**EXERCISE 1**

**NAME:** Lê Thị Mỹ Hương **STUDENT CODE:** 3122411077

|  |
| --- |
| **I. VERIFICATION AND VALIDATION** |

**Description**: The purpose is to help users to solve a 2-degree equation (ax2+bx+c).

**Spec**: Given input of a, b, and c; the system returns the outputs of x1 and x2 (extreme cases are temporarily not considered)

**Two systems are developed as follows. What are the problems of those two systems? Write down your answer here.**

A screenshot of a computer

Description automatically generated

**Problems:**

Both systems have the following issues:

* They do not check the discriminant (Δ); if Δ < 0, taking the square root causes an error.
* They do not handle special cases (a = 0 → no longer a quadratic equation).
* They do not classify the roots (two distinct roots, a double root, or no real roots).
* Syntax errors: writing 2a instead of 2 \* a.

**System 1:** Not transparent — only returns the roots without indicating the type of roots.

**System 2:** Correct formula but complicated operations (2 steps), prone to inconsistency when changing a, b, or c.

|  |
| --- |
| **II. TEST-CASES** |

**Description**: Some input values

**a)**

**A white rectangular object with black text

Description automatically generated**

**How many test-cases we need for the following function f1?**

* Minimum: 2 test cases for full branch coverage
* Recommended: 3 test cases to include a boundary test

**What are they?**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input | Output | Expected | Notes |
| TC01 | 12 | 24 | 24 | Test x>10 branch |
| TC02 | 10 | -10 | -10 | Test x<=10 branch |

**b)**

**A white rectangular object with black lines

Description automatically generated**

**How many test-cases we need to test this function?**

* 3 test cases for full branch coverage.

**What are they?**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input | Output | Expected | Notes |
| TC01 | 15 | 30 | 30 | Test x>10 |
| TC02 | 10 | -10 | -10 | Test 0<x<=10 |
| TC03 | -3 | -6 | -6 | Test x<=0 |
|  |  |  |  |  |

**c)**

**A white rectangular object with black lines

Description automatically generated**

**How many test-cases we need to test this function?**

* 3 test cases for full branch coverage.
* The second condition ( else if (x<2) ) is dead code, due to if(x<10) being true first

**What are they?**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input | Output | Expected | Notes |
| TC01 | -3 | -6 | -6 | Test x<10 |
| TC02 | 12 | 24 | 24 | Test x>=10 |
| TC03 | 1 | 2 | N/A | Dead code: else if (x < 2) never executes |

**d)**

**A white rectangular object with black lines

Description automatically generated**

**How many test-cases we need to test this function?**

* 2 test cases for full branch coverage.
* Choose inputs so that x \* x \* cos(x) is positive (to avoid math errors with ln=0 or negative). Example: x=1 is safe; avoid values where cos(x)=0 like x = π/2.

**What are they?**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input | Output | Expected | Notes |
| TC01 | 1 | 2 | 2 | cos(x) > 0, log() argument valid, condition true. |
| TC02 | 80 | 160 | N/A | cos(x) < 0, log() argument invalid (negative), condition undefined but function still returns 2 \* x |

**e) Check if your test-cases can detect error if findMax is implemented as follow**

**A screenshot of a computer program

Description automatically generated**

**How many test-cases we need to test this function?**

* 5 test-cases are enough to detect the bug and cover the important scenarios.

**What are they?**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input | Output | Expected | Notes |
| TC01 | (5,3,1) | 5 | 5 | Distinct numbers where max is unique |
| TC02 | (2,7,4) | 7 | 7 | All distinct numbers, unique max |
| TC03 | (3,8,8) | 0 | 8 | Two max values equal |
| TC04 | (9,9,5) | 0 | 9 | Two max values equal |
| TC05 | (4,4,4) | 0 | 4 | All numbers equal |

|  |
| --- |
| **II. PRACTICE 1** |

**Problem Description:**

Solve the quartic equation of the form:

**ax4+bx2+c=0**

by reducing it to a quadratic in y=x2

**Input:**  Three real numbers:

* a — coefficient of y2
* b — coefficient of y
* c — constant term

(Used to solve ay2+by+c=0, where y=x2)

**Output:**

Depending on the values of a, b, and c, the output will be one of the following:

* "Infinite solutions." - when a=0, b=0, c=0.
* "No solution." - when there are no real solutions.
* "The equation has N real solution(s): x1 x2 ... xN" - where N is the number of real solutions and each xi is a real number (e.g., 2 -2 1 -1).

