

---

**SN<sup>2</sup>ARK**

---

**Arithmetic Parser**

**User's Manual**

**Version 1.1**

Arithmetic Parse	Version: 1.1
User Manual Document	Date: 12/01/2023
006	

## Revision History

Date	Version	Description	Author
11/29/2023	1.0	Individual tasks were assigned to each team member, and everyone started to work on their tasks.	Kusuma M., Sophia J., Navya N., Anna L., Reeny H., Nimra S.
12/01/2023	1.1	The group worked on the final revision together.	Kusuma M., Sophia J., Navya N., Anna L., Reeny H., Nimra S.

Arithmetic Parse	Version: 1.1
User Manual Document	Date: 12/01/2023
006	

## Table of Contents

1. Purpose	4
2. Introduction	4
3. Getting Started	4
4. Advanced features	10
5. Troubleshooting	10
6. Example of uses	11
7. Glossary	12
8. FAQ	12

Arithmetic Parse	Version: 1.1
User Manual Document	Date: 12/01/2023
006	

# User Manual

## 1. Purpose

The purpose of this project is for SN<sup>2</sup>ARK to develop a multifunctional arithmetic expression evaluator utilizing C++. The objective is to create a simulation of a calculator, through a program, that can handle expressions as input and calculate the result using mathematical operations like (PEMDAS). Furthermore, SN<sup>2</sup>ARK will fully integrate the Software Development Process through timely deliverables within the semester. These deliverables include a detailed project plan, a requirements document, a design document aligned with the requirements, a set of test cases, and a fully developed product. As part of the scope of this project, each iteration should be timely and in line with the functionality of the requirements.

The purpose of this *User Manual Document* is to provide a comprehensive guide designed to help the users efficiently use the calculator's functionalities. It provides clear instructions, ranging from accessing the source code to inputting commands, guiding users through using various operators, navigating the user interface, and incorporating various examples. With troubleshooting guidance included, users can confidently operate the Arithmetic parser, maximizing its capabilities. By providing these step-by-step instructions and guidance to handle errors, the manual ensures users can effectively use the Arithmetic Parser. It acts as a reliable reference, enabling users to comprehend the calculator's features, and enhancing their overall experience with the Arithmetic parser.

## 2. Introduction

SN<sup>2</sup>ARK's Arithmetic Parser is a user-friendly arithmetic calculator responsible for evaluating mathematical expressions. It was developed using C++ and it is implemented to be used in a LINUX terminal. The main target audience of this product is students, TAs, and the Professors of the EESC 348 course, hence basic mathematical computation skills are required. The software consists of back-end features such as a tokenizer, parser, evaluator, and error handle, and front-end features such as receiving user input and returning valid values and error handling messages. It consists of specialized advanced features such as PEMDAS, floating value, and specific error messages.

The code resides in the GitHub Repository where it is accessible to the public. Users can utilize our software by cloning our code into their local machines and running the program in a Linux Terminal. The software can be launched with a simple command "make" entered into the Linux Terminal. Detailed instructions are in the next section, called Getting Started. Overall the product is lightweight, easy to install, and ready to use. Details regarding the specifics of using and handling the software are explained in the following sections of the *User Manual Document*.

















