CIS 200 - Lab 0301

1. Problem Statement

Create a program that contains a function to calculate the square root of a number.

2. Requirements

2.1 Assumptions

- Written in C++
- Only use command line input/output
- User will input real numbers only
- User will input only ‘y’ or ‘n’ when asked to repeat

2.2 Specifications

- The program will prompt the user to input a real number
    - For valid real number input
        - Calculate the square root
    - For invalid input
        - If input is negative
        - Terminate program

3. Decomposition Diagram

- Program
    - Input
        - User inputs real number via command line
        - User inputs ‘y’ or ‘n’ character
    - Process
        - Determine if input is valid
        - Use series approximation to calculate *sqrt*(*x*)
    - Output
        - Print result of square root operation
        - Print error for invalid data

4. Test Strategy

- Valid Data
- Invalid Data

5. Test Plan Version 1

Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass/Fail
Valid Data	1	*squareRoot*() at minimum possible				
Valid Data	2	*squareRoot*() above minimum				
Valid Data	3	*squareRoot*() higher order				
Valid Data	4	*squareRoot*() EVEN HIGHER				
Invalid Data	1	*squareRoot*() below minimum				

6. Initial Algorithm

```
1. Create Square Root Function squareRoot()
    a. Parameter
        i. Double Input x
    b. Method
        i. Assert that input is greater than 0
        ii. Calculate Square root via series approximation
            1. Set $X_n$ equal to $\frac{x}{2}$
            2. Evaluate such that $x_{n+1} = (x_n + \frac{x}{x_n}) / 2$
            3. Keep evaluating until $x_n - x_{n+1} < 0.0001$
        iii. Return Root Value
2. Create main()
    a. Prompt user to input double to use
        i. Run squareRoot()
    b. Prompt user if they would like to calculate again
        i. If input is ‘y’
            1. Run again
        ii. If input is ‘n’
            1. Exit
```

7. Test Plan Version 2

Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass/Fail
Valid Data	1	*squareRoot*() at minimum possible	0	0		
Valid Data	2	*squareRoot*() above minimum	1	1		
Valid Data	3	*squareRoot*() higher order	3	~*sqrt*(3)		
Valid Data	4	*squareRoot*() EVEN HIGHER	16	4		
Invalid Data	1	*squareRoot*() below minimum	-3	*assert*()		

8. Code

```
//Program Name: Square Root Calculator
//Programmer Name: Arthur Aigeltinger IV
//Description: Use series approximation to calculate the square root.
//Date Created: 10/03/18


#include <assert.h>
#include <cctype>
#include <iostream>

//Function Prototype
double squareRoot(double);

int main()
{
	//Initialize Variables
	double userIn = 0;
	char userChoice = 'y';

	do
	{
		//Default Choice
		userChoice = 'y';

		//Prompt
		std::cout << "Please insert a number to find the root of: ";
		std::cin >> userIn;
		std::cout << "Sqare root is " << squareRoot(userIn) << "!" << std::endl
<< std::endl;

		do
		{
			//Prompt
			std::cout << "Would you like to calculate another? (y)es/(n)o?"
<< std::endl;
			std::cin >> userChoice;
			userChoice = std::tolower(userChoice);

		} while (!(userChoice == 'y' || userChoice == 'n'));

	} while (!(userChoice == 'n'));

	system("pause");
	return 0;
}

//Description: Function that will take in a double and return either a valid output,
or
//Pre-condition: Having input that is valid to input.
//Post-Condition: Either an assertion or a valid square root.
double squareRoot(double x)
{
	assert(x >= 0);

	if (x == 0)
	{
		return 0;
	}

	double xN = x / 2;
	double xNp1 = (xN + x / xN) / 2;

	while (abs(xN - xNp1) > 0.0001) //Absolute value to account for values that
drop below 0
	{
		xN = xNp1;
		xNp1 = (xN + x / xN) / 2;
	}

	return xNp1;
}
```

