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:

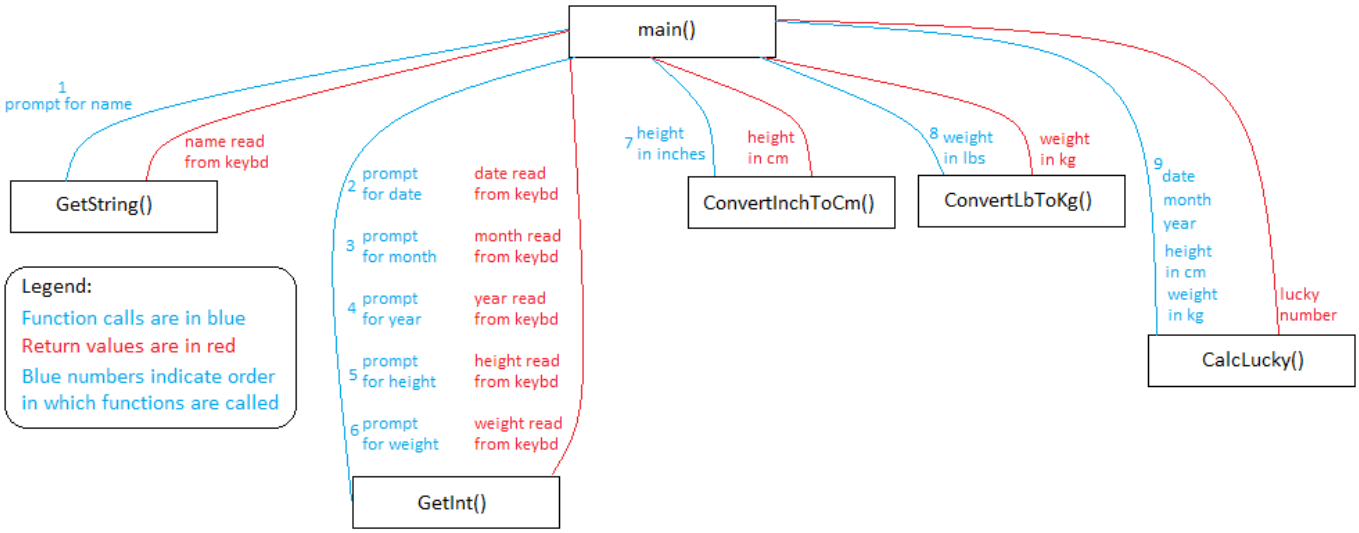
- Complete all of the objectives of the previous lab projects.
- Include the **cmath** header file and call predefined **C++ Standard Library functions**, specifically the **sqrt()** and **pow()** functions.
- Define and use numeric **named constants**.
- Write **arithmetic expressions** to perform numerical calculations.
- Use the **static_cast<> operator** to type cast a data value.
- Write a program involving **multiple functions**, call functions, pass parameters, and return values.
- Explain the **scope** and **lifetime** rules for **local variables**.
- Understand how **structure charts** are used to document the design.
- Translate **pseudocode** documenting the software design of a program into C++ code.
- Document formal **test cases** and perform testing to verify program correctness.

3 Prelab Exercises

1. Download the lab program template file (named *Lab04.cpp*) from the course website. Create a Code::Blocks project (or a project using your preferred C++ IDE) and add *Lab04.cpp* to the project. You will be completing the code in *Lab04.cpp* by writing statements to implement the lab project software requirements described in Section 4.

2. Read the description of the lab project exercise in Section 4. This description is called the **software requirements**.

3. Review the discussion in the Prelab Exercises section of Lab Project 3 regarding software design. A **structure chart** is a software design diagram which shows the relationships among the functions of a program. Here is the structure chart for this lab project showing the five functions you will write, the parameters to each function, the return values from each function, and the order in which the functions will be called.



Interpretation: *main*() first calls *GetString*() to get the customer's name, sending as a parameter a prompt string; *GetString*() returns the name that was read from the keyboard back to *main*(). Next, *main*() calls *GetInt*() five times sending prompts for the date, month, and year of the customer's birthday and the customer's height and weight; *GetInt*() returns the values that were read from the keyboard. Next, *main*() calls *ConvertInchToCm*() to convert the customer's height in inches to centimeters. Next, *main*() calls *ConvertLbToKg*() to convert the customer's weight in pounds to kilograms. Last, *main*() calls *CalcLucky*() passing the customer's birth date, month, and year, and the customer's height (in cm) and weight (in kg) as parameters; *CalcLucky*() calculates the customer's lucky number and returns it back to main.

