**EXERCISE 1**

**NAME:Trương Phú Kiệt STUDENT CODE: 3122411109**

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.

**System 1:**

- Công thức sai:

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

- `x2 = -b - sqrt(DELTA/2a)`: sai, vì `2a` bị đặt trong căn bậc hai.

- Không có bước tính \*\*Δ = b² – 4ac\*\*, nên công thức không đầy đủ.

- Không kiểm tra trường hợp \*\*Δ < 0\*\* (không có nghiệm thực) hay \*\*a = 0\*\* (không còn là phương trình bậc 2).

**System 2:**

- Công thức tính Δ đúng: `DELTA = b\*b – 4\*a\*c`.

- Công thức nghiệm:

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

- `x2 = (-b - sqrt(DELTA)) / 2a` : đúng

- Tuy nhiên:

- Vẫn \*\*chưa xử lý Δ < 0\*\* → sẽ báo lỗi khi tính `sqrt(DELTA)`.

- Chưa xử lý trường hợp \*\*a = 0\*\* (lúc đó phương trình thành bậc 1).

- Giao diện cần 2 bước (tính Δ rồi mới giải), gây phức tạp không cần thiết cho người dùng.

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;

}

**Hàm có 2 nhánh điều kiện:**

* Nhánh 1: x > 10
* Nhánh 2: x <= 10

**Bộ test-cases**

Để bao phủ đầy đủ logic, cần ít nhất **2 test-cases**. Ngoài ra, nên thêm test-case cho giá trị biên và số âm để kiểm tra kỹ hơn.

| **Test-case** | **Input x** | **Expected Output** | **Ghi chú** |
| --- | --- | --- | --- |
| TC1 | 11 | 22 | Bao phủ nhánh x > 10 |
| TC2 | 10 | -10 | Biên trên của nhánh x <= 10 |
| TC3 | -5 | 5 | Giá trị âm thuộc nhánh x <= 10 |

**Khả năng phát hiện lỗi từ bộ test-cases**

* Bộ test-case này đảm bảo kiểm tra cả hai nhánh của hàm.
* Các giá trị biên (**10**) và số âm (**-5**) giúp phát hiện lỗi nếu cài đặt sai logic cho nhánh x <= 10.
* Do đó, **3 test-case trên là đầy đủ và hiệu quả** để kiểm tra hàm f1.
  1. Check if your test-cases can detect error if *f*1 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?

**Hàm có 3 nhánh điều kiện:**

* Nhánh 1: x > 10
* Nhánh 2: 0 < x <= 10
* Nhánh 3: x <= 0

**Bộ test-cases**

Để bao phủ đầy đủ các nhánh, cần ít nhất **3 test-cases**, mỗi test-case kiểm tra một nhánh.

| **Test-case** | **Input x** | **Expected Output** | **Ghi chú** |
| --- | --- | --- | --- |
| TC1 | 11 | 22 | Bao phủ nhánh x > 10 |
| TC2 | 5 | -5 | Bao phủ nhánh 0 < x <= 10 |
| TC3 | 0 | 0 | Bao phủ nhánh x <= 0 |

**Khả năng phát hiện lỗi từ bộ test-cases**

* Bộ test-case này đảm bảo kiểm tra cả ba nhánh của hàm.
* Nếu cài đặt sai logic ở bất kỳ nhánh nào, test-case tương ứng sẽ phát hiện ra.
* Ví dụ: nếu nhánh thứ hai (khi 0 < x <= 10) không trả về -x, test-case **TC2** sẽ phát hiện lỗi.
* Do đó, **3 test-case trên là đầy đủ và hiệu quả** để kiểm tra tính đúng đắn của hàm f1.
  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?

**Hàm có 3 nhánh điều kiện:**

* Nhánh 1: x < 10
* Nhánh 2: x < 2 (**không bao giờ được thực hiện**, vì nếu x < 2 thì đã rơi vào nhánh 1)
* Nhánh 3: x >= 10

**Bộ test-cases**

Thực chất chỉ cần **2 test-cases**, vì nhánh x < 2 bị che khuất (**unreachable code**).

| **Test-case** | **Input x** | **Expected Output** | **Ghi chú** |
| --- | --- | --- | --- |
| TC1 | 5 | 10 | Bao phủ nhánh x < 10 |
| TC2 | 15 | 30 | Bao phủ nhánh x >= 10 |

**Khả năng phát hiện lỗi từ bộ test-cases**

* Bộ test-case này cho thấy rằng nhánh else if (x < 2) **không bao giờ được thực hiện** → đây là **dead code** (lỗi thiết kế logic).
* Do đó, chỉ cần **2 test-case** là đủ để bao phủ tất cả các đường chạy khả thi của hàm f2.
* Việc phát hiện **“nhánh chết”** này minh chứng cho sự bất hợp lý trong cách cài đặt điều kiện.
  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;

}

**Hàm có 2 nhánh điều kiện:**

* Nhánh 1: if (log(x \* x \* cos(x)) < 3 \* x)
* Nhánh 2: else

**Bộ test-cases**

Để bao phủ đầy đủ cả hai nhánh, cần ít nhất **2 test-cases**.

