**C++ Arithmetic Expression Evaluator**

User’s Manual

Version 1.0

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Purpose 4

2. Introduction 4

3. Getting started 4

4. Advanced features 4

5. Troubleshooting 4

6. Example of uses 4

7. Glossary 5

8. FAQ 5

**User’s Manual**

# Purpose

The purpose of this software is to allow users to enter mathematical expressions containing numbers, operators, and parentheses. It will evaluate the expressions and display the result.

# Introduction

The Arithmetic Expression Evaluator is a program developed in C++ that is meant to solve arithmetic expressions by means of the standard order of operations. Some features include: tokenization, parsing, evaluation, error handling, and input validation.

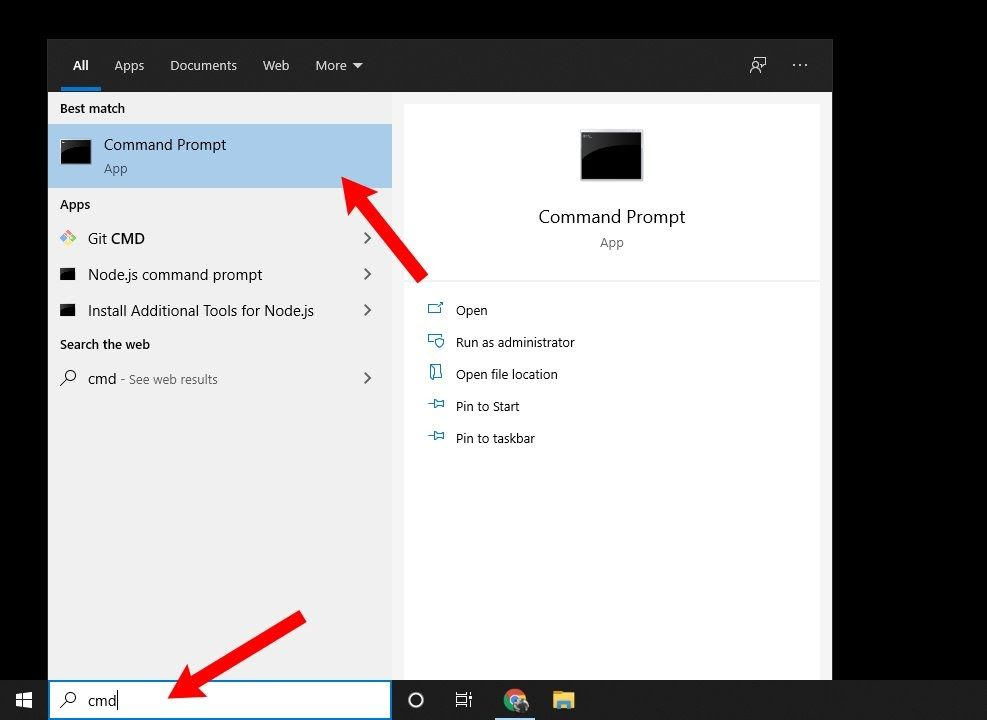
**2.1 Installation:**

Prerequisites:

* C++ Compiler
  + We recommend <https://gcc.gnu.org> for linux and <https://www.mingw-w64.org> for windows
* Git for your operating system
  + Visit <https://git-scm.com/download> and download git commands for your CLI

How to install (Beginners):

1. Open up the Command Prompt on your computer (note that this works with any CLI):



1. In the Command Prompt paste/type the following command:

git clone https://github.com/tahakhalid12/348-Group\_Project.git

1. Now you should have a clone of the 348-Group\_Project folder on your PC. The next step is to go to that folder by entering:

cd 348-Group\_Project

1. Next, to get to the program, enter:

cd Source Code

1. Once in that folder, go ahead and compile the program using your c++ compiler:

g++ -std=c++11 main.cpp -o MathMate

NOTE: Must use version 11 of C++ in order to use all the features in MathMate

1. Once the program is compiled and saved to the executable “MathMate” you have successfully installed the program! To use it, just run the MathMate executable by double-clicking it in the Source Code folder. Feel free to move this executable around to a more convenient location on your device.

How to install (Simplified):

1. Open any CLI and navigate to the directory in which you want to save the file

1. Paste in the command: git clone https://github.com/tahakhalid12/348-Group\_Project.git

1. Next navigate to the source code directory and compile the main function using:

g++ -std=c++11 main.cpp

Note: Must use C++ version 11

1. Once compiled to an executable, save the executable to wherever you would like to store the program

# Getting started

Now that you have successfully executed the program, you are greeted with our menu:

pic of menu

Here you can choose:  
--------------------------------------------------------------------------------

C++ Arithmetic Expression Evaluator

--------------------------------------------------------------------------------

1 - Enter Expression

2 - History

3 - User Manual

4 - Quit

--------------------------------------------------------------------------------

Select an option:

You can now enter an arithmetic expression which can be done when you enter number one. You’ll be then prompted to enter an arithmetic expression that you want to evaluate:

