Group 32

Arithmetic Expression Evaluator in C++
Software Development Plan
Software Requirements Specifications

Version 2.1

Revision History

Date	Version	Description	Author
09/10/23	2.1	Opened document and added basic information.	Jordan B., Ryan G., Derek N.
14/10/23	2.3	Work done to complete section 3 and edits made to sections 1 and 2	Derek N.
15/10/23	2.4	Completed section 2.	Derek N.
15/10/23	2.5	Completed section 1, proof-reading.	Priyatam N., Manu R., Jordan B.

Table of Contents

Software Requirements Specifications

1. Introduction

1.1 Purpose

This document serves as a comprehensive outline of the project's specific requirements, functional and nonfunctional. The product will be a C++ program that can parse and evaluate arithmetic expressions containing the operators +, -, *, /, %, and ^ as well as numeric constants. The program will be able to handle the use of parentheses as well. The finalized product will be accompanied by an instructional user manual.

1.2 Scope

This document applies to the creation of the Arithmetic Expression Evaluator, defining necessary functional and nonfunctional requirements. These include requirements such as CLI communication between the program and the user, the design and implementation of the program's planned functions in C++ (expression parsing, operator precedence, parentheses handling, etc.) and the requirements of the user manual.

1.3 Definitions, Acronyms, and Abbreviations

- SRS - Software Requirements Specifications
- PEMDAS - "Parenthesis Exponent Multiplication Division Addition Subtraction" Order of operations
- CLI - Command-line Interface
- C++ - A high level programming language, which the Arithmetic Expression Evaluator will be written in.
- Arithmetic - Relating to operations on numbers. In this project, it refers to the abilities of the calculator.

1.4 References

Project Description (09/07/23, Professor Hossein Saiedian, University of Kansas)
Project Plan (09/15/23, Professor Hossein Saiedian, University of Kansas)

1.5 Overview

The document provides a description of the project and the functional and nonfunctional requirements in the following sections:

- Overall Description: Describes product features, requirements, characteristics, constraints, and interfaces.
- Specific Requirements: Provides an in-depth explanation of the functionality and requirements of each system element.
- Classification of Functional Requirements: A list of all functional requirements ordered by type.

2. Overall Description

2.1 Product perspective

2.1.1 System Interfaces

The program does not include system interfaces.

2.1.2 User Interfaces

The user will interact with the command line to input arithmetic expressions to be evaluated. This is the sole means of user communication with the program.

2.1.3 Hardware Interfaces

The system does not include hardware interfaces.

2.1.4 Software Interfaces

The software will utilize the command line interface for users to input arithmetic expressions to be evaluated.

2.1.5 Communication Interfaces

The program does not include communication interfaces between machines.

2.1.6 Memory Constraints

This program will not have any hard constraints on memory.

2.1.7 Operations

This program will be able to perform the following operations, with all operation precedence decided according to the PEMDAS standard.

- Addition and Subtraction
- Multiplication and Division
- Exponentiation and Modulo Division
- Unary Negation and Addition
- Any combination of the above operations with extraneous or necessary parentheses.

2.2 Product functions

This program will be able to perform the following functions:

Expression Parsing:
- Implement a function to tokenize the input expression
- Create a data structure to represent expression

Operator Support and Precedence:
- Determine the precedence of operators according to the PEMDAS rules, while using the operators: + (addition), - (subtraction), * (multiplication), / (division), % (modulo), ^ (exponentiation).

Parenthesis Handling:
- Evaluate expressions within parentheses in the correct order.

Numeric Constants:
- Recognize the numeric values in the input expression, which are assumed to be integer values.

User Interface:
- Utilize a user interface that allows for easy input and legible output.

Error Handling:
- Handle errors that would produce invalid or impossible results (such as division by zero or invalid input expressions by the user.)

2.3 User characteristics

The user base will mostly be composed of the graders of the project. Therefore they are expected to be competent in using technology like the command line interface to input expressions,.They are not expected to have characteristics that will require the project to be configured any differently than expected.

2.4 Constraints

- The project must be developed in C++.
- The project will be completed by the stated deadline (see 01: Project Management Plan.)
- The project will use object-oriented programming principles as the structure of the code.

