Orange

Arithmetic Expression Evaluator

User's Manual

Version 1.0

Revision History

Date	Version	Description	Author
12/02/23	1.0	Create a manual for the user to be able to understand how to use the program.	Mikaela Navarro

Table of Contents

User's Manual

1. Purpose

The user manual for the Arithmetic Expression Evaluator is a simple and user-friendly guide on how to use the program. This guide includes the following sections.

2. Introduction

The Arithmetic Expression Evaluator is simple calculator that takes in an expression from the user and evaluates it. It features expression parsing, operator support (i.e., + (addition), - (subtraction), * (multiplication, / (division), % (modulo), ^ (exponentiation), and e (scientific notation)), parenthesis handling, and numeric constants. To install and run the program, follow these steps:

1. On our main page in GitHub, navigate to Code and Download ZIP.
2. After downloading the ZIP file, place it where it's easily accessible, and open it.
3. Open the terminal and locate the 'code' folder. The user should be able to view the obj, src, include folders, and the makefile.
4. Then enter in 'make clean' and the user should see the following:
   rm -r obj/*.o
   rm -f calculator
5. Then enter in 'make' and the user should see the following:
   g++ -c -o obj/main.o src/main.cpp -Iinclude -O3 -std=c++11
   g++ -c -o obj/solver.o src/solver.cpp -Iinclude -O3 -std=c++11
   g++ -c -o obj/token.o src/token.cpp -Iinclude -O3 -std=c++11
   g++ -c -o obj/shunyard.o src/shunyard.cpp -Iinclude -O3 -std=c++11
   g++ -c -o obj/rpn.o src/rpn.cpp -Iinclude -O3 -std=c++11
   g++ -c -o obj/tests.o src/tests.cpp -Iinclude -O3 -std=c++11
   g++ -o calculator obj/main.o obj/solver.o obj/token.o obj/shunyard.o obj/rpn.o obj/tests.o -Iinclude -O3 -std=c++11 -lm
6. Now enter './calculator' and the user is in!

3. Getting started

Now that the user is running the program, a message from the Orange team will be provided. It will introduce the user on what operators can be used when entering an expression. The following instructions will explain in further detail:

1. The user is instructed to use these following operators: +, -, *, /, ^, **, e, and %.
2. The user can also use () or [].
3. The user can now type and enter any expression and receive the result of the expression.
4. If the user enters an invalid expression, an error message will return to the user. Examples below:
   a. 2 * (4 + 3 – 1
      i. Error: Unbalanced Brackets
   b. ((5 + 2) / (3 * 0))
      i. Error: Undefined Operation: Division by Zero
5. Once, the user wants to exit the program, enter 'quit' or use Ctrl+D

4. Advanced features

1. Scientific notation support
   a. Users are allowed to express numbers in a concise and convenient form.

   b. 2e3 -> Result: 2000
2. Handling large numbers
   a. The program is able to handle large numbers.
   b. 1000000 * 1000000 -> Result: 1e+12

5. Troubleshooting

1. Invalid expression error: Unbalanced brackets/parentheses
   a. Symptom: An error related to unbalanced brackets is returned.
   b. Solution: Ensure that every opening bracket has a corresponding closing bracket. Use parentheses '()' or square brackets '[]' in pairs.
2. Undefined operation: Division by Zero
   a. Symptom: An error indicating division by zero is returned.
   b. Solution: Avoid dividing by zero. Check the expression for instances where the denominator is zero.
3. Error: Invalid Numeric Constant: scientific notation invalid exponent
   a. Symptom: An error occurs when using scientific notation with an invalid exponent.
   b. Solution: Ensure that the exponent is a valid integer. Decimal exponents are not supported.
4. Unexpected Result or Error in Calculation
   a. Symptoms: The calculator provides an unexpected result or an error for a valid expression.
   b. Solution: Review the expression for accuracy. Check the order of operations and ensure correct usage of operators.
5. Difficulty Exiting the Program
   a. Symptoms: Users encounter issues when attempting to exit the program.
   b. Solution: To exit the program, type 'quit' or use the keyboard shortcut Ctrl+D.
6. Issues with Operator Precedence
   a. Symptoms: Users notice unexpected results due to operator precedence.
   b. Solution: Use parentheses () to explicitly specify the order of operations in complex expressions.
7. Compilation Errors During Installation
   a. Symptoms: Users encounter errors during the compilation phase.
   b. Solution: Check that the necessary dependencies are installed. Ensure that the GNU Compiler Collection (g++) is available on your system.

6. Examples

Here are examples of how to use the software to evaluate different types of arithmetic expressions:

Valid Expressions:

Input: 10 * 2 / 5
Output: Result: 4

Input: ((5 * 2) - ((3 / 1) + ((4 % 3))))
Output: Result: 6

Invalid Expressions:

Input: 2 * (4 + 3 – 1
Output: Error: Unbalanced Brackets

Input: ((5 + 2) / (3 * 0))
Output: Error: Undefined Operation: Division by Zero

7. Glossary of terms

- **Expression Parsing:** The process of analyzing a mathematical expression to understand its structure and components.
- **Operator Precedence:** The rules governing the order in which operators are applied when evaluating an expression.
- **Unary Negation:** The operation of negating a single operand, often denoted by a minus sign (-).
- **Compilation:** The process of translating source code into machine code or an intermediate code by a compiler.
- **Dependencies:** External libraries or tools that your program relies on for proper functioning.
- **GNU Compiler Collection (g++):** The compiler used to compile your C++ code.
- **Makefile:** A script specifying how to compile and build a program, used by the make utility.
- **Undefined Operation:** An operation that is not defined mathematically, leading to an error.

8. FAQ

How do I run the calculator program?

- There are instructions provided in this document on section 2.

Can the calculator handle basic arithmetic operations?

- Yes, our program handles operations like addition, subtraction, multiplication, and division.

Can this program run on multiple platforms?

- Yes, the program can run on multiple platforms such as Linux and macOS. The program uses the GNU Compiler Collection ('g++') and follows standard C++ practices.

Can you the program solve negative values when using the exponent operator(s) (^, **) or the scientific notation (e)?

- Users can use unary positive and negative operators. With the exponentiation (^, **) and scientific notation (e), unary negation is allowed. Below are examples of valid expressions with negative values:
  - 2 ** -3
  - -2 ^ 3
  - -2 ^ -3

Are decimals allowed to use?

- Decimals are allowed, however when using scientific notation (aeb), the exponent 'b' cannot be a decimal. The base in scientific notation can be a decimal. Below is an example of an expression that wrongly uses decimals in our program with the result:
  - 2e3.2 -> Error: Invalid Numeric Constant: scientific notation invalid exponent

Can you use brackets []?

- Yes brackets are allowed, and work the same as parentheses (). Below is a valid example:
  - 3 * (4 + [5 – 2]) / (7 * [2 + 1])