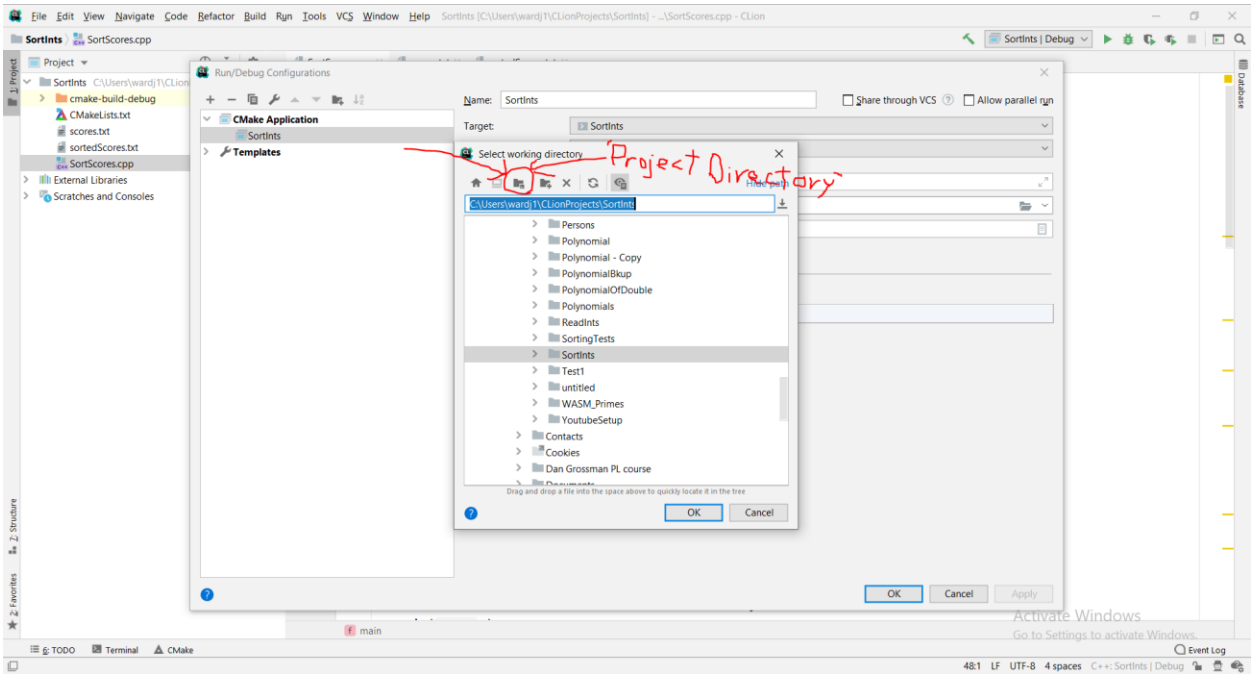
CSC 402/502 HW7 Instructor: Jeff Ward
Covers: C++ iterators 25 points

Setup Stage 1: Downloading and running the SortInts and Persons projects source code
There is a module on Canvas called "C++: Iterators and sorting". The associated videos walks through the source code (SortScores.cpp; PersonTests.cpp) and data files (scores.txt and sortedScores.txt; persons.txt and PersonsSorted.txt). These examples show how to call the *sort* function from the C++ standard library, passing it a couple of iterators that indicate the sequence to be sorted and also, optionally, passing it a comparator function that specifies the order in which you want the elements sorted. The ideas are familiar from the java.util.Arrays.sort and java.util.Collections.sort methods, as well as the java.util.Comparator interface. However, in these examples, we use a pair of iterators to indicate the beginning and end of the sequence that we want sorted, and also use a lot of overloading of the built-in C++ operators like <, >, <<, >>. This is standard (i.e. "idiomatic") practice in C++ but not so in Java.

Note: In order to enable, say, the SortScores.cpp program to find the scores.txt data file in the project directory, tell the project to use the project directory as the current working directory. In CLion (assuming that the name of the project is SortInts) you would do this by clicking the 'SortInts| Debug' dropdown, selecting 'Edit Configurations...', then clicking the folder icon in the 'Working directory' dropdown, and in 'Select working directory' dialog box, hitting the 'Project Directory' folder icon, which is shown in the screen shot below:



This video steps through the process: https://www.youtube.com/watch?v=dTtZEAfh_LM

Setup Stage 2: Downloading and running the starter code for HW7.
Create a new C++ project called MySortingTests. Add the MySortingTests.cpp and MyBubbleSort.h files to the project. Also download the BubbleSort.java and InsertionSort.java files, which you can view in whatever editor you prefer for Java files. A video is provided that steps through the code in these files.