| **Test-case** | **Input x** | **Expected Output** | **Ghi chú** |
| --- | --- | --- | --- |
| TC1 | 1 | 2 | Làm điều kiện if đúng (cos(1) > 0) |
| TC2 | 20 | 40 | Làm điều kiện if sai, rơi vào else |

**Khả năng phát hiện lỗi từ bộ test-cases**

* Về hành vi, cả hai test-cases đều cho ra cùng kết quả 2 \* x.
* Tuy nhiên, việc kiểm thử với cả hai nhánh đảm bảo rằng **cả if và else đều được thực thi**.
* Qua đó phát hiện rằng điều kiện if **không ảnh hưởng đến kết quả** → đây là **lỗi thiết kế logic (redundant condition)**.
  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?

**Hàm có 3 nhánh điều kiện:**

* Nhánh 1: (num1 > num2) && (num1 > num3)
* Nhánh 2: (num2 > num1) && (num2 > num3)
* Nhánh 3: (num3 > num1) && (num3 > num2)

**Bộ test-cases**

Để bao phủ đầy đủ các nhánh và phát hiện lỗi, cần ít nhất **5 test-cases** (bao gồm trường hợp đặc biệt khi tất cả giá trị bằng nhau).

| **Test-case** | **Input (num1, num2, num3)** | **Expected Output** | **Ghi chú** |
| --- | --- | --- | --- |
| TC1 | 5, 3, 1 | 5 | Bao phủ nhánh 1 (num1 lớn nhất) |
| TC2 | 2, 7, 3 | 7 | Bao phủ nhánh 2 (num2 lớn nhất) |
| TC3 | 4, 6, 9 | 9 | Bao phủ nhánh 3 (num3 lớn nhất) |
| TC4 | -3, -5, -7 | -3 | Phát hiện lỗi khi tất cả số âm (hàm sai trả về 0 thay vì -3) |
| TC5 | 10, 10, 10 | 10 | Trường hợp cả ba giá trị bằng nhau, hàm sai trả về 0 thay vì 10 |

**Khả năng phát hiện lỗi từ bộ test-cases**

* **TC1, TC2, TC3**: đảm bảo bao phủ cả ba nhánh điều kiện.
* **TC4**: giúp phát hiện lỗi khi hàm khởi tạo max = 0 và toàn bộ giá trị đầu vào là số âm.
* **TC5**: giúp phát hiện lỗi khi cả ba giá trị bằng nhau (không nhánh nào được chọn, hàm trả về sai).
* Do đó, cần **5 test-cases** để vừa bao phủ logic, vừa phát hiện các lỗi tiềm ẩn trong hàm findMax.

1. **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;

}

**Mô tả bài toán, các input / output có thể có của bài toán**

**Bài toán.** Giải phương trình dạng **tứ thức trùng phương**

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

Đặt y=x2 (≥0)y = x^2 \ (\ge 0)y=x2 (≥0) ⇒ giải phương trình bậc hai theo yyy:

* ay2+by+c=0a y^2 + b y + c = 0ay2+by+c=0

Với mỗi nghiệm yi≥0y\_i \ge 0yi​≥0 suy ra nghiệm thực của xxx: x=±yix = \pm \sqrt{y\_i}x=±yi​​.

**Quy ước giá trị trả về của hàm**

* -1: vô số nghiệm (trường hợp 0=00=00=0)
* 0 : không có nghiệm thực
* 2 : có 2 nghiệm thực (ví dụ ±r hoặc 0, 0)
* 4 : có 4 nghiệm thực (±√y1, ±√y2 với y1,y2>0y\_1, y\_2 > 0y1​,y2​>0)

**Input.** Ba số thực a b c (đọc từ stdin).  
**Output.** Một trong các chuỗi:

* " Infinite solutions."
* "No solution."
* " The equation has n real solution(s): <các nghiệm>"

**Xây dựng các test cases kiểm tra tính đúng đắn chương trình**

| ID | a | b | c | Kỳ vọng n | Kỳ vọng nghiệm (không cần đúng thứ tự; so sánh theo ε) | Ghi chú |
| --- | --- | --- | --- | --- | --- | --- |
| T1 | 0 | 0 | 0 | -1 | — | Vô số nghiệm |
| T2 | 0 | 0 | 5 | 0 | — | Vô nghiệm |
| T3 | 0 | 1 | 1 | 0 | — | x2=−1x^2 = -1x2=−1 (không thực) |
| T4 | 0 | 1 | 0 | 2 | 0, 0 | x2=0x^2 = 0x2=0 |
| T5 | 0 | 1 | -4 | 2 | -2, 2 | x2=4x^2 = 4x2=4 |
| T6 | 1 | 0 | 0 | 2 | 0, 0 | x4=0x^4 = 0x4=0 |
| T7 | 1 | -5 | 4 | 4 | -2, -1, 1, 2 | y={1,4}y = \{1, 4\}y={1,4} |
| T8 | 1 | 5 | 4 | 0 | — | Hai nghiệm yyy âm |
| T9 | 1 | -2 | -8 | 2 | -2, 2 | y={4,−2}y = \{4, -2\}y={4,−2} |
| T10 | 1 | 2 | 1 | 0 | — | Δ=0, y=−1\Delta = 0,\ y = -1Δ=0, y=−1 |
| T11 | 2 | -8 | 8 | 2 | -2, 2 | Δ=0, y=2\Delta = 0,\ y = 2Δ=0, y=2 |

