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 of you will submit the lab project to Canvas for grading. To ensure you each receive the same score, it is imperative that you follow the instructions in §5 *What to Submit for Grading and by When* (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, including implementing pseudocode.
- Write function definitions, call functions, pass parameters, define local variables, return a value from a function.
- Open a text file for reading/writing and read/write from/to the text file.
- Perform and document testing by writing test cases.
- #include the *cstdlib* header file in order to access the *exit*() function.
- Write if statements, if-else statements, and if-elseif-... statements (your book calls the latter if/else statements).
- Write an if statement to check if an input/output file stream was successfully opened.

3 Prelab Exercises

3.1 Prelab Exercise 1: 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 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: Create Repl.it Project and Add *main.cpp* to the Project

You may use any C++ development environment you wish to complete the project. As we did for Lab Project 5, we will be opening an input file for reading and an output file for writing in this program. If you wish to use an online C++ IDE, we recommend that you use Repl.it.

If you have not already done so, before your lab session, create a free Repl.it account. After doing that, create a new C++ project and rename the project "Lab 6" (Note: if you do not have an account and are logged in as @anonymous, you will not be able to rename the project from the default random project name that was assigned to it when the project was created.) Then upload the *main.cpp* source code file you extracted from the Lab Project 6 zip archive to your Repl.it project, overwriting the *main.cpp* file that was added to the project when you created it.

Before proceeding to the next prelab exercise, edit the header comment block of the provided *main.cpp* file and add your information to the AUTHOR1: comment. If you work with a partner, add the AUTHOR2: comment including his or her information. Remember that only one member of the team has to upload the submission zip archive to Canvas but you will each earn the same score as long as you follow the instructions in this section and in §5.

For help using Repl.it, click the ? button in the lower-right corner of the IDE window. If you require more assistance, ask your lab TA or a UGTA for help in your lab session. Note that when when you are finished with the lab project, you may download your project files in a zip archive: Click the three dots symbol in the title bar of the *File Pane* and select *Download as Zip* from the menu. This will download a zip archive named *Lab-6.zip* (assuming you named your project *Lab 6* as requested).

Note: the downloaded zip archive is **not** the zip archive you are required to upload to Canvas for grading. Please consult §5 for that information.

3.3 Prelab Exercise 3: Placing Testing Input File *coeffs.txt* and Output File *roots.txt* in the Correct Folder

Read §3.2 first. The Lab Project 6 program shall read input from a text file named **coeffs.txt**. A sample coeffs.txt file is provided in the lab project zip archive. You may upload this file to your Repl.it project and use it as Test Case 1 (this file contains the coefficients documented in Test Case 1 of the provided source code file). For you other test cases, simply edit this file and change the values of the coefficients *a*, *b*, and *c*. Now proceed to Prelab Exercise 4.

3.4 Prelab Exercise 4: Study the Software Design

Lab06 is our largest program yet: the provided program template in *main.cpp* contains 637 lines of text but most of those are comment lines, so the actual lines of code (LOC) is more around 150, and I also gave you some of the code, so the actual number of lines of code you must write from scratch is around 100.

Below are the function headers and a brief description of each function. More software design is included in the function header comments for each function in *main.cpp* in the form of pseudocode. Your task is to read and implement the pseudocode.

void calc_complex_root
(double a, double b, double c, double disc, double& real, double &imag)
Calculates the two complex roots of the quadratic equation with coefficients *a*, *b*, and *c*, and discriminant *disc*. Note that we know we have complex roots when the discriminant is < 0. The complex roots are complex conjugates, i.e., the first root is *real* + *imag***i** and the second root is *real* - *imag***i**. Note that *real* and *imag* are output parameters from this function: on entry, any value they contain is not important, but on return, each variable sends a value back to the corresponding argument in the function call in *main*(). This is necessary because in C++ a function can return only one value.

double calc_real_root
(double a, double b, double c, double disc, int operation)
Calculates one of the real roots of the quadratic equation. Parameter *operation* is to be ADD when we are calculating the first root (-b **+** ...) / (2a) and *operation* must be SUB when calculating the second root (-b **-** ...) / (2a).

