**EXERCISE 1**

**NAME: Phạm Văn Tính**

**STUDENT CODE: 3122411214**

1. **VERIFICATION AND VALIDATION**

**Description:** The purpose is to help users to solve a 2-degree equation (a*x*2+b*x*+c).

**Spec:** Given input of *a*, *b*, and *c*; the system returns the outputs of *x*1 and *x*2 (extreme cases are temporarily not considered)

Two systems are developed as follows.

|  |  |  |  |
| --- | --- | --- | --- |
| SYSTEM 1 | SYSTEM 2 | | |
| a b c    Press here to get *x1*  solutions *x2* |  | Step 1: DELTA calculation  a b c    Press here to get DELTA *delta*  (discriminant) |  |
|  | | |
|  | Step 2: Solutions resolving  DELTA *x1*  Press here to get  solutions *x2* |  |
| Code:  x1 = (-b +sqrt(DELTA))/2a x2 = -b -sqrt(DELTA/2a) | Code:  DELTA = (b\*b-4\*a\*c)  x1 = (-b +sqrt(DELTA))/2a x2 = (-b -sqrt(DELTA))/2a | | |

What are the problems of those two systems? Write down your answer here.

Dưới đây là các **vấn đề** chính của cả hai hệ thống:

* Công thức chuẩn là x1 = (-b + sqrt(DELTA))/(2\*a), x2 = (-b - sqrt(DELTA))/(2\*a). Ở SYSTEM 1, 2 viết sai cú pháp, sai công thức và đặt nhầm vị trí dấu ngoặc.
* Không xử lí các trường hợp ngoại lệ:
  + Không kiểm tra a==0 (khi đó không còn phương trình bậc hai, phải xử lý thành phương trình bậc nhất hoặc vô số/không có ngiệm)
  + Không kiểm tra DELTA < 0 (với số thực, sqrt âm gây lỗi)
  + Không kiểm tra trường hợp nghiệm trùng DELTA = 0
  + Không kiểm tra chia cho 0 (nếu a==0 sẽ chia đôi thành 0)
  + Không đề cập xử lý kiểu số (int, float,…)

1. **TEST-CASES**

**Description:** Some input values

* 1. How many test-cases we need for the following function *f*1. What are they?

int f1(int x) { if (x > 10)

return 2 \* x; else

return -x;

}

**Mục tiêu kiểm thử:** kiểm tra hai nhánh x > 10 (true) và x ≤ 10 (false).

**Phân lớp tương đương:**

* Lớp A: x > 10
* Lớp B: x ≤ 10

**Giá trị biên cần xét:** điểm biên là x = 10 (chuyển nhánh).

**Số test tối thiểu:** 2 (một cho mỗi lớp).

**Test-case cụ thể + expected output:**

1. x = 11 → f1(11) = 2\*11 = 22 (kiểm nhánh true).
2. x = 10 → f1(10) = -10 (kiểm nhánh false).

**Gợi ý thêm (khuyến nghị):** thêm x = -5 → f1(-5) = 5 để chắc chắn xử lý số âm.

* 1. Check if your test-cases can detect error if *f*1 is implemented as follow

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?

**Mục tiêu kiểm thử:** phát hiện sai logic do thêm nhánh else if (x > 0) và else return 2\*x (những số âm trả 2\*x thay vì -x).

**Phân lớp tương đương (theo ý định ban đầu):**

* L1: x > 10
* L2: 0 < x ≤ 10
* L3: x ≤ 0

**Số test tối thiểu để phát hiện lỗi:** 3 (mỗi lớp một test).

**Test-case cụ thể + expected (của hàm đúng) / output buggy:**

1. x = 11 → expected 22 (cả hai phiên bản trả 22).
2. x = 5 → expected -5 (cả hai phiên bản trả -5).
3. x = -1 → expected 1 (phiên bản đúng: -(-1)=1), **phiên bản sai** trả 2 \* (-1) = -2 → **phát hiện lỗi**.

**Giải thích:** test với x=-1 cho thấy hiệu ứng khởi tạo nhánh else trả 2\*x gây sai với các số âm; đây là test bắt buộc để phát hiện lỗi.

* 1. 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?

**Mục tiêu kiểm thử:** phát hiện lỗi logic do thứ tự điều kiện khiến else if (x < 2) **không bao giờ đạt** (dead code).

**Phân lớp tương đương:** xét không trùng nhau theo điều kiện biểu diễn:

* L1: x < 2
* L2: 2 ≤ x < 10
* L3: x ≥ 10

Tuy nhiên vì code viết if (x < 10) trước nên thực tế chương trình phân nhánh là:

* Nhánh A: x < 10 → trả 2\*x (bao gồm L1 và L2)
* Nhánh B: x ≥ 10 → vào else if (x < 2) (không đạt) → rơi xuống else → trả 2\*x

**Số test tối thiểu:** 2 (cũng có thể 3 để minh họa dead code).

**Test-case cụ thể + expected (theo ý định) / actual (theo code hiện tại):**

1. x = 1 → **ý định**: nếu muốn phân biệt x<2 có thể trả -x (theo mục tiêu gốc nếu đó là ý định) — nhưng code hiện tại trả 2\*1 = 2. Nếu spec mong -1, test này phát hiện sai.
2. x = 10 → trả 20 (cả hai trường hợp giống nhau — không phân biệt).
3. x = 11 → trả 22 (thể hiện nhánh else).