We can document the more detailed design of each function using pseudocode, flow charts, or some other technique. For this lab project, I will give you the **pseudocode** for each function. Pseudocode is not written in a specific language nor is there a fixed syntax. Rather pseudocode can be simply written as English sentences, mathematical equations, and symbolic representation. The pseudocode for the lab program is shown below. Study both the structure chart and the pseudocode before your lab session so when you get to the lab you will be well prepared to begin writing the code.

```
Program Lab4

    Define const double CM_PER_INCH ← 2.54     -- There are 2.54 cm in an inch
    Define const double LB_PER_KG ← 2.20462262 -- There are 2.20462262 pounds in a kilogram

    Function CalcLucky (pDate : int, pHeight : double, pMonth : int, pWeight : double, pYear : int)
    Returns int
        int term1 ← 100 × pMonth^2
        int term2 ← 10 × pDate^3
        int term3 ← (term1 + term2) ÷ pYear
        double term4 ← pWeight^6 ÷ pHeight
        Return ConvertToInt (term3 + √term4) mod 10 + 1
    End Function CalcLucky

    Function ConvertInchToCm (pInches : double) Returns double
        Return pInches × CM_PER_INCH
    End Function ConvertInchToCm

    Function ConvertLbToKg (pLbs : double) Returns double
        Return pLbs / LB_PER_KG
    End Function ConvertLbToKg

    Function GetDouble (pPrompt : string) Returns double
        Display pPrompt
        Read a double from the keyboard into a double variable named n
        Return n
    End Function GetDouble

    Function GetString (pPrompt : string) Returns string
        Display pPrompt
        Read a string from the keyboard into a string variable named s
        Return s
    End Function GetString

    Function main () Returns 0
        Display "Zelda's Lucky Number Calculator"
        string name ← GetString("What is your first name? ")
        int month ← GetInt("In what month were you born? ")
        int date ← GetInt("What was the date? ")
        int year ← GetInt("What was the year (yyyy)? ")
        int heightInch ← GetInt("What is your height (inches)? ")
        int weightLb ← GetInt("What is your weight (lbs)? ")
        double heightCm ← ConvertInToCm(heightInch)
        double weightKg ← ConvertLbToKg(weightLb)
        int lucky ← CalcLucky(date, heightCm, month, weightKg, year)
        Display name ", Your lucky number is " lucky ". Thank you, that will be $25."
        Return 0
    End Function main

End Program Lab4
```

4 Lab Exercise

The world-renowned numerologist *Zelda the Great*[1] has discovered a truly amazing formula for divining a person's lucky number. However, Zelda—as great as she is at numerology—is no programmer, so she has asked you to write a program that will make the task of computing a customer's lucky number much easier.

After signing the requisite legal documents to establish a legally-binding mutually-agreeable contractual arrangement in which you will provide Zelda with a bug-free program that computes and displays lucky numbers in exchange for $1,000 in cold hard cash[2], she

1 I've met Zelda. Quite honestly, she's really not all that great. I'd say she's mediocre at best, but you didn't hear me say that. She does, however, make some fantastic brownies.

2 Tax free!

reveals that the lucky number is based on a person's birthdate (month, date, and four-digit year), their height (in centimeters), and their body weight (in kilograms). Let *m* be the customer's birth month (1 = Jan, 2 = Feb, … 12 = Dec), *d* be the date, *y* be the birth year (yyyy format), *h* the customer's height in cm, and *w* the weight in kg. Then the formula for the lucky number is,

$$lucky\,number = ConvertToInt\left(\frac{100\,m^2 + 10\,d^3}{y} + \sqrt{\frac{w^6}{h}}\right) mod\ 10 + 1$$

The program she wants you to write must meet these **software requirements**,

1. It shall ask the customer to enter his or her name (just the first name).
2. It shall ask the customer to enter his or her birthdate by entering the month, date, and year. The year shall be entered in YYYY format.
3. It shall ask the customer to enter his or her height in inches.
4. It shall ask the customer to enter his or her weight in lbs.
5. It shall calculate the correct lucky number using the super-secret-way-mysterious formula given above with height and weight converted to cm and kg, respectively. The division of the numerator $100m^2 + 10d^3$ by the denominator *y* shall be performed as integer division but the division of w^6 by *h* shall be performed as floating-point division. The contents of the parentheses shall be converted to an integer by truncation (hint: type cast) before performing the modulus operation.
6. It shall display the customer's name, lucky number, and shall ask the customer to pay Zelda $25 for the Secrets of the Universe™.