## 3. Getting started

*\*Pictures shown in this guide are from a MacBook Pro.*

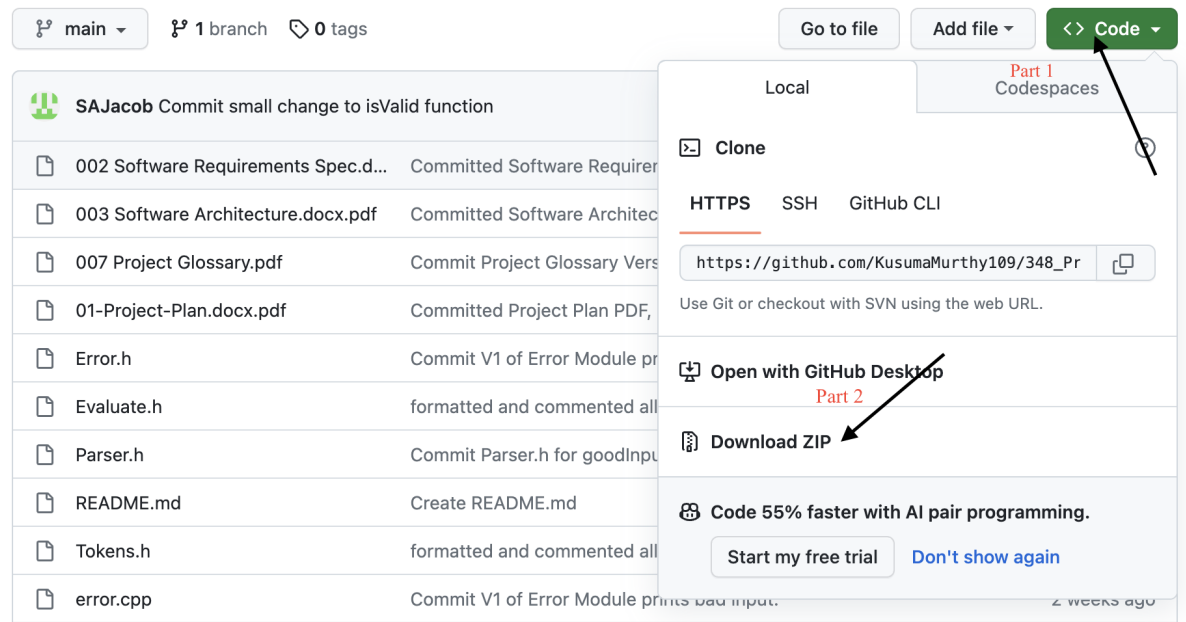
The source code for SN<sup>2</sup>ARK's Arithmetic Parser is housed in GitHub, using the Public Repository link: [https://github.com/KusumaMurthy109/348\\_Project.git](https://github.com/KusumaMurthy109/348_Project.git).

1. First, get access to a working laptop or device that has at least one of these functionalities: a terminal, has the ability to connect to a terminal/cycle server, another type of Linux-based environment, or IDEs such as Visual Studio Code, PyCharm, Replit, Visual Studio Pro, and others.
2. Once this is complete, open up a web browser of the user's choosing and click on the link for the GitHub provided above. It should then open to this page:

Arithmetic Parse	Version: 1.1
User Manual Document	Date: 12/01/2023
006	

	SAJacob Commit small change to isValid function	296a93e last week	77 commits
	002 Software Requirements Spec.d...	Committed Software Requirements Spec Version 1.2	last month
	003 Software Architecture.docx.pdf	Committed Software Architecture PDF, Version 1.1	3 weeks ago
	007 Project Glossary.pdf	Commit Project Glossary Version 1.1	last month
	01-Project-Plan.docx.pdf	Committed Project Plan PDF, Version 1.2	2 months ago
	Error.h	Commit V1 of Error Module prints bad input.	2 weeks ago
	Evaluate.h	formatted and commented all files	2 weeks ago
	Parser.h	Commit Parser.h for goodInput function	last week
	README.md	Create README.md	2 months ago
	Tokens.h	formatted and commented all files	2 weeks ago
	error.cpp	Commit V1 of Error Module prints bad input.	2 weeks ago
	evaluate.cpp	Commit test case changes to evaluate and parser	last week
	main.cpp	Commit Division By Zero and goodInput function	last week
	makefile	Commit actual version of makefile for all modules	last week
	parser.cpp	Commit test case changes to evaluate and parser	last week
	tokens.cpp	formatted and commented all files	2 weeks ago

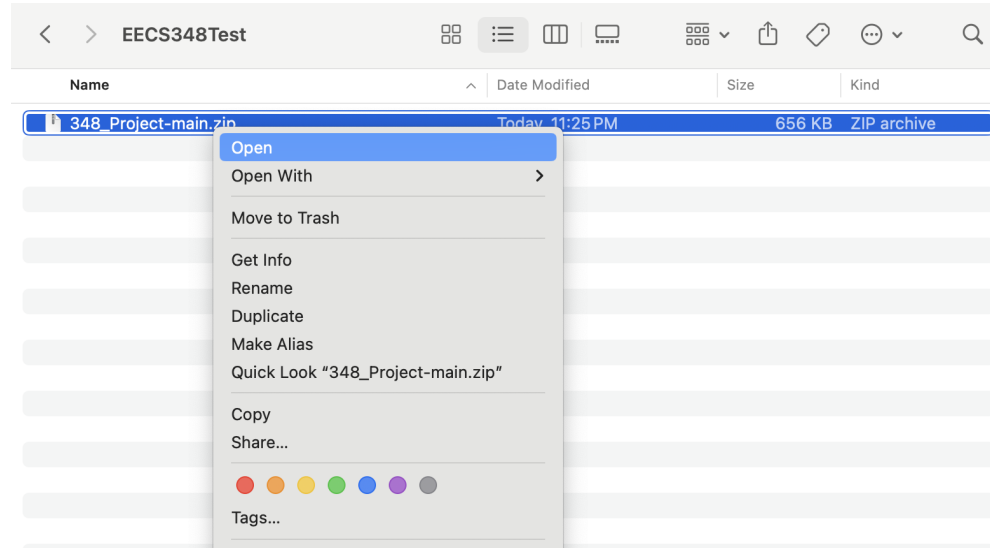
- From here, one can download the GitHub repository onto their local device by clicking on the top right green button called Code, and then clicking on the Download Zip File button, as shown with the arrows.



