Akdemy

Gabriel Garcia

November 24, 2015

1 Alphabet and Lexem Formation Pattern

∑	Padrão de formação dos lexemas	∑	Padrão de formação dos lexemas	∑	Padrão de formação dos lexemas
numero	(d)+	==	"=="	*	"*"
string	"(letraUcaracterUdígitoUsimbolosVálidosExcetoAspas)*"	==	"=="	*	"*"
id	(letraU_)(letraU_Udígito)*	=	"="	;	";"
final	"final"	(	"("	begin	"begin"
int	"int"	)	")"	end	"end"
byte	"byte"	<	"<"	readln	"readln"
string	"string"	>	">"	write	"write"
while	"while"	!=	"!="	writeln	"writeln"
if	"if"	>=	">="	VRAI	"TRUE"
else	"else"	<=	"<="	FAUX	"FALSE"
and	"and"	,	","	boolean	"boolean"
or	"or"	+	"+"	byte	"0x"HH tal que H ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
not	"not"	-	"-"	quebra	'\n'

Figure 1: Alphabet and Lexem Formation Pattern

2 Lexical Analyser Automata

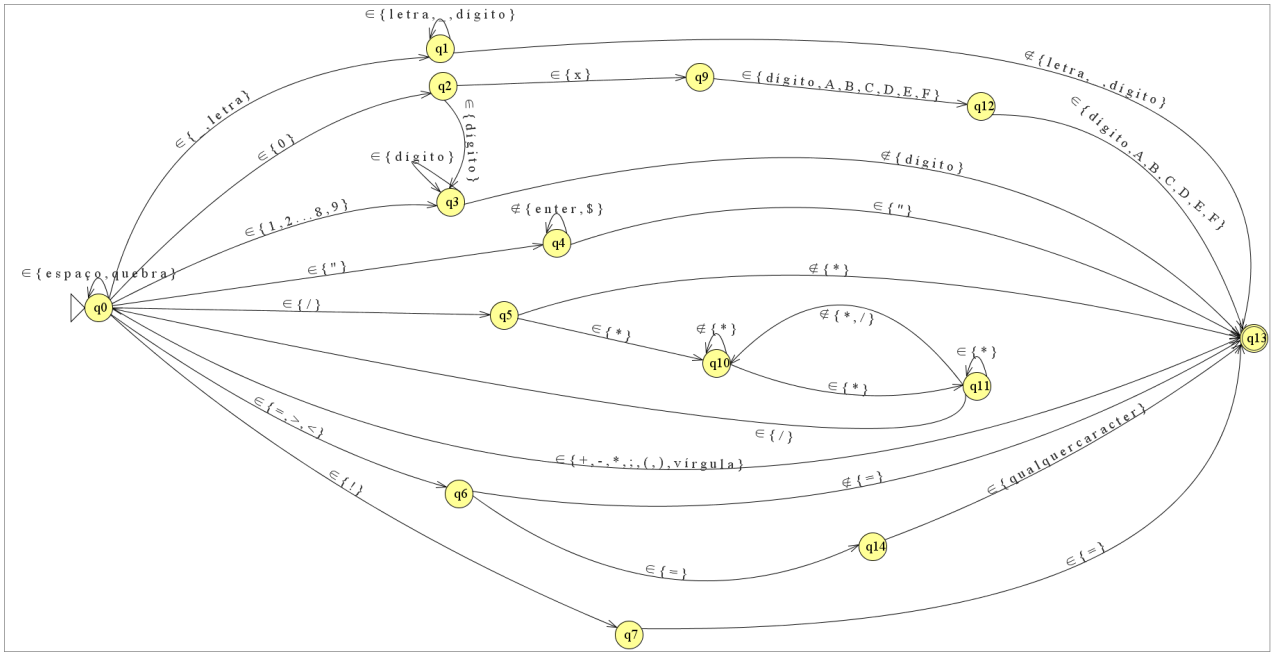

Figure 2: Lexical Analyser Automata

3 Grammar

$S \rightarrow (1)(2)\{(3)D(4)\} * (5)(6)B(7)$

$D \rightarrow (\text{int}(8)|\text{byte}(9)|\text{boolean}(10)|\text{string}(11))\text{id}(12)(13)(14)[= [(+(15)|-(16))](17)\text{const}(18)(19)]$