Here is a screenshot of my program in action. Make yours function similarly.

```
Zelda's Lucky Number Calculator

What is your first name? Cletus
In what month were you born? 3
What was the date? 13
What was the year (yyyy)? 1970
What is your height (inches)? 80
What is your weight (lbs)? 120
Cletus, your lucky number is 5. Thank you, that will be $25.
```

4.1 Testing

One of the most important phases of the software development process is **testing**. Just because a program compiles does not mean it is perfect. It only means there are no **syntax errors** in the code, but there could still be (and most likely will be) **logical errors** (which everyone calls **bugs**) that will cause the program to not meet the software requirements. For example, suppose you wrote the code so that when dividing $100m^2 + 10d^3$ by *y* it performed floating-point division rather than integer division. The program may display that Cletus' lucky number is 3 rather than 5. Since the program does not display the correct lucky number, it does not meet the software requirements.

Testing is an important part of software engineering education—there are entire classes devoted to the subject—and there are many different types or ways of testing a program. In CSE100 we will focus on what is called **acceptance testing**, which just means that we will just run the program and test it with different inputs to ensure that it meets the requirements.

A **test case** is a specification that is used to test the program against. The test case specifies **input data** to the program and documents **expected output** (or **behavior** if you wish). The test case is written **before** executing the program; in fact, test cases can be written during the **software requirements analysis phase** of the software life cycle, which occurs before coding even begins.

After the program compiles, it is executed, and the test case input data is used as input to the program. The program produces some output, and you compare the expected output (which was documented in the test case) to the actual, observed output. If they match, then the test case **passes** and should be documented as such. If the expected output does not match the observed output, then there are two possibilities: (1) the **test case is incorrect**, i.e., the expected output documented in the test case is incorrect because you made a mistake in determining what the expected output should be; or (2) there is a **bug** in the program. For situation (1) you should rewrite the test case to be correct. For situation (2) you should correct the bugs in the program. In either case, after fixing either your test case or the program code, you repeat running the program and performing the test cases until the program **passes** all of them. As an example of a test case, a test case for this lab program could be written and documented like that shown on the next page.

Test Case 1
DESCRIPTION:
Tests that the program computes and displays the correct lucky number.

TEST CASE INPUT DATA:
Customer name = Cletus
Birthdate month = 3
Birthdate date = 13
Birthdate year = 1970
Height in inches = 80
Weight in lbs = 120

EXPECTED OUTPUT GIVEN THE INPUT;
"Cletus, your lucky number is 5. Thank you, that will be $25."

OBSERVED OUTPUT:
"Cletus, your lucky number is 5. Thank you, that will be $25."

TEST CASE RESULT: PASSED

For this lab program, you shall write and complete **two test cases** with different input data. Document your test cases and testing results at the top of your source code file in the comment header block (see Item 1 in Section 4.2 below). Indicate if each of the test cases **passed** or **failed**.

4.2 Additional Programming Requirements

1. Modify the **header comment block** as the top of your *Lab04.cpp* source code file so it contains your information and your test cases. Document the results of your two test cases, indicating in the TEST CASE RESULT section if each test case PASSED or FAILED.
2. **Format** your code **neatly** and use proper **indentation** as shown in the example programs of the textbook and the source code the instructor posts online.
3. Your program **shall** define a double **constant** named *CM_PER_INCH*. Assign the value 2.54 to this constant. Use this constant when converting the height from inches to centimeters. Constants are normally defined outside of above **main()**, for example,

```
#include <blah-blah-blah>
using namespace std;

const double CM_PER_INCH = 2.54;

int main()
{
    ...
}
```

4. Your program **shall** define a double **constant** named *LB_PER_KG*. Assign the value 2.20462262 to this constant. Use this constant when converting the weight from pounds to kilograms.
5. Use the **double** data type for any variables which will contain real numbers (those would be height in cm and weight in kg). Use **int** if the variable will only store integer numbers (those would be month, date, year, height in inches, weight in pounds, and lucky number). Note that arithmetic on integer values is more efficient (i.e., it takes less time) than arithmetic on real values, so when possible, variables should always be defined as integers unless the numbers that will be stored in that variable are real numbers.
6. The predefined function from the C++ Standard Library to compute the square root of a number is named **sqrt()**. To use it, you would write something like this,

```
#include <cmath>  // cmath must be included or you will get syntax errors
...
double w, z = 9;
...
w = sqrt(z);      // assigns 3.0 to w
```

5 What to Submit for Grading and by When

Upload the *Lab04.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 3 Oct**. Consult the online syllabus for the late and academic integrity policies.

6 Grading Rubric

1. Lab Exercise Program (0 to 5 pts)

a. Assign **1 pt** for properly **documenting the two test cases** and the **testing results** in the header comment block.
b. 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 pt**.
d. If the submitted program compiles but fails testing and the student completed more than 50% of the required code correctly, assign **+3 pts**.
e. If the submitted program compiles and passes testing, assign **+4 pts**.

4. Deadline was 4:00am Sat 3 Oct

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