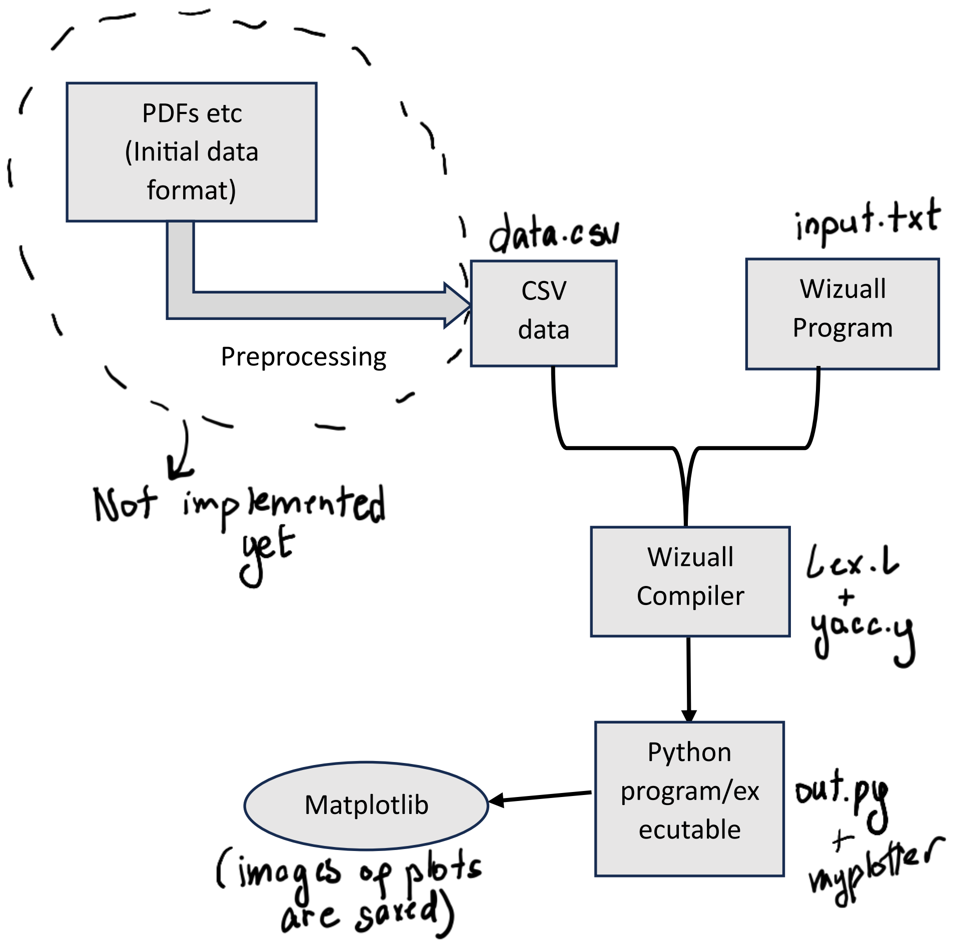
Compiler Construction Assignment

Siddharth Chaitra Vivek – 2022A7PS0569G

Introduction

This is a report on my attempt to design a custom language and a corresponding compiler that converts the commands to an equivalent python script. The primary motivation is to allow visual representation of tabular data which is preprocessed into an easy format (like csv, which I have worked with) from sources like pdfs. The plotting of the data is done using python matplotlib library in my implementation

Proposed Scheme and Pipeline



Program Structure

```
Wizuall_Compiler/
├── data.csv          # Input CSV file with data
├── input.txt         # Sample input to be parsed
├── lex.l             # Flex lexical analyzer source
├── lex.yy.c          # Generated C file from lex.l
├── makefile          # Build script for the project
├── myplotter         # Compiled executable (from lex + yacc)
├── out.py            # Python script for plotting (likely uses matplotlib)
├── yacc.tab.c        # Generated parser code from yacc
├── yacc.tab.h        # Generated header from yacc
├── yacc.y            # Yacc grammar file
```

