CS 10C Programming Concepts and Methodologies 2

Skip to Main Content

Assignment 1: Templates

No documentation is required in this assignment.

Assignment 1.1 [25 points]

Rewrite **this high scores program** using an STL vector instead of an array. (Note, you will be using an STL vector, not the MyVector class developed in lesson 19.)

Clarifications and Additional Requirements:

- Your program must use the following four function headers exactly as they appear here:

```
void getVectorSize(int& size);
void readData(vector<Highscore>& scores);
void sortData(vector<Highscore>& scores);
vector<Highscore>::iterator findLocationOfLargest(
                                const vector<Highscore>::iterator startingLocation,
                                const vector<Highscore>::iterator endingLocation);
void displayData(const vector<Highscore>& scores);
```

- The size parameter from the given code won't be needed now, since a vector knows its own size.
- Notice that the findLocationOfLargest() function does not need the vector itself as a parameter, since you can access the vector using the provided iterator parameters.
- The name field in the struct must still be a c-string
- The focus of this assignment is to use iterators. You must use iterators wherever possible to access the vector. As a result, you must not use square brackets, the push_back() function, the at() function, etc. Also, the word "index" shouldn't appear in your code anywhere. You won't get full credit if you miss an opportunity to use iterators.
- You should still ask the user to enter the number of scores there will be, and then you should create a vector with the required capacity. You can do this by using the vector class's constructor that creates a vector with the capacity indicated by its parameter. For example, to create a vector of size 100, use this:

  ```
  vector<sometype> myExampleVector(100);
  ```

  It is possible to write our program without bothering to indicate a size if we simply use push_back() to add each high score struct to the vector. We aren't doing it that way, because I want you to practice using iterators.
- You could sort the scores by simply calling the STL sort() algorithm. I would suggest that you try this out because it's something you should know, but for your submitted program you are required to sort the vector as it is done in the given code, except using iterators to access the items in the vector. You won't get credit for the assignment if you use the STL sort() function.
- In your displayData() function you'll need to use const_iterator instead of iterator. See lesson 19.4.
- Here is an example that shows how to use an iterator to access a particular member of a struct:

  ```
  swap((*iter1).firstmember, (*iter2).firstmember);
  ```

This will work just fine, but there is a C++ operator that combines these two (dereference and then select). I works like this:

```
swap(iter1 -> firstmember, iter2 -> firstmember);
```

(Note that you won't actually use this example code, because when you swap, you should just swap the entire struct. There's no need to swap the individual members of the struct separately.)

Assignment 1.2 [20 points]

Convert the OrderedPair class, which is provided below, into a templated class. Note that it will only work with types that have the operators + and < and << overloaded. But you should be able to try your templated class out with types string, MyString, double, FeetInches, Fraction, etc. I would encourage you to try these out.

Also, create a programmer-defined exception class named "DuplicateMemberError" and add an if statement to each of the two mutators to throw this exception if the precondition has not been met. The precondition is given as a comment in the header file. Notice that you can test your exception handling by entering the same number twice when prompted for two numbers.

Put your class in a namespace named "cs_pairs"

Finally, to show that your class will work with different types, and also to show that you know how to code a client that uses the templated class, modify the given client file so that it uses your class using int as the type parameter, and then, in the same main(), repeat the code again with a few changes necessary to make it use ordered pairs of Strings instead of ordered pairs of ints. One of the things you'll have to change is the generation of the random values for the ordered pairs. Here's what I used:

```
    string empty = "";
    myList2[i].setFirst(empty + char('a' + rand() % 26));
    myList2[i].setSecond(empty + char('A' + rand() % 26));
```