$\{, \text{id}(12)(13)(14)[= [(+(15)| - (16))](17)\text{const}(18)(19)]\}*;$

$D \rightarrow \text{final id}(20) = [(+(15)| - (16))]\text{const}(21)(22)(23)(24)(25)$

$B \rightarrow \text{begin}\{C\} * \text{end}$

$C \rightarrow \text{id}(26) = E;(27)$

$C \rightarrow \text{while } (28)E(29)(30)(C|B)(31)$

$C \rightarrow \text{if } E(32)(33)(C|B)(34)[else\ (C|B)](35)$

$C \rightarrow;$

$C \rightarrow \text{readln id};(36)(37)$

$C \rightarrow \text{write } E(38)\{, E(39)\}*;$

$C \rightarrow \text{writeln } E(38)\{, E(39)\}*;(40)$

$E \rightarrow F(41)[(< (42)| > (43)| <= (44)| >= (45)| == (46)|! = (47))F(48)]$

$F \rightarrow [(+(49)| - (50))]G(51)\{(+(52)| - (53)|\text{or}(54))G\}*$

$G \rightarrow H(55)\{(*(56)|/(57)|and(58))H(59)\}*$

$H \rightarrow [(60)not]J(61)$

$J \rightarrow$ “(”E“)”

$J \rightarrow \text{const}(62)$

$J \rightarrow \text{id}(63)$

- (1) { “sseg SEGMENT STACK

  byte 4000h DUP(?)
  sseg ENDS

  dseg SEGMENT PUBLIC

  byte 4000h DUP(?)
  ” }

- (2) { S.addr = 0x4000 }
- (3) { D.addr = S.addr }
- (4) { S.addr = D.addr }
- (5) { “dseg ENDS

  cseg SEGMENT PUBLIC
  ASSUME CS:cseg, DS:dseg

  strt:
  ” }
- (6) { S.attrs }
- (7) { “mov ah, 4Ch
  int 21h
  cseg ENDS
  END strt” }
- (8) { D.t = inteiro
  D.addr += 2 }
- (9) { D.t = byte
  D.addr++ }
- (10) { D.t = booleano
  D.addr++ }
- (11) { D.t = string
  D.addr += 0x100 }
- (12) { se id.classe != vazio então
  id.classe = “variavel”
  senão
  erro(“identificador ja declarado”) }
- (13) { id.addr = D.addr }
- (14) { se D.t == byte ou D.t == booleano
  S.addr++
  se não se D.t == inteiro
  S.addr += 2
  se não se D.t == string
  S.addr += 0x100
  fim se }

- (15) { D.s = “-” }
- (16) { D.s = “+” }
- (17) { se D.s == “-” ou D.s == “+” então
  se id.tipo != byte ou id.tipo != inteiro então
  erro(“tipos incompatíveis”)
  fim se
  se id.tipo == byte e D.s == “-” então
  id.tipo = inteiro
  fim se
  fim se }
- (18) { se não (id.tipo == const.tipo ou (id.tipo == inteiro e const == byte)) então
  erro(“tipos incompatíveis”)
  fim se }
- (19) { se id.tipo == string então
  S.attrs += “mov bx, id.addr”
  para cada caractere na string const.lexema faça
  S.attrs += “mov al, caractere”
  S.attrs += “mov DS:[bx], al”
  S.attrs += “add bx, 1”
  fim para cada
  S.attrs += “mov al, 24h”
  S.attrs += “mov DS:[bx], al”
  se não se id.tipo == inteiro então
  S.attrs += “mov ax, D.s+const.lexema”
  S.attrs += “mov DS:[id.addr], ax”
  se não se id.tipo == byte então
  S.attrs += “mov al, const.lexema”
  S.attrs += “mov DS:[id.addr], al”
  se não se id.tipo == booleano então
  S.attrs += “mov al, (const.lexema == TRUE ? FFh : 00h)”
  S.attrs += “mov DS:[id.addr], al”
  fim se }
- (20) { se id.classe != vazio então
  id.classe = “constante”
  senão
  erro(“identificador ja declarado”) }
- (21) { se D.s == “-” ou D.s == “+” então
  se const.tipo != byte ou const.tipo != inteiro então

erro(“tipos incompatíveis”)
fim se
se const.tipo == byte e D.s == “-” então
const.tipo = inteiro
fim se
fim se }

