

# Manual do Usuário Compilador D+

Lucas Ferreira de Almeida

13/10/2021

## Conteúdo

<b>1</b>	<b>Sobre o compilador</b>	<b>2</b>
1.1	Árvore de diretórios e Arquivos . . . . .	2
1.2	Como utilizar . . . . .	4
<b>2</b>	<b>Tokens</b>	<b>7</b>
2.1	Identificadores . . . . .	7
2.2	Literais . . . . .	7
2.3	Palavras Reservadas . . . . .	7
2.4	Pontuação . . . . .	7
2.5	Operadores . . . . .	8
<b>3</b>	<b>Diagramas de transição</b>	<b>9</b>
3.1	Identificadores . . . . .	9
3.2	Literais . . . . .	9
3.3	Pontuação . . . . .	12
3.4	Palavras Reservadas . . . . .	12
3.5	Operadores . . . . .	13
<b>4</b>	<b>Fatoração</b>	<b>14</b>
<b>5</b>	<b>Análise do Primeiro Símbolo</b>	<b>15</b>
<b>6</b>	<b>Tabela de Derivação</b>	<b>16</b>
6.1	Declarações . . . . .	16
6.2	Comandos . . . . .	17
6.3	Expressões . . . . .	17
<b>7</b>	<b>Validações</b>	<b>18</b>
7.1	Lexica . . . . .	18
7.2	Sintática & Semântica . . . . .	18

# 1 Sobre o compilador

## 1.1 Árvore de diretorios e Arquivos

```
-- DS_PLUS_COMPILER
----- Entrada (Pasta aonde fica o arquivo .d para ser compilado)
----- Src (Classes da aplicação)
----- File.cs (Classe para manusear arquivos)
----- LEXICA.cs (Classe para realizar a analise Léxica)
----- MEPA.cs (Classe para realizar a geração de código MEPA)
----- SEMANTICO.cs (Classe para realizar a analise Semântica)
----- Simbolos.cs (Classe com os objetos utilizados na TS)
----- SINTATICO.cs (Classe para realizar a analise Sintática)
----- Tokens.cs (Classe com os tokens)
----- doc
----- DTs (Diagramas de transição)
----- Img (Diagramas de transição em imagem)
----- Latex (Arquivo latex para gerar a documentação)
----- mu.pdf (Manual do usuário)
----- Config.cs (Arquivo com as configurações globais da aplicação)
----- DS_PLUS_COMPILER.cs (Main)
```

