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
loop	PR_LOOP
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

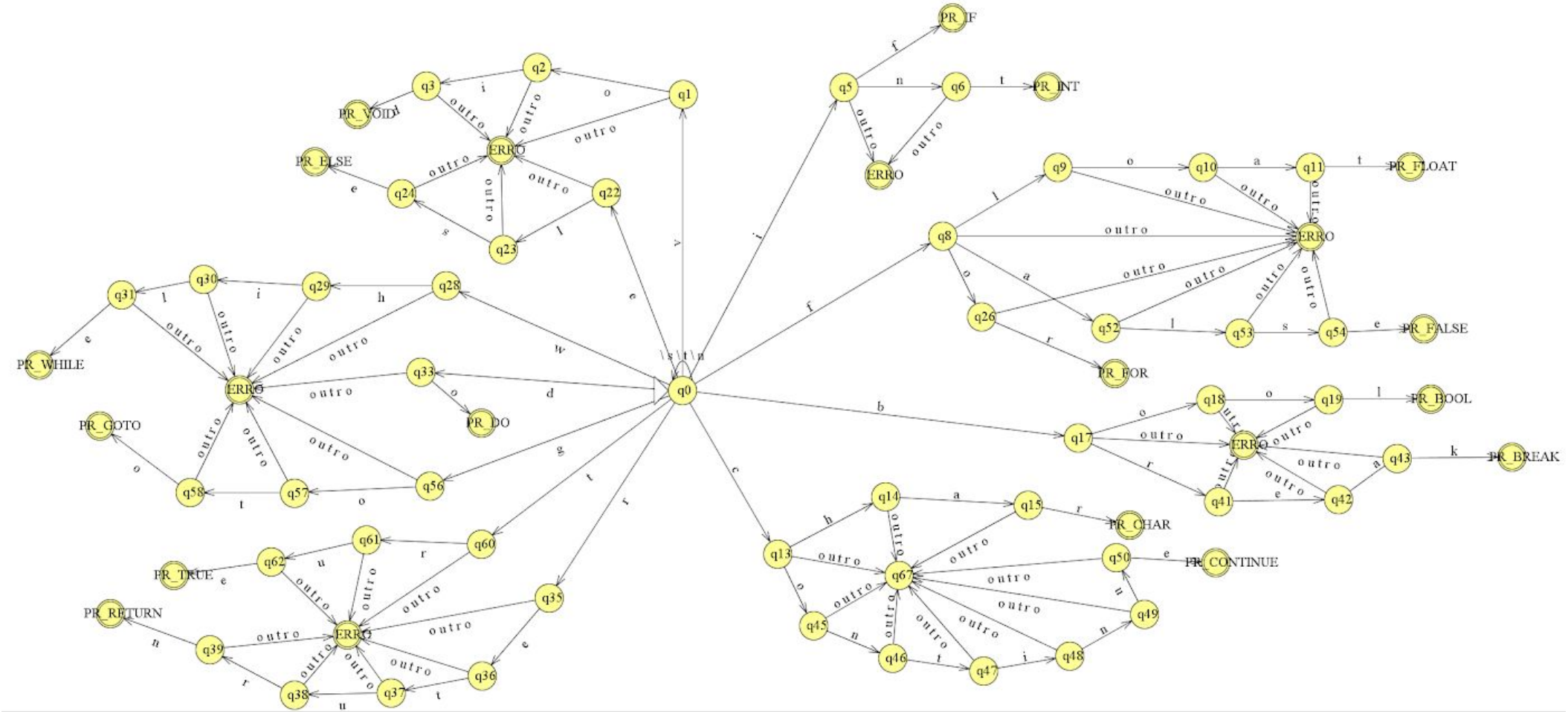
inteiros	LB_INT
reais	LB_FLOAT
caracteres	LB_CHAR
strings	LB_STRING
booleanos	LB_BOOL