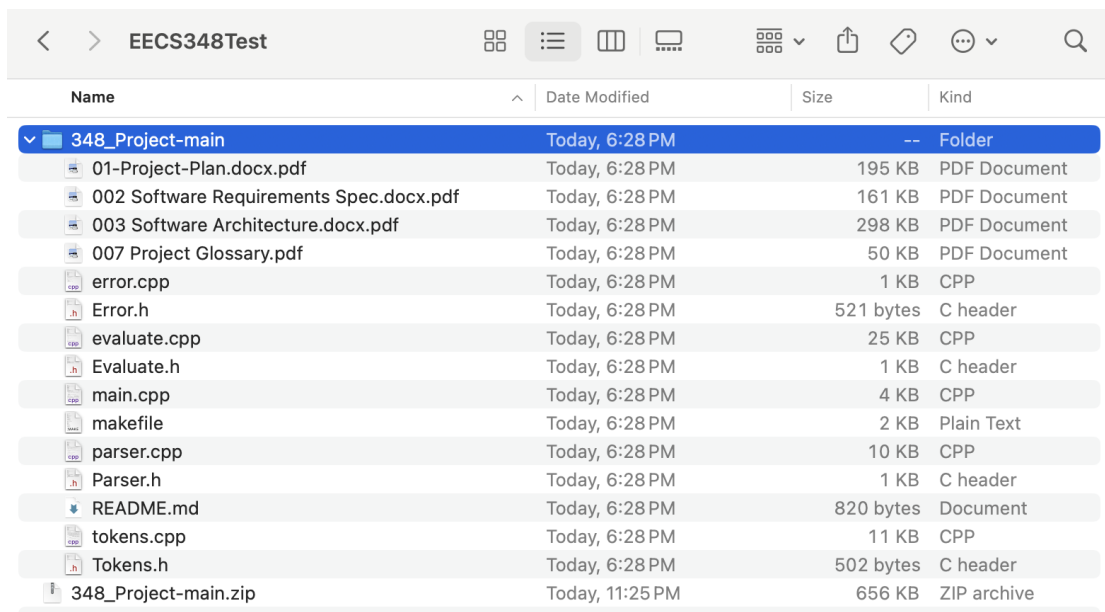
The screenshot shows a GitHub repository page for 'SAJacob Commit small change to isValid function'. At the top right, there are buttons for 'Go to file', 'Add file', and a green 'Code' button. An arrow points to the 'Code' button. Below the repository name, there are tabs for 'Local' and 'Codespaces'. The 'Local' tab is active, showing options to 'Clone' the repository using 'HTTPS', 'SSH', or 'GitHub CLI'. The 'HTTPS' option is selected, showing the URL 'https://github.com/KusumaMurthy109/348\_Pr'. Below the 'Clone' section, there are options to 'Open with GitHub Desktop' and 'Download ZIP'. An arrow points to the 'Download ZIP' option. At the bottom, there is a banner for 'Code 55% faster with AI pair programming' with a 'Start my free trial' button and a 'Don't show again' link.

Arithmetic Parse	Version: 1.1
User Manual Document	Date: 12/01/2023
006	

- Once the Zip is downloaded, go to the folder or area in one's computer that this Zip is to be housed and saved in, as preferred by the user. Unzip the file by double-clicking or right-clicking and selecting the unzip or open button.



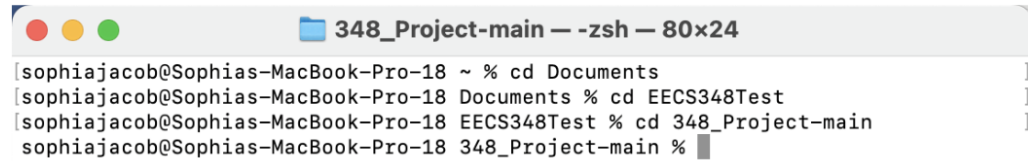
- This should be similar to how the user's folder should look like.



