Mini-Pascal Compiler - Software Design Document

Allen Burgett

2016-12-6

1 Summary

This program can scan a document for Pascal type tokens and return them to the user.

2 Structure

The complier consists of five key parts and main function that ties them all together.

3 Scanner

The program relies primarily on JFlex, a Java program that builds a state machine to process a grammar. The JFlex spec file contains all the criteria for what constitutes as a Pascal token. The JFlex spec generates Java code, based on the defined state machine. This Java can be compiled and added to an existing Java Program.

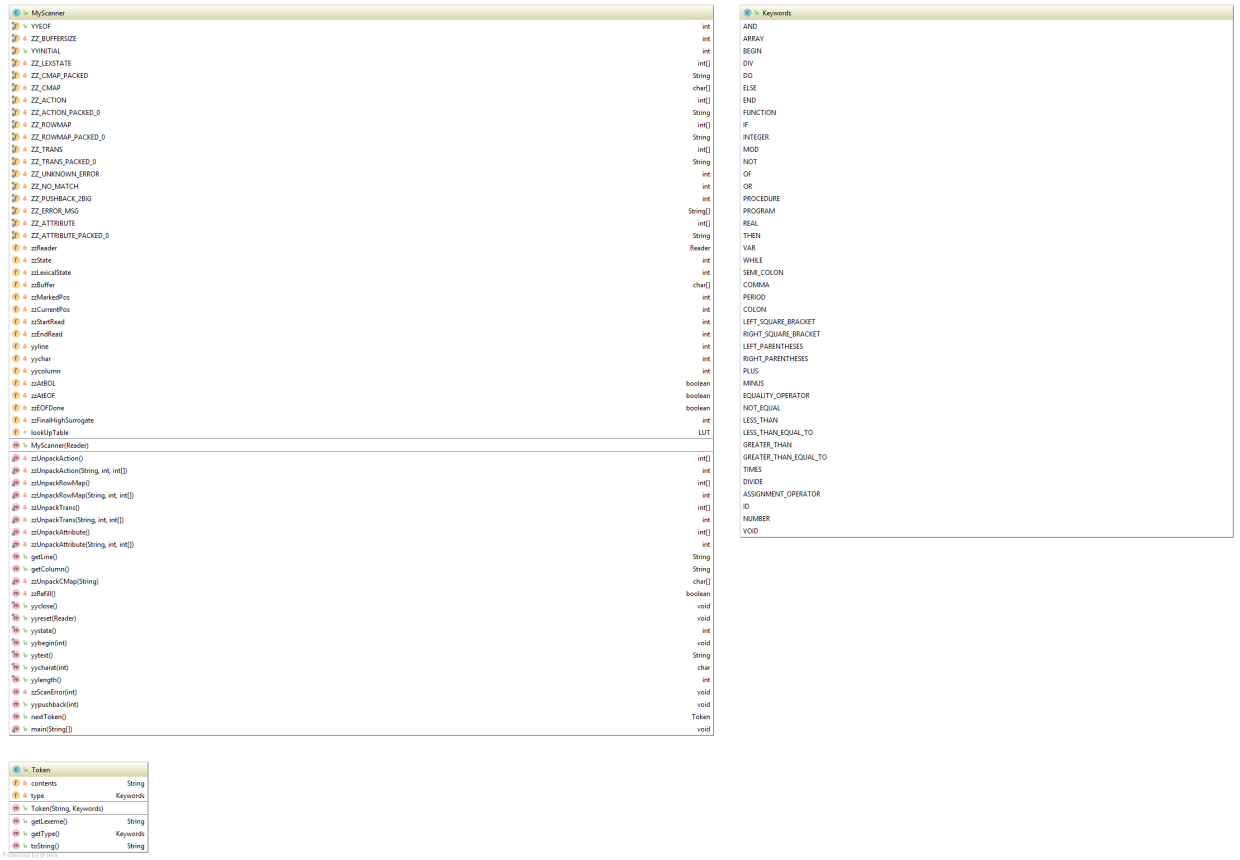

Figure 1: Scanner Package Diagram

3.1 MyScanner.JFlex

This file defines the parameters of a state machine. The generated state machine is used to identify Pascal Tokens. This generation creates the MyScanner.java file. Therefore the best way to adjust the scanner is by redefining the .jflex file.

4 Parser

The parser acts as the main driver for the compiler. It utilizes both the scanner and the syntax tree to parse the individual tokens into a tree structure.

