CS 315 PROJECT ONYXIA

LEXICAL ANALYSIS AN IMPLEMENTATION OF PARSER

Arda Gültekin , ID: 21401142, Section 02

Çağatay Küpeli, ID: 21402290, Section 02

Koray Gürses, ID: 21401254, Section 02

Complete BNF Description:

Program

<program> ::= <statement list> EOF

Truth Values

<true> ::= TRUE

<false> ::= FALSE

| "

<truth values> ::= <false>

| <true>

| <connective expression>

| <relation expression>

Constants

<char> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

<string> ::= <char>

| <char> <string>

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<integer> ::= <digit>

| <digit> <integer>

<constant Identifier> ::= <string>

<constant Name> ::= <string> | <integer>

<constant> ::= <constant Identifier> <constant Name>

Connectives

<connective expression> ::= <variable> <connective sign> <variable>

| <variable> <connective sign> <relation expression>

| <relation expression> <connective sign> <variable>

| <relation expression> <connective sign> <relation expression>

<connective sign> :== 'and' | 'or' | '->' | '~'

Relations

<relation> ::= '<'

| '>'

| '=='

<relation expression> ::= <integer> <relation> <integer>

Variable

<identifier> ::= <string>

<variable name> ::= <constants>

<variable> ::= <identifier> <variable name>

<label statement> ::= <variable> | <constant>

Predicates

<lp> ::= '('

<rp> ::= ')'

<lbrace> ::= '{'

<rbrace> ::= '}'

<caller Identifier> ::= <string>

<caller Name> ::= <string>

<caller> ::= <caller Identifier> <caller Name>

<return statement> ::= RETURN

| RETURN <lp> <truth values> <rp>

<predicates> ::= <caller> <lp> <rp>

| <caller> <lp> <rp> <lbrace> <statement list> <rbrace>

Predicates Instantiations

<comma> ::= ','

<parameter> ::= <truth value> | <variable>

<parameter list> ::= <parameter>

| <parameter> <comma> <parameter list>

<predicates instantiations> ::= <caller> <lp> <parameter list> <rp> <lbrace> <statement list> <rbrace>

Assignment

<equal sign> ::= '='

<assignment statement> ::= <variable> <equal sign> <truth values>

Selection

<expression> ::= <connective expression>

| <relation expression>

| <variable>

| <constant>

| <truth values>

<if clause> ::= IF <lp> <expression> <rp> THEN

<if statement> ::= <if clause> <lbrace> <statement list> <rbrace>
| <if clause> <lbrace> <statement list> <rbrace> ELSE <lbrace> <statement list> <rbrace>
| <if clause> <lbrace> <statement list> <rbrace> ELSE <if statement>

Looping statements

<while clause> ::= WHILE <lp> <expression> <rp>

<while statement> ::= <while clause> <lbrace> <statement lis> <rbrace>

Input Statement

<input> ::= <truth values>

<input list> ::= <input>
| <input> <comma> <input list>

<input statement> ::= INPUT <lp> <input list> <rp>

Output Statement

<output> ::= <truth values>

<output statement> ::= OUTPUT <lp> <output> <rp>

Statement

<statement> ::= <truth value>
| <if statement>
| <while statement>
| <assignment statement>
| <return statement>
| <label statement>
| <input statement>
| "

<statement list> ::= <statement>

| <statement> <statement list>

Explanation of Language Constructs

- We' ve defined true and false tokens. Also expressions, strings and integers can be truth values.
- <char> is defined both in lower and upper case. <string> is consist of chars.
- <digit> is defined by numbers from zero to nine. <integer> is consist of digits.
- A <constant> is a <constant Identifier> and <constant Name>.
- <constant Identifier> is a string and <constant Name> can be cluster of strings or integers. Which are used by defining <constant>.
- A constant can be an identifier followed by a constant name.
- Connective expressions are basically expressions using connective signs such as "and, or,implies or negation".
- In <relation> we've defined the relation signs. We used <relation> and <integer> while defining the <relation expression>.
- While defining the <relation expression> we used <integer> <relation> and <integer> format (Such as: 5 < 3).
- <identifier> is a string and <variable name> can be a cluster of constants. Which are used while defining <variable>. A variable can be a variable name or it can be an identifier followed by a variable name(Such as int a).
- A <label statement> can be <variable> or <constant>.
- We've defined different kinds of parenthesis at the beginning of predicates part.
- <caller> is the name of a function, written in onyxia language.
- We defined a <return statement> which can return an empty space or truth values.
- <predicates> can call a empty function or a function with <statement list>
- We also defined a comma as <comma>.

