# Tokens Utilizados

## Palavras Reservadas

|  |  |
| --- | --- |
| void | PR\_VOID |
| int | PR\_INT |
| real | PR\_REAL |
| char | PR\_CHAR |
| bool | PR\_BOOL |
| if | PR\_IF |
| then | PR\_THEN |
| else | PR\_ELSE |
| end-if | PR\_END\_IF |
| for | PR\_FOR |
| while | PR\_WHILE |
| do | PR\_DO |
| return | PR\_RETURN |
| break | PR\_BREAK |
| continue | PR\_CONTINUE |
| goto | PR\_GOTO |
| true | PR\_TRUE |
| false | PR\_FALSE |
| var | PR\_VAR |
| main | PR\_MAIN |
| scan | PR\_SCAN |
| scanln | PR\_SCANLN |
| print | PR\_PRINT |
| println | PR\_PRINTLN |

## Operadores

|  |  |
| --- | --- |
| + | OP\_ADICAO |
| - | OP\_SUBTRACAO |
| \* | OP\_MULTIPLICACAO |
| / | OP\_DIVISAO |
| % | OP\_MODULO |
| ? | OP\_TERNARIO |
| : | OP\_DOISPONTOS |
| ! | OP\_NEGACAO |
| & | OP\_ENDERECO |
| . | OP\_PONTO |
| -> | OP\_FLECHA |
| < | OP\_MENOR |
| > | OP\_MAIOR |
| == | OP\_IGUALDADE |
| != | OP\_DIFERENCA |
| <= | OP\_MENORIGUAL |
| >= | OP\_MAIORIGUAL |
| = | OP\_IGUAL |
| += | OP\_ADICAOIGUAL |
| -= | OP\_SUBTRACAOIGUAL |
| \*= | OP\_MULTIPLICACAOIGUAL |
| /= | OP\_DIVISAOIGUAL |
| %= | OP\_MODULOIGUAL |
| ++ | OP\_INCREMENTO |
| && | OP\_AND |
| || | OP\_OR |

## Sinais de Pontuação

|  |  |
| --- | --- |
| , | SP\_VIRGULA |
| ; | SP\_PONTOEVIRGULA |
| ( | SP\_ABREPARENTESES |
| ) | SP\_FECHAPARENTESES |
| [ | SP\_ABRECOLCHETES |
| ] | SP\_FECHACOLCHETES |
| { | SP\_ABRECHAVES |
| } | SP\_FECHACHAVES |

