348 Industries

EECS 348 Fall 2023 Team Project: Arithmetic Evaluation Program

Test Case

Version <1.0>

Revision History

Date	Version	Description	Author
11/27/23	1.0	First Iteration of the Test Case document for the EECS 348 Fall 2023 Team Project. This document will use the implementation of the program and highlight the test cases used against the program to identify faults, errors and failures.	Dev Patel Jay Chung Ruth Higgason Colin Marett

Table of Contents

Test Case

1. Purpose

The test case document for the arithmetic problem evaluator project is a guide to systematically validate and verify the functionality, performance, and robustness of the system. Its overall purpose is to ensure that the arithmetic evaluator meets the specified requirements and functions accurately. The document outlines test cases for various aspects, including the verification of requirements to make sure it follows specifications, functional validation to confirm if the basic arithmetic operations are correct, error handling to assess the system's response to invalid inputs, performance testing to evaluate responsiveness under different conditions and user interface testing to check the usability. In essence, the test case document is a comprehensive tool that aids in thorough testing, fostering the development of a reliable and accurate basic arithmetic problem evaluator.

2. TABLE:

Test Case ID	Test Case Description	Test Data	Expected Results	Actual Result	Pass/Fail
MULT-1	Testing * operator	3*5	15	15	P
MULT-2	Testing * operator with parentheses	(3)*(5)	15	15	P
MULT-3	Testing negative sign with operator	(3)*(-5)	-15	-15	P
MULT-4	Testing many operations	3*5*2*4	120	120	P
MULT-5	Testing negative with many operations	-(3*(-5))	15	15	P
ORDER-1	Testing order of operations with * and +	3*5+2	17	17	P
ORDER-2	Testing order with +, -, and *	3-4*2+8	3	3	P
ORDER-3	Testing order with +, -, *, and /	2*3+4/2-1	7	7	P
ORDER-4	Testing order with +, -, *, /, and ()	(2+3)*3-3/(3-2)	12	12	P
ORDER-5	Testing order with all operators	(3+1)^2-6/3+2*(-7)	0	0	P
ADD-1	Testing + operator	2+4	6	6	P
ADD-1	Testing + operator with parentheses	(2)+(4)	6	6	P
ADD-3	Testing negative sign with operator	(2)+(-4)	-2	-2	P
ADD-4	Testing many operations	2+4+5+10	21	21	P

ADD-5	Many operations with negative sign	2+3+(-5)+6	6	6	P
SUB-1	Testing - operator	4-3	1	1	P
SUB-2	Testing - operator with parentheses	(5)-(3)	2	2	P
SUB-3	Testing negative sign with operator	5-(-3)	8	8	P
SUB-4	Testing many operations	18-9-4-3	2	2	P
SUB-5	Many operations with negative sign	14-(-2)-3-(-1)	14	14	P
DIV-1	Testing / operator	99/3	33	33	P
DIV-2	Testing / operator with parentheses	(6)/(3)	2	2	P
DIV-3	Testing negative sign with operator	-6/2	-3	-3	P
DIV-4	Testing many operations	(10/2)/5	1	1	P
DIV-5	Many operations with negative sign	-(15/3)/5	-1	-1	P
POW-1	Testing ^ operator	2^3	8	8	P
POW-2	Testing ^ operator with parentheses	(3+2)^2	25	25	P
POW-3	Testing negative sign with operator	-2^3	-8	-8	P
POW-4	Testing many operations	(2^2)^2	16	16	P
POW-5	Many operations with negative sign	(-2^2)^3	64	64	P
PAR-1	Testing lots of ()	(((2))+3)	5	5	P
PAR-2	Testing more ()	((2))-((3)*4)	-10	-10	P
PAR-3	Testing incorrect number of ()	((3)-4	-1	-1	P
PAR-4	Testing incorrect order of ()	((3))-4)(	-1	-1	P
MOD-1	Testing % operator	17%5	2	2	P
MOD-2	Testing % operator with parentheses	(34%33)%2	1	1	P
MOD-3	Testing negative sign with operator	(-2)%3	-2	-2	P
MOD-4	Testing many operations	51%35%13	3	3	P

3. Environmental needs

3.1.1 *Hardware: No unique hardware was used in the testing process. The tests were conducted on a regular windows environment on a laptop.*

3.1.2 *Software: No unique software is required to run the project. Any environment that can run a C++ program will be able to conduct and run tests on this program which allows for portability. This program was tested on repl.it and VScode.*

4. Special procedural requirements

- Input Validation:
  Ensure that the software handles invalid inputs gracefully, such as non-numeric characters or expressions that violate basic arithmetic rules.

- Error Handling:
  Verify that the software provides clear and meaningful error messages for scenarios such as division by zero or other exceptional conditions.

- Operator Precedence:
  Confirm that the software correctly follows the order of operations (PEMDAS/BODMAS) for complex expressions involving multiple operators.

- Boundary Testing:
  Test the software with extreme input values, including the maximum and minimum values supported by the data types used.

- Performance Testing:
  Evaluate the software's performance by testing it with a large number of calculations to ensure it operates efficiently and without significant delays.