**Giải thích:** quan trọng là hiểu yêu cầu ban đầu: nếu tác giả muốn x<2 trả -x, thì thứ tự phải là kiểm x<2 trước x<10. Test x=1 sẽ phát hiện lỗi.

**Kết luận:** cần sửa thứ tự điều kiện (hoặc sửa logic) — testcases nên bao gồm giá trị trong x<2, trong [2,10), và >=10.

* 1. 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;

}

**Mục tiêu kiểm thử:** 2 vấn đề chính: (1) chương trình *vô nghĩa* vì cả hai nhánh trả cùng giá trị (hàm luôn trả 2\*x); (2) tồn tại **vấn đề miền xác định** cho log(...) — nếu biểu thức bên trong ≤ 0 thì log gây lỗi runtime.

**Phân lớp tương đương / domain check:**

* L1: x \* x \* cos(x) > 0 (log hợp lệ)
* L2: x \* x \* cos(x) = 0 (log(0) undefined)
* L3: x \* x \* cos(x) < 0 (log âm → lỗi)

**Số test tối thiểu:** 3 (một cho mỗi trường hợp domain).

**Test-case cụ thể + expected (với chương trình đúng về mặt toán học) / actual:**

1. x = 1 → x\*x\*cos(x) = 1 \* cos(1) ≈ 0.5403 > 0 → log hợp lệ. Do cả hai nhánh trả 2\*x, f3(1)=2.
2. x = 0 → x\*x\*cos(x) = 0 → log(0) **sẽ gây lỗi runtime (domain error)**. Test này kiểm tra xử lý ngoại lệ/domain.
3. x = π (~3.14159) → cos(π) = -1 → x\*x\*cos(x) âm → log(…) lỗi runtime.

**Giải thích:** mặc dù logic nhánh trả như nhau (vô nghĩa), test phải kiểm tra **độ an toàn (robustness)**: hàm cần kiểm tra điều kiện trước khi gọi log và xử lý trường hợp domain (ví dụ: trả mã lỗi, hoặc không thực hiện so sánh nếu argument ≤ 0). Nếu spec mong hành vi khác (ví dụ muốn trả giá trị phụ thuộc vào điều kiện), thì cần sửa code.

* 1. 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?

**Mục tiêu kiểm thử:** phát hiện hai lỗi chính:

1. max khởi tạo = 0 → nếu cả ba số âm, hàm trả 0 (sai).
2. Dùng > (strict) bỏ qua trường hợp có hai (hoặc ba) giá trị **bằng nhau** dẫn đến không gán max → trả giá trị khởi tạo (0) → sai với các trường hợp tie.

**Phân lớp tương đương cần xét:**

* L1: num1 là max duy nhất (ví dụ 3,2,1)
* L2: num2 là max duy nhất (ví dụ 1,3,2)
* L3: num3 là max duy nhất (ví dụ 1,2,3)
* L4: Có tie (ví dụ num1 == num2 > num3)
* L5: Tất cả âm (ví dụ -1,-2,-3)
* L6: tất cả bằng nhau (0,0,0)

**Số test tối thiểu để phát hiện lỗi:** 3 (nhưng tốt nhất là 4–5 để phủ đầy các tình huống tie và số âm).

**Test-case cụ thể + expected / buggy output:**

1. (3, 2, 1) → **expected** 3, **buggy** trả 3 (vẫn đúng).
2. (3, 3, 1) → **expected** 3, **buggy** **có thể trả 0** (vì none of (num1>num2)&&(num1>num3) etc true) → **phát hiện lỗi**.
3. (-1, -2, -3) → **expected** -1, **buggy** trả 0 → **phát hiện lỗi khởi tạo**.
4. (2,3,3) → **expected** 3, **buggy** trả 0 → phát hiện lỗi tie ở vị trí khác.
5. (0,0,0) → **expected** 0, **buggy** trả 0 (vô phương) — tuy nhiên ở đây kết quả trùng nên không phát hiện lỗi.
6. **PRATICE 1**

* Mô tả bài toán, các input / output có thể có của bài toán
* Xây dựng các test cases kiểm tra tính đúng đắn chương trình
* Viết đoạn mã tự động kiểm tra chương trình cho bên dưới đúng hay sai?

#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;

}

**1. Mô tả bài toán**

Bài toán yêu cầu xây dựng hàm:

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

Hàm này dùng để **giải phương trình bậc tư** dạng:

Với

* Đầu tiên, coi → phương trình trở thành phương trình bậc hai theo biến y.
* Sau đó, từ nghiệm y, tìm các nghiệm t bằng cách lấy căn bậc hai:
  + Nếu y>0 ⇒ y > 0 ⇒ có 2 nghiệm thực
  + Nếu y=0 ⇒ y = 0 ⇒ có nghiệm kép t = 0
  + Nếu y<0 ⇒ y < 0 ⇒ không có nghiệm thực.

