NUMBERCRAFTERS

Arithmetic Expression Evaluator

User's Manual

Version <1.5>

Revision History

Date	Version	Description	Author
<02/12/23>	<1.1>	Added purpose and introduction	Landon Bever
<02/12/23>	<1.2>	Added glossary and FAQ	Blake Smith
<03/12/23>	<1.3>	Added Troubleshooting	Kyle Spragg
<03/12/23>	<1.4>	Added Examples of Uses	Aidan Ingram
<03/12/23>	<1.5>	Added Getting Started and Advanced Features	Hale Coffman

Table of Contents

Test Case

1. Purpose

The user manual for the arithmetic expression evaluator aims to serve as an accessible and comprehensive guide, empowering users to efficiently utilize the features of the program. This manual is designed to demystify the process of interacting with the software, providing step-by-step instructions and examples for users at various levels of familiarity with arithmetic expressions and C++ programming.

2. Introduction

This software is a powerful tool designed to swiftly and accurately evaluate arithmetic expressions entered through the terminal. Whether you are a seasoned programmer or someone new to coding, this user-friendly tool simplifies the process of performing mathematical operations with ease.

Key Features:

1. **Supported Operations:** The software accommodates a variety of mathematical operations, including addition, subtraction, multiplication, division, modulo, exponentiation, and the use of parentheses.
2. **Handling of Positive and Negative Numbers:** It seamlessly manages positive and negative numbers, providing a robust solution for expressions involving both integers and floating-point values.
3. **User-Friendly Terminal Interface:** The software adopts a straightforward approach, enabling users to input expressions directly into the terminal. The output is then promptly displayed, eliminating the need for complex command structures.

Installation and Execution:

1. **Download the Software:** Begin by downloading the software package from the designated source.
2. **Extract the Files:** Extract the downloaded files to a directory of your choice.
3. **Open Terminal:** Navigate to the directory where the software is extracted and open the terminal.
4. **Compile the Code:** Use a C++ compiler to compile the source code. For example, you can use the following command: g++ v.1.2.0.cpp -o v.1.2.0
5. **Run the Program:** Execute the compiled program in the terminal with the following command: ./v.1.2.0
6. **Using the Software:** Upon execution, you should be prompted to enter an expression in the terminal. Examples can be seen in section 6 of this document. The software will process the expression and promptly display the result in the terminal.

3. Getting started Hale

1. **Entering An Expression**: Once the software has been downloaded and executed, an expression must be entered into the input prompt to be solved. This expression can contain any of these operators, +, -, *, /, ^. Also, the expression can utilize parenthesis to set an order of evaluation. If no parenthesis is present, the expression will be evacuated with exponents first, then multiplication and division, and finally addition and subtraction (PEMDAS).
2. **Executing the Expression**: After correctly inputting your expression into the calculator, simply press enter and the software will read and evaluate the expression.

3. **Error Handling**: If the expression entered is incorrectly inputted in any way, the software will throw an error in relation to what is especially wrong with the expression. If this happens, the software must be re-run with the expression entered correctly.
4. **Common Errors: (See Troubleshooting for more)**
    a. Not Enough Operands: If this error is thrown, there exists an operator with too few operands in the expression. Make sure to check all of operators and operands connected to those operators.
    b. Unmatched Parenthesis in Expression: If this error is thrown, there exists either too many or too few open, or close, parenthesis. Make sure to check all the opening parenthesis have exactly one closing parenthesis to match the parenthesis.
    c. Unrecognized Arithmetic Symbol: If this error is thrown, there exists a character in-between two operands that is not one of the recognized arithmetic symbols our software is built to handle, +, -, *, /, ^. Make sure to check if all the operators are compatible with the software.

4. Advanced features Hale

The software does not contain any advanced features currently. Please to refer to any of the team members found in the 'README' document, in the GitHub, with feature suggestions.

