Lab 5 – Sorting, C++ Templates, and Recursion
CS 240 Fall 2016
October 21, 2016

This lab is intended to review sorting, provide some experience with C++ templates and the STL (including iterators), and to force you to think recursively.

Download (from Blackboard) and inspect the file *Lab5.tar.gz*. The tar ball contains files for a very simple C++ Sorter class, along with a driver program (in *Lab5.cpp*) that uses Sorter. Notice that Sorter contains a single dynamic array of integers, and several functions (signatures in *Sorter.h* and empty implementations in *Sorter.cpp*) for several different sort algorithms. You will implement these functions according to the specifications and requirements below, and then use templates to generalize the code to operate on types other than int.

Part 1 Simple Iterative Sorting: Part 1 is extremely easy. Simply implement functions insertionSortI(), selectionSortI(), and bubbleSortI(), in *Sorter.cpp*. (The 'I' as the last character in the function names indicates that these are implemented as iterative sorts, not recursive.) The functions should cause the dynamic array of integers in the Sorter object to be placed in non-decreasing sorted order. Find descriptions of the algorithms (along with code for selection sort and insertion sort) in the class slides. Note that the function signatures do not need to be changed in Sorter (to take an array parameter for example), because the array is contained *within* the Sorter object, not passed as an argument.

Part 2 Recursive Sorting: The usual examples of recursive sorts are merge sort and quicksort. However, as an exercise to get you to think recursively, please implement selectionSortR(), insertionSortR(), and bubbleSortR(), as *recursive* functions. Some notes about this task:

- You should not change the function signatures; each should return void and take no parameters. You may, however, define and use "auxiliary" functions that are called by your functions, and that take parameters. You should not use any additional data members to implement the recursion; specify smaller problems by passing parameters to your auxiliary recursive functions.
- It is true that selection sort, insertion sort, and bubble sort are inherently iterative sorting algorithms. I am asking you to implement corresponding recursive versions of these sorts that capture the spirit and behavior of the three sorts. In particular, this means:
    - Your recursive insertion sort should place numbers "in order", one by one, into the sorted portion of the final array.
    - Your recursive selection sort should find the "next element" for each spot in the array (but recursively).
    - Your bubble sort should exchange *only neighboring elements*.
    - None of your recursive functions may use any kind of loop (for, while, repeat, etc.)
- As described in class, a function that uses *direct recursion* calls itself, and function that uses *indirect recursion* may use multiple function calls where some function(s) in the chain gets called by other(s) again (and again!) before returning. To duplicate the behavior of nested loops, consider indirect recursion and/or multiple different functions for the three sort functions.

Part 3 Templates: Next, copy your working Sorter class into a class called TSorter, in files *TSorter.cpp* and *TSorter.h*. Please keep Sorter, in *Sorter.h* and *Sorter.cpp*, in tact---you will submit your code for both Sorter and TSorter. Update TSorter in the following ways:

- Make TSorter be a C++ template class that accepts a single type parameter to specify what kind of elements the
- class stores.
- Replace the dynamic integer array with a vector from the C++ STL. So whereas Sorter uses an integer array to store contained integers, TSorter should use a C++ vector to store whatever kind of object the template class is instantiated with. Replace int-based indices on loops with iterators, and make all other necessary changes.

Follow the format of *Lab5.cpp*, and write a test program for TSorter, using two more types of data (in addition to integers), namely C++ strings and some kind of object that you write yourself. Call your test program *Lab5T.cpp*, and include it in your submission.

Pack up your code according to our submission conventions, and submit to Blackboard.