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 are going to switch on this project and change the way we handle groups. I believe we are far enough into the semester that most of you have probably settled on the partner you are going to work with for the rest of the term. So, I 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. You are your partner may continue to belong to the same Lab Finals Group *n* to the end of the term if you wish. Of course, you may always leave a group and join another group, so the partner you work with now does not have to remain your partner. 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 Lab Objectives

After completing this assignment the student should be able to:

- Complete all of the objectives of the previous lab projects.
- Write function definitions, call functions, pass parameters, define local variables, return a value from a function.
- Open a text file for reading and writing and read from and write to a text file.

3 Prelab Exercises

To make maximal use of your 50 minute lab session, perform these prelab exercises prior to your lab session so will be better prepared to begin writing the program for the lab project.

3.1 Prelab Exercise 1: Read Software Requirements

Read §4 *Software Requirements* of this lab project document which describes what the lab project program will do, i.e., the **software requirements**. Then come back here and complete the remaining prelab exercises.

3.2 Prelab Exercise 2: Mathematics Review – Quadratic Equations

The lab project is to write a program which finds the roots of a quadratic equation. If that sentence makes sense to you and you know how to find the roots of a quadratic equation, then you may proceed to the next prelab exercise. Otherwise, spend some time reading this section to review quadratic equations, what the roots of a quadratic equation are, and how to find them. It will pay off when you write your test cases, implement the design, and test and debug your program.

[Ref: Math is Fun: Quadratic Equations] A quadratic equation is a **polynomial** equation of degree 2, commonly written in the form $p(x) = ax^2 + bx + c$, where x is a variable and a, b, c are called the **coefficients** of the equation. A **homogeneous** quadratic equation is one that is equal to 0, e.g., $p(x) = 3.1x^2 + 2x - 5 = 0$. Note that the coefficients in this problem are positive or negative **real numbers** (in mathematics, they could also be complex numbers, but our program will only handle real numbers).

If x is a real number and $p(x) = 0$ then x is called a **root** of the equation. The degree of a polynomial equation is the value of the largest exponent, so a polynomial of degree n will have exactly n roots. Since the degree of a quadratic equation is 2, such an equation equation will have exactly two roots. However, depending on the values of the coefficients, some or all of these roots may be complex (i.e., *not* real numbers).

The well-known quadratic formula can be used to determine the two real roots of a quadratic equation—if they exist,

$$root_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$
$$root_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

The $b^2 - 4ac$ term under the radical is called the **discriminant**, and if the discriminant is ≥ 0, then both roots are real.

Exercises (See Appendix A for the Solutions)
Consider the equation: $-3x^2 + 6x - 18 = 0$.

Exercise 1: Write out the discriminant by substituting the values of the coefficients.
Exercise 2: Evaluate the discriminant.
Exercise 3: Is the discriminant positive, zero, or negative?
Exercise 4: Based on your answer to Ex. 3, how many real roots does the equation have?
Exercise 5: Based on your answer to Ex. 3, how many complex roots does the equation have?
Exercise 6: Based on your answer to Ex. 3, does the equation have two real roots which are equal to each other?
Exercise 7: Using your calculator, find the real roots of the equation (do not be concerned about any complex roots).

Consider the equation: $-3x^2 + 6x + 18 = 0$.

Exercise 1: Write out the discriminant by substituting the values of the coefficients.
Exercise 2: Evaluate the discriminant.
Exercise 3: Is the discriminant positive, zero, or negative?
Exercise 4: Based on your answer to Ex. 3, how many real roots does the equation have?
Exercise 5: Based on your answer to Ex. 3, how many complex roots does the equation have?
Exercise 6: Based on your answer to Ex. 3, does the equation have two real roots which are equal to each other?
Exercise 7: Using your calculator, find the roots of the equation (do not be concerned about any complex roots).