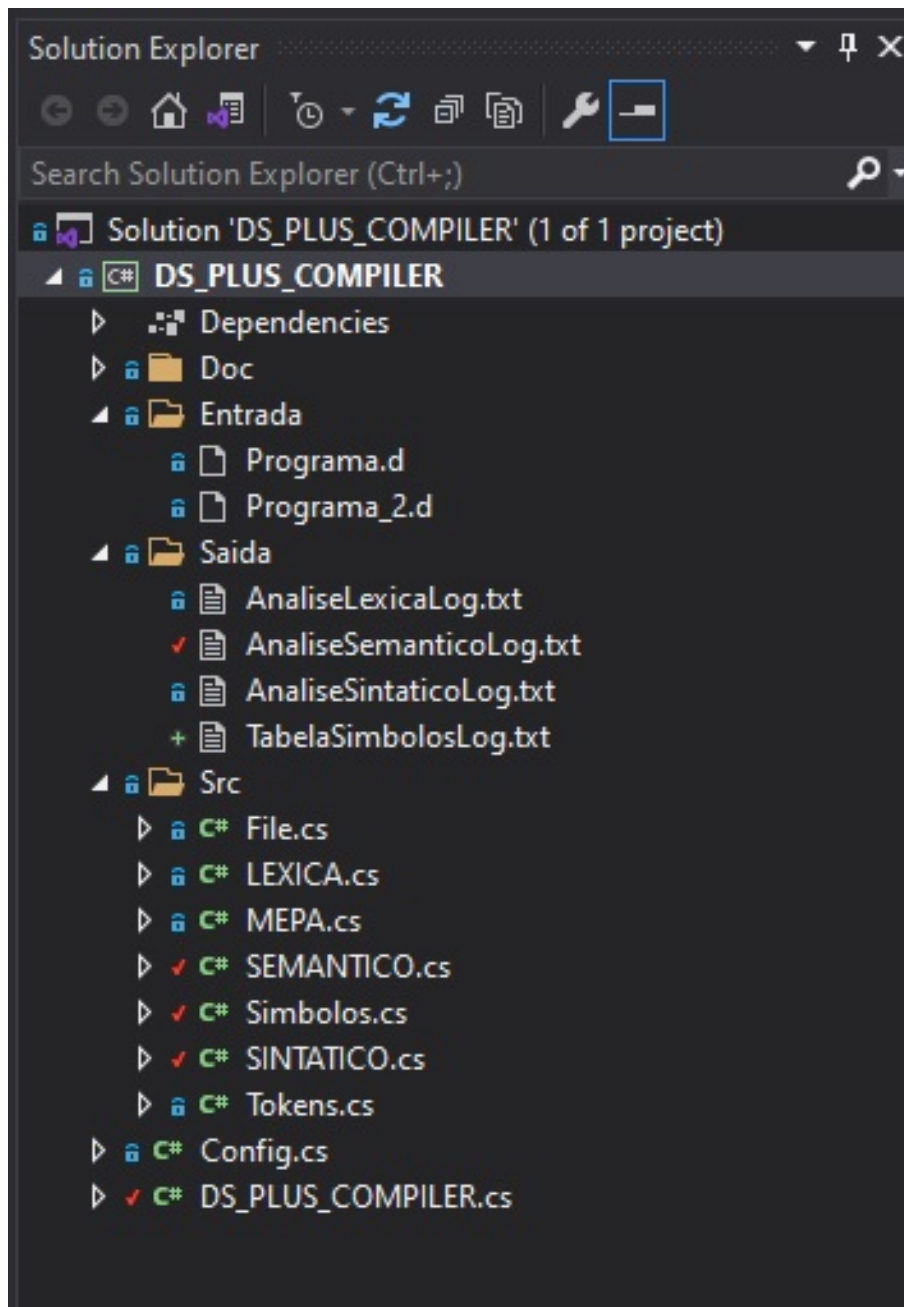


Figura 1: Árvore da aplicação.

## 1.2 Como utilizar

O arquivo de entrada deverá ser copiado para a pasta "Entrada", após isso é necessário rodar a aplicação. Todas as etapas das análises serão printadas no console e serão gerados arquivos com cada respectiva etapa.

```
BEM VINDO AO D+ COMPILER!

----- (INICIO) PRINT LEXICO -----

      LEXEMA                                TOKEN

      main                                PR_MAIN
      (                                  ABRE_PARENTESES
      )                                  FECHA_PARENTESES
      string                             PR_STR
      _teste                             ID
      =                                  OP_ATRI
      "teste de string"                  LIT_STR
      ;                                  PONTO_VIRGULA
      int                                PR_INT
      _inteiro                           ID
      =                                  OP_ATRI
      45                                 LIT_INT
      ;                                  PONTO_VIRGULA
      float                             PR_FLT
      _real                              ID
      =                                  OP_ATRI
      45.45                             LIT_FLT
      ;                                  PONTO_VIRGULA
      int                                PR_INT
      _int2                              ID
      ;                                  PONTO_VIRGULA
      _int2                              ID
      =                                  OP_ATRI
      _inteiro                           ID
      +                                  OP_SOMA
      _real                              ID
      ;                                  PONTO_VIRGULA
      print                             PR_PRINT
      (                                  ABRE_PARENTESES
      _int2                              ID
      )                                  FECHA_PARENTESES
      ;                                  PONTO_VIRGULA
      end                                PR_END
                                          FIM

----- (FIM) PRINT LEXICO -----
```

Figura 2: Print no console da análise Léxica.

C:\Users\nameless\source\repos\DS\_PLUS\_COMPILER\DS\_PLUS\_COMPILER\bin\Debug\net5.0\DS\_PLUS\_COMPILER.exe

----- (INICIO) - PRINT - SINTATICO -----

BLOCO	LEXEMA	SITUAÇÃO
programa	main	OK
decl-main	(	OK
decl-main	)	OK
decl-var	var	OK
espec-tipo	int	OK
decl-var	_teste1	OK
com-atr	=	OK
literal	3	OK
decl-var	;	OK
decl-var	var	OK
espec-tipo	int	OK
decl-var	_teste2	OK
com-atr	=	OK
literal	2	OK
decl-var	;	OK
decl-var	var	OK
espec-tipo	int	OK
decl-var	_teste3	OK
decl-var	;	OK
com-sel	if	OK
decl-var	_teste1	OK
op-relac	>	OK
decl-var	_teste2	OK
com-sel	then	OK
com-atr	_teste3	OK
com-atr	=	OK
literal	10	OK
decl-var	;	OK

Figura 3: Print no console da análise Sintática.

