# Project Report

**Introduction:**

The provided code is a C++ implementation of a calculator, with a focus on mathematical operations. The calculator includes basic arithmetic operations, trigonometric functions, logarithms, factorials, and area calculations for various shapes. Additionally, there is a test fixture using the DeepState testing framework to verify the correctness and reliability of the calculator functions.

## CODES GITHUB LINK: https://github.com/sai-sunandh/CS567_Final_Project.git

**Code Structure:**

1. **Function Definitions:**
    - The code defines several mathematical functions for different operations, such as addition, subtraction, multiplication, division, modulus, powers/exponents, roots, factorial, trigonometry, logarithms, Euclidean distance, and area calculations.
    - Each function takes an array of numbers (**nums[]**) as input and performs the corresponding mathematical operation.
2. **Calculator Function:**
    - The **calculator** function acts as a switchboard for calling specific mathematical operations based on the user's choice.
    - It takes an integer **choice** and an array of numbers **nums[]** as parameters and dispatches the operation accordingly.
3. **DeepState Testing Fixture:**
    - The **TEST(Calculator, BasicOperations)** function is a test fixture that utilizes the DeepState testing framework.
    - It randomly selects a calculator operation (**choice**) and generates operands (**nums[]**) based on the operation's requirements.
    - The result of the calculator operation is then checked for correctness, ensuring it is not NaN (Not a Number), not infinity, and finite.
4. **Test Case Considerations:**
    - The test fixture accounts for the fact that certain operations (factorial, trigonometry, logarithms) require only one operand, while others require two.
    - It ensures that the generated operands are within a reasonable range (0 to 8956).

5. **Comments and Style:**
    - The code includes comments for each function, providing a brief explanation of its purpose and functionality.
    - There are some typographical errors in the code (e.g., **#inlcude** should be **#include**), which should be corrected.
6. **External Dependencies:**
    - The code depends on the DeepState testing framework, as indicated by the inclusion of headers (**#include <deepstate/DeepState.hpp>**).

The provided code includes a test fixture using the DeepState testing framework to test the calculator functions. Here's an overview of how the testing is performed:

1. **DeepState Testing Framework:**
    - The code includes the necessary headers for the DeepState testing framework: **#include <deepstate/DeepState.hpp>**.
    - The **TEST** macro is used to define a test fixture named "Calculator" with the test case "BasicOperations."
2. **Test Fixture Implementation:**
    - The test fixture generates a random choice (integer) for a calculator operation using **DeepState_IntInRange(1, 12)**.
    - Based on the choice, the maximum number of operands (**MAX_OPERANDS**) is determined. Some operations (choice 8, 9, 10) require only one operand, while others require two.
    - An array (**nums[]**) is then created to store the randomly generated operands. The values are generated using **DeepState_DoubleInRange(0, 8956)**.
3. **Calculator Function Invocation:**
    - The **calculator** function is called with the randomly generated choice and operands.
    - The result of the calculator operation is stored in the **result** variable.
4. **Assertions:**
    - The test includes three assertions to check the validity of the result:
        - **ASSERT_FALSE(isnan(result))**: Ensures that the result is not NaN (Not a Number).
        - **ASSERT_FALSE(isinf(result))**: Ensures that the result is not infinity.

- **ASSERT_TRUE(isfinite(result))**: Ensures that the result is finite.

The testing approach involves randomly selecting a calculator operation and generating operands within a specified range using the DeepState testing framework. The test assertions verify the correctness and validity of the calculator's results, checking for common issues like NaN and infinity. The use of DeepState provides a systematic and automated way to perform a variety of tests on the calculator functions. To complete the testing process, the commented-out main function should be uncommented and executed to run the DeepState test suite.

