<Company Name>

Arithmetic Expression Calculator

User's Manual

Version 1.0

Revision History

Date	Version	Description	Author
03/12/2023	1.0	Document Creation	Sam Muehlebach, Josh Welicky, Jennifer Aber, Mark Kitchin, Basim Arshad, Jawad Ahsan

Table of Contents

User Manual

1. Purpose

This guide is intended to provide instructional support for end users of the Arithmetic Expression Calculator, including installation instructions, features, use instructions, troubleshooting methods, and frequently asked questions.

2. Introduction

The Arithmetic Expression Calculator provides a way to evaluate arithmetic expressions via a command line interface. The Arithmetic Expression Calculator provides several features. It allows the user to enter a simple or compound arithmetic expression, with support for the addition, subtraction, multiplication, division, exponential, and modulus operators. Additionally, the software allows the user to enter decimal and whole numbers as operands in entered expressions. The software allows for the creation of compound expressions with support for parenthetical grouping of internal expressions. Furthermore, the software supports the use of the + and – characters as unary operators for operands. The software is also forgiving of invalid user input, ensuring that input containing errors will not result in a complete system failure.

2.1 Installation

The Arithmetic Expression Calculator needs a C++ compiler to run. The simplest way to verify the presence of a C++ compiler is to open the command terminal of the computer in use and enter the command `g++ -v`. If the machine has a C++ compiler installed, the terminal should display the version information. Otherwise, it will display a message such as: "g++ is not recognized as an internal or external command". If the machine in use does not contain a C++ compiler, one should be downloaded and installed.

Once the user has verified support for C++, the Arithmetic Expression Calculator is installed using the following instructions:

1. In an Internet Browser, navigate to the GitHub repository containing the software: https://github.com/sammuehlebach/EECS-348-Team-Project

2. All necessary files are contained in a ZIP file. Click the Arithmetic_Expression_Calculator.zip link and press ⤓ to download the file to the computer.

3. Unzip the downloaded file to any directory. This can be accomplished in a command terminal with the command:

```
unzip Arithmetic_Expression_Calculator.zip
```

4. From a command line prompt, navigate to the folder containing the extracted files, named Arithmetic_Expression_Calculator, and type in the following command:

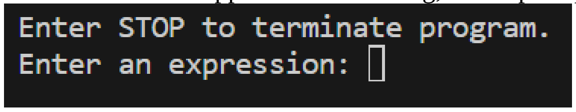
```
g++ -o calculator Driver.cpp
```

5. The above command creates an executable called 'calculator' in the current directory. To run the application, type the following command:

```
./calculator
```

3. Getting started

1. Once the application is running, it will prompt the user for an expression. The command will resemble this:

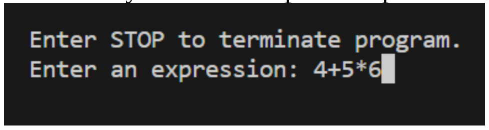


```
Enter STOP to terminate program.
Enter an expression: 
```

Figure 1: Initial software prompt.

2. In the second line of the message, the enter the arithmetic expression to be evaluated. The expression will need to be entered via the keyboard of the computer in use.
    a. The digits of multi-digit operands must be entered together, unseparated by spaces. The same principle applies to decimal numbers. Decimal points are represented with the . character. Operands must be separated by binary operators.
    b. When utilizing a binary operator, it must be placed between two operands. The addition operation can be denoted with the + character. The subtraction operator is denoted by the - character. The multiplication operation is denoted by the * character. The division operator is denoted by the / character. The exponentiation operation is denoted by the ^ character or the sequence of characters **. The modulus operation is denoted by the % character, and cannot be used on decimal numbers. For operations where the commutative property does not apply, such as division or subtraction, the position of the operands will determine the result of the operation. For instance, the when "3-2" is entered, 2 will be subtracted from 3.
    c. When utilizing a unary operator, the + character denotes a positive number, while the - character denotes a negative number. A unary operator must be placed directly to the left of an operand, such as the example -3.
    d. When utilizing parenthetical grouping, denote the beginning of the parenthetical expression with the (character. Then, enter the desired expression. Finally, enter the) character to denote the end of the parenthetical expression. Valid input must contain an equal number of opening and closing parentheses.

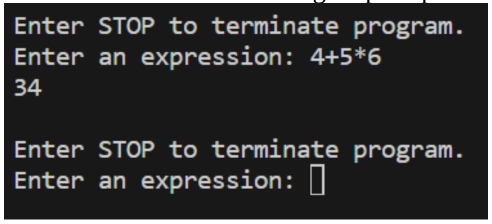
When the expression has been completed, submit the expression by pressing the ENTER key on the keyboard. A completed expression will resemble the following figure:



```
Enter STOP to terminate program.
Enter an expression: 4+5*6
```

Figure 2: Entered arithmetic expression.

