1 Instructions

- You may work with one partner as a group of two programmers on this lab project if you wish or you may work alone.
- We have created 80 Canvas groups named *Project Groups*. You can find the groups in Canvas by navigating to the *People* page and then clicking on the tab labeled *Groups*.
- If you intend to work with a partner, find an empty group and add yourself to the group and ask your partner to do likewise. If you will not be working with a partner, please proceed to add yourself to an empty group.
- When you have completed the lab project, the *group leader* (who is the first student to add him- or herself to the group) shall be responsible for uploading the submission zip archive to Canvas prior to the deadline.
- The other group member shall be responsible for downloading the submitted zip archive, extracting it, verifying that it contains the proper *main.cpp* file, and that the *main.cpp* file builds correctly, i.e., if it built without errors before you submitted it.
- To ensure you each receive the same score, it is imperative that you follow these instructions and those in §5 *Submission Information* (i.e., put both of your ASURITE ID's in the submission archive filename and write both of your names, ASURITE ID's, and email addresses in the AUTHOR1: and AUTHOR2: lines of the header comment block for each source code file).
- What to submit for grading, and by when, is discussed in §5; read it now.

2 Learning 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

We recommend that you complete prelab exercises 1-4 **before** your lab section meets.

3.1 Prelab Exercise 1: Create Project, Add *main.cpp* to the Project, Edit *AUTHOR* Comment Lines

- Create a new C++ project in Repl.it (or whichever C++ IDE you are using). Name your project *lab03*.
- Add a source code file named *main.cpp* to the project.
- When you download and extract the Lab Project 3 zip archive, you will find a source code file named *main.cpp* in the *src* directory. This file is a template for the C++ program you will write. Copy-and-paste the contents of this file into the *main.cpp* file that you created in Repl.it (or your favorite C++ IDE).
- Modify the header comment block to complete the AUTHOR1: line (and AUTHOR2: line if you work with a partner).

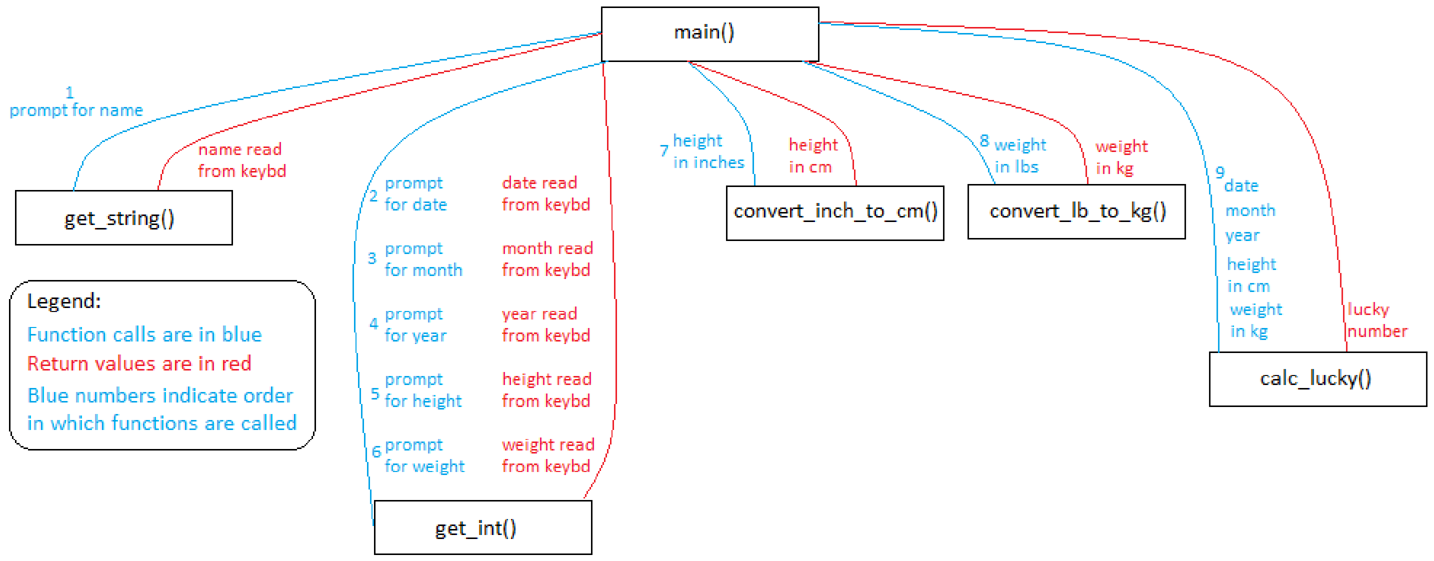
3.2 Prelab Exercise 2: Read Software Requirements

Read §4 of this lab project document, which describes what the lab project program will do, i.e., the **software requirements**. Then, come back here and read the remainder of this section.

