Course Project

Second Deliverable: The Parser

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

Concepts of Programming Languages
Professor Jose Garrido
Section W01
4308

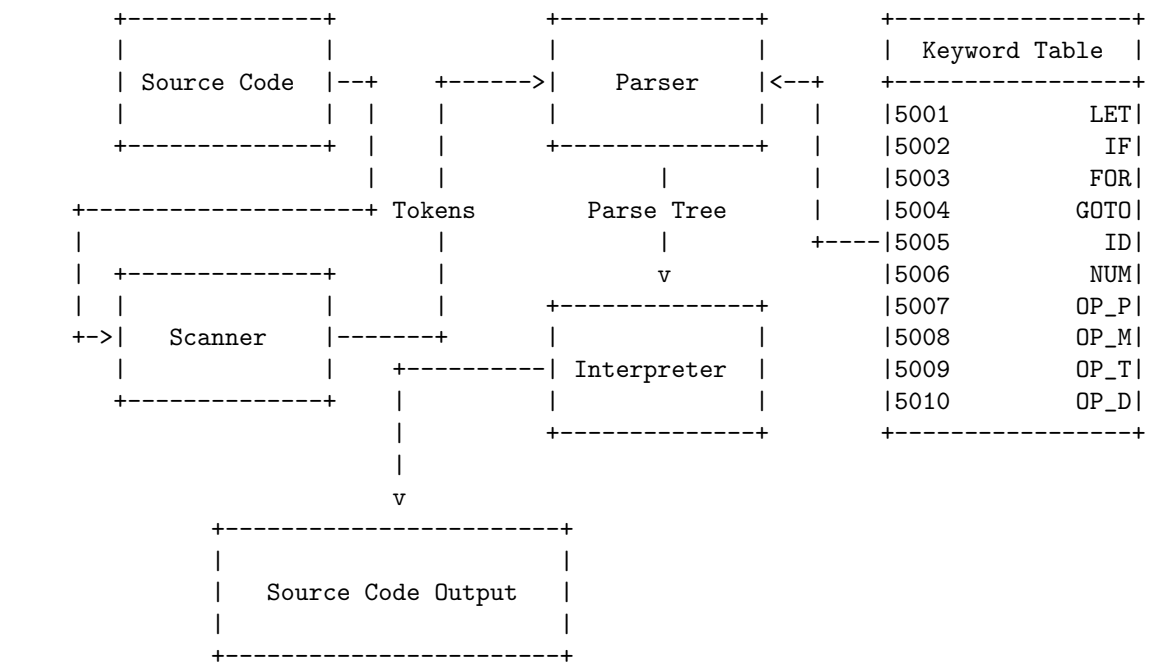
Bachelors of Computer Science
Kennesaw State University
1100 South Marietta Pkwy SE
Marietta, GA 30060
October 26, 2021

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.



Updates From Previous Deliverable

- Changed file workflow to allow for command line file passing instead of specifying the name during the program.
- Added error handling for when the file is not recognized.
- Changes to the BNF of the program

Deliverable Goals

I am building a parser that will utilize top down parsing.

- 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
- THEN
- ELSE
- FOR
- TO
- NEXT
- GOTO
- +
- -
- *
- /
- =

Backus Normal Form of Basic Subset

$$
\begin{aligned}
BASIC\ PROGRAM ::= & \\
& | EXPRESSION, EXPRESSION \\
& | EXPRESSION \\
EXPRESSION ::= & \\
& | LET \\
& | IF \\
& | FOR \\
& | GOTO \\
& | IDENTIFIER + EXPRESSION \\
& | IDENTIFIER - EXPRESSION \\
& | IDENTIFIER * EXPRESSION \\
& | IDENTIFIER / EXPRESSION \\
LET ::= & \\
& | IDENTIFIER = EXPRESSION \\
IDENTIFIER ::= & \\
& | [a - zA - Z] \\
& | [a - zA - Z], [a - zA - Z] \\
IF ::= & \\
& | EXPRESSION\ \textbf{THEN}\ STATEMENT \\
& | EXPRESSION\ \textbf{THEN}\ STATEMENT\ \textbf{ELSE}\ IF \\
FOR ::= & \\
& | EXPRESSION\ \textbf{TO}\ EXPRESSION\ \textbf{NEXT} \\
GOTO ::= & \\
& | [0 - 9] \\
& | [0 - 9], [0 - 9]
\end{aligned}
$$

Source Code


Code — lexer.c — vim lexer.c — 80×24
15 #include <stdio.h>
14 #include <errno.h>
13 #include <string.h>
12
11 #define IN 1
10 #define OUT 0
9
8 int main(int argc, char *argv[])
7 {
6 FILE *fp;
5 int errnum;
4 fp = fopen(argv[1], "r");
3 if (fp == NULL)
2 {
1 errnum = errno;
0 fprintf(stderr, "Value of %d\n",errno);
1 perror("Error opening this file");
2 fprintf(stderr, "Error opening file: %s\n",
strerror(errnum));
3 }
4 else
5 {
@
"lexer.c" 45L, 988B written
16,29-43 Top


Code — parser.c — vim parser.c — 80×24
0 #include <stdio.h>
1 #include <string.h>
2 int main ()
3 {
4 int index;
5 int word_count = 10;
6 char words[word_count][45];
7
8 for (index = 0; index < word_count; index++)
9 {
10 scanf("%s", words[index]);
11 }
12
13 for (index = 0; index < word_count; index++)
14 {
15 printf("%s \t\t|--> index: %i", words[index], index);
16
17 if(!strcmp("LET",words[index]))
18 {
19 printf("\t\t\tEXPRESSION: LET");
20 }
21 else if(!strcmp("IF",words[index]))
22 {
1,1 Top

Compiling


Code — -zsh — 80×24
quindarius at 19:48:03 Second Deliverable/Code -> cc parser.c -o parser
quindarius at 19:48:17 Second Deliverable/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 interpreter. 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.