SAIGON UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

Software Testing

HO CHI MINH CITY, SEPTEMBER 2025

**NAME: DANG NGUYEN TAM NHU**

**STUDENT CODE: 3122411142**

**CLASS: DCT122C5**

**LAP 1 TEST CASES**

A picture containing text, clipart

Description automatically generated

**I. VERIFICATION AND VALIDATION**

**Description:** The purpose is to help users to solve a 2-degree equation (ax

2+bx+c).

**Spec:** Given input of a, b, and c; the system returns the outputs of x1 and x2 (extreme cases are temporarily not

considered)

Two systems are developed as follows:

Ảnh có chứa văn bản, ảnh chụp màn hình, biểu đồ, Phông chữ

Nội dung do AI tạo ra có thể không chính xác.

**System 1 problems**

1. DELTA is not defined in the code

* The program doesn’t calculate ***DELTA = b\*b – 4\*a\*c*** before using it

1. Wrong fomula implementation:

The formula is written incorrectly as follows: x2 = (-b - sqrt(DELTA))/2a

The exact formula is as follows: ***x2 = (-b - sqrt(DELTA))/2a***

* This changes the order of operations, effectively computing:

x2 =

instead of the correct:

***x2 =***

**Common problem of both systems**

Division formula is wrong in code:

x1 = (-b + sqrt(DELTA))/2a

x2 = (-b - sqrt(DELTA))/2a

* In most programming languages, 2a is invalid syntax, it must be (2\*a).

Otherwise, the program won’t run

**II. TEST-CASES**

Description: Some input values

a) How many test-cases we need for the following function f1. What are they?

int f1(int x) {

if (x > 10)

return 2 \* x;

else

return -x;

}

* Identify branches
* Branch 1: x > 10 → return 2\*x
* Branch 2: x <= 10 → return -x
* Choose test-cases
* Branch 1 (True): x = 11 → f1(11) = 22
* Branch 2 (False): x = 10 → f1(10) = -10
* Conclusion: 2 test-cases cover all branches

b) Check if your test-cases can detect error if f1 is implemented as follows

int f1(int x) {

if (x > 10)

return 2 \* x;

else if (x > 0)

return -x;

else

return 2 \* x;

}

In this case, how many test-cases we need to test this function? What are they?

* Identify branches
* Branch 1: x > 10 → 2\*x
* Branch 2: 0 < x <= 10 → -x
* Branch 3: x <= 0 → 2\*x
* Choose test-cases
* Branch 1: x = 11 → 22
* Branch 2: x = 5 → -5
* Branch 3: x = -3 → -6
* Conclusion: 3 test-cases to cover all branches and detect the error (especially branch 3)

c) How many test-cases we need to test this function? What are they?

int f2(int x) {

if (x < 10)

return 2 \* x;

else if (x < 2)

return -x;

else

return 2 \* x;

}

In this case, how many test-cases we need to test this function? What are they?

* Analyze branches
* if(x<10): True if x < 10
* else if(x<2): never True because x<10 is handled first → unreachable
* else: x >= 10
* Test-cases
* True branch: x = 5 → 10
* Else branch: x = 10 → 20
* Conclusion: 2 test-cases, unreachable branch ignored

d) How many test-cases we need to test this function? What are they?

int f3(int x) {

if (log(x \* x \* cos(x)) < 3 \* x)

return 2 \* x;

else

return 2 \* x;

}

* Analyze branches
* True branch and False branch return same result
* For branch coverage:
* True: x = 1
* False: x = -1
* Conclusion: 2 test-cases cover the condition

e) Check if your test-cases can detect error if findMax is implemented as follows

int findMax(int num1, int num2, int num3) {

int max = 0;

if ((num1 > num2) && (num1 > num3))

max = num1;

if ((num2 > num1) && (num2 > num3))

max = num2;

if ((num3 > num1) && (num3 > num2))

max = num3;

return max;

}

In this case, how many test-cases we need to test this function? What are they?

* Analyze branches
* Error: using > ignores equality
* Max could return 0 if two or three numbers are equal
* Test-cases to detect error
* Different numbers, max in the middle: (3,5,2) → 5
* Max first: (7,5,2) → 7
* Max last: (1,2,5) → 5
* Two numbers equal max: (5,5,2) → 5 (code returns 0)
* All equal: (4,4,4) → 4 (code returns 0)
* Conclusion: 5 test-cases detect all faults.

**III. PRATICE 1**

* Describe the problem, including all possible inputs and outputs.
* Create test cases to verify the correctness of the program.
* Write code to automatically check whether the program produces correct results.

