**C++ Arithmetic Evaluator**

User’s Manual

Version <1.0>

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 02/Dec/2023 | 1.0 | Initial User manual for the Arithmetic Evaluator | Kobe Jordan |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Purpose 4

2. Introduction 4

3. Getting started 4

4. Advanced features 4

5. Troubleshooting 4

6. Example of uses 4

7. Glossary 5

8. FAQ 5

# 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.

# 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 ^, as well as numeric constants.

# Getting started

To use the Arithmetic Expression Evaluator, follow these steps:

* **Installation:**
  + Download the source code from the [GitHub Repository](https://github.com/codyduong/EECS-348-Project).
  + Compile the program using a C++ compiler (e.g., g++).
  + Execute the compiled binary file.
* **Entering Expressions:**
  + Input your arithmetic expression when prompted.
  + Use valid operators (+, -, \*, /, %, ^) and parentheses to define the order of evaluation.
* **Viewing Results:**
  + The program will display the result of the evaluated expression.

# Advanced features

Our program does not support any advanced features.

# Troubleshooting

Encounter an issue? Refer to this section for solutions:

* **Division by Zero:**
  + If you attempt to divide by zero, the program will display an error message. Please ensure your expression adheres to mathematical rules.
* **Invalid Expressions:**
  + Invalid expressions (missing operands, incorrect operators) will prompt error messages. Double-check your input for accuracy.

# Examples

To familiarize yourself with the software, let's walk through some examples:

1. **Simple Addition:**
   1. Input: **3 + 4**
   2. Result: **7**
2. **Simple Subtraction:**
   1. Input: **6 – 4**
   2. Result: **2**
3. **Simple Multiplication:**
   1. Input: **5 \* 3**
   2. Result: **15**
4. **Simple Division:**
   1. Input: **32 / 4**
   2. Result: **8**
5. **Exponentiation:**
   1. Input: **4^2**
   2. Result: **16**
6. **Modulus:**
   1. Input: **7 % 3**
   2. Result: **1**

# Glossary of terms

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

# Arithmetic Evaluator:

# A program designed to parse and evaluate arithmetic expressions, handling operators and numeric constants according to mathematical rules.

# 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.

# C++ Compiler:

# A tool that translates the C++ source code into machine code, enabling the execution of the Arithmetic Expression Evaluator.

# Binary File:

# The compiled executable file generated from the source code, which users can run to execute the Arithmetic Expression Evaluator.

# 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).

# Division by Zero:

# An error scenario where the user attempts to divide a number by zero, which is mathematically undefined and prompts an error message.

# Invalid Expressions:

# Expressions that do not adhere to mathematical rules, containing errors such as missing operands or incorrect operators, resulting in error messages.

# Modulus:

# The operator '%' that returns the remainder of the division of one number by another.

# 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: As of the current version, the program supports 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.

*.*