SN²ARK CO.

Arithmetic Parser
Software Architecture Document

Version <1.1>

Revision History

Date	Version	Description	Author
11/03/2023	1.0	Team members completed all of the 003 Software Architecture Document.	Sophia Jacob, Reeny Hunag, Navya Nittala, Kusuma Murthy, Anna Lin, Nimra Syed
11/10/2023	1.1	Team members edited sections 1.5 and 8, removed the blue text (instructions given by the professor), and lastly proofread the document.	Sophia Jacob, Reeny Hunag, Navya Nittala, Kusuma Murthy, Anna Lin, Nimra Syed

Table of Contents

Software Architecture Document

1. Introduction

The general overview of the ***Software Architecture Document*** is laid out in the Introduction Section. This will give a comprehensive guide of how SN^{2}ARK will proceed with the project's implementation phase, with careful consideration to design. Specifically, the Introduction Section will contain the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the ***Software Architecture Document***.

1.1 Purpose

The purpose of this project is for SN^{2}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^{2}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 ***Software Architecture Document*** is to provide developers, stakeholders, and users with a clear picture and understanding of the division of the software components. It provides a vision of how the system is to be designed and what particular features are included in each of the components. It provides developers with a clear structure of how the software should be developed and as a result, how it should behave. It is intended that the group is to use classes and Object-Oriented Programming behavior of the C++ language. The architecture decision that SN^{2}ARK has made on the system is to fully complete the functionality of the Arithmetic Parser and have the user interact through the command line. If the team has time, they will try to implement a GUI or an interface for the user to interact with.

This document identifies the components that are necessary for developing this system. It also defines the structure and properties of the components that are involved in the system and also the interrelationships between these components. This document will specifically go over the scope of the document, definitions, acronyms and abbreviations, references, representation, goals, construction, logical view (packages, modules, subsystems), interface description, and quality (how the architecture contributes to all of the capabilities). Software architecture plays an important role in software development by providing a high-level structural framework for a system, allowing for the system to maintain a standard of organization, scalability, and maintainability. It assists with the detailing and separation of concerns, fosters reusability, and ensures flexibility when choosing components. This document will outline how the project intends to optimize performance, address quality attributes, and aid in risk management.

1.2 Scope

This ***Software Architecture Document*** is in association with the overarching plan of the Arithmetic Parser project. It encapsulates the management process, objectives, development, iterations within the project, and detailed accounts of the design and architecture planned to be carried through the implementation phase. The plans defined in the document adhere to the *Requirements Document*, as mentioned on Canvas. The scope of this document will contain pertinent details on how the team will carry out the implementation phase of the project. Specifically, details on the architecture of the code and the way it is intended to be structured will be defined here. This document will influence and define the behavior of the program.

1.3 Definitions, Acronyms, and Abbreviations

Specific definitions, acronyms, and abbreviations for the Arithmetic Parser project are provided in the ***Glossary***.

1.4 References

For the ***Software Architecture Document***, the list of referenced artifacts includes

1. 001 Project Plan, SN^{2}ARK, 09/24/23
2. 002 Software Requirements Specification 10/07/23
3. Iteration Plans
4. Project Management Plan
5. Requirements Document
6. Design Document
7. Test Cases
8. 007 Glossary, SN^{2}ARK, 09/24/23

1.5 Overview

The ***Software Architecture Document*** presents different perspectives of the software architecture. It's a critical resource for software developers and users as it helps them understand and communicate the architectural decisions of the software project. This document contains:

- The Architectural Representation describes what software architecture is for the current system and how it is represented. For the Arithmetic Parser, SN^{2}ARK will employ three specific views of architectural representation, User Interface, Logic, and User Interface Logic.
- Architectural Goals and Constraint describes the software requirements and objectives that have some significant impact on the architecture. It also captures the special constraints that may apply. For the Arithmetic Parser, SN^{2}ARK has outlined detailed requirements and plans in the ***Software Requirements Specifications*** and ***Project Plan*** Documents
- Logical View describes the architecturally significant parts of the design model. For SN^{2}ARK, the User Interface Layer, the Logic Layer, and the User Interface Logic Layer.
- Interface Description describes major entity interfaces, including screen formats, valid inputs, and resulting outputs. For the Arithmetic Parser, SN^{2}ARK will include a legible command-line interface that will allow for user input and readability in input/output from the end-user perspective.
- Quality decisions describe how the software architecture contributes to all capabilities (other than functionality) of the system. For the Arithmetic Parser, SN^{2}ARK will ensure that all functionality and results are met through continuous access to the quality of the system by reviewing and testing.

