Pontifícia Universidade Católica de Minas Gerais

Bacharelado em Ciência da Computação

Fabio Silva Campos Melo

Gustavo Lescowicz Kotarsky

Lucas Dutra Ponce de Leon

Compilador Linguagem L

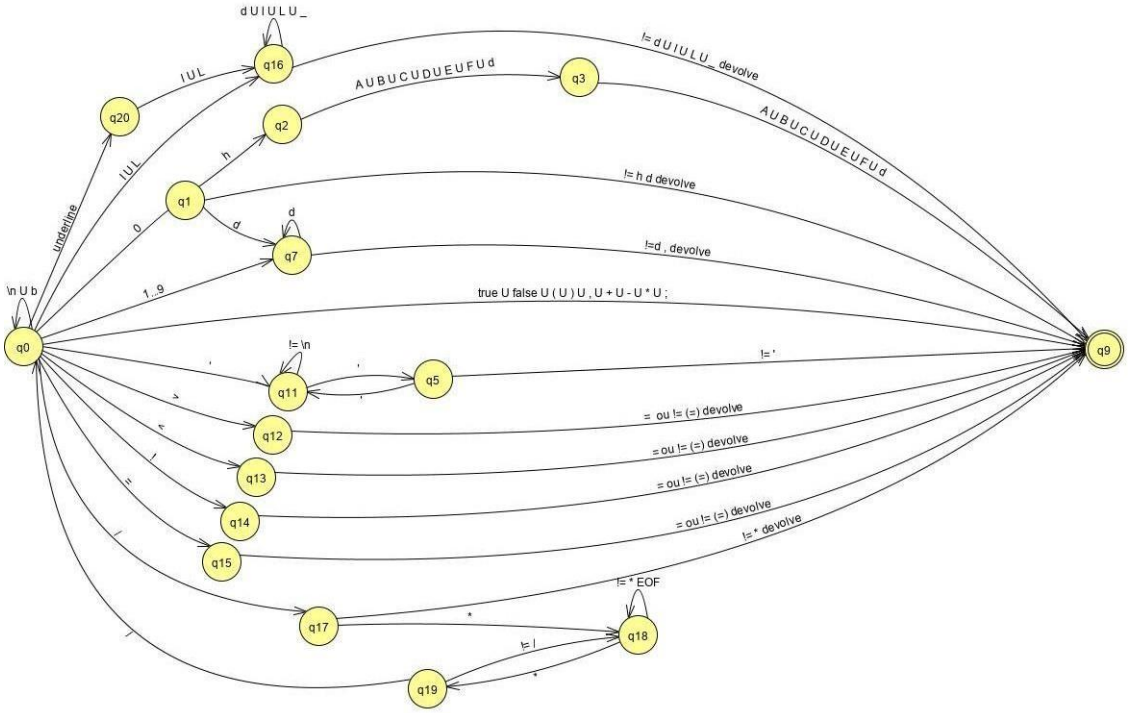
Belo Horizonte, Minas Gerais

2019

Alfabeto, padrão de formação dos lexemas:

Num	Token	Lex
1	*Byte*	0h(A U B U C U D U E U F)(A U B U C U D U E U F) U d
2	*Integer*	[-]d
3	*String*	‘(!= \n)*’
4	*Boolean*	true U false
5	id	(_ l U _ L U l U L)+ (d U L U l U _)*
6	while	while
7	if	if
8	else	else
9	and	and
10	or	or
11	not	not
12	=	=
13	==	==
14	(	(
15	)	)
16	<	<
17	>	>
18	!=	!=
19	>=	>=
20	<=	<=
21	,	,
22	+	+
23	-	-
24	*	*
25	/	/
26	;	;
27	begin	begin
28	end	end
29	then	then
30	readln	readln
31	main	main
32	write	write
33	writeln	writeln
34	true	true
35	false	false
36	boolean	boolean
37	string	string
38	const	const
39	byte	byte
40	integer	integer

Autômato:


d U I U L U _
q16
l U L
q20
A U B U C U D U E U F U d
q3
!= d U I U L U _ devolve
A U B U C U D U E U F U d
q2
h
q1
!= h d devolve
underline
0
d
d
q7
!=d , devolve
1 .. 9
\n U b
true U false U (U) U , U + U - U * U ;
q0
q9
!= \n
'
'
q11
q5
!= '
>
q12
= ou != (=) devolve
<
q13
= ou != (=) devolve
!
q14
= ou != (=) devolve
=
q15
= ou != (=) devolve
!= * devolve
/
q17
!= * EOF
*
q18
!= /
/
q19
*

Gramática:

S -> {**D**}+ main {**C**}+end

D->(integer|boolean|byte|string)id**[=[-]v_const]**{,id**[=[-]v_const]**};|

const id =[-]v_const;

C -> id = **EXP**;|

while ‘(‘ **EXP** ’)’ **W |**

if ‘(‘**EXP**’)’ then (**C** [else (**W**)] **|** begin {**C**} end [else (**W**)]) **|**

; **|**

readln’(‘ id ’)’; **|**

(write **|** writeln)’(‘ **EXP**{,**EXP**} ’)’;

W -> begin {**C**} end **|** **C**

EXP -> **EXPS** [(==**|!**=|<**|**>|<=|>=) **EXPS**]

EXPS -> [+|-] **T** {(+|-|or) **T**}

T -> **F** {(*|and|/) **F**}

F -> ‘(‘ **EXP** ’)’| id **|** v_const **|** not **F**

Semântico

S -> {**D**}+ main {**C**}+end

D -> (integer**(1)** | boolean**(2)** | byte**(3)** | string**(4)**)**(5)** id**(7)** **[= [-(8)]v_const(9)]** {,id**(7)** **[= [-]v_const(9)]**}; | const**(6)** id**(7)** = [-]v_const**(9)**;

