DUE DATE: **By the deadline specified on Canvas**

CSE-443/543: High Performance Computing

Exercise #4

Max Points: 20

Objective: The objectives of this exercise are to refresh concepts related to:

- Work with files specified as a command-line argument
- Work with C++ classes
- Understand operator overloading and its use
  - Improve code reuse
  - Minimize the number of API methods a programmer has to remember
  - Reduce overhead of code maintenance

Fill in answers to all of the questions. For some of the questions you can simply copy-paste appropriate text from the Terminal window into this document. You may discuss the questions with your neighbor, TA, or your instructor.

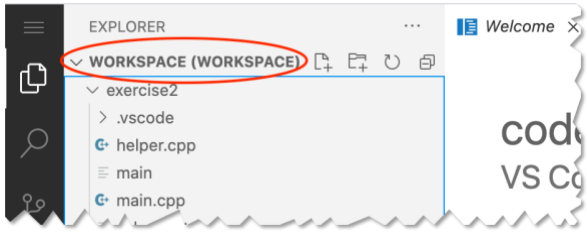
Name:

Part #0: Setting up the exercise on OSC

Estimated time: < 10 minutes

Exercise

1. Log into OSC's OnDemand portal via https://ondemand.osc.edu/. Login with your OSC id and password.
2. Startup a VS-Code server and connect to VS-Code. Ensure you switch to your workspace. Your VS-Code window should appear as shown in the adjacent screenshot.



3. Next, create a new VS-Code project in the following manner:
   a. Start a new terminal in VS-Code
   b. In the VS-Code terminal use the following commands:

```
$ # First change to your workspace directory
$ cd ~/cse443
$ # Use ls to check if workspace.code-workspace file is in pwd
$ # Next copy the basic template for a C++ project
$ cp -r /fs/ess/PMIU0184/cse443/templates/basic exercise4
$ # Copy the starter code for this exercise
$ cp /fs/ess/PMIU0184/cse443/exercises/exercise4/* exercise4
```

4. Now add the newly created exercise4 directory to VS-Code Briefly study the starter code in `main.cpp`. It is a repeat of Exercise #2 where we wrote code to find the 2nd maximum value. Except this method has been templatized to be very generic to work any data that meets certain requirements.

DUE DATE: **By the deadline specified on Canvas**

Part #1: Testing operation of the starter code

Estimated time: < 10 minutes

Exercise:

1. In the `main` function (in `main.cpp`) notice how the `get2ndMax` method is called with `int` as the template argument to have the method work with integers. The relevant code fragment is copy-pasted below:

```
// Get the second maximum value using the helper method.
auto max2nd = get2ndMax<int>(is);
```

   We can change the above template argument from `int` to `float` or `double` and the `get2ndMax` algorithm would just work with the other data types. Contrast this single source template-based approach with Java's code-duplication approach by viewing JDK source code for DualPivotQuicksort.java. How many versions of `sort` method are there in the source? Why? Also, notice how much code duplication has been done and think about how many LOC have to change if a tweak/bug-fix is needed to the sorting algorithm.
2. Test the program by setting the command-line argument to be `numbers.txt` (if needed, see video titled `Using command-line arguments in VS-Code` in the page titled `OnDemand & VS-Code demonstrations` Canvas).
3. The expected output is shown below:

```
$ ./main numbers.txt
2nd max = 82
```

Part #2: Modifying starter code to work with `Person` objects

Estimated time: < 20 minutes

Background: C++ uses a very different strategy for streamlining coding and API development via the use of operator overloading. The objective of operator overloading is to:

- Enable consistent style/pattern for coding
- Enable algorithmic descriptions that are data type agnostic
- Minimize the number of API methods a programmer has to remember
- Reduce overhead of code maintenance

Aspects of operator overloading to remember:

- Operator overloading is just like method overloading
- Operators are methods but with method names of the format `operator>>`, `operator<<`, `operator<`, `operator+` etc.
- Each operator has a specific signature that you need to use.
- Operator overloading is also used for typecasting.

Exercise:

The primary objective of this exercise is to modify `main` function to print 2nd maximum `Person`. Think about how you would go about accomplishing this 1-word change task prior to proceeding with the suggested steps below

1. Change one word in the main function as shown in the code fragment below:
```
// Get the second maximum value using the helper method.
auto max2nd = get2ndMax<Person>(is);
```
2. After the above 1-word change, try to compile the project. You will notice a lot of compiler errors. There are many errors because the compiler tries all kinds of options to get the program to work but will fail and print everything it tried. So, you will see a lot of error messages as the compiler tries a lot options to get your program to somehow work.
3. Briefly study the error messages you will notice error messages alluding to "`operator>>`" (stream extraction operator for reading), "`operator<<`" (stream insertion operator for printing), "`operator<`" (operator less-than for comparison). These error messages stem from the fact that the `get2ndMax` method is using those operators to perform operations but the `Person` class currently does not have those operators.
4. We have two choices here – either we can duplicate `get2ndMax` to work with `Person` or we can adapt the `Person` class to behave like any other data type. We will take the latter approach favoring consistency and maximizing code reuse.
5. Using the examples from lecture slides appropriately modify the `read`, `print`, and `lessThan` methods in `Person` class (both `.h` and `.cpp` files) to corresponding operators. All that is needed is changing names of the methods and adding declarations for operators in the header file.
6. Once you get your operators implemented correctly, the program should compile successfully.
7. Test the program by setting the command-line argument to be `persons.txt`. The expected output is:
```
$ ./main persons.txt
2nd max = 7 42 "james bond"
```
8. If you still feel unsure how or why this is a better engineering approach (i.e., algorithm did not change but we adapted our `Person` class to behave like standard classes such as `std::string`), ensure you discuss your questions with your peers and/or your instructor.

Part #3: Submit to Canvas via CODE plug-in

Estimated time: 5 minutes

In this part of the exercise, you will be submitting the necessary files via the Canvas CODE plug-in.

1. Download your `Person.h` and `Person.cpp` files from VS-Code to your local computer. **Ensure your file names does not have special characters**.
2. Upload the files using the "`Upload via CODE`" tab on Canvas.

Ensure you actually **submit** the URL generated by CODE plug-in in the final step as shown in the adjacent figure