<Company Name>

Calculator project
Software Architecture Document

Version <1.0>

[Note: The following template is provided for use with the Unified Process for EDUcation. Text enclosed in square brackets and displayed in blue italics (style=InfoBlue) is included to provide guidance to the author and should be deleted before publishing the document. A paragraph entered following this style will automatically be set to normal (style=Body Text).]

[To customize automatic fields in Microsoft Word (which display a gray background when selected), select File>Properties and replace the Title, Subject and Company fields with the appropriate information for this document. After closing the dialog, automatic fields may be updated throughout the document by selecting Edit>Select All (or Ctrl-A) and pressing F9, or simply click on the field and press F9. This must be done separately for Headers and Footers. Alt-F9 will toggle between displaying the field names and the field contents. See Word help for more information on working with fields.] Marked (shaded) areas: items that are OK to leave out.

Revision History

Date	Version	Description	Author
11/9/23	<1.0>	Created document	Sean Brady, Chris Harvey, Naran Bat, Peter Walsdorf, William Morris, Kristin Boeckmann

Table of Contents

Software Architecture Document

1. Introduction

*[The introduction of the **Software Architecture Document** provides an overview of the entire **Software Architecture Document**. It includes the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the **Software Architecture Document**.]*

The purpose of this document is to capture the architecture design of our project. It will cover the use cases, logical view, and interface of our program.

1.1 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 which have been made on the system.]

*[This section defines the role or purpose of the **Software Architecture Document**, in the overall project documentation, and briefly describes the structure of the document. The specific audiences for the document are identified, with an indication of how they are expected to use the document.]*

This document's role is to outline an architecture that keeps the project efficient and easy to use. It is a blueprint for the software construction and evolution. The audience is the people building the system, and they will use the document to look over the different pieces of the software that are supposed to be built, the order they are to go about things, and how they will attack building software. Another audience is a supervisor who uses the document to see how different pieces of the system are supposed to be built and in what way software is supposed to be made.

1.2 Scope

[A brief description of what the Software Architecture Document applies to; what is affected or influenced by this document.]

This document details how the program is implemented and helps the programmer plan function cohesion and coupling.

1.3 Definitions, Acronyms, and Abbreviations

*[This subsection provides the definitions of all terms, acronyms, and abbreviations required to properly interpret the **Software Architecture Document**. This information may be provided by reference to the project's Glossary.]*

N/A

1.4 References

*[This subsection provides a complete list of all documents referenced elsewhere in the **Software Architecture Document**. Identify each document by title, report number (if applicable), date, and publishing organization. Specify the sources from which the references can be obtained. This information may be provided by reference to an appendix or to another document.]*

N/A

1.5 Overview

*[This subsection describes what the rest of the **Software Architecture Document** contains and explains how the **Software Architecture Document** is organized.]*

The document contains detailed descriptions about the Calculator Project's architecture. The document is organized in such a format that we begin by describing our goals, an overview of the architecture and how we decomposed the design model. Next we describe the specific design modules and packages used in our project, followed by a description of our user interface. Lastly, we describe how the software architecture is of fundamental importance to all parts of our application.

2. Architectural Representation

[This section describes what software architecture is for the current system, and how it is represented. It enumerates the views that are necessary, and for each view, explains what types of model elements it contains.]

User Interface (UI) View:

- Model Elements
    - Display for showing input and results
    - Button for control

Calculator Logic View:

- Model Elements:
    - Arithmetic functions (addition, subtraction, etc. More on it below).
    - Variables to store operandi and results.
    - Functions for performing calculations.

Controller View:

- Model Elements:
    - Event handlers to manage user input
    - Logic to coordinate communication between UI and Calculator Logic
    - Error handling mechanisms

3. Architectural Goals and Constraints

[This section describes the software requirements and objectives that have some significant impact on the architecture; for example, safety, security, privacy, use of an off-the-shelf product, portability, distribution, and reuse. It also captures the special constraints that may apply: design and implementation strategy, development tools, team structure, schedule, legacy code, and so on.]