3.3 Prelab Exercise 3: Study the Software Design

As we have been discussing in the lectures, **software design** is the phase in the **software development process** where we plan **how** we will implement the code so it meets the software requirements. The result of software design is documetation describing the implementation. There are many different ways of documenting the design, with two common ones being **structure charts** and **pseudocode**.

A **structure chart** is a diagram which shows the the functions of the program in a hierarchy with *main*() being the topmost function in the structure chart. We draw functions in rectangles and use arcs to show relationships between functions, i.e., which function calls which other functions, the arguments passed to a function, and the return value from a function. The image below is the structure chart for this lab project, showing the six 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.



There are six functions in this design:

- *main*()
- *calc_lucky*()
- *convert_inch_to_cm*()
- *convert_lb_to_kg*()
- *get_int*()
- *get_string*()

First, *main*() will call *get_string*()—passing the string literal argument "What is your name? " as the string prompt to be displayed—to get the customer's name. *get_string*() will read the name from the keyboard and return the entered string back to *main*(), which shall save the user's name in a local variable named *name*, defined to be of the **string** data type.

Next, *main*() calls *get_int*() five times sending string prompts for the birth date, birth month, and birth year of the customer's birthday and the customer's height and weight; *get_int*() returns the values that were read from the keyboard, which are assigned to **int** variables *birth_date*, *birth_month*, *birth_year*, *height_inch*, and *weight_lb*.

Next, *main*() calls *convert _inch_to_cm*() to convert the customer's height in inches to centimeters and then *main*() calls *convert_lb_to_kg*() to convert the customer's weight in pounds to kilograms.

Then, *main*() calls *calc_lucky*() passing variables *birth_month*, *birth_date*, *birth_year*, *height_cm*, and *weight_kg* as the function arguments. *calc_lucky*() calculates the customer's lucky number using the formula from the Software Requirements and returns it back to *main*(), which then displays the lucky number and ends the program.

Pseudocode can be used to document the design of each function. I have given you the pseudocode for the lab project below. You job will be to 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 start writing the code. To complete the lab program, implement this pseudocode.

```
file main — name the file main.cpp in your IDE

    define double constant CM_PER_INCH ← 2.54        — There are 2.54 cm in an inch
    define double constant LB_PER_KG ← 2.20462262 — There are 2.20462262 pounds in a kilogram

    — -----------------------------------------------------------------------------------------------
    — calc_lucky() - Given the customer's birth date, month, year, height in cm, and weight in kg, calculates the lucky number and
    — returns it as an int. Note I use the notation "→ int" to indicate that the return value from the function is an int. I use the
    — left arrow ← to indicate the assignment operator.
    — -----------------------------------------------------------------------------------------------
    function calc_lucky (date: int, month: int, year: int, height: double, weight: double) → int
        define term1, term2, term3, lucky as int variables   — These are local variables
        define term4 as double variable
        term1 ← $100 \times month^2$
        term2 ← $10 \times date^3$
        term3 ← $(term1 + term2) \div year$
        term4 ← $weight^6 \div height$
        lucky ← static_cast_to_int($term3 + \sqrt{term4}$) modulus 10 + 1
        return lucky
    end function

    — -----------------------------------------------------------------------------------------------
    — convert_inch_to_cm() - Given a length or height in inches, calculates and returns the equivalent length in centimeters.
    — -----------------------------------------------------------------------------------------------
    function convert_inch_to_cm (inches: double) → double
        return inches × CM_PER_INCH
    end function

    — -----------------------------------------------------------------------------------------------
    — convert_lb_to_kg() - Given a weight in pounds, returns the equivalent value in kilograms.
    — -----------------------------------------------------------------------------------------------
    function convert_lb_to_kg (lbs: double) → double
        return lbs ÷ LB_PER_KG
    end function

    — -----------------------------------------------------------------------------------------------
    — get_int() - Displays the specified string prompt, reads an integer value from the keyboard, and returns the integer.
    — -----------------------------------------------------------------------------------------------
    function get_int (prompt: string) → int
        output prompt
        define n as int variable
        n ← input integer from keyboard
        return n
    end function

    — -----------------------------------------------------------------------------------------------
    — get_string() - Displays the specified string prompt, reads a string value from the keyboard, and returns the string.
    — -----------------------------------------------------------------------------------------------
    function get_string (prompt: string) → string
        output prompt
        define s as string variable
        s ← input string from keyboard
        return s
    end function
    — -------------------------------------------------------------------------------------------
    -- main()
    — -------------------------------------------------------------------------------------------
    function main () → int
        define birth_date, birth_month, birth_year, height_in, weight_lb, lucky as int variables
        define height_cm, weight_kg as double variables
        define name as string variable
        display "Zelda's Lucky Number Calculator"
        name ← get_string("What is your first name? ")
        birth_month ← get_int("In what month were you born? ")
        birth_date ← get_int("What was the date? ")
        birth_year ← get_int("What was the year (yyyy)? ")
        height_in ← get_int("What is your height (inches)? ")
        weight_lb ← get_int("What is your weight (lbs)? ")
        height_cm ← convert_inch_to_cm(height_in)
        weight_kg ← convert_lb_to_kg(weight_lb)
        lucky ← calc_lucky(birth_date, birth_month, birth_year, height_cm, weight_kg)
        display name ", Your lucky number is " lucky ". Thank you, that will be $25."
        return 0
    end function
end file
```