## Literais Básicos

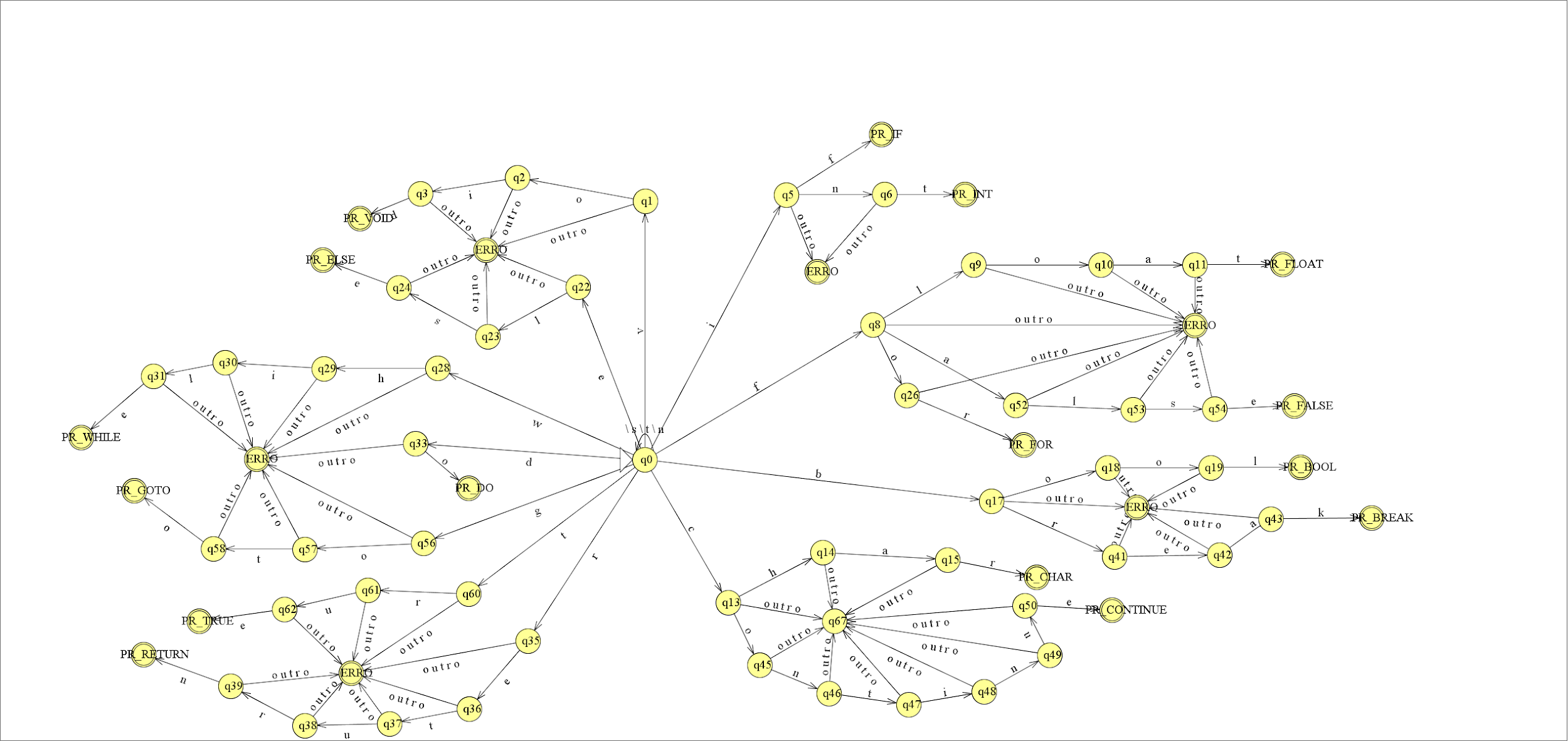
|  |  |
| --- | --- |
| inteiros | LB\_INT |
| reais | LB\_FLOAT |
| caracteres | LB\_CHAR |
| strings | LB\_STRING |
| booleanos | LB\_BOOL |

## Identificadores

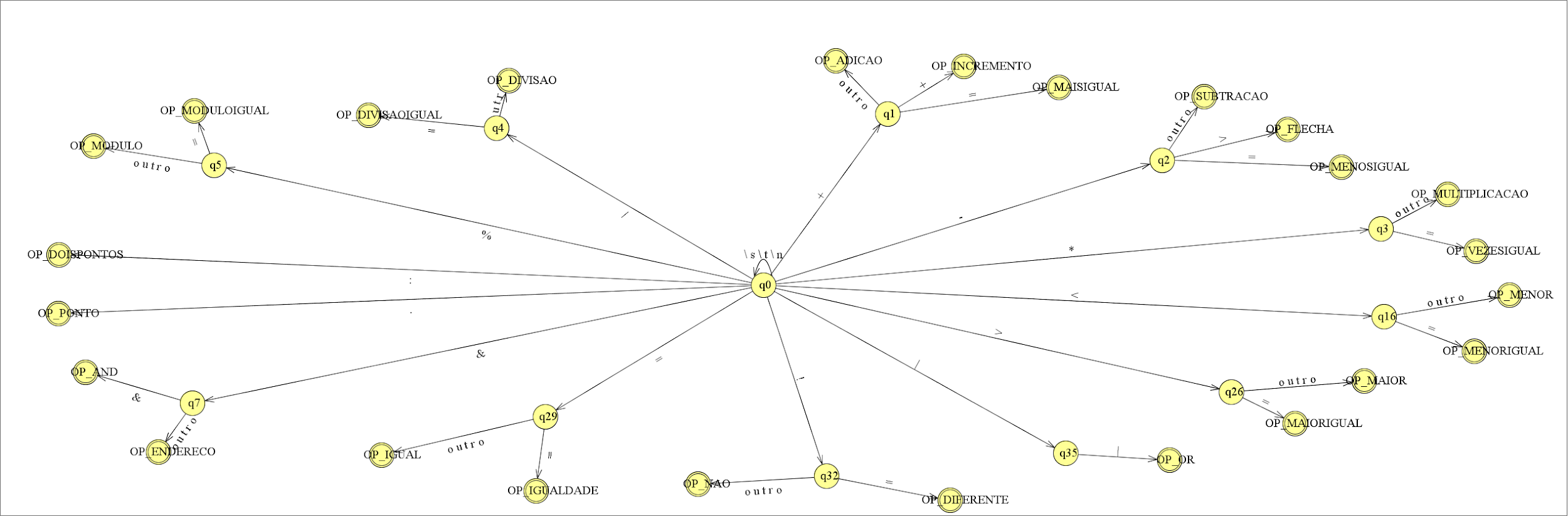
|  |  |
| --- | --- |
| identificadores | ID |

