Documentação Compilador L

Alunos:

Gustavo Lopes Rodrigues

Lucas Fonseca Saliba

Thiago Henriques Nogueira

Professor:

Alexei Manso Correa Machado

Alfabeto

N°	Token	Lexema
1	const	const
2	int	int
3	char	char
4	while	while
5	if	if
6	float	float
7	else	else
8	&&	&&
9	||	||

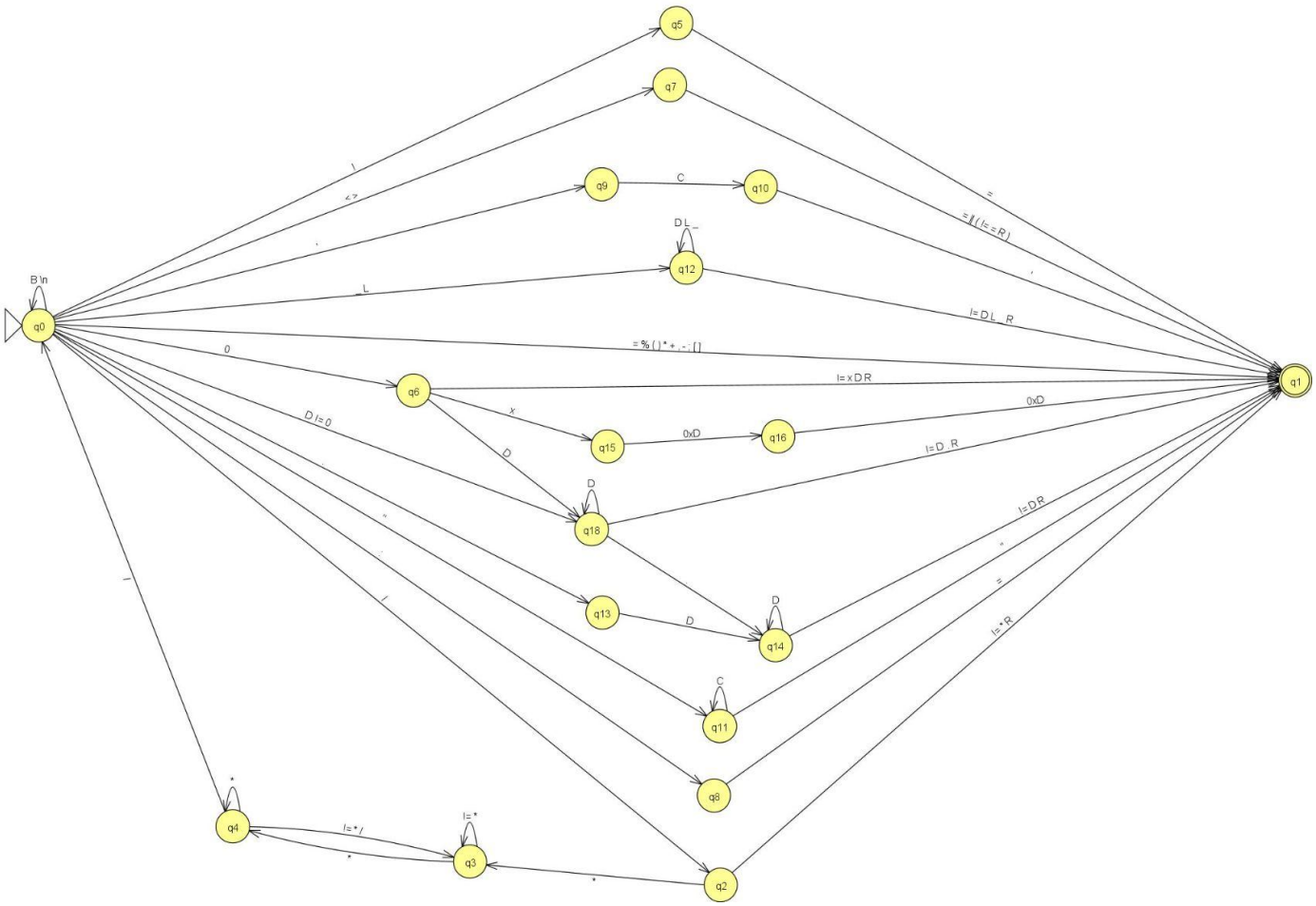
10	!	!
11	:=	:=
12	=	=
13	(	(
14	)	)
15	<	<
16	>	>
17	!=	!=
18	>=	>=
19	<=	<=
20	,	,
21	+	+
22	-	-
23	*	*
24	/	/
25	;	;
26	{	{
27	}	}
28	readln	readln
29	div	div
30	string	string
31	write	write
32	writeln	writeln

