

Kennesaw State University

College of Computing and Software Engineering

Department of Computer Science

CS 4308 Concepts of Programming Languages - Section W01

1st Project Deliverable

Bryson Phillip bphil91@students.kennesaw.edu

2/18/2022

Problem Statement:

In this three-part project, I am tasked with choosing a subset of the Ada programming language, describing its grammar, and implementing an interpreter. This interpreter should be complete with a parser and scanner, the latter of which will be described in this first section. The scanner, or lexical analyser, is responsible for identifying substrings within a text input program as various known lexemes, and then recording these lexemes as strings of tokens in an output file. The function that matches patterns can record one token at a time and detects if erroneous symbols are recognized.

Summary:

In this report, I will describe the grammar and source code for the lexical analyzer. I will show how the input files are read and how the tokens are computed. I will also describe how nonterminals in my grammar map to symbols in the Ada language and clarify some shortcomings in my scanner.

Detailed Description:

EBNF Grammar:

`<procedure> → procedure <title> is <varlist> begin <statementlist> end <title> ;`

`<title> → (a b c ... z A B C ... Z 0 1 2 3 4 5 6 7 8 9) {(a b c ... z A B C ... Z 0 1 2 3 4 5 6 7 8 9)}`

`<varlist> → <declaration> <varlist>`

`| <initialization> <varlist>`

`| ε`

`<declaration> → <title> : Integer;`

`<initialization> → <title> := <value>;`

<value> → (0 1 2 3 4 5 6 7 8 9) {(0 1 2 3 4 5 6 7 8 9)}

<statementlist> → <initialization> <statementlist>

|<conditional> <statementlist>

| ε

<conditional> → if <condition> then <statementlist> {elsif <condition> then <statementlist>} [else
<statementlist>] end if;

<condition> → <conditionB> {or <conditionB>}

<conditionB> → <comparison> {and <comparison>}

<comparison> → (<condition>)

| !(<condition>)

| <num> <operator> <num>

<num> → <title>

| <value>

<operator> → (< > <= >= /=)

Note: the ellipsis is not a symbol

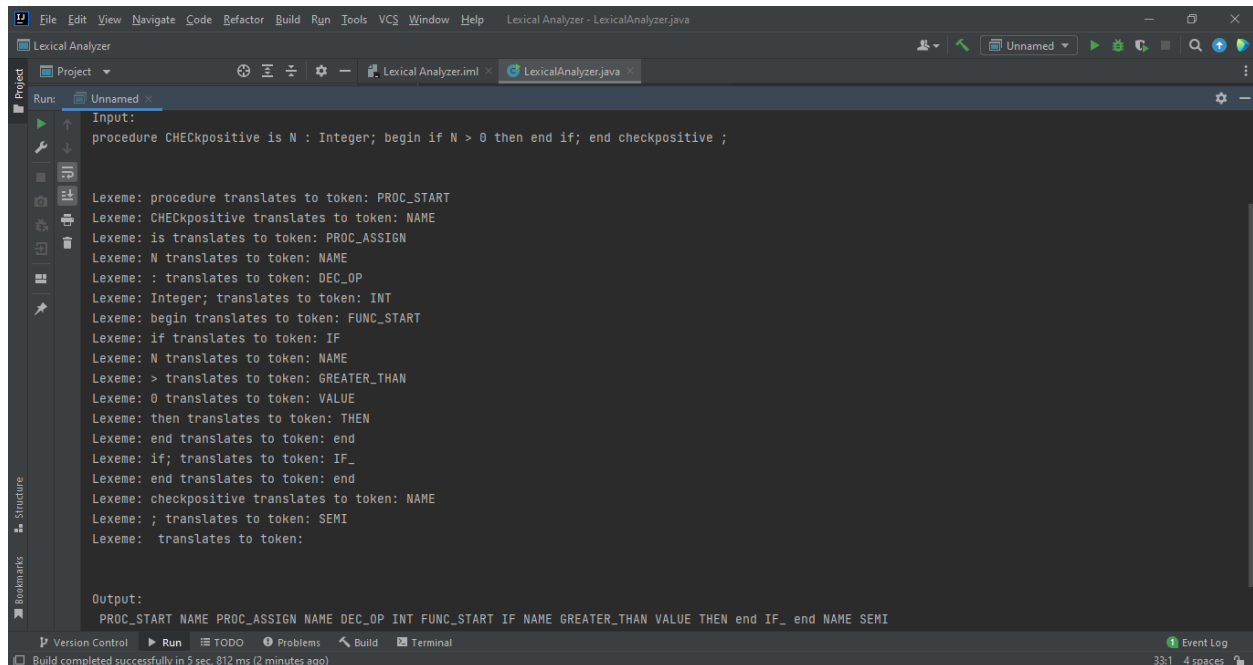
This grammar describes the rules involving <procedure> which represents a most basic function within Ada. Within a function, start and end statements can be made, integers can be declared and initialized, if-else statements can be nested, and complex conditional statements can be made. Integer variables can be declared as names which may be a string of letter or digit characters, and they may contain integers which must be a string of digits. <statementlist> represents a series of variable initializations and conditionals which make up the body of a function. It is defined recursively to have any possible length. The conditional if-else statements are described to have nested statement lists. The condition inside the if-else statement has multiple non-terminals to ensure proper parse trees are generated with correct operator precedence.