# Diagramas de Transição

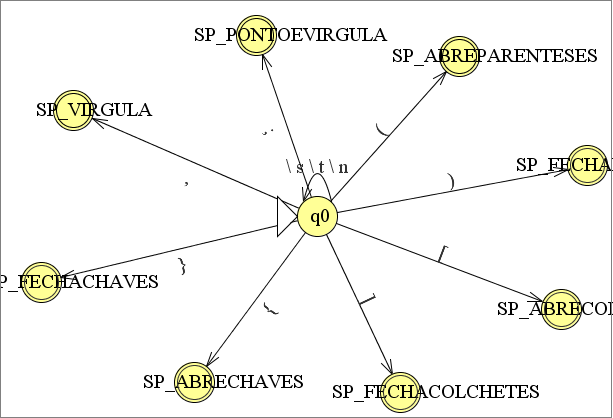
## Palavras Reservadas



## Operadores

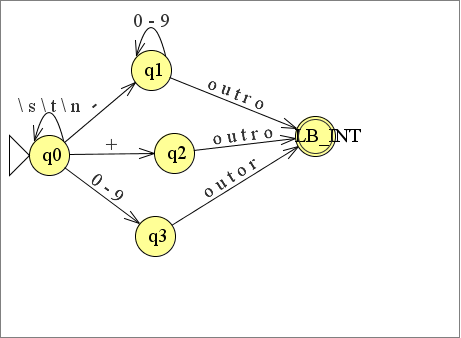


## Sinais de Pontuação

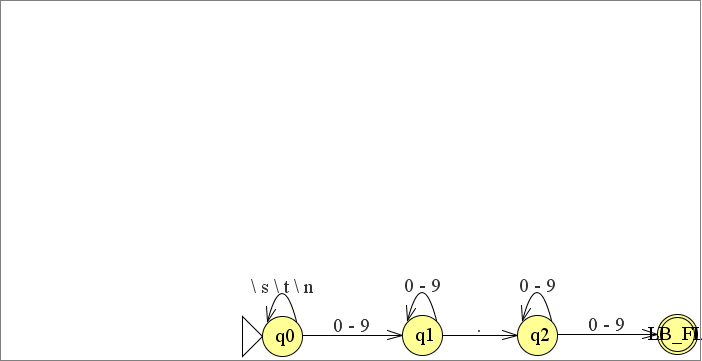


## Literais Básicos

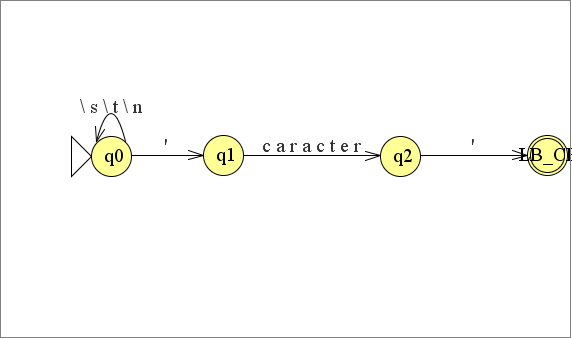
### Inteiros



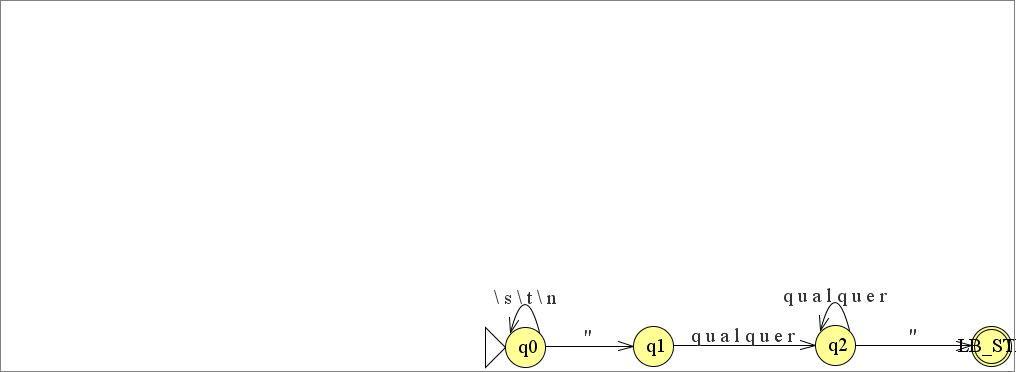
### Reais



### Caracteres



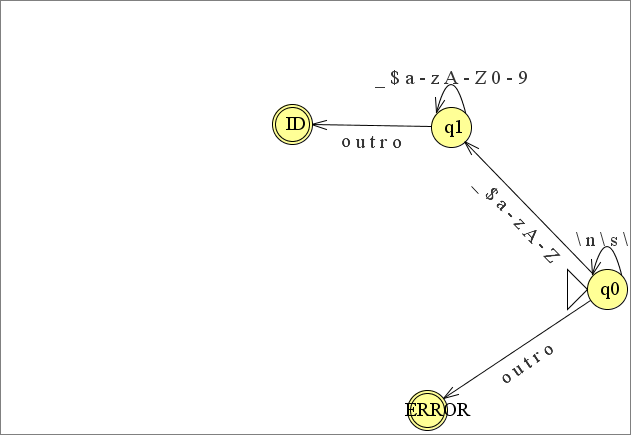
### Strings



### Booleanos



## Identificadores



# Técnicas utilizadas na gramática

## Gramática original

programa → lista-decl

lista-decl → decl lista-decl | decl

decl → decl-var | decl-main

decl-var → VAR espec-tipo var ;

decl-main → MAIN ( ) bloco END

espec-tipo → INT | REAL | CHAR

bloco → lista-com

lista-com → comando lista-com | ɛ

comando → decl-var | com-atrib | com-selecao | com-repeticao | com-leitura | com-escrita

com-atrib → var = exp ;

com-leitura → SCAN ( var ) ; | SCANLN ( var ) ;

com-escrita → PRINT ( exp ) ; | PRINTLN ( exp ) ;

