Kansas Instruments

Arithmetic Expression Evaluator

User's Manual

Version 1.05

Revision History

Date	Version	Description	Author
12/5/2024	1.01	Completed section 1 and section 2	Emma Du
12/8/2024	1.02	Completed section 5, section 7, section 8	Emma Du
12/11/2024	1.03	Completed section 3	Emma Du
12/12/2024	1.04	Completed section 4	Phoenix Brehm
12/12/2024	1.05	Made final revisions and edits	Everyone

Table of Contents

User Manual

1. Purpose

The purpose of this project is to develop a multifunctional arithmetic expression evaluator using C++. The objective is to create a simulation of a calculator through a program that can handle input expressions and calculate the result using PEMDAS mathematical operations. Our project also fully integrates the Software Development Process through timely deliverables completed throughout the semester. These deliverables include a project plan, requirements document, design document, a set of test cases, and a finalized product.

The purpose of the User Manual Document is to provide a comprehensive guide to help the user efficiently use the calculator's functionalities. The manual provides clear instructions regarding accessing the source code, inputting commands, using various operators, and navigating the user interface (while providing examples). Troubleshooting guidance is also provided to allow users the ability to operate the arithmetic parser with ease. By providing these step-by-step instructions and guidance on how to handle errors, our manual allows users to comprehend the calculator's features.

2. Introduction

The arithmetic parser is a user-friendly arithmetic calculator that evaluates mathematical expressions. It was developed using C++ and implemented for use by Windows and Linux devices in a Powershell/Bash terminal. The main target audience of our product is students, TAs, and Professors within the EECS department. The software consists of back-end features such as a lexer, parser, evaluator, and error handler, and the front-end features consist of receiving valid user input and generating error handling messages. This parser includes advanced features such as PEMDAS and returning user-specific error messages.

The code is in a public GitHub repository. Users can use the software by cloning our code into their local machines and running the code in a Linux Terminal. The software can be launched by typing g++ *.cpp -o output.exe and ./output.exe into the terminal.

3. Getting started

The source code is in Github, using the public repository link
https://github.com/KitMagar/Arithmetic-Expression-Evaluator

1. Use a laptop or device that has at least one of the following: a terminal, the ability to connect a terminal or cycle server, a Linux-based environment, or IDEs such as VS Code.
2. Copy paste the above link into a web browser and download the Github repository onto your local device by clicking on the green button, Code, and then clicking on the Download Zip File button.
3. Go to the folder that the Zip file has been saved to and unzip the file by right-clicking.
4. Go to the terminal in VS Code or a cycle server, and open the terminal. cd into the ProjectFiles.
5. Type ./output.exe into the terminal line and hit enter, or simply double click output.exe within the ProjectFiles folder
    a. If output.exe fails to start, go into the "in_case" folder and run program.exe instead
    b. Type `g++ *.cpp -o output.exe -static-libgcc -static-libstdc++` (compiled using MinGW) into the terminal line and hit enter, if output.exe is not present, to compile the software.
6. To enter the expression, type in a mathematical expression that could be inputted into a scientific calculator and hit enter. The correct output should display, or if invalid input is entered, an error message should be displayed.

4. Advanced features

One of the advanced features present in the program is the ability to allow distribution/implicit multiplication. Our program can evaluate expressions such as 4(3+7) despite the lack of operator between 4 and the parentheses. As well as allowing the use of '**' and '^' for exponentiation. Although these examples appear in the project files as "Invalid Input", our advanced features purposefully allow for these to become valid inputs.

5. Troubleshooting

Possible errors include:

1. Compilation Errors
    a. Make sure to have all of the files: lexer.cpp, lexer.h, parser.cpp, parser.h, evaluator.cpp, evaluator.h, errorhandler.cpp, errorhandler.h. Any errors from not having the correct files can be solved by downloading from GitHub.
    b. Make sure to have a C++ compiler and a terminal to run the software. This would ideally be Linux-based and can connect to the KU Cycle Servers
    c. Make sure to be in the correct directory that contains the files. Be careful to check where you download and place your file. Make sure to navigate to the folder for the product to run.
    d. Copying and pasting expressions into the command line can sometimes result in errors. If you encounter this error, it would be best to type in the expression manually.
2. Consistent Errors from Messages in Terminal
    a. These errors can be solved by:
        i. Ensuring parentheses are matched → Ex. "2 * (4+3-1" has unbalanced parentheses and should be rewritten as "2 * (4+3-1)"
        ii. Ensuring operators have operands → Ex. * 5 + 2 lacks an operand on the left
        iii. Ensure correct operator usage → Ex. 4/0 divides by zero, which is undefined in mathematics
        iv. Ensuring there are no mismatched parentheses → Ex. (((3+4) - 2) + (1) is missing a closing parentheses
        v. Ensuring there is not invalid operator usage → Ex. ((5+2) / (3*0)) attempts to divide by zero, which is mathematically undefined
        vi. Ensuring there is not invalid operator sequence → Ex. ((2-) 1+3) contains an operator "-" without a valid operand on its left
            1. Another common example would be unclear unary operations like 3**-2, this is unclear if you are requesting subtraction or a negative. To perform this operation use: 3**(-2) instead.
        vii. Ensuring there are no missing operands → Ex. ((4*2) + (-)) is missing an operand after the "-" operator
        viii. Ensuring there are no invalid characters → Ex. ((7*3)+2!) is not a valid operator in this context

