Kansas Instruments™ (not really)

Arithmetic Expression Evaluator in C++
User Manual

Version 1.2

Revision History

Date	Version	Description	Author
10/31/2023	1.0	User Manual Created, Section 1 completed	Benjamin Kozlowski, MJ McGee, Jacob Leehy, Nicholas Hausler, Steve Gan
11/24/2023	1.1	Review and complete the User Manual for versions 4-6 of the software	Benjamin Kozlowski, MJ McGee, Jacob Leehy, Nicholas Hausler, Steve Gan
12/01/2023	1.2	Finished Completing the User Manual, added final touches	Benjamin Kozlowski, MJ McGee, Jacob Leehy, Nicholas Hausler, Steve Gan

Table of Contents

User Manual

1. Introduction

1.1 Purpose

The purpose of the User Manual is to give a potential user insight into how to correctly utilize the Arithmetic Expression Evaluator in C++.

1.2 Scope

This User Manual goes over the entirety of the predicted usages of the Arithmetic Expression Evaluator in C++. It will include how to get started with the program, how to correctly enter information into it, any advanced features included in the program, and how to do any troubleshooting that may be required.

1.3 Definitions, Acronyms, and Abbreviations

EECS – Electrical Engineering and Computer Science

KU – University of Kansas

SA – Software Architecture

SAD – Software Architecture Document (This document)

SRS – Software Requirements Specifications

UML – Unified Modeling Language

1.4 References

GitHub Repository: The project will be documented through a GitHub repository containing all relevant information regarding the project, including the Software Development Plan and this file.

Software Development Plan (Document 1): The plan for the project including the organization, jobs, focuses, scope, deliverables, and general overview of the project.

Software Requirements Specifications (Document 2): The document denoting the requirements, both functional and non-functional, for the project.

Software Architecture Document (Document 3): This document goes over how the program will be created and organized, and how different parts of it will interact with one another.

Test Cases Document (Document 5): This document houses the multitude of test cases that were created to attempt to find and fix any bugs or problems with the program.

Meeting logs: Notes taken by the Recording/Secretarial Engineer describing the activities within each meeting and other miscellaneous information.

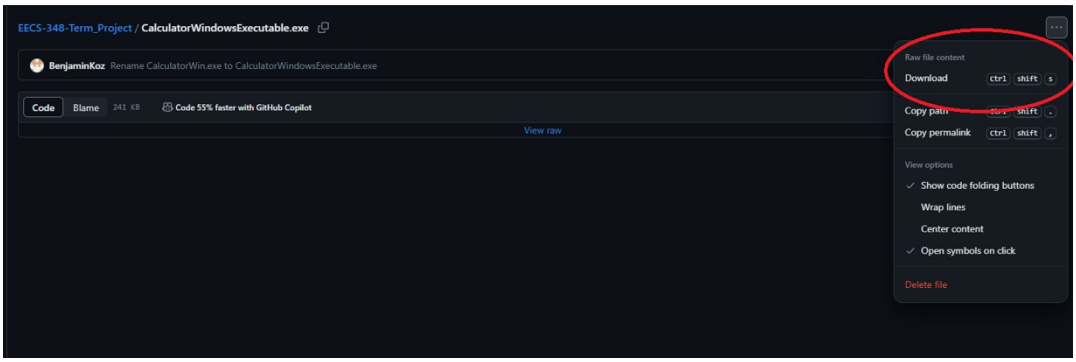
Glossary:

- EECS: Electrical Engineering and Computer Science
- KU: The University of Kansas
- UML: Unified Modeling Language

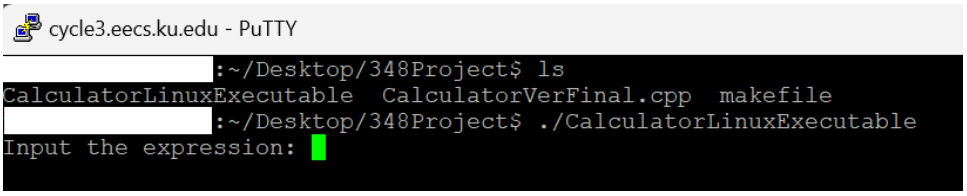
2. Getting Started

For Linux (Recommended):
To use the Arithmetic Expression Evaluator, the user will need to have a C++ compiler installed on their machine. Most Linux distros should come with a compiler so you can skip this step.