com-selecao → IF exp THEN bloco END-IF | IF exp THEN bloco ELSE bloco END-IF

com-repeticao → WHILE exp DO bloco LOOP

exp → exp-soma op-relac exp-soma | exp-soma

op-relac → <= | < | > | >= | == | <>

exp-soma → exp-mult op-soma exp-soma | exp-multi

op-soma → + | -

exp-mult → exp-simples op-mult exp-mult | exp-simples

op-mult → \* | / | DIV | MOD

exp-simples → ( exp ) | var | literal

literal → NUMINT | NUMREAL | CARACTERE | STRING

var → ID

## Fatoração

programa → lista-decl

// lista-decl → decl lista-decl | decl

lista-decl → decl lista-decl’

lista-decl’ → decl | ɛ

decl → decl-var | decl-main

decl-var → VAR espec-tipo var ;

decl-main → MAIN ( ) bloco END

espec-tipo → INT | REAL | CHAR

bloco → lista-com

lista-com → comando lista-com | ɛ

comando → decl-var | com-atrib | com-selecao | com-repeticao | com-leitura | com-escrita

com-atrib → var = exp ;

com-leitura → SCAN ( var ) ; | SCANLN ( var ) ;

com-escrita → PRINT ( exp ) ; | PRINTLN ( exp ) ;

