|  |
| --- |
| GELLI Program Documentation |
| January 10, 2023 |

# Overview

Table of Contents

[Overview 2](#_Toc134448817)

[1. Introduction 3](#_Toc134448818)

[I. Purpose 3](#_Toc134448819)

[II. Intended Audience and Reading Suggestions 3](#_Toc134448820)

[III. Project Scope 3](#_Toc134448821)

[2. Overview 3](#_Toc134448822)

[I. Product Perspective 3](#_Toc134448823)

[II. Product Features 3](#_Toc134448824)

[III. User Classes and Characteristics 4](#_Toc134448825)

[IV. Operating Environment 4](#_Toc134448826)

[V. Design and Implementation Constraints 4](#_Toc134448827)

[3. Language Basics 4](#_Toc134448828)

[I. Syntax 4](#_Toc134448829)

[4. Deliverables 6](#_Toc134448830)

[5. Implementation Plan 6](#_Toc134448831)

[6. High-Level Timeline/Schedule 7](#_Toc134448832)

## Introduction

|  |  |
| --- | --- |
|  | 1. Purpose   **This software requirements specification document is meant to cover the programming language that is currently referred to as GELLI. It will describe the nature of the language, how to use it syntactically, and describe the standard library that comes associated with it. This document will additionally describe the methods for integrating the Web Assembly bundle into JavaScript applications.**   1. Intended Audience and Reading Suggestions   **The intended audience for this document will be those wishing to develop and further understand the language that is GELLI. This may consist of high schoolers practicing in this program or extracurricular activities of Dr. Lowe’s Students that may be attempting to improve or expand upon this project. It is suggested that one follows the documentation in order while using Section 7 for review of the program.**   1. Project Scope   **This project aims to create a beginner-friendly programming language built on Rust 1.67.0 and then exported to JavaScript via WebASM. Helping the younger generation learn and develop coding skills through simulated robotics programming can be better achieved if the language they are using to do it is much easier to write in.** |

## Overview

|  |  |
| --- | --- |
|  | 1. Product Perspective   *This programming language is being created as a new, self-contained product that will be implemented into a robotics program that is being re-developed alongside this language. The robotics program, known as GradBot, was developed by Dr. Lowe and currently uses JavaScript to operate the simulated robots. The program is being re-developed by a team and this new language is to be implemented into the interface of the program as a simpler way for users to program the robots to function.*   1. Product Features   *This is a strongly typed language with two number types: NUMBER (float) and TEXT (string). There are no objects, but there are user defined types like C structs. This language refrains from using confusing sigils to make it easier for new programmers to enter. Following that idea, it operates using line returns (like Python) and uses 1-based arrays. Many additional features will be discussed in detail in the completed language documentation, and a structural description can be found below via the provided BNF.*   1. User Classes and Characteristics   *We anticipate that the main users of this project will be highschoolers learning to program robotics through GradBot. These are the users we most aim to please because they will be the primary users, and any requirements for the project have been tailored to them. They will have little to no programming experience which means their experience should be as smooth as possible.*    *The other users of the language will be developers. They will be those looking to implement more features into the language or plug the language into other applications, primarily GradBot. Although it is important that the programming language is accessible to these users, their ease of access is lower priority.*   1. Operating Environment   *This programming language will target Windows 10 and onwards operating systems using Google Chrome web browsers via a WebASM exported package. Any other compatibility achieved is a welcomed side effect.*   1. Design and Implementation Constraints   The interpreter is written in Rust and is built to target WebASM and Javascript integration. It is possible that due to the nature of interpretation the language could run slow in some instances. Additionally, Rust is not a widely used language and may present a learning curve to those who are new to the project but want to develop code. |

