AllenCompiler

Marissa Allen

Overview:

The AllenCompiler will read the input characters from a plain text file or a String, scan the file using the rules defined in the jflex generated Scanner class of the java code, and output a sequence of tokens for the parser. Every token will consist of the lexeme, or characters in input stream, and the token type. The scanner will distinguish which characters are keywords and which are unexpected tokens. The Scanner and its classes are stored in the scanner package.

Scanner:

The scanner program is going to scan through the input file and return a lexeme. The lexeme is going to be the string containing the actual characters that were read in that make up one particular token in the language. The token type is the individual token type descriptor of the keywords, identifiers, and symbols. The keywords and symbols are listed after the design section, anything else is an identifier.

Scanners have a state machine that does pattern matching, a diagram of the state machine can be seen below. As long as the scanner sees characters that are not spaces, newline characters, or tabs, it will keep adding them to the lexeme. If the scanner does see those, and it already has a lexeme, it's going to push any spaces, newline characters, or tabs back into the input stream and return the lexeme because

it's done. If the scanner sees a space, newline characters, or tab, and it doesn't have a lexeme yet it keeps going to find a lexeme and throws the whitespace character away.

This program contains four java files – Token, TokenType, ScannerTest, and Scanner. It also contains one jflex file called AllenScanner. The AllenScanner file contains the instructions on how each Token should be handled. The Token class is a constructor class for a token object. All the tokens will consist of the lexeme, or characters in the input stream, and the token type. The TokenType class consists of all of the different types a token can be. These types are then passed into the scanner class. There is a type for every keyword and symbol as well as two types for ID and number. ScannerTest takes an input file or a string and feeds the characters into the scanner class. As long as the scanner doesn't hit a null token, the scanner prints out any tokens that are not whitespace tokens or null tokens. The Scanner file is the java file that was created when the AllenScanner file was run. The scanner for this class uses the keywords and symbols defined for the token and prints out the tokens that are not whitespace tokens or null tokens along with any errors.

Recognizer:

The beginning of the parser is created by building a recognizer implemented as a top-down recursive descent parser. The recursive descent parser will act as a recognizer, answering the yes-no question "is the input a pascal program?". A recognizer is a parser that doesn't perform any syntax-directed translation. The recognizer will only say whether the input string is from the language described by the

grammar or not. In all of the non-terminal methods listed in the grammar, it will do this by looking at a stream of tokens from the scanner in a particular order, interpreting the token types, and comparing the current lookahead token to the token type. If the current lookahead token matches all of the expected token types listed in the method and any other methods called; then the method will execute the rule for the non-terminal symbol in the micro pascal grammar.

Symbol Table:

The symbol table is used to store information about each identifier (symbol name) found in a pascal program. The symbol table is created by implementing a hash map to store an entry into the symbol table by adding a key-value pair. Entries in the symbol table contain information about an identifier such as its character string (or lexeme), its datatype, and its kind.

This program contains three java files – KindEnum, SymbolTable, and SymbolTableTest. The KindEnum file contains the different kinds for every symbol table identifier. The SymbolTable file contains methods that allow a symbol to be added to the symbol table; along with methods checking to see if it is either a program, variable, program, array, function, or procedure identifier. The SymbolTableTest file uses JUnit tests to see whether the methods from the SymbolTable file are able to add a symbol to the symbol table and verify the kind of symbol it is.

Change Log:

02/15/2019	Marissa Allen	Added the Recognizer section of the compiler.
03/9/2019	Marissa Allen	Added the Symbol Table section of the compiler.

