**Rock Chalk Instruments**

**C++ Calculator**

**Software Architecture Plan**

**Version <1.1>**

# **Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 09/Nov/23 | 1.0 | *Completed Introduction*, *Architectural Representation*, *Architectural Goals and Constraints*, *Interface Description*, and *Quality.*  Completed most of *Logical View.* | Owen Deines, Connor Schroeder, Jordan Mcdaniels, Carson Abbott, Brisa Andrade |
| 12/Nov/23 | 1.1 | Polished up and finished *Logical View.*  Fixed formatting and polished up document. | Brisa Andrade  Carson Abbott |
|  |  |  |  |
|  |  |  |  |

# **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 5

3. Architectural Goals and Constraints 6

4. Logical View 7

4.1 Overview 7

4.2 Architecturally Significant Design Packages 9

5. Interface Description 11

6. Quality 12

# **Software Architecture Document**

# **1.** **Introduction**

## **1.1** **Purpose**

The purpose of this document is to outline the Software Architecture of the Calculator Project. This document will establish our architectural views and approach to the project that will be guiding us through the development process. The outline will also describe the significant architecture decisions we made and why we made them.

## **1.2** **Scope**

The implementation process for the C++ Calculator will be heavily influenced by this document. This Software Architecture Document lays out how the program is structured and plans out how functional requirements will be implemented.

## **1.3** **Definitions, Acronyms, and Abbreviations**

Postfix - A method of evaluating mathematical expressions computationally.

## **1.4** **References**

No references.

## **1.5** **Overview**

The scope of the Software Architecture Document includes the Architectural Representation, the Architecture Goals and Constraints, Logical View, Design Modules, Interface Description, and Quality assurance of the project.

# **2.** **Architectural Representation**

The current system will be designed under an object-oriented design. The main program will repeatedly ask the user to enter an expression. Once the user does, the main program will create an object of the Calc class that will, upon construction, call a parenthesis parsing method that splits the expression into chunks, calls all the operator methods for completed chunks, and repeats the last step for every chunk whose inner chunk(s) were evaluated until a final result is set as the object's property for the main program to retrieve and print.

# **3.** **Architectural Goals and Constraints**

In terms of how the program will be distributed and ran, the program code will be written into a .c file that is then converted to a Docker image. This image will be publicly available on the Docker Hub, and running it will entail downloading the image into a directory and instantiating a container of the image. As far as the development of the program goes, the .c file will be uploaded to a team GitHub where everyone can make contributions in their own time and frequently check up with other team members for important deadlines and goals.

# 

# **4.** **Logical View**

## **4.1** **Overview**

The program will be made up using the below listed functions and use the Stack Class, defined in the library <stack> to implement the Calculator class:

* Calculator class:
  + Constructor
  + Function Precedence
  + Function convertHelper
  + Function precendenceHelper
  + Function add
  + Function Subtract
  + Function Divide
  + Function Multiple
  + Function Modulus
  + Function Calculate
  + Maybe Function for interface

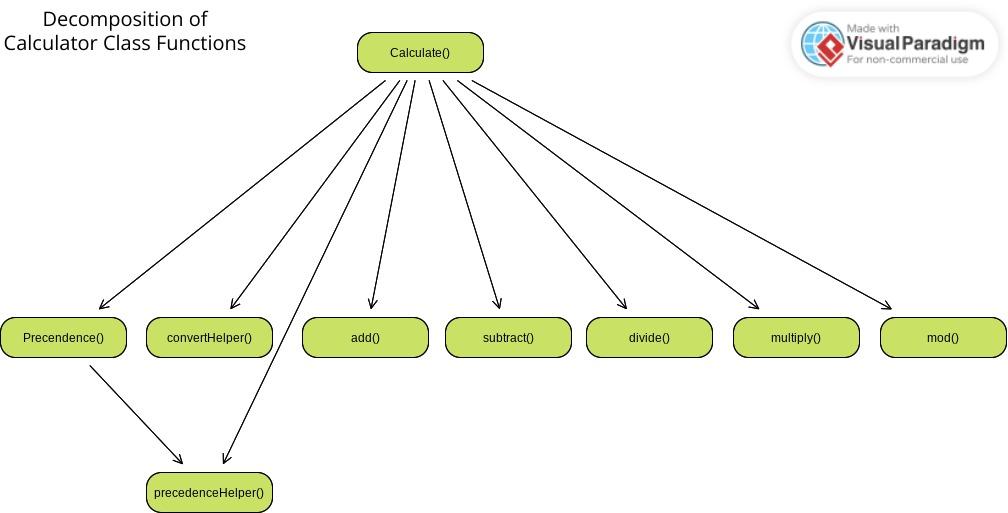
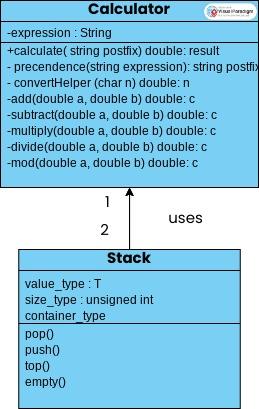
The decomposition of the design model is shown in the following diagram: 