Identificadores

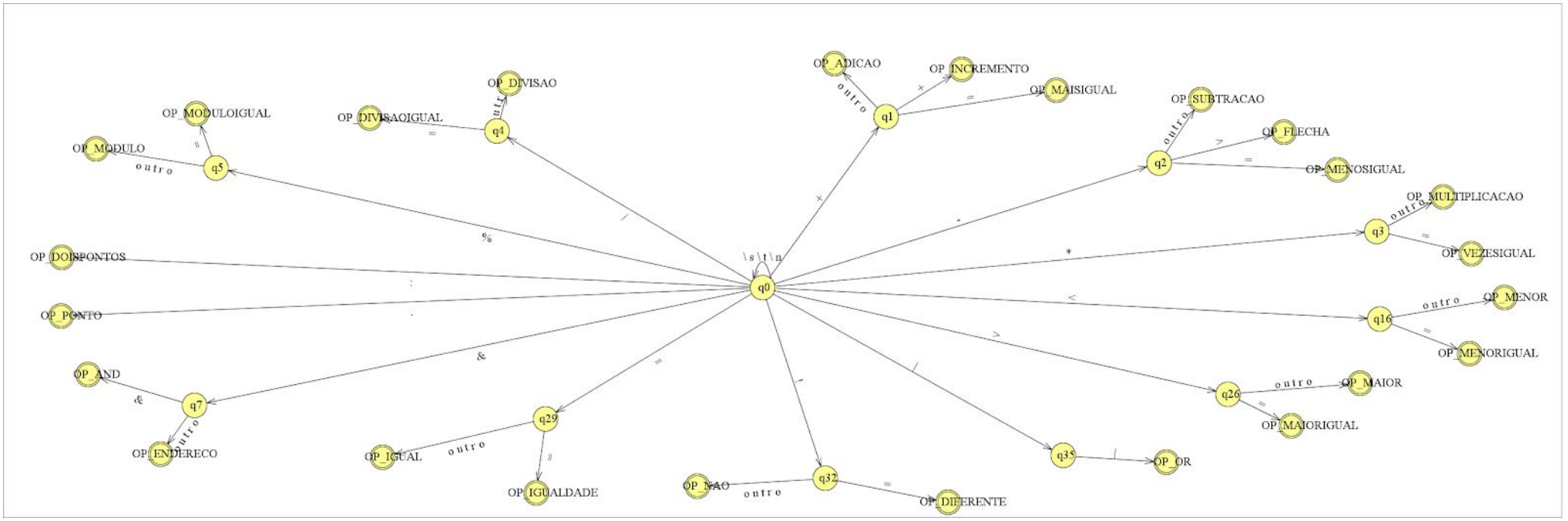
identificadores	ID

Diagramas de Transição

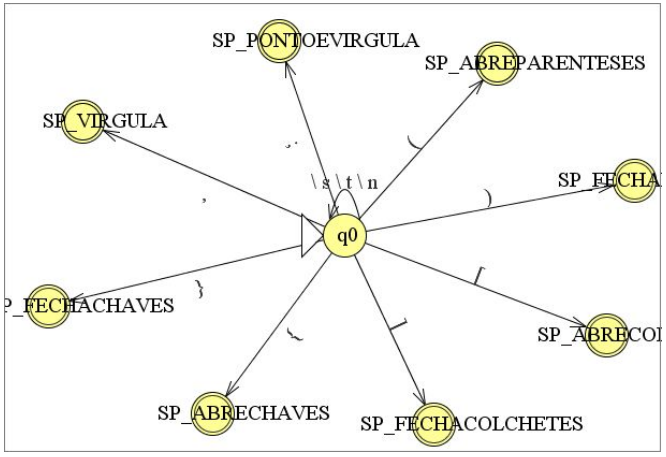
Palavras Reservadas



Operadores

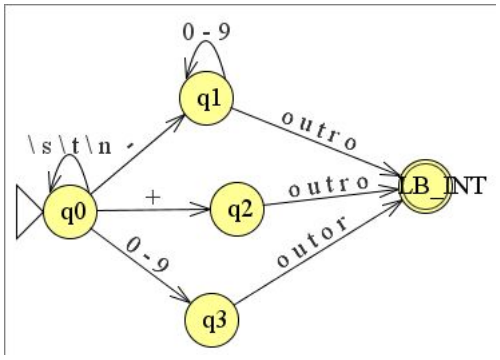


Sinais de Pontuação

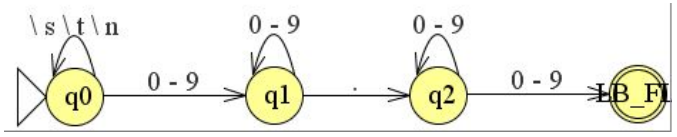


Literais Básicos

Inteiros



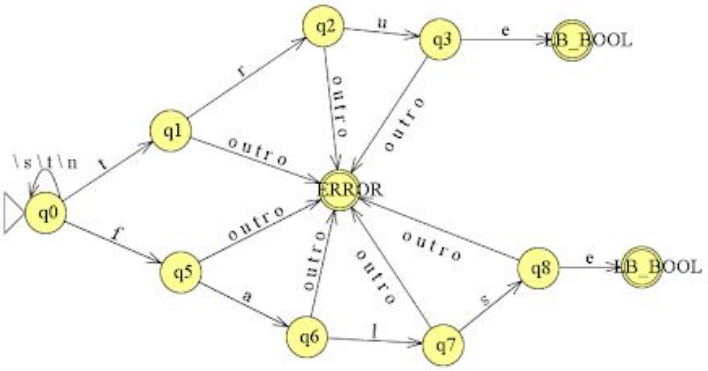
Reais



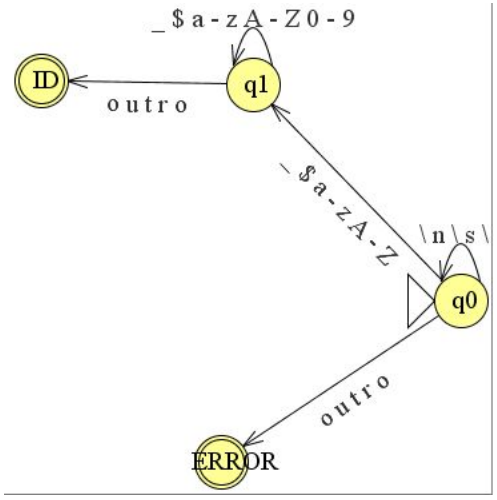
Caracteres

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 | ε
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 | ε
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 | ε
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 | ε
op-relac → <= | < | > | >= | == | <>
// exp-soma → exp-mult op-soma exp-soma | exp-multi
exp-soma → exp-mult exp-soma'
exp-soma' → op-soma exp-soma | ε
op-soma → + | -
// exp-mult → exp-simples op-mult exp-mult | exp-simples
exp-mult → exp-simples exp-mult'
exp-mult' → op-mult exp-mult | ε
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, ε	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, ε	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, ε	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, ε	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, ε	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.

Semântico

Tipo	Mensagem	Descrição
`Erro`	`'Identificador <{tipo}><{lexema}> já existente.'`	Ao tentar declarar uma variável com um identificador que já existe.
`Erro`	`'Identificador <{tipo}><{lexema}> não existe.'`	Ao tentar acessar uma variável que não foi definida.
`Erro`	`'Valor do tipo <{tipo}> não pode ser atribuído à variável <{variavel}> do tipo <{tipo_variavel}>'`	Ao tentar atribuir um valor de um tipo diferente do que foi declarado.
`Erro`	`'Operação inválida entre tipo <{tipo1}> e <{tipo2}>'`	Ao tentar fazer uma operação inválida entre tipos. Ex: ‘a’ + 6.
`Warning`	`'Variável <{lexema}> não foi inicializada.'`	A variável não foi inicializada, mas foi utilizada em algum momento.
`Warning`	`'Variável <{simbolo[0]}> foi declarada mas não foi utilizada.'`	A variável foi declarada, mas não foi utilizada em nenhum momento.