double discriminant(double a, double b, double c)
Calculates the discriminant of the quadratic formula given the coefficients *a, b,* and *c*. This is the same function we wrote in Lab 4 and is provided for you.

void open_input_file(ifstream& fin, string filename)
Called from *main*() to open the input file for reading whose name is specified by *filename*. Note that in *main*(), the *ifstream* object *fin* was defined and passed as an argument to this function. Upon entry, the input file is not open and upon return back to *main*(), it is. Note that we check to see if the input file was successfully opened and if not, this function does not return but rather calls *terminate*() to display an error message on the console (output window) and end the program by calling *exit*() with an exit code of ERR_OPEN_ INPUT_FILE (which is a global named constant and is equivalent to 1).

void open_output_file(ofstream& fout, string filename)
Called from *main*() to open the output file for writing whose name is specified by *filename*. Note that in *main*(), the *ofstream* object *fout* was defined and passed as an argument to this function. Upon entry, the output file is not open and upon return back to *main*(), it is. Note that we check to see if the output file was successfully opened and if not, this function does not return but rather calls *terminate*() to display an error message on the console (output window) and end the program by calling *exit*() with an exit code of ERR_OPEN_ OUTPUT_FILE (which is a global named constant and is equivalent to 2).

void output_complex_number(ofstream& fout, double real, double imag)
Sends the formatted complex number *real* + *imag***i** or *real* - *imag***i** to the output file stream *fout*.

void output_complex_roots(ofstream& fout, double real, double imag)
Sends the two formatted complex roots of the equation to the output file stream *fout*.

void output_quad_eqn(ofstream& fout, double a, double b, double c)
Sends the formatted quadratic equation to the output file stream *fout*.

void output_real_roots(ofstream& fout, double root1, double root2)A quadratic equation can have one repeated root (when the discriminant is 0) or two real roots (when the discriminant is > 0). This function handles both situations by sending the root(s) to the output file stream *fout.*

double read_coeff(ifstream& fin)
This function was written in Lab Project 4.

void terminate(string message, int exit_code)
This function is called from a few other functions when "something bad" happens and the program has to end. The three situations are: (1) When the input file cannot be opened for reading; (2) When the output file cannot be opened for writing; and (3) When the coefficient of *a* in the input file is 0, because then we do not have a quadratic equation to solve (when *a* = 0 it is the equation of a line). The function sends *message* to *cout* and then calls *exit*() passing *exit_code* as the argument to end the program.

void verify_quad_eqn(ofstream& fout, double a)
Called from *main*() after the coefficients *a*, *b*, and *c* have been read from the input file and the output file *roots.txt* has been opened. If *a* is 0, we do not have a quadratic equation (it is the equation of a line) so this function will call *terminate*() to end the program with an exit code of ERR_NOT_QUADRATIC_EQN (defined as a global named constant and is equivalent to 3).

main() – *main*() primarily just calls the other functions in the program. Read and implement the pseudocode.

Now proceed to the Prelab Exercise 5 in §3.5.

3.5 Prelab Exercise 5: Testing and Writing Test Cases

Open *main.cpp* in your text editor. In the *Testing* section of the header comment block, I have documented one test case, using the coefficients of one of the example equations in §4. For this exercise, you are to design and document **seven additional test cases** which are partially documented for you (see below). These test cases will fairly thoroughly test your code. Before writing the the code, you should write the following parts of all seven cases in the header comment block: (1) Write the coefficients in the *Input Data* section. (2) For the given coefficients *a*, *b*, and *c*, determine exactly what software requirement(s) the test case is testing and document this in the *Description* section. (3) Then, calculate the roots, if any, by using a calculator and document what the exact output from the program should be in the *Expected Output* section. This documentation can be completed before you even begin writing the code, and in fact, by documenting this part of your test cases before writing the code, it will make writing the code easier for you as you will have a better understanding of what the program is supposed to do.

Then, after your program is completed, run it against each of the eight test cases, and copy-and-paste the contents of *roots.txt* into the *Actual Output* section for each test case. Then document whether your test case passed or failed in the *Result* section. Note that the actual output from the program must exactly match the expected output per the output requirements documented in §4 Software Requirements.