This program will be run directly by the user and has no access to the user's system or any server so there aren't any security issues to consider for this product. We will not be using any off-the-shelf product or code. It will be distributed via an executable file that the user will download and run with a command line interface. We will develop this program using a modular coding paradigm with each function being a separate file which all runs together from the main cpp run file. We will use github to work together on the project and each file should be able to be changed without affecting the performance of the other files.

4. Use-Case View

[This section lists use cases or scenarios from the use-case model if they represent some significant, central functionality of the final system, or if they have a large architectural coverage—they exercise many architectural elements or if they stress or illustrate a specific, delicate point of the architecture.]

4.1 Use-Case Realizations

[This section illustrates how the software actually works by giving a few selected use-case (or scenario) realizations, and explains how the various design model elements contribute to their functionality. If a Use-Case Realization Document is available, refer to it in this section.]

5. Logical View

[This section describes the architecturally significant parts of the design model, such as its decomposition

into subsystems and packages. And for each significant package, its decomposition into classes and class utilities. You should introduce architecturally significant classes and describe their responsibilities, as well as a few very important relationships, operations, and attributes.]

The following section will provide a detailed breakdown of the modules, with their services and features, that will make up the final program.

5.1 Overview

[This subsection describes the overall decomposition of the design model in terms of its package hierarchy and layers.]

Decomposition of this application was done in two steps. First, all of the mathematical operations required by the program were separated out into modules that can be tested and used by other modules. Second, all of the utility functions required by the program to read, parse, perform, and return the results of the input were identified and added to the appropriate modules. This breakdown of functionalities will be useful when implementing our software product as all important operations will be able to be extensively tested to ensure correct results.

5.2 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.]

validInput:

- service: checks if a given input contains valid characters
- features: isValid(input), isParenthesis(char), isDigit(char), isOperator(char)

parser:

- service: determines the order of operations and handles parenthesis
- features: parse_input(tokens)

tokenizer:

- service: parses user input to create individual tokens for each part of the equation
- features: tokenize(input)

add:

- service: adds two numbers *a* and *b*
- features: add(a,b)

subtract:

- service: subtracts a number *b* from a number *a*
- features: sub(a,b)

multiply:

- service: multiply a number *a* by a number *b*
- features: mult(a,b)

divide:

- service: divide a number *a* by a number *b*
- features: div(a,b)

power:

- service: takes a number *a* to the power of a number *b*
- features: pwr(a,b)

modulus:

- service: given a number *a* and a number *b*, modulus gives the remainder of dividing a by b
- features: mod(a,b)

evaluate:

- service: using the other packages, the evaluate package will evaluate the expression input by the user. It will check that the input is valid, tokenize it, parse it, and solve it using the operator packages mentioned above and a queue
- features: eval(input)

get_input:

- service: gets input from the user
- features: get_input()

6. Interface Description

Main window:
- input box for user to input equations
- button to run script to check input for errors and evaluate expression
  - the user pressing the 'enter' key should also log this event
- window to log outputs
  - outputs are either an error message indicating invalid formatting/characters in the equation or the result of the expression
- button to close window
- button to minimize window

[A description of the major entity interfaces, including screen formats, valid inputs, and resulting outputs. If a User-Interface Prototype Document is available, refer to it in this section]

7. Size and Performance

[A description of the major dimensioning characteristics of the software that impact the architecture, as well as the target performance constraints.]

8. Quality

[A description of how the software architecture contributes to all capabilities (other than functionality) of the system: extensibility, reliability, portability, and so on. If these characteristics have special significance, such as safety, security or privacy implications, they must be clearly delineated.]

Maintainability:

The clear division of responsibilities among classes simplifies maintenance. Modification or updates in one component are less likely to affect others, promoting ease of debugging and enhancement

Scalability:

While a simple calculator may not demand extensive scalability, the modular architecture lays the groundwork for scalability. If the application were to evolve into a more complex mathematical tool, the existing structure would support such growth.

Usability:

The logical and UI separation enhances usability. The UI package can be redesigned without affecting the underlying logic, allowing for improvements in user experience and interface design without compromising the calculator's functionality.

Extensibility:

The modular structure allows for addition of new features or mathematical functions. Additional operations can be integrated without significantly impacting existing code.

Reliability:

Error handling mechanisms enhance reliability by capturing and managing errors gracefully. This prevents unexpected crashes and provides a better user experience.