// com-selecao → IF exp THEN bloco END-IF | IF exp THEN bloco ELSE bloco END-IF

com-selecao → IF exp THEN bloco com-selecao’

com-selecao’ → END-IF | ELSE bloco END-IF

com-repeticao → WHILE exp DO bloco LOOP

// exp → exp-soma op-relac exp-soma | exp-soma

exp → exp-soma exp’

exp’ → op-relac exp-soma | ɛ

op-relac → <= | < | > | >= | == | <>

// exp-soma → exp-mult op-soma exp-soma | exp-multi

exp-soma → exp-mult exp-soma’

exp-soma’ → op-soma exp-soma | ɛ

op-soma → + | -

// exp-mult → exp-simples op-mult exp-mult | exp-simples

exp-mult → exp-simples exp-mult’

exp-mult’ → op-mult exp-mult | ɛ

op-mult → \* | / | DIV | MOD

exp-simples → ( exp ) | var | literal

literal → NUMINT | NUMREAL | CARACTERE | STRING

var → ID

## Análise do primeiro símbolo

|  |  |  |  |
| --- | --- | --- | --- |
| **APS** | **P** | **P+** | **t** |
| programa | lista-decl | Prim(lista-decl) | VAR, MAIN() |
| lista-decl | decl | Prim(decl) | VAR, MAIN() |
| lista-decl’ | decl, ɛ | Prim(decl), Segue(lista-decl’) -> Segue(lista-decl) -> Segue(programa) -> ⦰ | VAR, MAIN() |
| decl | decl-var, decl-main | Prim(decl-var), Prim(decl-main) | VAR, MAIN() |
| decl-var | VAR | VAR | VAR |
| decl-main | MAIN() | MAIN() | MAIN() |
| espec-tipo | INT, REAL, CHAR | INT, REAL, CHAR | INT, REAL, CHAR |
| bloco | lista-com | Prim(lista-com) | VAR, ID, IF, WHILE, SCAN (, SCANL (, PRINT (, PRINTLN (, END |
| lista-com | comando, ɛ | Prim(comando), Segue(lista-com) -> Segue(bloco) -> END | VAR, ID, IF, WHILE, SCAN (, SCANL (, PRINT (, PRINTLN (, END |
| comando | decl-var, com-atrib, com-selecao, com-repeticao, com-leitura, com-escrita | Prim(decl-var), Prim(com-atrib), Prim(com-selecao), Prim(com-repeticao), Prim(com-leitura), Prim(com-escrita), | VAR, ID, IF, WHILE, SCAN (, SCANL (, PRINT (, PRINTLN ( |
| com-atrib | var | Prim(var) | ID |
| com-leitura | SCAN (, SCANLN ( | SCAN (, SCANLN ( | SCAN (, SCANLN ( |
| com-escrita | PRINT (, PRINTLN ( | PRINT (, PRINTLN ( | PRINT (, PRINTLN ( |
| com-selecao | IF | IF | IF |
| com-selecao’ | END-IF, ELSE | END-IF, ELSE | END-IF, ELSE |
| com-repeticao | WHILE | WHILE | WHILE |
| exp | exp-soma | Prim(exp-soma) | (, NUMINT, NUMREAL, CARACTER, STRING, ID |
| exp’ | op-relac, ɛ | Prim(op-relac), Segue(exp’) -> Segue(exp) -> ); | <=, <, >, >=, ==, <>, ); |
| op-relac | <=, <, >, >=, ==, <> | <=, <, >, >=, ==, <> | <=, <, >, >=, ==, <> |
| exp-soma | exp-mult | Prim(exp-mult) | (, NUMINT, NUMREAL, CARACTER, STRING, ID |
| exp-soma’ | op-soma, ɛ | Prim(op-soma), Segue(exp-soma’) -> Segue(exp-soma) -> Segue(exp’) -> Segue(exp) -> ); | +, -, ); |
| op-soma | +, - | +, - | +, - |
| exp-mult | exp-simples | Prim(exp-simples) | (, NUMINT, NUMREAL, CARACTER, STRING, ID |
| exp-mult’ | op-mult, ɛ | Prim(op-mult), Segue(exp-mult’) -> Segue(exp-mult) -> Prim(exp-soma’) | \*, /, DIV, MOD, +, -, ); |
| op-mult | \*, /, DIV, MOD | \*, /, DIV, MOD | \*, /, DIV, MOD |
| exp-simples | (, var, literal | (, Prim(var), Prim(literal) | (, NUMINT, NUMREAL, CARACTER, STRING, ID |
| literal | NUMINT, NUMREAL, CARACTER, STRING | NUMINT, NUMREAL, CARACTER, STRING | NUMINT, NUMREAL, CARACTER, STRING |
| var | ID | ID | ID |