This is important: when we employ an automated grading script, for each of the eight test cases, it will run your program using the test case data and will compare the actual output from your program with our expected/correct output. If there is *any* mismatch (including seemingly minor mistakes like missing or extra spaces, periods, newlines, etc), then the grading script will deduct points for a failed test case. Note that we may also use additional test cases other than the eight that are required.

Test Case 1
Description: When *a* is 0, we do not have a quadratic equation; check that the correct error message is sent to the output window and the program terminates.
Input: *a* = 0, *b* = don't care, *c* = don't care
Expected Output: *document what the expected output or behavior of the program should be*
Actual Output: *document what the actual output or behavior of the program was*
Result: *document PASS if the actual output or behavior matched the expected output or behavior, otherwise document* FAIL

Test Case 2
Description: *from the given coefficients, figure out and describe what is being tested*
Input: *a* is 1, *b* is 1, *c* is 1
Expected Output: *document what the expected output or behavior of the program should be*
Actual Output: *document what the actual output or behavior of the program was*
Result: *document PASS if the actual output or behavior matched the expected output or behavior, otherwise document* FAIL

Test Case 3
Description: *from the given coefficients, figure out and describe what is being tested*
Input: *a* is -1, *b* is -1, *c* is -1
Expected Output: *document what the expected output or behavior of the program should be*
Actual Output: *document what the actual output or behavior of the program was*
Result: *document PASS if the actual output or behavior matched the expected output or behavior, otherwise document* FAIL

Test Case 4
Description: *from the given coefficients, figure out and describe what is being tested*
Input: *a* is nonzero, *b* is less than -1, *c* is less than -1
Expected Output: *document what the expected output or behavior of the program should be*
Actual Output: *document what the actual output or behavior of the program was*
Result: *document PASS if the actual output or behavior matched the expected output or behavior, otherwise document* FAIL

Test Case 5
Description: *from the given coefficients, figure out and describe what is being tested*
Input: *a* is nonzero, *b* is greater than -1, *c* is greater than -1
Expected Output: *document what the expected output or behavior of the program should be*
Actual Output: *document what the actual output or behavior of the program was*
Result: *document PASS if the actual output or behavior matched the expected output or behavior, otherwise document* FAIL

Test Case 6
Description:
from the given coefficients, figure out and describe what is being tested
Input: *a* = 11.495, *b* = 33.2345678, *c* = -14.9876543
Expected Output: *document what the expected output or behavior of the program should be*
Actual Output: *document what the actual output or behavior of the program was*
Result: *document PASS if the actual output or behavior matched the expected output or behavior, otherwise document* FAIL

Test Case 7
Description:
from the given coefficients, figure out and describe what is being tested
Input: *a* = 999.99999999, *b* = 1234.567898765, *c* = 413.1234987645
Expected Output: *document what the expected output or behavior of the program should be*
Actual Output: *document what the actual output or behavior of the program was*
Result: *document PASS if the actual output or behavior matched the expected output or behavior, otherwise document* FAIL

4 Lab Exercise: Software Requirements

See §4 Lab Exercise: Software Requirements in the *Lab Project 4 document*. In that program, we calculated the one or two real roots of a quadratic equation. That program had some deficiencies—e.g., it could not handle complex roots—that we could not overcome until we learned about if, if-else, and **if-elseif-...** statements.

Two situations arise in Lab Project 6 that were not addressed in Lab Project 4: (1) If the discriminant is 0, then the equation has only one root, which is repeated. For example, if $a = 5$, $b = 3$, and $c = ?$ such that $b^2 - 4ac = 0$, then $b^2 = 4ac$, and $c = b^2 / (4a)$, and would equal 0.45 when $a = 5$ and $b = 3$. Then our two roots would be calculated as $root_1 = (-b + sqrt(0)) / (2a) = -b / (2a)$ and $root_2 = (-b - sqrt(0)) / (2a) = -b / (2a)$. Thus, $root_1 = root_2$ and the root is repeated. (2) And, if the discriminant is less than 0, then we would attempt to calculate the square root of a negative number which gives us a complex number as the result. In fact, the two complex roots are **complex conjugates** and we would have $root_1 = a + b\mathbf{i}$ and $root_2 = a - b\mathbf{i}$ where a is the **real** part of the imaginary number and b is the **imaginary** part of the number. Note: mathematicians write **i** following the imaginary part and electrical engineers—who are generally not mathematicians (and not very good engineers either, ha, hahaha)—erroneously write **j** following the imaginary part :) Computer scientists tend to lean more toward the mathematician side of the controversy so for Lab 6, we will write **i** for the imaginary part (also, I cannot spell **j**).