3.3 Prelab Exercise 3: Create Project, Add *main.cpp* to the Project, Edit *AUTHOR* Comment Lines

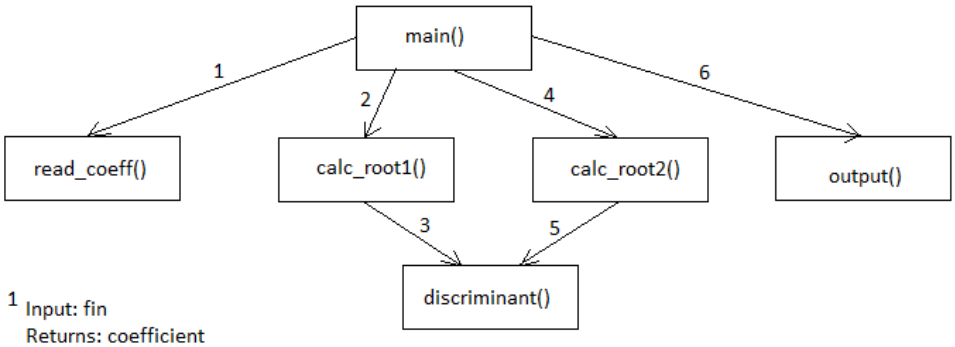
- Create a new C++ project in Repl.it (or whichever C++ IDE you are using). Name your project *lab04*..
- Add a source code file named *main.cpp* to the project.
- When you download and extract the Lab Project 4 zip archive, you will find a source code file named *main.cpp* in the *src* directory. This 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).

3.4 Prelab Exercise 4: Create Test Input File *coeffs.txt*

The program shall read input from a text file named *coeffs.txt*. The format of the file is documented in §4. In Repl.it, add a new file to your project and name the file *coeffs.txt*. Enter 2.5, 6.7, and -3 in the file for the coefficients *a*, *b*, and *c*, respectively.

3.5 Prelab Exercise 5: Study the Software Design

The program shall consist of the following six functions: *main*(), *read_coeff*(), *discriminant*(), *calc_root1*(), *calc_root2()*, *output*(), and *main*(). See the comments preceding each function definition in the pseudocode for an explanation of what the function does. Then, study the *structure chart* below, which illustrates the function calls that take place during program execution:



Here is the pseudocode that your program shall implement.

