Lab 2

Due Jan 23 by 11:59pm **Points** 25 **Submitting** a file upload **File Types** cpp

If-Statements And Simple Loops (Chap 3)

The **General Programming Lab Instructions** apply to this assignment.

Lab 2 consists of two C++ programs that are scored using Microsoft Visual Studio. Take note of the scoring rubric at the end of the assignment.

Program 1

If statements are simple but essential control statements. Begin this lab by creating a new solution and/or project and copying and pasting the code for lab 1, program 1 (`roots.cpp`) into the new project. Name the new source code file `roots2.cpp.`

Reasoning Behind the Requirements

I. The quadratic formula:

$$x1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

and

$$x2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

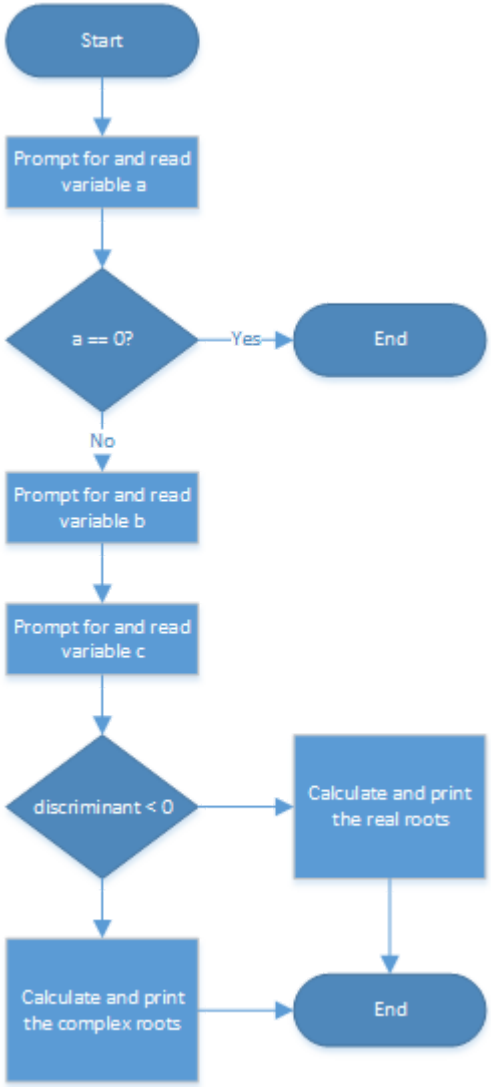
fails if coefficient `a` is 0; the other coefficients (`b` and `c`) may be 0 and `a` may be less than 0 without causing harm.

II. The discriminant, `b² - 4ac`, determines the kinds of roots that a quadratic equation has
   i. if the discriminant is greater than or equal to 0, the equation has two real roots (which may be equal)
   ii. else when the discriminant is less than 0, the equations has two complex roots

III. Complex numbers are displayed as two separate numbers (a real part and an imaginary part) with character decorations added to help separate and distinguish the two numbers. For example, `6 + 3i` and `6 - 3i`
   i. The '+', '-', and 'i' are just **character constants** (http://icarus.cs.weber.edu/~dab/cs1410/textbook/1.Basics/data.html#constant)
   ii. The real part (6 in the example) is calculated by the formula `-b / 2a`
   iii. The imaginary part (`3i` in the example) is calculated by the formula `sqrt(-discriminant) / 2a` (notice the use of the **unary minus** (http://icarus.cs.weber.edu/~dab/cs1410/textbook/2.Core/operators.html) operator)

Program Requirements (Changes from Lab 1)

1. Name the program `roots2.cpp`
2. If the user enters a 0 for `a`, print an appropriate error message and end the program (skip all calculations, do not loop). You may use `exit(0);` (the **exit** (http://icarus.cs.weber.edu/~dab/cs1410/textbook/2.Core/termination.html) function is described here) if you want but are not required to do so. Do not prompt the user to enter a different value for `a`
3. Your program must find all the roots without crashing
   a. Calculate the discriminant: `b² - 4ac`
   b. If the discriminant => 0, the code written in lab 1 still works
   c. else (the discriminant is < 0), add code to roots2 to calculate the two complex roots:
      i. `real = -b / 2a`
      ii. `imag = sqrt(-discriminant) / 2a`
   d. Print both roots and end the program
4. Do not include extraneous prompts, loops, reads, or pause statements at the end of the program
5. You may not use any functions from the stdio library



Test Case:

a = 2, b = 2, c = 2; `x1 = -0.5 + 0.866025i` and `x2 = -0.5 - 0.866025i`

The test cases from lab 1 must still work and display as before

---

Program 2

For program 2 you will write a simple number guessing game. The game will generate a random number in the range of 0 - 99. It will prompt the player to enter a guess and then tell the player if his/her guess was high or low. The program will loop, prompting, reading input, and providing feedback to the player. This operation allows the player to "zero in" and ultimately guess the game's target number.