33	mod	mod
34	[	[
35	]	]
36	true	true
37	false	false
38	boolean	boolean

basicConst: ‘c’ ∪ 0xDD ∪ “c*” ∪ d+[.d*] ∪ .d+
identifier: (l ∪ _)(d ∪ l ∪ _)*

Legenda:
Caractere Azul- Parte de ou uma Palavra Reservada.
Caractere Vermelho - Símbolo Especial.

Analisador Léxico



Gramática

```
S -> {(A | E)}* fim_arquivo // estado inicial
A -> (B | D) // declaracao var/const
B -> C identifier [:= [-]basicConst]{, identifier [:= [-]basicConst]}*; //
declaracao var
C -> (int | float | string | boolean | char)    // tipo var
D -> const identifier = [-]basicConst; // declaracao const
E -> [(F | G | I | K | L | S)]; // comando atribuicao / while / if /
readln / write / writeln / ;
F -> identifier[[M]] := M // atribuicao
G -> while (M) H // while
H -> (E | {{E}*}) // comando / sequencia de comandos
I -> if (M) J [else J] // if
J -> (E | {{E}*}) // comando / sequencia de comandos
K -> readln(identifier) // readln
L -> (write(M {, M}*) | writeln(M {, M}*)) // write / writeln
M -> N {(= | != | < | <= | > | >=) N}* // expressao precedencia 6
N -> [+ | -]O {(+ | - | ||) O}* // expressao precedencia 5
O -> P {(* | / | mod | div | &&) P}* // expressao precedencia 4
P -> (Q | !Q) // expressao precedencia 3
Q -> (R | float(M) | int(M)) // expressao precedencia 2
R -> (basicConst | identifier[[M]] | (M)) // expressao precedencia 1
```

Legenda:
Caractere Azul - Parte de ou uma Palavra Reservada.
Caractere Vermelho - Símbolo Não Terminal.

Esquema de Tradução

```
S -> {(A | E)}* fim_arquivo // estado inicial
A -> (B | D) // declaracao var/const
B -> C identifier[1][6] [:= [-[7]]basicConst[8][9]] {, identifier[1][6]
[:= [-[7]]basicConst[8][9]]}*; // declaracao var
C -> (int | float | string | boolean | char)[5] // tipo var
D -> const identifier[2] = [-[7]]basicConst[8][10]; // declaracao const
E -> [(F | G | I | K | L | S)]; // comando atribuicao / while / if /
readln / write / writeln / ;
F -> identifier[3][4][[[11]M[12]]] := M1[13] // atribuicao
G -> while (M[14]) H // while
H -> (E | {{E}*}) // comando / sequencia de comandos
I -> if (M[40]) J [else J] // if
J -> (E | {{E}*}) // comando / sequencia de comandos
K -> readln(identifier[3][4][15]) // readln
L -> (write(M[16] {, M1[17]}*) | writeln(M[16] {, M1[17]}*)) // write /
writeln
M -> N[33] {(= | != | < | <= | > | >=)[34] N1[35]}* // expressao
precedencia 6
N -> [+[29] | -[29]]O[30] {(+ | - | ||)[31] O1[32]}* // expressao
precedencia 5
O -> P[26] {(* | / | mod | div | &&)[27] P1[28]}* // expressao precedencia
P -> (Q[24] | !Q[25]) // expressao precedencia 3
Q -> (R[21] | float(M[22]) | int(M[23])) // expressao precedencia 2
R -> (basicConst[18] | identifier[3][[[11]M[12]]][19] | (M[20])) //
expressao precedencia 1
```

Legenda:
Caractere Azul - Parte de ou uma Palavra Reservada.
Caractere Vermelho - Símbolo Não Terminal.
[Número] Roxo - Regra com Verificação de Compatibilidade entre Classes, Tipos e Unicidade.