Restrictions:

- Variables must be integers
- Variables cannot be declared and initialized in the same line
- Names are simplified
- Complex expressions cannot be assigned to variables or used in conditionals
- Functions are basic and do not include arguments
- IO is not implemented
- All of other functionalities are not present

Source Code Description:

In the main function of LexicalAnalyzer, a scanner is created to read input from a file stream. The file path is specified by the user upon running the scanner. The contents of the file are stored in a string variable named 'input', and formatted to convert all whitespace to a single type of character. The function 'tokenize' reads the next lexeme from the input string and updates values for the position of the cursors 'curr_char' and 'lex_count'. It skips whitespace and records the current lexeme into its own variable, where it is either matched to a predetermined substring or determined to be a unique name or number. Two arrays are used to match a lexeme to a corresponding token, where it is appended to the initially empty 'output' string. If a substring is not recognized, an error message is displayed and an error code is put in its place within the 'output' string. 'Tokenize' runs until the cursor is at the end, and the program prints 'output'.



```
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help Lexical Analyzer - LexicalAnalyzer.java
Lexical Analyzer
Project
Run: Unnamed
Input:
procedure CHECKpositive is N : Integer; begin if N > 0 then end if; end checkpositive ;

Lexeme: procedure translates to token: PROC_START
Lexeme: CHECKpositive translates to token: NAME
Lexeme: is translates to token: PROC_ASSIGN
Lexeme: N translates to token: NAME
Lexeme: : translates to token: DEC_OP
Lexeme: Integer; translates to token: INT
Lexeme: begin translates to token: FUNC_START
Lexeme: if translates to token: IF
Lexeme: N translates to token: NAME
Lexeme: > translates to token: GREATER_THAN
Lexeme: 0 translates to token: VALUE
Lexeme: then translates to token: THEN
Lexeme: end translates to token: end
Lexeme: if; translates to token: IF_
Lexeme: end translates to token: end
Lexeme: checkpositive translates to token: NAME
Lexeme: ; translates to token: SEMI
Lexeme: translates to token:

Output:
PROC_START NAME PROC_ASSIGN NAME DEC_OP INT FUNC_START IF NAME GREATER_THAN VALUE THEN end IF_ end NAME SEMI

Version Control Run TODO Problems Build Terminal
Build completed successfully in 5 sec, 812 ms (2 minutes ago) 33:1 4 spaces
```

This illustrates the output of the code after scanning the input.txt file which will be included in the archive.

Conclusion:

In conclusion, this grammar represents a simplified version of a function within Ada. The terminal symbols can be recognized and mapped to tokens with the LexicalAnalyzer class.

References:

Textbook: Concepts of Programming Languages 11th Edition by Robert W. Sebesta

Website: <https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html> (For resources on the next() function)

Website: <https://www.geeksforgeeks.org/difference-between-ll-and-lr-parser/> (For resources on LL grammars)