3.4 Prelab Exercise 4: Writing and Documenting Test Cases

As we discussed in the lecture and in Lab Project 2, one of the most important stages of the software development process is **testing**. Again, the purpose of testing is to identify logical errors or bugs in the code. Although no amount of testing can ever prove that a reasonably large program is bug-free, a more-than-sufficient amount of testing should identify the most glaring errors. You may use Test Case 1 shown below to test your program; this test case is already documented in the header comment block of *main.cpp*. After you run your program using the test case input data, copy-and-paste the program output into the *Actual Output* section of the header comments and then document if the test case passed or failed in the *Test Case Result* section.

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

Input Data:
    Name: Wilma
    Birthdate month: 3
    Birthdate day: 13
    Birthdate year: 1970
    Height in inches: 80
    Weight in pounds: 120

Expected Output:
    Wilma, your lucky number is 5. Thank you, that will be $25.

Actual Output:
    Copy and paste the output from your program here

Test Case Result: *write* PASSED *or* FAILED

For this exercise, you shall create **two additional test cases** using different input data than Test Case 1. Your test cases **shall be completed and documented before** you even begin implementing the code. Make up some input data and use your calculator, Excel, or MatLab to determine by hand what the expected lucky number should be. Then, **document** the test case number, the test case input data, and the expected output in the **header comment block** of *main.cpp*. Then, after your complete writing the code, use your test cases to test your program. Document in the header comment block of *main.cpp* what the **actual output** from your program was, and indicate if the test case **passed** or **failed.** If the test case fails, the mistake is either in the test case itself or there is a bug in the program. You may need to re-calculate the expected lucky numbers by hand to check that your test case is correct. **Documenting the test cases and the testing results in the header comment block of *main.cpp* is worth 1 out of the 5 lab points, so do not skip this step.**

4 Software Requirements

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, quicker, and more lucrative.

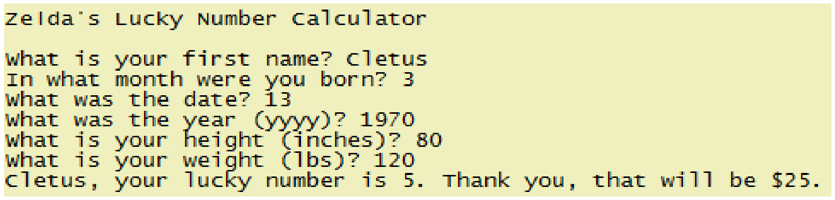
After signing the requisite legal documents to establish a legally-binding mutually-agreeable contractual arrangement in which you will provide Zelda with a bug-free program that computes and displays lucky numbers in exchange for $1,000 in cold hard cash[2], she reveals that the lucky number is based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

