Fancy Calculator Inc.

Arithmetic Expression Evaluator in C++
Software Requirements Specifications

Version <1.0>

Revision History

Date	Version	Description	Author
<10/02/2024>	<1.0>	SRS has been created	<Luke Reicherter>

Table of Contents

Software Requirements Specifications

1. Introduction

This Software Requirements Specification will detail all software that will be created during the development process of the Arithmetic Expression Evaluator in C++. All valuable information about the requirements of the project and use cases will be outlined in this document.

1.0 Purpose

The purpose of this document is to outline the functions and requirements of the Arithmetic Expression Evaluator in C++. This outline will be used during the development process to determine what functions are essential, desired, or optional. Not every functionality listed will be implemented into the code due to design constraints that will also be described within the SRS.

1.1 Scope

This document will provide a broad overview of each software requirement, and each use case associated with the final product. More information will be added to the SRS as the development process continues. Each requirement must be thoroughly detailed to make the implementation process as efficient as possible. Optional functions will also be detailed.

1.2 Definitions, Acronyms, and Abbreviations

SRS – Software Requirements Specifications
UI – User Interface
Big O Notation – Describes how the performance of an algorithm scales as the input data is increased

1.3 References

GitHub Repository: https://github.com/LukeReicherter/EECS-348-Group-Project

- Includes meeting logs and previous documents from the development process.

Visual Paradigm Online: https://online.visual-paradigm.com/

- Used to create the use case diagram within the Appendices section

1.4 Overview

The SRS contains the details of the software requirements for the product, including descriptions of the product functionality, specific requirements, and general information. The document is organized by topic, with these being: An overall description covering development constraints and general functionality. A section on specific requirements diving into the details about how things such as user interaction will work, including functional requirements and use cases. Following is an overview of classifications of the functional requirements and whether they are essential, desirable, or optional.

2. Overall Description

This section will provide context and background to better understand the requirements for this program. The details provided below will explain things such as functionality, environment, and developmental constraints of the project. Specifications of the program will be provided in section 3.

2.1 Product perspective

2.1.1 System Interfaces

- This program will interact with the user's command line interface.
- Users will be able to input expressions through this interface, and the program will process and evaluate the expression.

2.1.1 User Interfaces

- In the code's base form, the user will interact with the command terminal for the system the program is

being run on.
- Ideally, the user will be able to enter their input in a more polished way, like an input on a website with a detailed description of how the code runs.
- The input line will look like this
    - Enter the expression here:
- The user input will be handled by the top-level main file.

2.1.2 Software Interfaces

- The software will use the standard C++ libraries for mathematical operators, as well as any other libraries deemed necessary throughout the development process.
- This program should run on any device that can support C++

2.1.3 Memory Constraints

- This program should have very minimal memory usage due to the limited number of inputs and outputs being handled.
- To ensure that memory usage is low, we will attempt to implement the fastest and most efficient method of evaluating expressions using "big o notation"
- Recursive functions with O(n) will be the most common type of functions within the code
- A history function might be implemented to the final product. If so, the memory will be determined by how many expressions the user has input.
- The program might need to save only a limited number of inputs/outputs to save memory

2.2 Product functions

- The primary function of the Arithmetic Expression Evaluator is to evaluate arithmetic expressions that are input by the user
- The program will parse, compute, and then output the result of the expression.
- If the user were to input an invalid expression, an error message would display.

2.3 User characteristics

- The user of this program should understand basic arithmetic.
- They should be able to determine what a valid expression looks like.
- A user may use this program as a fast way to determine an expression, instead of doing the problem in their head.
- The user can use this program to compute small parts of a larger and more complex problem (i.e. limits, integrals, derivatives) so that the overall computation time is faster and more efficient than by hand.
- This program will have similar usage to a basic calculator.

2.4 Constraints

- This program must operate within the limitations of the system's memory and must be fast and efficient.
- All errors must be handled properly and a message describing the error should be outputted to the user
    - These errors might include dividing by 0, incorrect amount of parenthesis, and invalid types
- The code is also limited by the standard and external libraries of C++. More functionalities might need to be coded to fill in the gaps.

