SN^{2}ARK CO.

Arithmetic Parser
Test Case Document

Version 1.2

Revision History

Date	Version	Description	Author
11/20/2023	1.0	Team members started the Test Case Document. They started working on the individual's 10 test cases.	Sophia Jacob, Reeny Huang, Navya Nittala, Kusuma Murthy, Anna Lin, Nimra Syed
11/26/2023	1.1	Each team member completed their 10 individual Test Cases. This will have robust examples that will test the software and code in various ways.	Sophia Jacob, Reeny Huang, Navya Nittala, Kusuma Murthy, Anna Lin, Nimra Syed
11/29/2023	1.2	The team members went through all 61 test cases and explained the purpose of each test case and validated the test cases.	Sophia Jacob, Reeny Huang, Navya Nittala, Kusuma Murthy, Anna Lin, Nimra Syed
12/01/2023	1.3	Removed blue instructions text from the document, added more required details in the needed areas, and finalized this document to be submitted after reviewing all the sections.	Sophia Jacob, Reeny Huang, Navya Nittala, Kusuma Murthy, Anna Lin, Nimra Syed

Table of Contents

Test Case

1. Purpose

The purpose of this project is for SN^{2}ARK to develop a multifunctional arithmetic expression evaluator utilizing C++. The objective is to create a simulation of a calculator, through a program, that can handle expressions as input and calculate the result using mathematical operations like (PEMDAS). Furthermore, SN^{2}ARK will fully integrate the Software Development Process through timely deliverables within the semester. These deliverables include a detailed project plan, a requirements document, a design document aligned with the requirements, a set of test cases, and a fully developed product. As part of the scope of this project, each iteration should be timely and in line with the functionality of the requirements.

This ***Test Case Specification Document*** for the Arithmetic Parser defines a test case for an item that should be tested. The sections of the test case specification shall be ordered in the specified sequence. Additional sections may be included at the end. The Document will validate the user's functionality in the various features that are offered by SN^{2}ARK's project. Section 2 of the ***Test Case Specification Document*** contains the various test cases procured by each team member. This Document displays the functionality of regression testing, allowing the team to test different scenarios, then refactor and debug the source code. This was done throughout the process of creating modules and the Implementation Phase. Finally, in Section 2, the different headers include a Test Case Identifier, which is the number of the Test Case. The Test Item is a brief description of the Test Case. The Input Specification is what expression/input will be tested. The Expected Output Specification and Actual Output Specification will show what the user expects to see versus what is actually shown. The Test Case is validated by a Pass/Fail column. Section 6 will cover any other requirements or Test Cases.

2. Test cases

Table 1

Test Case Identifier	Test Item	Input Specification	Expected Output Specification	Actual Output Specification	Pass/Fail
001	Test to validate if the program handles invalid input of letters through the isValidCharacter method of the Parser module. It also tests the Error Handling Module to see if it will take in the Error and handle the output and exiting of the code gracefully. This test case ensures the error-handling requirement.	abcdefg	Invalid Character Error.	Invalid Character Error.	Pass
002	Test to validate if the	(10+30)(40*50)	Missing or	2000	Pass

	program handles expressions with missing operators through the isMathValid method of the Parser module. This test case ensures the error-handling requirement.		misplaced operands/operators.		
003	Test to validate if the program can handle unary operators not separated by parentheses at the beginning of the expression through the try-catch block in the Main module. This test case ensures that the error-handling requirement is met.	--5+2	Bad Input. Unary operators must be separated by parentheses.	Bad Input. Unary operators must be separated by parentheses.	Pass
004	Test to validate if the program can handle unary operators not separated by parentheses in the middle of the expression through the try-catch block in the Main module. This test case ensures that the error-handling requirement is met.	5--5	Bad Input. Unary operators must be separated by parentheses.	Bad Input. Unary operators must be separated by parentheses.	Pass
005	Test to validate if the program will handle a single operand through the evaluateRemainding method in the Evaluate module. This test case	1	1	1	Pass