Here is the header file, orderedpair.h. The syntax for declaring a constant in a class may look mysterious. To use constants in a class, we have to declare it inside the class, then assign it a value outside the class, as you'll see below. (That's actually not true for int constants -- they can be assigned inside the class -- but we want our code to be flexible enough to handle different types.)

```
#include <iostream>

/* precondition for setFirst and setSecond: the values of first and second cannot be equal,
except when they are both equal to DEFAULT_VALUE.
*/


namespace cs_pairs {
    class OrderedPair {
        public:
            // Use the first of the following two lines to make the non-templated version work.
            // Switch to the second one when you begin converting to a templated version.

            static const int DEFAULT_VALUE = int();
            // static const int DEFAULT_VALUE;

            OrderedPair(int newFirst = DEFAULT_VALUE, int newSecond = DEFAULT_VALUE);
            void setFirst(int newFirst);
            void setSecond(int newSecond);
            int getFirst() const;
            int getSecond() const;
            OrderedPair operator+(const OrderedPair& right) const;
            bool operator<(const OrderedPair& right) const;
            void print() const;
        private:
            int first;
            int second;
    };


    // Leave the following const declaration commented out when you are testing the non-templated
    // version. Uncomment it when you begin converting to a templated version.
    // The templated version will require a template prefix as well as some minor edits to the code.
    // const int OrderedPair::DEFAULT_VALUE = int();

}
```

Here is the implementation file, orderedpair.cpp

```
#include "orderedpair.h"
#include <iostream>
using namespace std;

namespace cs_pairs {
    OrderedPair::OrderedPair(int newFirst, int newSecond) {
        setFirst(newFirst);
        setSecond(newSecond);
    }

    void OrderedPair::setFirst(int newFirst) {
    // if statement to throw an exception if precondition not met goes here.
        first = newFirst;
    }

    void OrderedPair::setSecond(int newSecond) {
    // if statement to throw an exception if precondition not met goes here.
        second = newSecond;
    }

    int OrderedPair::getFirst() const {
        return first;
    }

    int OrderedPair::getSecond() const {
        return second;
    }

    OrderedPair OrderedPair::operator+(const OrderedPair& right) const {
        return OrderedPair(first + right.first, second + right.second);
    }


    bool OrderedPair::operator<(const OrderedPair& right) const {
        return first + second < right.first + right.second;
    }


    void OrderedPair::print() const {
        cout << "(" << first << ", " << second << ")";
    }
}
```

Here is the client file.

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include "orderedpair.h"
using namespace std;
using namespace cs_pairs;




int main() {
    int num1, num2;
    OrderedPair myList[10];

    srand(static_cast<unsigned>(time(0)));
    cout << "default value: ";
    myList[0].print();
    cout << endl;

    for (int i = 0; i < 10; i++) {
        myList[i].setFirst(rand() % 50);
        myList[i].setSecond(rand() % 50 + 50);
    }

    myList[2] = myList[0] + myList[1];

    if (myList[0] < myList[1]) {
        myList[0].print();
        cout << " is less than ";
        myList[1].print();
        cout << endl;
    }

    for (int i = 0; i < 10; i++) {
        myList[i].print();
        cout << endl;
    }

    cout << "Enter two numbers to use in an OrderedPair.  Make sure they are different numbers: ";
    cin >> num1 >> num2;
    OrderedPair x;

    /* use this before you've implemented the exception handling in the class:
    */

    x.setFirst(num1);
    x.setSecond(num2);

    /* use this after you've implemented the exception handling in the class:
    try {
        x.setFirst(num1);
        x.setSecond(num2);
    } catch (OrderedPair::DuplicateMemberError e) {
        cout << "Error, you attempted to set both members of the OrderedPair to the same number."
             << endl;
        x.setFirst(OrderedPair::DEFAULT_VALUE);
        x.setSecond(OrderedPair::DEFAULT_VALUE);
    }
    */

    cout << "The resulting OrderedPair: ";
    x.print();
    cout << endl;
}
```

Submit Your Work

Name your source code files a1_1.cpp, orderedpair.cpp, orderedpair.h, and pairsclient.cpp. Execute the programs and copy/paste the output into the bottom of the files a1_1.cpp and pairsclient.cpp, making it into a comment. Use the Assignment Submission link to submit the four files. When you submit your assignment there will be a text field in which you can add a note to me (called a "comment", but don't confuse it with a C++ comment). In this "comments" section of the submission page let me know whether the programs work as required.