Lab 8: Classes

Assigned: *Tuesday, December 1st, 2020*
Due: *Monday, December 7th, 2020*
Points: *100*

- There is NO MAKEUP for missed assignments.
- We are strict about enforcing the LATE POLICY for all assignments (see syllabus).

Introduction

The assignment for this week will utilize concepts you've learned regarding working with C++ classes.

It is HIGHLY RECOMMENDED that you develop the algorithms for each program FIRST and THEN develop the C++ code for it.

Step 1: Getting Ready

First, open a terminal window and log into the correct machine. Change into your CS 16 directory, create a **lab08** directory and change into it. There are no skeleton files provided for you.

Step 2: Create and Edit Your C++ Files

This week, you will need to create **3 files: headers.h, functions.cpp** and **newanagram.cpp**. These correspond to the one exercise in this lab, described below. It is worth 100 points and everything (all 3 files) must be submitted for full assignment credit.

IMPORTANT NOTE: This lab will be also graded for use of comments and style. See below for details. In addition, we will take *major* points OFF if you use C++ instructions/libraries/code that either (a) was not covered in class, or (b) was found to be copied from outside sources (or each other) without proper citation.

newanagram.cpp

This program is a repeat of the anagram.cpp exercise you did in **Lab 5**, but with a twist: You will implement the program using a user-defined class called **`AString`**! If you recall, Lab 5's exercise was a program that determines if 2 strings are anagrams. An anagram is a word or phrase formed by rearranging the letters of a different word or phrase, for example, the letters in the word "listen" can be re-arranged to spell "silent".

In this exercise, you have to define a class called **`AString`**. In this string you must declare:

- A member variable called **`StringValue`** of type string
- Two (2) constructors: one that will not take arguments and set the value of the member variable to an empty string and one that will take one argument to initialize the memember variable.
- Member accessor function **`getAString`**
- Member mutator functions **`cleanUp`** and **`countLetters`**.

For all of the above, be very careful how you choose them to be private or public.

Your program must also define a function called **`compareCounts`** that takes in 2 integer arrays and returns a Boolean value that says if the 2 arrays are equivalent in their values or not.

REQUIREMENTS, ASSUMPTIONS:

1. Be sure to utilize ONLY techniques we've covered in lecture. Do not use "special" arrays or pointers, do not use vectors, do not use built-in sorting functions, etc… You will get zero points if you do.

2. You are given 3 skeleton programs: **headers.h**, **functions.cpp**, and **newanagram_skeleton.cpp**. You must complete missing lines in all 3 files and submit all 3 as well.

3. The skeleton program **newanagram_skeleton.cpp** has lines in it that **MUST NOT CHANGE** from what you are given. You should rename it to **newanagram.cpp** before you submit it.

4. In all 3 skeleton files, there are comments in there as guidelines to help you finish this exercise. You should READ THEM CAREFULLY and remove them before you submit. Of course, you are still expected to place appropriate comments of your own, per the styling guidelines of this course.

5. The member function **`getAString`** should ask the user for an input (see examples below). The input can be a sentence containing space characters. The input is assigned to the member variable **`StringValue`**.

6. The member function **`cleanUp`** should "clean up" the variable **`StringValue`** by removing all non-alphabet characters in it and then transforming all the remaining to lower-case characters.

7. The member function **`countLetters`** takes 1 argument: an integer array of size 26. The function should count the frequency of occurrence of all the letters in **`StringValue`** and place that count in the integer array in the appropriate index (0 is for a, 1 is for b, etc…)

8. The *program* function **`compareCounts`** takes 2 argument: 2 integer arrays each of size 26. The function should compare to see that the 2 arrays are the same. It returns a Boolean value of **false** if it is and **true** otherwise, as is implied by the statement in the skeleton file:
   `bool badCount = compareCounts(ca1, ca2);`

9. There are 2 constructors as is evidenced by the way that the class objects are created in the **main** function in **newanagram_skeleton.cpp**.

A session should look like one of the following examples (including whitespace and formatting). The user input is shown bolded for your convenience here. You'll note the similarity to the exercise in Lab 5.

```
$ ./newanagram
Enter string value: Eleven plus two!!
Enter string value: Twelve plus three
The strings are not anagrams.

$ ./newanagram
Enter string value: Rats and Mice:)
Enter string value: in cat's dream?!
The strings are anagrams.
```

You should test your program with multiple examples before you submit it. **Make sure your outputs match the above**. The new line between each run example above is NOT part of the program. The strings printed by the program should include a newline at the end, but no other trailing whitespace (whitespace at the end of the line).

Step 3: Create a makefile and Compile the Codes with the make Command

In order to learn another way to manage our source codes and their compilations, we will first create a **makefile** and put in the usual g++ commands in it. Afterwards, whenever we want to compile our programs, the Linux command is a lot shorter – so this is a convenience.

Using your text editor, create a new file called **makefile** and enter the following into it:

```
all: newanagram

newanagram: newanagram.cpp headers.h functions.cpp
        g++ -o newanagram newanagram.cpp -Wall -std=c++11
```

Then from the Linux prompt, you can do one of two things: either issue separate compile commands for each project, like so:

```
$ make newanagram
```

Or, you can issue one command that will compile all the projects mentioned in the **makefile**, like so:

```
$ make
```

If the compilation is successful, you will not see any output from the compiler. You can then run your programs, for example:

```
$ ./newanagram
```

If you encounter an error, use the compiler hints and examine the line in question. If the compiler message is not sufficient to identify the error, you can search online to see when the error occurs in general.

Remember to re-compile the relevant files after you make any changes to the C++ code.

Step 4: Submit using GRADESCOPE

BEFORE YOU SUBMIT YOUR FINAL VERSION FILES: Make sure that you:

a) **STYLIZE** your program appropriately to additionally make it easier on others reading your code. Apply ALL the style pointers I have posted on Piazza. This includes the use of comments.

This will be graded for an *extra 20 points* beyond the automatic grade you get from Gradescope.

b) **MEET THE REQUIREMENTS OF THIS LAB**. This means, if I said certain features or program aspects were *required* to be in your final code, then they must be there.

If these requirements are not met, you will lose MAJOR points on the exercise. This simulates a real-world situation (e.g. your customer wants certain features, so you must provide it – while I am not your customer, per se, I *am* giving you points for your efforts!)

Log into your account on **https://www.gradescope.com/** and navigate to our course site. Select this assignment. Then click on the "Submit" button on the bottom right corner to make a submission. You will be given the option of uploading files from your local machine or submitting the code that is in a GitHub repo. Choose the first option and follow the steps to **upload your 3 files** to Gradescope.

You may submit this lab multiple times. You should submit only after you test it on your computer (and CSIL) and are satisfied that the programs run correctly. The score of the last submission uploaded before the deadline will be used as your assignment grade.

Step 5: Done!

Once your submission receives a score of 100/100 on the autograder, you are done with this assignment. REMEMBER that there are 20 additional points that will be scored according to your proper use of comments and styling. The total will then be normalized to 100 points again (i.e. 120 score will be 100%) to grade this lab.

WE WILL BE CHECKING FOR PLAGIARISM – DO NOT COPY FROM OTHER STUDENTS OR FROM SOURCES ONLINE! USE PROPER CITATION OF CODE THAT YOU DID NOT WRITE! THE CONSEQUENCES WILL - AT MINIMUM - BE A ZERO ON THIS LAB AND POSSIBLY A ZERO IN THIS COURSE AND YOU WILL BE REPORTED TO THE UNIVERSITY.