Figure 4.1.1: decomposition of calculator class functions.

Multiple helper functions are used to simplify the readability of the code, allow for high cohesion, and to allow for better reusability both in the current iterations of the calculator and potential (function overloads) edits in the future.

Since only two classes are being used in this program, a class diagram felt like an appropriate approach to describe the relationship between them. The Calculator class will depend on the template Stack class available through the <stack> library in order to define the precedence between operators, set up the postfix expression and to evaluate the result of the user submitted expression.

Figure 4.1.2: Class Diagram

## **4.2** **Architecturally Significant Design Modules or Packages**

[For each significant package, include a subsection with its name, its brief description, and a diagram with all significant classes and packages contained within the package.

For each significant class in the package, include its name, brief description, and, optionally, a description of some of its major responsibilities, operations, and attributes.]

Stack class: A template class supplied by the <stack> library in C++, it’s used to implement the stack data structure for various datatypes and objects. It will be used to define the order of operations in an expression and to calculate the result.

Calculator class: This will be the overall class for our calculator and will hold all of the functions/methods used for the operators.

Reference figure 4.1.2 for class diagram involving all classes to be used for the calculator.

* Constructor: This will get the information for the stack which includes all public, private, and the input from the User. The constructor is in charge of setting up the Calculator object.
* Function Precedence: This will sort the stack using left and right parenthesis. (checking for a left then a right parenthesis) and define the order in which the given expression will be solved. It will then put the stack into a string. The expected output is a string with no parenthesis and the expression in a postfix order.
* Function convertHelper: Convert the string into doubles from the precedence function. These doubles will then be put into a new stack with the correct order so that we can call the operator functions to solve the input.
* Function precedenceHelper: This function will set and compare the precedence between two operators to make sure that they are in proper order.
* Function add: This will include the + operator and will add the two inputs together from the stack and insert them back into the stack at the top.
* Function Subtract: This will include the - operator and will subtract the two inputs together from the stack and insert them back into the stack at the top.
* Function Divide: This will include the / operator and will divide the two inputs together from the stack and insert them back into the stack at the top.
* Function Multiple: This will include the \* operator and will multiply the two inputs together from the stack and insert them back into the stack at the top.
* Function Modulus: This will include the % operator and will add the two inputs together from the stack and insert them back into the stack at the top.
* Function Calculate: This function will be in charge of evaluating the postfix expression that is outputted from the precedence function. The expected output is the result of the user inputted expression.
* Maybe Function for interface: If we have time, we will implement an interface using HTML (The structure), CSS (Color and fonts), and JS (buttons/Evaluations). This will be an online webpage that will accept the user input and then return the output onto the page in a nice and styled format.

# **5.** **Interface Description**

The only interface that the user interacts with is the terminal. Once the software runs, the user is prompted with text saying, “Enter an expression:”. The valid inputs will be 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, \*, /, (, ). After this expression is inputted, the integer/float answer will be output to the terminal screen.

# 

# **6.** **Quality**

Outside of functional requirements, this software architecture contributes to other capabilities as well. One such capability is usability since the interface is very straightforward and mimics that of a hand-held calculator. This software is also reliable because the mathematical expressions that it solves hardly require any computing power and takes little time. This software also proves to be extensible because there is room for new functions to be added such as factorials, sin, cosine, tangent, etc....