## Basic Commands:

Firstly, I just checked whether the files present in the path by using ls command which gave me **list of files** present in the path

Then ran the file using deepstate and checked for the **basic test** using ./a.out which ran and passed successfully.

```
user@0a95356ef836:~/deepstate/cp$ ls
calculator.cpp  calculator.cpp~  calculator.h  calculator.h.gch  test.cpp  test.cpp~
user@0a95356ef836:~/deepstate/cp$ clang++ test.cpp -ldeepstate
user@0a95356ef836:~/deepstate/cp$ ./a.out
TRACE: Running: Calculator_BasicOperations from test.cpp(14)
TRACE: Passed: Calculator_BasicOperations
user@0a95356ef836:~/deepstate/cp$
```

Next, I ran the tests using **Fuzzing**

```
user@0a95356ef836:~/deepstate/cp$ ./a.out --fuzz --output_test_dir d --timeout 30
```

which will **randomly generate test cases** and then I got the following output.

```
CRITICAL: test.cpp(40): Result is infinity
ERROR: Failed: Calculator_BasicOperations
ERROR: Failed to create file `d/26d39ac057c379e68a38c2961e7b2484a583f434.fail`
INFO: Done fuzzing! Ran 174613 tests (5632 tests/second) with 33985 failed/140628 passed/0 abandoned tests
user@0a95356ef836:~/deepstate/cp$
```

Giving the failure ID to small.test to reduce

```
INFO: Saved test case in file `d/f2b679378b9803e249de97962c7aabb9d60503c8.fail`
CRITICAL: test.cpp(40): Result is infinity
ERROR: Failed: Calculator_BasicOperations
INFO: Saved test case in file `d/136f68d5aafe2d19762924af67ba61685c7b440e.fail`
INFO: Done fuzzing! Ran 195747 tests (6524 tests/second) with 38072 failed/157675 passed/0 abandoned tests
user@e1320627c342:~/deepstate/cp$ deepstate-reduce ./a.out d/136f68d5aafe2d19762924af67ba61685c7b440e.fail small.test
INFO:deepstate:Setting log level from DEEPSTATE_LOG: 2
Original test has 20 bytes
Applied 1 range conversions
Writing reduced test with 20 bytes to small.test
================================================================================
Iteration #1 3.0 secs / 2 execs / 0.0% reduction
Structured deletion: PASS FINISHED IN 0.02 SECONDS, RUN: 3.02 secs / 4 execs / 0.0% reduction
Structured edge deletion: PASS FINISHED IN 0.01 SECONDS, RUN: 3.03 secs / 6 execs / 0.0% reduction
Removed 1 byte(s) @ 0: reduced test to 19 bytes
Applied 1 range conversions
Writing reduced test with 19 bytes to small.test
3.03 secs / 7 execs / 5.0% reduction
================================================================================
Removed 1 byte(s) @ 12: reduced test to 18 bytes
```

Final reduction from the below screenshot is 95%

```
user@e1320627c342: ~/deeps
Byte range removal: PASS FINISHED IN 0.0 SECONDS, RUN: 3.22 secs / 57 execs / 90.0% reduction
Structured swap: PASS FINISHED IN 0.0 SECONDS, RUN: 3.22 secs / 57 execs / 90.0% reduction
Reduced byte 1 from 10 to 1
Writing reduced test with 2 bytes to small.test
3.22 secs / 59 execs / 90.0% reduction
================================================================================
Byte reduce: PASS FINISHED IN 0.01 SECONDS, RUN: 3.23 secs / 60 execs / 90.0% reduction
================================================================================
Iteration #2 3.23 secs / 60 execs / 90.0% reduction
Removed 1 byte(s) @ 0: reduced test to 1 bytes
Writing reduced test with 1 bytes to small.test
3.23 secs / 61 execs / 95.0% reduction
================================================================================
1-byte chunk removal: PASS FINISHED IN 0.01 SECONDS, RUN: 3.24 secs / 62 execs / 95.0% reduction
4-byte chunk removal: PASS FINISHED IN 0.0 SECONDS, RUN: 3.24 secs / 63 execs / 95.0% reduction
8-byte chunk removal: PASS FINISHED IN 0.0 SECONDS, RUN: 3.24 secs / 64 execs / 95.0% reduction
1-byte reduce and delete: PASS FINISHED IN 0.0 SECONDS, RUN: 3.24 secs / 64 execs / 95.0% reduction
4-byte reduce and delete: PASS FINISHED IN 0.0 SECONDS, RUN: 3.24 secs / 64 execs / 95.0% reduction
8-byte reduce and delete: PASS FINISHED IN 0.0 SECONDS, RUN: 3.24 secs / 64 execs / 95.0% reduction
Byte range removal: PASS FINISHED IN 0.0 SECONDS, RUN: 3.24 secs / 64 execs / 95.0% reduction
Byte reduce: PASS FINISHED IN 0.0 SECONDS, RUN: 3.25 secs / 65 execs / 95.0% reduction
================================================================================
Iteration #3 3.25 secs / 65 execs / 95.0% reduction
================================================================================
Completed 3 iterations: 3.25 secs / 65 execs / 95.0% reduction
Padding test with 19 zeroes
```