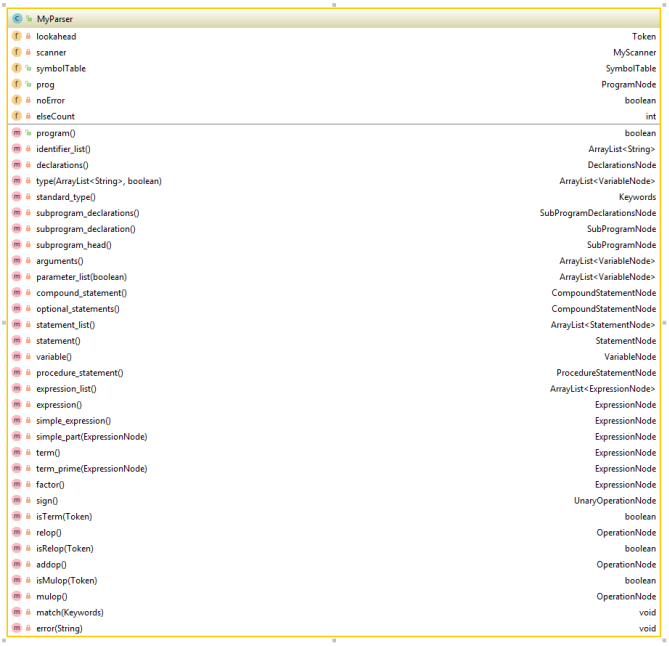

Figure 2: Parser Class Diagram

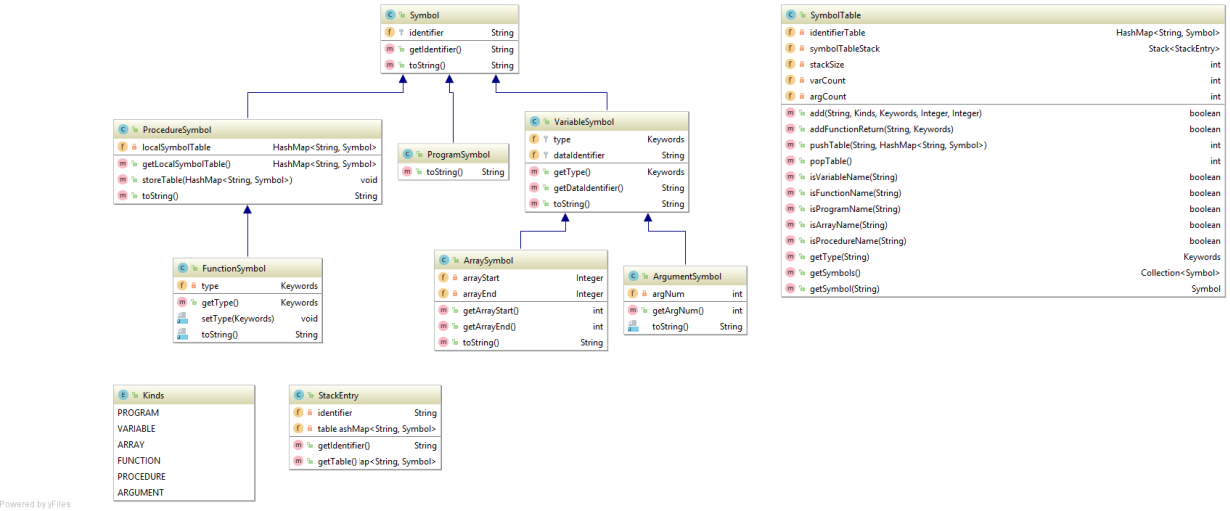

Figure 3: Parser Package Diagram

5 Syntax Tree

Contains individual classes for each possible structure in mini-pascal.