2.5 Assumptions and dependencies

Assumptions:

- The input's numeric values will be integers.
- The input expression will be of type string.

2.6 Requirements subsets

- Must be capable of handling excessive parentheses.

3. Specific Requirements

The Arithmetic Expression Evaluator in C++ Project has multiple requirements. The program itself must be written in C++, and must have the following functions:

- The program should solve expressions using +, -, *, /, %, and ^, and numeric constants.
- The program should also be able to decipher parentheses and read the input in the correct order using PEMDAS.
- The program should always return the correct values according to the user's input.

3.1 Functionality

3.1.1 Addition Operator (+)

The program should be able to interpret strings using the operator "+", which calls for the addition of two values, and then return the correct answer. For example, the input "2+2" should yield a numerical output of 4.

3.1.2 Subtraction Operator (-)

The program should be able to interpret strings using the operator "-", which calls for the subtraction of two values, and then return the correct answer. For example, the input "2-2" should yield a numerical output of 0.

3.1.3 Multiplication Operator (*)

The program should be able to interpret strings using the operator "*", which calls for the multiplication of two values, and then return the correct answer. For example, the input "2*5" should yield a numerical output of 10.

3.1.4 Division Operator (/)

The program should be able to interpret strings using the operator "/", which calls for the division of two values, and then return the correct answer. For example, the input "10/2" should yield a numerical output of 5.

3.1.5 Modulo Operator (%)

The program should be able to interpret strings using the Modulo operator "%", which calls for the division of two values, and then return the remainder of that operation. For example, the input "10%3" should yield a numerical output of 1.

3.1.6 Exponential Operator (^)

The program should be able to interpret strings using the operator “^”, which calls for one value to be raised to the exponential value of another. For example, the input “5^5” should yield a numerical output of 25. At this point, only “^” will be used to refer to the exponential operator.

3.1.7 *Numerical Constants*

The program should be able to interpret strings using numeric constants as values. At this point, only integer inputs will be accepted by the program. However, non-integer outputs can be returned.

3.1.8 *Parenthesis Handling*

The program should be able to handle expressions using parentheses and interpret them in the correct order. For example, (5+(10*5)) should return 55. PEMDAS rules will be assumed (see 3.1.9).

3.1.9 *Operator Precedence*

The program should interpret the inputs in PEMDAS order, with parentheses taking precedence over exponential functions, which takes precedence over multiplication/division functions, etc.

3.1.10 *Expression Parsing*

The program should be able to interpret complex inputs with respect to their structure and precedences.

3.1.11 *Error Handling*

The program should have detailed and informative error handling to manage input or logical errors and explain them to the user.

3.1.12 *User Interface*

Utilizing the command line interface, the program should have a user-friendly input system that has a legible and easy-to-understand output.

3.2 Use-Case Specifications

The potential use cases for the Arithmetic Expression Evaluator are as follows:

- Addition
- Subtraction (with and without parentheses)
- Multiplication
- Division
- Exponentiation
- Mixed Operators
- Complex addition with Extraneous Parentheses
- Mixed Operators with Extraneous Parentheses
- Nested Parentheses with Exponents
- Combination of Extraneous and Necessary Parentheses
- Extraneous Parentheses with Division
- Combining Unary Operators with Arithmetic Operations
- Unary Negation and Addition in Parentheses
- Negation and Addition with Negated Parentheses
- Unary Negation and Exponentiation
- Combining Unary Operators with parentheses
- Invalid Expressions

3.3 Supplementary Requirements

- The project must have a user manual or README file explaining how to use the program, including example inputs and outputs.

- Unit tests must be developed to verify the accuracy of the expression evaluator.
- The project code will include comments/documentation to explain the logic and functionality of programming decisions.

4. Classification of Functional Requirements

Functionality	**Type**
Expressing Parsing	Essential
Operator Support (+, -, *, /, %, ^)	Essential
Parenthesis Handling	Essential
Numeric Constants	Essential
Operator Precedence	Essential
User Interface	Essential
User Manual	Essential
Error Handling	Essential
Floating Point Input Support	Desirable
“**” as Alternative Exponential Operator	Desirable

5. Appendices

The appendices are not considered part of the requirements for this document.

-