UEE 1303(1067): Object-Oriented Programming

Lab #13: Standard Template Library

In this laboratory session you will learn:

- how to use STL vector and map containers

Lab 13-1: Vector

✓ A container is an object whose main purpose is to hold other objects. A `vector` contains an array of `n` objects indexed from 0 to `n`-1.

```
// lab13-1-1.cpp
#include <iostream>
#include <vector>
using namespace std;
int main()
{
   int n = 10;
   // allocate a vector with 10 elements
   vector<int> vec1(n);
   // use subscripting to access elements
   for (int i = 0; i < vec1.size(); i++)
      vec1[i] = i * i;
   for (int i = 0; i < vec1.size(); i++)
      cout << vec1[i] << " ";
   cout << endl;
   vector<int> vec2; // allocate an empty vector
   // use push_back() to add elements
   for (int i = 0; i < n; i++)
      vec2.push_back(i * 2);
   vector<int>::const_iterator iter;
   for (iter = vec2.begin(); iter != vec2.end(); iter++)
      cout << iter << " "; // use iterator to traverse 
container
   cout << endl;
   return 0;
}
```

- Please fix the compiler error here.
- Note that, `vec1[i]` and `vec1.at(i)` are similar to access elements in vector. However, `vec1.at(i)` provides range checking but `vec1[i]` does not.

✓ A `vector` of class objects can be created if the class has a default constructor.

```
// lab13-1-2.cpp
#include <iostream>
#include <vector>
using namespace std;
class Point2D
{
private:
    int x;
    int y;
public:
    Point2D(): x(0), y(0){}
    Point2D(int a, int b): x(a), y(b){}
    friend ostream &operator << (ostream &out, const
Point2D &p)
    {
        out << "(" << p.x << "," << p.y << ")";
    }
};

int main()
{
    int n = 10;
    vector<Point2D> vec(n); // call Point2D()
    for (int i = 0; i < vec.size(); i++)
        // call Point2D(int a, int b)
        vec[i] = Point2D(i*2, i*3);
    for (int i = 0; i < vec.size(); i++)
        cout << vec[i] << " ";
    cout << endl;
    return 0;
}
```

✓ Here demonstrates more operations supported by `vector`.

```
// lab13-1-3.cpp
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    int n = 5;
    vector<int> vec(n,-1); // vec = {-1,-1,-1,-1,-1}
    vector<int> u(3);
    for (int i = 0; i < 3; i++)
        u[i] = i; // u = {0,1,2}
    vec.insert(vec.begin()+2, u.begin(), u.end());
    // vec = {-1,-1,0,1,2,-1,-1,-1}
    vec.insert(vec.begin()+1,10);
    // vec = {-1,10,-1,0,1,2,-1,-1,-1}
    vec.pop_back(); // vec = {-1,10,-1,0,1,2,-1,-1}
    vec.erase(vec.begin()+3);
    // vec = {-1,10,-1,1,2,-1,-1}
    vec.clear(); // vec = {}
    for (int i = 0; i < vec.size(); i++)
        cout << vec[i] << " ";
    cout << endl;
    return 0;
}
```

- The functions `begin()` and `end()` return iterators to the first element and one-past-the-last element, respectively. It denotes the interval [begin,end).
- `vec.insert(p,x)` is used to add element `x` at position `p` and `vec.insert(p,first,last)` can insert a sequence `[first,last)` to position `p`.
- `vec.erase(p)` remove the element at position `p`. `vec.clear()` remove all elements.

✓ In <algorithm>, `sort()` is defined to sort the elements in increasing order. `reverse()` can reverse the elements in container and `find()` is used to find the specific element.

```
// lab13-1-4.cpp
```

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
int main()
{
   int n = 10;
   vector<int> vec(n);
   for (int i = 0; i < vec.size(); i++)
      vec[i] = rand()%n;
   // assume vec = {1,7,4,0,9,4,8,8,2,4}
   sort(vec.begin(), vec.end());
   // after sort, vec = {0,1,2,4,4,4,7,8,8,9}
   reverse(vec.begin(), vec.end());
   // vec = {9,8,8,7,4,4,4,2,1,0}
   for (int i = 0; i < vec.size(); i++)
      cout << vec[i] << " ";
   cout << endl;
   vector<int>::iterator  iter  =  find(vec.begin(),
vec.end(), 8);
   if (iter != vec.end())
      cout << "8 is in the vector" << endl;
   else
      cout << "8 is not in the vector" << endl;
   return 0;
}
```

- The function `sort()` places elements of the vector in increasing order based on a less-than operation < by default.

Lab 13-2: Map

✓ A map is a container whose elements are pairs of a key and a value. When indexed by the key, a map returns the corresponding value.

```
// lab13-2-1.cpp
#include <iostream>
#include <map>
#include <string>
using namespace std;
```

```
int main()
{
   map<int,string> classroom;
   classroom[9912345] = "Jacky";
   classroom[9923456] = "John";
   classroom[9934567] = "Mary";
   map<int,string>::const_iterator iter
   for (iter = classroom.begin();
        iter != classroom.end(); iter++)
   {
      cout << "ID: " << iter->first << " ";
      cout << "name: " << iter->second << endl;
   }
   return 0;
}
```

✓ Here is another example to use map

```
// lab13-2-2.cpp
#include <iostream>
#include <map>
#include <string>
using namespace std;
int main()
{
   map<string,int> age;
   age["Mary"] = 22;
   age["Jacky"] = 18;
   age["John"] = 20;
   map<string,int>::const_iterator iter;
   for ( iter = age.begin();
         iter != age.end(); iter++) {
   {
      cout << "name: " << iter->first << " ";
      cout << "age: " << iter->second << endl;
   }
   return 0;
}
```

- Note than map stores elements in increasing order based on a less-than operation <

Exercise 13-1 (Vector)

✓ Create a vector of Complex numbers and sort them by using the standard algorithm `sort()` in the order of decreasing absolute values. Note that you should define a Complex class.

✓ The output of the program should like as,

```
Enter n: 5
Original sequence:
(1.6,4.7) (1.6,4.5) (7.5,8.4) (6.4,6.9) (3.8,3.4)
Sorted sequence:
(7.5,8.4) (6.4,6.9) (3.8,3.4) (1.6,4.7) (1.6,4.5)
```

Exercise 13-2 (Set)

✓ Consider a text file of names, with one name per line, that has been compiled from several different source. A sample (input.txt) is shown in the following:

```
Brook Trout
Dinah Soars
Jed Dye
Brooke Trout
Jed Dye
Paige Turner
```

There are duplicate names in the file. We would like to generate an invitation list but do not want to send multiple invitations to the same person. Write a program that eliminates the duplicate names by using the set template class. Read each name from the file, add it to the set, and then output all names in the set to generate the invitation list without duplicates.

✓ Your program executes as follows:

```
> ./ex13-2 input.txt list.txt
```

✓ The context in "list.txt " is

```
Brook Trout
Dinah Soars
Jed Dye
Paige Turner
```