## Language Basics

1. Syntax
2. **Arithmetic Operators**
   1. '+' (addition): Adds two values together.
   2. '-' (subtraction): Subtracts the second value from the first value.
   3. '\*' (multiplication): Multiplies two values together.
   4. '/' (division): Divides the first value by the second value.
   5. '^' (exponentiation): Raises the first value to the power of the second value.
3. **Comparison Operators**
   1. '<' (less than): Returns true if the first value is less than the second value.
   2. '<=' (less than or equal to): Returns true if the first value is less than or equal to the second value.
   3. '>' (greater than): Returns true if the first value is greater than the second value.
   4. '>=' (greater than or equal to): Returns true if the first value is greater than or equal to the second value.
   5. '=' (equal to): Returns true if the two values are equal.
   6. '!=' (not equal to): Returns true if the two values are not equal.
4. **Assignment Operator**
   1. '=' (assignment): Assigns the value of the right-hand side to the left-hand side variable.
5. **Grouping Symbols**
   1. '(' (left parenthesis): Used to group and prioritize expressions in mathematical operations.
   2. ')' (right parenthesis): Used to group and prioritize expressions in mathematical operations.
   3. '{' (left curly brace): Used to denote the start of a structure declaration.
   4. '}' (right curly brace): Used to denote the end of a structure declaration.
   5. '[' (left square bracket): Used to denote the start of an array.
   6. ']' (right square bracket): Used to denote the end of an array.
6. **Other Operators**
   1. ':' (colon): Used to denote a type or a label.
   2. ',' (comma): Used to separate values or arguments.
   3. '.' (period): Used to access members of a structure.
   4. 'and' (logical AND): Returns true if both values are true.
   5. 'or' (logical OR): Returns true if at least one of the values is true.
   6. 'bit\_or' (bitwise OR): Performs a bitwise OR operation on two values.
   7. 'bit\_xor' (bitwise XOR): Performs a bitwise XOR operation on two values.
   8. 'bit\_and' (bitwise AND): Performs a bitwise AND operation on two values.
   9. 'bit\_sl' (bitwise shift left): Shifts the bits of the first value to the left by the number of bits specified in the second value.
   10. 'bit\_sr' (bitwise shift right): Shifts the bits of the first value to the right by the number of bits specified in the second value.
   11. 'bit\_not' (bitwise NOT): Performs a bitwise NOT operation on a value.
   12. 'mod' (modulo): Returns the remainder of the first value divided by the second value.
7. **Keywords**
   1. 'definitions' : Keyword used to begin a block of function or structure definitions
   2. 'end' : Keyword used to end a block of function or structure definitions. Used in conjunction with the keywords ‘definitions’, ‘structure’, ‘program’, ‘function’, ‘repeat’, ‘while’, and ‘if’.
   3. 'structure' : Keyword used to define a structure type
   4. 'is linked / is not linked' : Keyword used with links to indicate whether the variable is linked or not. Relies on a pairing with ‘not’ to not be a link.
   5. 'function' : Keyword used to define a function
   6. 'returns' : Keyword used to specify the return type of a function
   7. 'return' : Keyword used to return a value from a function
   8. 'changeable' : Keyword used to specify a mutable variable in functions
   9. 'array' : Keyword used to define an array type
   10. 'of' : Keyword used in array definition to specify the type of its elements
   11. 'nothing' : Keyword used to represent the absence of a return value in functions
   12. 'program' : Keyword used to begin the main code body
   13. 'quit' : Keyword used to exit a program
   14. 'link' : Keyword used to link two variables together
   15. 'unlink' : Keyword used to unlink two variables
   16. 'to' : Keyword used in variable linking to specify the target variable
   17. 'break' : Keyword used to break out of a loop
   18. 'continue' : Keyword used to continue to the next iteration of a loop
   19. 'number' : Keyword used to define a numeric data type
   20. 'text' : Keyword used to define a string data type
   21. 'if' : Keyword used to begin an if statement
   22. 'then' : Keyword used to specify the block of code to execute if the if condition is true
   23. 'else' : Keyword used to specify the block of code to execute if the if condition is false
   24. 'while' : Keyword used to begin a while loop
   25. 'repeat' : Keyword used to begin a repeat-times, repeat-for, and repeat-forever loop
   26. 'forever' : Keyword used to begin an infinite loop
   27. 'times' : Keyword used in for loop to specify the number of iterations
   28. 'for all' : Keyword used with ‘repeat’ to loop over items in an array.

## Deliverables

|  |  |
| --- | --- |
|  | The Project Deliverables outlined in this scope are focused on the development of a new programming language called GELLI. The language is designed to be beginner-friendly, with a strong emphasis on ease of use and simplicity. The completed software requirements specification document will provide a detailed description of the language's nature, syntax, and standard library, making it easier for users to learn and utilize the language. The BNF will serve as a structural description for the language, providing developers with the necessary guidelines to write code.  The new language will be implemented into a robotics program called GradBot as a simpler way for users to program the robots to function. This implementation will be self-contained and exported to JavaScript via WebASM. The operating environment will target Windows 10 and onwards operating systems using Google Chrome web browsers. An interpreter written in Rust and built to target WebASM and JavaScript integration will allow users to execute code written in the new language.  The language itself will be strongly typed, with two number types, a TEXT type, and user-defined types like C structs. It will refrain from using confusing sigils and operate using line returns like Python, making it more accessible for new developers. The language will also use 1-based arrays, which will simplify the process of working with arrays. However, due to the nature of interpretation, the language may run slow in some instances, and the learning curve for new developers who want to develop code using Rust might be steep. To counteract this, the documentation will provide details on the language's features, making it easier for high schoolers learning to program robotics through GradBot to use the language. |

## Implementation Plan

|  |  |
| --- | --- |
|  | Our plan of implementation consisted of first defining the language's purpose and goals: Before starting to develop a programming language, we considered how important to have a clear idea of what it's going to be used for and what it needs to achieve so aspects had to be taken in mind to keep the language syntax as simplified as possible.  Next, we considered the language that our client wishes for us to make use of for this language. In this case we had made use of the RUST programming language as the basis for construction of our parser and interpreter. It was during these steps after that the operators were fully considered at the point in their design for a simplified understanding. Ensuring the language had an easy-to-understand semantics was especially important at this stage if we wished for the language to be easy to understand. The syntax is the set of rules that determine how the language's code is written and structured. It is important to keep the syntax as clear and concise as possible, making it easy for users to write and understand the code we had provided. During the development of the parser and interpreter, it was important to consider the language's data types, variables, functions, and control structures. These elements defined how the language handles logical data, and they need to be carefully crafted to ensure that the language is easy to use.  After the parser and interpreter have been developed, the next step was to test the language thoroughly. This involved creating test cases that cover all the features of the language, including edge cases and error handling. The testing phase is often considered critical during this period for identifying bugs and ensuring that the language functions as intended.  Finally, once the language was fully tested and refined, it was to be released back to our client for the GradBot team to work upon the. This iterative process ensured that the language would be allowed to be upgraded by future teams that worked upon this code. |