**EXERCISE 1**

**NAME: PHẠM NHẬT PHƯƠNG STUDENT CODE: 3122411162**

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:  + Có 2a viết sai, có thể gây lỗi biên dịch, đúng phải là 2\*a  + Công thức đúng là x2 = [-b - sqrt(DELTA)] / (2\*a), nhưng hệ thống ghi là -b - sqrt(DELTA/2a) + Không rõ cách tính DELTA System2:  + Trả về delta trước, không hoàn toàn theo thông số chỉ yêu cầu x1, x2. |

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;

}

#include <iostream>

using namespace std;

// Hàm f1 theo đề

int f1(int *x*) {

    if (x > 10)

        return 2 \* x;    // Nếu x > 10 thì nhân đôi

    else

        return -x;       // Nếu x <= 10 thì lấy số đối

}

int main() {

    int x;

    cin >> x;             // Đọc 1 số nguyên từ stdin

    cout << f1(x) << endl; // In kết quả

    return 0;

A screenshot of a computer program

AI-generated content may be incorrect.}

|  |
| --- |
| Như vậy **cần ít nhất 3 test-case cơ bản**: x = -3, x = 10, x = 11. Để an toàn hơn, có thể thêm x = 0 và x = 20. => Tổng cộng: **5 test-case**. |

* 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;

}

#include <iostream>

using namespace std;

// Hàm f1 bị sai logic

int f1(int *x*) {

    if (*x* > 10) {

        return 2 \* *x*;

    } else if (*x* > 0) {

        return -*x*;

    } else {

        return 2 \* *x*; // BUG: lẽ ra phải là -x

    }

}

int main() {

    int x;

    cin >> x;              // đọc 1 số nguyên

    cout << f1(x) << endl; // in kết quả

    return 0;

}

In this case, how many test-cases we need to test this function? What are they?

A screenshot of a computer program

AI-generated content may be incorrect.

|  |
| --- |
| * Nếu chỉ test x > 0, sẽ **không phát hiện bug**. * Chỉ khi test x <= 0 (ví dụ x = -3), ta mới phát hiện hàm chạy sai (-x = 3 nhưng chương trình in -6). * Case x = 0 khó phát hiện lỗi vì -0 = 0 trùng với 2\*0 = 0.   Kết luận: **ít nhất 3 test-case** (-3, 5, 11). Nhưng để minh chứng lỗi rõ ràng thì nên dùng **5 test-case** như trên. |

* 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?

#include <iostream>

using namespace std;

// Hàm f2 như đề

int f2(int *x*) {

    if (*x* < 10)

        return 2 \* *x*;

    else if (*x* < 2) // Dead code, không bao giờ chạy

        return -*x*;

    else

        return 2 \* *x*;

}

int main() {

    int x;

    cin >> x;             // đọc 1 số nguyên

    cout << f2(x) << endl;

    return 0;

A screenshot of a computer

AI-generated content may be incorrect.}

|  |
| --- |
| Hàm này có **code thừa** ở nhánh else if (x < 2). Về test-case: chỉ cần **2 test-case** (x = 5, x = 10) để bao phủ toàn bộ. Test x = -3 cũng chạy được, nhưng thực ra nó **không mở nhánh mới**, vẫn thuộc x < 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;

}

#include <iostream>

#include <cmath>   // để dùng log và cos

using namespace std;

// Hàm f3 như đề

int f3(int *x*) {

    if (log(*x* \* *x* \* cos(*x*)) < 3 \* *x*)

        return 2 \* *x*;

    else

        return 2 \* *x*;

}

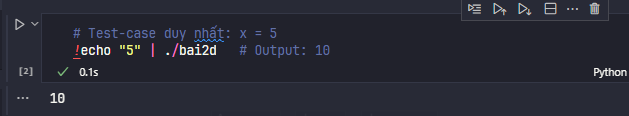
int main() {

    int x;

    cin >> x;

    cout << f3(x) << endl;

    return 0;

}

|  |
| --- |
| Hàm này thực chất **chỉ trả về 2 \* x** bất kể điều kiện → chỉ cần **1 test-case**.  **Test-case đại diện:** x = 5 (hoặc bất kỳ số nào hợp lệ để tránh log() lỗi domain). |

* 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?