3. Once a valid arithmetic expression has been submitted, a number will be printed to the terminal. This number should be interpreted as the final result of the arithmetic expression. If a message is printed to the terminal instead, an error of some kind, outlined by the message itself, was detected in the input. The terminal will once again prompt for another arithmetic expression.

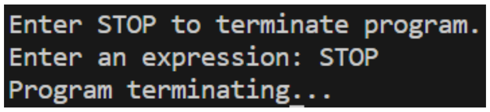


```
Enter STOP to terminate program.
Enter an expression: 4+5*6
34

Enter STOP to terminate program.
Enter an expression: 
```

Figure 3: The result is printed.

4. To end the application, type in the message “STOP”. This message must be typed in full capitalization. Otherwise, it will be interpreted as an invalid expression. The application will print the message “Program terminating...” to acknowledge the stop request.



```
Enter STOP to terminate program.
Enter an expression: STOP
Program terminating...
```

Figure 4: Program termination.

4. Advanced features

4.1 Alternate Exponentiation Syntax

The Arithmetic Expression Calculator allows users to utilize an alternate syntax for exponentiation. Instead of using the standard ^ character, users are permitted to indicate exponentiation via two asterisks, **. The base number should be placed to the left of these asterisks, and the desired exponent should be placed to the right of the asterisks. For example, the alternate syntax of the expression 2^4 is 2**4.

4.2 Unary Operator Containment

While encasing numbers with unary operators within parentheses is encouraged, it is not required. For instance, the expression 4 - (-2) can be rewritten as 4 - -2. However, this is not permitted when assigning positive or negative signs to expressions within parentheses unless the expression begins with such an action. Instead, such a term should be enclosed within parentheses as well. For instance, the expression 3 - -(3) will result in an error, but 3 - (-(3)) is permitted.

4.3 Minimized Decimal Values

When entering decimal operands less than 1 but greater than –1, such as 0.4 or –0.3, the zero in the ones digit is unnecessary. For example, 0.4 can also be entered as .4, and –0.3 can also be entered as -.3. However, when utilizing unary operators, the operator must be placed immediately next to the decimal point.

4.4 Non-integer Exponents

Non-integer exponents are permitted. They may be represented fractionally, such as 1/2, or as a decimal, such as 0.5. However, since a fraction is, by definition, an expression in and of itself, a fractional exponent must be contained within parentheses.

5. Troubleshooting

5.1 Compilation Problems

If the calculator application does not compile during the installation process outlined in Section 2.1, delete all files associated with the Arithmetic Expression Calculator, and repeat the installation process. If this still does not work, return to the GitHub repository for the software, and enter the Implementation file, which will contain four files titled Arithmetic.h, Driver.cpp, aVector.h, and theStack.h. Click on each file, and click the ⤓ button to download each file. Ensure that all files are located in the same directory on the computer. Then, follow the installation instructions 4 and 5.

5.2 Application Running Failure

If, for some reason, the application compiles but fails to run, there may have been a error in the compilation process. To resolve this, open a command terminal, navigate to the directory containing the calculator application and associated files, and enter the following commands:

```
rm calculator
```

```
            g++ -o calculator Driver.cpp
```

This will delete and recreate the calculator application.

5.3 "can only be preceded by" Messages

Messages containing the phrases "can only be preceded by" or "must be followed by" are printed when the application detects a misplaced operator, operand, or parentheses. For reference, the following principles must be applied when entering input.

- A number or operand must be preceded by a binary operator or opening parentheses.
- A binary operator must be preceded by a number or closing parentheses.
- An opening parenthesis must be preceded by an operator or another opening parenthesis.
- A closing parenthesis must be preceded by a number or closing parenthesis.

Verify that the desired input has met these criteria. If the input satisfies these criteria but still results in error messages, encase all operands within parentheses. Also ensure that any unary operators are placed immediately next to their associated numbers. Unary operators that are not will be interpreted as binary operators.

5.4 "Invalid Character Detected"

This message will be printed when an invalid character has been found in the entered input. Ensure that the desired arithmetic expression only contains numerical characters as well as the following characters: (,), +, -, *, /, ^, %, ., or the phrase "STOP" in exactly that form. Any other characters are not permitted in input.

5.5 Incorrect Output

If the output returned is not the predicted output, encase all subexpressions inside parentheses. For instance, the expression 3+4*8 should be rewritten as 3 + (4*8). This will manually ensure that the order of operations is followed.

5.6 Zero Division Error

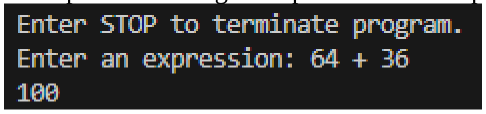
If the program returns the message "Division by zero is not allowed" or "Modulo by 0 is not allowed", the program has encountered an attempt to divide by or that the modulo of 0. Check the entered expression to check that 0 is not entered as the divisor of a division or modulo operation. Furthermore, check any subexpressions that are placed as a divisor in a division or modulo operation that might evaluate to zero. If a subexpression divisor evaluates to zero, this will be detected by the software.