5. Troubleshooting Hale

1. **Getting the Software to Run:**
    a. Downloading the Code:
        i. Make sure to find version v.1.2.0.cpp on the GitHub in the Version History folder.
        ii. From there, there should be a copy raw file and download file in the top right of the page.
        iii. Either copy and paste the code into your desired IDE or download and upload it to the IDE.
    b. Running the Code:
        i. If having issues with running the code, it could be a problem with your compiler or IDE:
            1. Check Compiler Settings:
                a. Ensure that your compiler settings are correct for your project. Verify that the selected language standard, compiler flags, and optimization settings are appropriate.
            2. Clean and Rebuild:
                a. Sometimes, build artifacts or intermediate files may be corrupted. Perform a clean build by deleting these files and then rebuilding the project.
            3. Check IDE Configuration:
                a. Verify that your IDE is configured properly. Ensure that the paths to compiler executables and necessary tools are set correctly in the IDE settings.
            4. Update IDE and Compiler:

a. Ensure that you are using the latest version of both your IDE and compiler. Updates may include bug fixes and improvements that could resolve compatibility issues.

5. Check Build Output:
    a. Review the build output in the IDE console or build log. Look for any error messages or warnings related to the compilation process.
6. Reconfigure Project:
    a. If you've made changes to your project structure or added new files, make sure the project configuration is updated to include these changes.
7. Verify Include Paths:
    a. If your project relies on external libraries or headers, confirm that the include paths are correctly set in your project settings.
8. Check Library Linking:
    a. Ensure that library linking is set up correctly. Confirm that the necessary libraries are specified in the project settings and that the linker can find them.
9. Examine Compiler Output Options:
    a. Check compiler output options in your IDE. Ensure that debugging information is included if you need to use debugging tools and review other relevant output options.
10. IDE-Specific Debugging Tools:
    a. Familiarize yourself with the debugging tools provided by your IDE. Learn how to set breakpoints, inspect variables, and step through the code to identify issues.
11. IDE Extensions and Plugins:
    a. If you're using IDE extensions or plugins, make sure they are compatible with your IDE version and are not causing conflicts.
12. Check IDE Console for Error Messages:
    a. If there are issues during the build process, the IDE console often provides additional information. Carefully examine any error messages or warnings that appear.

2. Problems when Evaluating:

a. If having issues with evaluating your expression it could be one of these things:

i. Not Enough Operands: If this error is thrown, there exists an operator with too few operands in the expression. Make sure to check all of operators and operands connected to those operators.

ii. Unmatched Parenthesis in Expression: If this error is thrown, there exists either too many or too few open, or close, parenthesis. Make sure to check all the opening parenthesis have exactly one closing parenthesis to match the parenthesis.

iii. Unrecognized Arithmetic Symbol: If this error is thrown, there exists a character in-between two operands that is not one of the recognized arithmetic symbols our software is built to handle, +, -, *, /, ^. Make sure to check if all the operators are compatible with the software.

6. Examples

1. Simple Addition:
   - Input: 2 + 3
   - Result: 5

2. Multiplication and Parentheses:
   - Input: 4 * (6 - 2)
   - Result: 16

3. Exponentiation:
   - Input: 2^3
   - Result: 8

4. Floating-Point Numbers:
   - Input: 1.5 + 2.5
   - Result: 4

5. Negative Numbers:
   - Input: 8 - (-3)
   - Result: 11

6. Complex Expression:
   - Input: 5 * (3 + 2) / 4^2
   - Result: 1.5625

7. Modulo Operation:
   - Input: 17 % 5
   - Result: 2

8. Multiple Operations in a Sequence:
   - Input: 3 + 5 * 2 - 4 / 2
   - Result: 11

9. Exponentiation with Parentheses:
   - Input: (2 + 3)^2
   - Result: 25

7. Glossary of terms

1. Arithmetic Expression:
   - Definition: A combination of numbers and operators (such as +, -, *, /) arranged to represent a mathematical computation.

2. Terminal:
   - Definition: A command-line interface where users can interact with the program by typing commands and receiving textual output.

3. Programming Language (C++):
   - Definition: C++ is a general-purpose programming language that provides low-level memory manipulation

features combined with high-level abstractions, making it suitable for various applications, including software development.