Program 1 Requirements and Pseudo Code

A. Create a number guessing game in a source code file named `guess.cpp`
B. You may not use any functions from the stdio library
C. The rest of the requirements are expressed in the pseudo code that also describes the game's behavior.

1. seed the random number generator (details given below)
2. generate a target number between 0 and 99 (details given below)
3. loop (e.g., `while (true)`) - statements a-g are inside the loop body
   a. prompt the user to guess a number from 0 to 99 (0 and 99 are allowed)
   b. read the guess
   c. if the guess is equal to -1 or less than 0 (your choice), terminate the program (this is a way to end the program early)
   d. if the guess and the target number are equal, print a success message (e.g., "Right" or "You Win" or similar) and terminate the program
   e. if the guess is less than the target number, print "Low" or a similar message
   f. if the guess is greater than the target number, print "High" or a similar message
   g. continue with next iteration of the loop (go back to step 3.a - end the loop body)

Generating Pseudo Random Numbers

A pseudo random number is a number that "looks" random - i.e., passes certain statistical tests for randomness - but which is generated deterministically or non-randomly. The standard C/C++ library provides a function named `rand` that will generate pseudo random numbers, one number per call. The sequence of numbers produced by a pseudo random number generator eventually repeats; rand has cycle length of 0 to 2^{16}-1. A seed number determines where in the cycle the generator starts. A second function named `srand` seeds or initializes the random number generator. It is common to use the current system time, a value that is constantly changing, to seed the generator. Doing this insures that the (relatively) short sequences of the pseudo random numbers used by a program are always different (how much fun would the game be if it always generated the same target number?). The following code fragment demonstrates how to seed the generator and how to generate a pseudo random number between 0 and 99:

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;
                        . . .
int main()
{
        srand((unsigned)time(nullptr)); // seed the generator
        int     target = rand() % 100;  // numbers [0..99]
                        .
                        .
                        .
}
```

Program Submission and Grading

- The programs should exhibit good programming style.
- Upload `roots2.cpp` (notice the 2) and `guess.cpp` to Canvas for grading.
- Be sure the program files are named correctly and that they do not include unspecified prompts, pauses, dummy reads, or menus. Make sure that the uploaded files are the correct files (i.e., verify that the file contents are correct).

Lab 2 Scoring

Criteria	Ratings	Pts
roots2.cpp The file name is correct (Canvas may add a "-1" or "-2" to the name, which is okay)		2.0 pts
roots2: Output Correct output in all cases.		8.0 pts
roots2: unary minus used when discriminant is < 0 rather than -1*discriminant.		1.0 pts
roots2: Test for a == 0 The program tests for the error condition a == 0. When found, the program prints an appropriate error message and terminates without producing a runtime error.		2.0 pts
guess.cpp The file name is correct (Canvas may add a "-1" or "-2" to the name, which is okay)		2.0 pts
guess: Early End Program ends on a "guess" of -1 or <0. Output is optional but must be appropriate if included; for example, don't say something like "Illegal entry" because the instructions state that a -1 is a valid way to terminate the program.		2.0 pts
guess: guess == target Behavior and output must be appropriate.		2.0 pts
guess: guess > target Behavior and output must be appropriate.		2.0 pts
guess: guess < target Behavior and output must be appropriate.		1.0 pts
Header files Programs only include the needed header files (stdfx.h and pch.h are removed or commented out).		2.0 pts

Criteria	Ratings	Pts
Miscellaneous The programs are clean and easy to read: variable names are appropriate, indentation is consistent, blank lines are used appropriately; both programs are created "Empty," and all "pause" statements and/or dummy reads are commented out or removed, etc.		1.0 pts
		Total Points: 25.0