This program has essentially the same software requirements as Lab Project 4 with some additions which I will mention as we go along. For the project, we are going to write a program which shall open an input text file named *coeffs.txt* containing the coefficients *a*, *b*, and *c*. The format of *coeffs.txt* shall be,

Contents of coeffs.txt
```
a b c
```

where each of *a*, *b*, and *c* is a real number separated by one or more spaces or tabs; there may or may not be an end of line character following *c* (it does not matter when reading the input). For example, if $a = 122.5$, $b = -6.7$, and $c = 3$, then *coeffs.txt* shall contain,

```
122.5 -6.7 3
```

where there is at least one space or tab character separating the values of *a* and *b* and the values of *b* and *c*. The program shall check to see if *coeffs.txt* was successfully opened for reading; if it was not, then the program shall output an error message to the output window which states,

```
Could not open 'coeffs.txt' for reading.
```

followed by the end of line character. The program shall then terminate with an exit code of 1 (see the description of global named constant ERR_OPEN_INPUT_FILE). Otherwise, the program shall next read *a*, *b*, and *c* from the input file and then close the file. Then, the output will be sent to an output file named *roots.txt* so the program shall open *roots.txt* for writing. It shall check to see if *roots.txt* was successfully opened for writing; if it was not, then the program shall output an error message to the output window which states,

```
Could not open 'roots.txt' for writing.
```

followed by the end of line character. The program shall then terminate with an exit code of 2 (see the description of global named constant ERR_OPEN_OUTPUT_FILE). Otherwise, the program shall check to see if the coefficients *a*, *b*, and *c* form a quadratic equation (by checking to see if *a* is nonzero). If *a* is zero, the program shall output an error message to *roots.txt*,

```
The coefficient for the x^2 term is zero so this is not a quadratic equation.
Please change the value of this coefficient and try again.
Exiting the program now.
```

where each line of text ends with an end of line character, and then the program exits with an exit code of -1. Otherwise, following along with the pseudocode for *main*(), *fout* is configured to output real numbers in fixed notation with five digits after the decimal point and we then output the quadratic equation to *roots.txt* in the format where *a* = 122.5, *b* = -6.7, and *c* = 3.

```
The equation 122.50000x^2 - 6.70000x + 3.00000 = 0
```

Note that the coefficients a, b, c shall be output in usual algebraic notation, i.e., conforming to these examples,

```
// a = 1, b = 1, c = 1
The equation x^2 + x + 1.00000 = 0
// a = -1, b = -1, c = -1
The equation -x^2 - x - 1.00000 = 0
// a = 123.456, b = 0, c = 0
The equation 123.45600x^2 = 0
// a = 123.456, b = 0, c = -567.98012345
The equation 123.45600x^2 - 567.98012 = 0
// a = 123.456, b = -411.911, c = -567.98012345
The equation 123.45600x^2 - 411.91100 - 567.98012 = 0
```

After outputting the equation (and note that the equation at this point is not followed by an end of line character because we have more output on the the same line coming up) we calculate the discriminant *disc*. If *disc* is 0 we have one repeated root, so we calculate that root and output this (when *a* = 5, *b* = 3, and *c* = 0.45),

```
The equation p(x) = 5.00000x^2 + 3.00000x + 0.45000 = 0 has one repeated root = -0.30000
```

Note that the text above shall all be output on one line of *roots.txt*, which shall be followed by an end of line character. If *disc* is negative, as it would be for *a* = 122.5, *b* = -6.7, and *c* = 3, then we have two complex roots and the output to *roots.txt* shall be,

```
The equation p(x) = 122.50000x^2 - 6.70000x + 3.00000 = 0 has two complex roots: root1 = 0.02735 + 0.15408i and root2 = 0.02735 - 0.15408i
```

Note that the text above shall all be output on one line of *roots.txt*, which shall be followed by an end of line character. Note also that the real parts of the roots are the same and the imaginary parts are the same, but in $root_1$ we have an addition operator and in $root_2$ we have a negation operator. Thus, $root_1$ and $root_2$ are complex conjugates.

