1 Instructions

You may work in **pairs** (that is, as a **group of two**) with a **partner** on this lab project if you **wish** or you may work **alone**. If you work with a partner, only submit **one** lab project with **both** of your **names** in the **source code file** to Blackboard for grading; you will each earn the **same** number of points. **What** to hand in, and **by when**, is discussed in **Section 5**; read it.

2 Lab Objectives

After completing this assignment the student should be able to,

- Complete all of the objectives of the previous lab projects.
- Effectively use **if-statements**, **if-else-statements**, and **if-elseif-elseif-...-statements** to alter flow of control.
- Write programs involving **multiple functions**.
- Effectively use the **pass-by-value** parameter passing technique.
- Perform and document **testing**.

3 Prelab

- First, skip to Section 4 and read the lab project software requirements. Then come back here.
- Create a new Code::Blocks C++ project named *Lab07.*
- Navigate to the course website and download the *Lab07.cpp* file. This file is a template containing most of the code for the lab project, but in various places, the code has not been completed. Your job shall be to complete the code by reading the comments and writing proper C++ code in the locations indicated by ??? symbols. Follow the instruction in Steps 31-35 of the Code::Blocks tutorial[1] to add *Lab06.cpp* to your Code::Blocks project.
- The *Lab07.cpp* source code file contains one test case documented in the header comment block which you may use to test the correctness of your program. You are two write two additional test cases, documenting them in the header comment block. Test Case 1 tests *f* = -1; write your two test cases so *f* = +1 in one of them and *f* = 0 in the other. You should write and document these two test cases before you write the code, but I am not asking you to submit them early.

4 Lab Exercise

Zelda (see Lab Project 4) has been quite busy divining from the stars to find an *Even Better Lucky Number Formula™* and she thinks she finally has it. The new formula is based on the same information as the previous formula (birthdate month, date, and year; height in centimeters; and weight in kilograms), but now also depends on the **color** of the customer's **eyes** and the **lengths** of the customer's **index** and **ring fingers**. In the *Even Better Lucky Number Formula™* shown below, the value of *e* is based on this table of eye colors,

Eye Color	***e***
Black	3
Blue	-2
Green	17
Brown	5
Gray	12
Hazel	-8
Red	4
Pink	0
Violet	11

$$lucky\,number = ConvertToInt\left(\left|2m^{2.2}+3d^{3.3}+5\sqrt{y}+f\left(\frac{eh}{3}+\frac{11w}{5}\right)\right|\right) mod\ 10+1$$

Comments on the formula,

- *m* is the month of the customer's birthdate, *d* is the date, and *y* is the four-digit year. These are **ints**.
- *h* is the customer's height, in cm, and *w* is the customer's weight, in kg. These are **ints**.
- *ConvertToInt* (...) in the formula means that the contents of the parentheses is converted to an integer value by **truncation** before performing the modulus operation. Hint: use the `static_cast<int>` operator.
- The vertical bar symbols surrounding the expression inside the parentheses represent the **absolute value** function. The absolute value of *n* is *n* if *n* is nonnegative and -*n* if *n* is negative. In the C++ Standard Library, there is an absolute value function named **abs()**, which is accessed by including the **cstdlib** header file. The input parameter can be an **int** or a **double**, so if you are using MSVS or Microsoft Visual C++ Express Edition, and you attempt to pass an **int** to **abs()**, then you will need to type cast the **int** value to a **double**, e.g., **int x = abs(static_cast<double>(n));**
- Variables *e*, *h*, and *w* are **ints**. The divisions of *eh* by 3 and 11*w* by 5 are to be performed as **integer divisions**.

1 http://devlang.com/cse100_codeblocks

- **Height** and **weight** will be input using **British units,** i.e., in inches and pounds. Height *h* and weight *w* in the formula are to be expressed in **SI units**, i.e., *h* is in centimeters and *w* is in kilograms. When converting from British units to SI units, **round** the converted height and weight to the nearest integer. The source code file contains a function named *Round*() that you may call to do this. It would be instructive to study *Round*() and convince yourself you understand how it works.
- The lengths of the **ring** and **index** fingers are input as **real numbers** in **inches,** e.g., 3.5" or 4.1". If the person's index finger is longer than his or her ring finger then the value of variable *f* is **+1**. If the person's index finger is shorter than his or her ring finger then the value of *f* is **-1**. If the two fingers are of identical length then *f* is 0[2].