- Now that the user has downloaded the Zip of the source code for this product, go to the terminal either housed on one's machine, in Visual Studio Code, or cycle servers. For example, if the user is a Mac User, then terminal is a pre-installed application that can be opened on the user's laptop.

Arithmetic Parse	Version: 1.1
User Manual Document	Date: 12/01/2023
006	

- After opening the terminal, the terminal needs to enter the folder or area on the local machine where the source code is housed in. To do this, the command “cd [name\_of\_file\_to\_enter]” will be used. For example, if I have housed the 348\_Project-main folder I have downloaded from GitHub in my Documents Folder which houses another folder called EECS348Test, I will use these commands:

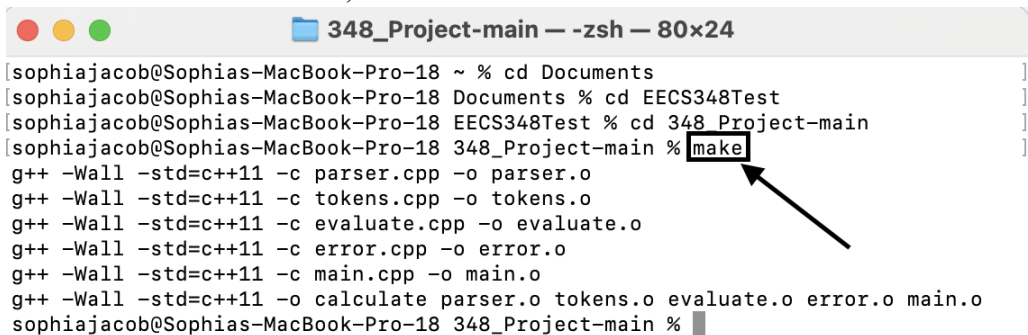


```

348_Project-main — -zsh — 80x24
[sophiajacob@Sophias-MacBook-Pro-18 ~ % cd Documents
[sophiajacob@Sophias-MacBook-Pro-18 Documents % cd EECS348Test
[sophiajacob@Sophias-MacBook-Pro-18 EECS348Test % cd 348_Project-main
[sophiajacob@Sophias-MacBook-Pro-18 348_Project-main %

```

- Now, type the word “make” into the terminal and press the Enter button on the user’s keyboard. This will run a set of commands, which will allow the code to run.

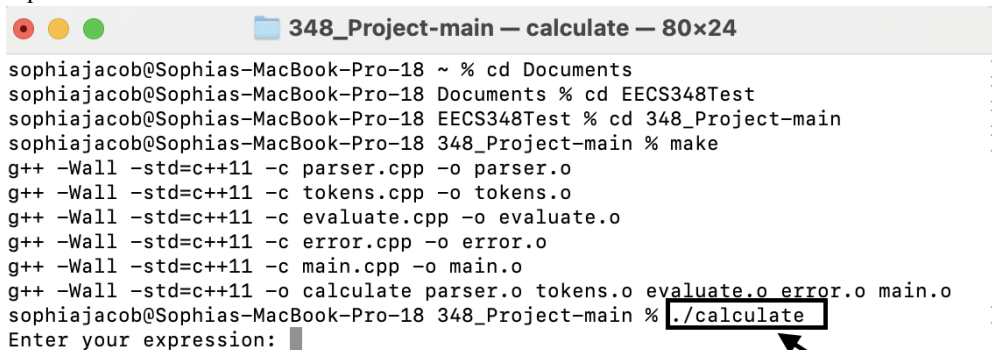


```

348_Project-main — -zsh — 80x24
[sophiajacob@Sophias-MacBook-Pro-18 ~ % cd Documents
[sophiajacob@Sophias-MacBook-Pro-18 Documents % cd EECS348Test
[sophiajacob@Sophias-MacBook-Pro-18 EECS348Test % cd 348_Project-main
[sophiajacob@Sophias-MacBook-Pro-18 348_Project-main % make
g++ -Wall -std=c++11 -c parser.cpp -o parser.o
g++ -Wall -std=c++11 -c tokens.cpp -o tokens.o
g++ -Wall -std=c++11 -c evaluate.cpp -o evaluate.o
g++ -Wall -std=c++11 -c error.cpp -o error.o
g++ -Wall -std=c++11 -c main.cpp -o main.o
g++ -Wall -std=c++11 -o calculate parser.o tokens.o evaluate.o error.o main.o
[sophiajacob@Sophias-MacBook-Pro-18 348_Project-main %

```

- Finally, type in ./calculate into the terminal line. From here, the user will be able to type in their expression to be evaluated.



