EECS 348 Group 16

Arithmetic Expression Calculator
Software Requirements Specifications

Version 1.2

Revision History

Date	Version	Description	Author
13/10/2023	1.0	Document Creation	Sam Muehlebach, Josh Welicky, Jennifer Aber, Jawad Ahsan, Basim Arshad, Mark Kitchin
28/11/2023	1.1	Numbered functional requirements.	Josh Welicky
29/11/2023	1.2	Further detailed the functional requirements.	Sam Muehlebach, Mark Kitchin, Josh Welicky

Table of Contents

Software Requirements Specifications

1. Introduction

1.1 Purpose

The purpose of this document is to outline the expected behavior of the project, the Arithmetic Expression Calculator. This document will include the overall description of the requirements—including functional requirements, non-functional requirements and constraints—specific requirements, and a classification of the functional requirements to elucidate the software requirements of the project.

1.2 Scope

The focus of this document is the documentation of the requirements for the software product, which will evaluate arithmetic expressions and will handle operators +,-,*,/,% and ^, numeric operands, and matching parentheses. This document also contains the details of the use case specifications—in the form of a UML use case diagram—of the product.

1.3 Definitions, Acronyms, and Abbreviations

See the Glossary.

1.4 References

Project Description	—	Can be found in the EECS 348 Canvas page in the project file.
Glossary	—	Can be found in the Appendices section.

1.5 Overview

This *Software Requirements Specification* contains the following information:

Overall Description	—	describes the general factors that affect the product and its requirements. Includes information on interfaces with the product, product functions, user characteristics, constraints, and more.
Specific Requirements	—	describes the specific functional and supplementary requirements of the product. Also contains use case specifications in the form of a UML use case diagram.
Classification of Functional Requirements	—	lists all functional requirements, ordering them by the classifications: essential, desirable, and optional.
Appendices	—	lists the definitions of several terms used in this document.

2. Overall Description

2.1 Product perspective

2.1.1 System Interfaces

Not applicable.

2.1.2 User Interfaces

This software will implement a command line interface (CLI), where users will input an arithmetic expression that satisfies the basic criteria for a valid expression, further outlined in the Project Description document. If the input is valid, the result will be displayed. If not, an error message will be displayed instead.

2.1.3 Hardware Interfaces

Not applicable.

2.1.4 Software Interfaces

This software will be compatible with all mainstream operating systems—such as Windows, macOS, and Linux—and will implement the <iostream>, <string> and <cmath> C++ libraries.

2.1.5 Communication Interfaces

Not applicable.

2.1.6 Memory Constraints

This software will not be on a scale large enough to warrant explicit memory constraints.

2.1.7 Operations

Not applicable.

2.2 Product functions

This product will be able to take in an arithmetic expression as an input. It will parse it and evaluate the expression according to the order of arithmetic operations GEMDAS. The product will obtain input and display output through a user-friendly command-line-interface. The product will be able to detect and respond to invalid arithmetic expressions.

2.3 User characteristics

Users of this product will be familiar with the concepts of basic algebra, such as the order of operations and the concepts of each arithmetic operation. There is no specific subset of individuals who will be using this product, such as students or employees.

2.4 Constraints

This product will be constructed in the C++ programming language using the principles of object-oriented programming. Documentation and comments detailing functionality and logic are required in the source code for this project. All project deliverables will be completed by December 7th, 2023. There is no budget or this project.

2.5 Assumptions and Dependencies

Assumptions: Valid user input, working hardware, compatible OS (Mac, Windows, Linux), and inputted numbers being standard integers or floating-point numbers.
Dependencies: C++ compiling tool, access to standard C++ libraries (<iostream>, <string>, and <cmath>), access to a terminal to run program, and access to a supported operating system.

2.6 Requirements subsets

Not applicable.

3. Specific Requirements

3.1 Functionality

1. Arithmetic Expression Parsing

The system should be able to divide an inputted arithmetic expression into a series of tokens (numbers, operators, parentheses). The system should also be able to convert these tokens from infix ordering to postfix ordering for easy analysis.

2. Addition Operator

If the “+” operator is detected, the system should be able to perform addition on the operand or expression preceding the “+” and after the “+”.

3. Subtraction Operator

If the “-” operator is detected, the system should be able to perform subtraction the operand or expression preceding the “-” and after the “-”.

