# COMP6991 Rust Logo Interpreter Assignment

**Logo Language Overview**

- Logo is a programming language derived from Lisp and others.
- Older programmers often had their first programming experience with Logo.
- Key feature is a "turtle" for drawing by picking up and putting down a pen and moving around.

**Assignment Goals**

- Practice designing and
- Focus on modern prog
- Have fun creating an a                                                                    cting with programming history.

**Assignment Requirement**

- Build a Logo interprete                                                              nsvg crate.
- Handle various Logo c                                                              conditionals, and math operations.
- Implement design exce

# 1. Introduction

## Tokens

- A token can be a procedure (like a function), a variable (prefixed by `:`), or a value (prefixed by `"`).
- Procedures always take a fixed number of arguments.
- Values in Logo are always strings, but some like `"TRUE"` and `"FALSE"` are interpreted as booleans and some can be parsed as numbers.

## Program Structure

- A logo program consists of lines of text split into tokens by whitespace.
- Lines starting with `//` or empty lines are ignored as comments.
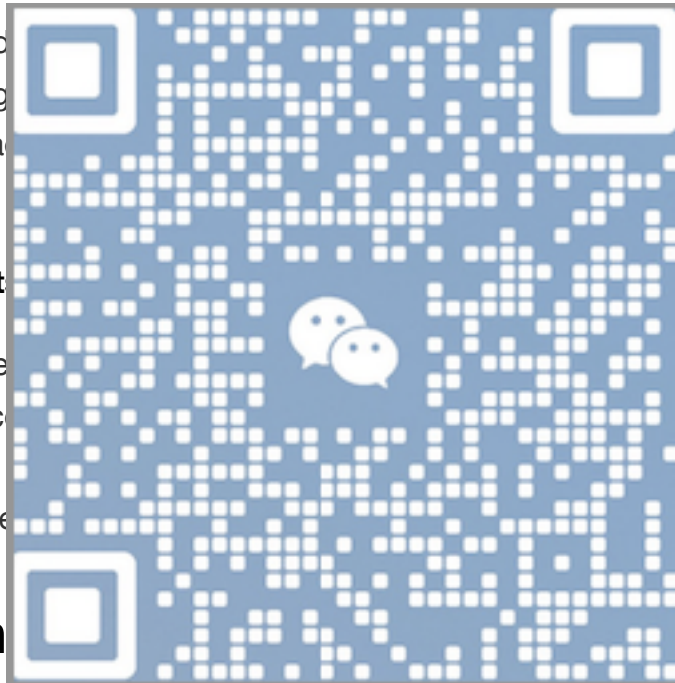
## 2. Introduction to Unsvg

- The assignment uses the `unsvg` crate to generate SVG or PNG images.
- `unsvg::Image` represents an image and has methods like `draw_simple_line`.
- `unsvg::get_end_coordinates` returns where a line drawn from a given point would end.

## 3. How Your Program Will Work

- Produce a program called `rslogo` that takes four arguments: a logo program file (.lg), the output SVG/PNG file path (.svg or.png), image height, and image width.
- Read the logo program, parse and execute it line by line.
- Exit with a non-zero return code and print an error message if there's an issue.
- Write an SVG or PNG u

## 4. Design Exce

- Options for design exc <br> ng 80% test coverage, using a parser combina <br> sions, building a zero-copy program, contributing
- Markers will consider a <br> fficient.

## 5. The Tasks To

### Part 1: Turtle Cor

- Control the "turtle" which is like an invisible pen that can draw on the image.
- Turtle starts "up" (not drawing) in the center of the screen facing straight up.
- Commands include `PENUP`, `PENDOWN`, `FORWARD`, `BACK`, `LEFT`, `RIGHT`, `SETPENCOLOR`, `TURN`, `SETHEADING`, `SETX`, `SETY`.
- Turtle can go off the image without causing an error.

### Part 2: Variables and Queries (20%)

- Implement the `MAKE` command to create and assign variables.
- Implement the `ADDASSIGN` command for variable increment.
- Support "queries" like `XCOR`, `YCOR`, `HEADING`, `COLOR`.
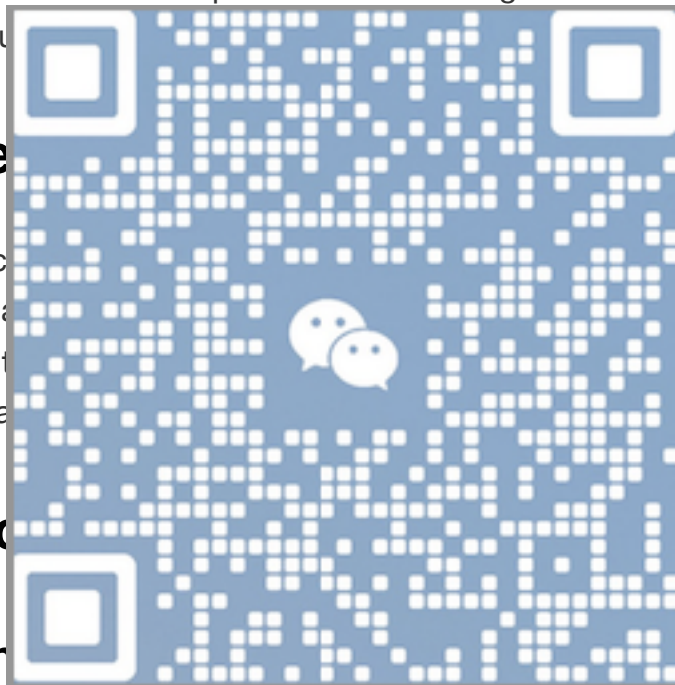
## Part 3: IFs, WHILE, [ ] (20%)

- Implement `IF EQ` and `WHILE EQ` commands for conditional execution and looping.

## Part 4: Implementing Maths and Comparisons using a Stack (20%)

- Implement operations in Polish Notation like `EQ`, `NE`, `GT`, `LT`, `AND`, `OR`, `+`, `−`, `*`, `/`.
- Implement stack operations for `IF` and `WHILE`.

## Part 5: Logo Defined Procedures (20%)

- Implement procedures analogous to functions in other languages.
- Procedures are defined          procedure name and arguments, and ending

# 6. Common Qu

## Design Approach

- Line-by-line approach:                                          ta and state.
- Parse then execute ap                                          Tree and read from it.

## Planning

- A happy middle is sugg                                          ugh the assignment without spending more than 30 minutes planning.

## Using AI

- Permitted uses of AI include seeking help with concepts, pattern matching, generating skeletons, and writing tests.

# 7. Other Information

## Submission

- See instructions at the bottom of the page.