The organization of the Software Architecture Document is carefully structured to provide clear goals for the development of the Project and its expected behavior and outcome for the final product. Specifically, the document will cover the major design components and the approach/style the team has decided to work with to model the design of the project. The constraints of the requirements and the scope of the Arithmetic Parser will be defined here. Decisions on quality and the interface design will also be shown and highlighted in the ***Software Architecture Document***.

2. Architectural Representation

These are the specific views of the architectural representation.

User Interface (command line) Components: Display Module. Input Module
Logic Components: Tokenizer Module, Parser Module
User Interface Logic Components: Evaluation Module, Error Handler Module

The User Interface for the Arithmetic Parser is the top layer. Its components consist of the display module,

where user input and calculation output are displayed, and the input module, where user input is processed through other modules. The Logic layer is then composed of the tokenizer module, and parser module that analyzes and ensures the expression is valid and determines the necessary operations. The final layer is the User Interface Logic which performs the calculations and handles any possible errors through the evaluation module and error handler module.

The software architecture for the Arithmetic Parser follows a layered architectural style. This approach is essential to the design of the Arithmetic Parser. A layered approach allows the overarching process to be divided into numerous and detailed smaller components. This will facilitate the reusability of our code, as well. Specific to SN^{2}ARK's project, each layer is represented by different classes based on hierarchy. The components include but are not limited to an input module, tokenizer module, parser module, evaluation module, error handler module, and a display module. The display module shows the input and output of calculations. The following is the list of modules in the hierarchy of the layered approach.

The input module will allow user input from the command line (keyboard input). This module will read in and store this input. In the future, if time permits, the input can be made in a more user-friendly GUI or interface. The tokenizer module will break down the user input given from the input module into tokens, ie, operators, operands, and parentheses that will then be analyzed and used by other modules (numbers, operators, operands, parentheses, etc.). The parser module will take the components of the tokenizer module, and determine the orders of operation for the user's expression. The Stacks Data Structure will be implemented here for parenthesis handling. Operator precedence will be adhered to here. Then, the evaluation module will take the expression and perform arithmetic calculations following the PEMDAS. If errors are identified during the parsing or evaluation process, the error handler module will intercept and handle the error and display an appropriate message in the terminal.

Specifically noting the connectors in this design view, the modules interact and connect through loose coupling. Each component and module described above has its functionality and can work separately, however, for the user to get their desired result for their expression, the components are required to work cohesively and be connected to the next part in our layered approach. This is why the components are connected in a loose coupling format. Specifically, the components will communicate by passing the required details from one module to the next. For example, the input will pass the expression to the tokenizer module for the inputs to be broken down into their separate operators.

3. Architectural Goals and Constraints

Detailed Software Requirements for SN^{2}ARK's Arithmetic Parser are outlined in the ***Software Requirements Specifications*** Document. The overarching architectural goals for the Arithmetic Parser are to provide an accurate, functional interface for evaluating arithmetic expressions following the rules of PEMDAS. This is done by utilizing a layered approach for the system design by grouping related functionality. It is intended that the Professor, TAs, and other students will be able to test the design and implementation of the project. The architectural constraints include being limited to C++ programming language within the Linux environment, and accurate functionality on the University of Kansas Cycle Servers. The program development based on the architecture model would be the resulting product which would be distributed via our GitHub Repository. Other constraints include team schedule which is limited to Wednesday and Friday, and our team structure which includes all team members performing their specific responsibilities as outlined in the ***Project Plan***. For the architecture implementation, each team member is expected to contribute to the development of each module. In addition, each team member may specialize in a module based on their title and skills. The final constraint consists of completing the project before 5 December 2023.

4. Use-Case View - N/A

4.1 Use-Case Realizations - N/A

5. Logical View

5.1 Overview

The software architecture of the arithmetic parser is layered according to the process of individual project components. Our design design model consists of three major layers:

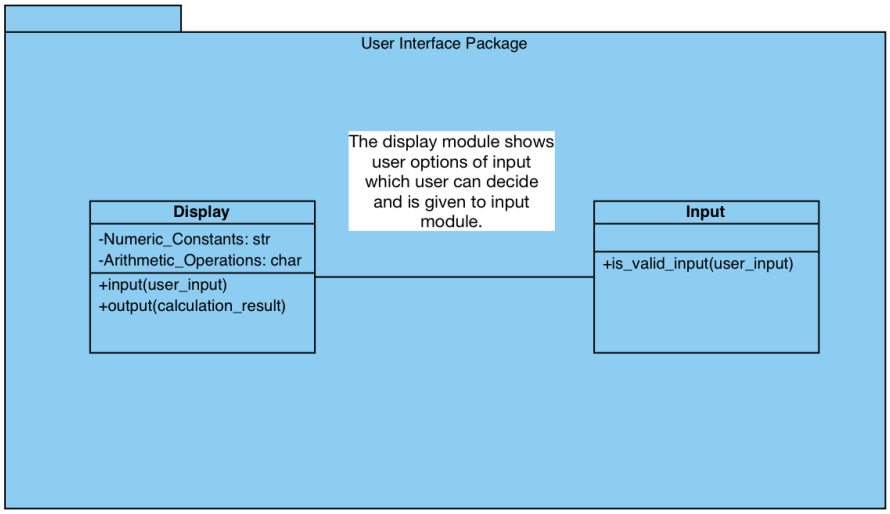
1. In our User Interface (Layer 1/Top), we have a display module and an input module. The display module is responsible for displaying or showing the inputs and outputs of the calculations. The input module is responsible for handling user input in the form of keyboard input.
2. In our Logic (Layer 2/Middle), we have a tokenizer module and the parser module. The tokenizer module would specifically tokenize the arithmetic expressions from the User Interface Layer into operands, operators, and parentheses. The parser module would be responsible for taking in and analyzing the tokenized components of the arithmetic equation into the hierarchy of the operation, following the order of operations of PEMDAS which includes evaluating the precedence of parentheses and sub-expressions.
3. In our User Interface Logic (Layer 3/Bottom), we have two components: the evaluation module and the error handling module. The evaluation module is responsible for the actual arithmetic calculation, and it will calculate the result of the expression following the PEMDAS rule. The error handling module is responsible for handling errors during parsing and evaluation stages, if necessary, and it will display error messages on the user interface when necessary.

5.2 Architecturally Significant Design Modules or Packages

(1) User Interface Package:

Package Description: the point of human-computer interaction and communication through arithmetic expressions on the Command Line.

Diagram:



1. **Module #1: Display Module**

Description: Show the input and output of calculations.

a. Responsibilities: shows inputs and outputs to the user
b. Operations: input(user_input); output(calculation_result) → the only two function that interacts with the user
c. Attributes:
    i. operators: String

2. **Module #2: Input Module**

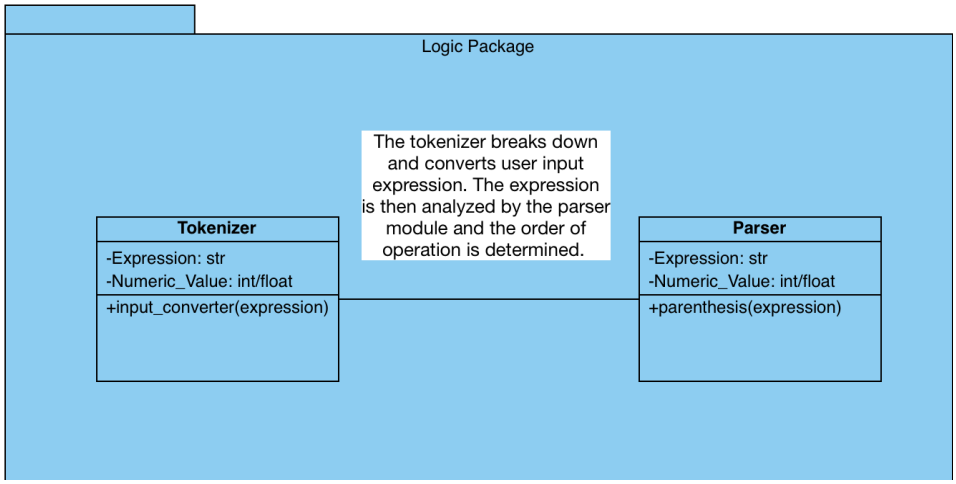
Description: Allow user input from the command line (keyboard input).

a. Responsibilities: Takes in user input
b. Operations: integer/float values; is_valid_input(user_input) → validates if the output is in accordance with an integer or float
c. Attributes: N/A

(2) Logic Package:

Package Description: Breaks down the user input by numbers, operators, and parentheses, and determines the order of operations.