6. Examples

- Addition: Can be performed to any combination of integer and decimal values

```
Welcome to the arithmetic Evaluator
When you're ready to quit type "QUIT"

Enter Arithmetic Expression: 3+9

12
```

- Subtraction: Can be performed to any combination of integer and decimal values

```
Welcome to the arithmetic Evaluator
When you're ready to quit type "QUIT"

Enter Arithmetic Expression: 12.4-8.1

4.3
```

- Multiplication: Can be performed to any combination of integer and decimal values. Distribution in multiplication can also be used (one of our advanced features)

```
Welcome to the arithmetic Evaluator
When you're ready to quit type "QUIT"

Enter Arithmetic Expression: 37*1.4

51.8
```

```
Welcome to the arithmetic Evaluator
When you're ready to quit type "QUIT"

Enter Arithmetic Expression: 5(1+3)

20
```

- Division: Can be performed to any combination of integer and decimal values as long as the divisor is not zero

```
Welcome to the arithmetic Evaluator
When you're ready to quit type "QUIT"

Enter Arithmetic Expression: 16/2

8
```

- Modulus: Can be performed to any combination of integer and decimal values as long as the second operand is not zero

```
Welcome to the arithmetic Evaluator
When you're ready to quit type "QUIT"

Enter Arithmetic Expression: 13%6

1
```

- Exponent: Can be performed to any combination of integer and decimal values. ** or ^ (one of our advanced features) can be used

```
Welcome to the arithmetic Evaluator
When you're ready to quit type "QUIT"

Enter Arithmetic Expression: 5^2

25
```

```
Welcome to the arithmetic Evaluator
When you're ready to quit type "QUIT"

Enter Arithmetic Expression: 5**2

25
```

- Parentheses: Ensure all parentheses are closed and balanced (equal number of opening and closing

parentheses)

```
Welcome to the arithmetic Evaluator
When you're ready to quit type "QUIT"

Enter Arithmetic Expression: (8+3)-(5*2)

1
```

- Unary operators: Ensure value along with the unary operator is in parentheses

```
Welcome to the arithmetic Evaluator
When you're ready to quit type "QUIT"

Enter Arithmetic Expression: +(-2)*(-3)-((-4)/(+5))

6.8
```

- Combination of all operators

```
Welcome to the arithmetic Evaluator
When you're ready to quit type "QUIT"

Enter Arithmetic Expression: (((2**(1+1))+((3-1)^2))/((4/2)%3))

4
```

7. Glossary of terms

1. Lexer - A component of the software that reads the input arithmetic expression and breaks it down into smaller units called tokens, such as numbers, operators, and parentheses.
2. Parser - The part of the software that interprets the tokens provided by the lexer and organizes them into a structure that reflects the mathematical hierarchy of operations (following PEMDAS rules).
3. PEMDAS - An acronym representing the order of operations in mathematics: Parentheses, exponents, multiplication and division, addition and subtraction (from left to right). It ensures the proper evaluation of complex expressions.
4. Operand - A value (can be number or variable) upon which an operator performs an operation. For example, in the expression 3 + 5, the numbers 3 and 5 are operands.

8. FAQ

Q: What operations are invalid?

A: Operations involving consecutive operations such as "++" or "--" are considered invalid. However, operations such as "**" or "*+" or "/-" are considered valid. Having a unary operator paired with a regular operator is considered valid input.

Q: Does the arithmetic parser handle nested expressions?

A: Yes, the arithmetic parser handles complex nested expressions by allowing the user to utilize parentheses to set operator precedence.

Q: Can you perform complex integrations that are more than one line?

A: Yes, there is no limit as to how long the input expression must be, as long as the inputted expression follows the valid expression rules.

Q: How long does it take to install the software and run the software?

A: The software runs within 10 ms. The installation and set-up process is very easy and user friendly, and requires little time.