

Gramática com Expressões Regulares LL(1) da Linguagem L

S → {**Declarar**}* {**Comando**}*

Declarar → “final” “id”[1]“<-” [-][11] const [15][21][22][1]“;” | (int[5] | boolean[6] | byte[7] | string[8]) id[2][10][18] **ListaIds**“;”

ListaIds → [[19][2]Atrib][3] {“,” “id[2][9]” [[17][2]Atrib][3]}*

Atrib → “<-” [-][12] const [16][21][20][1]

Comando → **Atribuicao** | **Repeticao** | **Teste** | **Nulo** | **Leitura** | **Escrita**

Atribuicao → “id”[3][4] “<-” **Expressao** [42]“;”[4]

Repeticao → “while”[23]“(“**Expressao**[43]“)”[24] **Blocowhile** [25]

Blocowhile → “begin”{**Comando**}*“endwhile” | **Comando**

Teste → “if”[19]“(“**Expressao**[43][20]“)” (**Blocoif** | **Comando** [**BlocoElse**])

Blocoif → “begin”{**Comando**}+“endif”“else”[21]“begin”{**Comando**}+[22]“endelse”

BlocoElse → “else”[21] **Comando**[22]

Nulo → “;”

Leitura → “readln”“(“id[3][4][44][26]“)”“;”

Escrita → “write”“(“**ListaExpressoes**“)”“;” | “writeln”“(“**ListaExpressoes**[27]“)”“;”

ListaExpressoes → **Expressao**[45][28] {“,” **Expressao**[45][28]}*

Expressao → **Exp1**[23][16] [(“=”[33][17.1] | “!=”[30][17.2] | “<”[30][17.3] | “>”[30][17.4] | “<=”[30][17.5] | “>=”[30][17.6]) **Exp2** [36][41][18]]

Exp → [- [12] **T1**[24][13] {(“+”[34][14.1] | “-”[31][14.2] | “|”[31][14.3]) **T2** [37][40] [15]}*

T → **F1**[25][9] {(“*”[32][10.1] | “&&”[32][10.2] | “/”[32][35][10.3]) **F2** [38][39][11]}*

F → “!”**F1**[26][8] | (“**Expressao**[27]“)”[7] | [“-”[13] const[21][28][6] | id[3][29][5]

Verificação de unicidade, classes e tipos

[1] {se id.classe != NULO entao ERRO senao id.classe = classe-const}
 [2] {se id.classe != NULO entao ERRO senao id.classe = classe-var}
 [3] {se id.classe = NULO entao ERRO}
 [4] {se id.classe != classe-var entao ERRO}
 [5] {Declarar.tipo = tipo-inteiro}
 [6] {Declarar.tipo = tipo-lógico}
 [7] {Declarar.tipo = tipo-byte}
 [8] {Declarar.tipo = tipo-string}
 [9] {id.tipo = Listaids.tipo}
 [10] {id.tipo = Declarar.tipo}
 [11] {Declarar.isnegativo = true}
 [12] {Atrib.isnegativo = true}
 [13] {F.isnegativo = true}
 [15] {se Declarar.isnegativo = true entao const.tipo = tipo-inteiro}
 [16] {se Atrib.isnegativo = true entao const.tipo = tipo-inteiro}
 [17] {Atrib.tipo = id.tipo}
 [18] {Listaids.tipo = id.tipo}
 [19] {Atrib.tipo = Listaids.tipo}
 [20] {se Atrib.tipo != const.tipo entao ERRO}
 [21] {se const.lexema != NULO e const.lexema >= 0 e const.lexema <= 255 entao
 const.tipo = tipo-byte senao const.tipo = tipo-inteiro}
 [22] {se id.tipo == NULO entao id.tipo = const.tipo senao se id.tipo != const.tipo entao ERRO}
 [23] {Expressao.tipo = Exp.tipo}
 [24] {Exp.tipo = T.tipo}
 [25] {T.tipo = F.tipo}
 [26] {F.tipo = F1.tipo}
 [27] {F.tipo = Expressao.tipo}
 [28] {F.tipo = const.tipo}
 [29] {F.tipo = id.tipo}
 [30] {se Expressao.tipo = tipo-string entao ERRO}
 [31] {se Exp.tipo = tipo-string entao ERRO}
 [32] {se T.tipo = tipo-string entao ERRO}
 [33] {se Expressao.tipo != tipo-string e Expressao.tipo != tipo-inteiro e Expressao.tipo !=
 tipo-lógico e Expressao.tipo != tipo-byte entao ERRO}
 [34] { se Exp.tipo != tipo-string e Exp.tipo != tipo-inteiro e Exp.tipo != tipo-lógico e
 Exp.tipo != tipo-byte e Exp.tipo != tipo-string entao ERRO}
 [35] {se T.tipo = tipo-byte entao T.tipo = tipo-inteiro; T.isdiv = true}
 [36] {se Expressao.tipo != Exp2.tipo e (Expressao.tipo != tipo-inteiro e Exp2.tipo != tipo-
 byte ou Expressao.tipo != tipo-byte e Exp2.tipo != tipo-inteiro) entao ERRO}
 [37] {se Exp.tipo != T2.tipo e (Exp.tipo != tipo-inteiro e T2.tipo != tipo-byte ou Exp.tipo !=
 tipo-byte e T2.tipo != tipo-inteiro) entao ERRO}
 [38] {se T.tipo != F2.tipo e (T.tipo != tipo-inteiro e F2.tipo != tipo-byte ou T.tipo != tipo-byte
 e F2.tipo != tipo-inteiro) entao ERRO}

```

[39] {se (F1.tipo = tipo-inteiro e F2.tipo = tipo-byte) ou (F1.tipo = tipo-byte e F2.tipo = tipo-
inteiro) entao T.tipo = tipo-inteiro}
[40] {se (T1.tipo = tipo-inteiro e T2.tipo = tipo-byte) ou (T1.tipo = tipo-byte e T2.tipo = tipo-
inteiro) entao Exp.tipo = tipo-inteiro}
[41] {Expressao.tipo = tipo-lógico}
[42] {se id.tipo != Expressao.tipo entao ERRO}
[43] {se Expressao.tipo != tipo-lógico entao ERRO}
[44] {se id.tipo != tipo-byte e id.tipo != tipo-inteiro e tipo.tipo != tipo-string entao ERRO}]
[45] {se Expressao.tipo != tipo-inteiro e Expressao.tipo != tipo-byte e Expressao.tipo !=
tipo-string}