**Quy ước so sánh số học**

* Dùng sai số ε=10−7\varepsilon = 10^{-7}ε=10−7.
* Chuẩn hoá -0.0 → 0.0 trước khi so sánh.
* Khi Δ=0\Delta = 0Δ=0, nếu nghiệm bậc hai là y0<0y\_0 < 0y0​<0 ⇒ không có nghiệm thực cho xxx; nếu y0=0y\_0 = 0y0​=0 ⇒ nghiệm kép x=0,0x = 0, 0x=0,0; nếu y0>0y\_0 > 0y0​>0 ⇒ x=±y0x = \pm \sqrt{y\_0}x=±y0​​ (đếm 2 nghiệm).

**Viết đoạn mã tự động kiểm tra chương trình cho bên dưới đúng hay sai?**

#include <bits/stdc++.h>

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;

}

bool almostEqual(double a, double b, double eps = 1e-6) {

    return fabs(a - b) <= eps;

}

// So sánh hai đa tập nghiệm (không xét thứ tự). Cho phép nghiệm lặp như {0,0}.

bool multisetEqual(vector<double> a, vector<double> b, double eps = 1e-6) {

    if (a.size() != b.size()) return false;

    auto norm = [eps](double v){

        // làm tròn để gom các nghiệm gần nhau

        return (long long) llround(v / eps);

    };

    unordered\_map<long long,int> cnt;

    for (double v : a) cnt[norm(v)]++;

    for (double v : b) {

        long long k = norm(v);

        auto it = cnt.find(k);

        if (it == cnt.end() || it->second == 0) return false;

        it->second--;

    }

    return true;

}

struct Test {

    double a, b, c;

    int expectedN;

    vector<double> expectedRoots; // để trống nếu -1 hoặc 0 nghiệm

    string name;

};

int main() {

    vector<Test> tests = {

        {0, 0, 0,  -1, {},                      "Infinite solutions"},

        {0, 0, 5,   0, {},                      "No solution (a=b=0,c!=0)"},

        {0, 1, -4,  2, {-2, 2},                 "Linear in y, y=4"},

        {0, 1, 0,   2, {0, 0},                  "Linear in y, y=0 → 0 & -0"},

        {1, 0, -1,  2, {-1, 1},                 "x^4 - 1 = 0"},

        {1, 0, 1,   0, {},                      "x^4 + 1 = 0 (no real)"},

        {1, -5, 4,  4, {-2, -1, 1, 2},          "y=1,4"},

        {1, 2, 1,   0, {},                      "(x^2+1)^2"},

        {1, -2, 1,  2, {-1, 1},                 "(x^2-1)^2 (double y)"},

        {2, 8, 8,   0, {},                      "(x^2+2)^2"},

    };

    int passed = 0, total = (int)tests.size();

    for (int i = 0; i < total; ++i) {

        const auto &t = tests[i];

        double x[4] = {0,0,0,0};

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

        bool ok = true;

        if (n != t.expectedN) {

            ok = false;

        } else if (n > 0) {

            vector<double> got(x, x + n);

            if (!multisetEqual(got, t.expectedRoots)) ok = false;

        }

        if (ok) {

            cout << "[PASS] #" << (i+1) << " - " << t.name << "\n";

            passed++;

        } else {

            cout << "[FAIL] #" << (i+1) << " - " << t.name << "\n";

            cout << "  Expected n=" << t.expectedN;

            if (!t.expectedRoots.empty()) {

                cout << ", roots=";

                for (double v: t.expectedRoots) cout << v << " ";

            }

            cout << "\n  Got n=" << n << ", roots=";

            for (int k=0; k<n; ++k) cout << x[k] << " ";

            cout << "\n";

        }

    }

    cout << "\nSummary: " << passed << "/" << total << " tests passed.\n";

    return (passed==total ? 0 : 1);

}

**[PASS] #1 - Infinite solutions**

**[PASS] #2 - No solution (a=b=0,c!=0)**

**[PASS] #3 - Linear in y, y=4**

**[PASS] #4 - Linear in y, y=0 → 0 & -0**

**[PASS] #5 - x^4 - 1 = 0**

**[PASS] #6 - x^4 + 1 = 0 (no real)**

**[PASS] #7 - y=1,4**

**[PASS] #8 - (x^2+1)^2**

**[PASS] #9 - (x^2-1)^2 (double y)**

**[PASS] #10 - (x^2+2)^2**

**Summary: 10/10 tests passed.**

**---o0o---**

**(End)**