Calculator Project

Software Architecture Document

Version <1.0>

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 11/12/2023 | 1.0 | Created Software Architecture Document |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 4

1.1 Purpose 4

1.2 Scope 4

1.3 Definitions, Acronyms, and Abbreviations 4

1.4 References 4

1.5 Overview 4

2. Architectural Representation 4

3. Architectural Goals and Constraints 5

4. Logical View 5

4.1 Overview 5

4.2 Architecturally Significant Design Packages 5

5. Interface Description 5

6. Size and Performance 5

7. Quality 5

Software Architecture Document

# Introduction

## Purpose

The purpose of this Software Architecture document is to give an insight into how the program is organized and how different parts of the program relate to each other. This is important because it may not be obvious how a program is supposed to work, and it provides users and developers information on how a program needs to be maintained or updated.

## Scope

This document can be used to help develop the first iteration of the software, maintain, or even update the software. It can be used by other professionals, students, and KU faculty.

## Definitions, Acronyms, and Abbreviations

System Architecture - The set of principle design decisions made about a system to be developed

Software Component - Encapsulates a subset of the system’s functionality and/or data.

Software Connector - A pointer or link between two data structures

Software Topology - Maps the applications and components from the solution onto the hardware topology.

Abstraction - The process of [generalizing](https://en.wikipedia.org/wiki/Generalization) [concrete](https://en.wikipedia.org/wiki/Abstract_and_concrete) details, such as [attributes](https://en.wikipedia.org/wiki/Attribute_(computing)), away from the study of [objects](https://en.wikipedia.org/wiki/Object_(computer_science)) and [systems](https://en.wikipedia.org/wiki/System) to focus attention on details of greater importance.

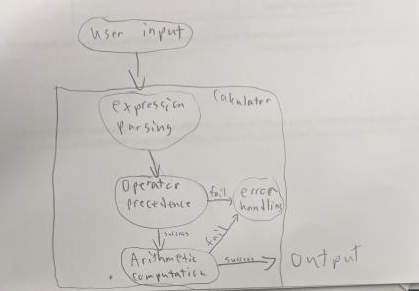
## References

N/A

## Overview

This document contains the Architectural Representation, Goals and Constraints, Use-Case and Logical Views, the description of the Interface, and the importance of Size and Quality.

# Architectural Representation



Above is a flow chart of our program. It will start off with user input asking the user either through a user interface, or command line interface for the equation. Afterwards it will pass that to the expression to the expression parsing method that will tokenize that function and then using the operator precedence function it will make sure it is handled in the correct order which at any point if it fails pass it to the error handling function. After Operator precedence it will compute the answer outputting the answer to the user or passing it back to error handling function if it fails.

# Architectural Goals and Constraints

Portability

* Designed to be platform-independent so that it can run easily on many operating systems and computers.

Design/Implementation Strategy

* Terminal based calculator, with a robust error-handling system.

Development Tools

* To be programmed in C++
* Stored/Collarborated on through Github

# Logical View

## Overview

The calculator software architecture is divided into several key modules or packages to handle different aspects of its functionality. These modules include Expression Parsing, Operator Precedence, Arithmetic Computation, Error Handling.

## Architecturally Significant Design Modules or Packages

Expression Parsing

* Responsible for breaking down the user input expression into manageable tokens and organizing them for further processing.

Operator Precedence

* Ensures that operations are performed in the correct order using PEMDAS.

Arithmetic Computation

* Executes arithmetic computations based on the expression.

Error Handling

* Detects and handles errors such as division by zero or incorrectly inputted equations.

# Interface Description

The inputs will either include a terminal prompt, text file, or user interface that contains the math operations that need to be parsed. The output will use a terminal or user interface and will provide the solution to the operation.

# Size and Performance

There are no size or performance concerns for this particular project because of the lack of complexity. However, proper coding and architectural conventions are still used.

# Quality

The software architecture is not only beneficial for the functionality of the program, but can also benefit the extensibility, reliability, portability, and size of a program. Quality architecture is key to a well-organized and professional program.