```

Geração de código

```

[1] {se const.tipo = tipo-inteiro
    sword const.lexema
senao se const.tipo = tipo-string
    sword const.lexema+"$"
senao se const.tipo = tipo-lógico
    sword const.lexema}

[2] {ListaIds.isatribuicao = true}

[3] {se ListaIds.isatribuicao = false

    se ListaIds.tipo = tipo-inteiro
        assembly.getDeclaracoes().put("sword ?");
senao se ListaIds.tipo = tipo-byte
    assembly.getDeclaracoes().put("byte ?");
senao se ListaIds.tipo = tipo-string
    assembly.getDeclaracoes().put("byte ?");
senao se Lista.tipo = tipo-lógico
    assembly.getDeclaracoes().put("byte ? ");}

[4] {mov ax, DS: [Expressao.end]
    se id.tipo != Expressao.tipo entao  cwd
    mov DS:[id.end], ax}

[5] {F.end = id.end}

[6] {se const.tipo = tipo-string entao
    dseg SEGMENT PUBLIC
    byte "const.lexema$"
    dseg ENDS
    F.end = contator_dados
    contator_dados += 256

    senao

```

```
F.end = NovoTemp
mov regA, const.lexema
mov F.end, regA}
```

[7] {F.end = Expressao.end}

[8] {F.end = NovoTemp
mov regA, F1.end
neg regA
add regA, 1
mov F.end, regA}

[9] {T.end = F1.end}

[10.1] {T.operador = *}

[10.2] {T.operador = &&}

[10.3] {T.operador = /}

[11] {mov ax, DS:[T.end]
mov bx, DS:[T2.end]
se Exp.tipo != tip-inteiro entao
cwd
se F2.tipo != tipo-inteiro entao
mov cx, DS:[ax]
mov ax, DS:[bx]
cwd
mov DS:[bx], ax
mov DS:[ax], cx}

se T.operador = "*" entao
imul bx
senao se T.operador = "/" entao
idiv bx
senao
and ax,bx
T.end = NovoTemp
mov DS:[T.end], ax}

[12] {Exp.negado = true}

[13] {se Exp.negado = true
Exp.end = NovoTemp
mov ax, DS:[T1.end]
neg ax
mov DS:[Exps.end], ax}
{Exp.end = T1.end}

[14.1] {Exp.operador = +}

[14.2] {Exp.operador = -}

[14.3] {Exp.operador = ||}

[15] {mov ax, DS:[Exp.end]
mov bx, DS:[T₂.end]
se Exp.tipo != tipo-inteiro entao
cwd
se T₂.tipo != tipo-inteiro entao
mov cx, DS:[ax]
mov ax, DS:[bx]
cwd
mov DS:[bx], ax
mov DS:[ax], cx}
se Exp.op = “+” então
add ax,bx
senao se Exp.op = “-” então
sub ax,bx
senao
and ax,bx
Exp.end = NovoTemp
mov DS:[Exp.end], ax

