CS 3850—Object-Oriented Programming
California State University, Stanislaus
R. L. Zarling    Winter, 2006

Laboratory 2: Monday, January 9

Save each of these projects using the name given in the heading. Often you will be asked to modify a previously written program—to do so, first copy the program, giving it the new name, and then modify the copy.

lab02a.cpp: from pointers to iterators

Rewrite lab01b.cpp so that instead of using pointers to a templated type as parameters and return value, it uses iterators. You must first determine what kind of iterator is needed; use the least restrictive iterator type that has all the capabilities you need. Explain in the introductory program comments why neither an input iterator nor an output iterator will work; then write the function using a forward iterator. (Note: As discussed in class, “forward iterator” is not a concept that the C++ compiler knows about, so this is “merely” a naming issue—by calling it a forward iterator, and documenting it that way, you are telling potential clients what kinds of iterators will work with your code and what won’t. You are also promising that your code only uses the iterator with operations that an InputIterator can do.) As before, make the second parameter const just for practice, although it doesn’t matter much since the parameter is passed by value. Test it with the same driver program you used for lab01b.cpp (this shows that pointers can be used as forward iterators). Print a listing and sample run to turn in.

lab02b.cpp: using an iterator

Replace the char array from lab02a.cpp with an STL string, and the incidental char pointers with instances of string::iterator. You will need to include the header file <string>. (The STL string type is documented in the appendix of your text under basic_string, the type from which string is derived.) You will also need to change the cin.getline call to use the void getline ( istream &, string & ) form, which is defined in <string>. Because string iterators are random access iterators, you can compute the position of the found character in the string by forming the difference of iterators, just as you used the difference of pointers in lab01b.cpp. Print a listing and sample run.

lab02c.cpp: using a stream iterator

As you saw in part a, the architecture of maxrange required its parameters to be at least forward iterators. But by changing the architecture slightly, we can allow input iterators as parameters.

Write a function maxvaluerange with the following signature

template < class InputIterator >
int maxvaluerange ( InputIterator start, const InputIterator end );

As before, this function should find the largest int in the given range. In addition to being less general than before, in that it only works for a range of int (which we will fix in part d), the crucial difference here is that instead of returning an iterator referencing the largest element, it returns the value of the largest element. Thus, the argument you made in part a proving that you could not use an input iterator for that architecture is no longer valid for this one. In the process, you lose the ability to locate where the maximum occurred. In practice for most applications this is a very serious loss, so the STL algorithm max_element was written to the architecture in part a instead of this one.

Test maxvaluerange by creating an input iterator of int attached to cin along the lines of the sample program on page 45 of your text. To specify end-of-file for the integer input stream, type Ctrl-D. Print a listing and a sample run.
lab02d.cpp: deducing the iterator’s base type

To generalize `maxvaluerange` to work with iterators based on value types other than `int`, we need a way to specify that `maxvaluerange` returns the type whatever-the-base-type-of-InputIterator-is. As described in §2.7 of our text, how to do this has been the subject of some controversy, but it has finally been standardized.

Instead of specifying `int` as the return type, use the following expression

```cpp
typename iterator_traits<InputIterator>::value_type
```

The keyword `typename` specifies that the next expression is to interpreted as a type name (instead of the name of an identifier, for instance). The rest of the expression deduces the base type of `InputIterator`, which of course is the template parameter of your class.

If you used a temporary variable of type `int` in `lab02c.cpp`, as I think you did, you will also need to change its type to the base type of `InputIterator` in the same way. Add a statically-initialized array of floats to your revised program:

```cpp
float farray[] = { 7.2, 13.8, 1.1, 83.77, 25.0 };
```

and add a call to `maxvaluerange` for the array of floats:

```cpp
flargest = maxvaluerange( farray, farray + sizeof(farray)/sizeof(float) );
```

Print a listing and sample run to turn in.