;
   cin>>no;
      // vector for storing objects ...
   vector<alfa> alfadia;
   for(int i=0;i<no;i++)
   {
   // 'obj' below, is constructed via default ctor
   alfa obj;
   alfadia.push_back(obj);

   }
```

```
}
```

Lab Tasks

1. **Write a program that pushes and pops elements from a stack until it is empty using STL.**

2. **Create a class student.cpp as shown below.**

Student.cpp

```
#include <iostream>
#include <string>
using namespace std;

class student
{
public:
student();
student(string , char )
{}

//access funstions
string getnewname()
{}
char      getgrade()
{}
//mutator funstions
void setnewnaem(string);
{}
void setgrade(char);
{}
//destructor
~student();

private:
          char grade;
          string newname;
}
```

2. Create a **vector** of class students.

3. Create a class fill vector that will fill student information in the student class method will look like the function definition below.

```
void fillvector(vecor<student>&)
```

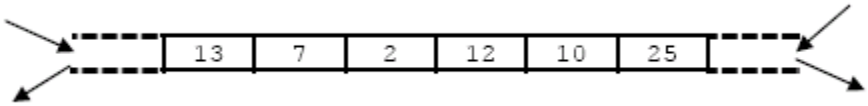
4. Ask the user that how many student are in your class. Based on the user's requirement fill the information in the student class using vectors.

5. Make another function **printvector()** and print all the information of the student placed in the vector class.

```
void printvector(const vecor<student>&)
```

The deque

- The term deque (pronounced "**deck**") is an abbreviation for '**double-ended queue**'. It is a dynamic array that is implemented so that it can grow in both directions.
- So, inserting elements at the end and at the beginning is fast. However, inserting elements in the middle takes time because elements must be moved. Deque structure can be depicted as follow:



- Deque reallocation occurs when a member function must insert or erase elements of the sequence:
    1. If an element is inserted into an empty sequence, or if an element is erased to leave an empty sequence, then iterators earlier returned by begin() and end() become invalid.
    2. If an element is inserted at the first position of the deque, then all iterators, but no references, that designate existing elements become invalid.
    3. If an element is inserted at the end of the deque, then end() and all iterators, but no references, that designate existing elements become invalid.
    4. If an element is erased at the front of the deque, only that iterator and references to the erased element become invalid.
    5. If the last element is erased from the end of the deque, only that iterator to the final element and references to the erased element become invalid.
- Otherwise, inserting or erasing an element invalidates all iterators and references.
- The following general deque example declares a deque for floating-point values, inserts elements from 1.2 to 12 at the front of the container, and prints all elements of the deque:

Lab Practice 1

```
#include <iostream>
#include <deque>
using namespace std;

int main()
{
   // deque container for floating-point elements declaration
   deque<float> elem, elem1;
   // insert the elements each at the front
   cout<<"push_front()\n";
   for(int i=1; i<=10; ++i)
   // insert at the front
   elem.push_front(i*(1.2));
   // print all elements separated by a space
   for(int i=0; i<elem.size(); ++i)
      cout<<elem[i]<<' ';
   cout<<endl;
   // insert the elements each at the back
   cout<<"\npush_back()\n";
   // insert at the back
   for(int i=1; i<=10; ++i)
```

```
    elem1.push_back(i*(1.2));

  // print all elements separated by a space

  for(int i=0; i<elem1.size(); ++i)

  cout<<elem1[i]<<' ';

  cout<<endl;
      cin.get();
  return 0;

}
```

- deque constructor, constructs a deque of a specific size or with elements of a specific value or with a specific allocator or as a copy of all or part of some other deque.
- All constructors store an allocator object and initialize the deque.
- None of the constructors perform any interim reallocations.

Lab Practice 2

```
// deque, constructors

#include <deque>

#include <iostream>

using namespace std;



int main()

{

  deque <int>::iterator deq0Iter, deq1Iter, deq2Iter, deq3Iter, deq4Iter, deq5Iter,
deq6Iter;

  // create an empty deque deq0

  deque <int> deq0;
```

```
// create a deque deq1 with 10 elements of default value 0

deque <int> deq1(10);

// create a deque deq2 with 7 elements of value 10

deque <int> deq2(7, 10);

// create a deque deq3 with 4 elements of value 2 and with the allocator of deque deq2

deque <int> deq3(4, 2, deq2.get_allocator());

// create a copy, deque deq4, of deque deq2

deque <int> deq4(deq2);

// deque deq5 a copy of the deq4[_First, _Last) range

deq4Iter = deq4.begin();

deq4Iter++;

deq4Iter++;

deq4Iter++;

deque <int> deq5(deq4.begin(), deq4Iter);

// create a deque deq6 by copying the range deq4[_First, _Last) and the allocator of
deque deq2

deq4Iter = deq4.begin();

deq4Iter++;

deq4Iter++;

deq4Iter++;

deque <int> deq6(deq4.begin(), deq4Iter, deq2.get_allocator());

// ---------------------------------
```

```
cout<<"Operation: deque <int> deq0\n";

cout<<"deq0 data: ";

for(deq0Iter = deq0.begin(); deq0Iter != deq0.end(); deq0Iter++)

   cout<<*deq0Iter<<" ";

cout<<endl;

cout<<"\nOperation: deque <int> deq1(10)\n";

cout<<"deq1 data: ";

for(deq1Iter = deq1.begin(); deq1Iter != deq1.end(); deq1Iter++)

   cout<<*deq1Iter<<" ";

cout<<endl;

cout<<"\nOperation: deque <int> deq2(7, 3)\n";

cout<<"deq2 data: ";

for(deq2Iter = deq2.begin(); deq2Iter != deq2.end(); deq2Iter++)

cout<<*deq2Iter<<" ";

cout<<endl;

cout<<"\nOperation: deque <int> deq3(4, 2, deq2.get_allocator())\n";

cout<<"deq3 data: ";

for(deq3Iter = deq3.begin(); deq3Iter != deq3.end(); deq3Iter++)

   cout<<*deq3Iter<<" ";

cout<<endl;

cout<<"\nOperation: deque <int> deq4(deq2);\n";

cout<<"deq4 data: ";
```

```
   for(deq4Iter = deq4.begin(); deq4Iter != deq4.end(); deq4Iter++)

      cout<<*deq4Iter<<" ";

   cout<<endl;

   cout<<"\nOperation1: deq4Iter++...\n";

   cout<<"Operation2: deque <int> deq5(deq4.begin(), deq4Iter)\n";

   cout<<"deq5 data: ";

   for(deq5Iter = deq5.begin(); deq5Iter != deq5.end(); deq5Iter++)

      cout << *deq5Iter<<" ";

   cout << endl;

   cout<<"\nOperation1: deq4Iter = deq4.begin() and deq4Iter++...\n";

   cout<<"Operation2: deque <int> deq6(deq4.begin(), \n"

      "     deq4Iter, deq2.get_allocator())\n";

   cout<<"deq6 data: ";

   for(deq6Iter = deq6.begin(); deq6Iter != deq6.end(); deq6Iter++)

      cout<<*deq6Iter<<" ";

   cout<<endl;
         cin.get();
   return 0;

}
```

3. Write a program using STL that declares a deque for floating-point values, inserts elements from 1.2 to 12 at the front of the container, and prints all elements of the deque.

Deliverables

Source code for Task 2 and Task 3