[16] {Expressao.end = Exp1.end}

[17.1] {Expressao.op = “=”}

[17.2] {Expressao.op = !=}

[17.3] {Expressao.op = <}

[17.4] {Expressao.op = >}

[17.5] {Expressao.op = <=}

[17.6] {Expressao.op = >=}

[18] {mov ax, DS:[Expressao.end]}
mov bx, DS:[Exp₂.end]
cwd
mov cx, DS:[ax]
mov ax, DS:[bx]
cwd
mov DS:[bx],ax

```

mov DS:[ax],cx}
{cmp ax, bx}
{ RotVerdadeiro = NovoRot }
{ se Exp.op = “=” então
  je RotVerdadeiro
senao se Exp.op = “!=” então
  jne RotVerdadeiro
senao se Exp.op = “<” então
  jl RotVerdadeiro
senao se Exp.op = “>” então
  jg RotVerdadeiro
senao se Exp.op = “>=” então
  jge RotVerdadeiro
senao
  jle RotVerdadeiro }
{ mov al, 0 }
{ RotFim = NovoRot }
{ jmp RotFim }
{ RotVerdadeiro: }
{ mov AL, 0FFh }
{ RotFim: }
{ Exp.end:=NovoTemp }
{ Exp.tipo:=tipo-lógico }
{ mov Exp.end, AL }

```

[19] {RotFalso = NovoRot}
 {RotFim = NovoRot}

[20] {mov ax, DS:[Expressao.end] }
 {cmp ax, 0
 je RotFalso }

[21] {jmp RotFim }
 {RotFalso:}

[22] {RotFim:}

[23] {RotInicio = NovoRot}
 {RotFim = NovoRot}
 {RotInicio:}

[24] { mov ax, DS:[Exp.end] }
 {cmp ax, 0
 je RotFim}

```
[25] {jmp RotInicio}  
      {RotFim:}
```

```
[26] se(id.tipo = tipo-string)  
      "mov ax, 0"  
      "mov cx, 10"  
      "mov dx, 1"  
      "mov bh, 0"  
      "mov bl, ds:[di]"  
      "cmp bx, 2Dh"  
      String rot = rotulo.newRot();  
      "jne " + rot  
      "mov dx, -1"  
      "add di, 1"  
      "mov bl, ds:[di]"  
      rot + " :"  
      "push dx"  
      "mov dx, 0"  
      String rot1 = rotulo.newRot();  
      rot1 + " :"  
      "cmp bx, 0Dh");  
      String rot2 = rotulo.newRot();  
      "je " + rot2  
      "imul cx"  
      "add bx, -48"  
      "add ax, bx"  
      "add di, 1"  
      "mov bh, 0"  
      "mov bl, ds:[di]"  
      "jmp " + rot1  
      rot2 + " :"  
      "pop cx"  
      "imul cx"  
      "mov DS:[" + id_aux.getEnd() + "], ax"  
  }else{  
      "mov si, " + id_aux.getEnd()  
      String rotString = rotulo.newRot();  
      rotString + " :"  
      "mov al, ds:[di]"  
      "cmp al, 0dh ; verifica fim string"  
      String rot2 = rotulo.newRot();  
      "je " + rot2 + " ; salta se fim string"  
      "mov ds:[si], al ; próximo caractere"  
      "add di, 1 ; incrementa base"  
      "add si, 1");  
      "jmp " + rotString + " ; loop"  
      rot2 + " :");
```

```

"mov al, 024h ; fim de string"
"mov ds:[si], al ;grava $"
} // End else

```

```

[27] {"mov ah, 02h"
      "mov dl, 0Dh"
      "int 21h"
      "mov DL, 0Ah"
      "int 21h"}

```

```

[28] if(Exp_tipo.equals("tipo_string")){
      "mov dx, " + Exp_end);
      "mov ah, 09h");
      "int 21h");
    }else{
      "mov ax, DS:[" + Exp_end + "]");
      "mov di, " + stringEnd + " ; end. string temp.");
      "mov cx, 0 ;contador");
      "cmp ax,0 ;verifica sinal");
      String rot = rotulo.newRot();
      "jge " + rot + " ;salta se numero positivo");
      "mov bl, 2Dh ;senao, escreve sinal ");
      "mov ds:[di], bl");
      "add di, 1 ;incrementa indice");
      "neg ax ;toma modulo do numero");
      rot + ":"");
      "mov bx, 10 ;divisor");
      String rot1 = rotulo.newRot();
      rot1 + ":"");
      "add cx, 1 ;incrementa contador");
      "mov dx, 0 ;estende 32bits p/ div.");
      "idiv bx ;divide DXAX por BX");
      "push dx ;empilha valor do resto");
      "cmp ax, 0 ;verifica se quoc. 0");
      "jne " + rot1 + " ;se nao 0, continua");
      String rot2 = rotulo.newRot();
      rot2 + ":"");
      "pop dx ;desempilha valor");
      "add dx, 30h ;transforma em caractere");
      "mov ds:[di],dl ;escreve caractere");
      "add di, 1 ;incrementa base");
      "add cx, -1 ;decrementa contador");
      "cmp cx, 0 ;verifica pilha vazia");
      "jne " + rot2 + " ;se nao pilha vazia, loop");
      "mov dl, 024h ;fim de string");
      "mov ds:[di], dl ;grava '$");
      "mov dx, " + stringEnd);
      "mov ah, 09h");
    }

```



```
    "int 21h");  
} // End else
```