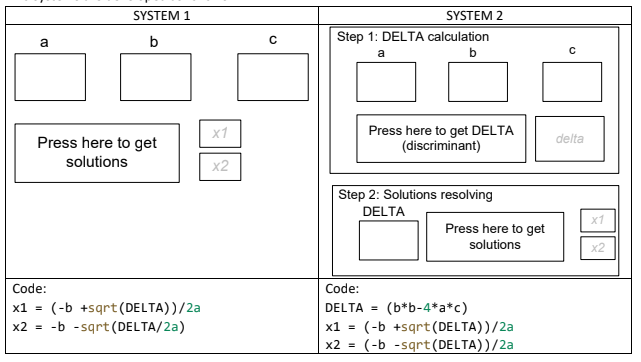
EXERCISE 1

NAME: Huỳnh Duy Khang

STUDENT CODE: 3122411088

1. VERIFICATION AND VALIDATION



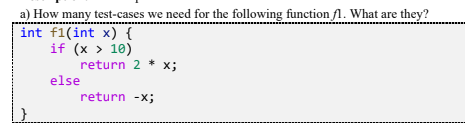
System 1:

* Hệ thống không tính và chưa khởi tạo DELTA trước → nếu DELTA < 0 thì sqrt(DELTA) gây lỗi.
* Hệ thống chưa xét tới trường hợp a=0, nếu a=0 thì khi chia cho 2a sẽ gây lỗi
* Code bị sai cú pháp ở nghiệm thứ nhất x1, thiếu ‘\*’ tại 2a: phải sửa lại thành x1 = (-b + sqrt(DELTA)) / (2\*a);
* Code bị sai công thức và cú pháp ở nghiệm thứ 2, x2 = -b - sqrt(DELTA/2a): phải sửa lại thành x2 = (-b - sqrt(DELTA)) / (2\*a)

System 2:

* Hệ thống chưa xét trường hợp DELTA<0, nếu DELTA<0 thì sqrt(DELTA) gây lỗi.
* Hệ thống chưa xét tới trường hợp a=0, nếu a=0 thì khi chia cho 2a sẽ gây lỗi
* Code bị sai cú pháp ở nghiệm thứ nhất và 2 x1,x2 thiếu ‘\*’ tại 2a: phải sửa lại thành x1 = (-b + sqrt(DELTA)) / (2\*a) và x2 = (-b - sqrt(DELTA)) / (2\*a)

1. TEST-CASES



a) Cần 2 test case để kiểm tra:

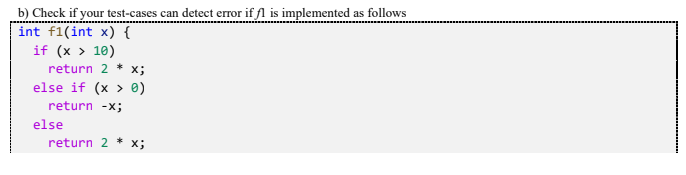
Test case 1: kiểm tra các số lớn hon 10

* Input: 11
* Kết quả mong đợi: 2\*11=22
* Output: 22
* Kết quả: đúng

Test case 2: kiểm tra các số từ 10 trờ xuống

* Input:1
* Kết quả mong đợi: -1
* Output: -1

b)

Cần 3 test case để kiểm tra:

Test case 1: kiểm tra các số lớn hơn 10

* Input: 11
* Kết quả mong đợi: 2\*11=22
* Output: 22

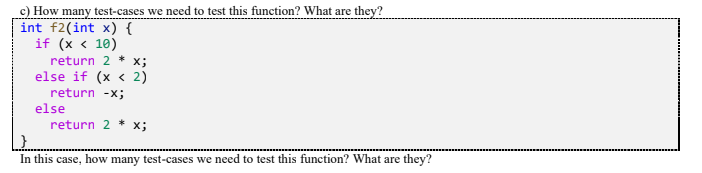
Test case 2: kiểm tra các số lớn hơn 0 và nhỏ hơn hoặc bằng 10

* Input:9
* Output: -9

Test case 3: kiểm tra các số nhỏ hơn hoặc bằng 0

* Input:-11
* Output: -22

c)



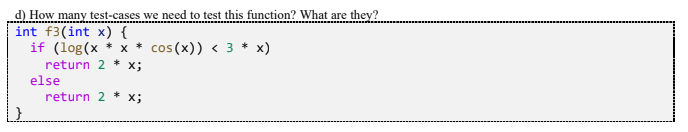
1. Ta có: if (x < 10) luôn true nếu x < 2, nên nhánh else if (x < 2) không bao giờ chạy
2. Cả 2 đều trả về 2\*x

Từ (1) và (2) => Cần 1 test case để kiểm tra:

Test case 1

* Input: 4
* Output: 8

d)

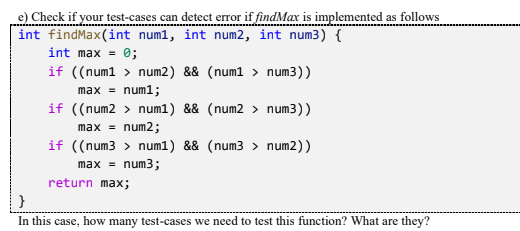


Cả hai nhánh đều return 2\*x → Chỉ cần 1 test case.

