**EXERCISE 1**

**NAME: Huỳnh Minh Quân STUDENT CODE: 3122411167**

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:**

* **Verification:**
  + Chưa gán biến và tính DELTA
  + x1 = (-b + sqrt(DELTA))/2a: Thiếu phép tính nhân ở mẫu -> Sai công thức tính nghiệm
    - Công thức đúng: x1 = (-b + sqrt(DELTA))/2\*a
  + x2= -b - sqrt(DELTA/2a): Đặt sai dấu ngoặc “()” trong phép tính -> Sai công thức tính nghiệm
    - Công thức đúng: x2 = (-b - sqrt(DELTA))/2\*a
* **Validation:**
  + Input: a = 1, b = 2, c = 1
  + Expect: DELTA = 0, x1 = -1, x2 = -1
  + Output: Không có DELTA -> KHÔNG TÍNH ĐƯỢC NGHIỆM

**System 2:**

* **Verification:**
  + x1 = (-b +sqrt(DELTA))/2a: Thiếu phép tính nhân ở mẫu -> Sai công thức tính nghiệm
    - Công thức đúng: x1 = (-b + sqrt(DELTA))/2\*a
  + x2 = (-b -sqrt(DELTA))/2a:Thiếu phép tính nhân ở mẫu -> Sai công thức tính nghiệm
    - Công thức đúng: x2 = (-b - sqrt(DELTA))/2\*a
* **Validation:**
  + Input: a = 1, b = 2, c = 1
  + Expect: DELTA = 0, x1 = -1, x2 = -1
  + Output: DELTA = 0, Không tính được x1 và x2 -> KHÔNG TÍNH ĐƯỢC NGHIỆM

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;

}

* **Test-cases:**
  + Case 1: x > 10
    - Input: x = 11
    - Expect: x = 22
    - Output: x = 22
* Case 1 đúng
  + Case 2: x <= 10
    - Input: x = 10
    - Expect: x = -10
    - Output: x = -10
* Case 2 đúng
  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?

* **Test-cases:**
  + Case 1: x > 10
    - Input: x = 11
    - Expect: x = 22
    - Output: x = 22
* Case 1 đúng
  + Case 2: x <= 10
    - Input: x = 1
    - Expect: x = -1
    - Output: x = -1
* Case 2 đúng
  + Case 3: x < 0 ( do branch If thứ 2 cần test 2 điều kiện )
    - Input: x = -1
    - Expect: x = -2
    - Output: x = -2
* Case 3 đúng
  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?

* **Test-cases:**
  + Case 1: x < 10
    - Input: x = 9
    - Expect: x = 2
    - Output: x = 2
* Case 1 đúng
  + Case 2: x < 2
    - Input: x = 1
    - Expect: x = -1
    - Output: Không chạy được -> Dead code
* Case 2 sai
  + Case 3: x ≥ 10
    - Input: x = 10
    - Expect: x = 20
    - Output: x = 20
* Case 3 đúng
  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;

}

* **Test-case**:
  + Case 1: log(x) < 0
    - Input: x = 1 -> log(x) < 0 < 3\*x
    - Expect: x = 2
    - Output: x = 2
* Case 1 đúng
  + Case 2: log(x) > 0
    - Ịnput: x = 2 -> log(x) > 0, log(x) < 3\*x
    - Expect: x = 4
    - Output: x = 4
* Case 2 đúng
  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?

* **Test-cases:**
  + Case 1: 3 số ngẫu nhiên
    - Input: num1 = 2, num2 = 4, num3 = 10
    - Expect: max = num3 = 10
    - Output: max = num3
* Case 1 đúng
  + Case 2: có 2 số bằng nhau và lớn nhất
    - Input: num1 = 1, num2 = 5, num3 = 5
    - Expect: max = num2 = num3
    - Output: 0 (do điều kiện if thiếu trường hợp 2 số bằng nhau -> max giữ nguyên giá trị)
* Case 2 sai
  + Case 3: 3 số bằng nhau
    - Input: num1 = num2 = num3 = 5
    - Expect: max = 5
    - Output: 0 (do tất cả điều kiện if không thoả -> max giữ nguyên giá trị)
* Case 3 sai

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 yêu cầu bài toán:
  + Cho một chương trình tìm nghiệm của phương trình bậc 4:

ax4 + bx2 + c = 0

* + Trong đó a,c,b là các số thực
  + Đặt y = x2 để phương trình trở thành phương trình bậc 2 có nghiệm y sau đó giải tìm nghiệm x
* Input:
  + Nhập vào các giá trị số thực a,b,c
* Output:
  + Nếu n trả về -1 phương trình có vô số nghiệm
  + Nếu n trả về 0 phương trình vô nghiệm
  + Nếu trả ra nghiệm thực thì đồng thời lưu vào mảng x[]
* Các test-cases:
  + Case 1: a ≠ 0, b = 0
    - Input: a = 1, b = 0, c = -4
    - Expect: x = ±√2 -> phương trình có 2 nghiệm
    - Output: 2 nghiệm: x[√2, -√2]
* Case 1 đúng
  + Case 2: a = 0, b ≠ 0
    - Input: a = 0, b = 2, c = -8
    - Expect: x = ±2 -> phương trình có 2 nghiệm
    - Output: 2 nghiệm: x[-2, 2]
* Case 2 đúng
  + Case 3: a > 0, b > 0
    - Input: a = 1, b = 2, c = 1
    - Expect: y = -1 -> phương trình vô nghiệm
    - Output: No Solution -> y = -1 (phương trình vô nghiệm)
* Case 3 đúng
  + Case 4: a > 0, b < 0
    - Input: a = 1, b -5, c = 4
    - Expect: x = ±1, x = ±2 -> phương trình có 4 nghiệm
    - Output: 4 nghiệm: x[2, -2, 1, -1]
* Case 4 đúng
  + Case 5: a < 0
    - Input: a = -1, b = 0, c = 1
    - Expect: x = ±1 -> phương trình có 2 nghiệm
    - Output: 2 nghiệm: x[1, -1]
* Case 5 đúng
  + Case 6: a = b = c
    - Input: a = b = c = 0
    - Expect: 0 = 0 -> phương trình vô số nghiệm (return -1)
    - Output: Infinite solutions -> return 0 (phương trình vô số nghiệm)
* Case 6 đúng
  + Case 7: a = 0, b = 0, c ≠ 0
    - Input: a = b = 0, c = 5
    - Expect: 5 = 0 -> phương trình vô nghiệm (return 0)
    - Output: No solution -> return -1 (phương trình vô nghiệm)
* Case 7 đúng
* Kết luận:
  + Sau khi kiểm tra các test-cases cho đoạn mã bên dưới thì tất cả đều ra kết quả output thoả yêu cầu các test-case cho thấy đoạn mã được thiết kế đúng

**---o0o---**

**(End)**