C:\Users\nameless\source\repos\DS\_PLUS\_COMPILER\DS\_PLUS\_COMPILER\bin\Debug\net5.0\DS\_PLUS\_COMPILER.exe

com-rep | ; | OK

PROGRAMA VALIDO SINTATICAMENTE!

----- (INICIO) - PRINT - TABELA - DE - SIMBOLOS -----

NOME	TIPO	ESCOPO	INICIALIZADA	ATIVO	STATUS
teste1	INT	Global	False	True	Inserindo no Global
teste1	INT	Global	True	True	Inicializa a variavel
teste2	INT	Global	False	True	Inserindo no Global
teste2	INT	Global	True	True	Inicializa a variavel
teste3	INT	Global	False	True	Inserindo no Global
teste3	INT	Global	True	True	Inicializa a variavel
teste_float	FLOAT	Global	False	True	Inserindo no Global
i	INT	Global	False	True	Inserindo no Global
i	INT	Global	True	True	Inicializa a variavel
int_local	INT	Local	False	True	Inserindo no Local
int_local	INT	Local	True	True	Inicializa a variavel
int3	INT	Local	False	True	Inserindo no Local
int3	INT	Local	True	True	Inicializa a variavel
int_local	INT	Local	True	False	Saida no while.
int3	INT	Local	True	False	Saida no while.
teste1	INT	Global	True	False	Saida no main.
teste2	INT	Global	True	False	Saida no main.
teste3	INT	Global	True	False	Saida no main.
teste_float	FLOAT	Global	False	False	Saida no main.
i	INT	Global	True	False	Saida no main.

Figura 4: Print no console da Tabela de simbolo.

```

----- (INICIO) -PRINT-ANALISE-SEMANTICA-----
ESTRUTURA      LEXEMA(S)      TIPO      SITUAÇÃO
Main            _teste1      declarada  OK
Main            _teste1      atribuicao  OK
Main            LIT_INT - LIT_INT  OpSoma     OK
Main            LIT_INT + LIT_INT  OpSoma     OK
Main            LIT_INT * LIT_INT  OpRelac    OK
Main            _teste2      declarada  OK
Main            _teste2      atribuicao  OK
Main            LIT_INT / LIT_INT  OpRelac    OK
Main            _teste3      declarada  OK
ComSelec        _teste1      busca na tabela  OK
ComSelec        _teste1      inicializada  OK
ComSelec        LIT_INT > LIT_INT  OpRelac    OK
ComSelec        _teste2      busca na tabela  OK
ComSelec        _teste2      inicializada  OK
Main            _teste3      atribuicao  OK
ComSelec        _teste2      busca na tabela  OK
ComSelec        _teste2      inicializada  OK
ComSelec        LIT_INT > LIT_INT  OpRelac    OK
ComSelec        _teste1      busca na tabela  OK
ComSelec        _teste1      inicializada  OK
Main            _teste3      atribuicao  OK
Main            _teste3      atribuicao  OK
Main            _teste_float  declarada  OK
Main            _teste_float  atribuicao  OK
ComEscrita      _teste_float  busca na tabela  OK
ComEscrita      _teste_float  inicializada  OK
Main            _i      declarada  OK
Main            _i      atribuicao  OK
ComRepeticao     _i      busca na tabela  OK
ComRepeticao     _i      inicializada  OK
ComRepeticao     LIT_INT < LIT_INT  OpRelac    OK
ComRepeticao     _i      atribuicao  OK
Main            _int_local  declarada  OK
Main            _int_local  atribuicao  OK
Main            _int_local2  declarada  OK
Main            _int_local2  atribuicao  OK
Main            _int3      declarada  OK
Main            _int_local  busca na tabela  OK
Main            _int_local  Tipos      OK
Main            _int_local  busca na tabela  OK
Main            _int_local  inicializada  OK
Main            LIT_INT + LIT_INT  OpSoma     OK
Main            _int_local2  busca na tabela  OK
Main            _int_local2  inicializada  OK

PROGRAMA VALIDO SEMANTICAMENTE!

```

Figura 5: Print no console da análise Semântica.