## Tabela de derivação

### Declarações

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TD** | **VAR** | **MAIN ()** | **INT** | **REAL** | **CHAR** |
| programa | lista-decl | lista-decl | - | - | - |
| lista-decl | decl | decl | - | - | - |
| decl | decl-var | decl-main | - | - | - |
| decl-var | VAR | - | - | - | - |
| decl-main | - | MAIN () | - | - | - |
| espec-tipo | - | - | INT | REAL | CHAR |

### Comandos

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TD | SCAN ( | SCANLN ( | PRINT ( | PRINTLN ( | IF | WHILE | VAR | ID | END |
| bloco | lista-com | lista-com | lista-com | lista-com | lista-com | lista-com | lista-com | lista-com | lista-com |
| lista-com | comando | comando | comando | comando | comando | comando | comando | comando | comando |
| comando | com-leitura | com-leitura | com-escrita | com-escrita | com-selecao | com-repeticao | com-atrib | - | - |
| com-atrib | - | - | - | - | - | - | var | - | - |
| com-leitura | SCAN ( | SCANLN ( | - | - | - | - | - | - | - |
| com-escrita | - | - | PRINT ( | PRINTLN ( | - | - | - | - | - |
| com-selecao | - | - | - | - | IF | - | - | - | - |
| com-repeticao | - | - | - | - | - | WHILE | - | - | - |
| var | - | - | - | - | - | - | - | ID | - |

### Expressões

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TD | <= | < | > | >= | == | <> | + | - | \* | / | DIV | MOD | NUMINT | NUMREAL | CARACTERE | STRING | ID | ( | ); |
| exp | - | - | - | - | - | - | - | - | - | - | - | - | exp-soma | exp-soma | exp-soma | exp-soma | exp-soma | exp-soma | - |
| op-relac | <= | < | > | >= | == | <> | - | - | - | - | - | - | - | - | - | - | - | - | - |
| exp-soma | - | - | - | - | - | - | - | - | - | - | - | - | exp-mult | exp-mult | exp-mult | exp-mult | exp-mult | exp-mult | - |
| op-soma | - | - | - | - | - | - | + | - | - | - | - | - | - | - | - | - | - | - | - |
| exp-mult | - | - | - | - | - | - | - | - | - | - | - | - | exp-simples | exp-simples | exp-simples | exp-simples | exp-simples | exp-simples | - |
| op-mult | - | - | - | - | - | - | - | - | \* | / | DIV | MOD | - | - | - | - | - | - | - |
| exp-simples | - | - | - | - | - | - | - | - | - | - | - | - | literal | literal | literal | literal | var | ( | - |
| literal | - | - | - | - | - | - | - | - | - | - | - | - | NUMINT | NUMERAL | CARACTERE | STRING | - | - | - |
| var | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | ID | - | - |

# Mensagens de Erro

## Léxico

OBS: %s corresponde ao lexema que ocasionou o erro.