Your assignment: Implementing *Insertion sort* using C++ iterators

Add to the MySortingTests project a new file called MyInsertionSort.h. (In CLion you would do this by right-clicking the MySortingTests folder in the project explorer and selecting "New->C/C++ Header File".)

MyInsertionSort.h is going to contain code similar to the code in MyBubbleSort.h, except you will implement Insertion sort. ADD COMMENTS AT THE TOP OF THE FILE PROVIDING YOUR NAME AND A ONE SENTENCE DESCRIPTION OF THE FILE CONTENTS. Your file needs to define one function template whose header should be:

```
template <typename RandomAccessIterator>
void myInsertionSort(RandomAccessIterator beginIter, RandomAccessIterator endIter) {
```

(Well, if you want to call the template parameter something other than 'RandomAccessIterator', that is okay. That identifier has no special meaning to the C++ compiler or linker. I chose that name as a signal to anyone reading the header that the body of the function will add and/or subtract integers from the iterators using + and/or -. But your function template MUST receive a type as a template parameter and MUST take two parameters of that type, as shown above. Inside the body of your function template, you will treat those parameters as iterators, as discussed below.)

The myInsertionSort function template must use the Insertion sort algorithm to sort the elements in the range [beginIter, endIter). Look at InsertionSort.java for a refresher on how Insertion sort works. Base your solution on the following basic iterator operations: ==, !=, <, <=, >, >=, *, +, -, ++, --. (You do not need to use all of them.) Your implementation should not need to be very long: the sample solution created by the instructor has a body that is only nine lines in length.

Once you have completed your myInsertionSort function template, uncomment the lines of code in MySortingTests.cpp that are involved in testing and timing it. Rebuild the project (in CLion: Build->Rebuild Project) and run it. The output should look something like the following. (I've added some commentary in red.)

Generating 50000 random values ...
Sorting each collection ...
Time for sort on vec1: 0.015955 seconds <== Note: sort is avg case O(n log n)
Time for sort on arr1: 0.008974 seconds
Time for myBubbleSort on vec2: 27.9921 seconds <== Bubble sort is avg case $O(n^2)$
Time for myBubbleSort on arr2: 5.55014 seconds
Time for myInsertionSort on vec3: 9.2981 seconds <== Insertion sort is avg case $O(n^2)$
Time for myInsertionSort on arr3: 1.52991 seconds

Swapping last two elements of each sorted collection ... <== Creating a case that is good for ...
Sorting each collection ... Bubble and Insertion sorts: O(n)
Time for sort on vec1: 0.007011 seconds
Time for sort on arr1: 0.002022 seconds
Time for myBubbleSort on vec2: 0.001026 seconds
Time for myBubbleSort on arr2: 0 seconds
Time for myInsertionSort on vec3: 0.000997 seconds
Time for myInsertionSort on arr3: 0.000999 seconds
Swapping first and last elements of each sorted collection ... <== Creating a case that is ...
Sorting each collection ... O(n) for Insertion sort, $O(n^2)$ for Bubble sort
Time for sort on vec1: 0.006981 seconds
Time for sort on arr1: 0.001997 seconds
Time for myBubbleSort on vec2: 16.1737 seconds
Time for myBubbleSort on arr2: 2.80948 seconds
Time for myInsertionSort on vec3: 0.002993 seconds
Time for myInsertionSort on arr3: 0 seconds
All checks passed!

Troubleshooting:
If it seems that the compiler does not "see" your MyInsertionSort.h file, then check to make sure that it is part of the source code for the project. In CLion, in your CMakeLists.txt file, you should see MyInsertionSort.h in a line like this:

```
add_executable(MySortingTests MySortingTests.cpp MyInsertionSort.h MyBubbleSort.h)
```

Submit your code for automated grading:
On Canvas, visit "Assignments"->"HW7: C++ iterators" and click the "Load HW7: C++ iterators" button at the bottom of the page. This will take you to Mimir. Upload your MyInsertionSort.h file and have Mimir test it. You can revise your program and resubmit if necessary.