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

```
— ------------------------------------------------------------------------------------------------------------------------------------------------------------
— read_coeff() – This function is called from main() three times. The file input stream object named fin in main() is passed as the argument
— (it must be passed by-reference, so make sure to put the & symbol following the data type of the parameter). It reads the next coefficient
— from the file and returns it. Note that this function does not close the file; it is closed in main() after this function has been called the
— third time to return the c coefficient.
— ------------------------------------------------------------------------------------------------------------------------------------------------------------
function read_coeff (fin : ifstream&) → double
    define double variable coeff
    read from fin into coeff          — Reading from an input stream is like reading from cin, we use the >> operator
    return coeff
end function

— ------------------------------------------------------------------------------------------------------------------------------------------------------------
— discriminant() – Given the coefficients a, b, and c of the quadratic equation, this function calculates the value of the discriminant
— and returns the value. This function is called from calc_root1() and calc_root2().
— ------------------------------------------------------------------------------------------------------------------------------------------------------------
function discriminant (a : double, b : double, c : double) → double
    return b^2 – 4ac
end function

— ------------------------------------------------------------------------------------------------------------------------------------------------------------
— calc_root1() – Given the coefficients of the quadratic equation, this function uses the "-b + ..." root equation to calculate the first root.
— This function is called from main().
— ------------------------------------------------------------------------------------------------------------------------------------------------------------
function calc_root1 (a : double, b : double, c : double) → double
    return (–b + √discriminant (a, b, c)) ÷ (2a)
end function

— ------------------------------------------------------------------------------------------------------------------------------------------------------------
— calc_root2() – Given the coefficients of the quadratic equation, this function uses the "-b - ..." root equation to calculate the second root.
— This function is called from main().
— ------------------------------------------------------------------------------------------------------------------------------------------------------------
function calc_root2 (a : double, b : double, c : double) → double
    return (–b – √discriminant (a, b, c)) ÷ (2a)
end function

— output() – The input parameters are the three coefficients a, b, and c, and the two roots root1 and root2. See the Software Requirements
— for a discussion of the format of the output file. This function is called from main().
— ------------------------------------------------------------------------------------------------------------------------------------------------------------
function output (a : double, b : double, c : double, root1 : double, root2 : double) → nothing
    define ofstream object named fout specifying to open roots.txt
    configure fout so real numbers are displayed in fixed notation with five digits after the decimal point
    send to fout "The equation ", a, "x^2 + ", b, "x + ", c, " = 0 has roots ", root1, " and ", root2, endline
    close fout
end function

— ------------------------------------------------------------------------------------------------------------------------------------------------------------
— main() – The program begins executing in main(). main() opens the input file for reading, reads the three coefficients from the file, and
— closes the file. Then, it calculates the two roots of the quadratic equation (the coefficients shall be chosen such that there are no complex
— roots). Finally, it outputs the equation and the roots.
— ------------------------------------------------------------------------------------------------------------------------------------------------------------
function main () → int
    define ifstream object named fin specifying to open coeffs.txt
    a ← read_coeff (fin)          — Call read_coeff() passing fin as the arg. Store the returned value in a double var named a
    b ← read_coeff (fin)          — Call read_coeff() passing fin as the arg. Store the returned value in a double var named b
    c ← read_coeff (fin)          — Call read_coeff() passing fin as the arg. Store the returned value in a double var named c
    close fin                     — Close coeffs.txt since we are done reading from the file
    root1 ← calc_root1 (a, b, c)  — Call calc_root1() passing the coeffs as args. Store the returned root in a double var named root1
```

```
        root2 ← calc_root2 (a, b, c)   — Call calc_root2() passing the coeffs as args. Store the returned root in a double var named root2
        output (a, b, c, root1, root2) — Call output() passing the coeffs and roots as args. output is a void function and does not return a value.
        return 0                       — main() must return 0 to indicate to the OS that the program finished normally.
    end function main

end file
```

3.6 Prelab Exercise 6: Testing

After uploading *main.cpp* to Repl.it, edit the file. In the *Testing* section of the header comment block, we have documented one test case, using the coefficients of the example equation from §4. For this exercise, you are to design and document **three additional test cases**, for a total of four test cases.

You may choose any values for the coefficients, but make certain to choose coefficients such that the discriminant is non-negative; if you do not, then your program will crash when it attempts to calculate the square root of a negative number. We will not test your program using illegal coefficients.

Test cases should be written even before the software design stage, which precedes the program implementation stage, or, in other words, you do not need to finish writing the program before you write test cases. All of the information you must have to write a test case is found in the Software Requirements.

Therefore, for this prelab exercise, use your calculator to determine what the two roots should be for the quadratic equation specified by the coefficients you choose for each test case. Document the coefficients in the *Input Data* section of each test case. Document what the *Expected Output* will be from the program. After your program is written, test your program against each of the four test cases. For each test case, document the *Actual Output* in the test case and indicate if the test cased Passed or Failed (it passes if the *Expected Output* matches the *Actual Output*).

Recall that a test case may fail for one of two reasons: (1) There is a bug in the code which must be located and fixed; or (2) You made a mistake when calculating the expected output for a test case and the output from the program is correct, and the error is not in the code, but rather in the test case itself. In this case, re-calculate what the expected output should be, and then run the program against the test case again.

4 Software Requirements

1. When the program starts, it shall open an input text file named *coeffs.txt* for reading. The contents of *coeffs.txt* shall be as follows, where *a*, *b*, and *c* are real numbers separated by spaces:

   coeffs.txt
   a b c

2. After the input file is opened, the program shall read the three coefficients *a*, *b*, and *c* of a quadratic equation.
3. After reading the final coefficient *c*, the input file shall be closed.
4. The program shall calculate the two real roots of the quadratic equation specified by the coefficients *a*, *b*, *c*. Important Note: the coefficients shall be selected such that the discriminant is non-negative, making all of the roots real numbers.
5. The program shall open an output text file named *roots.txt* for writing.
6. The program shall output the quadratic equation to the output file in the format shown below, where *a*, *b*, and *c* are the coefficients, $root_1$ is the real root calculated using the root formula -*b* **+** ..., and $root_2$ is the real root calculated using the root formula -*b* **–** .... Note: All real numbers shall be printed in fixed point notation with five digits after the decimal point.

   roots.txt
   The equation *a*x^2 + *b*x + *c* = 0 has roots $root_1$ and $root_2$.

