CSCI 201 – Computer Science 1

Homework 2

Objectives. The objectives of this assignment are:
1. Learn how to track the number of conditionals evaluated when a program runs.
2. Learn to employ the divide and conquer strategy to reduce the number of conditionals evaluated.
3. Learn to construct a complex, nested, if-else statement in C++.

Question 1. Due on Day 4 of Module 1.
Recall that in Lab 2, we created a program that maps a numeric wavelength into a color of the visible spectrum. In this question, we will create an improved version of the program that has the following additional features.

1. *Deal with all possible inputs.* If the wavelength is lesser than 400.0, the program will print the message "wavelength too small".If the wavelength is greater than 700.0, the program will print the message "wavelength too large".
2. *Report the number of comparisons that were performed.* In order to do that, define a global counter variable at the beginning of the program. Increment this variable *immediately before* each condition that is checked.

With these changes the sample executions would look something like this:

```
Please input a visible wavelength:  495.6
Your wavelength corresponds to the color Blue.
You evaluated 4 conditional expressions.
```

```
Please input a visible wavelength:  795.6
Your wavelength is too large.
You evaluated 8 conditional expressions.
```

Carry out the following tasks:
1. *Designing a solution.* Figure 1 partially desribes the logic structure of the program. (Note that the first condition checks if the input is less than the lower bound, i.e., 400.) Modify the flowchart structure from Figure 1 to initialize a counter, and increment the counter before each conditional. Extend the modified flowchart to cover all the cases, and test it using Raptor. (Note that the last condition will be used to check if the input is greater than the upper bound, i.e., 700.)
2. *Implementing the solution.* Write a C++ program that implements your design, *using the nested if-else statement.*

What to turn in: Create a folder `HW2Q1` in CourseFiles and upload the following:
1. **The Design.** Raptor flowchart.
2. **Script file.** Start a script session. Display the program source (.cpp file). Compile the program,

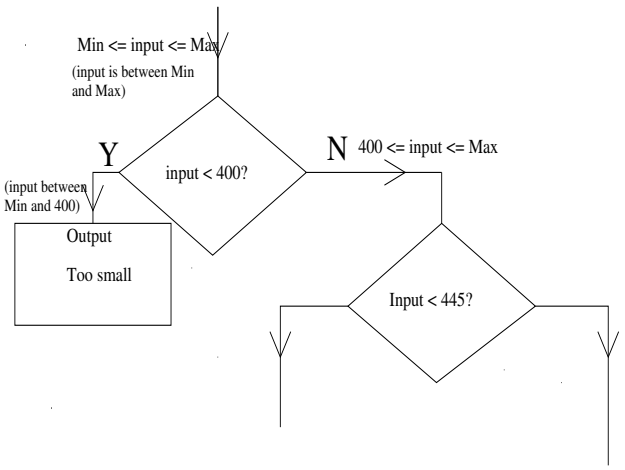

Figure 1: Partial flowchart for HW Question 1, starting with lower end of visible wavelength

and do one test.
3. **Testing table.** Create a testing table following the directions given at the end of this document.

Question 2. Due on Day 6 of Module 1.
One problem with checking out the conditionals in a linear fashion (i.e., compare the input against the dividers from left to right) is that we test a large number of conditions in some cases. We can reduce this number, if we start by trying a divider that is roughly in the middle. Figure 2 partially desribes the logic structure of the program. It shows a possible flowchart where the first comparison is made with 570. If the result is 'Y' (left branch) we will compare only with the dividers that are smaller than 570; otherwise, we compare only with the dividers that are greater than 570.

Carry out the following tasks:
1. Modify the flowchart structure from Figure 2 to include an increment of the counter before each conditional. Extend the modified flowchart to cover all the cases, and test it using Raptor. The resulting flowchart be such that the maximum possible number of comparisons is minimized.
2. Translate your flowchart into C++ code.

What to turn in: Create a folder `HW2Q2` and upload the following:
1. **The Design.** Raptor flowchart.
2. **Script file.** Start a script session. Display the program source (.cpp file). Compile the program, and do one test. Upload script file.
3. **Testing table.** Completed testing table.

Creating a Testing Table. Testing involves the following steps. As each step is completed, the results are recorded in a **Testing Table.**
Step 1. Brainstorm to come up with a set of good test inputs that will verify **all** the behaviors of the program. Record these in Column 1 of the table. We should have a set of test cases that check all possible branching paths through the flowchart.

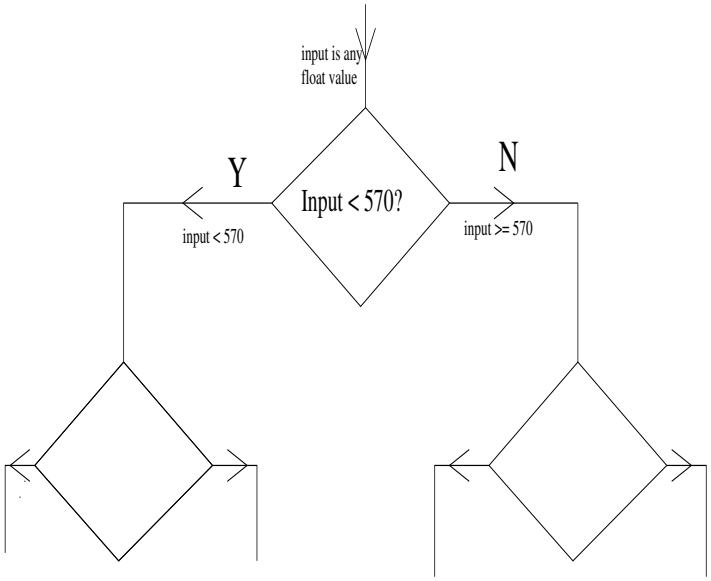

Figure 2: Partial flowchart for HW Question 2

Step 2. For each test input in Column 1, manually compute and record the expected output in the second column. *Remember that the output is both the color and the number of comparisons performed.*
Step 3. Run the program on each input and record the **actual output** into the third column of this table.
Step 4. If you find errors, note them in the fourth column of the table, fix them, and test again. **REMEMBER: there is no shame in finding an error in your program; rather, it shows that you did a good job of testing it.** The important thing is to document the errors and correct them.
Step 5. The fifth column lists the action that was taken, if any, to correct the program after the test.

Additional points to keep in mind:
1. All items must be **neatly presented.** Any item that is not requested as a printout may be **neatly** hand-written.
2. *The layout of the source code.* The code should be neatly indented and easy to read. Comments should be included to explain parts of the program as needed, and variable names should be meaningfully chosen. Please read sections 2.14 and 2.15 of the text for other suggestions.
3. *Use of Named Constants.* These prevent the occurrence of "magic numbers". In the flowchart, 475.0 is a "magic number" since a person reading it cannot immediately recognize its purpose. Instead, we put in a declaration `const double BLUE=475.0;` and then replace every occurrence of `475.0` with `BLUE`, which makes the program easier to follow. Read section 3.6 of the textbook for more details.

Extra credit for research. If you were told ahead of time that most of the input values are between 445 and 475, how would you organize the flowchart so that the number of conditionals evaluated is likely to be small? Construct the resulting flowchart and justify your organization. Can you think of other scenarios for input distribution? Try to analyze how you would design the

flowchart for each case. Turn this in separately in by creating a folder `ExtraCreditHW2` within your `CourseFiles` folder; upload the flowchart and the written analysis.