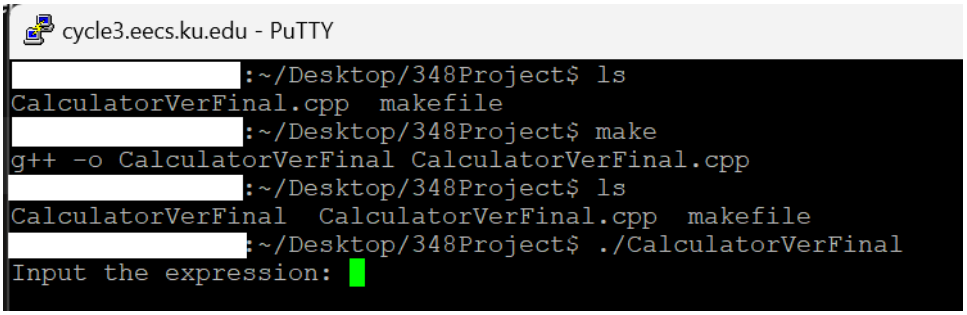
To get started with the Arithmetic Expression Evaluator, the user will need to download the correct files, including the executable for your OS or both the C++ code and makefile.



If the user opts to download the executable directly, using a Linux terminal of the user's choice (bash recommended), the user must navigate to the directory where the file is stored (likely Downloads by default) and enter ‘./CalculatorLinuxExecutable’ into the terminal. This will run the program. If the name of the file has changed from the default, replace ‘CalculatorLinuxExecutable’ with the new file name.



If the user opts to download both the C++ code and the makefile, the user will need to compile the code themselves. Users will need to open a Terminal and navigate to the folder containing the source code and makefile for the calculator (both files must be contained within the same directory). From the terminal, the user should enter the ‘make’ command, and an executable file should be created within the folder (NOTE: the makefile is only compatible with the g++ compiler). By default, this executable will be called ‘CalculatorVerFinal’. Now, the user should run the executable using the ‘./CalculatorVerFinal’ command. This will run the program. If the name of the file has changed from the default, replace ‘CalculatorVerFinal’ with the new file name.

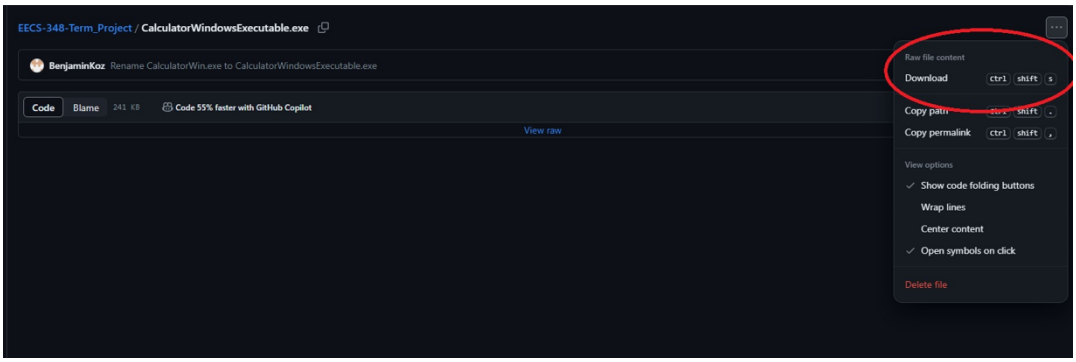


Once these steps have been followed, the user should see the following message in the terminal: “Input the Expression: “. This means the program has begun and the user may now input the expression they would like to be evaluated.

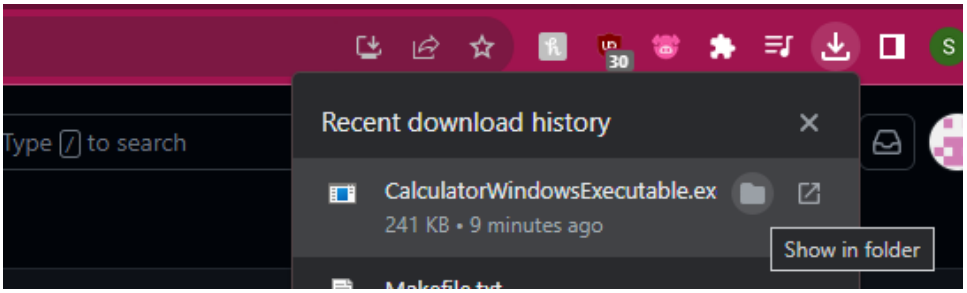
For Windows:

To use the Arithmetic Expression Evaluator on a Windows machine, the user will need to have a C++ compiler installed on their machine. Users can follow this tutorial to install a C++ compiler.