Esquema de Tradução

```
[1]
{
se id.classe != vazio
erro
senao
id.classe = classe-var
}

[2]
{
se id.classe != vazio
erro
senao
id.classe = classe-const
}

[3]
{
se id.classe == vazio
erro
}

[4]
{
se id.classe != classe-var
erro
}

[5]
{
C.tipo = token.lex
}

[6]
{
id.tipo = C.tipo
}

[7]
{
const.flag = 1
}

[8]
```

```
{
se const.flag == 1 e const.tipo != real e const.tipo != inteiro
erro
}

[9]
{
se id.tipo != const.tipo e !(id.tipo == real e const.tipo == inteiro)
erro
}

[10]
{
id.tipo = const.tipo
}

[11]
{
se id.tipo != string
      erro
senao id.flag = 1
}

[12]
{
se M.tipo != inteiro
      erro
}

[13]
{
se id.flag == 1 e M1.tipo != caractere
erro
senao
se id.flag != 1 e id.tipo != M1.tipo e !(id.tipo == real e M1.tipo ==
inteiro)
erro
}

[14]
{
se M.tipo != logico
erro
}
```

```
[15]
{
se id.tipo == logico
erro
}

[16]
{
se M.tipo == logico
erro
}

[17]
{
se M1.tipo == logico
erro
}

[18]
{
R.tipo = const.tipo
}

[19]
{
se id.flag == 1
R.tipo = caractere
}

[20]
{
R.tipo = M.tipo
R.end = M.end
}

[21]
{
Q.tipo = R.tipo
Q.end = R.end
}

[22]
{
se M.tipo != real e M.tipo != inteiro
erro
senao
```

```
Q.tipo = real
Q.end = M.end
}
[23]
{
se M.tipo != real e M.tipo != inteiro
erro
senao
Q.tipo = inteiro
Q.end = M.end
}

[24]
{
P.tipo = Q.tipo
P.end = Q.end
}

[25]
{
se Q.tipo != logico
erro
}

[26]
{
O.tipo = P.tipo
O.end = P.end
}

[27]
{
O.op = token.lex
}

[28]
{
se O.op == and
se O.tipo != logico ou P1.tipo != logico
erro

senao se O.op == * ou O.op == /
se (O.tipo != real e O.tipo != inteiro) ou (P1.tipo != real e P1.tipo !=
inteiro)
erro

senao
```

```
se O.tipo != P1.tipo
O.tipo = real

senao
se O.tipo == inteiro e O.op == /
erro

senao
se O.tipo != inteiro ou P1.tipo != inteiro
erro

senao
O.tipo = P.tipo
}

[29]
{
N.flag = 1
}

[30]
{
se N.flag == 1 e O.tipo != real e O.tipo != inteiro
erro
senao
N.tipo = O.tipo
N.end = O.end
}

[31]
{
N.op = token.lex
}

[32]
{
se N.op == or
      se N.tipo != logico ou O1.tipo != logico
      erro
senao

      se (N.tipo != real e N.tipo != inteiro) ou (O1.tipo != real e
O1.tipo != inteiro)
erro
senao
se N.tipo != O1.tipo
N.tipo = real
```

```
}

[33]
{
M.tipo = N.tipo
M.end = N.end
}

[34]
{
M.op = token.lex
}

[35]
{
se M.op == "==" e M.tipo == string e N1.tipo == string
M.tipo = logico

senao se M.tipo == caractere e N1.tipo == caractere
M.tipo = logico

senao se (M.tipo == real ou M.tipo == inteiro) e (N1.tipo == real ou
N1.tipo == inteiro)
M.tipo = logico

senao
erro
}

[40]
{
se M.tipo != logico
erro
}
```