Fancy Calculator Inc.

Arithmetic Expression Evaluator in C++

User’s Manual

Version <1.0>

Revision History

Date	Version	Description	Author
12/10/2024	1.0	The user manual has been created and fully filled out	Luke Reicherter

Table of Contents

Test Case

1. Purpose

The Arithmetic Expression Evaluator in C++ is an easy-to-use program designed to parse and compute basic arithmetic expressions. This program allows for the use of integer and decimal values, as well as the following unary and regular operators: +, -, *, /, %, **, (,). The expression will be evaluated according to PEMDAS.

2. Introduction

To install the program, open a terminal window and type the following commands:

```
git clone https://github.com/LukeReicherter/EECS-348-Group-Project.git

cd EECS-348-Group-Project

cd AEE_Code

make
```

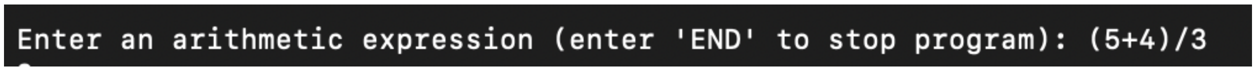
If this method does not work, use https://github.com/LukeReicherter/EECS-348-Group-Project to manually download and compile the code.

To run the code, type the following command into the terminal after following the installation instructions

```
./AEEprogram
```

3. Getting started

To start, simply enter any expression into the input like. An example is given below:



```
Enter an arithmetic expression (enter 'END' to stop program): (5+4)/3
```

The result of the expression will be displayed in the line immediately below the input line. The program will then allow the user to input a different expression.

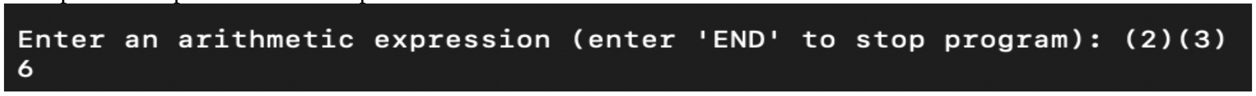
The valid operators are as follows: +, -, *, /, %, **, (,)

To end the program, the user can type "END" or "end" at any point

IMPORTANT NOTE: The "-" operator is always considered unary. For example, "1-1**6" will tokenize to ["1", "+", "-1", "**", "6"] and evaluate to "2" instead of "0".

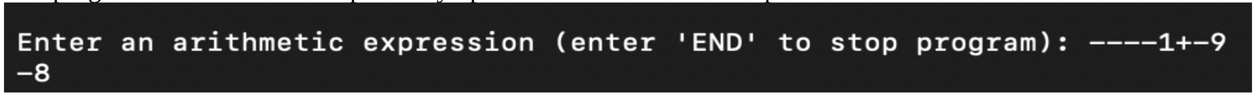
4. Advanced features

The Arithmetic Expression Evaluator has a few extra features that allow for more flexibility in the input expression. The program can handle multiplication of numeric constants inside of parentheses without a multiplication operator. An example is as follows:



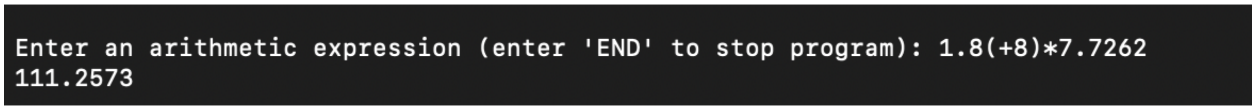
```
Enter an arithmetic expression (enter 'END' to stop program): (2)(3)
6
```

The program can also take multiple unary operators in a row. For example:



```
Enter an arithmetic expression (enter 'END' to stop program): ----1+-9
-8
```

The program also allows for floating point numbers to be input into an expression. For example:



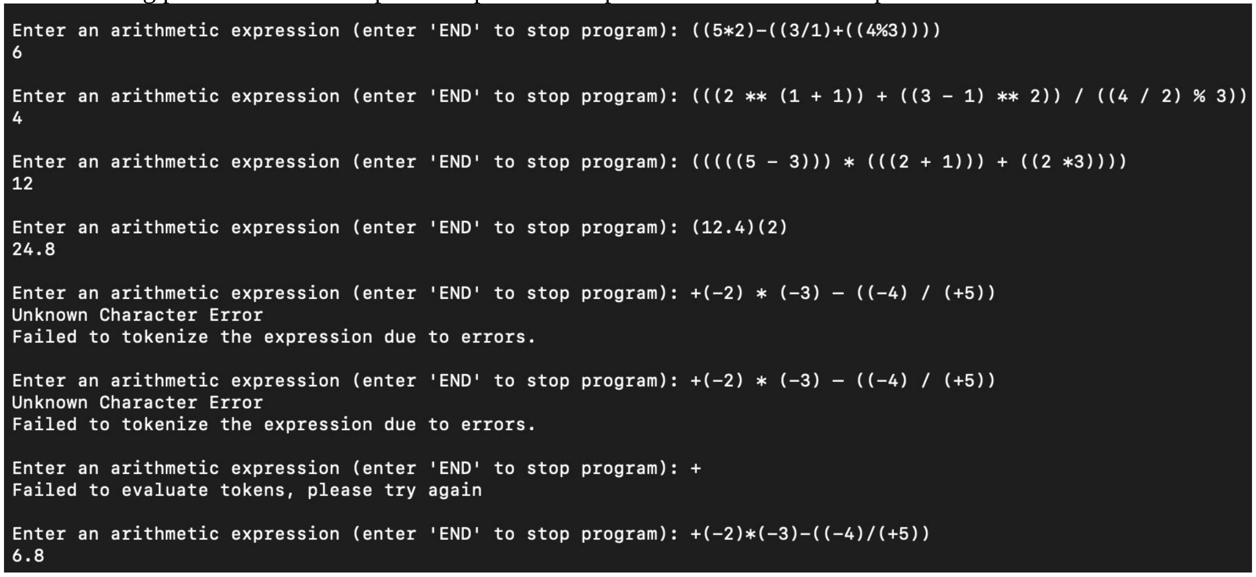
```
Enter an arithmetic expression (enter 'END' to stop program): 1.8(+8)*7.7262
111.2573
```

5. Troubleshooting

If the user would like to calculate a result of an expression with a higher precision than currently allowed, the precision value can be altered in the AEEmain.cpp file. In line 26, the variable named “PRECISION” can be set equal to a higher number to allow for a more accurate result. If precision is set too high, it might result in integer overflow, which will produce an inaccurate result past a certain decimal value. The recommended precision is 7, which is the currently set value.

6. Examples

The following pictures show examples of inputs and outputs of the Arithmetic Expression Evaluator:



```
Enter an arithmetic expression (enter 'END' to stop program): ((5*2)-((3/1)+((4%3))))
6

Enter an arithmetic expression (enter 'END' to stop program): (((2 ** (1 + 1)) + ((3 - 1) ** 2)) / ((4 / 2) % 3))
4

Enter an arithmetic expression (enter 'END' to stop program): ((((5 - 3))) * (((2 + 1))) + ((2 *3))))
12

Enter an arithmetic expression (enter 'END' to stop program): (12.4)(2)
24.8

Enter an arithmetic expression (enter 'END' to stop program): +(-2) * (-3) - ((-4) / (+5))
Unknown Character Error
Failed to tokenize the expression due to errors.

Enter an arithmetic expression (enter 'END' to stop program): +(-2) * (-3) - ((-4) / (+5))
Unknown Character Error
Failed to tokenize the expression due to errors.

Enter an arithmetic expression (enter 'END' to stop program): +
Failed to evaluate tokens, please try again

Enter an arithmetic expression (enter 'END' to stop program): +(-2)*(-3)-((-4)/(+5))
6.8
```

7. Glossary of terms

Git clone – makes a copy of the given directory to the computer

Cd – Change directory

Make – Creates the program executable using the makefile

Precision – Used by the Arithmetic Expression Evaluator to determine the amount of decimals the will be printed

8. FAQ

1. How does the Arithmetic Expression Evaluator work?
    - The program handles the user input and output in the main file
    - The input expression is sent to the tokenizer as a string
    - The string is indexed through each character to determine the value of each character
    - The tokenizer returns a vector of strings, with each string holding a token
    - The vector of strings is sent to the parser and evaluator
    - The parser uses recursion to create a priority stack with multiple levels
    - Each level represents the priority of operators in PEMDAS order

- The evaluation and parsing process occurs in the same file
- Once the parsing and evaluation is complete, a double is returned to the main file
- The output is then printed

2. What is the purpose of this project?
    - The purpose of this project was to learn the steps involved in the development process of a product
    - We focused on the requirements, design, coding, and testing phases of development
    - For each phase, a corresponding document has been created
    - These documents can be accessed through the provided git hub links above

3. How can I implement additional operators into the program?
    - To add additional operators, start by going to the Tokenizer.cpp file
    - Operator detection occurs below line 189
    - This is where an additional operator check can occur
    - Now head to the ParserandEvaluator.cpp file
    - Decide what priority level the operator should have (or create your own level)
    - Add the token check to the given priority level, as well as the correct equation for the operator.