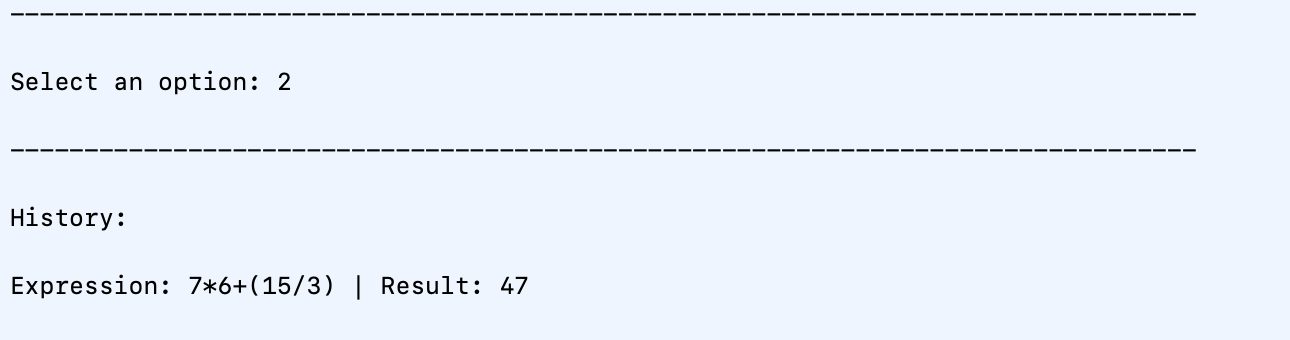


For example:



You can use any combination of operators as well as numeric constants and parentheses to define precedence and grouping.

2. History:

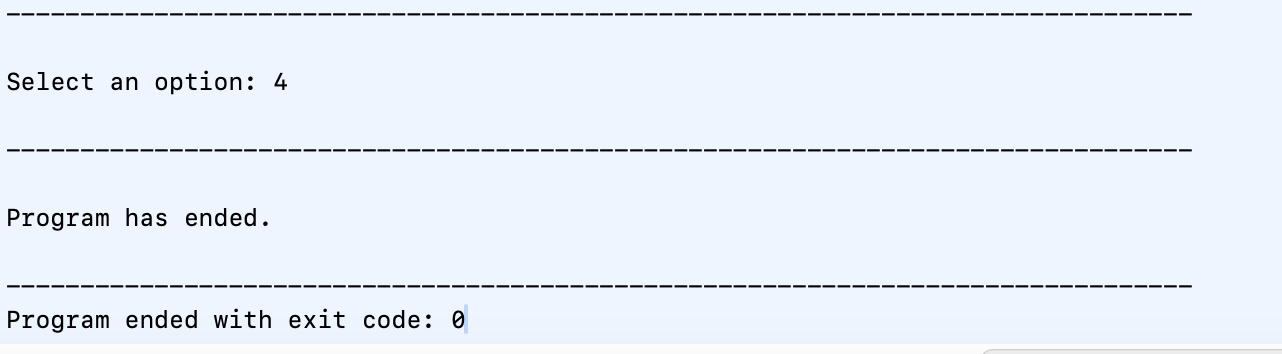
Option 2 allows you to view a history of previously evaluated expressions. If you've entered and evaluated multiple expressions during the session, you can review them here.

3. User Manual:

For detailed instructions and information on supported operators and functions, select option 3. The user manual provides guidance on using advanced features, mathematical functions, and any specific syntax rules.

4. Quit:

To exit the C++ Arithmetic Expression Evaluator, select option 4. This will close the program and end the session.



# Advanced features

The program allows for the history to be tracked using the CalculatorHistory class. This accounts for all evaluated expressions and results that a user may input. Users are able to view the history through the menu options. With this, you can view a list of the previous expressions and the corresponding results. After every input from the user, the results and expression are added to a parameter labeled ‘history’. These results that are stored can be displayed as well whenever the user decides to from the menu. With this mechanism for tracking history, users can easily view the history of expressions and their corresponding results through the program’s user interface. This keeps a record of previous calculations performed by the users and allows them to refer back to their work or review it.

The program also includes a manual for the user that is accessible through the menu. The user manual provides information about which operators are supported, how to enter expressions, and how to properly use menu options. The purpose of the user manual is to display a comprehensive user manual that accurately and properly guides users on how to use the arithmetic expression evaluator. The user manual is divided into several sections that each provide information on specific topics for the user. First, the user is welcome to the program followed by three different sections that may be of help to the user. The manual describes the available menu options and their functionalities, guidance on how to properly input arithmetic expressions using different numbers and operators, and finally a message that informs users that they can access their history of calculations.

# Troubleshooting

Some common problems that arise with users, and how to solve them.

1. Error: Invalid Expression

* Symptoms: The result displays "Error: Invalid expression."
* Possible Causes:
* The entered expression contains syntax errors.
* Solution:
* Double-check the expression for correct syntax, matching parentheses, and valid operators.

