Foobar Inc

C++ Arithmetic Evaluator User's Manual

Version <0.2>

Revision History

Date	Version	Description	Author
02/Dec/2023	0.1	Initial User manual for the Arithmetic Evaluator	Kobe Jordan
03/Dec/2023	0.2	Add more troubleshooting errors, link to more GitHub links such as the README.md as well	Cody Duong

Table of Contents

User’s Manual

1. Purpose

The purpose of this user manual is to serve as a comprehensive guide for users to understand and effectively use the arithmetic evaluator. It aims to provide clear instructions, troubleshoot common issues, and offer examples for a seamless user experience.

2. Introduction

Welcome to the User Manual for the Arithmetic Expression Evaluator in C++. This software is designed to parse and evaluate arithmetic expressions with operators like +, -, *, /, %, and ^, and supports parentheses and normal PEMDAs operator precedence. **Note**: we do not support implicit multiplication, i.e. “1(2)” does not get parsed as “1*2”, and will instead enter the program into an error state. View ***Section 8. FAQ*** for more frequently asked questions and idiosyncrasies about the Arithmetic Expression Evaluator.

3. Getting started

To use the Arithmetic Expression Evaluator, follow these steps:

3.1 Installation

- Download the source code from the GitHub Repository.
    - Optionally: follow provided instructions in the GitHub Repository README.md.
- Compile the program using a C++ compiler (e.g., g++).
- Execute the compiled binary file.

3.2 Usage

- **Note**: The compiled binary file may have different file extensions depending on operating system. The Linux executable is expected to be named **expression_parser**, while Windows executable is expected to be named **expression_parser.exe**.
- The compiled binary file supports three flags.
- Without the `-c` or `--continuous` flag, the program will expect a provided algebraic expression as the input. Example input: `expression_parser 1+1`.
- With the `-c` or `--continuous` flag, the program will run in continuous prompt mode. Errors will not exit the program, and will prompt the user for algebraic expressions to evaluate until exiting the program by typing `exit`, `exit()`, `q`, `quit`, `quit()`, or pressing CTRL+D. This mode is mutually exclusive with a provided expression. Example input: `expression_parser -c`
- With the `-d` or `--debug` flag. Displays debug information for developers such as evaluated input and parsed abstract syntax tree (AST).
- With the `-v` flag. Will not evaluate any other flags or a provided expression and will simply display version information.

4. Troubleshooting

Encounter an issue? Refer to this section for solutions:

- **“Error: Division by zero”**
  - If you attempt to divide by zero, the program will display an error. Please ensure your expression adheres to mathematical rules.
- **“Error: Modulus by zero”**
  - If you attempt to modulus by zero, the program will display an error. Please ensure your expression adheres to mathematical rules.
- **“Error: Invalid operator”**
  - This operator or syntax is not supported. If you get this error while attempting to exponentiate using “**”, instead look to use the ‘^’ character. Supported operators are as follows: +, -, *, /, %, and ^, representing addition, subtraction, multiplication, division, modulo, and exponentiation respectively.
- **“Error: Expected an expression or number, received: {char}”**
  - This error indicates that the arithmetic evaluator expected an expression or number. Double check your operators are followed by operands on both sides.
- **“Error: Expected closing parenthesis, received: {char}”**
  - This error indicates that the program expected a closing parenthesis but instead received the following character.
    - If the {char} is equivalent to “\0 (end of expression)” this indicates that you are likely missing a closing parenthesis within your expression.
    - If the {char} is any other value this indicates that you are likely missing an operand, or your expression is invalid in some other way.
- **“Error: Missing opening parenthesis”**
  - This error indicates that the program expected an opening parenthesis to match with the number of closing parentheses.
- **“Error: Empty parentheses”**
  - This error indicates that there was an empty set of parentheses within the expression. This is not supported behavior.

Any other issues that can not be resolved can be logged as an issue at the GitHub Repository Issues Page.