4. Compiler:
   - Definition: A program that translates human-readable source code into machine code or an intermediate code that can be executed by a computer.

5. Source Code:
   - Definition: The human-readable version of a computer program, typically written in a programming language like C++, before it is compiled into machine code.

6. Execute/Run:
   - Definition: To start the execution of a compiled program, resulting in the computer carrying out the instructions specified in the source code.

7. Directory:
   - Definition: A file system container that holds files and other directories. It provides a way to organize and store files on a computer.

8. Compilation:
   - Definition: The process of converting human-readable source code into machine-readable code by a compiler.

9. Floating-Point Numbers:
   - Definition: Numbers that contain a decimal point or use scientific notation, allowing representation of a wide range of values, including fractional ones.

10. Modulo Operation:
    - Definition: A mathematical operation that returns the remainder when one number is divided by another.

11. Exponentiation:
    - Definition: The operation of raising a base to the power of an exponent, represented as "base^exponent."

12. Parentheses:
    - Definition: Symbols "(" and ")" used to group expressions and alter the standard order of operations.

13. User-Friendly:
    - Definition: Designed to be easily understood and used by people, especially those with little or no technical expertise.

14. Command-Line Interface (CLI):
    - Definition: A text-based interface where users interact with the software by typing commands into a terminal.

15. Infix Notation:
    - Definition: A way of writing mathematical expressions where the operators are written between their operands. For example, `a + b`.

16. Postfix Notation:
    - Definition: A way of writing mathematical expressions where the operators come after their operands. For example, `a b +`.

17. Stack:
    - Definition: A data structure that follows the Last In, First Out (LIFO) principle, where the last element added is

the first one to be removed.

18. Operator Precedence:
- Definition: A set of rules that define the order in which operators are evaluated when an expression is parsed.

8. FAQ

1. What is the purpose of the arithmetic expression evaluator?

The arithmetic expression evaluator is a software tool designed to swiftly and accurately evaluate mathematical expressions entered through the terminal. Whether you are a seasoned programmer or new to coding, this user-friendly tool simplifies the process of performing mathematical operations with ease.

2. What operations are supported by the software?

The software accommodates a variety of mathematical operations, including addition, subtraction, multiplication, division, modulo, exponentiation, and the use of parentheses.

3. How do I install and run the arithmetic expression evaluator?

Please refer to the "Installation and Execution" section of the user manual for detailed instructions on downloading, extracting, compiling, and running the software.

4. Can the evaluator handle both positive and negative numbers?

Yes, the software seamlessly manages positive and negative numbers, providing a robust solution for expressions involving both integers and floating-point values.

5. What is the input method for expressions?

Users can input expressions directly into the terminal. The software processes the expression and promptly displays the result in the terminal.

6. How do I handle errors, such as division by zero?

The software includes error handling, and in the case of a division by zero or other errors, appropriate error messages are displayed to guide the user.

7. Can I use parentheses to control the order of operations?

Yes, parentheses can be used to group expressions and control the standard order of operations. The software respects the grouping specified by parentheses in the input expressions.

8. Is there a limit to the length or complexity of expressions that can be evaluated?

The software is designed to handle a wide range of expressions. However, extremely long or complex expressions may reach the limits of the system's memory and processing capacity.

9. How do I interpret the result displayed in the terminal?

The result displayed in the terminal is the outcome of the evaluated arithmetic expression. It is important to review the expression entered to ensure correctness.

10. Where can I find examples of valid expressions?

Refer to Section 6 of the user manual for examples of valid expressions and their corresponding results. This section provides practical illustrations to help users understand the input format.

11. What is the difference between postfix and infix notation?

Postfix notation places operators after their operands (e.g., a b +), while infix notation positions operators between operands (e.g., a + b).

12. How does the software convert infix to postfix?

The software uses the Shunting Yard Algorithm within the `InfixToPostfix` function to convert infix expressions to postfix notation.

13. How does the software evaluate postfix expressions?

The software utilizes the `CalculatePostfix` function, employing an evaluation stack to perform calculations and obtain the result.