Doing Math With C++

Software Architecture Document

Version 1.0

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 07/11/23 | 1.0 | First draft | Manish Singh, William Hedges, Eric Loseke,AU |
| 09/11/23 | 2.0 | Second Draft | Manish Singh, William Hedges, Eric Loseke,AU |
| 11/11/23 | 3.0 | Final Draft | Manish Singh |
|  |  |  |  |

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 4

4. Logical View 5

4.1 Overview 5

4.2 Architecturally Significant Design Packages 5

5. Interface Description 5

6. Quality 5

Software Architecture Document

# Introduction

This document sheds light on the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the Software Architecture. It describes the overall purpose of the software architecture, its scope, necessary definitions, and abbreviations to convey the overlying concept with the architecture.

## Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions that have been made on the system. This document will come handy to anyone who wants to know about our software architecture, its components and assembly of those components.

## Scope

This document has its scope limited to decisions taken to design and the overall final structure of the software architecture for our Calculator project. All the explanations and the terms defined within this document has been designed keeping our Calculator software at the center.

## Definitions, Acronyms, and Abbreviations

Software Architecture: The set of principal design decisions made about a system to be developed.

Object-oriented Design: Software architectural design where core functionality is stored in separate modules and each part has one specific responsibility.

Stack: Stack is a linear data structure that follows a particular order in which the operations are performed.

## References

Saiedian, Hossein. (2023, Oct. 31). *Software architecture and design concepts*. Department of Electrical Engineering and Computer Science, University of Kansas. <https://canvas.ku.edu/courses/104607/files/folder/Lecture%20notes?preview=8204647>

## Overview

The document has five sub-sections namely architectural representation, Architectural goal and constraints, Logical view, interface description, and quality.

**2. Architectural representation:**

This section focuses on explaining the software architecture for the current system. It details the object-oriented classes used to break down expressions and calculate the desired results.

**3. Architectural goals and constraints:**

This section discusses the modules used in the system and the reasons behind their selection. It provides insights into the design decisions and constraints that influenced the architecture.

**4. Logical view:**

This section delves into the classes and packages used for this product. it talks about their usage and functionality.It is further divided into 2 subheadings.

**4.1**- Overview:

It describes the overall surface interaction of each class with one other. Details about cohesion and coupling can be found in this section..

**4.2**-Architecturally significant design modules or packages:

A more detailed overview of classes and packages that play a huge role in the product.

**5. Interface Descriptions:**

This section describes the interface the user will be working with while using our software.

**6. Quality**

This section is a brief summary of the safety precautions and reliability of the components used to design the product.

# Architectural Representation

For our product, we will be using an object-oriented design. Our product will have primarily two sections: control section and a Calculator class. The controller section perform the inputs and outputs for the program. It makes a call to the Calculator module containing the Calculator class to do all the functional task of the program. Calculator class where we will have the core calculator functionality. The controller section also uses the Calculator class functions and attributes to turn the user given expression into a single numeric value.

Here, the logical view and the developmental view are necessary.. For the logical view, it would contain the elements of the two models: the control section and the Calculator class; one could look at a UML diagram that would have been defined in the requirements. For the developmental view, we would look at the system from the programmer’s perspective and we see the program in terms of the packages models and in this case the only module/package that we are importing is the Calculator module.

# Architectural Goals and Constraints

In the design of the architecture for our C++ program, the primary objective is to establish an efficient mechanism for evaluating mathematical expressions. To achieve this, the control section, in our program, main file, invokes a dedicated Calculator class, which, in turn, utilizes a stack module with functions to delve into the expression computation. The Calculator class uses the built-in C++ Stack.h module. The stacks' Last-in-first-out nature aligns well with the hierarchical structure of mathematical expressions, making it an ideal tool for tracking operators and operands as we navigate through the calculations. The stack empowers the calculator class to handle complex expressions with nested operators and operands in a manner that aligns with the specific requirements of our application. A stack is used to store operands, while another stack is used to store operators and parentheses. The program manipulates these stacks based on the encountered characters in the input expression to effectively evaluate the mathematical expression following the rules of operator precedence and parentheses.By leveraging two distinct stacks—one for operands and another for operators and parentheses—the calculator class adeptly manages the complexities posed by nested expressions.

The architecture supports the resusability of the product as the functions defined in Calculator class can be used to make another calculator applications with higher functionality or any other product that utilizes the basic calculation of mathematical expressions. The two-section architecture distributes the labor between the sections so that no single section is overloaded with the tasks.

In summary, the architecture revolves around the collaboration between the control section i.e. main file and a calculator class equipped with a stack module used for expression computation. This design choice enhances modularity, and promotes reusability, ultimately enabling the calculator class to efficiently handle expressions with a deep level of nesting*.*

# Logical View

## Overview

We are implementing an object-oriented design. We will import the Calculator module and use that functionality in a main file to put the pieces together where we will call smaller subroutines within our main file to utilize the in-built Calculator methods. These methods will include PerformSimpleOperation() and Evaluate() which can be used in a main file to evaluate a mathematical expression and interact with the Stack. The Attributes of this function are +current\_value which tracks the current value of the operation.

The Calculator module brings in the built-in C++ Stack class and allows us to use the Stack data structure in order to push numbers and operators onto a Stack to arrange in order of importance from the mathematical order of operations. Therefore, as far as package hierarchy goes, the Calculator class is the top of the hierarchy and then next the Calculator class brings in the built-in C++ library Stack Class.



## Architecturally Significant Design Modules or Packages

**4.2.1** Calculator Module: This will be the built-in C++ module containing the Calculator class.

Calculator Class: This is our core calculator functionality. The methods that go with this calculator class are PerformSimpleOperation() and Evaluate() which can be used in a main file to evaluate a mathematical expression and interact with the Stack.

Stack Class: This is a generic Stack Class that builds the stack data structure from the C++ library containing push/pop methods.



# Interface Description

The user will be prompted by text in the command line asking them to enter a mathematical expression. Valid inputs include any mathematical expression that can parse and evaluate arithmetic expressions containing operators +, -, \*, /, %, and ^ as well as numeric constants (integers and floating-point numbers).

Operators and operands must be correctly matched.

The expression must adhere to mathematical rules (e.g., no division by zero).

The user will then enter their mathematical expression and hit enter. Then, the program will produce the numeric value of that expression and print that value to the screen.

# Quality

The architecture accounts for operator precedence and parentheses, ensuring accurate expression evaluation. By separating functionality into the calculator class, it allows for the systematic handling of complex expressions, maintaining fidelity to the rules of mathematical operations. It also increases portability of this module because we could in theory apply the Calculator classes functionality to another project by simply importing the Calculator module.

By utilizing well-established stack data structures in that Calculator Class, potential errors related to expression parsing and calculation are minimized, contributing to the overall reliability of the system