C -> id**(10)** = **EXP(11)**;|

while '(' **EXP(12)** ')' **W** |

if '('**EXP(12)**')' then (**C** [else (**W**)] | begin {**C**} end [else (**W**)]) |

; |

readln'(' id**(13)** ')'; |

(write | writeln)'(' **EXP**{,**EXP**} ')';

W -> begin {**C**} end | **C**

EXP -> **EXPS(24)** [(==|!=|<|>|<=|>=) **EXPS1(25)**]

EXPS -> [+**(20)**|-**(21)(22)**] **T** {(+|-|or) **T1(23)**}

T -> **F(18)** {(*|and|/) **F(19)**}

F -> '(' **EXP(14)** ')'| id**(15)** | v_const**(16)** | not **F1(17)**

DECLARAÇÕES

(1) auxD.tipo = integer

(2) auxD.tipo = boolean

(3) auxD.tipo = byte

(4) auxD.tipo = string

(5) auxD.classe = var

(6) auxD.classe = const

(7) se id.classe == sem_classe

entao id.classe = auxD.classe

se(!=const)

id.tipo = auxD.tipo

senao

ERRO(Declarado)

(8) sinal = true

(9) se v_const.tipo != auxD.tipo

entao ERRO(TIPO INCOPATIVEL)

COMANDOS

(10) se id.classe == sem_classe

entao ERRO(ID não declarado)

se id.classe == const

entao ERRO(CLASSE INCOPATIVEL)

(11) se EXP.tipo != id.tipo

entao ERRO(TIPOS INCOPATIVEIS)

(12) se EXP.tipo != boolean

entao ERRO(TIPOS INCOPATIVEIS)

(13) se id.classe == sem_classe

entao ERRO(ID não declarado)

se id.classe == const

entao ERRO(CLASSE INCOPATIVEL)

se id.tipo == boolean

entao ERRO(ID INCOPATIVEL)

EXPRESSÕES

(14) F.tipo = EXP.tipo

(15) se id.classe == sem_classe

ERRO(ID não declarado)

senao se id.classe == const

ERRO(CLASSE INCOPATIVEL)

senao F.tipo = id.tipo

(16) F.tipo = v_const.tipo

(17) se F1.tipo != boolean

ERRO(TIPOS INCOPATIVEIS)

senao

F1.tipo = boolean

(18) T.tipo = F.tipo

(19) se op == ‘*’

se F.tipo == inteiro && F1.tipo == byte

|| F.tipo == byte && F1.tipo == inteiro

|| F.tipo == inteiro && F1.tipo == inteiro

|| F.tipo == byte && F1.tipo == byte

entao T.tipo = inteiro

se op == ‘/’

se T.tipo == inteiro && F.tipo == byte

|| T.tipo == byte && F.tipo == inteiro

|| T.tipo == inteiro && F.tipo == inteiro

|| T.tipo == byte && F.tipo == byte

entao T.tipo = inteiro

senao

ERRO(TIPOS INCOPATIVEIS)

se op == ‘and’

se T.tipo != logico || F.tipo != logico

ERRO(TIPOS INCOPATIVEIS)

(20) sinalFlag == ‘+’

(21) sinalFlag == ‘-‘

(22) se sinalFlag == (+||-) && EXPS.tipo != (inteiro) || EXPS.tipo != (byte)

ERRO(TIPOS INCOPATIVEIS)

senao sinalFlag == ‘-‘

T.tipo = inteiro

EXPS.tipo = T.tipo

(23) se op == or

se EXPS.tipo != boolean && T1.tipo != boolean

ERRO(TIPOS INCOPATIVEIS)

senao

EXPS= boolean

se op == '+'

se EXPS.tipo == string && T1.tipo == string

//concatena string

senao se EXPS.tipo == inteiro && T1.tipo == inteiro

|| EXPS.tipo == inteiro && T1.tipo == byte

|| EXPS.tipo == byte && T1.tipo == inteiro

EXPS = inteiro

senao se EXPS.tipo == byte && T1.tipo == byte

EXPS = byte

senao

ERRO(TIPOS INCOPATIVEIS)

se op == '-'

se EXPS.tipo == inteiro && T1.tipo == inteiro

|| EXPS.tipo == inteiro && T1.tipo == byte

|| EXPS.tipo == byte && T1.tipo == inteiro

EXPS = inteiro

senao se EXPS.tipo == byte && T1.tipo == byte

EXPS = byte

senao

ERRO(TIPOS INCOPATIVEIS)

(24) EXP.tipo = EXPS.tipo

(25) se op == ‘==’

se EXPS.tipo == string && EXPS1.tipo == string

|| EXPS.tipo == inteiro && EXPS1.tipo == byte

|| EXPS.tipo == byte && EXPS1.tipo == inteiro

|| EXP.tipo == inteiro && EXPS1.tipo == inteiro

EXPS1.tipo = boolean

senao se op == (‘!=’ || ‘<’ || ‘>’ || ‘<=’ || ‘>=’)

se EXP.tipo == inteiro && EXPS1.tipo == inteiro

|| EXP.tipo == byte && EXPS1.tipo == inteiro

|| EXP.tipo == inteiro && EXPS1.tipo == byte

|| EXP.tipo == byte && EXPS1.tipo == byte

EXP.Tipo = boolean

senao

ERRO(TIPOS INCOPATIVEIS)