<EECS 348 Term Project>

<Arithmetic Expression Evaluator>

User's Manual

Version <1.1>

Revision History

Date	Version	Description	Author
<01/DEC/23>	<1.0>	<First Version of the User Manual is created. The exact features and purpose are added>	<Zach Sevart, Asa Maker, Lingfeng Li>
<02/DEC/23	<1.1>	<First version of the User Manual>	<Zach Sevart, Asa Maker, Lingfeng Li>
<03/DEC/23>	<1.2>	<Final version of the User Manual>	<Zach Sevart, Asa Maker, Lingfeng>

Table of Contents

Test Case

1. Purpose

- This project aims to develop an arithmetic expression evaluator in C++ that can accurately parse and evaluate expressions containing basic arithmetic operators and numerical constants. The purpose of the project is not only to achieve a functionally complete final product, but also to follow strict program engineering principles to ensure quality and consistency throughout the development process.

2. Introduction

2.1 Features

- Expression Parsing:
  - The program will parse arithmetic expressions and take into account PEMDAS

- Operator Support
  - The program supports standard arithmetic operators to use in evaluating user provided expressions. These include addition, subtraction, multiplication, division, exponentiation, and the modulo operator
- Parentheses Handling
  - The program can handle arithmetic expressions that use parentheses, determining the order of operations.
- Numeric Constraints
  - The program can recognize and determine numeric constants in the expressions provided by the user

2.2 Installation

- Visit the GitHub Repository:
  - Open your web browser and go to the GitHub repository where the C++ program file is located. The link will be provided in this document to access the repository containing this program's files
- Locate the File:
  - Navigate through the repository to find EECS 348 Term Project C++ program file. It will be named "main.cpp"
- Click on the File:
  - Click on the file to view its contents.
- Download the File:
  - Look for the "Download" button or the "Raw" button on the file page.

- Right-click on the "Download" or "Raw" button and select "Save Link As" or "Save Target As" (depending on your browser).
- Choose the destination on your local machine where you want to save the file and click "Save."

2.3 Running the Program

- Open Terminal or Command Prompt:
  - It is necessary to have a compiler installed on your machine
  - The program can be run using Terminal, Command Prompt, or an IDE that is capable of compiling and running C++ code.
- Run the C++ Program:
  - Execute the compiled program following any provided instructions.
  - The code will then prompt the user for an input in order to complete the necessary actions requested by the user.
- View Output:
  - The program's output will be located in the terminal of the platform they are using to compile and run the program.

3. Getting started

3.1 Entering Expressions

- When prompted, the user will enter an expression they want evaluated by the program.
- The program can evaluate numerical expressions that can include arithmetic operations and parentheses.
- The program uses order of operations (PEMDAS) to evaluate the expressions when necessary.
- Inputs that are not valid numerical expressions will not be evaluated by the program.

3.2 Using Operators

- The addition operator (+) is used to find the total of two numerical terms within the expression or to represent a positive number. To calculate the sum, it is to be put in between two numerical terms, which can include numbers or other smaller expressions within the overall expression. To give a numerical term a positive sign, add an addition operator directly left of the specified term.
- The subtraction operator(-) is used to find the difference of two numerical terms within the expression. To calculate the difference, the subtraction operator needs to be put in between two numerical terms in the user's provided expression for it to be a valid input.

To give a numerical term a negative sign, add a subtraction operator directly to the right of the specified term.

- The multiplication operator (*) calculates the product of two numerical terms. For it to be properly used, the multiplication operator needs to be placed in between two numerical terms that the user wishes to find the product of.
- The division operator (/) calculates the quotient of two numerical terms in the expression provided by the user. To calculate the quotient, the division operator needs to be put in between two numerical terms. Incorrect usage of the division operator includes attempting to divide using zero.
- The modulo operator (%) calculates the remainder of the quotient of two numerical terms. To properly use the modulo operator, place it in between two numerical terms of the expression.
- The exponentiation operator (^) calculates the product of a given numerical term multiplied by itself a specified number of times. The standard format for using exponentiation is n^m, where n is the number you want to multiply by itself and m is the number of times you want n to be multiplied.
- Incorrect usage for all operators includes pairing more than one operator directly next to each other, and not correctly implementing operators into an expression, which includes hanging operators in the expression.

3.3 Using Functions

- The main function of this program is to calculate arithmetic expressions given by the user.
- This program is equipped with parentheses handling, which prioritizes the terms of the input expression that are within parenthesis. Specific calculations that the user wants to be done before others should be bounded by a left and right parenthesis.
- The expression provided by the user must be valid in order for the program to correctly evaluate it, this includes correctly implementing operators in your function (2.2) and only providing the program with arithmetic expressions.

