Arithmetic Expression Evaluator

Test Cases Document

Version <1.5>

Revision History

Date	Version	Description	Author
<29/11/23>	<1.1>	Added purpose	Kyle Spragg
<29/11/23>	<1.2>	Added test case identifier	Landon Bever
<29/11/23>	<1.3>	Added test item	Blake Smith
<29/11/23>	<1.4>	Added input specifications	Hale Coffman
<29/11/23>	<1.5>	Added output specifications	Aidan Ingram

Table of Contents

Test Case

1. Purpose

The purpose of the Test Cases Document is to verify if the calculator will function properly and produce accurate results for several input scenarios. The goal is to ensure two things: reliability and accuracy. If the calculator does not meet the expectations of these requirements, then there must be more debugging and development on the software.

Test Case ID	Test Case Description	Test Data	Expected Results	Actual Results	*Pass/Fail*
Add_001	***Test arithmetic operator (+)***	**4 + 5**	**9**	**9**	***Pass***
Sub_001	***Test arithmetic operator (-)***	**10 - 5**	**5**	**5**	***Pass***
Mult_001	***Test arithmetic operator (*) for multiplication***	**2 * 3**	**6**	**6**	***Pass***
Div_001	***Test arithmetic operator (/) for division***	**25 / 5**	**5**	**5**	***Pass***
Mod_001	***Test arithmetic operator (%) for modulo***	**20 % 2**	**0**	***0***	***Pass***
Exp_001	***Test arithmetic operator (^) for exponentials***	**2 ^ 3**	**8**	***8***	***Pass***
Neg_001	***Test unary negation***	**-(-(3)) + 1**	**4**	**4**	***Pass***
MixAS_001	***Test addition and subtraction cohesion***	**5 + 3 - 4**	**4**	**4**	***Pass***
MixMD_001	***Test multiplication and division cohesion***	**6 * 7 / 25**	**1.68**	**1.68**	***Pass***
MixALL_001	***Test all operators and cohesion***	**8 * 9 + 4 – 3 / 10**	**75.7**	**75.7**	***Pass***
Paren_001	***Test parenthesis***	**(2+1) + (1)**	**4**	**4**	***Pass***
Paren_002	***Test extraneous parentheses***	**(((2+4))) + ((2))**	**8**	**8**	***Pass***

Err_001	***Test operator error***	***5 + 1**	**Not enough operands for operator ***	**Not enough operands for operator ***	***Pass***
Err_002	***Test mismatched parenthesis***	**((2+1)**	**Unmatched parenthesis in expression**	**Unmatched parenthesis in expression**	***Pass***
Err_003	***Unsupported symbol/char***	**(2 @ 3)**	**Unrecognized arithmetic symbol '@'**	**Unrecognized arithmetic symbol '@'**	***Pass***

2. Test case identifier

Test Case Identifiers:

- **Add_001**
- **Sub_001**
- **Mult_001**
- **Div_001**
- **Mod_001**
- **Exp_001**
- **Neg_001**
- **MixAS_001**
- **MixMD_001**
- **MixALL_001**
- **Paren_001**
- **Paren_002**
- **Err_001**
- **Err_002**
- **Err_003**

3. Test item

For each test case identifier:

- **Add_001** → properly testing the addition feature of the calculator.
- **Sub_001** → properly testing the subtraction feature of the calculator.
- **Mult_001**→ properly testing the multiplication feature of the calculator.
- **Div_001** → properly testing the division feature of the calculator.
- **Mod_001** → properly testing the modulo feature of the calculator.
- **Exp_001** → properly testing the exponentiation feature of this calculator.
- **Neg_001** → properly testing the negation feature of the calculator.

- **MixAS_001 →** properly testing mixed addition and subtraction features of the calculator.
- **MixMD_001 →** properly testing mixed multiplication and division features of the calculator.
- **MixALL_001 →** properly testing mixed operations of all features of the calculator above.
- **Paren_001 →** properly testing intentional parentheses in expressions in the calculator.
- **Paren_002 →** properly testing extraneous parentheses in expressions in the calculator.
- **Err_001 →** properly testing error-handling regarding improper operators in the calculator.
- **Err_002 →** properly testing error-handling regarding mismatched parentheses in the calculator.
- **Err_003 →** properly testing error-handling regarding improper symbols in the calculator.

All these test cases are necessary to ensure the proper functionality of our calculator, as defined in the requirements specification document <02-Software-Requirements-Spec>

4. Input specifications

Input required by each case included the numbers being calculated and the specific operator(s). It could also include parenthesis or error handling depending on the test case.
For each test case identifier:

- **Add_001 → include numbers 4, and 5, and the '+' operator.**
- **Sub_001 → include numbers 10, and 5, and the '-' operator.**
- **Mult_001 → include numbers 2, and 3, and the '*' operator.**
- **Div_001 → include numbers 25, and 5, and the '/' operator.**
- **Mod_001 → include numbers 20, and 2, and the '%' operator.**
- **Exp_001 → include numbers 2, and 3, and the '^' operator.**
- **Neg_001 → include numbers 3, and 1, and the '-' unary negation operator.**
- **MixAS_001 → include numbers, 5, 4, and 3 and the '+' and '-' operators.**
- **MixMD_001 → include numbers 25, 7, and 6, and the '*' and '/' operators.**
- **MixALL_001 → include numbers 8, 9, 4, 3, and 10 and the '+', '-', '*', and '/' operators.**
- **Paren_001 → include numbers 2, and 1, and the '+' operator as well as '(' and ')' operators.**
- **Paren_002 → include numbers 2, and 4, and the '+' operator as well as extraneous '(' and ')' operators.**
- **Err_001 → include numbers 5, and 1, as well as the extraneous '*' operator to break the calculation.**
- **Err_002 → include numbers 2, and 1, as well as the '+' and mismatched '(' and ')' operators to break the calculation.**
- **Err_003 → include numbers 2, and 3, and an unrecognized '@' symbol to break the calculation.**

For these specific test cases, there might not be a direct involvement of files or databases unless the calculator is part of a larger system. Memory usage and terminal messages would depend on the implementation of the calculator and its user interface.

In a testing-exclusive environment, the code below was written as the input specifications to evaluate the program. The test function takes an expression, an expected value, and case number. It returns nothing and prints out a statement showing success or failure.

```
void test(string expression, float expected, int casenum) {
// Simple function we made to try and test every expected case that should
return a value
unordered_map<string, int> opMap;
opMap["^"] = 4;
opMap["*"] = 3;
opMap["/"] = 3;
opMap["%"] = 2;
opMap["+"] = 1;
opMap["-"] = 1;
opMap["("] = -1;
opMap[")"] = -1;

try {
vector<string> tokens = Tokenize(expression, opMap);
vector<string> postfix = InfixToPostfix(tokens, opMap);
float actual = CalculatePostfix(postfix, opMap);

// Set column width to 10
cout << left << setw(5) << "CASE " << setw(5) << casenum << ": " << setw(20)
<< expression << " | ";

if (expected != actual) {
cout << "FAILURE. Expected Value: " << setw(10) << expected << " Actual Value:
" << actual << endl;
} else {
cout << "PASSED. Expected Value: " << setw(10) << expected << " Actual Value:
" << actual << endl;
}
} catch (const exception &e) {
cerr << left << setw(5) << "CASE " << setw(5) << casenum << ": " << setw(20)
<< expression << " | ";
cerr << "FAILED. Unexpected Error: " << e.what() << endl;
} catch (...) {
cerr << left << setw(5) << "CASE " << setw(5) << casenum << ": " << setw(20)
<< expression << " | ";
cerr << "FAILED. Unexpected Error." << endl;
}
}
```

5. Output specifications

For each test case identifier:

- **Add_001 → 9**
- **Sub_001 → 5**
- **Mult_001 → 6**
- **Div_001 → 5**
- **Mod_001 → 0**
- **Exp_001 → 8**
- **Neg_001 → 4**
- **MixAS_001 → 4**
- **MixMD_001 → 1.68**
- **MixALL_001 → 75.7**
- **Paren_001 → 4**
- **Paren_002 → 8**
- **Err_001 → Error: Not enough operands for operator ***
- **Err_002 → Error: Unmatched parenthesis in expression**
- **Err_003 → Error: Unrecognized arithmetic symbol '@'**
-

The final output of our test program is provided below, directly from the testing data above. These test cases were the ones chosen that best evaluated the project requirements and tried to fully encapsulate the scope. We avoided the 15 cases presented in the initial project requirements, to avoid repetition.

```
CASE 1    : 4 + 5               | PASSED. Expected Value: 9          Actual Value: 9
CASE 2    : 10 - 5              | PASSED. Expected Value: 5          Actual Value: 5
CASE 3    : 2 * 3               | PASSED. Expected Value: 6          Actual Value: 6
CASE 4    : 25 / 5              | PASSED. Expected Value: 5          Actual Value: 5
CASE 5    : 20 % 2              | PASSED. Expected Value: 0          Actual Value: 0
CASE 6    : 2 ^ 3               | PASSED. Expected Value: 8          Actual Value: 8
CASE 7    : -(-(3)) + 1         | PASSED. Expected Value: 4          Actual Value: 4
CASE 8    : 5 + 3 - 4           | PASSED. Expected Value: 4          Actual Value: 4
CASE 9    : 6 * 7 / 25          | PASSED. Expected Value: 1.68       Actual Value: 1.68
CASE 10   : 8 * 9 + 4 - 3 / 10  | PASSED. Expected Value: 75.7       Actual Value: 75.7
CASE 11   : (2 + 1) + (1)       | PASSED. Expected Value: 4          Actual Value: 4
CASE 12   : (((2 + 4))) + ((2)) | PASSED. Expected Value: 8          Actual Value: 8
CASE 13   : * 5 + 1             | FAILED. Unexpected Error: Not enough operands for operator *
CASE 14   : ((2 + 1)            | FAILED. Unexpected Error: Unmatched parenthesis in expression
CASE 15   : 2 @ 3               | FAILED. Unexpected Error: Unrecognized arithmetic symbol '@'
```

6. Environmental needs

6.1.1 Hardware (nothing particular for the arithmetic expression project)

N/A

6.1.2 Software (nothing particular for the arithmetic expression project)

N/A

6.1.3 Other

It is pertinent to note that the test cases present in this document are commented out, but present, in the source code.

7. Special procedural requirements

N/A

8. Inter-case dependencies

N/A