Software Testing Project Report

Group members:

Riddhi Chatterjee (IMT2020094)
Ankrutee Arora (IMT2020034)

Link to our GitHub repository: https://github.com/Riddhi-Chatterjee/Large-Number-Calculator

Goal of the project:

The aim of this project is to perform mutation testing (on source code) (both at a unit level and at an integration level) of a Large Number Calculator.
The calculator is capable of operating with very large numbers, far beyond the maximum capacity of any datatype for numbers in any programming language. It is capable of providing the results of the computations upto an arbitrary precision as per the need of the user.

The following operations can be performed by the Large Number Calculator:

```
int choice;
cout<<endl;
cout<<"Choose an operation:"<<endl;
cout<<"[1]: Add two numbers"<<endl;
cout<<"[2]: Subtract two numbers"<<endl;
cout<<"[3]: Multiply two numbers"<<endl;
cout<<"[4]: Divide two numbers"<<endl;
cout<<"[5]: Compute square root of a number"<<endl;
cout<<"[6]: Compute the value of PI"<<endl;
```

What is Mutation Testing?

Mutation testing (for source code) involves testing the source code by making small modifications, thereby creating mutants. The goal is to demonstrate that these mutants can be "killed," indicating that even minor changes to the code can impact the program's execution and its resulting output.

Killing a mutant can be done weakly or strongly:

- **Weakly killing** a mutant means the memory state after executing the mutated statement differs from the original state, while the output on a test case may remain unchanged.
- **Strongly killing** a mutant requires a change in the program's output on a test case when a statement is mutated compared to when it is not.

In this context, the chosen approach for source code testing is mutation testing, specifically with the intention to strongly kill mutants.

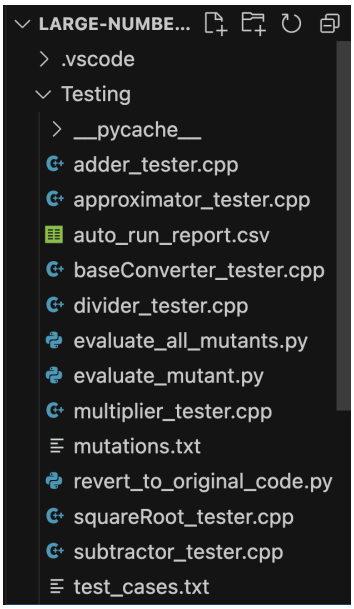
Mutation operators used:

- ## Unit level mutation operators:
  - **ArOR:** Arithmetic Operator Replacement
  - **COR:** Conditional Operator Replacement
  - **ROR:** Relational Operator Replacement
  - **AVI:** Absolute Value Insertion
  - **AsOR:** Assignment Operator Replacement
  - **UOD:** Unary Operator Deletion
  - **UOI:** Unary Operator Insertion
- ## Integration level mutation operators:
  - **IVPR:** Integration Parameter Variable Replacement
  - **IUOI:** Integration Unary Operator Insertion
  - **IPEX:** Integration Parameter Exchange
  - **IMCD:** Integration Method Call Deletion
  - **IREM:** Integration Return Expression Modification

Tools used:

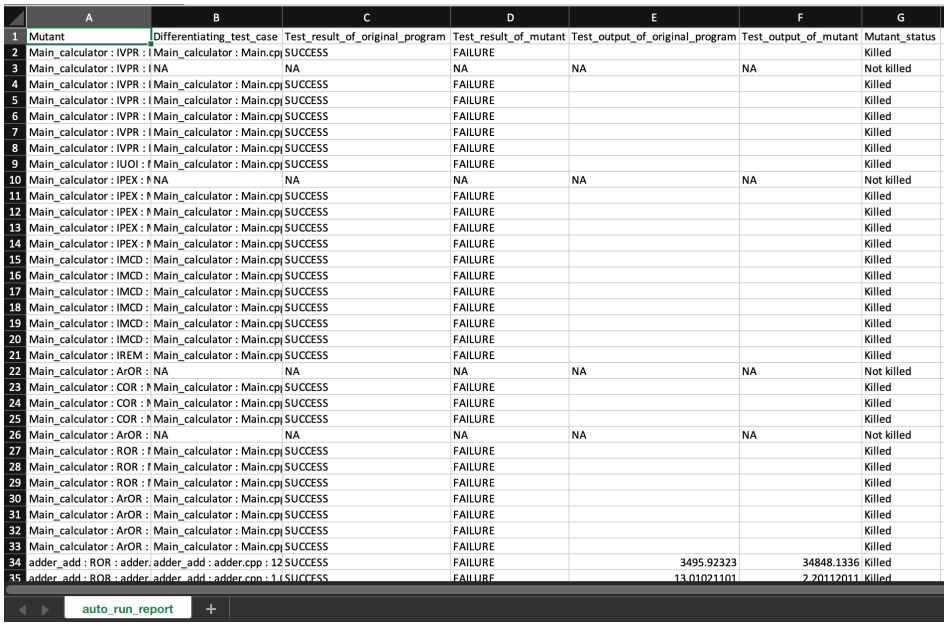
Our large number calculator is written in c++. Due to a lack of proper mutation testing support from the existing tools for mutation testing of c++ source code, we developed our own tool for mutation testing. Our mutation testing tool has been developed in python.