Test case 1

* Input: 1
* Output: 2

e)



Bug: Nếu có 2 số bằng nhau và lớn nhất → tất cả if fail → max giữ nguyên 0

Có 4 test-case cần kiểm tra

Test case 1 : Số lớn nhất tại num1:

* Input : findMax(3,2,1)
* Kết quả mong đợi: 3
* Output: 3
* Kết quả: đúng

Test case 2: Số lớn nhất tại num2:

* Input : findMax(2,3,1)
* Kết quả mong đợi: 3
* Output: 3
* Kết quả: đúng

Test case 3: Số lớn nhất tại num3:

* Input : findMax(2,1,3)
* Kết quả mong đợi: 3
* Output: 3

Test case 4: Có 2 số lớn nhất và bằng nhau:

* Input : findMax(3,3,1)
* Kết quả mong đợi: 3
* Output: 0
* Kết quả: sai

1. Practice 1
2. Mô tả bài toán

Bài toán giải phương trình bậc 4 với ba hệ số (a,b,c)

ax^4+bx^2+c=0

Đặt ẩn x^2=y, giải hệ phương trình ay^2+by+c=0, tính DELTA, nếu Delta<0, thì phương trình vô nghiệm, nếu Delta=0; thì phương trình ẩn y có 1 nghiệm kép, kiểm tra nghiệm kép lớn lơn 0 thì phương trình có 2 nghiệm, ngược lại nghiệm bé hơn 0 thì phương trình vô nghiệm. Nếu Delta>0 thì phương trình ẩn y có 2 nghiệm phân biệt, nếu cả 2 nghiệm bé hơn 0 thì phương trình vô nghiệm, 1 trong 2 nghiệm lớn hơn 0 thì phương trình có 2 nghiệm, 2 nghiệm đều lớn hơn 0 thì phương trình có 4 nghiệm

* Input: 3 số thực a,b,c
* Output: số nghiệm của phương trình (0,2,4)

1. Test-case

Test-case 1: a=0, b=0, c=0 => vô số nghiệm

Test-case 2: a=0, b=0, c≠0 → vô nghiệm

Test-case 3: a=0, b≠0, c<0 → có 2 nghiệm

Test-case 4: a≠0, Δ < 0 → vô nghiệm. (a=1,b=0,c=1)

Test-case 5: a≠0, Δ > 0, chỉ một y ≥ 0 → 2 nghiệm. (a=1,b=0,c=-1)

Test-case 6: a≠0, Δ = 0, nghiệm y < 0 → vô nghiệm . (a=1,b=2,c=1)

Test-case 7: a≠0, Δ = 0, nghiệm y ≥ 0 → 2 nghiệm (a=1,b= -2,c=1)

Test-case 8: a≠0, Δ > 0, cả hai y1,y2 ≥ 0 → 4 nghiệm (a=1,b= -3,c=2)

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

#include <iostream>

#include <cmath>

#include <vector>

#include <string>

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;

}

// Struct test case

struct TestCase {

double a, b, c;

int expectedSolutions;

string description;

};

int main() {

// Danh sách test cases

vector<TestCase> tests = {

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

{0,0,5,0,"No solution."},

{0,1,-4,2," The equation has 2 real solution"},

{1,0,1,0,"No solution."},

{1,0,-1,2,"The equation has 2 real solution"},

{1,2,1,0,"No solution."},

{1,-2,1,2,"The equation has 2 real solution"},

{1,-3,2,4,"The equation has 4 real solution"}

};

int passCount = 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);

cout << "Test " << i+1<<"( "<<tests[i].a<<","<< tests[i].b<<","<< tests[i].c<<")" << " (" << tests[i].description << "): ";

if (n == tests[i].expectedSolutions) {

cout << "PASS (solutions = " << n << ") ";

for (int i = 0; i < n; i++) {

cout << x[i] << " ";

}

cout<<endl;

passCount++;

} else {

cout << "FAIL (expected " << tests[i].expectedSolutions

<< ", got " << n << ")" << endl;

}

}

cout << "\nSummary: " << passCount << "/" << tests.size()

<< " test cases passed." << endl;

return 0;

}