## 2 Tokens

### 2.1 Identificadores

Identificador		ID
---------------	--	----

### 2.2 Literais

Inteiros		LIT_INT
Reais		LIT_FLT
Caractere		LIT_CHAR
String		LIT_STR
Boolean		LIT_BOOL

### 2.3 Palavras Reservadas

void		PR_VOID
int		PR_INT
float		PR_FLT
char		PR_CHAR
bool		PR_BOOL
if		PR_IF
then		PR_THEN
else		PR_ELSE
end		PR_END
for		PR_FOR
while		PR_WHILE
do		PR_DO
loop		PR_LOOP
return		PR_RETURN
true		PR_TRUE
false		PR_FALSE
var		PR_VAR
main		PR_MAIN
scan		PR_SCAN
print		PR_PRINT

### 2.4 Pontuação

,		VIRGULA
;		PONTO_VIRGULA
(		ABRE_PARENTESES
)		FECHA_PARENTESES
[		ABRE_COLCHETES
]		FECHA_COLCHETES
{		ABRE_CHAVES
}		FECHA_CHAVES

## 2.5 Operadores

+	OP_SOMA
-	OP_SUB
*	OP_MULT
/	OP_DIV
%	OP_MOD
.	OP_PONTO
<	OP_MENOR
>	OP_MAIOR
==	OP_IGUAL
!=	OP_DIF
<=	OP_MEN_IGUAL
>=	OP_MAI_IGUAL
=	OP_ATRI



### 3 Diagramas de transição

#### 3.1 Identificadores

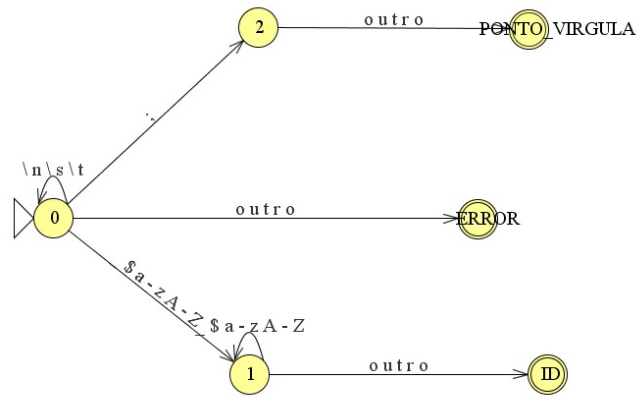


Figura 6: Indetificadores.

#### 3.2 Literais



Figura 7: Literais (String).

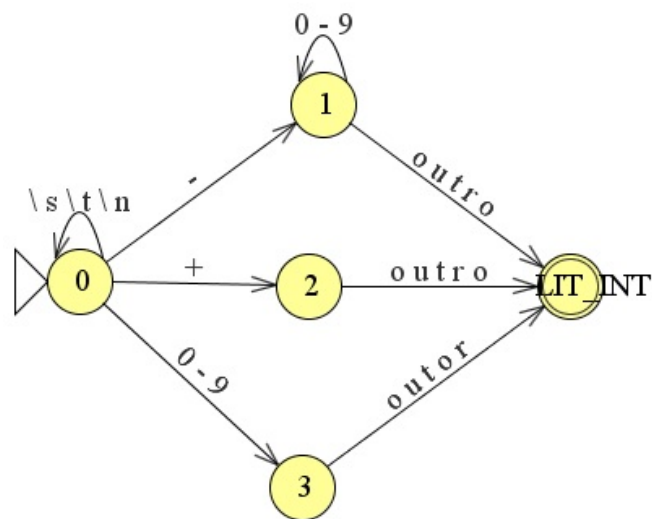


Figura 8: Literais (Inteiro).

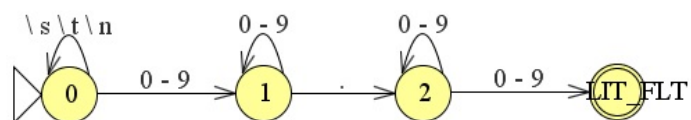


Figura 9: Literais (Reais).



Figura 10: Literais (Char).