2.5 Assumptions and dependencies

- This program assumes you are running in a C++ viable environment
- This program assumes the system has enough resources to handle to computation
- It also assumes the user understands basic arithmetic and how to create a valid expression

3. Specific Requirements

This section outlines all the software requirements of the Arithmetic Expression Evaluator in C++. The specifications will ensure that the system functions properly, and developers create the best possible program. The testers will use this to ensure the current implementation follows the requirements. These requirements will be described through various means, which include use cases and alternative specifications.

3.1 Functionality

Core/Main Function

- Starting the program
    - This Function be the first to run within the program
    - In this functions file will be the call to the various C++ libraries used in the code
    - It will also call any other functions that were created in a different C++ file
    - The main function will take no input, and will run upon initialization
- Handling the user input
    - This function will allow the user to input the expression they want evaluated
    - This expression will be stored as a string.
    - The function will then error check the string for any invalid expressions
        - Not all error checks are handled by the main function. (Ex: The parenthesis function will determine if the correct amount of parenthesis is in the expression.)
    - The function will then pass the user's string to the various functions needed to fully evaluate the input (mainly the parenthesis function).
- Output
    - Once the expression has been evaluated, the answer will be output to the terminal.
    - This output should be formatted in a way that is easy to understand for the user

Parenthesis/Order Function

- Error Check
    - This function will take the input sent from the main function
    - It will then determine if the correct amount of parenthesis are present
    - It will also determine if the parenthesis are in the correct order (Ex: ")))(((which is invalid)
- Determine Order
    - This function will find the "innermost" expression according to PEMDAS
    - It will then send that section of the string to one of the PEMDAS functions listed below
    - The user's input string will then be altered to reflect the completed PEMDAS functions answer (Ex: "7/ (1+7)" becomes "7/ (8)")
    - The next "innermost" function is found, and the process repeats until the expression is completely evaluated

- Send Output
    - This function will return the evaluated expression to the main function

PEMDAS Functions

- + (Addition)
    - Users must be able to add any given numbers (or equations) of their choice.
    - This function takes in two (or more) numbers/equations and outputs the sum.
- - (Subtraction)
    - Users must be able to subtract any given numbers (or equations) of their choice.
    - This function takes in two (or more) numbers/equations and outputs the difference.
- * (Multiplication)
    - Users must be able to multiply together any given numbers (or equations) of their choice.
    - This function takes in two (or more) numbers/equations and outputs the product.
- / (Division)
    - Users must be able to divide any given numbers (or equations) of their choice.
    - This function takes in two (or more) numbers/equations and outputs the quotient.
- % (Modular division)
    - Users must be able to get the remainder of a division equation of their choice
    - This function takes in two (or more) numbers/equations and outputs the remainder of dividing them.
- ** (Exponentiation)
    - Users must be able to raise any given number or equation of their choice to a power of their choice.
    - This function takes in two (or more) numbers/equations and outputs the first raised to the power of the second.

3.2 Use-Case Specifications

Parse Arithmetic Expression

- This use case describes how the system parses an arithmetic expression entered by the user
- Preconditional: The user has a valid input
    1. The user inputs an expression
    2. The evaluator sends the input to the Parenthesis/Order Function
    3. The “inner-most” expression is found and sent to the corresponding PEMDAS function
    4. Process repeats until expression is fully evaluated

Display Expression Output

- This use case describes how the system outputs the expression
- Preconditional: The expression has been fully evaluated

1. The Parenthesis/Order Function sends the evaluated expression to the Main Function
2. The system displays the result to the user's screen

Handle Bad Input

- This use case describes what will happen if the evaluator detects bad input
- Precondition: Bad input is provided by the user
    1. The user inputs an invalid expression
    2. An error check detects an invalid input
    3. The error check prints the error type to the user's screen

View History

- This use case is optional and does not necessarily represent the final product
- It describes how the user can view previous inputs and outputs
    1. The user chooses to view history
    2. The screen will display all previous inputs and outputs
    3. The user can optionally choose to delete the history to save memory

3.3 Supplementary Requirements

Performance Requirements

- The time complexity should be kept to a minimum
- Ideally, the parsing and evaluation should complete with linear time O(n)
- This will allow for the result to be displayed quickly

Usability Requirements

- The user should be able to easily understand how to input expressions
- Error messages should be detailed
- The input should not be too strict
    - For example: Too many spaces

Reliability Requirements

- Each expression entered by the user should be evaluated to the correct result
- The program should not crash if the user inputs an invalid expression
- These goals will be achieved by thoroughly testing the program