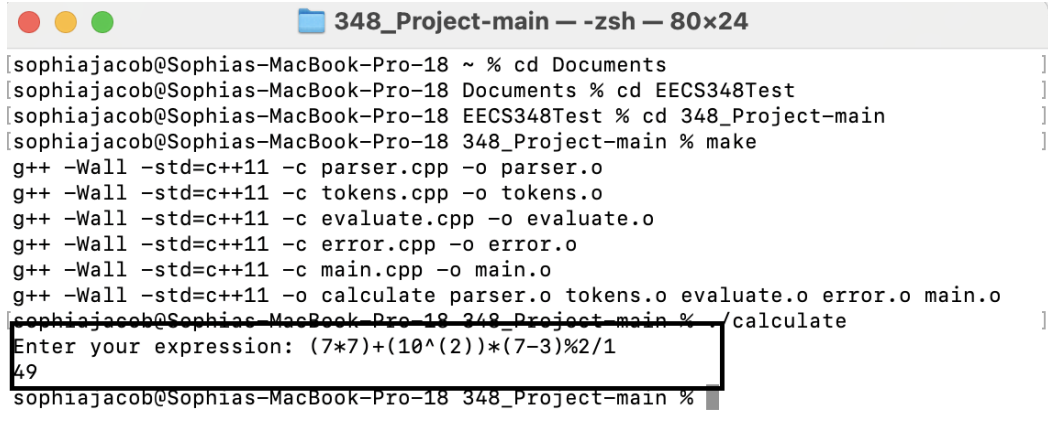
```

348_Project-main — calculate — 80x24
[sophiajacob@Sophias-MacBook-Pro-18 ~ % cd Documents
[sophiajacob@Sophias-MacBook-Pro-18 Documents % cd EECS348Test
[sophiajacob@Sophias-MacBook-Pro-18 EECS348Test % cd 348_Project-main
[sophiajacob@Sophias-MacBook-Pro-18 348_Project-main % make
g++ -Wall -std=c++11 -c parser.cpp -o parser.o
g++ -Wall -std=c++11 -c tokens.cpp -o tokens.o
g++ -Wall -std=c++11 -c evaluate.cpp -o evaluate.o
g++ -Wall -std=c++11 -c error.cpp -o error.o
g++ -Wall -std=c++11 -c main.cpp -o main.o
g++ -Wall -std=c++11 -o calculate parser.o tokens.o evaluate.o error.o main.o
[sophiajacob@Sophias-MacBook-Pro-18 348_Project-main % ./calculate
Enter your expression:

```

Arithmetic Parse	Version: 1.1
User Manual Document	Date: 12/01/2023
006	

10. To enter the expression, type in a mathematical expression of the user's choice, that could normally be inputted into a scientific calculator. Note that this calculator can also do modulus and unary operator calculations that some scientific calculators cannot do. Once this has been typed, then press Enter, and a result (or error message if inputted incorrectly) will display.



```

348_Project-main — -zsh — 80x24
sophiajacob@Sophias-MacBook-Pro-18 ~ % cd Documents
sophiajacob@Sophias-MacBook-Pro-18 Documents % cd EECS348Test
sophiajacob@Sophias-MacBook-Pro-18 EECS348Test % cd 348_Project-main
sophiajacob@Sophias-MacBook-Pro-18 348_Project-main % make
g++ -Wall -std=c++11 -c parser.cpp -o parser.o
g++ -Wall -std=c++11 -c tokens.cpp -o tokens.o
g++ -Wall -std=c++11 -c evaluate.cpp -o evaluate.o
g++ -Wall -std=c++11 -c error.cpp -o error.o
g++ -Wall -std=c++11 -c main.cpp -o main.o
g++ -Wall -std=c++11 -o calculate parser.o tokens.o evaluate.o error.o main.o
sophiajacob@Sophias-MacBook-Pro-18 348_Project-main % ./calculate
Enter your expression: (7*7)+(10^(2))*(7-3)%2/1
49
sophiajacob@Sophias-MacBook-Pro-18 348_Project-main %

