Tên: Phạm Nguyễn Viết Vinh

MSSV: 3122411249

**I. VERIFICATION AND VALIDATION**

Bài toán: giải phương trình bậc 2 ax² + bx + c = 0.

- SYSTEM 1:

Cho người dùng nhập a, b, c.  
Khi bấm nút → hiện kết quả x1, x2.

Code :

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

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

- SYSTEM 2:  
Cho người dùng nhập a, b, c.

Bấm nút → tính DELTA = b\*b - 4ac.

Nhập DELTA

Bấm nút -> tính nghiệm

Code:

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

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

Vấn đề là:

* System1: Công thức x2 sai ( sai dấu ngoặc), phải là  
  x2 = (-b - sqrt(DELTA)) / 2a
* System 2: đúng công thức nhưng chia làm 2 bước thủ công, không cần phải tính delta rồi nhập lại delta mới tính được nghiệm

**II. TEST-CASES**

1. Hàm f1:

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

Có 2 nhánh:

1. x > 10 trả về 2 \* x
2. x <= 10 trả về -x

Để test đủ branch coverage, cần ít nhất 2 test case:

* Một giá trị lớn hơn 10 (ví dụ: x = 11).
* Một giá trị nhỏ hơn hoặc bằng 10 (ví dụ: x = 5 hoặc x = -3).

Đáp án:

* Số lượng test case tối thiểu: 2
* Test case đề xuất:  
  1. x = 11 → kiểm tra nhánh x > 10

2. x = 5 (hoặc âm) → kiểm tra nhánh else

1. Hàm f1 bị sửa thành:

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

Ở đây có 3 nhánh khác nhau:

1. x > 10 trả về 2 \* x
2. 0 < x <= 10 trả về -x
3. x <= 0 trả về 2 \* x (KHÁC với hàm ban đầu, vì ban đầu là return -x)

Nếu chỉ dùng 2 test case ở phần a:

* x = 11: vẫn trả về 2 \* x → OK
* x = 5: vẫn trả về -x → OK

Không phát hiện lỗi khi x <= 0 vì chưa có test case cho nhánh này.  
Nên ta cần thêm 1 test case với x <= 0 (ví dụ x = -3) để kiểm tra.

Đáp án:

* Số lượng test case tối thiểu: 3
* Test case đề xuất:  
  1. = 11 → kiểm tra x > 10  
  2. x = 5 → kiểm tra 0 < x <= 10  
  3. x = -3 → kiểm tra x <= 0 (sẽ phát hiện bug: hàm mới trả về 2 \* x thay vì -x)

c)

Hàm:

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

Phân tích:

* Thực tế logic này có lỗi (vì x < 2 đã bị bao trong x < 10, nhánh else if sẽ không bao giờ chạy).
* Như vậy, nhánh thứ 2 unreachable ⇒ cần ít nhất 2 test case để kiểm tra:  
  1. x < 10 → chạy nhánh đầu tiên
  2. x >= 10 → chạy nhánh cuối (else)

Test case đề xuất:

* x = 5 → kiểm tra nhánh x < 10
* x = 10 → kiểm tra nhánh else

Khi chạy test case, chúng ta sẽ phát hiện else if (x < 2) không bao giờ chạy được → đây là một lỗi thiết kế (unreachable code).

Đáp án câu c:

* Số lượng test case tối thiểu: 2
* Test case đề xuất: x = 5, x = 10
* Ghi chú: Code có lỗi logic (dead code ở nhánh else if)

d) Hàm:

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

Phân tích:

* Cả hai nhánh đều trả về 2 \* x.
* Không quan trọng điều kiện if là đúng hay sai, kết quả luôn như nhau.
* Do đó, về mặt chức năng, chỉ cần 1 test case duy nhất cũng đủ để kiểm thử.

Test case đề xuất:

* x = 5 (bất kỳ giá trị nào khác 0 đều được)

Đáp án câu d:

* Số lượng test case tối thiểu: 1
* Test case đề xuất: x = 5

e) Hàm

| 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;  } |
| --- |

Phân tích:

* Có 3 if tách rời, không phải if-else if.
* Nếu có 2 số bằng nhau và lớn nhất, không nhánh nào được chọn → trả về max = 0 ⇒ LỖI.

