
Visualization Compiler with Intermediate C Code Generation

---

1. Introduction

This project implements a domain-specific compiler that processes custom `.wiz` language files to generate Python data visualization scripts. The compiler uses a two-stage code generation strategy:

- First, it generates an intermediate C program.
- This C program, when compiled and executed, produces the final Python visualization code.

The compiler provides an automated, extensible system to generate various types of plots directly from simple `.wiz` files, helping visualize structured datasets.

Additionally, the dataset used for visualization is extracted from a PDF document through a preprocessing step and saved into a structured CSV file (`dataset.csv`).

---

2. Data Preprocessing

Before the compiler can operate, the raw data is extracted from a PDF file. A preprocessing tool reads the PDF and converts the numerical information into a standard CSV file named `dataset.csv`.

This CSV serves as the input for all plotting functions during the visualization stage.

---

3. Project Architecture Overview

The project follows a classical compiler design broken into the following key components:

3.1. Lexical Analysis (`tokenizer.l`)

Tool: Flex

Purpose:

- Breaks down `.wiz` source files into basic tokens such as keywords, identifiers, numbers, operators, and punctuation.

3.2. Syntax Analysis (`grammar.y`)

Tool: Bison

Purpose:

- Parses token streams into syntactically valid structures.
- Builds a high-level semantic understanding of assignments, function calls, conditionals, loops, and expressions.

3.3. Symbol Table Management (`symbol_registry.c`, `symbol_registry.h`)

Purpose:

- Keeps track of declared variables and their values.
- Ensures no duplicate declarations.
- Enables variable lookup and update during code generation.

3.4. Intermediate C Code Generation (`emitter.c`, `emitter.h`)

Purpose:

- Generates C source code (`out.c`) containing formatted `printf` statements.
- These statements output valid Python visualization code line-by-line when executed.

3.5. Final Python Code Generation and Execution

- The generated C program (`out.c`) is compiled into an executable (`visualizeit`).

- Running `visualizeit` prints Python code, which is saved into a `.py` file.
- The Python script is then executed to create the final plot.

3.6. Automation Script (`main_controller.sh`)

Purpose:

- Automates the workflow from compilation to execution.
- Processes all `wiz` files inside `vizscripts/`.

---

4. Visualization Functions

Six `wiz` scripts are implemented, each corresponding to a different plot type:

Plot Type	Wiz File	Description
Bar Chart	`barchart.wiz`	A traditional bar graph showing each index.
Box Plot	`boxplot.wiz`	Summarizes spread, quartiles, and outliers.
Line Plot	`lineplot.wiz`	Connects data points to show trends.
Density Plot (KDE)	`density.wiz`	Smooth probability density estimation.
ECDF Plot	`ecd.wiz`	Cumulative distribution function.
Stacked Area Plot	`stacked.wiz`	Area plot with stacked series.

Each plot is generated by compiling and executing a separate `.wiz` file into its corresponding `.py` visualization script.

---

5. Grammar Approach and Design Decisions

- Typed grammar using `%union` for numbers, identifiers, and strings.
- Simple and flexible expression parsing.
- Minimal syntax constructs for clarity and simplicity.
- Dynamic generation of Python code through intermediate C programs.

---

6. How to Build and Run the Project

Requirements

Install necessary tools:

sudo apt install gcc flex bison python3
pip install matplotlib seaborn

Folder Structure

Folder/File	Purpose
`vizscripts/`	Contains `.wiz` files for each plot type
`tokenizer.l`	Lexer (Flex)
`grammar.y`	Parser (Bison)
`symbol_registry.c/h`	Symbol table management
`emitter.c/h`	Intermediate C code generator
`main_controller.sh`	Main automation script
`dataset.csv`	Input data extracted from PDF
`Makefile`	Compilation script

Build the Project

make clean
make

This compiles the main compiler executable, `vizrunner`.

Run the Visualization Automation

chmod +x main_controller.sh
./main_controller.sh dataset.csv

What happens:

- For each `.wiz` file:
    - Compiles it into a C program (`out.c`).
    - Compiles `out.c` to create an executable (`visualizeit`).
    - Runs `visualizeit` to generate the corresponding Python script.
    - Executes the Python script to display the plot.

Example Output

Building project...
Processing barchart.wiz...
Successfully generated and executed barchart.py
Processing boxplot.wiz...
Successfully generated and executed boxplot.py
...
All processing is complete.

---

7. Conclusion

This project showcases a complete compiler-based visualization system with:

- Lexical and syntax analysis.

- Symbol management.
- Intermediate C code generation.
- Final dynamic Python visualization.

The architecture is modular, extensible, and automates data visualization workflows from a simple, user-friendly `Wiz` language.