```

11. If the user wants to evaluate another expression, they must retype in ./calculate, and then they can input their new expression.
12. Here is a quick guide on how to use the various operators:
- Addition:** The user can add any two numbers, including integers and decimals, together and receive output. They can also have addition as part of a larger expression, as seen in the previous and below examples. The + sign can also be used as an unary operator for numbers. For example, typing in the expression: (+2)+3, is completely valid and will regard the first + as the sign for the 2 digit. A warning on using these unary operators is to make sure that there are adequate parentheses to distinguish between the unary operator and the addition sign. For example, typing in an expression like 2 + + 3, is quite ambiguous, but saying 2 + (+3) is not.
  - Subtraction:** The user can subtract any two numbers, including integers and decimals, together and receive output. They can also have subtraction as part of a larger expression, as seen in the previous and below examples. The - sign can also be used as an unary operator for numbers. For example, typing in the expression: (-2)-3, is completely valid and will regard the first - as the sign for the 2 digit. A warning on using these unary operators is to make sure that there are adequate parentheses to distinguish between the unary operator and the subtraction sign. For example, typing in an expression like 2 - - -3, is quite ambiguous, but saying 2 - (-(-3)) is not. A user can also use unary operators with nested parentheses as shown in the previous example.
  - Multiplication:** The user can multiply any two numbers, including integers and decimals, together and receive output using the \* symbol. They can also have multiplication as part of a larger expression, as seen in the previous and below examples. An example of this can be: 5\*7, or (5+7)\*(3+2). A warning on using the \* symbol is not to have two consecutive operators together with the \* symbol unless it's an unary operator. For example, 5\*+3 is valid, but doing 5\*^2 is not a good input. Also, do not assume that (5)(3) means (5)\*(3), because there must be an operator that separates the two parentheses areas.



Arithmetic Parse	Version: 1.1
User Manual Document	Date: 12/01/2023
006	

- d. Division: The user can divide any two numbers, including integers and decimals, together and receive output using the / symbol. They can also have division as part of a larger expression, as seen in the previous and below examples. An example of this can be:  $5/7$ , or  $(5+7)/(3+2)$ . A warning on using the / symbol is not to have two consecutive operators together with the / symbol unless it's a unary operator. For example,  $5/+3$  is valid, but doing  $5/^2$  is not a good input. Also, do not assume that  $(5)(3)$  means  $(5)/(3)$ , because there must be an operator that separates the two parentheses areas. Another aspect to note is that division by zero is not mathematically allowed. If the user divides by 0, the product will output that it is a bad input due to division by zero.
  - e. Modulus: The user can find the remainder of any two integers, together and receive output using the % symbol. They can also have a modulus or the remainder as part of a larger expression, as seen in the previous and below examples. An example of this can be:  $5\%7$ , or  $(5+7)\%(3+2)$ . A warning on using the %/ symbol is not to have two consecutive operators together with the % symbol unless it's a unary operator. For example,  $5/+3$  is legal, but doing  $5\%^2$  is not a good input. Another aspect to note is that division by zero is not mathematically allowed. Since finding the remainder requires the product to compute the division, having  $5\%0$ , for example, is not allowed. If the user does a modulus by 0, the product will output that it is a bad input due to division by zero. Another warning to note is that if a user is trying to find the remainder of two numbers, both numbers must be integers and not decimals. Finding the remainder of decimals is not allowed in a C++ environment. If the user does make this error, the product will output that it is a bad input due to the Modulus operator requiring integers. For example,  $5.3\%2.3$  is not allowed.
  - f. Parentheses: The user can use extraneous, nested, and necessary parentheses in their expression. For example,  $(5+2)*(3+2)$  is allowed, and  $((5+2)+(3+2))$  is allowed. This functionality can help the user give less ambiguity to their expression and show which parts of the expression should be evaluated first by the product. A warning to the user is to make sure that they have balanced parentheses, for example, parentheses should always be closed and there should be the same amount of parentheses that are open and closed. The product will output bad input due to a Parenthesis Error if there are unbalanced parentheses in the user's expression.
  - g. Operator Precedence: The product ensures that the expression is evaluated with operator precedence, or PEMDAS in mind. For example, doing  $5+2*3^2$ , will evaluate correctly as 23, and not linearly as 441. If the user wants to evaluate differently from the operator precedence, then the user should make sure to add correct parentheses as they see fit.
13. To interpret the results of the Arithmetic Parser, one can verify the accuracy by using an actual calculator or through handwritten math. If there is an error message to be displayed, they will be very detailed as to what went wrong when the user typed in the expression. For example, there are Division by Zero errors, Invalid Character Errors, and Parenthesis Errors to name a few. The user can then take the feedback given by the Arithmetic Parser, and change the expression again to good input that can actually be evaluated. Again, interpreting the results of a good input expression can be done through simple Math checking or through a calculator. If the user gives a good input, the result which is a number (integer or decimal) will display in its own line on the terminal.
  14. Once the user is done with their interaction with the Arithmetic Parser, they can just click the x button on the terminal or close the application/device.

