Group 0

Arithmetic Expression Evaluator in C++

User’s Manual

Version <1.0>

Revision History

Date	Version	Description	Author
<09/12/2024>	<1.0>	<User's Manual Version 1.0>	<Samantha Adorno, Mason West, Evan Rogerson, Ben Haney, Nick Heyer, Rahul Nesan, Zain Cheema>

Table of Contents

Test Case

1. Purpose

The purpose of this document is to serve as an easy-to-understand guide on how to use the Arithmetic Expression Evaluator in C++ developed by Group 0. This User's Manual contains an introduction, guide to getting started with the program, an overview of more advanced features, a troubleshooting guide, examples of software usage, a glossary of terms used throughout this document, and a few frequently asked questions.

2. Introduction

The Arithmetic Expression Evaluator in C++ can be used to evaluate expressions for the user. It receives input from the user as a string, parses it, and returns the result of the expression. The program uses Reverse Polish Notation (also known as postfix notation) to simplify expression parsing and evaluation. We decided to use RPN after learning about it in EECS210 because it made evaluation easier by avoiding the need for multiple nested loops or recursion.

The program is capable of performing addition (+), subtraction (-), multiplication (*), division (/), modulo (%), and exponentials (**). It also handles parenthesis, which can be used to adjust the order of operations, and unary operators (+/-). The program also detects and throws errors for invalid inputs such as mismatched parenthesis or division by 0.

3. Getting started

Using the Arithmetic Expression Evaluator is simple! Here's a quick guide to get started:

Step 1: Compile/Run the Program

Compile the code using a C++ compiler. If using gcc, the following command may be helpful:

```
g++ -o evaluator.exe final.cpp
```

Once the code is successfully compiled, run the resulting executable.

Step 2: Enter Your Expression

Type in the arithmetic expression you'd like to evaluate. For example:

- 3 + 5
- (4 + 2) * 3
- 2 ** (3 + 1)

Expressions may include the following operators: +, -, *, /, %, **. They may also contain parentheses and unary operators (+/-).

Step 3: Review the Results

The program will instantly evaluate your expression and display the result. If something's off (like dividing by zero or using invalid characters), the program will alert you with an error message.

Step 4: Exiting the Program

When you are finished evaluating expressions, simply type 'END' to exit the program.

4. Advanced features

While mostly simple in use, the Arithmetic Expression Evaluator in C++ does offer a few more advanced features. One such feature is the use of parentheses. Parentheses can be used to group parts of an expression and alter the order of operations. For example, (2+3) * 4 ensures that 2 and 3 are added before the result (5) is multiplied by 4 to reach the final result (20).

Another advanced feature the program offers is error handling. If the user enters an invalid expression the program will print an error notifying the user. There are a variety of error messages which notify users of the problem with their expression. Without error handling, the program would crash and have to be ran again in order to continue functioning.

The final advanced feature of our code is that it can both handle decimal input and return decimal results. This is made possible by our iterative approach to the coding of the program. Input decimals by simply typing a period within the number. For example, if you input 1.2 + 1.3, the program will correctly return 2.5. This can also be helpful when dividing numbers that do not divide evenly. The result will be a decimal.

5. Troubleshooting

Below is a list of common errors that users may encounter during use of the Arithmetic Expression Evaluator and how to solve them.

Problem	Solution
The program doesn't start	Ensure that you compiled it correctly and that your C++ compiler is installed.
"Invalid Expression" error	Check for typos, like extra operators or mismatched parentheses.
Division by zero error	Double-check your input to avoid dividing by zero.
Unexpected results	Use parentheses to clarify the order of operations in your expression.

6. Examples

Below are a few examples of the Arithmetic Expression Evaluator in use:

1. 3 + 4 * 2
   Result: 11
2. -3 + 4 * 2
   Result: 5
3. 3 + 4 * (2 - 1)
   Result: 7
4. 2 ** 3 + 1
   Result: 9
5. (2 + 3) * 4
   Result: 20
6. 10 / 0
   Result: Division by zero
7. 3 + (4 * (2 -))
   Result: Mismatched parentheses
8. 3.2 + 5.3
9. Result: 8.5

7. Glossary of terms

Arithmetic Expression
A mathematical statement involving numbers, operators, and parentheses, used to compute a value.

Compiler
A software that translates C++ source code into executable machine code.

Error Handling
A feature of the program designed to detect and respond to invalid input or unexpected situations.

Executable File
A binary file produced by the compiler that can be executed to run the program.

Expression Parsing
The process of analyzing an arithmetic expression to interpret and evaluate it.

Modulo
An operator that returns the remainder of the division of two numbers.

Postfix Notation
Another term for Reverse Polish Notation.

Reverse Polish Notation (RPN)
A mathematical notation in which operators follow their operands.

Unary Operator
An operator that operates on a single operand, such as + or -.

8. FAQ

Q: Can I use the software to evaluate trigonometric or logarithmic functions?
A: Not yet! Right now, it only supports basic arithmetic operations.

Q: How do I enter exponentiation?
A: Use **. For example, 2 ** 3 computes 2^3 = 8.

Q: How does it handle errors?
A: The program checks for common mistakes, like missing parentheses or dividing by zero, and provides helpful error messages.

Q: Is there a limit to the size of expressions I can input?
A: The program can handle expressions with hundreds of thousands of characters, but practical limits depend on your system's memory.