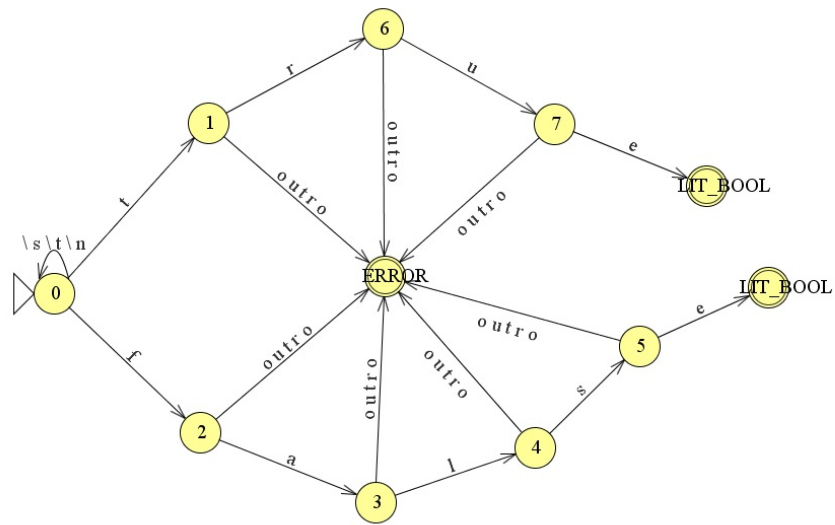


Figura 11: Literais (Boolean).

### 3.3 Pontuação

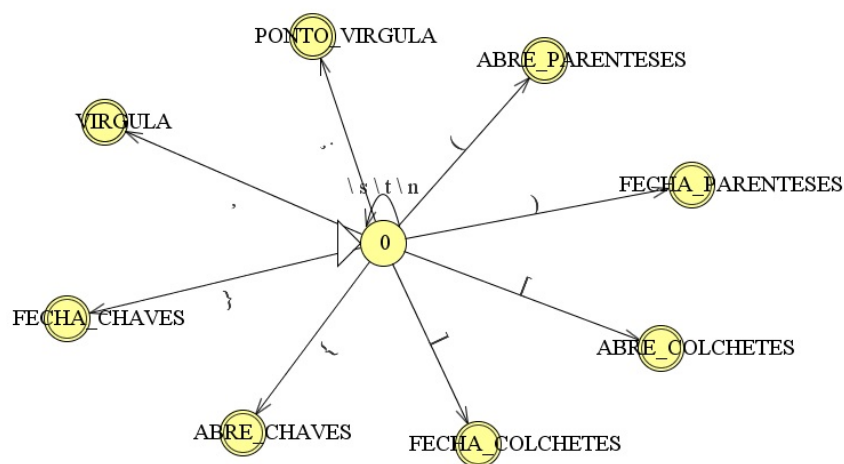


Figura 12: Literais (Boolean).

### 3.4 Palavras Reservadas

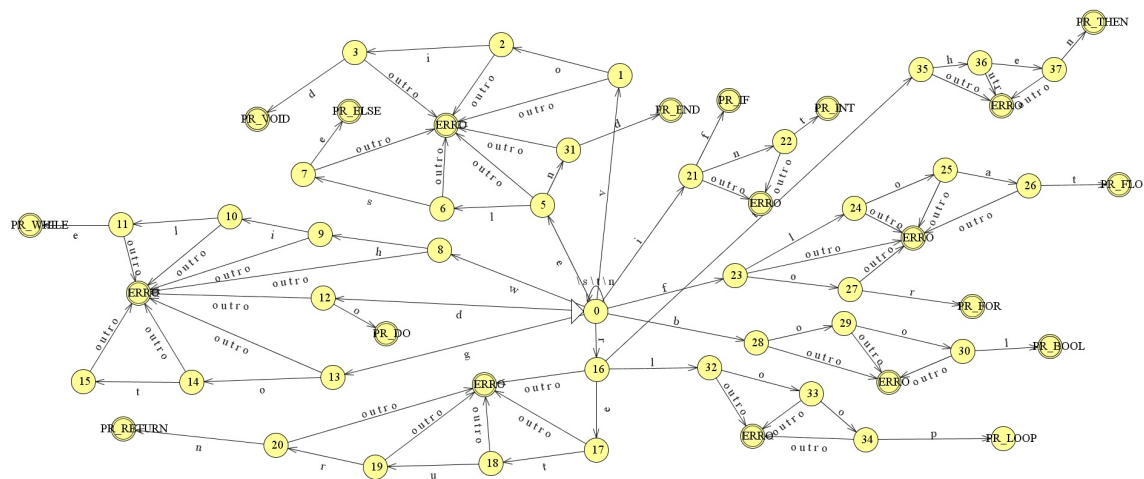


Figura 13: Literais (Boolean).