#include <iostream>

using namespace std;

// Hàm tìm số lớn nhất trong 3 số

// Lỗi: không xử lý đúng khi có 2 số bằng nhau

int findMax(int *num1*, int *num2*, int *num3*) {

    int max = 0; // lỗi: gán sẵn 0

    if ((*num1* > *num2*) && (*num1* > *num3*)) max = *num1*;

    if ((*num2* > *num1*) && (*num2* > *num3*)) max = *num2*;

    if ((*num3* > *num1*) && (*num3* > *num2*)) max = *num3*;

    return max;

}

int main() {

    int a, b, c;

    cin >> a >> b >> c;  // nhập 3 số

    cout << findMax(a, b, c) << endl;

    return 0;

A computer screen shot of text

AI-generated content may be incorrect.}

A screenshot of a computer

AI-generated content may be incorrect.

|  |
| --- |
| * **Cases 1–3**: Kiểm tra khi từng số **riêng biệt là lớn nhất** → bao phủ 3 nhánh if. * **Cases 4–6**: Kiểm tra khi **2 số bằng nhau và lớn nhất** → code sai, trả về 0. * **Case 7**: Kiểm tra khi **3 số bằng nhau** → code sai, trả về 0. * **Case 8**: Kiểm tra khi tất cả **số âm**, kết quả đúng phải là số âm lớn nhất, nhưng code có thể trả về 0. Do -2 được để ở num 2 tức là if 2 nó mới bắt được cho nên -2 vẫn là số lớn nhất. Nếu như -2 rơi vào if 1 thì kết quả sẽ trả về 0 là số lớn nhất. |

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:**  **Input/output của bài toán:**  **Các test-case cần thiết:**  #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; // vô số nghiệm      }      if (*a* == 0 && *b* == 0) {          return 0; // vô nghiệm      }      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;  }  static const double EPS = 1e-9;  bool almost\_equal(double *a*, double *b*) {      return fabs(*a* - *b*) < EPS;  }  void normalize(vector<double>& *v*) {      for (auto &x : *v*) if (fabs(x) < EPS) x = 0.0;      sort(*v*.begin(), *v*.end());  }  struct *TestCase* {      double a, b, c;      int expected\_n;      vector<double> expected\_roots;  *string* desc;  };  int main() {      vector<*TestCase*> tests = {          {0,0,0, -1, {}, "a=b=c=0 → vô số nghiệm"},          {0,0,5, 0, {}, "a=b=0,c≠0 → vô nghiệm"},          {0,1,-4, 2, {2,-2}, "a=0,b=1,c=-4 → x=±2"},          {1,0,1, 0, {}, "x^4+1=0 → vô nghiệm thực"},          {1,-5,4, 4, {2,-2,1,-1}, "x^4 - 5x^2 + 4=0 → (x=±1,±2)"},          {1,0,0, 2, {0,0}, "x^4=0 → nghiệm kép x=0"},      };      int passed=0, failed=0;      for (auto &tc : tests) {          double out[4] = {NAN,NAN,NAN,NAN};          int n = solveQuartic(tc.a, tc.b, tc.c, out);          vector<double> got;          if (n > 0) got.assign(out, out+n);          vector<double> exp = tc.expected\_roots;          normalize(got);          normalize(exp);          bool ok = (n == tc.expected\_n);          if (ok && n > 0) {              for (int i = 0; i < n; i++) {                  if (!almost\_equal(got[i], exp[i])) {                      ok = false;                      break;                  }              }          }          if (ok) {              cout << "[PASS] " << tc.desc << "\n";              passed++;          } else {              cout << "[FAIL] " << tc.desc << "\n";              cout << "  Input: a=" << tc.a << " b=" << tc.b << " c=" << tc.c << "\n";              cout << "  Expected n=" << tc.expected\_n << " roots=";              for (auto v : exp) cout << v << " ";              cout << "\n  Got n=" << n << " roots=";              for (auto v : got) cout << v << " ";              cout << "\n";              failed++;          }      }      cout << "\nSummary: " << passed << " passed, " << failed << " failed.\n";      return failed == 0 ? 0 : 1;  }     | **STT** | **Input (a, b, c)** | **Kỳ vọng** | **Lý do kiểm thử** | | --- | --- | --- | --- | | 1 | (0, 0, 0) | Infinite solutions | Trường hợp vô số nghiệm | | 2 | (0, 0, 5) | No solution | Phương trình hằng vô lý | | 3 | (0, 2, -8) | 2 nghiệm (±2) | Bậc 2: 2y - 8 = 0 → y=4 → x=±2 | | 4 | (1, 0, -4) | 4 nghiệm (±√2, ±√-2 bỏ) → thực tế 4 nghiệm: ±√2, ±√2 (trùng) | Delta dương, 2 nghiệm y, đều dương | | 5 | (1, 0, 4) | No solution | Delta < 0 | | 6 | (1, -5, 4) | nghiệm y1=4, y2=1 → x=±2, ±1 (4 nghiệm thực) | Trường hợp có đủ 4 nghiệm phân biệt | | 7 | (1, 2, 1) | nghiệm y=-1 (bỏ), nghiệm duy nhất | Trường hợp delta = 0 nhưng nghiệm âm | | 8 | (1, -2, 1) | nghiệm y=1 (dương) → x=±1 (2 nghiệm) | Trường hợp delta=0, nghiệm dương | |

**---o0o---**

**(End)**