<Project Name>

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** |
| <26/09/23> | <1.0.0> | First additions | Xavier Ruyle |
| <01/10/23> | <1.0.0> | Partially completed sections 2 and 3 | Henry Hoopes |
|  |  |  |  |
|  |  |  |  |

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 this SRS is to explain the behavior of the calculator. It describes what the calculator needs as well as the intended users that it will be directed toward.

## 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 SRS applies to the Calculator, which is made to parse, and calculate mathematical expressions. It provides the user with the ability to calculate basic arithmetic with operators using a user interface in the command line. The calculator is designed for users who want a simple and minimalistic calculator through a command line 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.]

Calculator: The program itself

UI: User interface

IfN: Infix Notation

PN: Postfix Notation

EXPR: Expression

## 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.]

N/A – no references at this moment

## Overview

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

The rest of the SRS will contain the overall description which will detail the general factors which affect the calculator software and its requirements. It will then go into detail about the specific requirements that are needed to create the calculator. It will then classify the functional requirements in a table format and categorize them by type. Lastly, the appendices are available for further reading.

# 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 → I think this is redundant/same as software interfaces

This project won’t interface with another system, but will use C++ libraries to utilize stack and queue data structures

### User Interfaces

The user will interact with our software through the command line. Our program will receive typed input f rom the user and will print output to the command line.

### Hardware Interfaces

There won’t be any additional hardware interfaces. The software will receive input and send output using the standard input/output library in C++.

### Software Interfaces

*I think this is where we define what libraries we will be using? So, stack,queue, and stdio libraries I think*

*For this project, our software will interface with C++ libraries to include I/O functionality and stack and queue data structures.*

### Communication Interfaces

This project will perform all the computation on one machine and won’t communicate with other systems. There will be no communication interfaces.

### Memory Constraints

There won’t be any necessary memory constraints on the software. It will be running on a general purpose computer, so there will be much more memory available than our program will use.

### Operations

??

## Product functions

The product will take in a typed arithmetic expression from the user, evaluate the result of the expression, and print the result to the user. The software must handle the arithmetic operations in the correct order. It must be able to evaluate expressions inside parentheses, and detect when there are unmatched parentheses. In addition, it must be able to handle invalid operations, such as division by 0 or invalid operators. In order to handle arithmetic expressions, it must be able to recognize the numeric constants as integers.

## User characteristics

This software should be usable for the largest possible group of users. So, the only requirements on the users is that they understand how to type input into the program. Even if they don’t understand how to format input, our software should respond to illegal input and instruct the user on the correct form of input.

## Constraints

The project must be developed in the language C++. Also, the product should have all code well documented for ease of legibility. In addition, we are required to use a data structure, such as a stack or a tree, to represent the arithmetic expression’s structure.

## Assumptions and dependencies

## Requirements subsets

# 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.]

### <Command Line Input>

*The product should receive an arithmetic expression from the user through input in the command line.*

### <Expression Parsing>

*The software should parse the expression input into a data structure that stores the structure of the expression, so the program can calculate the expressions by order of operations.*

### <Numeric Constant>

*The software should recognize numeric constants in the user input and assume they are integer form.*

### <Operator Support>

The software should be able to handle the following operators + (addition), - (subtraction), \* (multiplication), / (division), % (modulo), and ^ (exponential).

### <Parentheses Support >

The software should be able to evaluate expressions enclosed inside parentheses. In addition, the software should be able to detect unmatched parentheses and notify the user of the invalid input.

### <Operator Precedence>

*The software should evaluate the expression in order of operator precedence. So, expressions inside parentheses should be evaluated first, then exponential expressions, then multiplication, division, or modulo expressions, and finally addition and subtraction expressions.*

### *<Error Handling>*

*The software will handle invalid input from the user, such as incorrect operators, or operators without numbers. When these expressions are input, the program will notify the user of the error in their input.*

## 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.]

## 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.]

### <Development Language>

*The software should be written in the programming language C++*

### *<Comments>*

*The software should contain clear comments to describe the functionality of the program source code.*

### Any other non-functional requirements / constraints?

# 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** |
| ... |  |
| ... |  |

# Appendices

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