4. Multiplication Operator

If the “*” operator is detected, the system should be able to perform multiplication the operand or expression preceding the “*” and after the “*”.

5. Division Operator

If the “/” operator is detected in an expression, the system should be able to divide the numeric constant preceding the “/” by the one following it.

6. Modulo Operator

If the “%” operator is detected in an expression, the system should modulo the number immediately before the operator by the number immediately after it.

7. Exponentiation Operator

If the “^” operator is detected in an expression, the system should calculate the exponentiation of the number immediately to the left of the operator to the power of the number immediately to the right of the operator.

8. Process Integer Constants

The system should be able to detect and tokenize any whole number operands present in an inputted arithmetic expression. These numbers can be a single digit in length or multiple digits, but they should be processed as a single token.

9. Process Decimal Constants

The system should be able to detect and tokenize any decimal number operands encountered in an inputted arithmetic expression. These decimal values can include a digit immediately before the decimal point, such as 0.3, or not, such as .3. These operands should be processed as a single token.

10. Match Parentheses

The system should track the number of opening and closing parentheses found during the tokenization of the input. To reject unmatched parentheses, the system should reject input that does not contain an equal number of opening and closing parentheses. If they are not, the system should prompt the user with an error.

11. Utilize Operator Precedence

The system should decide which calculations to prioritize first using the order of operators PEMDAS or GEMDAS. Operations within parentheses should be processed first, then exponentiation, then multiplication and division, then addition and subtraction.

12. Handle Zero Division

When the number 0, or an expression equating 0, is assigned as the denominator of the division or modulus operations, the system should throw an error message to prevent any division by zero.

13. Handle Invalid Expressions

The system should be able to detect errors present in a user's input expression. These include invalid characters like letters, two binary operators placed together without a number to separate, and two numbers placed together without an operator to separate.

14. Process Unary Negation

When a "+" or "-" is detected immediately to the left of a number character, such that no whitespace is between them, and a number is not to the left of the operator, that operator should be considered a unary operator, and should be tokenized along with the rest of the expression and evaluated appropriately.

15. Obtain Input

The system should be able to receive a user's input properly and accurately for calculation through a command line interface. The system should continue to ask the user for input until the command "STOP" is entered.

16. Display Output

The system should be able to calculate the result of the user's entered expression and display it to the output screen in an easily readable format.

*17. ** Operator*

If the "**" operator is present in an expression, the system should be able to calculate the operand to the left of the operator to the power of the operand to the right, and this should be tokenizable to be used with all other parts of the expression.

18. Match Brackets

The system should be able to read a user's operation and recognize if all opening brackets in the operations are matched with closing brackets. If they are not, the system should prompt the user with an error. Extraneous brackets, as in brackets that do not change the way the expression is evaluated, should be allowed without causing an error.

19. ! Operator

If a "!" is detected following a constant, the factorial of the constant should be evaluated and tokenized along with the rest of the expression, assuming the expression is valid and the integer preceding the "!" is a nonnegative integer

20. Fractional Exponents

If an exponent operation is detected with the operation ^ or **, and the power is a fraction, the program should be able to tokenize and evaluate the expression, meaning that it should be able to perform roots of exponents and non-integer exponentiation.

21. Sine Operation

If the "sin(x)" expression is in the operation, where x is an integer or an internal expression, the system should be able to tokenize and evaluate the expression as the Sine of x, allowing this value to be used in combination with all other expressions.

22. Cosine Operation

If the "cos(x)" expression is in the operation, where x is an integer or an internal expression, the system should be able to tokenize and evaluate the expression as the Cosine of x, allowing this value to be used in combination with all other expressions.

23. Tangent Operation

If the “tan(x)” expression is in the operation, where x is an integer or an internal expression, the program should be able to tokenize and evaluate the expression as the Tangent of x, allowing this value to be used in combination with all other expressions.