- We defined <parameter> which can be a truth value or a variable. Then we use this on <parameter list> to fill the list.
- <predicates instantiations> represents a function with multiple inputs.
- We defined an <equal sign> for further usage on <assignment statement>.
- We use equal sign to assign a variable to a truth value.
- An <expression> can be a connective expression or a relation expression or a variable, or a constant, or a truth value.
- When we defined “if”, we have used <if clause> and <if statement>.
- <if clause> shows that between the if parentheses, there is a expression.
- <if clause> and <statement list> are used to define the <if statement>.
- We also defined else if and its statements.
- While has defined in a very similar way to if.
- <while clause> is used to show how the while statement will be defined.
- We return truth values after our while statements.
- <input> can be string or integer and used to create the <input list>, allocated with commas.
- <input statement> is defined with the input token and <input list>.
- <output> can be true or false or other truth values.
- <output statement> is defined with the output token and <output>.
- A statement can be a truth value or an if/while statement or assignment statements or return statements or label statement or input statement. It can also be null.
- <statement list> is consist of statements.

Description of How Nontrivial Tokens Defined:

- Our functions have “func” before them. We did this so our lex code can work with our BNF.
- Also, our reserved words are “boolean,cons,func” which are used as identifiers.
- We used “ “ as a part of our language to concretize the difference between a string and a variable. For example,” boolean a” is a variable while “booleana” is a string.
- We used “while, if, return” as literals.

- Our motivation was to create a easily readable and trackable language as much as possible.

Lex Description File:

%Option main

true (true)

false (false)

truthValues ({true}|{false})

integer [0-9]+

char [A-Za-z]

string [A-Za-z]+

varIdentifier (boolean)

constantIdentifier (cons)

constantName ({string})

constant ({constantIdentifier}" "*{constantName})

variableName ({string})

variable ({varIdentifier}" "*{variableName})

labelStmt ({variable}|{constant})

relation (<|>|==)

relationExp ({integer}" "*{relation}" "*{integer})

connective (and|or|->|~)

connectiveExp (({variable}" "*{connective}" "*{variable})|({variable}" "*{connective}" "*{truthValues})|({truthValues}" "*{connective}" "*{variable})|({truthValues}" "*{connective}" "*{tru\

thValues}))

```
expression ({connectiveExp}|{relationExp}|{variable}|{constant})

stmt ({labelStmt}|{returnStmt}|{assignmentStmt}|{outputStmt}|{ifStmt}|{whileStmt}|{inputStmt})

lp "("

rp ")"

lbrace "{"

rbrace "}"

if (if)

else (else)

ifClause ({if}" "*{lp}" "*{expression}" "*{rp})

ifStmt (" "{else}?({ifClause}" "*{lbrace}{stmt}*{rbrace})+)

return (return)

returnStmt ({return}" "*({lp}" "*{truthValues}" "*{rp})?)

comma ","

parameter ({truthValues}|{variable})

parameterList ({comma}" "*{parameter})

parameters ({parameter}" "*{parameterList}*)

callIdentifier (func)

callerName ({string})

caller ({callIdentifier}" "{callerName})

predicate ({caller}" "*{lp}" "*{parameters}*" "*{rp}({lbrace}{stmt}+{rbrace})?)

assignmentSign (=)

assignmentStmt ({variable}" "*{assignmentSign}" "*({truthValues}))

while (while)

whileClause ({while}" "*{lp}" "*{expression}" "*{rp})

whileStmt ({whileClause}{lbrace}" "*{stmt}*" "*{rbrace})

output (output)

outputs ({truthValues})
```

```
input (input)

inputs ({truthValues})

inputStmt ({input}" "*{lp}" "*{inputs}" "*{rp})

program ({stmt}*}

%%

{input} printf("INPUT");

{caller} printf("CALLER");

{output} printf("OUTPUT");

{while} printf("WHILE");

{assignmentSign} printf("ASSIGNMENT SIGN");

{return} printf("RETURN");

{true} printf("TRUE");

{false} printf("FALSE");

{if} printf("IF");

{else} printf("ELSE");

{connectiveExp} printf("CONNECTIVE EXPRESSION");

{connective} printf("CONNECTIVE");

{relationExp} printf("RELATION EXPRESSION");

{variable} printf("VARIABLE");

{constant} printf("CONSTANT");

{relation} printf("RELATION");

{lp} printf("LP");

{rp} printf("RP");

{lbrace} printf("LBRACE");

{rbrace} printf("RBRACE");

{comma}  printf("COMMA");
```

Example Program:

true

false

9

14

a

ab

dsad

ASdsa

asd9asd

boolean a

boolean asd

boolean 9

cons 14

cons abc

<

>

==

>>

5 < 6

boolean a > 83

boolean a == boolean c

boolean a and int b

boolean a or boolean c

boolean a -> boolean dsa

~true

```
}

{

(

)

if (  boolean a   ){ boolean a = true    }else if( 6 < 7 ){ }  else { return    }

func a()

func sdasa(){    return(true)        }

func bfdr(boolean a,  asad, 121,   true)

boolean a = 7

boolean b = sdada

boolean dsa = true

while   ( 5 < 21      ){     return }

output(     true    )

output   (  false)

input (  dsads)

input( 2312132)

input(false)
```