UNIVERSIDADE FEDERAL DO AGRESTE DE PERNAMBUCO
COMPILADORES - PROFª MARIA SIBALDO
ALUNOS: VALDIR DIAS DA SILVA JÚNIOR
LUIZ DAVI MENDES DE MATOS

GRAMÁTICA - BNF

 # ---------- 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"

<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> ::= + | - | * | /

 # ---------- Variáveis ----------

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

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

 # ---------- Parâmetros ----------

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

parametros para chamada de função pois não precisa passar o type

<params_call> ::= <identifier> (, <params_call> | <empty>)*

 # ---------- Função ----------

<declaration_func> ::= func <type> <identifier> (<params>) { <block> <return_statement>};

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

 # ---------- Procedimento ----------

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

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

 # ---------- Operações ----------

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

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

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

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

 # ---------- Start ----------

<program> ::= program { <block> } end

 # ---------- Expression ----------

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

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

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

 # ---------- Condicional ----------

<if_statement> ::= if(<expression>){<block3>} endif <else_part>

<else_part> ::= else { <block3> } endelse | <empty>

<while_statement> ::= while(<expression>){<block2>}endwhile

IF chamado somente dentro do while, pois dentro dele pode ter BREAK E CONTINUE (block2)

<if_statement2> ::= if(<expression>){<block2>} endif <else_part2>

ELSE chamado somente dentro do while, pois dentro dele pode ter BREAK E CONTINUE (block2)

<else_part2> ::= else { <block2> } endelse | <empty>

 # ---------- Incondicional ----------

somente pode está dentro de um WHILE, e no IF/ELSE dentro do WHILE

<unconditional_branch> ::= continue ; | break ;

 # ---------- Return ----------

<return_statement> ::= return <return_statement2> ;

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

 # ---------- Print ----------

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

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

 # ---------- Blocos ----------

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

block2 é o bloco que contém break/continue que só pode ser chamado dentro de um while e não pode declarar função e procedimento dentro

<block2> ::= <declaration_var> | <call_proc> | <call_func> | <identifier> | <if_statement2> | <while_statement> | <unconditional_branch> | <print_statement>

block3 é o bloco do if/else, que não pode declarar função e procedimento dentro

<block3> ::= <declaration_var> | <call_proc> | <call_func> | <identifier> | <if_statement> | <while_statement> | <print_statement>