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

The program she wants you to write shall implement the following **software requirements**:

1. The behavior of the program is illustrated in the image below. The output from your program shall be the same.
2. When the program begins, it shall read the following personal information from the user:
   a. The customer's first name.
   b. The customer's birth month (as an integer, with 1 = January, 2 = February, ..., 12 = December).
   c. The customer's birth date (as an integer).
   d. The customer's birth year (as in integer, in YYYY format).
   e. The customer's height in inches (as an integer).
   f. The customer's weight in pounds (as an integer).
3. Prior to calculating the lucky number, the program shall convert the height in inches to centimeters and the weight in pounds to kilograms.
4. The program shall calculate the lucky number using the super-secret-way-mysterious formula given above:
   a. The height and weight shall be in centimeters (cm) and kilograms (kg), respectively.
   b. 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.
   c. The contents of the parentheses shall be converted to an integer by truncation (hint: use the static_cast<> operator) before performing the modulus operation.
5. The program 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 our program in action. Write your program so it outputs the same string prompts and messages.


```
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 Additional Programming Requirements

1. Modify the **header comment block** at the top of *main.cpp* source code file so it contains the first author's information in the AUTHOR1: comment. If there is only one author, then delete the AUTHOR2: comment. However, if there are two authors, document the second author's information in the AUTHOR2: comment.
2. Be sure the output from your program matches that as described in the software requirements and in the example run shown above.

[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. I mean, man, they are killer!!!

[2]Tax free!

3. To truncate a floating point value (a **double** value) to obtain an integer (an **int**) use the **static_cast<int>(*d*)** operator, where *d* is the floating point value to be truncated and converted to the **int** data type.
4. The C++ Standard Library function for calculating the square root of a number is *sqrt*(). To access the function, you must #include the *cmath* header file. To calculate the square root of *z* and assign the floating point result to a double variable named *x*, write: double x = sqrt(z); where *x* does not have to be defined if it has already been defined and *z* is a value of either the **int** or **double** data type.
5. Important note for Microsoft Visual Studio users (If you are not using MSVS as your C++ IDE, then ignore this item). Because of the way Microsoft has written their implementation of the C++ Standard Library, if you attempt to pass an **int** to *sqrt*(), the compiler will report a syntax error for reasons that will become apparent later in the course. In this case, you would need to typecast the **int** value to a **double** value, e.g.,

   ```
   double x = sqrt(static_cast<double>(z));
   ```
6. C++ does not have an arithmetic operator that performs exponentiation. Rather, The Standard Library contains a function named *double pow(double base, double exp*) which computes $base^{exp}$ and returns the result as a floating point value. To calculate -3.5 to the exponent 17.3, we would write: double z = pow(-3.5, 17.3); where *z* does not have to be defined if it has been defined prior to this statement.
7. One of the required three test cases is already documented for you in the header comment block. **Document your two required test cases** and the testing results in the header comment block.
8. Always write your code in a way to **enhance readability**. This includes **writing comments** to explain what the code is doing, **properly indenting** the statements inside *main*() and the other functions, and **using blank lines** to separate the various parts of the program. For guidance, study the example programs in the textbook and the source code the instructor writes in class or posts online.

5 Submission Information

- For help information about how to use Canvas, please visit the Canvas Student Guide.
- You are permitted to work in a group of 2 students. In Canvas, we have created 80 groups, as described in §1 *Instructions*. Please follow the instructions in that section to add yourself to a group and add your partner to your group if you wish to work with a partner. Consult the help documentation about how to join a group in this page of the Canvas Student Guide.
- When you are ready to submit your Lab 3 project, begin by using the file explorer program of your operating system to create a new empty folder named **lab03-*asurite*** where *asurite* is your ASURITE user id that you use to log in to MyASU (e.g., mine is *kburger2* so my folder would be named **lab03-kburger2**). If you worked with a partner in a team of two on the project, put both of your ASURITE id's in the name of the new folder, e.g., **lab03-*asurite1-asurite2***.