Using input test file to get to know the reason for the failure of test case

And the reason for the test case failure is Not a number is returned back from the calculator function which is not acceptable.

```
user@e1320627c342:~/deepstate/cp$ ./a.out --input_test_file small.test
WARNING: No test specified, defaulting to first test defined (Calculator_BasicOperations)
TRACE: Initialized test input buffer with 20 bytes of data from `small.test`
TRACE: Running: Calculator_BasicOperations from test.cpp(14)
CRITICAL: test.cpp(37): Result is NaN
ERROR: Failed: Calculator_BasicOperations
ERROR: Test case small.test failed
```

I tried for another failed Test case which gave me return of infinity number error

```
user@e1320627c342:~/deepstate/cp$ ./a.out --input_test_file small.test
WARNING: No test specified, defaulting to first test defined (Calculator_BasicOperations)
TRACE: Initialized test input buffer with 20 bytes of data from `small.test`
TRACE: Running: Calculator_BasicOperations from test.cpp(14)
CRITICAL: test.cpp(40): Result is infinity
ERROR: Failed: Calculator_BasicOperations
ERROR: Test case small.test failed
```

# Universal Mutators

**Generating Mutants:**

We can create directory “m” for mutants by using mkdir command

We can generate mutants using the following command
**mutate calculator.cpp**

```
user@0a95356ef836:~/deepstate/cp/m$ mutate calculator.cpp
```

Then we can change directory using cd “m” to see the mutants produced in the m directory by using ls command

```
calculator.mutant.0.cpp    calculator.mutant.208.cpp  calculator.mutant.318.cpp  calculator.mutant.428.cpp
calculator.mutant.1.cpp    calculator.mutant.209.cpp  calculator.mutant.319.cpp  calculator.mutant.429.cpp
calculator.mutant.10.cpp   calculator.mutant.21.cpp   calculator.mutant.32.cpp   calculator.mutant.43.cpp
calculator.mutant.100.cpp  calculator.mutant.210.cpp  calculator.mutant.320.cpp  calculator.mutant.430.cpp
calculator.mutant.101.cpp  calculator.mutant.211.cpp  calculator.mutant.321.cpp  calculator.mutant.431.cpp
calculator.mutant.102.cpp  calculator.mutant.212.cpp  calculator.mutant.322.cpp  calculator.mutant.432.cpp
calculator.mutant.103.cpp  calculator.mutant.213.cpp  calculator.mutant.323.cpp  calculator.mutant.433.cpp
calculator.mutant.104.cpp  calculator.mutant.214.cpp  calculator.mutant.324.cpp  calculator.mutant.434.cpp
calculator.mutant.105.cpp  calculator.mutant.215.cpp  calculator.mutant.325.cpp  calculator.mutant.435.cpp
calculator.mutant.106.cpp  calculator.mutant.216.cpp  calculator.mutant.326.cpp  calculator.mutant.436.cpp
calculator.mutant.107.cpp  calculator.mutant.217.cpp  calculator.mutant.327.cpp  calculator.mutant.437.cpp
calculator.mutant.108.cpp  calculator.mutant.218.cpp  calculator.mutant.328.cpp  calculator.mutant.438.cpp
calculator.mutant.109.cpp  calculator.mutant.219.cpp  calculator.mutant.329.cpp  calculator.mutant.439.cpp
calculator.mutant.11.cpp   calculator.mutant.22.cpp   calculator.mutant.33.cpp   calculator.mutant.44.cpp
```