	ensures that the evaluation of valid expression requirements is met.				
006	Test to validate that the program can handle unbalanced parentheses through the isBalancedParentheses method in the Parser module. This test case ensures that the combination of extraneous and necessary parentheses and error-handling requirements are met.	((((((5)+7))))	Parentheses Error.	Parentheses Error.	Pass
007	Test to validate that the program follows operator precedence through the evaluateExpression method in the Evaluate module. This test case ensures that evaluation according to operator precedence requirement is met.	(10+2-4*5%10/20^2)	12	12	Pass
008	Test to validate if the program can handle unary operators not separated by parentheses in the middle of the expression through the try-catch block in the Main module. This test case ensures that the valid	5^(--2)	Bad Input. Unary operators must be separated by parentheses.	Bad Input. Unary operators must be separated by parentheses.	Pass

	expression and error-handling requirements are met.				
009	Test to validate if the program can handle unary operators not separated by parentheses in the middle of the expression through the try-catch block in the Main module. This test case ensures that the valid expression and error-handling requirements are met.	5+-2	Bad Input. Unary operators must be separated by parentheses.	Bad Input. Unary operators must be separated by parentheses.	Pass
010	Test to validate if the program is able to catch extraneous decimal points through the isValidCharacter method in the Parser module. This test case ensures that the valid expression and error-handling requirements are met.	0.9.9+1	Invalid Character Error.	Invalid Character Error.	Pass
011	Test to validate nested negative signs within parenthesis. This test case is purposefully designed to check the Evaluate and Tokenizer Module. Tokenizer should give input of -1* when it sees these nested negatives and Evaluate should be	5^(-(-2))	25	25	Pass

	able to read and evaluate it. This test case validates the unary operators and parentheses requirements.				
012	Test to validate that the program handles Division by Zero Errors. This tests to see if the Evaluate Module can successfully catch and handle DivisionByZero errors and if it will exit the code gracefully when computing. This test case validates the bad input handling requirement.	5/(3*2+3-4-5)	Division by Zero Error.	Division by Zero Error.	Pass
013	Test to validate combining unary operators with arithmetic operators and negated parenthesis. This is to wholly test if the Evaluate Module can parse through the expression given by the user, with complex expressions. Tokenizer should successfully give input where the negative unary operator is concatenated with the digit it is associated with, for Evaluate to work properly.	+(-7)^(-3+2)*(+5)-(-(-(-2*7)))+1	14.2857	14.2857	Pass

	This is to validate the unary operator and arithmetic operator requirement.				
014	Test to validate that the program handles modulus by floating point error, since C++ Compilers cannot handle the modulus operator for anything other than integers. This Test Case validates the Evaluate Module to see if it can gracefully catch floating point modulus error and respond by going to the Error Handling Module. This is in accordance with the bad input handling requirement.	5/7%9+11*3+2^5	Modulus Operator requires integers.	Modulus Operator requires integers.	Pass
015	Test unary operators inside and outside parentheses. This is testing the Evaluate and Tokenizer Module to identify if unary and arithmetic operators are separated in terms of their capabilities. This tests the operator and evaluation requirement.	-(+2+(3+5)*9)+10	-64	-64	Pass
016	Test a negative exponent with other operator precedence. This wholly tests the	5^-4 + 11 * 10 - 9 / 33	109.729	109.729	Pass

	Evaluate Module by seeing if it can handle negative exponents in its expression and output, not an integer, but a decimal value. It also tests the functionality and requirement of operator precedence by removing any parenthesis.				
017	Test a misplaced operator nested inside of an expression. This tests the goodInput function for the Parser Module and the Error Handling Module. Before going to the Evaluate Class, it must check if the user has inputted a good expression that can be evaluated. If the Parser Module parses through the expression and sees that there is an error in the user input, then it must go to the Error Handling module to exit the code gracefully. This tests the bad input functionality.	((2*)5/3+9+7)	Missing or misplaced operands/operators.	Missing or misplaced operands/operators.	Pass
018	Test that has all valid operators and extraneous and nested parentheses. This tests the parenthesis-handling requirement from the Evaluate Module and	(5+3)-(2%10)+((2^(7-6))/7)*3	6.85714	6.85714	Pass

	sees if it returns a decimal value when necessary. It tests for unnecessary parenthesis in the expression to check if Evaluate can handle this.				
019	Test to see if a number modulus 0 results in an error (since you cannot divide by 0 to get a remainder). This tests the Evaluate Module and if it can gracefully handle errors while calculating. It should go to the Error Module that will help it gracefully exit the code. This checks the bad input handling requirement.	(3%0)+(10*7)	Division by Zero Error.	Division by Zero Error.	Pass
020	Test to validate if an error message occurs when there is a missing operand. This tests the Error Handling Module and the Parser Module. Again, a user must put good input for it to go to the Evaluate Module by checking with Parser. If there is a misplaced operator, there is an error and must go to the Error Module from Parser. This tests the bad input handling requirement.	(5*2)+(9-3)+7-(7-)	Missing or misplaced operands/operators.	Missing or misplaced operands/operators.	Pass
021	Test to validate if an error message occurs	(5/7)*(3+2)-	Missing or misplaced	Missing or misplaced	Pass

