Kansas Instruments

Arithmetic Expression Evaluator

Test Case

Version 1.03

Revision History

Date	Version	Description	Author
12/11/24	1.01	Started section 5	Vrishank Kulkarni
12/11/24	1.02	Started Section 7	Jakob Huffman
12/12/24	1.03	Completed Document	Everyone

Table of Contents

Test Case

1. Purpose

The purpose of this document is to create an extensive list of test cases and document them thoroughly.

Things that should be tested include but are not limited to. Valid mathematical expressions like unary operations and excess parentheses, and basic arithmetic. Nested excess parentheses, nested parentheses. Invalid inputs of nonsensical math (such as operators acting on operators, or mismatched parentheses), invalid characters, a mix of invalid characters and numbers. Empty parentheses, expressions of only operators, parentheses surrounding single numbers, excess parentheses surrounding single numbers. And testing of our advanced features like implied multiplication and multiple representations of exponentiation

2. Test case identifier, Test item, Input specifications, Output specifications

Expected Output: Each test case must produce specific outputs that align with the requirements for the arithmetic expression evaluator. The outputs should be precise and adhere to the expected results based on the PEMDAS rules.
Features Required:

Test Case Identifier	Test Item	Test Specification	Expected Output	Actual Output	Pass/Fail
001	Test to check if the program handles valid expressions with addition	3+4	7	7	pass
002	Test to check if the program handles valid expressions with subtraction with parentheses	8-(5-2)	5	5	pass
003	Test to check if the program handles valid expressions with multiplication and division	10*2/5	4	4	pass
004	Test to check if the program handles valid expressions with exponentiation with a double *	2**3	8	8	pass
005	Test to check if the	2^3	8	8	pass

	program handles valid expressions with exponentiation with a ^				
006	Test to check if the program handles valid expressions with mixed operators	4*(3+2)%7-1	5	5	pass
007	Test to check if the program handles valid expressions Complex Addition with Extraneous Parentheses	(((2+3)))+(((1+2)))	8	8	pass
008	Test to check if the program can handle valid expressions with mixed operators and extraneous parentheses	((5*2)-((3/1)+((4%3))))	6	6	pass
009	Test to check if the program handles valid expressions with multiplication and addition	3+5*2	13	13	pass
010	Test to check if the program can handle divisibility by 0	5/0	"Error: Division by "zero"	"Error: Division by zero"	pass
011	Test to check if the program can handle unmatched parentheses.	2+(4*	"Error: mismatched parentheses"	Error: mismatched parentheses	pass
012	Test to check if the program can handle invalid characters.	1abc+2	Error: Invalid character 'a' at position 1. Error: Invalid character 'b' at position 2. Error: Invalid character 'c' at position 3.	Error: Invalid character 'a' at position 1. Error: Invalid character 'b' at position 2. Error: Invalid character 'c' at position 3.	pass
013	Test to check if the program can handle	3^0	1	1	pass

	any number to the power of 0.				
014	Test to check if the program can handle any expressions with nested parentheses with exponents:	(((2**(1+1))+((3-1)**2))/((4/2)%3))	4	4	pass
015	Test to check if the program can handle a combination of extraneous and necessary parentheses	(((((5 - 3))) * (((2 + 1))) + ((2 * 3))))	12	12	pass
016	Test to check if the program can handle extraneous parentheses with division	((6/(3)))/(2)	1	1	pass
017	Test to check if the program can handle combining unary operators with arithmetic operations	−1+2**2−(−3)	6	6	pass
018	Test to check if the program can handle unary negation and addition in parentheses	-(+1)+(+2)	1	1	pass
019	Test to check if the program can handle negation and addition with negated parentheses	-(-(-3)) + (-4) + (+5)	-2	-2	pass
020	Test to check if the program can handle unary negation and exponentiation	+2 ** (-3)	0.125	0.125	pass
021	Test to check if the program can handle combining unary operators with parentheses	-(+2) * (+3) - (-4) / (-5)	-6.8	-6.8	pass

022	Test to check if the program can handle invalid operator sequences	*5+2	"Error: invalid operator sequence"	"Error: invalid operator sequence"	pass
023	Test to check if the program can handle distributive property	5(2+3)	25	25	pass
024	Test to check if the program can handle invalid expressions with mismatched parentheses	(((3+4)-2)+(1)	Mismatched parentheses detected. Check for balanced parentheses	Mismatched parentheses detected. Check for balanced parentheses	pass
025	Test to check if the program can handle invalid expressions with invalid operator usage	((5+2)/(3*0))	Error: Division by zero	Error: Division by zero	pass
026	Test to check if the program can handle single inputs surrounded by parentheses	((1))	1	1	pass
027	Test to check if the program can handle missing operands	((4*2)+(-))	"Error: Invalid operator sequence"	"Error: Invalid operator sequence"	pass
028	Test to check if the program can handle spaces	3 + 4	7	7	pass
029	Test for blank inputs	(blank line / no input)	Cannot do an evaluation with an empty expression	Cannot do an evaluation with an empty expression	pass
030	Test for "QUIT" command	QUIT	Program ends	Program ends	pass

3. Environmental needs

3.1.1 Hardware

In order to run our Arithmetic Expressions Evaluator, you will need the following:

- Processor:
  - Any x86 or x86-64 CPU
- Memory (RAM):
  - OS Required Minimum
  - Recommended: 2GB or more
- Storage:
  - OS Required Minimum
  - 20MB for Arithmetic Expressions Evaluator files
  - 3GB for the terminal to run the program executable

3.1.2 Software

Our C++ code is platform-dependent and is not compatible with Mac*. This is because our code relies on certain libraries from the Windows C++ compiler we used. Additionally, our program is an executable **.exe** file which isn't compatible with Mac. *If you only have the C++ code*, you will need to have a MinGW compiler to deliver an executable. For Windows & Linux users, all that is required to run our Arithmetic Expression Evaluator are the minimum system requirements to have the OS running. Additionally, you will need a basic terminal (e.g. Powershell, Zsh, Bash, etc.) to interact with the program. We recommend running the latest OS provided by Microsoft or your Linux distributor.

*If you would like to run a .exe file MacOS, you will need a VM or Windows executable emulator such as Wine, Whisky, Parallels, VMWare, etc.

3.1.3 Other

If the user intends to edit or integrate the parser into a larger compiler, an IDE such as Visual Code or text editors such as Vim or Nano may be used.

4. Special procedural requirements

1. Special Setup Constraints:

Environment Setup: Ensure the environment has a functional C++ compiler g++ (MinGW) and the required project files, including main.cpp, and lexer.cpp, parser.cpp, evaluator.cpp, and errorhandler.cpp also each file has a header file included. **Input Validation**: Input expressions should be provided as valid strings. Operators on valid and invalid test cases are used to get the expected output. **Resource Allocation**: Allocate sufficient memory to handle large expressions and complex parentheses nesting during evaluation. **Logging Configuration**: Enable detailed logging in the ErrorHandler class to capture error messages for test data.

2. Operator Intervention:

During testing, the tester must input expressions in the command line or provide predefined test inputs via in code. Manual validation is required to confirm whether error messages and outputs match expectations.

3. Output Determination Procedures:

Use the ErrorHandler's displayErrors() function to verify all logged errors are displayed correctly. Ensure that valid expressions produce accurate evaluation results as per the order of operations (PEMDAS).

4. Special Wrap-Up:

Clear any temporary data structures or files created during testing. Reset the testing environment, including clearing error logs in the `ErrorHandler`.

5. Intercase dependencies

No intercase dependencies are present within the logic and execution of this program.