CSci 1113: Introduction to C/C++
Programming for Scientists and Engineers
Homework 2
Spring 2019

Due Date: Monday, Feb. 18 before 5:30pm.

Purpose: In this homework you get the chance to write more C++ programs. These programs will involve some of the basic C++ constructs we have studied so far, including arithmetic operations and more complicated use of selection statements and boolean expressions. They will also involve simple looping structures.

Instructions: This is an individual homework assignment. There are two problems worth 30 points each. Solve each problem below by yourself (unlike the labs, where you work collaboratively), and submit each solution as a separate C++ source code file. Here are a few important reminders:

- Unlike the computer lab exercises, this is *not* a collaborative assignment. You must design, implement, and test the solution to each problem on your own without the assistance of anyone other than the course instructor or TAs. See the collaborative rules on the class website for more information.
- Remember to follow carefully all naming conventions, submit the correct files through the class website by the deadline, follow the example input and output, and ensure your code runs on cselabs computers running the Linux operating system.
- Also remember to use good style.
- In each problem one or more test cases is given as an example; however, your problem will also be graded on other test cases which you will not know in advance. (One part of programming is making up your own test cases and testing your program on them.)

Problem A: Planned Breaks (30 points)

When working on homework, it is often useful to take planned breaks. While it may seem strange, taking a short break every once and a while can make you much more productive in the long run. It is important, however, to make sure that a break doesn't run longer than intended. That's why it's recommended to plan your breaks, figure out when they should be done, and set an alarm to remind you to get back to work.

This problem is about a program a C++ teacher wrote to help him take effective breaks. The program asks the user for the current time, and how long of a break the user wants to take. Based on this information, the program computes when the break should end. The user could then take that time and set an alarm to make sure a break doesn't run longer than intended.

Unfortunately, the teacher seems to have made quite a few mistakes. In this problem, your task is to work with the provided, non-functional code and clean it up until you have a correct, working program. You should start by resolving the syntax errors that are

preventing the code from being compiled. Once done with that you will need to resolve the various logical issues with the code. This may require re-writing parts of the program.

Input validation: No input validation is needed for this problem. You can assume all inputs are valid. (i.e., you can assume that a reasonable time was entered, and that no integers are negative. You cannot, however, assume the break will be short, for example, the code should work with a long (40 hour) break just as well as a short (2 minute) break.

Output validation: Like question 1B on the last homework, only reasonably formatted times should be reported back: minutes should be reported between 00 and 59, hours between 1 and 12, and time should be reported as either AM or PM as appropriate.

12:00 Clarification: The exact crossover point between AM and PM, as well as the correct labeling for Noon (12:00PM) and midnight (12:00AM) are somewhat confusing. To be clear time goes from 11:59PM ⇒ 12:00AM(midnight) ⇒ 12:01AM⇒ . . . ⇒ 11:59AM ⇒ 12:00PM(noon) ⇒ 12:01PM ⇒ . . ..

Here are five examples of the program's expected behavior, when it is working correctly.

Example 1:

```
Input the current time as hours minutes and A (am) or P (pm):  10 13 A
Input the break time as hours minutes 1 22
Break is done at 11:35AM
```

Example 2:

```
Input the current time as hours minutes and A (am) or P (pm):  5 55 A
Input the break time as hours minutes 10 10
Break is done at 4:05PM
```

Example 3:

```
Input the current time as hours minutes and A (am) or P (pm):  11 07 P
Input the break time as hours minutes 0 53
Break is done at 12:00AM
```

Example 4:

```
Input the current time as hours minutes and A (am) or P (pm):  4 14 P
Input the break time as hours minutes 40 4
Break is done at 8:18AM
```

Example 5:

```
Input the current time as hours minutes and A (am) or P (pm):  10 45 A
Input the break time as hours minutes 1 15
Break is done at 12:00PM
```

To start this problem, download *brokenBreakTime.cpp* from the class Canvas page. When you are done, you should have a working source file for this program. You are not required to take, or turn in, notes about the bugs in the original code. You are only responsible for creating a working version of the program, based on the broken version available through canvas.