Diagram:



1. **Module #1: Tokenizer Module**

Description: Breaks down user's input into tokens (numbers, operators, operands, parentheses, etc.)

d. Responsibilities: breaks down user input into tokens (numbers, operators, parentheses, etc.)
e. Operations: input_converter(expression) → converts and returns the user's string expression into valid integer/floating number values which can be evaluated later in the parser module
f. Attributes:
    i. expression: str
    ii. numeric values: Integer/Float

2. **Module #2: Parser Module**

Description: Determines the order of operations based on operator precedence and handles

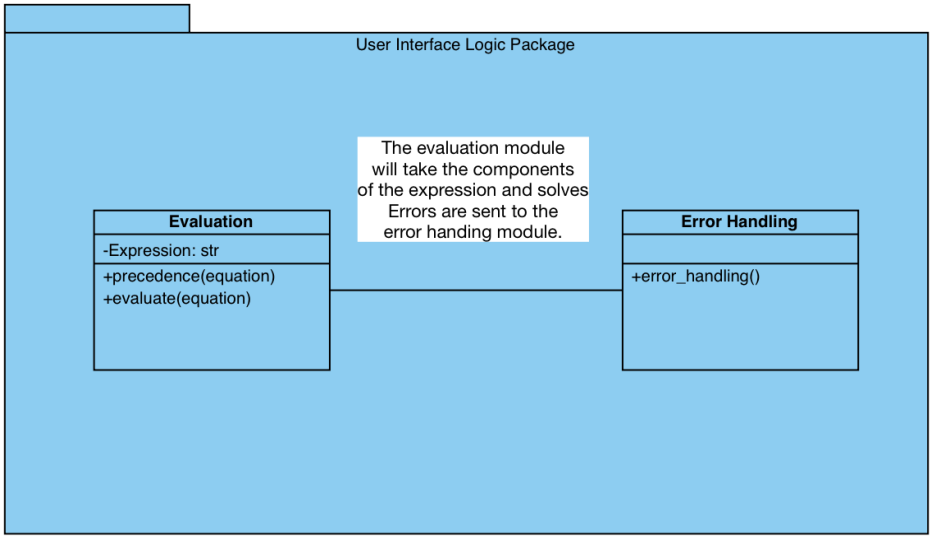
parentheses

g. Responsibilities: determines the order of operations based on operator precedence and handles parentheses
h. Operations: parenthesis(expression) → implemented utilizing stack data structure
i. Attributes:
    i. Expression: str
    ii. Numeric Values: int/float

(3) User Interface Logic Package:

Package Description: Evaluates the user input by inspecting and manipulating the arithmetic expression to follow PEMDAS formulae and outputs the calculated results.

Diagram:



1. **Module #1: Evaluation Module**

   Description: Calculates the result of the expression following the PEMDAS rule, or performs the actual arithmetic calculations.

   a. Responsibilities: calculates the result of the expression following the PEMDAS rule, or performs the actual arithmetic calculations
   b. Operations: precedence(equation); evaluate(equation) → this includes addition, subtraction, multiplication, modulo
   c. Attributes:
       i. Expression: str

2. **Module #2: Error Handling Module**

   Description: Handles errors during parsing and evaluation stages. This will also display error messages on the user interface when necessary.

   a. Responsibilities: handles errors during parsing and evaluation stages. This will also display error messages on the user interface when necessary
   b. Operations: error_handling() → incorporates try and except block methodology
   c. Attributes: N/A

6. Interface Description

The interface of the SN^{2}ARK's Arithmetic Parser includes a legible command-line interface that allows users to enter expressions and displays the calculated results. Valid user inputs consist of numeric constants, such as π, e, and integers, and arithmetic operators, including addition(+), subtraction(-), multiplication(*), division(/), modulo(%), and exponentiation(^). The resulting output is expected to follow operator precedence which is defined as the precedence of the operators according to the PEMDAS rules, while also implementing the logic to evaluate the expression while considering operator precedence and parentheses handling. For example, for the following input 8 - (5 - 2), the expected output is 5. This is because the parentheses ensure that the subtraction inside them is performed first, leading to the correct result. Additionally, for the input 4 * (3 + 2) % 7 - 1, the expected output is 5. This is because this expression combines multiple operators and parentheses to correctly calculate the result step by step, first solving within the parenthesis.

7. Size and Performance - N/A

8. Quality

The software architecture contributes to the reliability of the system as the Arithmetic Parser will adhere to the test cases and refer to the ***Test Case Document*** for the Test Cases. The quality will be ensured by the TA's and the Professor. There are no prominent safety, security, or privacy implications in this Project. The quality will be determined if it can handle the operations that are listed in the ***Software Requirements Document*** including parenthesis-handling and operator precedence. In the context of reliability, the project will work on all requirements. As for extensibility, the modules and the way we have decomposed the project will allow for future reference and usage of these modules as they will be more generalized for many cases of calculator operations. Finally, in terms of portability, the input will come from the command line so a laptop or device with similar interfaces will be needed.