Arithmetic Parse	Version: 1.1
User Manual Document	Date: 12/01/2023
006	

#### 4. Advanced features

SN<sup>2</sup>ARK's Arithmetic Parser contains many features that allow for a smooth and accurate user experience. Essential features include parsing, evaluating, and handling mathematical expressions through operator precedence. It consists of a Tokenizer module that parses user input into a vector of strings, a parser module that checks for valid expressions and raises errors when necessary, and an evaluator module that utilizes stacks to calculate the result of user-inputted mathematical expressions. The software handles basic arithmetic operations such as addition, subtraction, multiplication, division, exponentiation, and modulo, which are not all supported by traditional calculators. For example, expressions with “(“ and “%” could not be computed on a traditional calculator. The Arithmetic Parser can also handle floating-point numbers. The product also includes error handling for scenarios like unbalanced parentheses, consecutive operators, and other syntactic issues. The implementation allows users to input mathematical expressions via the command line and receive accurately evaluated results or error messages accordingly. Considering overall functionality, the Arithmetic Parser adheres to the requirements as specified by the project guidelines.

#### 5. Troubleshooting

Some possible problems include:

1. Compilation errors
  - a. Make sure to have all of these files: main.cpp, error.cpp, evaluate.cpp, parser.cpp, tokens.cpp, Error.h, Evaluate.h, Parser.h, Tokens.h, and makefile
    - i. Any errors associated with not having the correct files can be mitigated by downloading the Zip from GitHub, as specified in the Getting Started Section.
  - b. Make sure you have a C++ compiler, and a terminal to run the product software.
    - i. The terminal, for the best user experience, should be Linux-based or have the ability to connect to KU's Cycle Servers.
  - c. Make sure you are in the correct directory that contains those files
    - i. This can be done by being intentional with where the user downloads and places their Zip file. Make sure to navigate to that folder for the product to run. If the user wants to do this on the terminal, refer to the Getting Started Section for detailed instructions.
  - d. Once you have used the 'make' command, use ./calculate to run the program
    - i. If there is an error encountered on this step, make sure to check that your terminal or area where the product is being run, has a Linux-based environment and/or can handle Linux commands. If this is not a possibility, try typing in other similar "make" commands for the user's specific terminal.
  - e. Copying and pasting expressions onto the command line can result in an error in the way it is copied and pasted.
    - i. This happens very rarely, however, if the user is trying to copy and paste an expression to evaluate from a file into the command line, sometimes the compiler does not understand the copy and paste. Please try to type in the expression manually as frequently as possible to avoid this error.
2. Consistent error messages from the terminal
 

*\*Parts marked in red are the incorrect parts of the expression.*

  - a. Make sure you are using all operands correctly and all parentheses are closed.
  - b. Check the examples below to see the valid types of expressions.
  - c. "Missing or misplaced operands/operator"
    - i.  $(5-6(5)) \rightarrow (5-6*(5))$
    - ii.  $5-1**2 \rightarrow 5-1*2$  or  $5-1^2$
    - iii.  $((15 / 3) + (7 * 2)) - * - (4 + 3) * 2 \rightarrow ((15 / 3) + (7 * 2)) * (- (4 + 3)) * 2$
    - iv.  $3+ / 6-3 \rightarrow 3+6-3$  or  $3/6-3$
    - v.  $36+*45 \rightarrow 36+45$  or  $36*45$
    - vi. make sure no operands are next to each other e.g. ++, -+, \*\*, etc.