Syntax

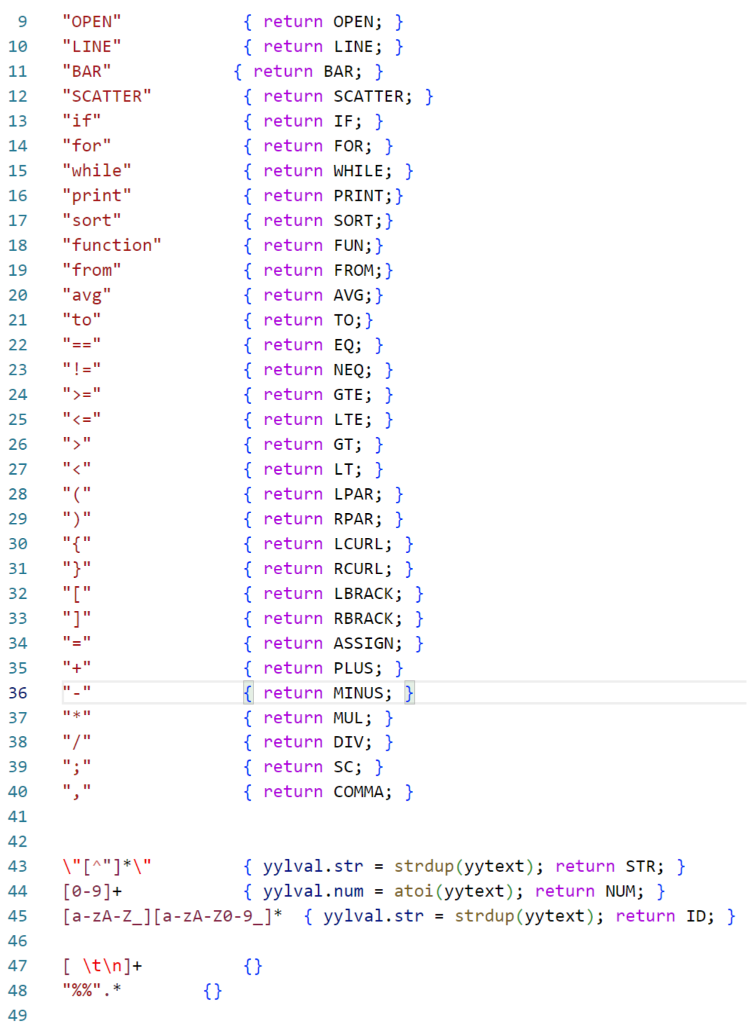
Wizuall Syntax	**Python Output**
OPEN df "file.csv";	df = pd.read_csv("file.csv")
LINE x y;	plt.plot(x, y); plt.savefig('line.png')
SCATTER a b;	plt.scatter(a, b); plt.savefig('bar.png')
BAR x;	plt.hist(x); plt.savefig('scatter.png')
print(val);	print(val)
a = b + c;	a = b + c
a = sort list;	a = sorted(list)
list = [1, 2, 3];	list = [1, 2, 3]
fun func_name(parameters) { ... }	def name(params): ...
for i from 1 to 5 { ... }	for i in range(1, 5+1): ...
if (x > y) { ... }	if x > y: ...
while (x < 10) { ... }	while x < 10: ...

Semantics and Implementation

I decided to build from scratch and add the necessary functionality based on the need taking the skeleton of the code from the flex and bison compiler we used for calcilist

Here the Wizuall compiler utilizes Lex and Yacc for compiling a custom domain-specific language into Python code. The Lex file (lex.l) is responsible for tokenizing the input, identifying keywords (e.g., OPEN, LINE, PRINT), operators (==, !=, =, etc.), identifiers, string literals, and numbers. Tokens are passed to the Yacc parser for syntactic analysis.

The Yacc file (yacc.y) defines a grammar that maps high-level commands into valid Python constructs. For instance, commands like OPEN data "file.csv"; are translated into data = pd.read_csv("file.csv"), and plotting commands like LINE x y; generate Matplotlib code. The parser supports basic control structures (if, for, while), function definitions, variable assignments, and common data processing operations. The result is a Python script that integrates pandas and matplotlib functionality, providing a streamlined way to script data operations using a simplified syntax.



```
"OPEN"              { return OPEN; }
"LINE"              { return LINE; }
"BAR"             { return BAR; }
"SCATTER"           { return SCATTER; }
"if"                { return IF; }
"for"               { return FOR; }
"while"             { return WHILE; }
"print"             { return PRINT;}
"sort"              { return SORT;}
"function"          { return FUN;}
"from"              { return FROM;}
"avg"               { return AVG;}
"to"                { return TO;}
"=="                { return EQ; }
"!="                { return NEQ; }
">="                { return GTE; }
"<="                { return LTE; }
">"                 { return GT; }
"<"                 { return LT; }
"("                 { return LPAR; }
")"                 { return RPAR; }
"{"                 { return LCURL; }
"}"                 { return RCURL; }
"["                 { return LBRACK; }
"]"                 { return RBRACK; }
"="                 { return ASSIGN; }
"+"                 { return PLUS; }
"-"                 { return MINUS; }
"*"                 { return MUL; }
"/"                 { return DIV; }
";"                 { return SC; }
","                 { return COMMA; }


\"[^"]*\"           { yylval.str = strdup(yytext); return STR; }
[0-9]+              { yylval.num = atoi(yytext); return NUM; }
[a-zA-Z_][a-zA-Z0-9_]*  { yylval.str = strdup(yytext); return ID; }

[ \t\n]+            {}
"%%".*          {}

```

The various tokens formed after tokenization by the lexer

1. **OPEN Statement**

```
OPEN ID STR SC {
    print_indent();
    fprintf(output, "%s = pd.read_csv(%s)\n", $2, $3);
}
```

- **Purpose:** To load a CSV file into a Pandas DataFrame.
- **Example Input:** `OPEN df "data.csv";`
- **Output Code:** `df = pd.read_csv("data.csv")`
- **Explanation:** This rule uses the `ID` as the variable name and the `STR` as the filename. It simplifies data loading by hiding `pandas` syntax behind a single custom command.