### 3.5 Operadores

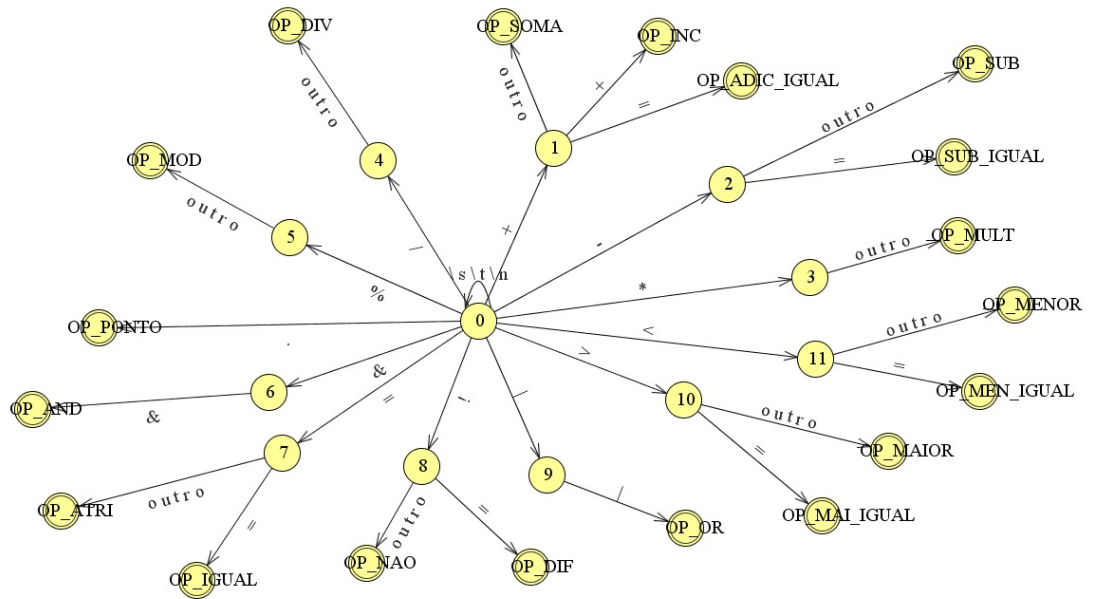


Figura 14: Literais (Boolean).

## 4 Fatoração

```
programa → lista-decl

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

com-atrib → var = exp ;
com-leitura → SCAN ( var ) ;
com-escrita → PRINT ( exp ) ;

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

bloco → lista-com
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 | &

op-relac → <= | < | > | >= | == | <>
// exp-soma → exp-mult op-soma exp-soma | exp-multi
exp-soma → exp-mult exp-soma'
exp-soma' → op-soma exp-soma |
op-soma → + | -

// exp-mult → exp-simples op-mult exp-mult | exp-simples
exp-mult → exp-simples exp-mult'
exp-mult' → op-mult exp-mult | &
op-mult → * | / | DIV | MOD
exp-simples → ( exp ) | var | literal

literal → NUMINT | NUMREAL | CHAR | STRING
var → ID
```

## 5 Análise do Primeiro Símbolo

APS	P	P+	t
programa	lista-decl	P(lista-decl)	VAR, MAIN()
list-decl	decl	P(decl)	VAR, MAIN()
lista-decl'	decl, &	P(decl), S(lista-decl')-> S(lista-decl)-> S(programa)-> #	VAR, MAIN()
decl	decl-var, decl-main	P(decl-var), P(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	P(lista-com)	VAR, ID, IF, WHILE, SCAN( PRINT(, END
lista-com	comando, &	P(comando), S(lista-com)-> S(bloco)-> #	VAR, ID, IF, WHILE, SCAN( PRINT(, END
comando	decl-var, com-atrib, com-selecao, com-repeticao, com-leitura, com-escrita	P(decl-var), P(com-atrib), P(com-selecao), P(com-repeticao), P(com-leitura), P(com-escrita)	VAR, ID, IF, WHILE, SCAN( PRINT(
com-atrib	var	P(var)	ID
com-leitura	SCAN(	SCAN(	SCAN(
com-escrita	PRINT(	PRINT(	PRINT(
com-selecao	IF	IF	IF
com-selecao'	END, ELSE	END, ELSE	END, ELSE
com-repeticao	WHILE	WHILE	WHILE
exp	exp-soma	P(exp-soma)	(, INT, REAL, CHAR, STR, ID