7. After send the output to the output file, the program shall close the output file.
8. The program shall end by returning zero from the *main*() function.

For example, consider this example input file and the output file that shall be produced given this input:

coeffs.txt

```
2.5 6.7 -3
```

Then our equation is $p(x) = 2.5x^2 + 6.7x - 3 = 0$, and the output should be,

roots.txt

```
The equation 2.50000x^2 + 6.70000x + -3.00000 = 0 has roots 0.39078 and -3.07078.
```

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.

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. One of the required three test cases is documented in the header comment block. **Document *your* three test cases** and the testing results in the header comment block.
3. 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 What to Submit for Grading and by When

- 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 4 project, begin by using the file explorer program of your operating system to create a new empty folder named **lab04-*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 **lab04-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., **lab04-*asurite1-asurite2***.
- If you named your Repl.it project *lab04* as requested in Prelab Exercise 1, then when you download your project from Repl.it, it will download a zip archive named *lab04.zip* containing *main.cpp*. Extract *main.cpp* from the downloaded zip archive and copy *main.cpp* to the empty lab04-*asurite* or lab04-*asurite1-asurite2* folder. Then, compress the folder creating a zip archive named lab04-*asurite*.zip or lab04-*asurite1-asurite2*.zip.
- Submit the lab04-*asurite*.zip or lab04-*asurite1-asurite2*.zip archive to Canvas by the deadline using the *L4: Lab Project 4* 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 06 Mar**. 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 08 Mar**. The penalty for a one day late submission is 10% of the project points (-0.5 pts). For a two day late submission, the penalty is 20% or -1 pt.
- The Canvas submission link will become unavailable exactly at 11:59pm on Mon 08 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 the document. 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

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 **three** 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 only one or two test cases *documenting the input data and the expected output from the program* then assign **0.5 pt** (if no test cases are documented then assign 0 pts). You do not need to be concerned with checking if the expected values are correct or not.
c. Next, for the three 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 less than three test cases are 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 06 Mar

a. Canvas will automatically deduct 10% for a submission between 11:59pm Sat 06 Mar and 11:59pm Sun 07 Mar.
b. Canvas will automatically deduct 20% for a submission between 11:59pm Sun 07 Mar and 11:59pm Mon 08 Mar.
c. After Mon 08 Mar, submissions are not accepted for any reason

Appendix A - Prelab Exercise 3.2 Solutions

Consider the equation: $-3x^2 + 6x - 18 = 0$.

Exercise 1: 6^2 - 4(-3)(-18)
Exercise 2: 36 - (-12)(-18) = 36 - 216 = -180
Exercise 3: Negative
Exercise 4: No real roots
Exercise 5: Two complex roots
Exercise 6: No identical roots
Exercise 7: Did not complete this exercise since all of the roots are complex

Consider the equation: $-3x^2 + 6x + 18 = 0$.

Exercise 1: 6^2 - 4(-3)(18)
Exercise 2: 36 - (-12)(18) = 36 - 216 = 252
Exercise 3: Positive
Exercise 4: Two real roots
Exercise 5: No complex roots
Exercise 6: No identical roots
Exercise 7: The two real roots are -1.6457513111 and 3.6457513111

$$root_1 = \frac{-6 + \sqrt{(252)}}{(2)(-3)} = \frac{-6 + 15.8745078664}{-6} = -1.6457513111 root_2 = \frac{-6 - \sqrt{(252)}}{(2)(-3)} = \frac{-6 - 15.8745078664}{-6} = 3.6457513111$$