3.4 Interpreting Results

- The output of the program will be the calculation of the user-inputted expression

4. Troubleshooting

4.1 Common Problems and Solutions

- Unexpected Results in Calculations
  - Problem: The computed results appear inconsistent.
  - Solution: Verify adherence to the order of operations (PEMDAS). Confirm the appropriate utilization of parentheses and reevaluate operator precedence.
- Program Instabilities and Error Manifestations
  - Problem: The program encounters crashes or exhibits error messages.
  - Solution: Communicate errors effectively. Error messages will now be more informative and guide you towards the resolution of invalid expressions or exceptional scenarios, such as division by zero.
- Parenthesis Inconsistencies
  - Problem: Challenges with the management of parentheses.
  - Solution: Maintain balance in the use of parentheses. Ensure each opening parenthesis corresponds to a closing counterpart and that the encapsulated expression complies with PEMDAS.
- Operator Irregularities
  - Problem: Unanticipated outcomes stemming from operator-related issues.
  - Solution: Validate operator adherence to the PEMDAS hierarchy. Confirm that parentheses duly influence the precedence of operations for accurate calculations.
- Anomalies in Exponentiation Operations
  - Problem: Unexpected behavior in exponentiation (^) operations.
  - Solution: Scrutinize the logic governing exponentiation to ensure it conforms to the prescribed rules of PEMDAS.
- Failure to Recognize Numeric Constants
  - Problem: Numeric constants are not being appropriately identified or computed.
  - Solution: Review the logic pertaining to the identification and computation of numeric constants. Ensure flexibility for both integers and potential future inclusion of floating-point numbers.
- Verification of Program Accuracy
  - Problem: Uncertainty regarding the program's correctness.
  - Solution: Institute comprehensive testing protocols to validate the accuracy of the expression evaluator. Regularly subject the program to diverse inputs to ascertain its reliability.

5. Examples

5.1 Basic Arithmetic Expressions and outputs

- 3 + 4
  - This expression has an output of 7
- 8 - (5 - 2)
  - This expression has an output of 5
- 10 * 2 / 5
  - This expression has an output of 4
- 2^3
  - This expression has an output of 8

5.2 Complex Expressions and outputs

- 4 * (3 + 2) % 7 - 1
  - This expression has an output of 5
- (((((5 - 3))) * (((2 + 1))) + ((2 * 3))))
  - This expression has an output of 12
- -(-(-3)) + (-4) + (+5)
  - This expression has an output of -2
- -(+2) * (+3) - (-4) / (-5)
  - This expression has an output of -6.8

6. Glossary of terms

Arithmetic Expression:

- Definition: A mathematical expression containing numbers, arithmetic operators (+, -, *, /, %), and potentially parentheses and exponentiation, to be evaluated for a numerical result.

Arithmetic Expression Evaluator:

- Definition: A C++ program designed to parse and evaluate arithmetic expressions, considering operators +, -, *, /, %, and ^, as well as numeric constants. It handles expressions with parentheses to establish precedence and grouping.

Arithmetic Operator (Addition, Subtraction, etc.):

- Definition: Symbols representing basic mathematical operations, including addition (+), subtraction (-), multiplication (*), division (/), modulo (%), and exponentiation (^).

Data Structure:

- Definition: A structural format utilized within the program, such as a stack or tree, to represent the organization of the arithmetic expression during the parsing and evaluation processes.

Error Handling:

- Definition: Implementation of mechanisms to manage exceptional scenarios, including but not limited to division by zero and invalid expressions.

GitHub:

- Definition: A web-based platform for version control and collaborative program development, commonly used for hosting and managing project code repositories.

Numeric Constants:

- Definition: Fixed numerical values within an expression, recognized and calculated by the program.

Numerical Expression:

- Definition: A mathematical expression involving numbers and arithmetic operations, designed to produce a numerical result.

Operator Precedence:

- Definition: The hierarchy that determines the sequence in which operators are applied, particularly following the PEMDAS rules.

Parsing Techniques:

- Definition: Methods employed to break down an arithmetic expression into its constituent parts, facilitating subsequent evaluation.

PEMDAS:

- Definition: An acronym representing the order of operations in arithmetic expressions: Parentheses, Exponents, Multiplication and Division (from left to right), and Addition and Subtraction (from left to right).

Tokenization:

- Definition: The process of breaking down an input arithmetic expression into individual tokens, such as operators, numeric constants, and parentheses.

Unit Tests:

- Definition: Specific tests designed to verify the correctness of individual units or components within the expression evaluator.

User Interface:

- Definition: The interactive command-line interface allowing users to input expressions and view calculated results

7. FAQ

What is the purpose of this program?

- Answer: This program is designed to evaluate arithmetic expressions entered by the user. It supports basic arithmetic operations such as addition, subtraction, multiplication, division, modulo, and exponentiation, following the PEMDAS order of operations.

How do I use the program to calculate an expression?

- Answer: Simply input your arithmetic expression, adhering to standard mathematical notation. The program will parse and evaluate the expression, providing the calculated result.

What operators does the calculator support?

- Answer: The calculator supports the following operators:
    - Addition (+)
    - Subtraction (-)
    - Multiplication (*)
    - Division (/)
    - Modulo (%)
    - Exponentiation (^)

Does the calculator consider the order of operations (PEMDAS)?

- Answer: The calculator follows the PEMDAS rules, ensuring proper precedence of operations: Parentheses, Exponents, Multiplication and Division (from left to right), and Addition and Subtraction (from left to right).

Can I use parentheses in my expressions?

- Answer: Certainly! Parentheses can be used to explicitly define the order of evaluation. The calculator will prioritize operations within parentheses first.

Is there a limit to the length or complexity of expressions I can input?

- Answer: The program is designed to handle a wide range of expressions. However, excessively long or complex expressions may encounter limitations depending on system resources.

Can I use variables in my expressions?

- Answer: Currently, the calculator recognizes and evaluates numeric constants only. Support for variables is not available in the current version.

What happens if I input an invalid expression?

- Answer: The program is equipped with robust error handling. If you input an invalid expression, the calculator will provide a clear and informative error message detailing the issue.

How can I view the calculated result?

- Answer: The program provides a user-friendly command-line interface. Simply input your expression, and the calculated result will be displayed on the same interface.