A sample run of my program is shown in **Figure 1**. Yours shall work in an identical fashion, i.e., produce the same output for the same input. When the eye color menu is displayed, the user should enter a value in the range [1, 9]. If he or she does not, then the program shall **terminate** with an appropriate **error message** stating that an invalid eye color was selected. To make the program terminate early, i.e., before the end of *main()* is reached, you call the *exit()* function from the C++ Standard Library. The function declaration for *exit()* is in a header file named **cstdlib** which must be included. Function *exit()* has one input parameter, which is an int. Negative integer values are used in C and C++ to indicate to the OS that the program terminated abnormally—because something "bad" happened; therefore, when an invalid eye color is selected, call **exit(-1)** to terminate the program.. **Figure 2** shows how my program operates when an invalid eye color menu item is selected. Yours shall work similarly.

```
Zelda's Lucky Number Calculator

What is your name? Cletus
In what month were you born? 8
What was the date? 31
What was the year (yyyy)? 1961
What is your height (inches)? 70
What is your weight (lbs)? 190
What color are your eyes?
1.  Black
2.  Blue
3.  Green
4.  Brown
5.  Gray
6.  Hazel
7.  Red
8.  Pink
9.  Violet
Select Eye Color: 4
How long is your index finger? 3.1
How long is your ring finger? 3.8
Cletus, your lucky number is 10. Thank you, that will be $25.
```

Figure 1. Sample Run of the Lucky Number Program

```
Zelda's Lucky Number Calculator

What is your name? Wilma
In what month were you born? 9
What was the date? 15
What was the year (yyyy)? 1993
What is your height (inches)? 63
What is your weight (lbs)? 130
What color are your eyes?
1.  Black
2.  Blue
3.  Green
4.  Brown
5.  Gray
6.  Hazel
7.  Red
8.  Pink
9.  Violet
Select Eye Color: 55
An invalid eye color was selected. The program is terminating...
```

Figure 2. Program Terminating Because of Invalid Eye Color

2 The ratio of the lengths of a person's index finger to his or her ring finger is known as the 2D:4D digit ratio. In general, men have shorter index fingers than ring fingers (their 2D:4D digit ratio is less than 1) and women have longer index fingers than ring fingers (their digit ratio is greater than 1), although some men have a 2D:4D ratio greater than 1 and some women have a 2D:4D ratio less than 1. Scientists aren't really sure why this is, but there is speculation that it may have something to do with exposure to testosterone in the womb. See http://en.wikipedia.org/wiki/Digit_ratio.

4.1 Additional Programming Requirements

1. Update the **header comment block** in the source code template with your author information, your lab date and time, your lab TA, and the two test cases you are to write for the prelab exercise.
2. Carefully **format** your code and follow the **indentation** of the text as shown in the example programs of the textbook.
3. When you complete the program, you will **run it three times** using the three test cases you have documented in the header comment block as the **input**. For each test case document the **Observed Output**. If the **Expected Output** matches the **Observed Output** for all three test cases, then your program is probably implemented correctly. However, if there is a mismatch then it could be caused by one of two things: (1) the calculations you did in determining the expected output were incorrect and your test case contains incorrect data; or (2) your program has a bug in it and it is not outputting the correct results. In the case of (1) you should recalculate the expected output and then rerun the test. In the case of (2) you should locate the bug(s) and correct them. Keep re-running the program on the test cases until all test cases pass. When your testing is compete, **document** your if each of the test cases **passed** or **failed**.

5 What to Submit for Grading and by When

Upload the *Lab07.cpp* C++ source code file to Blackboard using the lab submission link by the deadline. If your program does not compile or run correctly, upload what you have completed for grading anyway (you will generally receive some partial credit for effort). The deadline for the complete lab project is **4:00am Sat 31 Oct**. Consult the online syllabus for the late and academic integrity policies.

6 Grading Rubric

1. Prelab Exercise 2 (1 pt)

The student was to have written two test cases documenting them in the header comment block of their *Lab07.cpp* source code file. The header comment block in the source code file should also document the actual output and whether the test case PASSED or FAILED.

a. For *two* reasonably well-written and sufficiently documented test cases in the .cpp file, assign **+1 pt**.
b. If the student made an attempt to write the test cases, but they are somewhat poorly written or not sufficiently detailed, assign **+.5 pts**.
c. If the student submitted less than two test cases, assign **+.25 pts** if the one submitted test case(s) is sufficiently documented.
d. Assign **+0 pts** for no documented test cases and/or for not documenting the actual output and the test cases results in the .cpp file.

3. Lab Exercise Program (0 to 4 pts)

a. If the submitted program does, or does not, compile and the student completed less than 50% of the required code correctly, assign **+1 pt**.
c. If the submitted program does not compile and the student completed more than 50% of the required code correctly, assign **+2 pts.**
d. If the submitted program compiles and the student completed more than 50% of the required code correctly, assign **+3 pts**.
e. If the submitted program compiles and is implemented perfectly, or close to perfect with only one or two minor mistakes, assign **+4 pts**.

4. Deadline was 4:00am Sat 31 Oct

1. Assign 20% bonus calculated on the earned pts for a submission prior to 4:00am Thu 29 Oct.
2. Assign 10% bonus calculated on the earned pts for a submission between 4:00am Thu 29 Oct and 4:00am Fri 30 Oct.
3. Deduct 0.5 pt for a submission between 4:00am Sat 31 Oct and 4:00am Sun 1 Nov.
4. Deduct 1 pt for a submission after 4:00am 1 Nov.