	when an expression ends with an operator that is not just a closing parenthesis. This tests the Error Handling Module and the Parser Module. Again, a user must put good input for it to go to the Evaluate Module by checking with Parser. If there is a misplaced operator, there is an error and must go to the Error Module from Parser. This tests the bad input handling requirement.		operands/operators.	operands/operators.	
022	Test to validate if the program catches the error due to invalid character '#'. This utilizes the Parser Module that checks if the expression is valid such as invalid characters. It also tests whether or not the Error Handling module can output this expression and exit gracefully. This tests the bad input handling requirement.	8+(6#3)	Invalid Character Error.	Invalid Character Error.	Pass
023	Test to validate if the program catches the operator error when it attempts to divide by 0. This tests the evaluate expression in the Evaluate Module. This tests to	(9+7)/((5*1)+(-5))	Division by Zero Error.	Division by Zero Error.	Pass

	see if the Evaluate Module can successfully catch and handle DivisionByZero errors and if it will exit the code gracefully when computing. This test case validates the bad input handling requirement.				
024	Test to validate if the unary '+' works in the Evaluate Module. This is in accordance with the unary operator and subtraction operator of the Evaluate Module.	+(+4-7)	-3	-3	Pass
025	Test to validate if the program correctly evaluates this equation with the unary '-' works. Having consecutive unary operators together is bad input because it needs to specify whether or not we want subtraction or unary operators by parenthesis. This tests the Main Module to see if it will catch such errors accordingly and exit gracefully. This test case validates the bad input handling requirement.	9--(1225)	Bad Input. Unary Operators must be separated by parentheses.	Bad Input. Unary Operators must be separated by parentheses.	Pass

026	Test to validate if the program catches the error of dividing by 0. This tests the evaluate expression in the Evaluate Module. This tests to see if the Evaluate Module can successfully catch and handle DivisionByZero errors and if it will exit the code gracefully when computing. This test case validates the bad input handling requirement.	(9)%(1+1-2)	Division by Zero Error.	Division by Zero Error.	Pass
027	Test to validate combining unary operators with arithmetic operators and negated parenthesis. Test getPrecedence, isBalancedParenthes, and evaluate function in both the Parser and Evaluate modules.	+(-2) * (-3) – ((-4) ^ (+5))	1030	1030	Pass
028	Test to validate nested parentheses with exponent and modulo. Tests the getPrecedence and isBalanced Parentheses function in the Parser and Evaluate Module.	2^(8%(79-45-9-2))	256	256	Pass
029	Test to validate unary negation and addition in parentheses Test unary signs in the	-(+2) + (+2)	0	0	Pass

	Evaluate Module				
030	Test to validate exponentiation and addition/subtraction. Test the exponential requirement in the evaluate module.	2^5+3-9	26	26	Pass
031	Test to validate precedence and multiplication and subtraction. Test precedence with getPrecedence in the Evaluate module. The input should be evaluated following PEMDAS and the Evaluate module should parse through the input without parenthesis by solely following operator precedence rules.	92/5-4+9*2	32.4	32.4	Pass
032	Test to validate parentheses without operators. This is to validate missing operators or operands. Test the isMathValid in the Parser Module. It also checks to ensure that the module checks that valid input contains operators or parentheses between unary operators.	(5-6(5))	Missing or misplaced operands/operators.	Missing or misplaced operands/operators.	Pass
033	Test to validate exponential by 0 and operator precedence. This is to validate unary operators with exponents.	(5-4+-1^0)	Bad Input. Unary Operators must be separated by parentheses.	Bad Input. Unary Operators must be separated by parentheses.	Pass
034	Test to validate operator precedence.	(1+5)^2/3/-2	-6	-6	Pass