2. Error: Insufficient Operands for Unary Operator

* Symptoms: The result displays "Error: Insufficient operands for unary operator."
* Possible Causes:
* Using a unary operator without a preceding operand.
* Solution:
* Ensure there is a valid operand before using a unary operator.

3. Error: Insufficient Operands for Operator '<operator>'

* Symptoms: The result displays "Error: Insufficient operands for operator '<operator>'."
* Possible Causes:
* Using a binary operator without enough operands.
* Solution:
* Make sure there are enough operands for the specified binary operator.

4. Error: Attempted Division/Modulo by Zero

* Symptoms: The result displays "Error: Attempted division/modulo by zero."
* Possible Causes:
* Attempting to divide or use modulo with a zero divisor.
* Solution:
* Avoid division or modulo by zero.

5. Error: Unmatched Parentheses

* Symptoms: The result displays "Error: Unmatched parentheses."
* Possible Causes:
* Missing or mismatched parentheses in the expression.

Solution:

* Ensure all parentheses are correctly matched and closed.

6. Error: Invalid Expression Format

* Symptoms: The result displays "Error: Invalid expression format."
* Possible Causes:
* Unexpected formatting issues in the expression.
* Solution:
* Check the expression for unexpected characters or formatting errors.

7. Error: Unknown Operator

* Symptoms: The result displays "Error: Unknown operator."
* Possible Causes:
* Using an unsupported or unrecognized operator.
* Solution:
* Only use valid operators supported by the program.

8. Error: Division by Zero

* Symptoms: The result displays "Error: Division by zero."
* Possible Causes:
* Attempting to divide by zero.
* Solution:
* Avoid dividing by zero in your expressions.

# Examples

**Combining unary operators with arithmetic operations:**

Enter an arithmetic expression: +(-2)\*(-3)-((-4)/(+5))

Result: 6.8

Explanation: This expression combines unary operators “+” and “-” with multiplication, division, and addition. This does not affect the order of operations or final output.

**Mixed operators with extraneous parentheses:**

Enter an arithmetic expression: ((4\*3)-((4/1)+((4%3))))

Result: 7

Explanation: This expression combines various types of operators with multiple sets of extraneous parentheses.

**Nested Parentheses with Exponents:**

Enter an arithmetic expression: (((2^(1+1))+((3-1)^2))/((4/2)%3))

Result: 4

Explanation: This expression includes nested parentheses and multiple instances of exponentiation and also includes the utilization of the modulus operator.

# Glossary of terms

Some words and terms that are used in this manual aren’t understood by everyone. Here we give some definitions to words that some people may not be familiar with:

**Unary**: Unary operators work on a single operand, and don’t need multiple values to produce a new output.

Examples include: Making a number negative (-), or positive (+).

**Tokenization**: Breaking down the parts of an expression into “tokens” that can be further evaluated and compared to each other.

Example: 10+10/5+5 would result in tokens of “10+10” and “5+5” to later be compared by “/”

**Expression**: Any mathematical statement which consists of numbers and operations.

Example: 1 + 3, 42, etc.

**Extraneous**: Not relevant to the problem at hand.

Example: Extraneous parentheses are included in the expression ((4+3)).

**Parsing**: To analyze into logical syntactic components.

Example: 78\*8 is parsed into 78, \*, and 8 read right to left.

# FAQ

**Q: What operators does the calculator support?**

A: The operator supports basic arithmetic operators such as addition (‘+’), subtraction (‘-’), multiplication (‘\*’), division (‘/’), modulo (‘%’), and exponentiation (‘^’).

**Q: Can I use parentheses to group expressions?**

A: Yes, the calculator allows you to use parentheses to group expressions since it follows the standard rules of precedence. For example, an input of ‘2 \* (3 + 4)’ will give a result of 14.

**Q: Can the calculator handle decimals?**

A: Yes, the calculator can handle decimals. For example, an expression such as ‘2.5 + 1.5’ will give a result of 4. Along with this, the calculator can also use decimals paired with exponential numbers. For example, an expression of ‘2.5^2’ will result in 6.25

**Q: How do I exit the calculator?**

A: Users can easily terminate the calculator by going to the main menu and selecting option #4: ‘Quit’. Then press enter to confirm the choice. This will exit the program and also provide a message that indicates the program has ended.

**Q: Can I use variables, constants like π or e, or other advanced functions like on some phone calculators?**

A: The current version of the Arithmetic Expression Evaluator unfortunately does not support user-defined variables, constants like π or e, or other advanced functions beyond the basic arithmetic operators and exponentiation. The calculator focuses on evaluating arithmetic expressions.

**Q: Can I save questions?**

A: The Arithmetic Expression Evaluator saves expressions and the corresponding results in the program. You can access this by selecting ‘History’ in the menu. This displays the history of the evaluated expressions and their results.

**Q: Is there a maximum limit length of expressions?**

A: Although the Arithmetic Expression Evaluator is designed to handle a wide variety of expressions, extremely long or rather complex expressions may reach the limits of the program's input handling capacity.