Để test đầy đủ, cần kiểm tra các tình huống sau:

| Trường hợp | num1 | num2 | num3 | Expected | Ghi chú |
| --- | --- | --- | --- | --- | --- |
| num1 lớn nhất | 3 | 2 | 1 | 3 | Kiểm tra nhánh 1 |
| num2 lớn nhất | 8 | 9 | 7 | 9 | Kiểm tra nhánh 2 |
| num3 lớn nhất | 4 | 5 | 6 | 6 | Kiểm tra nhánh 3 |
| Hai số bằng nhau & lớn nhất | 5 | 5 | 1 | 5 | Hàm trả về 0 ⇒ bug |
| Ba số bằng nhau | 7 | 7 | 7 | 7 | Hàm trả về 0 ⇒ bug |

Đáp án câu e:

* Số lượng test case tối thiểu: 5
* Test case đề xuất:
  1. (3, 2, 1) → Expect 3
  2. (8, 9, 7) → Expect 9
  3. (4, 5, 6) → Expect 6
  4. (5, 5, 1) → Expect 5 (phát hiện bug: trả về 0)
  5. (7, 7, 7) → Expect 7 (phát hiện bug: trả về 0)

**III. PRATICE 1**

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

1. Giải phương trình bậc 4 ẩn x dạng:

ax4 + bx2 + c = 0

1. Đặt y = x^2 ⇒ phương trình trở thành phương trình bậc 2:

ay2 + by + c = 0

1. Tìm nghiệm y, sau đó lấy căn bậc hai để ra nghiệm x.

Input / Output:  
**Input:** 3 số thực a, b, c  
**Output:**

* -1 nếu có vô số nghiệm (a=b=c=0)
* 0 nếu vô nghiệm thực
* n số nghiệm thực (2 hoặc 4 nghiệm) được lưu vào mảng x[].

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

| Test Case | a | b | c | Ý nghĩa | Expected Output |
| --- | --- | --- | --- | --- | --- |
| TC1 | 0 | 0 | 0 | Phương trình 0=0 | n = -1 (vô số nghiệm) |
| TC2 | 0 | 0 | 5 | 5=0 (vô nghiệm) | n = 0 |
| TC3 | 0 | 1 | -4 | Bậc 2 theo x²: x² - 4=0 | x=±2 ⇒ n=2 |
| TC4 | 1 | 0 | 1 | x⁴+1=0 (vô nghiệm thực) | n=0 |
| TC5 | 1 | 0 | -1 | x⁴-1=0 ⇒ x²=±1 | y1=1 ⇒ x=±1 ⇒ n=2 |
| TC6 | 1 | -2 | 1 | x⁴-2x²+1=0 ⇒ (x²-1)²=0 | nghiệm kép ±1 ⇒ n=2 |
| TC7 | 1 | -5 | 1 | x⁴-5x²+4=0 | y=4,1 ⇒ x=±2, ±1 ⇒ n=4 |

• Code kiểm thử tự động

| #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;  }  void runTest(const string& name, double a, double b, double c, int expectedCount) {  double x[4];  int n = solveQuartic(a, b, c, x);  cout << name << " => Expected count=" << expectedCount  << ", Got count=" << n  << " => " << (n == expectedCount ? "PASS" : "FAIL")  << endl;  }  int main() {  cout << "=== TEST solveQuartic ===" << endl;  runTest("TC1 (0,0,0)", 0,0,0, -1);  runTest("TC2 (0,0,5)", 0,0,5, 0);  runTest("TC3 (0,1,-4)", 0,1,-4, 2);  runTest("TC4 (1,0,1)", 1,0,1, 0);  runTest("TC5 (1,0,-1)", 1,0,-1, 2);  runTest("TC6 (1,-2,1)", 1,-2,1, 2);  runTest("TC7 (1,-5,4)", 1,-5,4, 4);  return 0;  } |
| --- |