1. **Problem Statement**

Program calculates the square root of a given float number with assertions and constraints placed.

1. **Requirements**
   1. **Assumptions**
      1. To continue to enter values or not, user types “y” for yes and “n” for no
      2. User does not enter 0 as it will be an undefined number
   2. **Specifications**
      1. Add function call to assert() at the beginning of the square root function
      2. Loop until user says n for no
      3. Write an assert() statement with whose assertion failed, the name of the source file, and the line number where it happened.
2. **Decomposition Diagram**

|  |  |  |
| --- | --- | --- |
| **Main** | | |
| **Input** | **Process** | **Output** |
| Float number to be user to calculate its square root | Calculate square root of the float number |  |
|  |  | Prompt user to enter more values or quit |
|  | Assert and stop executing if the input variable x is negative | Assertion errors |
|  |  | Thank You Message |
|  |  | Welcome Message |

1. **Test Strategy**
   1. Valid Data
   2. Invalid Data
2. **Test Plan Version 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | User types in y for yes and n for no |  |  |  |  |
| Valid | 2 | User types in negative number |  |  |  |  |
| Valid | 3 | User types in an int instead of double |  |  |  |  |

1. **Initial Algorithm**
   1. In main function
      1. Initialize variables to store data in
      2. Add loop and loop until user types in n and wants to quit
         1. Prompt user for the number
         2. Call function *squareRoot* and pass in the given number
      3. Ask the user “Continue to enter values? ‘y’ or ‘n’: ”
   2. In *squareRoot* function
      1. Add assertion statement here if the number is negative, so if it fails, the code does not go forward.
      2. Initialize a variable to store current number.
      3. Initialize a variable to store the next number
      4. Add loop and loop until the difference between current number and the next number is greater than or equal to 0.0001
         1. Assign one-half of input number to variable for current number
         2. Using the formult ***(xn + x/xn)/2***with xn to be the current number and ***x*** to be the user input*,* assign it to the variable for next number.
2. **Test Plan Version 2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | User types in y for yes and n for no | y | Loops until user types n to quit |  |  |
| Valid | 2 | User types in negative number | -3 | Assertion fails. Gives the exact line number, file, and assertion type |  |  |
| Valid | 3 | User types in an int instead of double | 5 | Program converts it and uses it as double |  |  |

1. **Code**

#include <iostream>

#include <string>

#include <assert.h>

#include <cmath>

using namespace std;

float squareRoot(float x);

int main()

{

float userInput;

string quit;

float output;

do {

cout << "Give me a number to find its square root: " << endl;

cin >> userInput;

output = squareRoot(userInput);

cout << "Square root of " << userInput << " is " << output << endl;

do {

cout << "Continue to enter values?" << endl;

cin >> quit;

if (quit != "y" && quit != "n")

{

cout << "Invalid Input. Please try again." << endl;

}

} while (quit != "y" && quit != "n");

} while (quit == "y");

system("pause");

return 0;

}

float squareRoot(float x)

{

assert(x >= 0);

float x\_n, x\_n1 = 0, temp;

x\_n = x / 2;

x\_n1 = (x\_n + x / x\_n) / 2;

temp = x\_n1;

while (fabs(x\_n1 - x\_n) > 0.0001) {

x\_n1 = (temp + x / temp) / 2;

x\_n = temp;

temp = x\_n1;

}

return x\_n1;

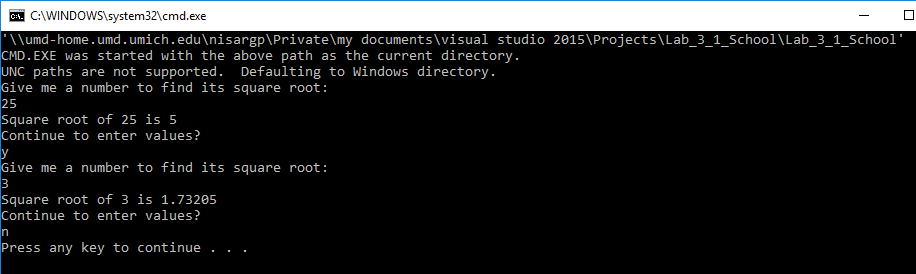
}

1. **Updated Algorithm**
   1. In main function
      1. Initialize variables to store data in
      2. Add loop and loop until user types in n and wants to quit
         1. Prompt user for the number
         2. Call function *squareRoot* and pass in the given number
      3. Ask the user “Continue to enter values? ‘y’ or ‘n’: ”
         1. If invalid input, loop until user enter valid input
   2. In *squareRoot* function
      1. Add assertion statement here if the number is negative, so if it fails, the code does not go forward.
      2. Initialize a variable to store current number.
      3. Initialize a variable to store the next number
      4. Assign one-half of input number to variable for current number
      5. Using the formult ***(xn + x/xn)/2***with xn to be the current number and ***x*** to be the user input*,* assign it to the variable for next number.
      6. Initialize a temporary variable to be the calculated next number.
      7. Add loop and loop until the difference between current number and the next number is greater than or equal to 0.0001
         1. ~~Assign one-half of input number to variable for current number~~
         2. Using the formult ***(xn + x/xn)/2***with xn to be the current number and ***x*** to be the user input*,* assign it to the variable for next number.
         3. Assign the temporary variable to current variable
         4. Assign the calculated next number to temporary variable
2. **Test Plan Version 3**