A peek into our code base would reveal a ‘Testing’ folder which contains the source code for our mutation testing tool and the testers that we wrote for each of the modules of the Large Number Calculator:



Description of files in the Testing folder:

- **X_tester.cpp:** This represents a tester written for the module X. We have 7 modules as follows: Adder, Approximator, BaseConverter, Divider, Multiplier, SquareRoot, and Subtractor.
- **auto_run_report.csv:** This is the final report generated by our mutation testing tool and contains the following information:



	A	B	C	D	E	F	G
1	Mutant	Differentiating_test_case	Test_result_of_original_program	Test_result_of_mutant	Test_output_of_original_program	Test_output_of_mutant	Mutant_status
2	Main_calculator : IVPR : I	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
3	Main_calculator : IVPR : I	NA	NA	NA	NA	NA	Not killed
4	Main_calculator : IVPR : I	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
5	Main_calculator : IVPR : I	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
6	Main_calculator : IVPR : I	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
7	Main_calculator : IVPR : I	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
8	Main_calculator : IVPR : I	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
9	Main_calculator : IUOI : I	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
10	Main_calculator : IPEX : N	NA	NA	NA	NA	NA	Not killed
11	Main_calculator : IPEX : N	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
12	Main_calculator : IPEX : N	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
13	Main_calculator : IPEX : N	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
14	Main_calculator : IPEX : N	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
15	Main_calculator : IMCD :	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
16	Main_calculator : IMCD :	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
17	Main_calculator : IMCD :	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
18	Main_calculator : IMCD :	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
19	Main_calculator : IMCD :	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
20	Main_calculator : IMCD :	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
21	Main_calculator : IREM :	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
22	Main_calculator : ArOR :	NA	NA	NA	NA	NA	Not killed
23	Main_calculator : COR : M	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
24	Main_calculator : COR : M	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
25	Main_calculator : COR : M	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
26	Main_calculator : ArOR :	NA	NA	NA	NA	NA	Not killed
27	Main_calculator : ROR : M	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
28	Main_calculator : ROR : M	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
29	Main_calculator : ROR : M	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
30	Main_calculator : ArOR :	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
31	Main_calculator : ArOR :	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
32	Main_calculator : ArOR :	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
33	Main_calculator : ArOR :	Main_calculator : Main.cp	SUCCESS	FAILURE			Killed
34	adder_add : ROR : adder.	adder_add : adder.cpp : 12	SUCCESS	FAILURE	3495.92323	34848.1336	Killed
35	adder_add : ROR : adder.	adder_add : adder.cpp : 1	SUCCESS	FAILURE	13.01021101	2.20112011	Killed

auto_run_report +

- **evaluate_mutant.py:** This is the main module for the mutation testing tool which contains code to evaluate a mutant against the original code with the help of the test cases present in test_cases.txt
- **evaluate_all_mutants.py:** Uses the evaluate_mutant.py module to evaluate all the mutations present in mutations.txt
- **test_cases.txt:** This contains the test_cases which we wrote to be able to test our original program and the mutants.
  - The format of a test case for a module is as follows:
    <Title> : <Filename> : <1st argument of function> : <2nd argument of function>…. : <expected output>
  - The format of a test case for the entire application is as follows:
    Main_calculator : Main.cpp : <choice of operation> : <num1> : <num2> : <precision required> : <expected output>

    Note: num1 and num2 might be optional. They are required based on the chosen operation.
- **mutations.txt:** This contains the various mutations. Each mutation has the following syntax:
  <Title> : <Operator_type> : <Filename> : <start line No.> : <start column No.> : <end line No.> : <end column No.> : <replacement line 1> : <replacement line 2> : …

  Basically this tells us to remove the segment of the code starting from ‘start line No.’ and ‘start column No.’ and ending at ‘end line No.’ and ‘end column No.’, with a concatenation of all the replacement lines provided.

Steps to run our tool:

- Go inside the Testing folder…
- To evaluate a single mutant run the evaluate_mutant.py file and pass the line number of the mutant you want to evaluate from the mutations.txt file. This will create a manual_run_report.csv containing all the relevant information regarding the evaluation of the chosen mutant.
- To evaluate all the mutants in the mutations.txt file, run the evaluate_all_mutants.py file. This will create a auto_run_report.csv file which contains all the relevant information regarding the evaluation of all the mutants present in the mutations.txt file.

Summary of mutation score:

- main application : 28/32 mutants killed = 87.5% mutants killed
- adder : 28/33 mutants killed = 84.5% mutants killed
- approximator : 32/40 mutants killed = 80% mutants killed
- baseConverter = 34/35 mutants killed = 97.1% mutants killed
- divider: 94/109 mutants killed = 86.23% mutants killed
- squareRoot: 7/7 mutants killed = 100% mutants killed
- multiplier : 12/12 mutants killed = 100% mutants killed
- subtractor: 7/11 mutants killed = 63.6% mutants killed

Overall mutation score —> Mutants killed = 242/279 = 86.74%

Vulnerabilities/Limitations found:

- All the sanity checks are done in the Main.cpp and no sanity check is performed in the individual modules. Thus, if by any chance the checks and balances of the Main.cpp file fails or someone decides to use the individual modules in his/her code, then the app wouldn't be robust towards invalid inputs.
- The approximator just truncates the result upto the required amount of precision. It doesn't do any rounding off operation. Though this error is very subtle, it can cause problems when high accuracy is desired.
- The individual modules can only work with positive numbers, and the subtractor module can only perform an operation of num1- num2, where num1 must be greater than or equal to num2.

References used:

- https://en.wikipedia.org/wiki/Mutation_testing#:~:text=Mutation testing involves modifying a,of mutants that they kill.
- A tool that we had searched for but didn't find much helpful: https://ucase.uca.es/mucpp/
- https://www.cs.cornell.edu/courses/cs5154/2021sp/resources/MutationTesting.pdf
- Class slides and notes!!

Contributions:

- Testers were written by Ankrutee. She also wrote half of the test cases and mutants
- evaluate_mutant.py, evaluate_all_mutants.py and remaining test cases and mutants were written by Riddhi.
- The Large Number Calculator project was a project done by both of us in our 4th Semester.