Assignment 1

COMP-322, Winter 2017

Due: April 3rd, 2017 - 11:55pm

Please read the entire pdf before starting. You must do this assignment individually.

In this assignment, you will experiment with multiple ways to pass a function as input to another function. First, you will use a function pointer, which is similar to how one would do things in C. Then, you will use an abstract class, similar to how one does things in Java. Finally, you will use a Lambda expression similar to how one would do things in JavaScript (or C++ :)).

The idea of passing a function to another function is key in many cases when asynchronous programming is involved. This is because code doesn't normally flow from a single "main" method. The normal paradigm is to pass a "callback" function to another function. The callback function specifies what to do once the task is completed.

Sort with function pointers

Start with either of the 2 sort functions that you wrote in assignment one. Rename whichever one you keep to be simply `sortWithPointers` and add an extra input in your function header. The function should take an additional input, which is a function taking 2 `ints` as input and returning a `bool` representing whether the first argument should be before the the 2nd argument or not in the final sorted orders. Modify the contents of your `sort` function to use the results of this function.

For simplicity, you will probably want to add a typedef statement to the top of your file. For example, if you add `typedef bool (*compFunction)(int, int);` then you can use `compFunction` as a type to represent a function pointer to a function with 2 int inputs that returns bool.

Hint: Look at lecture 8 slides for help with the syntax.

Test Sort Function With Pointers

Write a function `testSortPointers` in which you demonstrate calling your new sort function first to sort an array (make one up) in increasing order, and next to sort it in decreasing order. To do this, you should define two functions `isLessThan` and `isGreaterThan`, each of which take 2 ints as input and return a bool representing whether the first argument is less than (or greater than) the second argument respectively.

(One could imagine that many other functions could be written instead such as to sort based on the absolute value of a number. Feel free to add additional methods if you like.)

Sort with interfaces

Next, we will do the same thing except with interfaces.

1. First, define a class `IComparer` which has one method defined on it `isBefore`. This should be a pure virtual method which takes two ints as input and returns a bool.
2. Create an additional method `sortWithInterfaces` which takes as input an `IComparer` instead of the function pointer. The body of the function should be largely identical other than when you use the input.
3. Next define a class `LessThanComparer` which extends `IComparer`. The class should have just the one method `isBefore` and should return whether the first input is less than the second input.
4. Next define a class `GreaterThanComparer` which extends `IComparer`. The class should have just the one method `isBefore` and should return whether the first input is greater than the second input.
5. Next, define a class `IsClosestToComparer` which extends `IComparer`. The class `IsClosestToComparer` should maintain a private property `center` and the method `isBefore` should return `true` if the first number is closer to `center` than the second number is (using the absolute value of the number). This private property should be set in the constructor for the object.

Test sort with interfaces

Write a function `testSortWithInterfaces` in which you demonstrate calling your newest `sort` function with all 3 different types of comparers.

Using a Lambda Expression

It is far from trivial to use function pointers to accomplish something such as we did with `IsClosestToComparer`. The problem is if we want to keep the comparison center point a variable, we need our function to have 3 inputs rather than 2. This would break the `sort` function that we wrote.

To address this, we will start by using a Lambda expression instead. However (and unfortunately), Lambda expressions can only be converted to function pointers implicitly if there are no *captured* variables. A captured variable is the idea that the function returned will *capture* a variable from its surroundings (in this case `center`.

As such, we will have to modify our `sort` function yet again. Write a function `sortWithStandardFunction` which is identical to the `sortWithFunctionPointers` (from the very first part) except that the input should be `function<bool(int,int)>` Note that `function` is a templated type defined inside the standard library (remember to add an appropriate using statement and also to include the file `functional`). The template type specifies the sort of function. In this case, the type will represent a function taking 2 ints as input and returning a bool.

Now that you have this 3rd version of the function, a benefit right away is that there *is* an implicit conversion between function pointers and `function` types. So your first sort function version is actually deprecated and you can see this by the fact that you can call your `sortWithStandardFunction` the same way that you call `sortWithFunctionPointers`.

The added benefit, is that we can call this now using a Lambda expression.

Write a function `generateNearestTo`. The function should take as input an `int center` and return a `function<bool(int,int)>`. The function it returns should take 2 things as input and return true if the first is closer (based on absolute value) to `center` than the 2nd is to it.

Hint: Your function is *capturing* the variable `center` so make sure that you include it inside your capture list (the bit between the square brackets) when you define the lambda expression.

Lastly, write a function `testSortWithLambdas` in which you first call `generateNearestTo` to generate a function that will compare with a distance based on comparing to the number 3 and then call it again to generate a 2nd function by comparing it to the number 10. Call your `sortWithStandardFunction` function to show how you would sort based on each of those functions.

What To Submit

You should submit one file called `FunctionalPrograming.cpp`. This file should contain all of the above code, copied here for convenience:

1. 3 sort functions: `sortWithPointers`, `sortWithInterfaces`, and `sortWithStandardFunction`
2. 3 test functions: `testSortWithPointers`, `testSortWithInterfaces` and `testSortWithLambdas`.
3. 2 functions : `isLessThan` and `isGreaterThan`
4. 4 classes: `IComparer`, `LessThanComparer`, `GreaterThanComparer`, `IsClosestComparer`.
5. 1 function that returns a function: `generateNearestTo`.