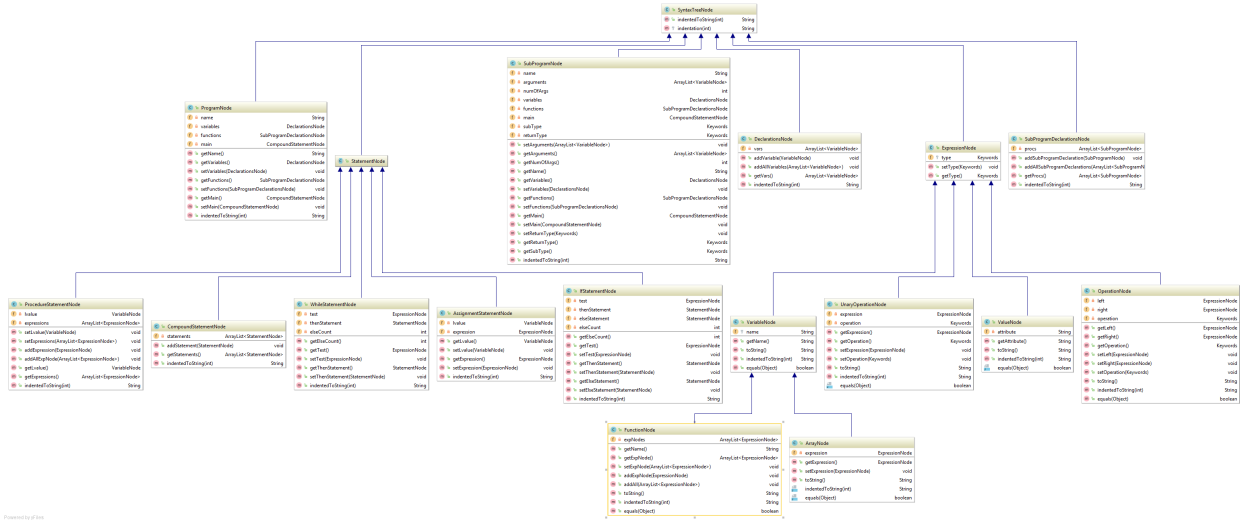

Figure 4: Syntax Tree Package Diagram

6 Semantic Analyzer

The Analyzer restructures expression nodes into their logical mathematical order and folds value nodes together as necessary.

7 Code Generator

The Generator produces MIPS assembly code based on the tree generated by the parser.

Production Rules

program -> **program id ;**
declarations
subprogram_declarations
compound_statement
.

identifier_list -> **id** |
id , *identifier_list*

declarations -> **var** *identifier_list* **:** *type* **;** *declarations* |
λ

type -> *standard_type* |
array [num : num] of *standard_type*

standard_type -> **integer** |
real

subprogram_declarations -> *subprogram_declaration* **;**
subprogram_declarations |
λ

subprogram_declaration -> *subprogram_head*
declarations
subprogram_declarations
compound_statement

subprogram_head -> **function id** *arguments* **:** *standard_type* **;** |
procedure id *arguments* **;**

arguments -> **(** *parameter_list* **)** |
λ

parameter_list -> *identifier_list* **:** *type* |
identifier_list **:** *type* **;** *parameter_list*

compound_statement -> **begin** *optional_statements* **end**

optional_statements -> *statement_list* |
λ

statement_list -> *statement* |
statement **;** *statement_list*

statement -> *variable* **assignop** *expression* |
procedure_statement |
compound_statement |
if *expression* **then** *statement* **else** *statement* |
while *expression* **do** *statement* |
read (id) |
write (expression)

variable -> **id** |
id [*expression* **]**

procedure_statement -> **id** |
id (*expression_list* **)**

expression_list -> *expression* |
expression **,** *expression_list*

expression -> *simple_expression* |
simple_expression **relop** *simple_expression*

simple_expression -> *term simple_part* |
sign term simple_part

simple_part -> **addop** *term simple_part* |
λ

term -> *factor term_part*

term_part -> **mulop** *factor term_part* |
λ

factor -> **id** |
id [*expression* **]** |
id (*expression_list* **)** |
num |
(*expression* **)** |
not *factor*

sign -> **+** |
-

Lexical Conventions

1. Comments are surrounded by **{** and **}**. They may not contain a {. Comments may appear after any token.

2. Blanks between tokens are optional.

3. Token **id** for identifiers matches a letter followed by letter or digits:
   letter -> [a-zA-Z]
   digit -> [0-9]
   id -> letter (letter | digit)*

The ***** indicates that the choice in the parentheses may be made as many times as you wish.

1. Token **num** matches numbers as follows:
   digits -> digit digit*
   optional_fraction -> . digits | λ
   optional_exponent -> (E (+ | - | λ) digits) | λ
   num -> digits optional_fraction optional_exponent

2. Keywords are reserved.

3. The relational operators (**relop**'s) are:
   =, **<>**, **<**, **<=**, **>=**, and **>**.

4. The **addop**'s are **+**, **-**, and **or**.

5. The **mulop**'s are ***, **/**, **div**, **mod**, and **and**.

6. The lexeme for token **assignop** is **:=**.

Our Mini-Pascal keywords and symbols, the definitive list.

	KEYWORDS	**SYMBOLS**
1)	and	;
2)	array	,
3)	begin	.
4)	div	:
5)	do	[
6)	else	]
7)	end	(
8)	function	)
9)	if	+
10)	integer	-
11)	mod	=
12)	not	<>
13)	of	<
14)	or	<=
15)	procedure	>
16)	program	>=
17)	real	*
18)	then	/
19)	var	:=
20)	while	