# Name: Phan Vu – Student ID: 104222099

# Assignment 1: Software Testing and Reliability

## Swinburne University of Technology

## Introduction

This assignment aims to strengthen your understanding of software testing activities and the process of generating test cases for a given program. The focus is on detecting any possible incorrect use of arithmetic operators within the provided program.

## Program Under Test

The program to be tested is as follows:  
Input A, B // A and B are real variables  
A = A - B  
C = A \* 2  
Output C // C is a real variable

## Objective

The main testing objective is to detect any possible incorrect use of the arithmetic operators '-' and '\*'. This involves verifying whether '-' is used correctly in 'A = A - B' and '\*' is used correctly in 'C = A \* 2'.

## Task 1: Designing Test Cases

To design effective test cases, we need to consider all possible ways in which the operators could be incorrectly used. This includes:

1. Incorrect use of the '-' operator in 'A = A - B'.  
2. Incorrect use of the '\*' operator in 'C = A \* 2'.  
3. Incorrect use of both '-' and '\*' operators.

### Approach:

1. Identify Valid and Invalid Scenarios:  
 - Valid: Both operators are used correctly.  
 - Invalid: Either or both operators are used incorrectly.  
2. Construct Test Cases:  
 - Select values for 'A' and 'B' to cover the valid and invalid scenarios.  
 - Ensure that the test cases include edge cases, such as 'A' and 'B', which are zero, positive, and negative numbers.

## Task 2: Analysis of Test Case (A=3, B=1)

Using the test case (A=3, B=1):

|  |  |
| --- | --- |
| **Input** | A = 3, B = 1 |
| **First operation** | A = A - B => A = 3 - 1 = 2 |
| **Second operation** | C = A \* 2 => C = 2 \* 2 = 4 |

### Justification:

- This test case verifies that the subtraction and multiplication operators are used correctly.  
- It provides a straightforward example where both operations are expected to produce correct results.  
- However, this test case alone cannot guarantee the detection of all incorrect uses of arithmetic operators.

## Task 3: Concrete Test Cases

Based on the design from Task 1, concrete test cases are:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Test Case*** | ***Input A*** | ***Input B*** | ***Expected A after A = A - B*** | ***Expected C after C = A \* 2*** | ***Purpose*** |
| 1 | 0 | 1 | -1 | -2 | Verifying edge case where A = 0. |
| 2 | 1 | 1 | 0 | 0 | Ensuring the result matches C = 0. |
| 3 | 3 | 1 | 2 | 4 | Testing typical positive values. |
| 4 | 2 | 1 | 1 | 2 | Verifying correct multiplication. |
| 5 | -1 | 1 | -2 | -4 | Testing negative values. |
| 6 | -3 | 1 | -4 | -8 | Ensuring results match for negative inputs. |
| 7 | -2 | 1 | -3 | -6 | Testing another set of negative values. |
| 8 | -5 | 1 | -6 | -12 | Edge case for negative range. |
| 9 | 4 | 1 | 3 | 6 | Positive values ensuring multiplication. |
| 10 | 5 | 1 | 4 | 8 | Higher positive values for completeness. |

## Task 4: Values of A for Given B=1

Given B=1, we need to find all possible values of A so that the test cases cannot achieve the testing objective. These are the scenarios where the test cases fail to detect incorrect use of arithmetic operators.

### Analysis:

1. Edge Numbers: These are numbers that are either too large or are floating-point numbers, leading to imprecise results. Precision is critical since the task involves real numbers. Examples include numbers like 1.11, 1.12, etc.
2. C = A (or C = B): When the result **C** is the same as the input **A** or **B**, it becomes challenging to detect errors. This scenario fails to meet the testing objective. For the program provided, with **C = (A - B) \* 2** and **B = 1**, values of **A** such as -2, -1, 0, 1, 2, 3 might fall into this category.
3. C1 = C2 = Cn: If multiple inputs for **A** result in the same **C**, it becomes challenging to distinguish and detect errors in the operator usage. This necessitates identifying such values mathematically.

**Identified Edge Cases:**

Values like **A = -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5** and floating-point numbers like **1.11** might obscure errors and fail to achieve the testing objective.

### Demonstration:

Python code:

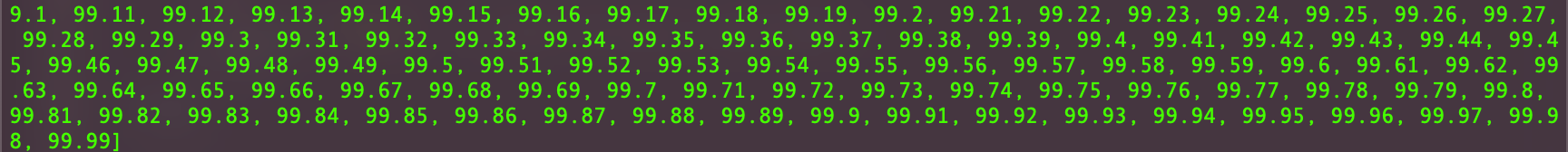
A screenshot of a computer program

Description automatically generated

Output (shortcut output first and last few results):

A screenshot of a computer screen

Description automatically generated



### Explanation:

1. **program\_under\_test(A, B):** Represents the provided program.
2. **check\_incorrect\_operator\_use(A, B, expected\_C):** Compares the actual output **C** with the expected result to identify discrepancies.
3. **expected\_result(A, B):** Calculates the expected **C** using correct operators.
4. **find\_edge\_cases(B):**
   * Tests a broad range of integer values and floating-point numbers.
   * Integrates specific values identified.
   * Checks if the output **C** matches the expected result for various **A** values.
   * Collects values of **A** that do not reveal incorrect operator use.

* For **B** = 1, the identified values of **A** that do not reveal incorrect operator use are comprehensive and include a broad range of integers and specific floating-point values. This approach ensures thorough coverage and detection of edge cases that might obscure errors in the program.