<Project Name>

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** |
| 3/21/2024 | 1.0 | Fill out Section 2 | Shravya |
| 3/22/2024 | 1.1 | Clean up section 2 | Katie |
|  |  |  |  |
|  |  |  |  |

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

# 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.]

## 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.]

## 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.]

## 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.]

## Overview

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

# Overall Description

**2.1 Product Perspective**

**2.1.1 System Interfaces:** Behind the scenes, the program will interface with system resources for memory management and error handling. This will be handled by the operating system and not our program.

**2.1.2 User Interfaces:** The user interfaces with the program through a command-line interface. The user will enter Boolean expressions to be evaluated through the command line. Finally, the command line will be used to display the calculated results of the evaluations to the user. The interface should be user-friendly and provide clear prompts and feedback.

**2.1.3 Hardware Interfaces:** The program does not have direct hardware interfaces as it operates solely in software. However, it relies on the underlying hardware for memory allocation and CPU processing.

**2.1.4 Software Interfaces:** The software interfaces with the C++ standard library for input/output operations and string manipulation. Additionally, it may interface with external libraries or modules for specific tasks such as parsing.

**2.1.5 Communication Interfaces:** There are no explicit communication interfaces required as the program operates locally on the user's machine without any network interaction.

**2.1.6 Memory Constraints:** The program should manage memory efficiently to handle expressions of varying complexity while adhering to memory constraints of the system. This includes proper memory allocation and deallocation to avoid memory leaks.

**2.1.7 Operations:** The program performs operations such as expression parsing, evaluation, error handling, and user interaction. It also involves managing data structures to represent Boolean expressions and their final evaluation values.

**2.2 Product Functions:**

* Parse Boolean expressions containing operators (!, $, @, &, |) and Boolean constants such as T for True and F for False.
* Evaluate the parsed expressions according to operator precedence rules outlined in 3.1.
* Handle parentheses to determine the order of evaluation within expressions.
* Recognize and handle Boolean constants within expressions.
* Provide a user-friendly command-line interface for inputting expressions and displaying results.
* Implement robust error handling to manage scenarios outlined in 3.1.

**2.3 User Characteristics:** Users of the program are expected to have basic knowledge of Boolean expressions and the operators involved. They should be comfortable using a command-line interface for input and output.

**2.4 Constraints:**

* The program must be written in C++.
* Memory usage should be optimized to handle expressions of varying sizes without exceeding system limits.
* Error handling should be comprehensive to handle all possible scenarios gracefully.
* The program will use object-oriented programming principles.
* The program will include comments to explain the logic and functionality of the code to further readers and programmers on the team.
* The program will have unit tests to verify the correctness of different operator evaluations and complex expressions.
* The program must provide clear and informative error messages that the user will find easy to understand.
* The program will have a user-friendly text based interface for entering Boolean expressions and viewing the results.

**2.5 Assumptions and Dependencies:**

* The program assumes that the input expressions provided by users may not be well-formed and as such our program will test their validity.
* Dependencies include the C++ standard library for input/output operations and possibly external libraries for specific functionalities like parsing.

**2.6 Requirements Subsets:**

Requirements subsets include tasks related to expression parsing, operator precedence, parenthesis handling, Boolean constants recognition, user interface development, error handling, and any additional features or optimizations beyond the specified requirements. These subsets are essential for organizing and prioritizing tasks during development.

# 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.]

### <Functional Requirement One>

[The requirement description.]

## 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.]

# 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]