Name: Le Quang Hai

Id: 104175779

Notes: Many programming languages support **float** and **double** data types to represent real numbers, adhering to the IEEE 754 standard. In this assignment, we are using the **float** data type: **float.max** refers to the maximum positive finite value that can be represented by a float**, float.min** refers to the smallest positive normalized value that a float can represent. For simplicity, these extreme values may be denoted in shorthand as **float.max** and **float.min** throughout this document (|float.min| = |float.max|)

Task 1:

Since the testing objective of detecting ANY possible incorrect use of arithmetic operators in this program, the test cases must consider the scenarios where:

- The "-" in "A = A – B" is incorrect (the operator may have been mistyped to "A = A + B", "A = A * B", or "A = A / B")
- The "*" in "C = A * 2" is incorrect (the operator may have been mistyped to "C = A + 2", "C = A - 2", or "C = A / 2")
- Both operators are mistyped.

Therefore, we should mainly focus on the value of A, B that:

- Is a combination of positives, negatives, and zeros
- Makes the correct program and the incorrect one generate different outputs.
- Uses a combination of boundary values, typical values, and special cases (like zero and negative values).
- Represents the input domain (real numbers).
- Collides with failure input domain, all edge cases are covered.
- Cause calculating result to go beyond or below the limited range.

Since float number in programming languages can **only support a specific level of accuracy** in value. We must also consider a tolerant value range where result is acceptable and

We also consider the fact that "x/y" is only valid when y != 0.

Task 2:

Using test case (A=3, B=1) to test the above program is not able to achieve the required testing objective because the program is to work on REAL numbers, whereas such value of

A, B are, though considered real numbers, exclusively representative of integers – a special case of real numbers.

Task 3:

Test cases:

1. A = 5.469, B = 9.657

   The test case represents a random normal case with both positive inputs.

2. A = 0, B = 0

   The test case checks when all inputs are 0s.

3. A = 0, B = 3.7

   The test case checks a combination of positive value and zero

4. A = 3.7, B = 0

   The test case checks a combination of positive value and zero

5. A = 0, B = -3.7

   The test case checks a combination of negative value and zero

6. A = -3.7, B = 0

   The test case checks a combination of negative value and zero

7. A = -324.343, B = -22.31

   The test case represents a random normal case with both negative inputs.

8. A = -12.3, B = 213.2

   The test case represents a random normal case with inputs of both positive and negative values.

9. A = 12.3, B = -4.232

   The test case represents a random normal case with inputs of both positive and negative values.

10. A = 0, B = float.min

    This test case checks the functionality in case the final value exceeds the maximum value that can be represented (since A = A – B = 0 – float.min ≈ float.max then C = A * 2 is 2 times of the maximum float value).

The expected output depends on the behavior of the program and the handling of floating-point operations in the programming environment being used. For example I would recommend a validation so that the function only takes the A,B that that result in computations within the range of representable float values such as

float.min / 4 < A, B < float.max 4

11. A = float.min / 4, B = float.max / 4

This test case checks for the edge case as mentioned in test case 10. The test should result in A = A – B = float.min / 4 – float.max / 4 = float.min / 4 + float.min / 4 = float.min / 2 then C = A * 2 = 2* float.min / 2 = float.min, which is the smallest to be represented.

Task 4:

Given B = 1 now A = A – 1 and C = (A – 1) * 2

Consider all cases when the program does not work as expected and the test cases are possible to notify the error:

1. A – 1 = A + 1 => no possible value of A
2. A – 1 = A * 1 => no possible value
3. A – 1 = A / 1 => no possible value

➔ Any test case can detect if A – B operator is mistyped.

4. A * 2 = A + 2 => A = 2
5. A * 2 = A / 2 => A = 0
6. A * 2 = A – 2 => A = -2

➔ We must avoid the test cases of A being 2, 0 or -2.

We may create a simple program for these test cases

```
def test_program() -> None:
    b: float = 1.0
    a = -100.0
    while a < 100.0:
        expected = (a - b) * 2
        result = program(a, b)
        assert result == expected, f"Test case failed: {
            result} != {expected} for a = {a}"
        a += 0.1
        print("Test case pass for a = ", a)

    print("All test cases pass")


if __name__ == "__main__":
    test_program()
```

We might modify the test program that it accepts a tolerant mismatch range (since operations on floating number in most programming languages is not 100% precise)

```python
def test_program() -> None:
    b: float = 1.0
    a = -100.0
    tolerance = 1e-6  # You can adjust this value as needed
    while a < 100.0:
        expected = (a - b) * 2
        result = program(a, b)
        assert math.isclose(result, expected, rel_tol=tolerance, abs_tol=tolerance),
            f"Test case failed: {result} != {expected} for a = {a}"
        a += 0.1
        print("Test case pass for a = ", a)

    print("All test cases pass")


if __name__ == "__main__":
    test_program()
```

This will allow a tolerant range of 1e-6