- (22) { id.tipo = const.tipo
  id.addr = D.addr }

- (23) { se const.tipo == string então
  }

- (24) { se D.t == byte ou D.t == booleano
  S.addr++
  se não se D.t == inteiro
  S.addr += 2
  se não se D.t == string
  S.addr += strlen(const.lexema)
  fim se }

- (25) { se id.tipo == string então
  “byte const.lexema$”
  se não se id.tipo == byte então
  “byte const.lexema”
  se não se id.tipo == booleano então
  “byte (const.lexema == TRUE ? FFh : 00h)”
  se não
  “sword D.s+const.lexema”
  fim se }

- (26) { se id.classe == vazio então
  erro(“identificador nao declarado”)
  se não se id.classe == constante então
  erro(“classe de identificador incompativel”)
  fim se }

- (27) { se id.tipo == inteiro então
  se E.tipo == inteiro então
  “mov al, DS:[E.addr]”
  “mov al, DS:[E.addr+1]”
  “mov DS:[id.addr], al”
  “mov DS:[id.addr+1], ah”
  se não
  “mov al, DS:[E.addr]”

“mov ah, 0”
“mov DS:[id.addr], al”
“mov DS:[id.addr+1], ah”
fim se
se não se id.tipo == string então
“mov bx, E.addr”
“mov di, id.addr”
loop = NovoRot
end = NovoRot
“loop:”
“mov cl, DS:[bx]”
“cmp cl, 24h”
“je end”
“mov DS:[di], cl”
“add di, 1”
“add bx, 1”
“jmp loop”
“end:”
“mov DS:[di], cl”
se não
“mov al, DS:[E.addr]”
“mov DS:[id.addr], al”
fim se }

- (28) { loop = NovoRot
end = NovoRot
“loop:” }

- (29) { se E.tipo != booleano então erro(“tipos incompatíveis.”) fim se }

- (30) { “mov al, DS:[E.addr]”
“mov ah, 0”
“cmp ax, 0”
“je end” }

- (31) { “jmp loop”
“end: ” }

- (32) { false = NovoRot
end = NovoRot
se E.tipo != booleano então erro(“tipos incompatíveis.”) fim se }

- (33) { “mov al, DS:[E.addr]”
“mov ah, 0”
“cmp ax, 0”
“je false” }

- (34) { “jmp end”
  “false:” }
- (35) { “end:” }
- (36) { buffer = NovoTemp
  “mov dx, buffer”
  “mov al, 0FFh”
  “mov DS:[buffer], al”
  “mov ah, 0Ah”
  “int 21h”
  “mov ah, 02H”
  “mov dl, 0Dh”
  “int 21h”
  “mov dl, 0Ah”
  “int 21h” }
- (37) { end = NovoRot
  loop = NovoRot
  se id.tipo == string então
  “mov di, id.addr”
  “mov bx, buffer+2”
  “loop: ”
  “mov al, DS:[bx]”
  “cmp al, 0Dh”
  “je end”
  “mov DS:[di], al”
  “add di, 1”
  “add bx, 1”
  “jmp loop”
  “end:”
  “mov al, 24h”
  “mov DS:[di], al”
  se não se id.tipo == byte
  “mov di, buffer+2”
  “mov al, 0”
  “mov cl, 10”
  “loop:”
  “mov bl, DS:[di]”
  “cmp bl, 0Dh”
  “je end”
  “mul cl”
  “add bl, -48”
  “add al, bl”
  “add di, 1”

“jmp loop”
“end:”
“mov DS:[id.addr], al” se não se id.tipo == inteiro então
positive = NovoRot
“mov di, buffer+2”
“mov ax, 10”
“mov cx, 10”
“mov dx, 1”
“mov bl, DS:[di]”
“ jne positive”
“mov dx, -1”
“add di, 1”
“mov bl, DS:[di]”
“cmp bl, 0Dh”
“je end”
“imul cx”
“add bl, -48”
“mov bh, 0”
“add ax, bx”
“add di, 1”
“jmp loop”
“end:”
“pop cx”
“imul cx”
“mov DS:[id.addr], al”
“mov DS:[id.addr+1], ah” }