Arithmetic Expression Evaluator

Software Requirements Specifications

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.]

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 10/05/2023 | 1.0 | Base iteration of document | Jake Bernard |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

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. Overall Description 5

2.1 Product perspective 5

2.1.1 System Interfaces 5

2.1.2 User Interfaces 5

2.1.3 Hardware Interfaces 5

2.1.4 Software Interfaces 5

2.1.5 Communication Interfaces 5

2.1.6 Memory Constraints 5

2.1.7 Operations 5

2.2 Product functions 5

2.3 User characteristics 5

2.4 Constraints 5

2.5 Assumptions and dependencies 5

2.6 Requirements subsets 5

3. Specific Requirements 5

3.1 Functionality 5

3.1.1 <Functional Requirement One> 6

3.2 Use-Case Specifications 6

3.3 Supplementary Requirements 6

4. Classification of Functional Requirements 6

5. Appendices 6

**Software Requirements Specifications**

# Introduction

[The introduction of the **Software Requirements Specification (SRS)** provides an overview of the entire **SRS**. It includes the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the **SRS**.]

[Note: The **SRS** captures the complete software requirements for the system, or a portion of the system. Following is a typical **SRS** outline for a project **using use-case modeling**. This artifact consists of a package containing use cases of the use-case model and applicable Supplementary Specifications and other supporting information.]

[Many different arrangements of an **SRS** are possible. Refer to [IEEE830-1998] for further elaboration of these explanations, as well as other options for **SRS** organization.]

## Purpose

[Specify the purpose of this **SRS**. The **SRS** fully describes the external behavior of the application or subsystem identified. It also describes nonfunctional requirements, design constraints, and other factors necessary to provide a complete and comprehensive description of the requirements for the software.]

The purpose of the *Software Requirements Specifications* is to define and describe the necessary and desired functionality of the *A2E* software. This is includes all units of desired behavior (or *features*), all non-functional requirements (or *qualities*), and any constraints in the design or runtime behavior or environement of the application.

## Scope

[A brief description of the software application that the **SRS** applies to, the feature or other subsystem grouping, what Use-Case model(s) it is associated with, and anything else that is affected or influenced by this document.]

The *Software Requirements Specification* applies to the full *A2E* application, most importantly concerned with the interface the application presents to the end user and the functionality available to the end user provided by that interface.

## Definitions, Acronyms, and Abbreviations

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

**Feature**: a unit (or several units grouped tgoether) of functionality.

**Quality**: a description of how the system should operate that is not related to functionality.

**Constraint**: a limitation placed upon the design, implementation, or deployment process

**GUI:** Graphical user interface

For all else, see the *Project Glossary*.

## References

[This subsection provides a complete list of all documents referenced elsewhere in the **SRS**. 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.]

These documents are referenced by the *Software Requirements Specifications*:

* *Project Vision*, artifacts/project\_vision.pdf, Group Without a Name, 2023
* *Project Description*, artifacts/project\_description.pdf, University of Kansas, 2023
* *Project Plan*, artifacts/project\_plan.pdf, University of Kansas, 2023
* *Google C++ Style Guide*, https://google.github.io/styleguide/cppguide.html, Alphabet Inc., 2023

## Overview

[This subsection describes what the rest of the **SRS** contains and explains how the document is organized.]

The outline of the rest of the *Software Requirements Specifications* is as follows:

Overall Description — Contains general information about the application and gives context for the requirements, qualities, and constraints of the application.

Specific Requirements — Defines the requirements of the program with enough specification to enable a satisfactory implementation of the application and for which tests can be developed to determine if the requirements were satisfied.

Classification of

Functional Requirements — A list of all of the functional requirements, sorted and categorized as "Necessary", "Desirable", or "Optional".

Appendices — Contains all extra information referenced within the document that did not fit within the other categories.

# Overall Description

[This section of the **SRS** describes the general factors that affect the product and its requirements. This section does not state specific requirements. Instead, it provides a background for those requirements, which are defined in detail in Section 3, and makes them easier to understand. Include such items as:

## Product perspective

### System Interfaces

N/A.

### User Interfaces

The user is presented either with a command line interface or GUI depending upon the launch mode of the program. The command line interface affords the user with text-based input from the user's keyboard for the creation of user-defined arithmetical expressions. The desktop GUI application will afford the user with both input from the keyboard and from buttons for operations and numerals within the GUI. The user should be able to write, clear, and evaluate expressions in both scenarios. Errors in input should be handled with an appropriate alert to the user that the specified expression is not valid.

### Hardware Interfaces

The application should be a compiled program written in C++, so all interfacing with the hardware the application runs on should be handled during compilation.

### Software Interfaces

N/A

### Communication Interfaces

N/A

### Memory Constraints

The application should be able to run on any reasonable consumer-grade general purpose computer released within the last 10 years.

### Operations

**LOOK AT LATER**

The application should allow input of arithmetic expressions and evaluation of arithmetic functions.

## Product functions

The application is expected to act as a calculator and expression evaluator for arbitrary arithmetic expressions containing any order of a operations from a pre-defined set.

## User characteristics

The user is expected to have a basic knowledge of the meaning of various basic mathematical operators related to arithmetic, to be able to enter input via command line, to have familiarity with common GUI elements, and have some need for calculating arithmetical expressions.

## Constraints

The application is expected to be written in C++ and be able to run on the computers in the University of Kansas.

## Assumptions and dependencies

The application will depend upon the GCC compiler to compile the C++ code to a machine executable file and will be reliant on outside libraries to provide certain functionality for the GUI. It is implicitly assumed that the PEMDAS order of operations will be used.

## Requirements subsets

**REVIEW LATER**

Functional requirements may be sub-categorized as those which deal with the runtime of the program and those which deal with the interface of the program, though there is some overlap between the two.