exp'	op-relac, &	P(op-relac), S(exp')-> S(exp)-> );	<=, <, >, >=, ==, <>, );
op-relac	<=, <, >, >=, ==, <>	<=, <, >, >=, ==, <>	<=, <, >, >=, ==, <>
exp-soma	exp-mult	P(exp-mult)	(, INT, REAL, CHAR, STR, ID
exp-soma'	op-soma, &	P(op-soma), S(exp-soma')-> S(exp-soma)-> S(exp')-> S(exp)-> );	+, -, );
op-soma	+, -	+, -	+, -
exp-mult	exp-simples	P(exp-simples)	(, INT, REAL, CHAR, STR, ID
exp-mult'	op-mult, &	P(op-mult), S(exp-mult')-> S(exp-mult)-> P(exp-soma))	*,/,DIV, MOD,+, -, );
op-mult	*,/,DIV,MOD	*,/,DIV,MOD	*,/,DIV,MOD
exp-simples	(, var, literal	(, P(var), P(literal)	(, INT, REAL, CHAR, STR, ID
literal	INT, REAL, CHAR, STR	(, P(var), P(lit)	(, INT, REAL, CHAR, STR, ID
var	ID	ID	ID

## 6 Tabela de Derivação

### 6.1 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



## 6.2 Comandos

TD	SCAN(	PRINT(	IF	WHILE	VAR	ID	END
bloco	lst-com	lst-com	lst-com	lst-com	lst-com	lst-com	lst-com
lst-com	comando	comando	comando	comando	comando	comando	comando
comando	com-lei	com-lei	com-lei	com-lei	com-atr	-	-
com-atr	-	-	-	-	var	-	-
com-lei	SCAN(	-	-	-	-	-	-
com-esc	-	PRINT(	-	-	-	-	-
com-sel	-	-	IF	-	-	-	-
com-rep	-	-	-	WHILE	-	-	-
var	-	-	-	-	-	ID	-

## 6.3 Expressões

TD	<=	<	>	>=	==	<>	+	-	*	/
exp	-	-	-	-	-	-	-	-	-	-
op-rel	<=	<	>	>=	==	<>	-	-	-	-
exp-som	-	-	-	-	-	-	-	-	-	-
op-som	-	-	-	-	-	-	+	-	-	-
exp-mul	-	-	-	-	-	-	-	-	-	-
op-mul	-	-	-	-	-	-	-	-	*	/
exp-sim	-	-	-	-	-	-	-	-	-	-
lit	-	-	-	-	-	-	-	-	-	-
var	-	-	-	-	-	-	-	-	-	-

TD	DIV	MOD	INT	REAL	CHAR	STR	ID	(	);
exp	-	-	exp-som	exp-som	exp-som	exp-som	exp-som	exp-som	-
op-rel	-	-	-	-	-	-	-	-	-
exp-som	-	-	exp-mul	exp-mul	exp-mul	exp-mul	exp-mul	exp-mul	-
op-som	-	-	-	-	-	-	-	-	-
exp-mul	-	-	exp-sim	exp-sim	exp-sim	exp-sim	exp-sim	exp-sim	-
op-mul	DIV	MOD	-	-	-	-	-	-	-
exp-sim	-	-	lit	lit	lit	lit	var	(	-
lit	-	-	INT	REAL	CHAR	STR	-	-	-
var	-	-	-	-	-	-	ID	-	-

## 7 Validações

### 7.1 Lexica

Tipo	Mensagem
Erro	'Erro': Operador ou pontuação inválida. Lexema = '{Lexema}'.
Erro	'Erro': Char inválido. Lexema = '{Lexema}'.
Erro	'Erro': Comando não identificado. Lexema = '{Lexema}'.

### 7.2 Sintática & Semântica

Mensagem
Erro na estrutura '{estrutura}', esperado '{lexemas_esperados}' encontrado lexema '{lexema}'.
Erro na linha '{linha}' -> Tipo diferente da variavel declarada '{lexima}'.
Erro na linha '{linha}' -> Variavel '{lexima}' nao inicializada.
Erro na linha '{linha}' -> Variavel '{lexima}' já declarada.
Erro na linha '{linha}' -> Operação '{literal}' * '{literal}' inválida semânticamente.