1 Lab 7

Date: Mar 19, 2020

This document first describes the aims of this lab. It then provides necessary background. It then describes the exercises which need to be performed.

In the listings which follow, comments are any text extending from a # character to end-of-line.

1.1 Aims

The aim of this lab is to introduce you to STL containers and iterators. After completing this lab, you should have some familiarity with some STL containers, iterators and algorithms.

1.2 Background

A good source for background material for this lab is *this web page* which is at just the right level of detail. Please read it thoroughly upto and including the **Container classes and their associated iterator types** section. Please also scan the rest of the document as we will need some of that material in the third exercise. Make sure you understand the following:

- STL ranges represented by a half-open interval `[begin, end)`.
- The different iterator types.
- The container classes.

The facilities provided by the STL makes it possible to write code without loops. In the exercises which follow, we will write code both with and without explicit loops.

Important Note: Do not confuse the `cbegin()` and `cend()` iterators from your *Project 2* with the `begin()` and `end()` iterators in this lab. They are quite different.

1.3 Exercises

1.3.1 Starting up

Follow the *provided directions* for starting up this lab in a new git `lab7` branch and a new `submit/lab7` directory. Start a `script` session to log your interaction into a `lab7.LOG` file.

You will be doing all your work in your `submit/lab7` directory:

```
$ cd ~/i240?/submit/lab7
```

Copy over the exercises directory:

```
$ cp -r ~/cs240/labs/lab7/exercises .
```

1.3.2 Exercise 1: Outputting Containers via Elements

Change over to the 1-out directory. It contains a main.cc and out.hh. Familiarize yourself with `main.cc`; you will be modifying it slightly in this exercise and reusing it for subsequent exercises:

- The usage of `main.cc` is:

  ```
  $ ./main
  usage: ./main CONTAINER_SPEC INTS_DATA_FILE [INT]
  ```

  The arguments are:

  `CONTAINER_SPEC` Specifies the kind of container to be used.

  `INTS_DATA_FILE` The path to a file which should contain whitespace-separated integers. If specified as `-`, then the integers are read from standard input.

  `[INT]` This optional integer argument is used by subsequent exercises.

- After checking the validity of the arguments, `main()` uses `readIntsFromFile()` to read the data file specified by `INTS_DATA_FILE`.

- The guts of the action occurs within the `go()` function. Depending on the `containerSpec`, it creates the correct type of `container` from the incoming `ints vector`. For each container type it uses a constructor which uses the `[begin(), end())` range of the `ints vector`.

Compile the first exercise using the Makefile in the parent directory using `make -f ../Makefile`. You should be able to run the program using a test.data file in the parent directory:

```
$ ./main deque ../test.data
[ ]
[ 22 43 12 56 64 42 11 22 ]
$
```

The reason for the empty sequence on the first line of the output is that `loopFn()` is not fully implemented. Complete the implementation by looping through `[begin, end)`.

```
for (auto p = begin; p != end; ++p) {
  //output *p
}
```

Compile and test. Make sure to test all the different container types (listed within the `go()` function in `main.cc`). Note the different behavior for `set` and `multiset` which are sorted (unlike the mathematical definition of a set). Note also that the `set` container removes duplicates.

Note that `out.hh` contains an implementation for `noLoopFn()` which uses STL's `for_each()` algorithm. This algorithm's first two arguments specify the range over which to iterate. Its third argument is `print()`; this is a lambda function (a function which is anonymous and does not have a name) which prints its argument `i`.

When looking at the code for the `go()` function in main.cc, you may have noticed that one of the container types `"[]"` is unimplemented. This is meant to specify that the container to be used is a normal C++ array. You should now implement this:

- You will need to declare an array having `ints.size()` elements.
- You can use an index `0 <= i < ints.size()` to loop through `ints`, assigning successive elements from ints to the array. Note that even though `ints` is a `vector`, you can index it as `ints[i]` to get the `i`'th element.
- Once you have copied over the array, you can represent a range for the array as a pointer to its first element and a pointer one beyond its last element. You can use this range to call `loopFn()` and `noLoopFn()`.

Once you have made these changes, test and verify that you can use a C++ array as a container.

1.3.3 Exercise 2: Finding an Element in a Container

This exercise will search the container for the value specified by the optional third argument to `main()`.

Change over to the 2-find directory. It contains a find.hh file. It does not contain `main.cc`; copy over your completed `main.cc` from Exercise 1; change the `include "out.hh"` to `include "find.hh"`.

You should be able to compile the program (using the `Makefile` in the parent directory) without any errors, but it will not do anything as the implementations of `loopFn()` and `noLoopFn()` are empty.

Implement `loopFn()`. Simply iterate through the container over the range `[begin, end)`; dereference the iterator to get its value and compare with `arg`. If the comparison succeeds break out of the iteration loop and print `FOUND`; otherwise print `NOT FOUND`.

The `noLoopFn()` is even simpler. Use std::find() which takes 3 arguments: **`begin`**, **`end`** and the value being search for which is **`arg`**; **`std::find()`** returns an iterator; if the value being searched for is found then the iterator points to the found element; otherwise it is set to **`end`**. Use this to print `FOUND`, `NOT FOUND` as appropriate.

Compile and test to verify that your code works.

1.3.4 Exercise 3: Sorting Containers

Change over to the 3-sort directory. It contains a sort.hh file. It does not contain `main.cc`; copy over your completed `main.cc` from Exercise 1; change the `include "out.hh"` to `include "sort.hh"`.

Attempt to compile this exercise. It will fail with errors. The problem is that sort() requires a **random access iterator**; look at the section **Container classes and their associated iterator types** in the *background material* to check which containers have random access iterators. Comment out the other containers in the `go()` function. You can deactive code by putting it into a

```
#if 0

#endif
```

section.

You should now be able to compile and run the program. Verify that all the containers are now sorted.

1.4 Exercise 4: Collecting Elements with Removal

In this exercise, you will move elements from the specified container to a vector, except for the element which is specified by the optional `INT` argument to `main()`.

Change over to the 4-rm directory. It contains a rm.hh file. It does not contain `main.cc`; copy over your completed `main.cc` from Exercise 1; change the `include "out.hh"` to `include "rm.hh"`.

You should be able to compile the program (using the `Makefile` in the parent directory) without any errors, but it will not do anything as the implementations of `loopFn()` and `noLoopFn()` are empty.

Implement loopFn() by iterating over [begin, end). On each iteration copy over the current element over to the vector vec, if it is not equal to arg. If equal to arg do not copy it as long this is the first occurrence of arg in [begin, end). This can easily be taken care of by using some kind of isSeen flag. Note that you can append to a vector by using the push_back() member function.

The noLoopFn() is slightly more complicated:

1. Use std::find() to return an iterator x to the first occurrence of arg in [begin, end).
2. Declare a vector vec initialized with the elements from [begin, x).
3. If x != end, copy over the remaining elements [++x, end) to the vector. You can do so by using std::copy() with a back_inserter as **std**::**copy**(++x, end, **std**::**back_inserter**(vec)).
4. Use the provided outContainer() to output vec.

Compile and test.

1.4.1 Winding Up

Follow the *provided directions* for winding up this lab. Terminate your script session producing the log file lab7.LOG in your lab7 directory. Add all your files to git and commit. Then merge your lab7 branch into the master branch and commit your changes.