9. Updated Algorithm

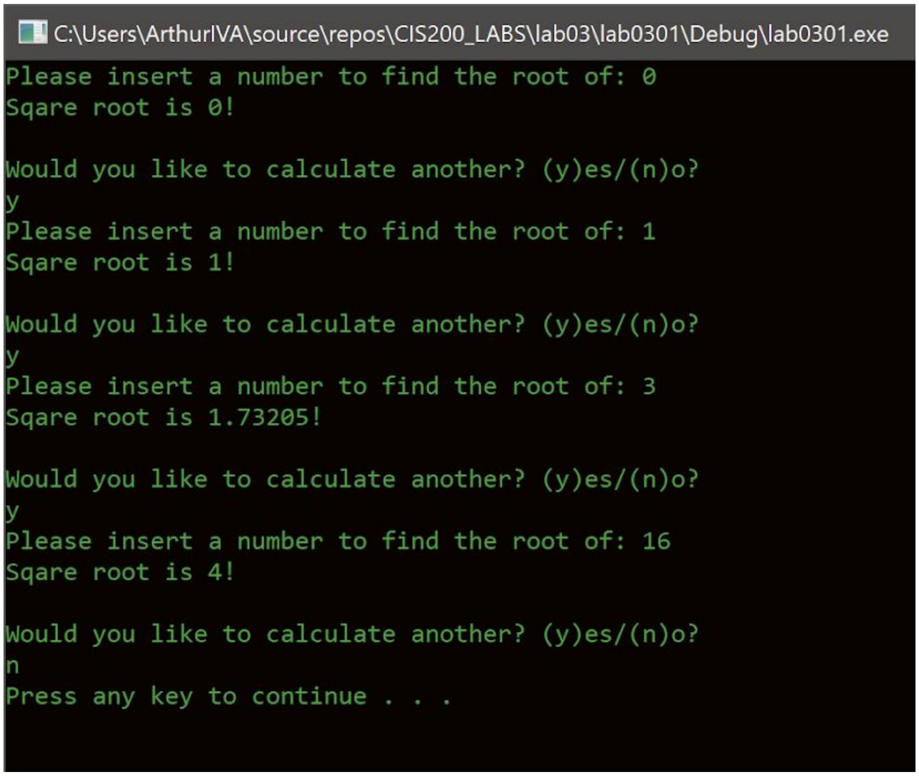
1. Create Square Root Function *squareRoot*()
    a. Parameter
        i. Double Input *x*
    b. Method
        i. Assert that input is greater than 0
        ii. Calculate Square root via series approximation
            1. Set X_n equal to $\frac{x}{2}$
            2. Evaluate such that $x_{n+1} = (x_n + \frac{x}{x_n}) / 2$
            3. Use absolute value to keep values in range
            4. Keep evaluating until $x_n - x_{n+1} < 0.0001$
        iii. Return Root Value
2. Create *main*()
    a. Prompt user to input double to use
        i. Run *squareRoot*()
    b. Prompt user if they would like to calculate again
    c. Treated as inner loop to that response only occurs with ‘y’ and ‘n’
        i. If input is ‘y’
            1. Run again
        ii. If input is ‘n’
            1. Exit

10. Test Plan Version 3

Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass/Fail
Valid Data	1	*squareRoot*() at minimum possible	0	0	0	Pass
Valid Data	2	*squareRoot*() above minimum	1	1	1	Pass
Valid Data	3	*squareRoot*() higher order	3	~*sqrt*(3)	1.73205	Pass
Valid Data	4	*squareRoot*() EVEN HIGHER	16	4	4	Pass
Valid Data	5	Exit Program	‘n’	Close	Close	Pass
Invalid Data	1	*squareRoot*() below minimum	-3	*assert*()	*assert*()	