	This is to check the functionality of the				
035	Test to validate if the program can handle unary operators not separated by parentheses in the middle of the expression through the try-catch block in the Main module. This test case ensures that the error-handling requirement is met.	-1--(-90)	Bad Input. Unary Operators must be separated by parentheses.	Bad Input. Unary Operators must be separated by parentheses.	Pass
036	Test to validate error message when missing operands between operators. This is to validate the missing operators or operands within the parser module. Test to validate if the program can handle unary operators not separated by parentheses in the middle of the expression through the try-catch block in the Main module. This test case ensures that the error-handling requirement is met.	(5-(--5(+12)))	Bad Input. Unary Operators must be separated by parentheses.	Bad Input. Unary Operators must be separated by parentheses.	Pass
037	Test to validate invalid exponential operator. This is to validate the missing operators or operands within the parser module. This tests the Error	5-1**2	Missing or misplaced operands/operators.	Missing or misplaced operands/operators.	Pass

	Handling Module and the Parser Module. Again, a user must put good input for it to go to the Evaluate Module by checking with Parser. If there is a misplaced operator, there is an error and must go to the Error Module from Parser. This tests the bad input handling requirement.				
038	Test to validate if spacing still provides good input This is to validate the Tokenizer and Evaluate module. This tests the subtraction, parenthesis, and good input requirements.	((5-1))	4	4	Pass
039	Test to validate nested operators with modulo. This is to validate the Parser module and ensure that unary operators are separated by an operand or parenthesis. Test to validate if the program can handle unary operators not separated by parentheses in the middle of the expression through the try-catch block in the Main module. This test case ensures that the error-handling	-(5(2%(5+1--4)))	Bad Input. Unary Operators must be separated by parentheses.	Bad Input. Unary Operators must be separated by parentheses.	Pass

	requirement is met.				
040	Test to validate nested operators within the Parser module. This test case ensures that the unary operator and nested parentheses requirement is met.	-(-(+(-(50-11))))	-39	-39	Pass
041	Test to validate division by 0. This is to validate the isMathValid in the Parser module. This tests to see if the Evaluate Module can successfully catch and handle DivisionByZero errors and if it will exit the code gracefully when computing. This test case validates the bad input handling requirement.	(5^2+1)/(12*2-6*4)	Division by Zero Error.	Division by Zero Error.	Pass
042	Tests to see if equations with variables are valid input. Tests the isMathValid method in the Parser module. Test to validate if the program handles invalid input of letters through the isValidCharacter method of the Parser module. It also tests the Error Handling	-(5x)^3	Invalid Character Error.	Invalid Character Error.	Pass

		Module to see if it will take in the Error and handle the output and exiting of the code gracefully. This test case ensures the error-handling requirement.				
	043	Test to see if '++ is invalid. Tests the try and catch in the mian is catching the consecutive '++' and Having consecutive unary operators together is bad input because it needs to specify whether or not we want subtraction or unary operators by parenthesis. This tests the Main Module to see if it will catch such errors accordingly and exit gracefully. This test case validates the bad input handling requirement.	-(-(-5++3))	Bad Input. Unary Operators must be separated by parentheses.	Bad Input. Unary Operators must be separated by parentheses.	Pass
	044	Checks if large floats are valid inputs, and they return a scientific number Test the Evaluate module to see if floats are valid inputs.	200/0.000001*3	1.6e+08	1.6e+08	Pass
	045	Checks if it accounts for a zero as a float	200*3+4/0.000000	Division by Zero Error.	Division by Zero Error.	Pass

3. **Test item**

Refer to *Table 1*.

4. **Input specifications**

All input values are specified in *Table 1*.
Databases: None
Files: None
Terminal Messages: "Enter your Expression:", "Missing or misplaced operands/operator", "Division by Zero Error", "Invalid Character", "Bad Input", "Modulus operator requires integers.", "Bad Input. Unary operators must be separated by parentheses.", "Parentheses Error."
Memory Resident Areas: None
Values Passed by Operating System: the return value of expression

5. **Output specifications**

Refer to *Table 1*.

6. **Environmental needs**

6.1.1 *Hardware - NA*

6.1.2 *Software - NA*

6.1.3 *Other*

A requirement to run the code for the Arithmetic Parser Project is to have a laptop or functioning device that allows for a Linux-based environment, or can connect to Cycle Servers. This will allow one to run the executable from the code to test various Arithmetic Expressions from the terminal.

7. **Special procedural requirements - NA**

8. **Intercase dependencies - NA**