3.2 Use-Case Specifications

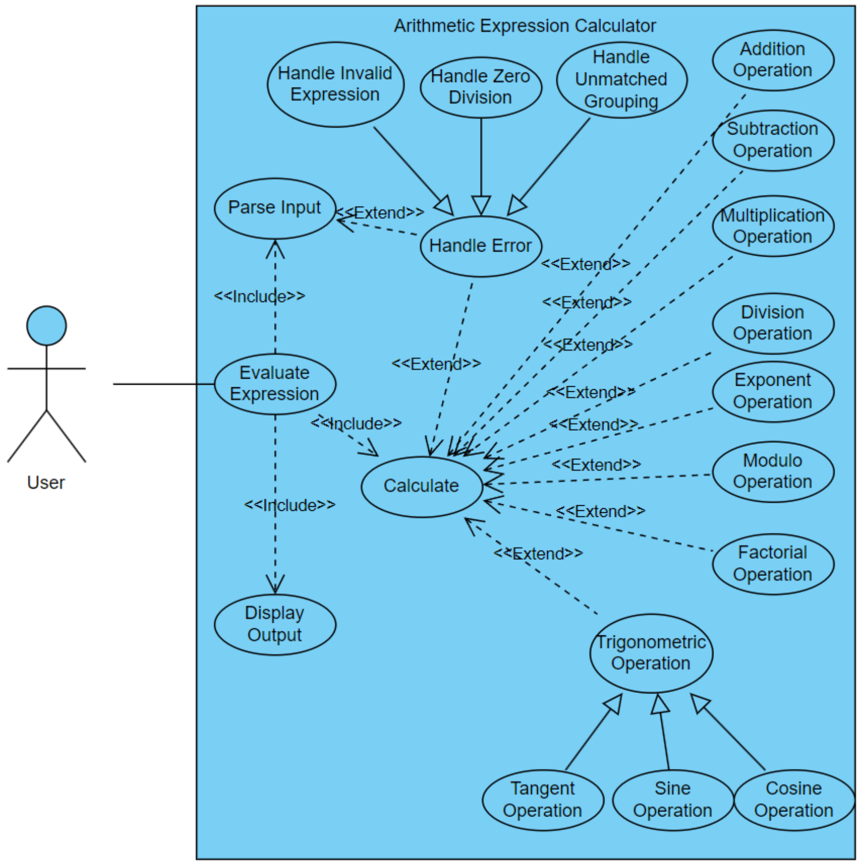

Figure 1

3.3 Supplementary Requirements

This product must be developed in the C++ programming language using data structures and object-oriented programming. This product, and all of its deliverables, must be completed by December 7th, 2023. This product should be compatible with all mainstream operating systems.

4. Classification of Functional Requirements

Functionality	Type
Integer Expression Parsing	Essential
+ Operator	Essential
- Operator	Essential
* Operator	Essential
/ Operator	Essential
% Operator	Essential
^ Operator	Essential
Process Integer Constants	Essential
Process Decimal Constants	Essential
Match Parentheses	Essential
Utilize Operator Precedence	Essential
Handle Zero Division	Essential
Handle Invalid Expressions	Essential
Process Unary Negation	Essential
Obtain Input	Essential
Display Output	Essential
** Operator	Desirable
Match Brackets ([])	Desirable
! Operator	Desirable
Fractional Exponents	Optional
Sine Operation	Optional

Cosine Operation	Optional
Tangent Operation	Optional

5. Appendices

Glossary:

GEMDAS: Acronym for Grouping, Exponent, Multiplication, Add, Subtract. Represents the order of operations when calculating a mathematical expression.

Parsing: The breakdown of a mathematical expression into its individual components.

PEMDAS: Acronym for Parentheses, Exponent, Multiplication, Add, Subtract. Represents the order of operations when calculating a mathematical expression.

Precedence: The order of operations within a given expression, or which parts of the expression are evaluated first. It can be thought of as an order of priority.

UML: An abbreviation for Unified Modeling Language. A visual language used for modeling, specification, and documentation. Used commonly in software engineering.

Unary Negation: Use of a minus sign (-) preceding an integer to indicate a value less than zero

Use Case Diagram: A UML diagram that is commonly used to discover and document the functional requirements of a software solution. A use case diagram maps the interactions between external entities and the system, as well as the purpose and results of those interactions.

***:* The symbol that can represent the exponential operation as a desirable feature.

%: The symbol used in C for the modulus, or remainder, operation.

!: The symbol representing the factorial operation. The factorial of an integer n is the result of the following expression: n! = (n) * (n-1) *…*1.