We can analyze the mutants by using the below command

```
user@0a95356ef836:~/deepstate/cp/m$ analyze_mutants calculator.cpp
```

And the mutation score I got is 1.0 which is 100% by running the mutants generated

```
#483: [26.93s 99.18% DONE]
RUNNING ./calculator.mutant.266.cpp...
./calculator.mutant.266.cpp KILLED IN 0.0515594482421875 (RETURN CODE 1)
  RUNNING SCORE: 1.0
================================================================================
#484: [26.98s 99.38% DONE]
RUNNING ./calculator.mutant.133.cpp...
./calculator.mutant.133.cpp KILLED IN 0.05186343193054199 (RETURN CODE 1)
  RUNNING SCORE: 1.0
================================================================================
#485: [27.03s 99.59% DONE]
RUNNING ./calculator.mutant.157.cpp...
./calculator.mutant.157.cpp KILLED IN 0.05173516273498535 (RETURN CODE 1)
  RUNNING SCORE: 1.0
================================================================================
#486: [27.08s 99.79% DONE]
RUNNING ./calculator.mutant.369.cpp...
./calculator.mutant.369.cpp KILLED IN 0.05122089385986328 (RETURN CODE 1)
  RUNNING SCORE: 1.0
================================================================================
MUTATION SCORE: 1.0
```

# Code Coverage:

When code with main function is saved and then the following command is used to observe code coverage for the calculator program

Where I took other .cpp file where I included main function into the calculator Program

```
user@0a95356ef836:~/deepstate/cp$ emacs mc.cpp
user@0a95356ef836:~/deepstate/cp$ clang++ --coverage -o calculator mc.cpp
user@0a95356ef836:~/deepstate/cp$ ./calculator
Enter operation choice:
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Modulus
6. Powers/Exponents
7. Calculating Roots
8. Factorial
9. Trigonometry
10. Logarithms
11. Euclidean Distance
12. Area
1
Enter operands (two values):
12
12
Result: 24
```

when executed gcov

```
user@0a95356ef836:~/deepstate/cp$ llvm-cov gcov mc.cpp
File '/usr/bin/../lib/gcc/x86_64-linux-gnu/7.5.0/../../../../include/c++/7.5.0/iostream'
Lines executed:100.00% of 1
/usr/bin/../lib/gcc/x86_64-linux-gnu/7.5.0/../../../../include/c++/7.5.0/iostream:creating 'iostream.gcov'

File 'mc.cpp'
Lines executed:41.43% of 70
mc.cpp:creating 'mc.cpp.gcov'
```


```
user@0a95356ef836:~/deepstate/cp$ llvm-cov gcov mc.cpp
File '/usr/bin/../lib/gcc/x86_64-linux-gnu/7.5.0/../../../../include/c++/7.5.0/iostream'
Lines executed:100.00% of 1
/usr/bin/../lib/gcc/x86_64-linux-gnu/7.5.0/../../../../include/c++/7.5.0/iostream:creating 'iostream.gcov'

File 'mc.cpp'
Lines executed:41.43% of 70
mc.cpp:creating 'mc.cpp.gcov'
```


From the above output we can observe that 41.43% code is getting covered using the –coverage .