#include <iostream>

#include <cmath>

using namespace std;

int solveQuartic(double a, double b, double c, double x[]) {

if (a == 0 && b == 0 && c == 0) {

return -1;

}

if (a == 0 && b == 0) {

return 0;

}

if (a == 0) {

double y = -c / b;

if (y < 0) return 0;

x[0] = sqrt(y);

x[1] = -sqrt(y);

return 2;

}

double delta = b \* b - 4 \* a \* c;

if (delta < 0) return 0;

double y1 = (-b + sqrt(delta)) / (2 \* a);

double y2 = (-b - sqrt(delta)) / (2 \* a);

int count = 0;

if (y1 >= 0) {

x[count++] = sqrt(y1);

x[count++] = -sqrt(y1);

}

if (y2 >= 0 && y2 != y1) {

x[count++] = sqrt(y2);

x[count++] = -sqrt(y2);

}

return count;

}

int main() {

double a, b, c;

cin >> a >> b >> c;

double x[4];

int n = solveQuartic(a, b, c, x);

if (n == -1) {

cout << " Infinite solutions." << endl;

} else if (n == 0) {

cout << "No solution." << endl;

} else {

cout << " The equation has " << n << " real solution(s): ";

for (int i = 0; i < n; i++) {

cout << x[i] << " ";

}

cout << endl;

}

return

0

;

}

* Problem description
* Input: a, b, c
* Output: number of real solutions n and the solutions x[]
* Special cases:
* a=b=c=0: Infinite solutions
* a=b=0, c≠0: No solution
* a=0, b≠0: Quadratic: b\*x^2 + c = 0
* a≠0: Standard quartic equation
* Test-cases:

|  |  |  |  |
| --- | --- | --- | --- |
| a | b | c | Result |
| 0 | 0 | 0 | Infinite solutions |
| 0 | 0 | 5 | No solution |
| 0 | 2 | -8 | x=2,-2 |
| 1 | -3 | 2 | [√2, -√2, 1.0, -1.0] ~ [1.4142, -1.4142, 1.0, -1.0] |

**Python code:**

import unittest

import math

# a. Function f1

def f1(x):

    return 2\*x if x > 10 else -x

# b. Function f1 wrong

def f1\_wrong(x):

    if x > 10:

        return 2\*x

    elif x > 0:

        return -x

    else:

        return 2\*x

# c. Function f2

def f2(x):

    if x < 10:

        return 2\*x

    elif x < 2:

        return -x

    else:

        return 2\*x

# d. Function f3

def f3(x):

    return 2\*x

# e. Function findMax

def findMax(num1, num2, num3):

    max\_val = 0

    if num1 > num2 and num1 > num3:

        max\_val = num1

    if num2 > num1 and num2 > num3:

        max\_val = num2

    if num3 > num1 and num3 > num2:

        max\_val = num3

    return max\_val

# Quartic solver

def solveQuartic(a,b,c):

    x = []

    if a==0 and b==0 and c==0:

        return -1, x

    if a==0 and b==0:

        return 0, x

    if a==0:

        y = -c/b

        if y < 0:

            return 0, x

        x.append(math.sqrt(y))

        x.append(-math.sqrt(y))

        return 2, x

    delta = b\*b - 4\*a\*c

    if delta < 0:

        return 0, x

    y1 = (-b + math.sqrt(delta)) / (2\*a)

    y2 = (-b - math.sqrt(delta)) / (2\*a)

    if y1 >= 0:

        x.append(math.sqrt(y1))

        x.append(-math.sqrt(y1))

    if y2 >= 0 and y2 != y1:

        x.append(math.sqrt(y2))

        x.append(-math.sqrt(y2))

    return len(x), x

# Unit test

class TestFunctions(unittest.TestCase):

    # f1

    def test\_f1(self):

        self.assertEqual(f1(11), 22)

        self.assertEqual(f1(5), -5)

        self.assertEqual(f1(-3), 3)

    def test\_f1\_wrong(self):

        self.assertEqual(f1\_wrong(11), 22)

        self.assertEqual(f1\_wrong(5), -5)

        # Wrong code still testing the error branch

        self.assertEqual(f1\_wrong(-3), 3)

    # f2

    def test\_f2(self):

        self.assertEqual(f2(5), 10)

        self.assertEqual(f2(15), 30)

        self.assertEqual(f2(1), 2)

    # f3

    def test\_f3(self):

        self.assertEqual(f3(1), 2)

        self.assertEqual(f3(10), 20)

    # findMax

    def test\_findMax(self):

        self.assertEqual(findMax(5,3,2), 5)

        self.assertEqual(findMax(3,6,1), 6)

        self.assertEqual(findMax(1,4,7), 7)

        # Wrong code will FAIL

        self.assertEqual(findMax(2,2,2), 2)

        self.assertEqual(findMax(-1,-2,-3), -1)

    # Quartic solve

    def test\_solveQuartic(self):

        n, x = solveQuartic(0,0,0)

        self.assertEqual(n, -1)

        n, x = solveQuartic(0,0,5)

        self.assertEqual(n, 0)

        n, x = solveQuartic(0,2,-8)

        self.assertEqual(n, 2)

        self.assertAlmostEqual(x[0], 2.0, places=6)

        self.assertAlmostEqual(x[1], -2.0, places=6)

        n, x = solveQuartic(1,-3,2)

        self.assertEqual(n, 4)

        expected = [math.sqrt(2), -math.sqrt(2), 1.0, -1.0]

        for a, b in zip(x, expected):

            self.assertAlmostEqual(a, b, places=6)

