**EXERCISE 1**

**NAME: Nguyễn Lê Nhật Minh**

**STUDENT CODE: 3122411125**

**PHẦN I: VERIFICATION & VALIDATION**

Vấn đề của hai hệ thống giải phương trình bậc hai

**Hệ thống 1**: Công thức x2 = - b -sqrt (DELTA/2a) là sai. Phải sửa là x2 = (-b - sqrt(DELTA)) / (2\*a)

**Hệ thống 2**: Công thức đúng, nhưng cần kiểm tra thêm trường hợp DELTA < 0 để tránh lỗi khi lấy căn số âm.

**PHẦN II: TEST CASES**

a) Hàm f1

int f1(int x) { if (x > 10) return 2 \* x; else return -x; }

**Test cases cần thiết:**

x = 11 → kiểm tra nhánh x > 10

x = 10 → kiểm tra nhánh else

x = 0 → kiểm tra giá trị biên

x = -5 → kiểm tra giá trị âm

**Tổng: 4 test cases**

b) Hàm f1 bị sửa sai

int f1(int x) { if (x > 10) return 2 \* x; else if (x > 0) return -x; else return 2; }

**Test cases cần thiết để phát hiện lỗi:**

x = 11 → kiểm tra nhánh đầu

x = 5 → kiểm tra nhánh x > 0

x = 0 → kiểm tra nhánh else

x = -3 → kiểm tra giá trị âm

**Tổng: 4 test cases**

c) Hàm f2

int f2(int x) { if (x < 10) return 2 \* x; else if (x < 2) return -x; else return 2 \* x; }

**Lỗi logic:** x < 2 không thể xảy ra sau x < 10, vì x < 2 ⊂ x < 10. Nhánh thứ hai không bao giờ được thực thi.

**Test cases:**

x = 1 → kiểm tra nhánh đầu

x = 10 → kiểm tra nhánh cuối

x = 2 → kiểm tra giá trị biên

**Tổng: 3 test cases**

d) Hàm f3(int x)

int f3(int x) { if (log(x \* x \* cos(x)) < 3 \* x) return 2 \* x; else return 2 \* x; }

**Lỗi logic:** Cả hai nhánh đều trả về 2 \* x, nên không cần kiểm thử phân nhánh.

**Test cases:**

Chỉ cần kiểm tra giá trị hợp lệ của x để tránh lỗi log hoặc cos

**Tổng: 2 test cases (ví dụ: x = 1, x = 10)**

e) Hàm 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; }

**Lỗi:** Nếu các số bằng nhau, max vẫn là 0 → sai.

**Test cases:**

num1 = 5, num2 = 3, num3 = 2 → max = num1

num1 = 2, num2 = 6, num3 = 4 → max = num2

num1 = 1, num2 = 2, num3 = 9 → max = num3

num1 = 5, num2 = 5, num3 = 5 → kiểm tra trường hợp bằng nhau

**Tổng: 4 test cases**

**PHẦN III: THỰC HÀNH VIẾT CODE KIỂM TRA**

**1. Mô tả bài toán**

Bài toán: Giải phương trình trùng phương:

ax4+bx2+c=0a x^4 + b x^2 + c = 0ax4+bx2+c=0

**Input:** 3 số thực a, b, c (hệ số phương trình).

**Output:**

Nếu vô số nghiệm → trả về -1

Nếu vô nghiệm → trả về 0

Nếu có nghiệm → trả về số lượng nghiệm thực và liệt kê nghiệm đó.

**2. Các test cases kiểm tra chương trình**

| **Test case** | **Input (a, b, c)** | **Ý nghĩa** | **Output** |
| --- | --- | --- | --- |
| 1 | (0, 0, 0) | Vô số nghiệm | -1 |
| 2 | (0, 0, 1) | Phương trình vô nghiệm | 0 |
| 3 | (0, 2, -8) | Giải phương trình bậc nhất | 2 nghiệm: x = 2, x = -2 |
| 4 | (1, -6, 9) | y = 3 (kép) → x = ±√3 | 2 nghiệm x = √3, x = -√3 |
| 5 | (1, 0, -4) | y = ±2 → x = ±√2, ±√-2 (loại) | 2 nghiệm x = √2, x = -√2 |
| 6 | (1, 0, 4) | delta < 0 → vô nghiệm | 0 |
| 7 | (1, 0, 0) | y = 0 → x = 0 | x = 0 |

**3. Viết đoạn mã kiểm thử tự động**

**Kiểm thử tự động bằng C++**

#include <iostream>

#include <cmath>

#include <cassert>

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;

}

void test\_solveQuartic() {

double x[4];

assert(solveQuartic(0, 0, 0, x) == -1); // TC1

assert(solveQuartic(0, 0, 1, x) == 0); // TC2

int n = solveQuartic(0, 2, -8, x); // TC3

assert(n == 2);

assert(fabs(x[0] - 2) < 1e-6 || fabs(x[1] - 2) < 1e-6);

n = solveQuartic(1, -6, 9, x); // TC4

assert(n == 2);

assert(fabs(x[0] - sqrt(3)) < 1e-6 || fabs(x[1] + sqrt(3)) < 1e-6);

n = solveQuartic(1, 0, -4, x); // TC5

assert(n == 4);

assert(fabs(x[0] - sqrt(2)) < 1e-6 || fabs(x[1] + sqrt(2)) < 1e-6);

assert(solveQuartic(1, 0, 4, x) == 0); // TC6

assert(solveQuartic(1, 0, 0, x) == 2); // TC7

assert(fabs(x[0]) < 1e-6 && fabs(x[1]) < 1e-6);

cout << "Tất cả test solveQuartic đều đúng!" << endl;

}

int main() {

test\_solveQuartic();

return 0;

}

**Hàm solveQuartic bằng Python**

import math

def solveQuartic(a, b, c):

x = []

if a == 0 and b == 0 and c == 0:

return -1, x # Vô số nghiệm

if a == 0 and b == 0:

return 0, x # Vô nghiệm

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

**Kiểm thử tự động cho solveQuartic**

def test\_solveQuartic():

def approx\_equal(a, b, tol=1e-6):

return abs(a - b) < tol

# TC1: Vô số nghiệm

n, x = solveQuartic(0, 0, 0)

assert n == -1

# TC2: Vô nghiệm

n, x = solveQuartic(0, 0, 1)

assert n == 0

# TC3: Phương trình bậc nhất → x = ±2

n, x = solveQuartic(0, 2, -8)

assert n == 2

assert any(approx\_equal(val, 2) for val in x)

# TC4: y = 3 → x = ±√3

n, x = solveQuartic(1, -6, 9)

assert n == 2

assert any(approx\_equal(val, math.sqrt(3)) for val in x)

# TC5: y = ±2 → x = ±√2

n, x = solveQuartic(1, 0, -4)

assert n == 4

assert any(approx\_equal(val, math.sqrt(2)) for val in x)

# TC6: delta < 0 → vô nghiệm

n, x = solveQuartic(1, 0, 4)

assert n == 0

# TC7: y = 0 → x = 0

n, x = solveQuartic(1, 0, 0)

assert n == 2

assert all(approx\_equal(val, 0) for val in x)

print("Tất cả test solveQuartic đều đúng!")

test\_solveQuartic()