**2. Các input có thể có**

Hàm nhận **3 hệ số thực**:

* a: hệ số của
* b: hệ số của .
* c: hằng số.

Các trường hợp input quan trọng:

**Trường hợp đặc biệt (degenerate cases):**

* + a = 0, b = 0, c = 0 → phương trình 0 = 0, **vô số nghiệm**.
  + a = 0, b = 0, c ≠ 0 → phương trình c = 0, **vô nghiệm**.
  + a = 0, b ≠ 0 → phương trình bậc nhất

**Trường hợp phương trình bậc hai chuẩn (a ≠ 0):**

* + Nếu Δ < 0 → **không có nghiệm y thực** → không có nghiệm t.
  + Nếu Δ = 0 → có một nghiệm kép y → kiểm tra y có ≥ 0 hay không.
  + Nếu Δ > 0 → có hai nghiệm phân biệt y1, y2 → kiểm tra từng y có ≥ 0 hay không để sinh nghiệm t.

**3. Các output có thể có**

Hàm trả về **số nghiệm hoặc trạng thái**, cụ thể:

* -1: nếu phương trình có **vô số nghiệm**.
* 0: nếu phương trình **không có nghiệm thực** cho t.
* 2: nếu có **hai nghiệm thực** (±√y hoặc nghiệm kép 0).
* 4: nếu có **bốn nghiệm thực** (khi có hai giá trị y1, y2 đều dương khác nhau).

Ngoài ra, nghiệm được lưu vào mảng x[].

**Ví dụ output cụ thể:**

* Input (0,0,0) → Output -1 (vô số nghiệm).
* Input (0,0,1) → Output 0 (vô nghiệm).
* Input (0,1,-4) → Output 2, nghiệm {2, -2}.
* Input (1,0,-1) → Output 2, nghiệm {1, -1}.
* Input (1,-5,4) → Output 4, nghiệm {2, -2, 1, -1}.

**4. Đoạn mã kiểm thử tự động**

**Mã nguồn**

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

// ================== Hàm solveQuartic (theo đề) ==================

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

if (a == 0) {

if (b == 0) {

if (c == 0) return -1; // vô số nghiệm

else return 0; // vô nghiệm

}

else {

double y = -c / b;

if (y < 0) return 0;

else {

x[0] = sqrt(y);

x[1] = -sqrt(y);

return 2;

}

}

}

else {

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 n = 0;

if (y1 >= 0) {

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

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

}

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

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

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

}

return n;

}

}

// ================== Hàm so sánh nghiệm (không cần thứ tự) ==================

bool compareSolutions(const vector<double>& got, const vector<double>& expected, double eps = 1e-6) {

if (got.size() != expected.size()) return false;

vector<bool> used(expected.size(), false);

for (double g : got) {

bool ok = false;

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

if (!used[i] && fabs(g - expected[i]) < eps) {

used[i] = true;

ok = true;

break;

}

}

if (!ok) return false;

}

return true;

}

// ================== Cấu trúc test-case ==================

struct TestCase {

double a, b, c;

int expectedCount;

vector<double> expectedRoots;

string desc;

};

int main() {

// Danh sách test-case

vector<TestCase> tests = {

{0,0,0,-1,{},"Vo so nghiem"},

{0,0,1,0,{},"Vo nghiem"},

{0,1,-4,2,{2,-2},"Bac nhat y=4"},

{0,1,0,2,{0,0},"Bac nhat y=0"},

{1,0,-1,2,{1,-1},"y^2 - 1 = 0"},

{1,-2,1,2,{1,-1},"Delta = 0"},

{1,0,1,0,{},"Delta < 0"},

{1,-5,4,4,{2,-2,1,-1},"4 nghiem thuc"}

};

int passed = 0;

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

double x[4];

int n = solveQuartic(tests[i].a, tests[i].b, tests[i].c, x);

vector<double> got;

for (int j = 0; j < n; j++) got.push\_back(x[j]);

cout << "Test " << i + 1 << " (" << tests[i].desc << "): ";

if (n == tests[i].expectedCount &&

compareSolutions(got, tests[i].expectedRoots)) {

cout << "PASS" << endl;

passed++;

}

else {

cout << "FAIL (got n=" << n << ", expected n=" << tests[i].expectedCount << ")" << endl;

if (n > 0) {

cout << " Roots got: ";

for (double r : got) cout << r << " ";

cout << endl;

}

}

}

cout << "==> Passed " << passed << "/" << tests.size() << " test cases." << endl;

return 0;

}

**Chạy chương trình**

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

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

**Kết luận:**

* Tất cả **8/8 test case** đều được chương trình test tự động đánh dấu **PASS**.
* Hàm solveQuartic hiện tại **đúng và ổn định** cho các tình huống chính.

**---o0o---**

**(End)**