- If you named your Repl.it project *lab03* as requested in Prelab Exercise 1, then when you download your project from Repl.it, it will download a zip archive named *lab03.zip* containing *main.cpp*. Extract *main.cpp* from the downloaded zip archive and copy *main.cpp* to the empty lab03-*asurite* or lab03-*asurite1*-*asurite2* folder. Then, compress the folder creating a zip archive named lab03-*asurite*.zip or lab03-*asurite1*-*asurite2*.zip.
- Submit the lab03-*asurite*.zip or lab03-*asurite1*-*asurite2*.zip archive to Canvas by the deadline using the *L3: Lab Project 3* submission page.
- If you worked with a partner, the first student who adds him- or herself to the group will be designated as the *group leader*. If you worked in a team, the group leader shall upload the solution zip archive to Canvas. If you worked alone, then obviously, you shall submit it.
- Once you submit a file in Canvas for grading, you can download the zip archive and verify that it is not corrupted and that it is named correctly and contains the correct *main.cpp* file. Do this as soon as you submit!
- If your program does not compile due to syntax errors or if it does not run correctly because one or more of the software requirements are violated (i.e., the code contains one or more bugs), then upload what you have completed for grading anyway because you will generally receive partial credit points for effort.

- The submission deadline is **11:59pm Sat 27 Feb**. We shall permit lab submissions, up to 48 hours after the deadline, or in other words, you must submit before the 48-hour late submission deadline of **11:59pm Mon 01 Mar**. The penalty for a late submission after the Sat deadline but prior to 11:59pm Sun 28 Feb will result in a 10% penalty deduction (-0.5 pts). The penalty for a submission after 11:59pm Sun 28 Feb but prior to the 48 hour late submission deadline of 11:59pm Mon 01 Mar shall be a 20% deduction, i.e., -1 pt.
- The Canvas submission link will become unavailable exactly at 11:59pm on Mon 01 Mar and you will not be able to submit your project for grading after that time. We do not accept emailed submissions or submissions that are more than 48 hours later. Do not ask.
- Consult the online syllabus for the academic integrity policies.
- Note that we may employ an automated grading script to grade your lab projects. This script will depend on you naming the submission files as requested in this section and in Prelab Execise 1. Failure to follow this naming convention may result in a score of 0 being assigned for your lab project, even if it builds and runs correctly.

6 Grading Rubric (5 pts)

6.1 Test Cases

When testing the student's program, use these test cases.

Content intentionally omitted

6.2 Testing and Documentation (1 pt)

The student was asked to create two test cases, documenting them in the header comment block of their *main.cpp* source code file. The students were to document the results of testing their program in the header comment block by copying-and-pasting the output from their program to the *Actual Output* section and indicate in the *Test Case Results* section if the test PASSED or FAILED.

a. If the student omits the testing section then assign 0 pts.
b. If the student lists two test cases *documenting the input data and the expected output from the program* then assign **0.5 pt** (if only one test case is documented then assign 0 pts). You do not need to be concerned with checking if the expected lucky number is correct or not.
c. Next, for the two test cases the student was to write, if he or she documented the *actual output* from their program and whether the *test cases passed or failed*, then assign an additional **0.5 pt** (if only one test case is documented then assign 0 pts).

6.3 Lab Program (0 to 4 pts)

Attempt to build the student's program. If it does not compile due to syntax errors, then assign partial credit for effort as explained below. If the program does build, then run the program against the three test cases above. Then assign points per this rubric.

a. **Assign 1 pt**:	Program does not compile. *Less than half* of the complete program was implemented.
b. **Assign 2 pts**:	Program does not compile. *More than half* of the complete program was implemented.
c. **Assign 2.5 pts**:	Program compiles. *Less than half* of the code was implemented. ≥1 test cases failed.
d. **Assign 3.5 pts**:	Program compiles. *More than half* of the code was implemented. ≥1 test cases failed.
e. **Assign 4 pts**:	Program compiles. All test cases pass.

6.4 Submission Deadline was 11:59pm Sat 27 Feb

a. Canvas will automatically deduct 10% (-0.5 pts) for a submission between 11:59pm Sat 27 Feb and 11:59pm Sun 28 Feb.
b. Canvas will automatically deduct 20% (-1 pt) for a submission between 11:59pm Sun 28 Feb and 11:59pm Mon 01 Mar.
c. After Mon 01 Mar, submissions are not accepted for any reason