---

2. **LINE Plot Statement**

```
LINE ID ID SC {
    fprintf(output,
        "plt.figure(figsize=(8, 4))\n"
        "plt.plot(df['%s'], df['%s'])\n"
        "plt.xlabel('%s')\n"
        "plt.ylabel('%s')\n"
        "plt.savefig('line.png')\n", $2, $3, $2, $3);
}
```

- **Purpose:** To create and save a line plot using matplotlib.
- **Example Input:** `LINE x y;`
- **Output Code:**

  ```
  plt.figure(figsize=(8, 4))
  plt.plot(df['x'], df['y'])
  plt.xlabel('x')
  plt.ylabel('y')
  plt.savefig('line.png')
  ```

- **Explanation:** This rule takes two identifiers (assumed to be column names) and generates a complete line plot with labeled axes and an output image.

---

3. **PRINT Statement**

```
PRINT expr SC {
    print_indent();
```

```
    fprintf(output, "print(%s)\n", $2);
}
```

- **Purpose:** To print the value of an expression.
- **Example Input:** `PRINT x;`
- **Output Code:** `print(x)`
- **Explanation:** This allows the user to print any valid expression, variable, or result. The expression is first evaluated and then passed to the Python `print()` function.

---

4. **SORT Statement**

```
SORT ID SC {
    print_indent();
    fprintf(output, "%s.sort_values(inplace=True)\n", $2);
}
```

- **Purpose:** To sort a DataFrame in-place.
- **Example Input:** `SORT df;`
- **Output Code:** `df.sort_values(inplace=True)`
- **Explanation:** This provides a simplified command for sorting datasets without needing detailed Pandas syntax.

---

5. **Assignment Statement**

```
assignment:
    ID ASSIGN expr SC {
        print_indent();
        fprintf(output, "%s = %s\n", $1, $3);
    }
```

- **Purpose:** To assign values or expressions to variables.
- **Example Input:** `total = x + y;`
- **Output Code:** `total = x + y`
- **Explanation:** Enables variable assignments using expressions, numbers, strings, or other variables. A fundamental feature for building logic.

---

6. **IF Statement**

```
IF condition LCURL p_items RCURL {
    fprintf(output, "if (%s):\n", $2);
    indent++;
    fprintf(output, "%s", $4);
    indent--;
}
```

- **Purpose:** To create conditional execution blocks.
- **Example Input:**

  ```
  IF x > y {
    PRINT "X is greater";
  }
  ```

- **Output Code:**

  ```
  if (x > y):
      print("X is greater")
  ```

- **Explanation:** This rule generates Python `if` statements with proper indentation. The condition is matched by another rule and printed in parentheses.

---

7. **FOR Loop**

```
FOR ID FROM expr TO expr LCURL statement RCURL {
    print_indent();
    fprintf(output, "for %s in range(%s, %s):\n", $2, $4, $6);
    indent++;
    print_indent();
    fprintf(output, "%s\n", $8);
    indent--;
}
```

- **Purpose:** To create a Python `for` loop over a range.
- **Example Input:**

  ```
  FOR i FROM 0 TO 5 {
    PRINT i;
  }
  ```

- **Output Code:**

  ```
  for i in range(0, 5):
      print(i)
  ```

- **Explanation:** Converts a high-level iteration construct into Python's `range()` based `for` loop. Handles indentation automatically.

---

8. WHILE Loop

```
WHILE condition LCURL p_items RCURL {
    fprintf(output, "while (%s):\n", $2);
    indent++;
    fprintf(output, "%s", $4);
    indent--;
}
```

- **Purpose:** To create a `while` loop.
- **Example Input:**

  ```
  WHILE x < 10 {
    x = x + 1;
  }
  ```

- **Output Code:**

  ```
  while (x < 10):
      x = x + 1
  ```

- **Explanation:** Simplifies loop creation, allowing repetitive operations until a condition is met.

---

9. Function Definition

```
FUN ID LPAR parameters RPAR LCURL p_items RCURL {
    fprintf(output, "def %s(%s):\n", $2, $4);
    indent++;
    fprintf(output, "%s", $7);
    indent--;
}
```

- **Purpose:** To define Python functions.
- **Example Input:**

  ```
  function add(x, y) {
    PRINT x + y;
  }
  ```

- **Output Code:**

```
def add(x, y):
    print(x + y)
```

- **Explanation:** Enables modular and reusable code. The function body is parsed recursively using `p_items`.