5. Examples

These examples contain possible input values either input directly into the Expression Parser or while in continuous input mode. For reference on how to input either of these modes view ***Section 3.2 Usage***. Otherwise, these are possible arithmetic expressions, not examples of entire command line inputs. For that it may also be more apt to read the GitHub Repository README.md.

1. **Simple Addition:**
    a. Input: **3+4**
    b. Result: **7**
2. **Simple Subtraction:**
    a. Input: **6 – 4**
    b. Result: **2**
3. **Simple Multiplication:**
    a. Input: **5 * 3**
    b. Result: **15**
4. **Simple Division:**
    a. Input: **32 / 4**
    b. Result: **8**
5. **Exponentiation:**
    a. Input: **4^2**
    b. Result: **16**
6. **Modulus:**
    a. Input: **7 % 3**
    b. Result: **1**
7. **Modulus 2:**
    For detailed information on Modulus operation: ***Section 7 FAQ 5: Why does modulo/modulus always return a positive remainder?***
    a. Input: **(-7)%(-3)**
    b. Result: **2**
8. **Parentheses:**
    a. Input: **(1+2)/3**
    b. Result: **1**
9. **Unary Negative:**
    a. Input: **(-1) + 1**
    b. Result: **0**

6. Glossary of terms

To assist you in understanding the technical terms used in this manual, refer to the glossary:

1. **Arithmetic Evaluator:**
    - A program designed to parse and evaluate arithmetic expressions, handling operators and numeric constants according to mathematical rules.
2. **GitHub Repository:**
    - An online platform where the source code of the Arithmetic Expression Evaluator is hosted, allowing users to download and contribute to the project. The url is at: https://github.com/codyduong/EECS-348-Project.
3. **C++ Compiler:**
    - A tool that translates the C++ source code into machine code, enabling the execution of the Arithmetic Expression Evaluator.
4. **Binary File:**
    - The compiled executable file generated from the source code, which users can run to execute the Arithmetic Expression Evaluator.
5. **Order of Evaluation:**
    - The sequence in which operators and operands in an arithmetic expression are processed, typically following the rules of PEMDAS (Parentheses, Exponents, Multiplication and Division, Addition and Subtraction).
6. **Modulo/Modulus:**
    - The operator '%' that returns the remainder of the division of one number by another. The implementation of modulus in our program is aligned with the number theory definition in that result of remainder is always positive. Read more at: https://en.wikipedia.org/wiki/Modulo#Variants_of_the_definition.

7. FAQ

Have a question? Check our Frequently Asked Questions:

1. **Q: How do I handle complex expressions with nested parentheses?**
   - **A:** The program is designed to automatically recognize and evaluate expressions within parentheses based on the order of operations (PEMDAS).
2. **Q: What happens if I enter an invalid character in my expression?**
   - **A:** Invalid characters, such as symbols or letters, will prompt an error message. Ensure your expression only includes valid operators and numeric constants.
3. **Q: Can I define and use custom variables in my expressions?**
   - **A:** No. As of the current version, the program supports only numeric constants.
4. **Q: What happens if I forget to close a parenthesis in my expression?**
   - **A:** The program will detect unmatched parentheses and display an error message. Make sure to balance opening and closing parentheses in your expression.
5. **Q: Why does modulo/modulus always return a positive remainder?**
   - **A:** Modulo is implemented using the number theory definition to ensure consistent behavior. Read more here: https://en.wikipedia.org/wiki/Modulo#Variants_of_the_definition**.**
6. **Q: Is implicit multiplication 1(2) = 1*2 supported?**
   - **A:** No.
7. **Q: Are unary negatives supported?**
   - **A:** Yes, a valid arithmetic expression includes but not limited to: "---1" which is equivalent to -1.
8. **Q: Is multiplication by juxtaposition implemented?**
   - **A:** No. A classic example of this is "16 / 2*2 + 1". This will be evaluated as "((16/2)*2+1)" which is "17" .