Ben Pallotti

Concepts of Programming Languages-4308-W02­ Summer 2023

**P2-Parser**

7/11/23  
Professor Sharon Perry  
100% Complete

**Introduction**

The parser works in conjunction with the scanner from P1 to process the tokens produced by the scanner and identify the structure and syntax of the Julia program. The parser is implemented in Java.

**Parser Implementation**

Takes a file as input and parses the file line by line. For each line, the parser checks if the current token is a keyword, an identifier, or an error. If the current token is a keyword, the parser prints the keyword and its symbol. If the current token is an identifier, the parser prints the identifier and its symbol. If the current token is an error, the parser prints an error message.

**Execution and Output**

To demonstrate the execution of the parser the input Julia files named Test1.jl were used. The input files were given by the assignment.

Here is the execution output:

A screen shot of a computer screen

Description automatically generated

**Conclusion**

The parser implementation complements the scanner by analyzing the syntax of the Julia program. It successfully recognizes the statements and constructs. The recursive descent parsing technique handles the different language constructs. The parser has been tested with a sample text and produces the expected output.

**Below are the 3 files: JuliaScanner.java, JuliaParser.java, and JuliaAssignmentMain.java.**

package projects;

package projects;

/\*

\* Class: CS 4308 Section W02

\* Term: Summer 2023

\* Name: Ben Pallotti

\* Instructor: Sharon Perry

\* Project: P2-Parser

\*/

import java.io.File;

import java.io.FileNotFoundException;

import java.util.HashMap;

import java.util.Scanner;

public class JuliaParser {

private Scanner fileScanner;

private int lineNumber;

private String currentToken;

private HashMap<String, String> lexemeSymbolTable;

public void parse() throws FileNotFoundException {

while (fileScanner.hasNext()) {

currentToken = fileScanner.next();

lineNumber++;

// Check if the current token is a keyword

if (lexemeSymbolTable.containsKey(currentToken)) {

System.out.println(currentToken + " - " + lexemeSymbolTable.get(currentToken));

} else {

// Check if the current token is an identifier

if (Character.isLetter(currentToken.charAt(0))) {

System.out.println(currentToken + " - IDENTIFIER");

} else {

// The current token is not a keyword or an identifier, so it is an error

System.out.println("Error: Invalid token " + currentToken + " at line " + lineNumber);

}

}

}

}

public static void main(String[] args) throws FileNotFoundException {

Scanner userInput = new Scanner(System.in);

System.out.println("Enter File Location:");

String filePath = userInput.nextLine(); // Read user input

JuliaParser parser = new JuliaParser(filePath);

parser.parse();

userInput.close();

}

public JuliaParser(String filename) throws FileNotFoundException {

fileScanner = new Scanner(new File(filename));

lineNumber = 1;

currentToken = null;

lexemeSymbolTable = new HashMap<>();

lexemeSymbolTable.put("function", "FUNCTION");

lexemeSymbolTable.put("(", "OPEN\_PARENTHESIS");

lexemeSymbolTable.put(")", "CLOSE\_PARENTHESIS");

lexemeSymbolTable.put("=", "ASSIGNMENT\_OPERATOR");

lexemeSymbolTable.put("!=", "NOT\_EQUAL\_OPERATOR");

lexemeSymbolTable.put("+", "ADDITION\_OPERATOR");

lexemeSymbolTable.put("+=", "ADDITION\_ASSIGNMENT\_OPERATOR");

lexemeSymbolTable.put("<", "LESS\_THAN\_OPERATOR");

lexemeSymbolTable.put("==", "EQUAL\_TO\_OPERATOR");

lexemeSymbolTable.put("if", "IF");

lexemeSymbolTable.put("then", "THEN");

lexemeSymbolTable.put("print", "FUNCTION");

lexemeSymbolTable.put("else", "ELSE");

lexemeSymbolTable.put("end", "END");

lexemeSymbolTable.put("//", "COMMENT");

//Loop to parse numbers

for (int i = 0; i < 20; i++) {

lexemeSymbolTable.put(Integer.toString(i), "NUMBER");

}

}

}