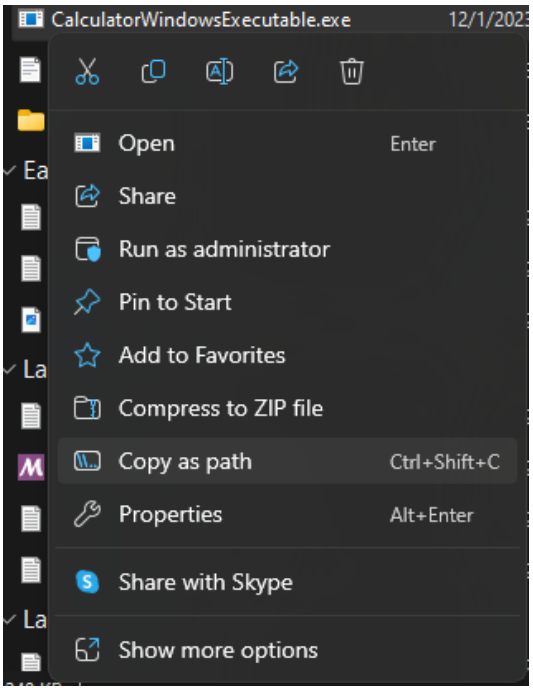
Once the user has a C++ compiler installed, they can navigate to the github and download the CalculatorWindowsExecutable.exe file. To do this, first click on the file name. This should take you to the file on github from where you can download it by clicking the download button under the menu in the top right corner as seen below:



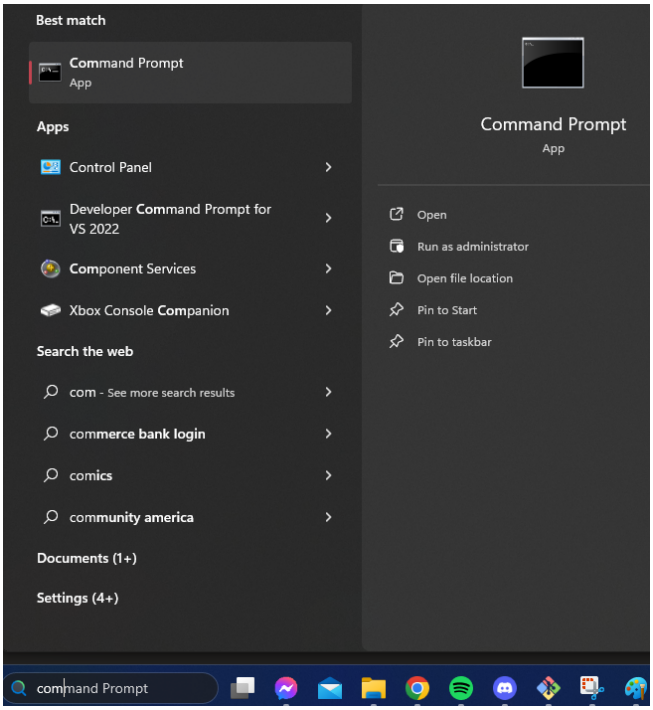
Once the file is downloaded, the user should view where the file is downloaded on their computer. This will most likely be the downloads folder. On chrome, this can be checked by hovering the download icon in the top right of the browser and showing the file in the folder. Click show in folder to open the file explorer to the correct location.



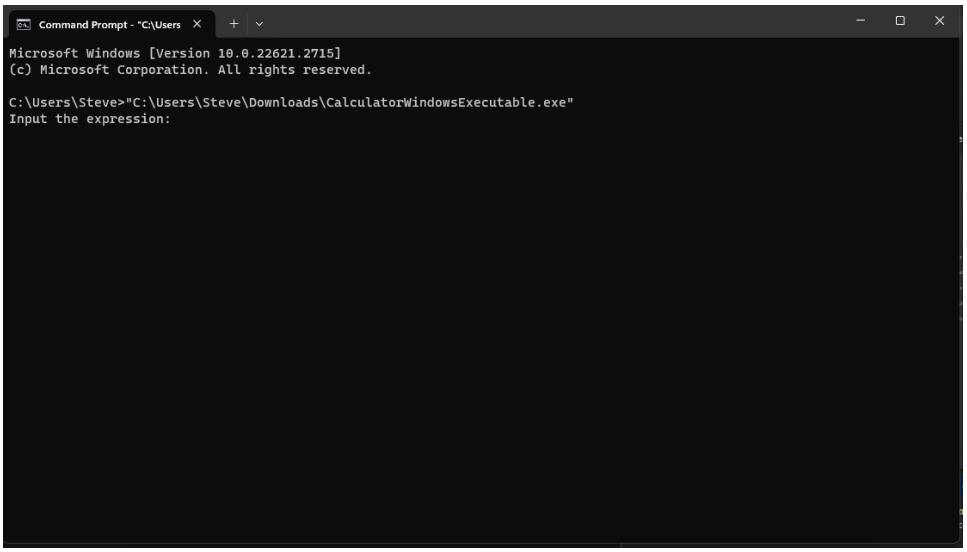
Right-click on the file in the open file location. There should be a copy as path option under the file. Click on that.



Now open a command prompt. To do this, type “command prompt” into your windows search bar.