When you are done, name the source code file <username>_2A.cpp. Here you replace <username> with your U of M email address; for example, if your email address is

`smithx1234@umn.edu` your file should be named `smithx1234_2A.cpp`. Remember to follow this naming convention diligently. Then submit your file using the Homework 2 Problem A link on the class website.

Problem B: More Colors (30 points)

Simple color analysis occurs in many applications. For example, a pixel in a satellite image might be some shade of green for an area covered with vegetation, but a different color if deforestation has occurred. An astronomical image might have a star or planet of a certain color, and that can indicate something about that bodies' characteristics.

This problem asks you to construct a C++ program to work with color again. First have the user input some values. You may assume the values input are always integers, not doubles or characters, etc. To begin the input, have the user input a tolerance. Validate that tolerance: if it is negative ask the user to input a nonnegative tolerance. Do this repeatedly until they enter a nonnegative number. Then ask the user to input red, green, and blue components. Each should be an integer between 0 and 255, inclusive. Validate these: if any is outside that range, have the user input three new components. Do this repeatedly until the user enters three valid components.

Once the user has input all valid values, then check if the color is a "near gray," i.e., all the components are within (i.e., less than or equal to) the tolerance of each other, and output a message telling if they are or are not. For example, if the tolerance is 5 and the color is (199, 201, 204), then the components are within 5 of each other. But if the tolerance is 3 and the components as (101, 99, 103), then the components are not since the green and blue components differ by 4.

If the components are within the tolerance of each other, then your program should also identify which component or components have the largest value, and then output a message identifying that component or components. (If the components are not within tolerance, your program should *not* do this part.) For example, for the color (28, 100, 97) the largest component is green. For the color (0, 100, 100), the largest are green and blue. Note there are a number of possibilities — your program needs to be able to handle all the possible cases.

Here are four examples of input and output that show the desired program behavior as well as the input and output format and language. As usual, user input is underlined. Also, as usual, make sure you follow this output carefully. For example, notice that the language changes slightly when there is a single largest component versus when there are two or three.

First example:

```
Input tolerance:  2
Input red, green, and blue components:  255 255 255
The color is a near gray.  All components are equal.
```

Second Example:

```
Input tolerance:  0
Input red, green, and blue components:  100 110 100
The color is not a near gray.
```

Third Example:

```
Input tolerance:  -1
Tolerance must be nonnegative.
Input tolerance:  4
Input red, green, and blue components:  99 103 103
The color is a near gray.  The green and blue components are largest.
```

Fourth Example:

```
Input tolerance: -2
Tolerance must be nonnegative.
Input tolerance:  -1
Tolerance must be nonnegative.
Input tolerance:  5
Input red, green, and blue components:  0 0 257
All components must be between 0 and 255, inclusive.
Input red, green, and blue components:  99 100 104
The color is a near gray.  The blue component is largest.
```

Some hints:

- Make sure you understand the program specification given above. There are a number of different tasks in this program, and understanding them all, and how they fit together, is a key part of the assignment.
- Look carefully at the examples above for the input prompts, any validation messages, output message language, when new lines start, etc.
- The program logic for this program is significantly different from the logic in previous homeworks. So take some time before you start programming to think carefully about where you need to use repetition, where you need to use selection statements, what the conditions for those statement will be, etc.
- When constructing a complicated programs, it is often useful to implement it in parts. For example, for this program, write the input and validation code, and test and correct it, before writing any of the code for checking whether a color is near gray and what its largest components are.

Test your program on a variety of input to ensure it handles the different cases correctly. When you have written, tested, and corrected your solution, save your file as <username>_2B.cpp, where you replace <username> with your U of M username, and submit it through the class website. As usual, be diligent in following this naming convention.