11. Screenshots

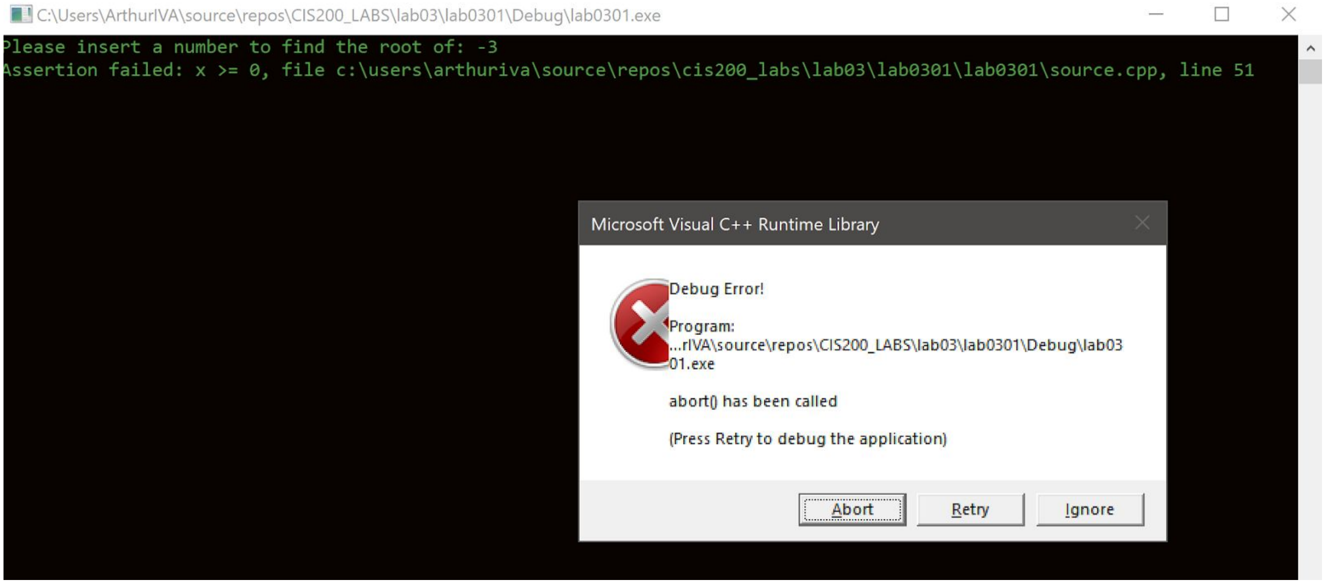
Valid Test Cases 1, 2, 3, 4, 5



C:\Users\ArthurIVA\source\repos\CIS200_LABS\lab03\lab0301\Debug\lab0301.exe

```
Please insert a number to find the root of: 0
Sqare root is 0!

Would you like to calculate another? (y)es/(n)o?
y
Please insert a number to find the root of: 1
Sqare root is 1!

Would you like to calculate another? (y)es/(n)o?
y
Please insert a number to find the root of: 3
Sqare root is 1.73205!

Would you like to calculate another? (y)es/(n)o?
y
Please insert a number to find the root of: 16
Sqare root is 4!

Would you like to calculate another? (y)es/(n)o?
n
Press any key to continue . . .
```

Invalid Test Case 1



C:\Users\ArthurIVA\source\repos\CIS200_LABS\lab03\lab0301\Debug\lab0301.exe

```
Please insert a number to find the root of: -3
Assertion failed: x >= 0, file c:\users\arthuriva\source\repos\cis200_labs\lab03\lab0301\lab0301\source.cpp, line 51
```

Microsoft Visual C++ Runtime Library

Debug Error!

Program:
...rIVA\source\repos\CIS200_LABS\lab03\lab0301\Debug\lab0301.exe

abort() has been called

(Press Retry to debug the application)

12. Error Log

Error Type (Logic/Runtime)	Cause of Error	Solution to Error
Logic	Lack of absolute value in calculating the square root.	Include absolute value such that values do not dip below zero.

13. Status

The program is fully operational in current form.