SET-221
Software Testing Technologies

LAB # 03

LAB Title

Google Test Basics: Setting up google test for a sample application in C++

Assessment of CLO: 04, PLO: 03

Student Name:			
Roll No.			
Semester		Session	

S. No.	Perf. Level / Criteria	Excellent (2.5)	Good (2)	Satisfactory (1.5)	Needs Improvement (0 ~ 1)	Marks Obtained
1	Project Execution & Implementation	Fully functional, optimized, and well-structured.	Minor errors, mostly functional.	Some errors, requires guidance.	Major errors, non-functional, or not Performed.	
2	Results & Debugging Or Troubleshooting	Accurate results with effective debugging Or Troubleshooting.	Mostly correct, some debugging Or Troubleshooting needed.	Partial results, minimal debugging Or Troubleshooting.	Incorrect results, no debugging Or Troubleshooting, or not attempted.	
3	Problem-Solving & Adaptability (VIVA)	Creative approach, efficiently solves challenges.	Adapts well, minor struggles.	Some adaptability, needs guidance.	Lacks innovation or no innovation, unable to solve problems.	
4	Report Quality & Documentation	Clear, structured, with detailed visuals.	Mostly clear, minor gaps.	Some clarity issues, missing details.	Poorly structured, lacks clarity, or not submitted.	
Total Marks Obtained Out of 10						

Experiment evaluated by

Instructor's Name	Engr.Bushra Aziz		
Date		Signature	

Objective: This lab aims to guide you through the process of creating a simple calculator application in Visual Studio using C++, and subsequently writing comprehensive unit tests using Google Test.

Setting up the Calculator Application in Visual Studio

Step 1: Create a New Project

1. Open Visual Studio.
2. Click "Create a new project."
3. Select "Empty Project" and click "Next."
4. Enter a project name (e.g., "CalculatorApp") and choose a location.
5. Click "Create."

Step 2: Add Source Files

1. In the Solution Explorer, right-click "Source Files" and select "Add" -> "New Item."
2. Choose "C++ File (.cpp)" and name it "Cal.cpp." Click "Add."
3. Add another C++ file named "Cal.h" for the header.

Step 3: Implement the Calculator Logic

Cal.h:

```
#include <cmath> // For pow()
class Calculator {
public:
    double add(double a, double b);
    double subtract(double a, double b);
    double multiply(double a, double b);
    double divide(double a, double b);
    double modulus(int a, int b);
    double power(double base, double exponent);

};
```

Cal.cpp:

```
// Cal.cpp

#include "Cal.h"

double Calculator::add(double a, double b) {
    return a + b;
}

double Calculator::subtract(double a, double b) {
    return a - b;
}

double Calculator::multiply(double a, double b) {
    return a * b;
}

double Calculator::divide(double a, double b) {
```

```
    if (b == 0) {
        return 0; // Or throw std::runtime_error("Division by zero");
    }
    return a / b;
}

double Calculator::modulus(int a, int b) {
    if (b == 0) {
        return 0; // Or throw an exception
    }
    return a % b;
}

double Calculator::power(double base, double exponent) {
    return std::pow(base, exponent);

}
```

Step 4: Add a Main Function (Optional, for Console Output Testing)

1. Right-click "Source Files" and add a new "C++ File (.cpp)" named "Calculator.cpp."

Calculator.cpp:

```
#include <iostream>
#include "Cal.h"

using namespace std;

int main()
{
    std::cout << "Sample Calculator Application!\n";

        Calculator calc;
        std::cout << "2 + 3 = " << calc.add(2, 3) << std::endl;
        std::cout << "5 - 1 = " << calc.subtract(5, 1) << std::endl;
        std::cout << "4 * 6 = " << calc.multiply(4, 6) << std::endl;
        std::cout << "10 / 2 = " << calc.divide(10, 2) << std::endl;
        std::cout << "10 / 0 = " << calc.divide(10, 0) << std::endl;
        std::cout << "10 % 3 = " << calc.modulus(10, 3) << std::endl;
        std::cout << "5 ^ 4 = " << calc.power(5, 4) << std::endl;
        return 0;

    }
```

Step 5: Build and Run (Optional)

1. Press Ctrl+Shift+B to build the solution.
2. Press Ctrl+F5 to run the application (if you added main.cpp).

Add a Test File

1. Right-click the solution, select "Add" -> "New Project."
2. Select "GoogleTest" and click "Next."
3. Name the project "Calculator_Test" and click "Create."

Step 3: Write Test Cases

// calculator_tests.cpp

```
#include "pch.h"
#include "C:\Users\baziz\source\repos\Calculator\Cal.cpp"
 TEST(CalculatorTests, TestAdd) {
    Calculator calc;
    EXPECT_EQ(5.0, calc.add(2.0, 3.0));
    EXPECT_EQ(-1.0, calc.add(2.0, -3.0));
    EXPECT_EQ(0.0, calc.add(0.0, 0.0));
 }

 TEST(CalculatorTests, TestSubtract) {
    Calculator calc;
    EXPECT_EQ(1.0, calc.subtract(3.0, 2.0));
    EXPECT_EQ(5.0, calc.subtract(2.0, -3.0));
    EXPECT_EQ(0.0, calc.subtract(0.0, 0.0));
 }

 TEST(CalculatorTests, TestMultiply) {
    Calculator calc;
    EXPECT_EQ(6.0, calc.multiply(2.0, 3.0));
    EXPECT_EQ(-6.0, calc.multiply(2.0, -3.0));
    EXPECT_EQ(0.0, calc.multiply(0.0, 5.0));
 }

 TEST(CalculatorTests, TestDivide) {
    Calculator calc;
    EXPECT_EQ(2.0, calc.divide(6.0, 3.0));
    EXPECT_EQ(-2.0, calc.divide(6.0, -3.0));
    EXPECT_EQ(0.0, calc.divide(0.0, 5.0));
    EXPECT_EQ(0.0, calc.divide(10.0, 0.0));
 }

 TEST(CalculatorTests, TestModulus) {
    Calculator calc;
    EXPECT_EQ(1, calc.modulus(5, 2));
    EXPECT_EQ(0, calc.modulus(10, 5));
    EXPECT_EQ(2, calc.modulus(8, 3));
    EXPECT_EQ(0, calc.modulus(8, 0));
    EXPECT_EQ(-1, calc.modulus(-5, 2));
 }

 TEST(CalculatorTests, TestPower) {
    Calculator calc;
    EXPECT_EQ(8.0, calc.power(2.0, 3.0));
    EXPECT_EQ(1.0, calc.power(5.0, 0.0));
    EXPECT_EQ(0.25, calc.power(2.0, -2.0));
    EXPECT_EQ(9.0, calc.power(-3.0, 2.0));

 }
```

Step 5: Build and Run Tests

1. Build the solution (Ctrl+Shift+B).
2. Open the "Test Explorer" (Test -> Windows -> Test Explorer).

3. Click "Run All Tests" in the Test Explorer.
4. Observe the test results.

Task: Your task is to modify Calculator application code and

1. Add a squareRoot (double a) function to the Calculator class. Ensure that the function handles negative inputs appropriately (e.g., return 0 or throw an exception).
2. Add a factorial (int n) function to the Calculator class. Handle negative inputs and large inputs appropriately (e.g., return 1 for negative inputs or throw an exception for large values that would cause an integer overflow).
3. Create new test cases for the squareRoot and factorial functions. Include test cases for: Positive inputs, Negative inputs, Zero inputs, Boundary conditions (e.g., very large numbers for factorial).