Kansas Instruments™ (not really)

Software Requirements Specifications

Version 1.4

Revision History

Date	Version	Description	Author
9/26/2023	1.0	Document has been created and work on sections one, two, and three began.	Benjamin Kozlowski, Steve Gan, MJ McGee, Jacob Leehy, Nicholas Hausler
10/2/2023	1.1	Work on section one finalized, work on section two assigned for completion.	Benjamin Kozlowski, Steve Gan, MJ McGee, Jacob Leehy, Nicholas Hausler
10/6/2023	1.2	Work on section three began, use case specification and diagram completed.	Benjamin Kozlowski, Steve Gan, Jacob Leehy, Nicholas Hausler
10/10/2023	1.3	Work on descriptions for functionality	Benjamin Kozlowski, Steve Gan, MJ McGee, Jacob Leehy, Nicholas Hausler
10/13/2023	1.4	Final review and polishing before submission.	Benjamin Kozlowski, Steve Gan, MJ McGee, Jacob Leehy

Table of Contents

Software Requirements Specifications

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification is to fully describe the behavior, requirements, constraints, and other requirement aspects of the *Arithmetic Expression Evaluator in C++*. This includes everything needed to provide a comprehensive description of the requirements of the software and the user interactions.

1.2 Scope

This document applies to the *Arithmetic Expression Evaluator in C++* as defined in the product description file. The project is associated with a use case model which can be accessed via header 3.2 and is associated with a project design document which is also contained within a GitHub repository dedicated to the project.

1.3 Definitions, Acronyms, and Abbreviations

SRS - Software Requirement Specification

EECS - Electrical Engineering and Computer Science

KU - University of Kansas

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.

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

1.5 Overview

The remainder of the SRS will document various expected behaviors for the calculator that will be produced. It will include descriptions of the ways in which a user may interact with the system, including user interfaces and software functionality. It will also document a detailed list of required functionalities and constraints, categorizing them based on their level of importance and type of interaction with the system. Using these requirements, a use case model will be developed to grant a visual representation of the system and its features which can be seen in section 3.2.

2. Overall Description

2.1 Product perspective

2.1.1 System Interfaces NA

2.1.2 User Interfaces

The user interface for the calculator will be a simple prompt-output-based interface. The user will receive a prompt to enter their desired expression and then the program will parse the expression and print the desired output to the screen.

2.1.3 Hardware Interfaces NA

2.1.4 Software Interfaces

The software interface for the calculator will involve receiving a string from the scanner function. The string will then need to be parsed. If there is an error, then an exception must be raised. The string will need to be divided up and delegated to a stack object with priority given to the order of operations. The program will iterate through the stack, afterwards returning the result back to the user through a print function.

2.1.5 Communication Interfaces NA

2.1.6 Memory Constraints NA

2.1.7 Operations NA

2.2 Product functions

The product is meant to be able to perform basic arithmetic with input. The arithmetic will include division, multiplication, modulo, subtraction, addition, and use of parenthesis to organize problems. The product will solve a problem and the output will be the solution for the arithmetic problem.

2.3 User characteristics

This product is meant to be able to be used by anyone with the intention of evaluating basic expressions including addition, subtraction, multiplication, and more. No specific knowledge of any kind is required outside of basic mathematical knowledge. Developers looking to work on or expand on this software should have a solid understanding of the basics of C++.

2.4 Constraints

The program will be coded solely in C++. It must be able to handle errors in input without crashing. It will provide an easily understandable interface for the user to interact with. The program must be able to run on the lab computers provided to the students in KU's School of Engineering.

2.5 Assumptions and dependencies

Users will be assumed to be familiar with basic input methodologies such as those defined in section 2.1

of the SRS. It is also assumed that users will possess the hardware needed to operate the program that will be provided in our final submission. Correct outputs from this program will be dependent upon correct user input, though some level of user error-catching will likely be implemented in the final project.

2.6 **Requirements subsets NA**

3. Specific Requirements

3.1 **Functionality**

The program will have a single overall purpose, which is to take in an expression given by the user and output the correct solution to that expression. This will be achieved by combining many different smaller functionalities and having them work in tandem with one another to produce the desired output.

3.1.1 User Input

The program will be able to receive user input in the form of a basic prompted input via a keyboard.

3.1.2 Output

The program will output the correctly calculated information to the console.

3.1.3 Order of Operations

The program will be able to determine operator precedence and parse expressions according to the standard order of operations. This will abide by the following order, where operations of equal precedence are operated on from left to right:

1. Grouping Symbols (parenthesis (), brackets [], braces {})
2. Exponentiation (^ and **)
3. Multiplication, Division, and Modulo (* and / and %)
4. Subtraction and Addition (- and +)

3.1.4 Addition (+)

The program will be able to perform computations involving the addition of multiple terms using the ‘+’ operator. (Tier 4 precedence)

3.1.5 Subtraction (-)

The program will be able to perform computations involving the subtraction of multiple terms using the ‘-’ operator. ‘-’ will function as subtraction when spaces are included between operands to avoid confusing subtraction with negation. (Tier 4 precedence)

3.1.6 Multiplication (*)

The program will be able to perform computations involving the multiplication of multiple terms using the ‘*’ operator. (Tier 3 precedence)