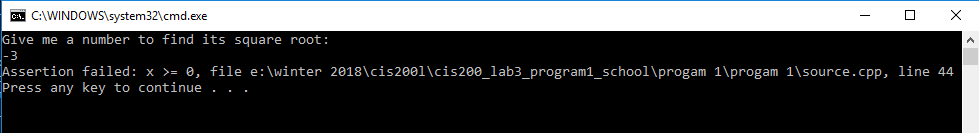
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | User types in y for yes and n for no | y | Loops until user types n to quit | Loops until user types n to quit | Pass |
| Invalid | 1 | User types in negative number | -3 | Assertion fails. Gives the exact line number, file, and assertion type | Assertion fails. Gives the exact line number, file, and assertion type | Pass |
| Valid | 2 | User types in an int instead of double | 5 | Program converts it and uses it as double | Program converts it and uses it as double | Pass |
| Valid | 3 | User types a positive number | 3 | 1.73205 | 1.73205 | Pass |
| Invalid | 2 | User types an undefined number | 0 | Assertion is called. Gives the exact line number, file, and assertion type | Program outputs –nan(ind) saying it is undefined | Fail |

1. **Screenshots**

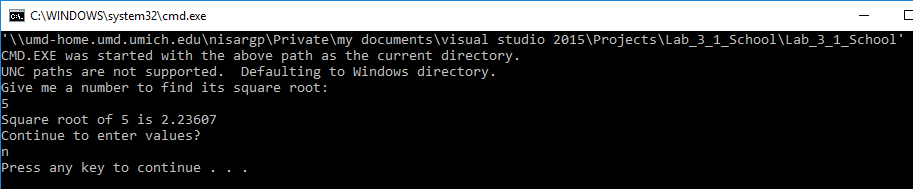
Valid Test Case 1 (Windows):



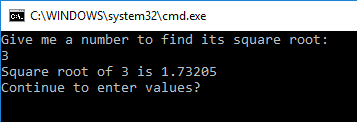
Invalid Test Case 1 (Windows):



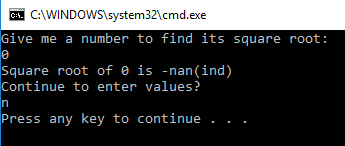
Valid Test Case 2 (Windows):



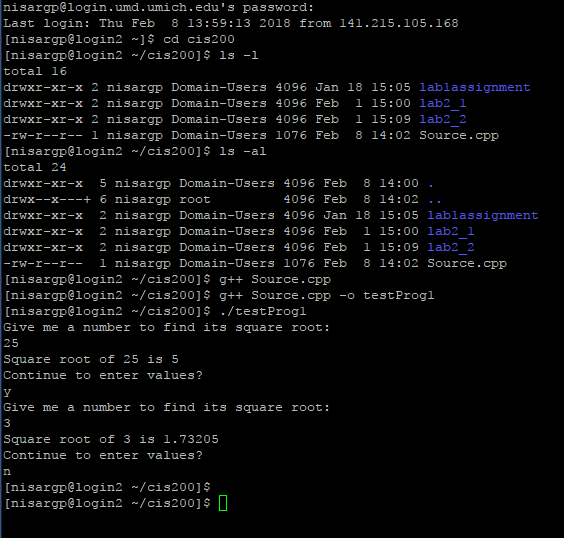
Valid Test Case 3 (Windows):



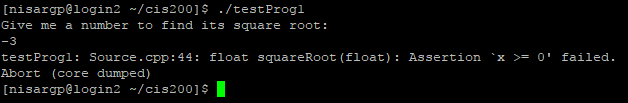
Invalid Test Case 2 (Windows):



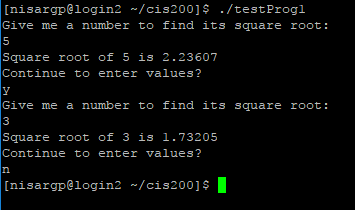
Valid Test Case 1 (Linux):



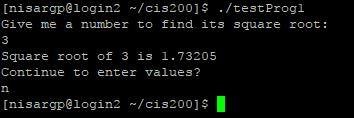
Invalid Test Case 1 (Linux):



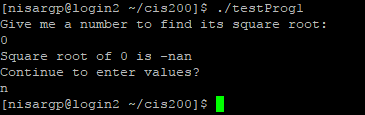
Valid Test Case 2 (Linux):



Valid Test Case 3 (Linux):



Invalid Test Case 2 (Linux):



1. **Status**

Program works perfectly whenever 0 is not entered. When it is entered, program gives the output as –nan(ind)