When the command prompt opens, the user can type ctrl + v to paste the file location into the command prompt and hit enter to run it. It should look like this:



Now the user should be able to enter their expression to be evaluated. If the program needs to be run more than once, the user can simply press the up arrow on their keyboard to copy the command to run program again and hit enter.

3. Accepted Characters

0, 1, 2, 3, 4, 5, 6, 7, 8, 9	Numerical Constants
‘ ’	Space Character
Operand + *Operand*	Addition
Operand - *Operand*	Subtraction
Operand * *Operand*	Multiplication
Operand / *Nonzero_Operand*	Division: Right operand must be non-zero
Integer_Operand % Integer_*Operand*	Modulo: Operands must be integers
Operand ^ *Operand*	Exponentiation
pi	The universal constant pi (3.14159265358979323)
e	The universal constant e (2.71828182845904523536)
(*expression here*)	Parenthesis

4. User Input and Basic Features

When inputting anything into the calculator, the following rules must be followed:

- **Operators** (+, -, *, /, %, ^) should always be followed and preceded by **one** space character. The exception to this is the case in which ‘-’ is being used to represent a negative number. In this case, the ‘-’ should be appended directly to the front of the operand in question.

  Valid Inputs:

  - 1 + 2 -> 3
  - (1 * 4) -> 4
  - (1 * 2) % (3 / 3) -> 0
  - -(1 * -2) -> 2

Invalid Inputs:

- \+ -> Error
- 1 + -> Error
- * 2 -> Error
- 1/1 -> Error

- **Operands** (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, pi, e, ()) must always be tied to one or more operators unless only one operand is entered into the calculator.

Valid Inputs:

- 4 -> 4
- -1 -> -1
- 1 / 2 -> 0.5
- 3 * (5 % 20) -> 15
- pi + e -> 5.8599

Invalid Inputs:

- 1 1 -> Error
- 1 -1 -> Error

- **Parenthesis** () can contain any expression and are typically used for the sake of enforcing an order of operations or adding clarity to an input expression. All opening parentheses '(' must be matched to a closing parenthesis ')' and the number of closing parentheses ')' may never exceed the number of opening parentheses '(' at any given time.

Valid Inputs:

- (1) -> 1
- ((1)) -> 1
- (1 + 1) * (5 + 3) -> 16

Invalid Inputs:

- 1(-> Error
- (5 -> Error
- ((7) -> Error
- 8) -> Error
- (1 * 5) + (1 / 2)) -> Error

- **Order of Operations** will be obeyed by the calculator and will follow the standard convention represented by the following table, with operators of equal precedence being acted upon from left to right:

Precedence	Operators	Description
1	( )	Grouping Symbols
2	^	Exponentiation
3	* / %	Multiplication, Division, and Modulo
4	+ -	Addition and Subtraction

Example Inputs:

- 1 + 2 * 3 -> 7
- (1 + 2) * 3 -> 9
- 1 + 2 ^ 2 -> 5
- (1 + 2) ^ 3 -> 27
- (3 + 2) ^ (1 + 2) -> 125

5. Advanced Features

The calculator will obey the standard order of operations to output mathematically correct answers. We decided to introduce some special features specifically to increase the user's range of possibilities for the Arithmetic Expression Evaluator. One such change is the allowed use of π (3.14159…), which should be denoted as 'pi' in input expressions. Another change is the use of Euler's number, 'e' (2.71828…). If used in an input expression, this should be denoted as 'e'.

6. Troubleshooting

- In the case that the file does not compile, the user should create a new folder and redownload the original files from GitHub.
- In the case that the file still does not compile, the user should attempt to compile the file using their terminal and the 'g++' command.
- If the program executes but raises an error, the user should check their input expression. Make sure the expression abides by the formatting guidelines posted earlier in the document.
    - The most common problems are likely to be improper usage of spaces and grouping symbols.

7. Glossary

- Operator(s)
    - This term encompasses the characters used to evaluate the expressions (+, -, *, /, ect.)
- Operand(s)
    - This term encompasses the items that the operators work on (numbers, e, pi, ect.)

8. FAQs

- Why is the most common error message 'Invalid combinations of operators and operands, or invalid spacing.'?
    - This is due to error handling redundancy that we incorporated to catch fringe case errors.
- What order of operations does the Calculator parse in?
    - The order of operations that is followed is PEMDAS, or Parenthesis, Exponent, Multiplication/Division/Modulo, and Addition/Subtraction. If two operators are equivalent, they are solved from left to right.
- Why make this project when calculators are free online?
    - We were told to for our term project, and it can be fun to meet up with friends and make something together.
- Why isn't my code running? When I click on the executable, it says "The code execution cannot proceed because libgcc_s_she-1.dll was not found. Reinstalling the program may fix this problem."
    - Please make sure you have a GCC compiler installed, and all code is saved to a single file. The code will not run from the executable if you do not have a compiler installed. It also will not run if you have a compiler installed on Visual Studio Code.