**Test Cases:** 9 test cases for full branch coverage

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input | Output | Expected | Notes |
| TC01 | (0,0,0) | Infinite solutions | Infinite solutions | Zero-degree equation: 0 = 0 ⇒ infinitely many solutions |
| TC02 | (0,0,1) | No solution | No solution | Zero-degree equation: 1 = 0 ⇒ no solution |
| TC03 | (0,2,8) | No solution | No solution | Quadratic equation: x² = -4 ⇒ no real roots |
| TC04 | (0,2,-8) | The equation has 2 real solution(s): 2 -2 | 2 roots: ±2 | Quadratic equation: x² = 4 ⇒ 2 real roots ±2 |
| TC05 | (1,0,-4) | The equation has 2 real solution(s): 1.41421 -1.41421 | 2 roots: ±√2 ≈ ±1.41421 | Quartic equation: Δ > 0, only one non-negative y ⇒ 2 real roots |
| TC06 | (1,0,4) | No solution | No solution | Quartic equation: Δ < 0 ⇒ no real y ⇒ no solution |
| TC07 | (1,-5,4) | The equation has 4 real solution(s): 2 -2 1 -1 | 4 roots: ±1, ±2 | Quartic equation: Δ > 0, both y₁ and y₂ ≥ 0 ⇒ 4 real roots |
| TC08 | (1,-4,4) | The equation has 2 real solution(s): 1.41421 -1.41421 | 2 roots: ±√2 ≈ ±1.41421 | Quartic equation: Δ = 0, repeated root y > 0 ⇒ 2 real roots |
| TC09 | (1,-1,0) | The equation has 3 real solution(s): 1 -1 0 | 3 roots: ±1, 0 | Quartic equation with c = 0: y₁ = 1, y₂ = 0 ⇒ 3 real roots |

**Code to Automatically Verify the Program**

from pathlib import Path

base\_dir = Path().resolve()  # Thư mục notebook chạy

input\_path = base\_dir / "input" / "testcasePrac1.inp"

code\_path = base\_dir / "code" / "practice1.cpp"

exe\_path = base\_dir / "code" / "practice1"

output\_path = base\_dir / "output" / "outputPrac1.txt"

with open(input\_path, "w") as f:

    f.write("""0 0 0

0 0 1

0 2 8

0 2 -8

1 0 -4

1 0 4

1 -5 4

1 -4 4

1 -1 0

""")

import subprocess

# !g++ code\_path -o code\_path

compile\_cmd = f"g++ {code\_path} -o {exe\_path}"

compile\_result = subprocess.run(compile\_cmd, shell=True, capture\_output=True, text=True)

expected = [

    "Infinite solutions",

    "No solution",

    "No solution",

    "2 roots: ±2",

    "2 roots: ±√2 ≈ ±1.41421",

    "No solution",

    "4 roots: ±1, ±2",

    "2 roots: ±√2 ≈ ±1.41421",

    "3 roots: ±1, 0"

]

run\_cmd = f"{exe\_path} < {input\_path}"

result = subprocess.run(run\_cmd, shell=True, capture\_output=True, text=True)

outputs = result.stdout.strip().split('\n')

with open( output\_path , "w") as f:

    for i, (output, exp) in enumerate(zip(outputs, expected), 1):

        f.write(f"Test {i}: Output={output:<55}  Expected={exp}\n")

        print(f"Test {i}: Output={output:<55}  Expected={exp}")