Finally, if *disc* is positive, we have two real roots and the output shall be (for *a* = 1, *b* = 2, *c* = -3),

```
The equation p(x) = x^2 + 2.00000x - 3.00000 = 0 has two real roots: root1 = 1.00000 and root2 = -3.00000
```

Note that the text above shall all be output on one line of *roots.txt*, which shall be followed by an end of line character. Lastly, *main*() shall close the output file and the program shall return 0 back to the operating system to indicate that it finished successfully.

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. Do not forget to document your test cases 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

- Note: There will be a 40% project penalty deduction (or -2 points) for not following these instructions and uploading incorrect files and/or incorrectly named files.
- Using the file explorer program of your operating system, create a new empty folder named **lab06-*asurite*** where *asurite* is your ASURITE username that you use to log in to MyASU (e.g., mine is *kburger2* so my folder would be named lab06-kburger2). If you worked with a partner on the project, put both of your ASURITE usernames in the name of the new folder **lab06-*asurite1*-*asurite2***, e.g., Fred Flintstone's and Barney Rubble's project submission archive would be named **lab06-fflint-brubble.zip**.
- Copy your completed *main.cpp* source code file to the folder your just created. This folder should contain only one file named *main.cpp*. Do **not** copy the input file *coeffs.txt* or the output file *roots.txt* to this folder. We do not need the data files because we will use our own when testing your program.
- Compress the folder creating a **zip archive** named **lab06-*asurite*.zip** or **lab06-*asurite1*-*asurite2*.zip**.
- Submit the zip archive to Canvas by the deadline using the *Lab Project 6* submission link.
- If you worked with a partner, then only one partner of the team submits to Canvas, but you will each receive the same score as long as you follow the rules for documenting both authors in the header comment block and properly naming your zip archive.
- If your program does not compile or run correctly, upload what you have completed for grading anyway because you will generally receive some partial credit for effort.
- The submission deadline is **11:59pm Sat 20 Mar**, with a 48 hour late submission deadline of **11:59pm Mon 22 Mar**.
- The Canvas submission link will become unavailable exactly at 11:59:00pm on Mon 22 Mar and you will not be able to submit your project for grading after that time. We do not accept emailed submissions.
- Consult the online syllabus for the late project submission and academic integrity policies.

6 Grading Rubric

doc	*build*	*partial*	*passed*	*points*	*remarks*
true	true	n/a	7	5.0	Documented all test cases, built, ran, and passed all test cases
false	true	n/a	7	4.5	Did not doc all test cases, built, ran, and passed all test cases
true	true	n/a	1-6	4.0	Documented all test cases, built, ran, and passed some but not all test cases
false	true	n/a	1-6	3.5	Did not doc all test cases, built, ran, and passed some but not all test cases
true	true	n/a	0	3.0	Documented all test cases, built, ran, and failed all test cases
false	true	n/a	0	2.5	Did not doc all test cases, built, ran, and failed all test cases
true	false	true	n/a	2.0	Documented all test cases, syntax errors, good effort writing most of the code
false	false	true	n/a	1.5	Did not doc all test cases, syntax errors, good effort writing most of the code
true	false	false	n/a	1.0	Documented all test cases, syntax errors, poor effort writing little of the code
false	false	false	n/a	0.5	Did not doc all test cases, syntax errors, poor effort writing little of the code
false	false	false	n/a	0.0	No submission or simply submitted the given template file no modifications

Deadline was 11:59pm Sat 20 Mar.

a. Deduct 0.5 pt (-10%) for a submission on Sun 21 Mar.
b. Deduct 1 pt (-20%) for a submission on Mon 22 Mar.
c. After Mon 22 Mar, late projects are not accepted for any reason.