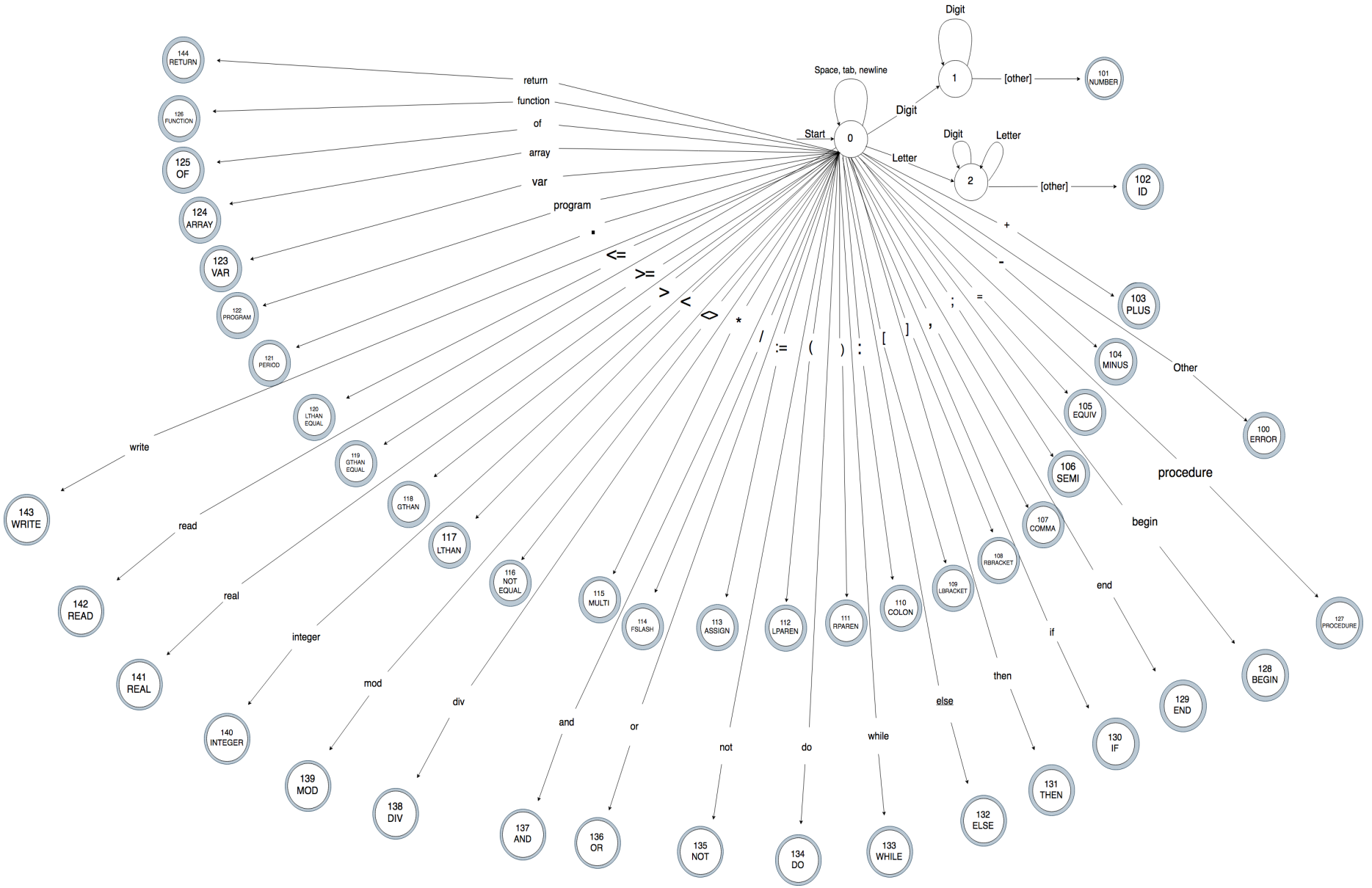
List of Keywords/Reserved Words:

program
var
array
of
function
procedure
begin
end
if
then
else
while
do
not
or
and
div
mod
integer
real
read
write
return

List of Symbols:

;

,
[
]
:
)
(
+
-
*
=
<>
<
<=
>=
>
/
:=
.


Start
0
Space, tab, newline
Digit
1
[other]
101 NUMBER
Digit
Letter
2
[other]
102 ID
Digit
Letter
return
function
of
array
var
program
.
<=
>=
>
<
<>
*
/
:=
(
)
:
[
]
,
;
=
+
-
Other
144 RETURN
126 FUNCTION
125 OF
124 ARRAY
123 VAR
122 PROGRAM
121 PERIOD
120 LTHAN EQUAL
119 GTHAN EQUAL
118 GTHAN
117 LTHAN
116 NOT EQUAL
115 MULTI
114 FSLASH
113 ASSIGN
112 LPAREN
111 RPAREN
110 COLON
109 LBRACKET
108 RBRACKET
107 COMMA
106 SEMI
105 EQUIV
104 MINUS
103 PLUS
100 ERROR
write
read
real
integer
mod
div
and
or
not
do
while
else
then
if
end
begin
procedure
143 WRITE
142 READ
141 REAL
140 INTEGER
139 MOD
138 DIV
137 AND
136 OR
135 NOT
134 DO
133 WHILE
132 ELSE
131 THEN
130 IF
129 END
128 BEGIN
127 PROCEDURE

FSM Design: Designed by Marissa Allen and Cohl Dorsey