Course Project

Second Deliverable: The Parser

Quin'darius Lyles-Woods
qlyleswo@students.kennesaw.edu

Concepts of Programming Languages
Professor Jose Garrido
Section W01
4308

Bachelors of Computer Science
Kennesaw State University
1100 South Marietta Pkwy SE
Marietta, GA 30060
October 25, 2021

Task

The development of an interpreter for a subset of **Basic Language**.

Project Goals

- Process a **Basic Language** source code file.
- Tokenize the source code file.
- Detect syntactical error.
- Display appropriate error messages during runtime.

Deliverable Goals

- Show statements regarding the input from scanner.
- Properly identify the tokens that the scanner has given.
- Output a parse tree file for the future interpreter.

Subset of the Basic Language

The subset of basic that I want to define for the course project is going to be kept as minimal as possible to focus on the process of developing an interpreter. With that in mind the lowest we can go is turing complete of course. For this to be true the language doesn't need much.

- Recursive Operators
  - `FOR...TO...NEXT`
- Conditional Jumps
  - `IF...THEN...{ELSE}`
  - `GOTO`
- Variables
  - `LET`

Tokens

- LET
- IF
- FOR
- GOTO
- +
- -
- *
- /
- =

Backus Normal Form of Basic Subset

$$
\begin{aligned}
BASIC\ PROGRAM ::= & \\
& \mid EXPRESSION, EXPRESSION \\
& \mid EXPRESSION \\
EXPRESSION ::= & \\
& \mid LET \\
& \mid IF \\
& \mid FOR \\
& \mid GOTO \\
& \mid IDENTIFIER\ +\ EXPRESSION \\
& \mid IDENTIFIER\ -\ EXPRESSION \\
& \mid IDENTIFIER\ *\ EXPRESSION \\
& \mid IDENTIFIER\ /\ EXPRESSION \\
LET ::= & \\
& \mid IDENTIFIER = EXPRESSION \\
IDENTIFIER ::= & \\
& \mid [a - zA - Z] \\
& \mid [a - zA - Z], [a - zA - Z] \\
IF ::= & \\
& \mid EXPRESSION\ then\ STATEMENT \\
& \mid EXPRESSION\ then\ STATEMENT\ else\ IF \\
FOR ::= & \\
& \mid EXPRESSION\ to\ EXPRESSION\ STATEMENT\ next \\
GOTO ::= & \\
& \mid [0 - 9] \\
& \mid [0 - 9], [0 - 9]
\end{aligned}
$$

Source Code

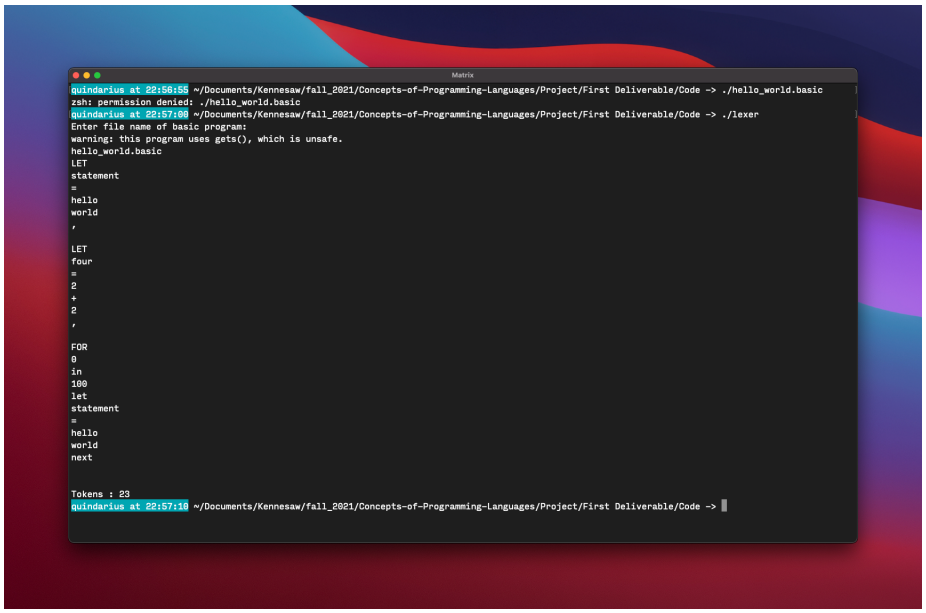
```c
#include <stdio.h>

#define IN 1
#define OUT 0

int main()
{
        char file_name[25];
        FILE *fp;
        printf("Enter file name of basic program: \n");
        gets(file_name);
        fp = fopen(file_name, "r");
        if (fp == NULL)
        {
                perror("Error opening this file");
        }
        int character,                          // Gets individual character input
            tokens,                             // Count of tokens
            state;                              // Determines if we are inside or outside a word
        state = OUT;
        tokens = 0;
        while ((character = fgetc(fp)) != EOF)  //Loops through the input
        {
                printf("%c",character);
                // If statement to check if we are in a word
                if(character == ' ' || character == '\n' || character == '\t')
                {
                        printf("\n");
                        state = OUT;
                }
                // If statement to tell if we are starting or finishing a word
                else if(state == OUT)
                {
                        state = IN;
                        tokens++;
                }
        }
        printf("\nTokens : %d\n",tokens);
        fclose(fp);
}
```

lexer.c

Compiling

```
quindarius at 22:55:33 ~/Documents/Kennesaw/fall_2021/Concepts-of-Programming-Languages/Project/First Delivera
ble/Code -> cc lexer.c -o lexer
quindarius at 22:56:00 ~/Documents/Kennesaw/fall_2021/Concepts-of-Programming-Languages/Project/First Delivera
ble/Code ->
```

Output



0.1 Summary

After finishing this project I have learned what it takes to build a simple lexer(Scanner) for a language. This feels very incomplete but I know that it is only on piece out of many when it comes to building an intepreter. I am glad I kept the subset super small since working by myself I could make the project a little more approachable and actually used the free mind space to make the lexer in C so I could use a very portable language that someday in the future might come in handy. I can see where this work will also help me in my Natural Language Processing class. stripping text of tokens is such a huge part of the field and I was glad to be able to do this in C because when working with large amounts of data you would probably want to use a fast language. C fills this requirement but now I need to do my part and learn how to optimize the program.