|  |  |
| --- | --- |
| **Mensagem** | **Descrição** |
| 'ERRO', 'Número real inválido: %s' | O número real (float) é inválido. |
| 'ERRO', 'Identificador inválido: %s' | O identificador de variável é inválido. |
| 'ERRO', 'String incompleta: %s' | A string está incompleta (falou um “). |
| 'ERRO', 'Char inválido: %s' | Caractere inválido. Esse erro pode ocorrer quando se tem mais de um caracter dentro de aspas simples (‘aa’). Strings usam aspas duplas. |
| 'ERRO', 'Operador inválido: %s' | Qualquer operador ou caractere que não seja suportado pela linguagem. |

## Sintático

OBS: %s corresponde ao token que ocasionou o erro.

|  |  |
| --- | --- |
| **Mensagem** | **Descrição** |
| 'Esperado identificador. Recebido %s' | O analisador esperava receber um identificador de variável. Identificadores válidos podem começar com \_ ou uma letra, e podem ser compostos por letras, números, \_ e $. |
| 'Esperado um literal (int, float, char ou string). Recebido %s' | O analisador esperava receber um literal básico (int, float, char ou string). Exemplo: 10 para int, 15.3 para float, ‘g’ para char, “teste” para string. |
| 'Esperado ). Recebido %s' | O analisador esperava receber um ) depois de uma expressão. |
| 'Esperado (exp), identificador ou literal. Recebido %s' | O analisador esperava receber uma expressão (entre parênteses), um identificador de variável ou um literal. Exemplo: (a+b), i, 10. |
| 'Esperado um dos operadores: \*, \*=, /, /=, %, %=. Recebido %s' | O analisador esperava receber um dos operadores descritos. |
| 'Esperado um dos operadores: +, +=, -, -=. Recebido: %s' | O analisador esperava receber um dos operadores descritos. |
| 'Esperado um dos operadores: <=, <, >, >=, ==, !=, <>. Recebido %s' | O analisador esperava receber um dos operadores descritos. |
| 'Esperado loop. Recebido %s' | O analisador esperava receber o comando loop após um bloco de comandos. Exemplo: while a = 10 do print(a) loop |
| 'Esperado do. Recebido %s' | O analisador esperava receber o comando do após uma expressão. Exemplo: while a = 10 do print(a) loop |
| 'Esperado while. Recebido %s' | O analisador esperava receber o comando while. Exemplo: while a = 10 do print(a) loop |
| 'Esperado end-if. Recebido %s' | O analisador esperava receber o comando end-if após um comando de seleção. Exemplo if a = 10 then print(a) end-if ou if a = 10 then print(a) else print(b) end-if |
| 'Esperado ;. Recebido %s' | O analisador esperava receber um ; após o comando de leitura/escrita. Exemplo: print(“Hello world”); scan(a); |
| 'Esperado ). Recebido %s' | O analisador esperava receber um ) após o comando de leitura/escrita. Exemplo: print(“Hello world”); scan(a); |
| 'Esperado (. Recebido %s' | O analisador esperava receber um ( após o comando de leitura/escrita. Exemplo: print(“Hello world”); scan(a); |
| 'Esperado print ou println. Recebido %s' | O analisador esperava receber um comando de escrita (print ou println). |
| 'Esperado scan ou scanln. Recebido %s' | O analisador esperava receber um comando de leitura (scan ou scanln). |
| 'Esperado ;. Recebido %s' | O analisador esperava receber um ; após um comando de atribuição. Exemplo: a = 10; |
| 'Esperado um tipo de variável (int, float ou char). Recebido %s' | O analisador esperava receber um tipo de variável. Exemplo: int a, char b. |
| 'Esperado ). Recebido %s' | O analisador esperava receber um ) após o comando main. Exemplo: main () |
| 'Esperado (. Recebido %s' | O analisador esperava receber um ( após o comando main. Exemplo: main () |
| 'Esperado main. Recebido %s' | O analisador esperava receber o comando main como primeiro comando do programa. Exemplo: main () |
| 'Esperado var ou main. Recebido %s' | O analisador esperava receber o comando main ou var, como comando de declaração. |