# main

if \_\_name\_\_ == "\_\_main\_\_":

    unittest.main(argv=[''], verbosity=2, exit=False)

**Test results:**

**Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, tài liệu

Nội dung do AI tạo ra có thể không chính xác.**

**C++ code:**

#include <gtest/gtest.h>

#include <cmath>

#include <vector>

using namespace std;

// a. Function f1

int f1(int x) {

if (x > 10)

return 2 \* x;

else

return -x;

}

// b. Function f1 wrong

int f1\_wrong(int x) {

if (x > 10)

return 2 \* x;

else if (x > 0)

return -x;

else

return 2 \* x;

}

// c. Function f2

int f2(int x) {

if (x < 10)

return 2 \* x;

else if (x < 2)

return -x;

else

return 2 \* x;

}

// d. Function f3

int f3(int x) {

return 2 \* x;

}

// e. Function findMax

int findMax(int num1, int num2, int num3) {

int max = 0;

if ((num1 > num2) && (num1 > num3)) max = num1;

if ((num2 > num1) && (num2 > num3)) max = num2;

if ((num3 > num1) && (num3 > num2)) max = num3;

return max;

}

// Quartic solver

int solveQuartic(double a, double b, double c, vector<double>& x) {

x.clear();

if (a == 0 && b == 0 && c == 0)

return -1; // Infinite solutions

if (a == 0 && b == 0)

return 0; // No solution

if (a == 0) {

double y = -c / b;

if (y < 0) return 0;

x.push\_back(sqrt(y));

x.push\_back(-sqrt(y));

return 2;

}

double delta = b \* b - 4 \* a \* c;

if (delta < 0) return 0;

double y1 = (-b + sqrt(delta)) / (2 \* a);

double y2 = (-b - sqrt(delta)) / (2 \* a);

if (y1 >= 0) {

x.push\_back(sqrt(y1));

x.push\_back(-sqrt(y1));

}

if (y2 >= 0 && y2 != y1) {

x.push\_back(sqrt(y2));

x.push\_back(-sqrt(y2));

}

return x.size();

}

// Unit test

// f1

TEST(FunctionTest, F1\_Correct) {

EXPECT\_EQ(f1(11), 22); // x > 10

EXPECT\_EQ(f1(5), -5); // x <= 10

EXPECT\_EQ(f1(-3), 3); // x <= 10

}

TEST(FunctionTest, F1\_Wrong) {

EXPECT\_EQ(f1\_wrong(11), 22);

EXPECT\_EQ(f1\_wrong(5), -5);

// Wrong code still testing the error branch

EXPECT\_EQ(f1\_wrong(-3), 3);

}

// f2

TEST(FunctionTest, F2\_Test) {

EXPECT\_EQ(f2(5), 10);

EXPECT\_EQ(f2(15), 30);

EXPECT\_EQ(f2(1), 2);

}

// f3

TEST(FunctionTest, F3\_Test) {

EXPECT\_EQ(f3(1), 2);

EXPECT\_EQ(f3(10), 20);

}

// findMax

TEST(FunctionTest, FindMax\_Test) {

EXPECT\_EQ(findMax(5, 3, 2), 5);

EXPECT\_EQ(findMax(3, 6, 1), 6);

EXPECT\_EQ(findMax(1, 4, 7), 7);

// Wrong code will FAIL

EXPECT\_EQ(findMax(2, 2, 2), 2);

EXPECT\_EQ(findMax(-1, -2, -3), -1);

}

// Quartic solve

TEST(QuarticSolverTest, BasicCases) {

vector<double> x;

// Infinite solutions

EXPECT\_EQ(solveQuartic(0, 0, 0, x), -1);

// No solution

EXPECT\_EQ(solveQuartic(0, 0, 5, x), 0);

EXPECT\_EQ(solveQuartic(0, 2, -8, x), 2);

EXPECT\_NEAR(x[0], 2.0, 1e-6);

EXPECT\_NEAR(x[1], -2.0, 1e-6);

EXPECT\_EQ(solveQuartic(1, -3, 2, x), 4);

vector<double> expected = { sqrt(2), -sqrt(2), 1.0, -1.0 };

for (size\_t i = 0; i < x.size(); i++)

EXPECT\_NEAR(x[i], expected[i], 1e-6);

}

// main

int main(int argc, char\*\* argv) {

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

**Test results:**

**Ảnh có chứa văn bản, ảnh chụp màn hình

Nội dung do AI tạo ra có thể không chính xác.**

**Python code compiled for C++**

!g++ -std=c++17 Lab1\_VV\_TestCases.cpp -IC:/Users/NHU/googletest/googletest/include -LC:/Users/NHU/googletest/build\_mingw/lib -lgtest -lgtest\_main -pthread -o test.exe

!test.exe

**Test results:**

**Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, tài liệu

Nội dung do AI tạo ra có thể không chính xác.**

**-- THE END --**