Maintainability Requirements

- The code will need to be properly documented
- Modular design to allow for easier code readability
- New iterations of code will be pushed to GitHub with proper naming conventions

4. Classification of Functional Requirements

Functionality	Type
"PEMDAS" Functions - All math operators, need to work in order according to PEMDAS	Essential
Parenthesis/Order Function - Will keep track of parenthesis - Determines if the amount of parenthesis is correct	Essential
Core/Main Function - Will be the superclass of the Arithmetic Expression Evaluator - Takes user input and calls the subclasses (ex: PEMDAS Functions)	Essential
UI/ User Interface - Formatting Functions that make the input/output look "cleaner" - Stores previous line clears, allows user to see previous inputs/outputs	Desirable
Floating-Point Support - This will allow the user to input decimal numbers - Result of expression will be more accurate	Desirable
Hosting - Website, domain, terminal, cloud, etc.	Desirable
Additional Calculator Functions - Trig Functions - Logarithmic Functions - Sqaure root, absolute value	Optional

5. Appendices

Appendix A: Sample Input and Output

The following examples are to be used as test cases for the Arithmetic Expression Evaluator. The format of these outputs will not represent the final design.

- Input: 8 * 9 + 1
    - Output: 73
- Input: (8 + 9) * (7 * 2)

- Output: 238

- Input: 7 – 4 + 6 / 2 * 4
  - Output: 15
- Input: 7 / (4 – 2)
  - Output: 3.5

Appendix B: Sample Error Messages

The following list provides error conditions that must be checked within the code. This list will be added to as more possible errors are found.

- Input: 7 / 0
  - Output: Divide by Zero Error
- Input:)7+0(()
  - Output: Mismatched Parenthesis Error
- Input: (8 h 9)
  - Output: Invalid Characters

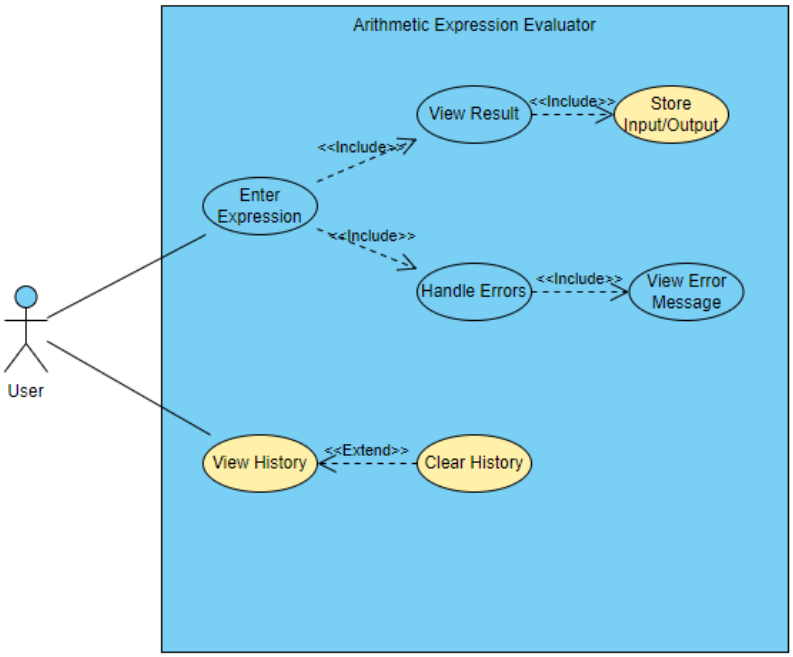
Appendix C: Possible Operators

The following list represents all possible operators that could be implemented in the code. Not all operators are essential to the completion of the project.

- Basic Arithmetic: +(Addition), -(Subtraction), *(Multiplication), /(Division), **(Exponentiation)
- Trigonometric Functions: sin(), cos(), tan(), arcsin(), arccos(), arctan()
- Logarithmic Functions: ln(), log()(Base 10)
- Miscellaneous: sqrt(), abs()

Appendix D: Use Case Diagram

The following diagram represents each use case for the program. Use cases that are yellow are subject to change and may not be present within the final product. The diagram was made using visual paradigm online.


Arithmetic Expression Evaluator
View Result
<<Include>>
Store Input/Output
<<Include>>
Enter Expression
<<Include>>
Handle Errors
<<Include>>
View Error Message
User
View History
<<Extend>>
Clear History