Gramática

Íris Viana dos S. Santana, Raquel da Silva Vieira
Bacharelado Em Ciência da Computação
Universidade Federal do Agreste de Pernambuco
{iris.viana,raquel.silvavieira}@ufrpe.br

***Resumo*— Será apresentado a gramática como parte do projeto desenvolvido na disciplina de compiladores, ministrada pela professora Maria Aparecida Amorim.**

PALAVRAS CHAVE
Compiladores, Gramática, Análise

1. Inicialização

Variável inicial : <main>::= main {<statement> }

2. Blocos

<statement> ::= <declaration_var> | <declaration_func> | <declaration_proc> | <call_func> | <call_proc> | <print_statement> | <if_statement> | <while_statement> | <empty>

statement2 é o bloco que contém break/continue, só pode ser chamado dentro de um while e não pode declarar função e procedimento
<statement2> ::= <declaration_var> | <call_proc> | <call_func> | <if_statement2> | <while_statement> | <unconditional_branch> | <print_statement>

statement3 é o bloco do if/else, não pode declarar função e procedimento dentro
<statement3> ::= <declaration_var> | <call_proc> | <call_func> | <if_statement> | <while_statement> | <print_statement>

3. Condicional

<if_statement> ::= if(<expression>){<statement3>} endif <else_part>

<else_part> ::= else { <statement3> } endelse | <empty>
<while_statement> ::= while(<expression>){<statement2>}endwhile

IF chamado somente dentro do while, pois dentro dele pode ter BREAK E CONTINUE (statement2)
<if_statement2> ::= if(<expression>){<statement2>} endif <else_part2>

ELSE chamado somente dentro do while, pois dentro dele pode ter BREAK E CONTINUE (statement2)
<else_part2> ::= else { <statement2> } endelse | <empty>

4. Incondicional

somente pode está dentro de um WHILE, e no IF/ELSE dentro do WHILE
<unconditional_branch> ::= continue ; | break ;

5. Retorno

<return_statement> ::= return <return_statement2> ;

<return_statement2> ::= <call_func> | <identifier> | <num> | <boolean>

6. Print

<print_statement> ::= print (<params_print>) ;

<params_print> ::= <identifier> | <call_func> | <call_op> | <boolean> | <num>

7. Declarações

1. Dados

<letter> ::= “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”

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

<num> ::= <digit> {<digit>}*

<boolean> ::= "True" | "False"

<type> ::= "int" | "bool"

<identifier> ::= <letter> (<letter> | <num>)*

<op_boolean> ::= { == | != | > | < | >= | <= }

<op_aritmetic> ::= {+ | - | * | / }

2. Variáveis

<declaration_var> ::= <type> <identifier> = <end_var>;

<end_var> ::= = <call_func> | <boolean> | <num> | <identifier> | <call_op>

3. Parâmetros

<params> ::= <type> <identifier> (, <params>)*

parametros para chamada de função pois não precisa passar o type
<params_call> ::= <identifier> (, <params_call>)*

4. Função

<declaration_func> ::= func <type> <identifier> (<params>) { <statement>

<return_statement>};

<call_func> ::= call func <identifier> (<params_call>);

5. Procedimento

<declaration_proc> ::= proc <identifier> (<params>) { <statement> };

<call_proc> ::= call proc <identifier> (<params_call>);

6. Operações

<call_op> ::= <num> <op_aritmetic> <call_op2> | <identifier> <op_aritmetic>

<call_op2> ::= <num> <call_op4> | <identifier> <call_op4>

<call_op3> ::= <call_op2> | <num>

<call_op4> ::= (<op_aritmetic> <call_op3>)* | <empty>

7. Expressão

<expression> ::= <identifier> <expression2> | <num> <expression2>

<expression2> ::= <op_boolean> <expression3>

<expression3> ::= <identifier> | <num>