3.1.7 Division (/)

The program will be able to perform computations involving the division of multiple terms using the '/' operator. (Tier 3 precedence)

3.1.8 Modulo (%)

The program will be able to perform computations involving the modulo of multiple terms using the '%' operator. (Tier 3 precedence)

3.1.9 Exponents (^)

The program will be able to perform computations involving exponential expressions using the '^' character. (Tier 2 precedence)

3.1.10 Exponents (**) - Desired

The program will be able to perform computations involving exponential expressions using the '**' operator. (Tier 2 precedence)

3.1.11 Parenthesis (())

The program will be able to perform computations involving parenthesis represented as '()'. A negative sign in front of an expression should be interpreted as multiplying that expression by -1. (Tier 1 precedence)

3.1.12 Brackets ([ ]) - Desired

The program will be able to perform computations involving brackets represented as '[]'. A negative sign in front of an expression should be interpreted as multiplying that expression by -1. (Tier 1 precedence)

3.1.13 Braces ({}) - Desired

The program will be able to perform computations involving braces represented as '{}'. A negative sign in front of an expression should be interpreted as multiplying that expression by -1. (Tier 1 precedence)

3.1.14 Error Handling

The program should be able to handle user errors without crashing, instead prompting the user for reinput.

3.1.15 Signed Integer Compatibility

The program must be able to compute expressions involving integers.

3.1.16 Floating Point Number Compatibility – Desired

The program should be able to compute expressions involving floating point values expressed using '.'.

3.1.17 *Numerical Constant Handling*

The program shall be able to handle certain numerical constants such as π and e.

3.2 Use-Case Specifications

There is only one activity done by the user, which is entering the expression. The rest of the work is handled by the program itself.

Use Case: Enter Expression

Scope: Arithmetic Expression Evaluator in C++

Level: User Goal

Context: The user wants a solution to their expression, so they enter it into the calculator.

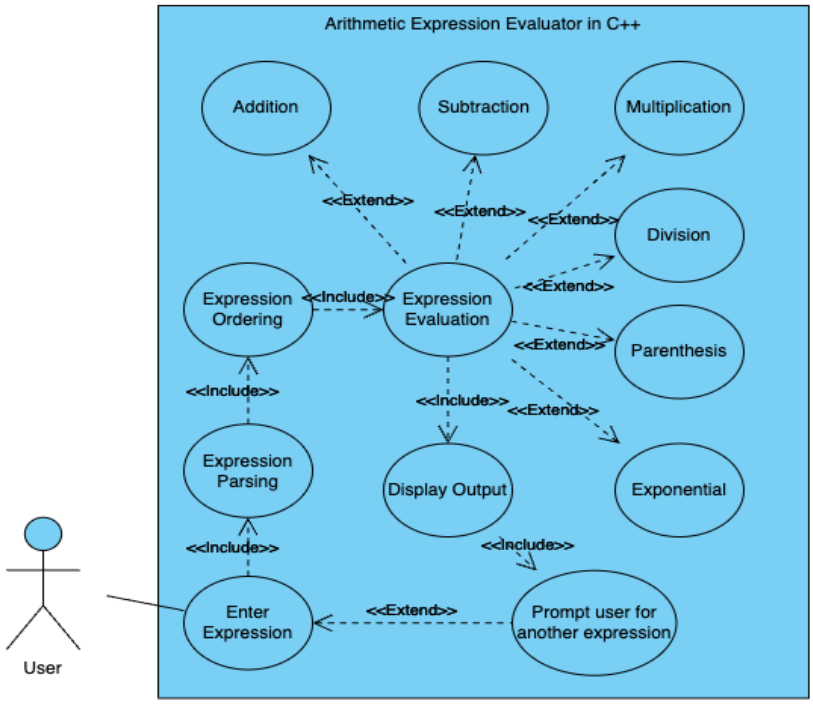
Multiplicity: A user may input one expression at a time.

Primary Actor: User

Supporting Actors: None

Main Success Scenario:

1. User enters their expression.
2. The expression is parsed according to parenthesis or other grouping methods.
3. The expression is evaluated in the proper order of operations.
4. If the expression calls for a certain type of evaluation (e.g., addition, subtraction, ect.), it is properly carried out.
5. The correct solution to the expression is returned.
6. The user is prompted for another expression.



3.3 **Supplementary Requirements**

The program must remain stable such that users may interact with it freely without handling errors.

The program must be functional on the Eaton lab computers provided by the KU EECS department.

4. Classification of Functional Requirements

Functionality	Type
User Input	Essential
Output	Essential
Order of Operations	Essential
Order of Operations	Essential
Addition using ‘+’	Essential
Subtraction using ‘-‘	Essential
Multiplication using ‘*’	Essential
Division using ‘/’	Essential

Modulo using ‘%’	Essential
Exponential Operation using ‘^’	Essential
Parenthesis using ‘( )’	Essential
Error Handling	Essential
Signed Integer Compatibility	Essential
Numeric Constants (i.e., π, e)	Essential
Brackets using ‘[]’	Desired
Braces using ‘{}’	Desired
Exponential Operation using ‘**’	Desired
Floating Point Values using ‘.’	Desired

5. Appendices

Visual Paradigm was used for the use-case diagram (not a part of requirements).