Análise Semântica

Tabela de Símbolos

Formato:

identificador	tipo	categoria	inicializada	utilizada	removida
nome da variável/função	tipo da variável/função (int, float, char)	se é variável ou função	se a variável foi inicializada	se a variável foi utilizada	se a variável foi removida

Funções criadas:

- `consultarSimbolo()`
  - Verifica se o símbolo existe na tabela de símbolos. Se existe, retorna o índice, senão, retorna -1.
- `consultarTipo()`
  - Retorna o tipo do símbolo.
- `consultarInicializada()`
  - Verifica se a variável foi inicializada.
- `removerSimbolo()`
  - Remove um símbolo, alterando a coluna “removida”.
- `adicionarSimbolo()`
  - Adiciona um símbolo na tabela de símbolos.
- `modificarInicializadaSimbolo()`
  - Marca um símbolo como inicializado ou não inicializado.
- `modificarUtilizadadaSimbolo()`
  - Marca um símbolo como utilizado ou não utilizado.

Mensagens de erros semânticos:

- `'Identificador <{tipo}><{lexema}> já existente.'`
  - Está presente na função `'var'`, chamada pela função `'decl_var'`
- `'Identificador <{tipo}><{lexema}> não existe.'`

    - Presente na função `'var'`, quando é chamada por qualquer outra função que somente utilize a variável já declarada.
- `'Valor do tipo <{tipo}> não pode ser atribuído à variável do tipo <{tipo_variavel}>'`
    - Presente na função `'com_atrib'`
- `'Operação inválida entre tipo <{tipo1}> e <{tipo2}>'`
    - Presente na função `'defineTipo'`
- `'Variável <{lexema}> não foi inicializada.'`
    - Presente na função `'var',` quando é chamada por qualquer outra função que somente utilize a variável já declarada.
- `'Variável <{simbolo[0]}> foi declarada mas não foi utilizada.'`
    - Presente na função `'sintatico'`, após fazer toda a análise sintática e semântica.

Log Semântico

- `'Símbolo <{lexema}> existe. Posição na tabela de símbolos: {i}.'`
    - Toda vez que uma variável for acessada.
- `'Símbolo <{lexema}> removido com sucesso.'`
    - Quando uma variável for removida.
- `'Símbolo <{lexema}> adicionado com sucesso.'`
    - Quando uma variável for criada.
- `'Símbolo <{lexema}> marcado como inicializado.'`
    - Quando uma variável for inicializada.
- `'Símbolo <{lexema}> marcado como utilizado.'`
    - Quando uma variável for utilizada.
- `'Valor do tipo <{tipo}> pode ser atribuído à variável <{variavel}> do tipo <{tipo_variavel}>'`
    - Quando o tipo atribuído aquela variável for válido.

Debug

Para utilizar o modo debug no analisador sintático e semântico, basta descomentar a linha 33 do arquivo core.py.

```
# TS = sintatico.sintatico(entrada, True)
```

Assim, ao realizar a análise, será gerada uma saída como no exemplo:

```
caller anterior: sintatico          caller atual: programa          token: PR_MAIN          lexema: main
caller anterior: programa           caller atual: lista_decl        token: PR_MAIN          lexema: main
caller anterior:                    caller atual: decl              token: PR_MAIN          lexema: main
caller anterior: decl               caller atual: decl_main         token: PR_MAIN          lexema: main
caller anterior: decl_main          caller atual: bloco             token: PR_VAR           lexema: var
caller anterior: bloco              caller atual: lista_com         token: PR_VAR           lexema: var
caller anterior:                    caller atual: comando           token: PR_VAR           lexema: var
caller anterior: comando            caller atual: decl_var          token: PR_VAR           lexema: var
caller anterior: decl_var           caller atual: espec_tipo        token: PR_INT           lexema: int
caller anterior: decl_var           caller atual: var               token: ID               lexema: a
```