# Specific Requirements

[This section of the **SRS** contains all software requirements to a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. When using use-case modeling, these requirements are captured in the Use Cases and the applicable supplementary specifications. If use-case modeling is not used, the outline for supplementary specifications may be inserted directly into this section, as shown below.]

## Functionality

[This section describes the functional requirements of the system for those requirements that are expressed in the natural language style. For many applications, this may constitute the bulk of the **SRS** package and thought should be given to the organization of this section. This section is typically organized by feature, but alternative organization methods may also be appropriate; for example, organization by user or organization by subsystem. Functional requirements may include feature sets, capabilities, and security.

Where application development tools, such as requirements tools, modeling tools, and the like, are employed to capture the functionality, this section of the document would refer to the availability of that data, indicating the location and name of the tool used to capture the data.]

### Parse and evaluate user input

The application is expected to be able to take user input, either through the GUI via clickable buttons, or through command line and/or GUI as keyboard input which is then parsed as an arithmetic expression, evaluated, and an answer returned.

### Basic operations

The application should be able to support the operations of addition (+), subtraction (-), multiplication (\*), division (/), exponentiation (^ or \*\*), and modulo (%).

### Order of operations

Mathematical operators should follow the PEMDAS rules when an expression is being evaluated. The modulo operator should have priority below division but above addition.

### Parenthetical Grouping

The application should be able to evaluate nested expressions enclosed in parentheses, with the priority of expressions becoming higher with higher parenthetical grouping depth. Empty parentheses should return a syntax error.

### Usage of Numeric Constants

The application should be able to recognize and utilize expressions involving common numeric constants, eg pi and Euler's number.

### Usage of floating point numbers

The application should accept both integers and floating point numbers as input.

### Error Handling

Syntax and arithmetic errors hould be handled within the program gracefully and the user should be notified of the error as the application continues functioning.

### Command-Line Interface

A command-line interface should be available to the user for the input of expressions and for displaying the results of evaluated expressions

### Advanced operations

The application should have support for more advanced mathematical functions/operators, eg trigonometric functions, logarithms, roots, etc.

### Multiple Numerical Modes

The application should be able to display numbers in multiple ways, eg as decimal numbers with a radix point, or as whole numbers, possibly multiplied by irrational constants, with fractional parts.

### Arbitrary Precision Arithmetic

The application should be able to handle numbers of arbitary size (within the system's memory capacity).

### Graphical User Interface (GUI)

The application should be able to run as a windowed desktop application with an easy-to-use graphical interface with a display similar to a calculator, with buttons that can be clicked on to add numbers and operators to the expression, to clear or evaluate expressions, or to change the mode of display for numbers.

### Variable Storage

The application should be expanded to add support for user-defined variables which can hold previously calculated values and may be used in future calculations during the program's runtime.

### Graphing Capabilities

The application should be able to graph 2D functions by using a special, reserved variable, which will act as the input variable to the resulting function produced by the user.

### Multiple GUI Styles

The user should be able to switch between several different styles of GUI from within the program.

### Calculation History

The user should be able to see a history of calculations performed since the program started running within a reasonable limit.

### Easter Eggs

Certain numbers (eg developer birthdays, holidays, numbers of symbolic significance) should result in unique displays or effects when they are the result of an expression.

## Use-Case Specifications

[In use-case modeling, the use cases often define the majority of the functional requirements of the system, along with some non-functional requirements.]

**[REMOVE THIS?]**

## Supplementary Requirements

[Supplementary Specifications capture other requirements, e.g., non-functional requirements and development constraints, that are not included in the use cases and non-functional requirements.]

### Made in C++

The application should be developed primarily using the C++ programming language.

### Object-Oriented Programming

The application should be developed using object-oriented programming principles.

### Follows Style Guidelines

The source code for the program should follow the Google C++ Style Guide.

### Comments and Documentation

The application should include comments in the source code and documentation that explains the functionality.

### Informative Error Messages

The error messages output by the program at runtime should be informative enough that the user understands where in the expression the error occured.

### Easy to Maintain, Scalable Codebase

The codebase of the application should be organized in such a manner that modifications and expansions to the program require minimal effort.

### Testable

The application must have a suite of test cases designed for it which can thoroughly expose any defects present within that iteration.

### Accessible

The end-user, and especially graders, should be able to seamlessly access and run the program without any configuration or extra steps.

### Portable

The final application should have builds able to run on different operating systems. Any external libraries included should emphasize portability.

### Enjoyable and Intuitive User Experience

The application should provide a high quality, intuitive user experience.

### Performant

The application should perform all operations reasonably quick and efficiently. The amount of memory taken up by the application during runtime should be minimal.

# Classification of Functional Requirements

[List, usually in a table, all functional requirements and order them by Type (Essential, Desirable, and Optional) or by order of appearance in the document.]

|  |  |
| --- | --- |
| **Functionality** | **Type** |
| Parse and evaluate user input | Essential |
| Basic operations | Essential |
| Order of operations | Essential |
| Parenthetical Grouping | Essential |
| Usage of Numeric Constants | Essential |
| Usage of floating point numbers | Essential |
| Error Handling | Essential |
| Command-Line Interface | Essential |
| Advanced operations | Desirable |
| Multiple Numerical Modes | Desirable |
| Arbitrary Precision Arithmetic | Desirable |
| Graphical User Interface (GUI) | Desirable |
| Variable Storage | Optional |
| Graphing Capabilities | Optional |
| Multiple GUI Styles | Optional |
| Calculation History | Optional |
| Easter Eggs | Optional |

# Appendices

[When appendices are included, the **SRS** should explicitly state whether or not the appendices are to be considered part of the requirements]

**[N/A or *See: Project Vision*]**