Arithmetic Parse	Version: 1.1
User Manual Document	Date: 12/01/2023
006	

- d. "Division by Zero Error"
  - i.  $0.0000000+0.111111/0$
  - ii.  $2/((30\%20)-10)$
  - iii.  $200*3+4/0.000000$
  - iv.  $(5^2+1)/(12*2-6*4)$
  - v. make sure the number after '/' is not zero or equal to zero in parenthesis e.g.  $9/0$ ,  $9/(5-5)$ ,  $9/(10\%2)$
- e. "Invalid Character Error"
  - i.  $0.99+..*0 \rightarrow .99*0$
  - ii.  $-(5x)^3 \rightarrow -(5)^3$
  - iii. make sure only valid operands show up
- f. "Modulus operator requires integers."
  - i.  $5/7\%9+11*3+2^5 \rightarrow 5\%9+11*3+2^5$  or  $7\%9+11*3+2^5$  or an integer before %
  - ii. Make sure the number before % is a whole number e.g. 10, 1 and 1000 but not a decimal e.g. 5.4, 4/2, .0001, (.1+5), or (9-.99)
- g. "Bad Input. Unary operators must be separated by parentheses."
  - i.  $-(-(-5++3)) \rightarrow -(-(-5+(+3)))$
  - ii.  $(5-(-5(+12))) \rightarrow (5-(-(-5+12)))$
  - iii.  $-1--(-90) \rightarrow -1-(-(-90))$
  - iv.  $-(5(2\%(5+1--4))) \rightarrow -(5(2\%(5+1-(-4))))$
  - v.  $3--6+++8 \rightarrow 3-(-6)+(+(+8))$
  - vi. make sure when you want to indicate a positive or a negative number close the number in parentheses e.g. (-5), (+5.9)
- h. "Parentheses Error."
  - i.  $(((((30/6)*(((4+2)-(12-8)*(15/5)) \rightarrow (30/6)*((4+2)-(12-8)*(15/5)))$
  - ii.  $4*((9/3)+(10-2)*(6/2)) \rightarrow 4*(9/3)+(10-2)*(6/2)$
  - iii. make sure you have equal '(' and ')'

## 6. Examples

- Addition
  - Addition can be performed to any combination of integer and decimal values.

```
Enter your expression: 10+20
30
```

- Subtraction
  - Subtraction can be performed to any combination of integer and decimal values.

```
Enter your expression: 10-20
-10
```

- Multiplication
  - Multiplication can be performed to any combination of integer and decimal values.

```
Enter your expression: 10*20
200
```

- Division

Arithmetic Parse	Version: 1.1
User Manual Document	Date: 12/01/2023
006	

- Division can be performed to any combination of integer and decimal values as long as the second operand is not zero.

```
Enter your expression: 10/20
0.5
```

- Modulus
  - Modulus can be performed to any combination of integer and decimal values as long as the second operand is not zero.

```
Enter your expression: 10^20
1e+20
```

- Exponential
  - Exponential can be performed to any combination of integer and decimal values.

```
Enter your expression: 10^20
1e+20
```

- Parentheses
  - To result in a valid number, make sure that all parentheses are closed and there is an equal amount of opening and closing parentheses.

```
Enter your expression: (10+20)*30
900
```

- Unary Operators
  - To properly use unary operators, make sure that the value along with the unary operators is in parentheses as shown below.

```
Enter your expression: ((-10)+(-20))*(-30)
900
```

- Combination of All Operators

```
Enter your expression: ((-10)+(-20))*(-30)/(40%50)^2
0.5625
```

## 7. Glossary of terms

Refer to *007 Project Glossary*

## 8. FAQ

Does the Arithmetic Parser handle complex nested expressions?

- Yes, the Arithmetic Parser handles complex nested expressions as it allows the user to utilize “(“ to set operator precedence. The valid forms of expression are shown in the Getting Started section of the *User Manual Document*.

What operations are invalid?

Arithmetic Parse	Version: 1.1
User Manual Document	Date: 12/01/2023
006	

- a. Operations involving two consecutive “++” or “--” are considered invalid, however, operations including “\*+” or “/-”, and subsequent operations like this are considered valid. Having an unary operator in conjunction with a regular operator is a valid input. Reference the Getting Started section or the Troubleshooting section of the *User Manual Document*.

Can we perform complex integrations which are more than one line?

- a. Yes, there is no set limit of how long the input expression needs to be, however, it needs to abide by the valid expression rules which are stated in the Getting Started or the Troubleshooting section of the *User Manual Document*.

How long does it take to install the software/run the software?

- a. The software runs within 100ms, it is very easy to install and set up. Refer to the Getting Started section of the *User Manual Document*.