5.7 "Modulo does not accept floats."

If the program returns this message, it has detected an attempt to use the modulo operator on a decimal operand. Ensure that the entered input does not contain an instance where a decimal number is associated with a modulo operator. Furthermore, if a modulo operation is present with parenthetical expressions as operands, ensure that those parenthetical expressions do not evaluate to decimal numbers.

6. Examples

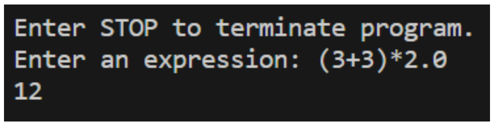
Example: Evaluating a simple arithmetic expression:



```
Enter STOP to terminate program.
Enter an expression: 64 + 36
100
```

Figure 5

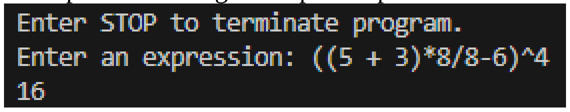
Example: Evaluating a parenthetical expression:



```
Enter STOP to terminate program.
Enter an expression: (3+3)*2.0
12
```

Figure 6

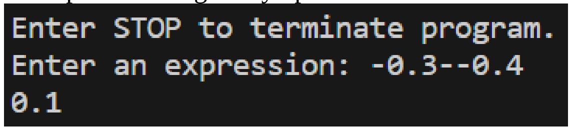
Example: Evaluating a complex expression:



```
Enter STOP to terminate program.
Enter an expression: ((5 + 3)*8/8-6)^4
16
```

Figure 7

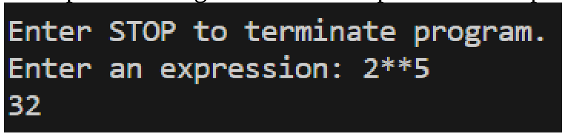
Example: Utilizing unary operators:



```
Enter STOP to terminate program.
Enter an expression: -0.3--0.4
0.1
```

Figure 8

Example: Utilizing the alternate exponentiation operator:



```
Enter STOP to terminate program.
Enter an expression: 2**5
32
```

Figure 9

7. Glossary of terms

Binary operator	— An arithmetic operator associated with two operands, such as addition, subtraction, etc.
Commutative Property	— A mathematical property of some operators that states that the position of the operands in relation to an operator does not affect the result of the operation. For instance, the multiplication operation is commutative, since a*b = b*a.
Integer	— A whole number, a number without a decimal point.
Operand	— Often used to denote a number in an arithmetic expression.
Operator	— Often used to denote the +, -, ^, /, *, and % characters in an arithmetic expression.
PEMDAS	— An acronym that represents the order of operator precedence. They take precedence in the order of parentheses, exponents, multiplication and division, and addition and subtraction.

Unary operator	—	A positive or negative sign, denoted by + or – respectively.

8. FAQ

Q: Can I use negative numbers with exponents?

A: Yes, this application allows negative exponents, such as: 2^-3.

Q; Does the program accept unary inputs for operands?

A: Yes, if an operand is preceded by a + or -, the program will recognize it as a positive or negative number and will evaluate the expression appropriately.

Q: Does this program work on all terminals?

A: Yes, as long as it is running on some sort of command line interface and the computer has a C++ compiler, the program should run successfully.

Q: Does the program support operator precedence (PEMDAS)?

A: Yes, the program solves operations with respect to operator precedence.

Q: Does the program support square root operations?

A: Yes, this can be done by using a fractional exponent such as 0.5 or 1/2.

Q: Is spacing between operands and operators required?

A: Spacing is only relevant in the following situations. Multi-digit/decimal operands cannot have spaces between digits/decimal points. Unary operators cannot be separated by a space from its associated operand. The ** representation of the exponent operator cannot be separated by spaces. Otherwise, spacing is irrelevant. 5 + 5 is just as valid as 5+5.

Q: Are numbers that end in .0 considered decimal numbers in modulo operations?

A: No. Numbers such as 1.0 are considered whole numbers and can be used in modulo operations. Only true decimal numbers, like 1.5, are disallowed from the modulo operator.

Q: Can decimal numbers be used with the modulus operation?

A: No. The modulus operation can only be used on whole numbers and expressions that evaluate to whole numbers.

Q: Can unary operators be used with parenthetical expressions?

A: Yes. If a parenthetical expression is placed at the beginning of an expression, it can be easily assigned a unary operator. However, to assign a unary operator to a parenthetical expression in the middle of an expression, encase it within parentheses. For example, 5 - +(3+2) is not a valid input, but 5 - (+(3+2)) is. Also, note that this expression can be expressed as 5-(3+2), eliminating the necessity of a unary operator.

Q: Can a number be multiplied with another operand simply with parentheses?

A: No. Expressions that attempt this, like 2(5), will raise an error, since two operands are placed together with no operator to separate them. Expressions like this should contain the * character to denote multiplication. For example, the correct way of entering the example is 2*(5).