Gramática Linguagem L

S -> {D ;} “main” { C[6] }$^{+}$ “end”

C -> “readln” “(“ id[1.1] “)” “;”[2 4 1] |
“write” “(” E[2 1.2] {“,” E[2 1.2]}“)” “;”[3] |
“writeln” “(” E[2] { “,” E[2] } “)” “;”[3 4] |
“while” “(” E[3] “)” [5] (C | “begin” {C} “end”) |
“if” “(“ E[3] “)” [5] “then”(C | “begin” {C} “end”)
[“else” [6.1](C | “begin” {C} “end”)] | id[1 4] “=” E [5 7] “;” |
“.”
;

D -> (“int” | “boolean” | “byte” | “string”) id[6 8] D_1[5.1 9] |
“const” id [6 10] “=” [“-”] constante_valor [21.4 22]

D_1 -> “=” E [11] D_2 | D_2

D_2 -> {“,” id [6 8] [“=” E [5 12]] }

E -> E_1[13] [(“<” | “>” | “!=” | “==” | “>=” | “<=”)[9] $E_{1.F}$[10 14]]

E_1 ->[“+” | “-”[15.1]] E_2[15 11 16]{ (“+”| “-” | “or”)[12] $E_{2.F}$ [10.1 18 15.2]}

E_2 -> E_3[13.1] {(“*” | “/” | “and”)[13] $E_{3.F}$ [10.2 20]}

E_3 -> “not” $E_{3.F}$ [14 21]| “(“ E[21.1] “)” | id[21.2] | constante_valor[21.3]

Ações semânticas:

[1] - { se (id.classe == vazio) == ERRO   -> identificador nao declarado}

[1.1] - { se (id.classe == vazio) == ERRO   -> identificador nao declarado
se id.classe == boolean} == ERRO -> tipos incompativeis
se (id.classe == constante == ERRO) ->tipo de classe inválido}

[1.2 -] { se (id.classe == vazio) == ERRO   -> identificador nao declarado
se id.classe == boolean} == ERRO -> tipos incompativeis }

[2] - {se (id.tipo != valor_lido.tipo) == ERRO} -> tipos incompatíveis

[3] - {se (E.tipo != boolean) == ERRO} -> tipos incompatíveis

[4] - {se (id.classe != variável) == ERRO} -> classe de identificador incompatível

[5] - {se (! (id.tipo == inteiro && E.tipo == byte) && (id.tipo != E.tipo)) == ERRO} -> tipos incompatíveis

[5.1] - {se (! (id.tipo == inteiro && D_1.tipo == byte) && (id.tipo != D_1.tipo)) = ERRO} -> tipos incompatíveis

[6] - {se (id.tipo != vazia) == ERRO}   -> identificador ja declarado

[9] - {switch()
case < : se(E_1.tipo == (string || boolean)) == ERRO

break;
case > : se(E_1.tipo == (string || boolean)) == ERRO
break;
case <= : se(E_1.tipo == (string || boolean)) == ERRO
break;
case >= : se(E_1.tipo == (string || boolean)) == ERRO
break;} -> tipos incompatíveis

[10] - se (! (E_1.tipo == inteiro && $E_{1.F}$.tipo == byte) && (! (E_1.tipo == byte && $E_{1.F}$.tipo == inteiro) && (E_1.tipo != $E_{1.F}$.tipo)) == ERRO
-> tipos incompatíveis

[10.1] - se (! (E_2.tipo == inteiro && $E_{2.F}$.tipo == byte) && (! (E_2.tipo == byte & $E_{2.F}$.tipo == inteiro) && (E_2.tipo != $E_{2.F}$.tipo)) == ERRO
-> tipos incompatíveis

[10.2] - se (! (E_3.tipo == inteiro && $E_{3.F}$.tipo == byte) && (! (E_3.tipo == byte & $E_{3.F}$.tipo == inteiro) && (E_3.tipo != $E_{3.F}$.tipo)) == ERRO
-> tipos incompatíveis

[11] - se (E_1.tipo == inteiro && E_2.tipo != (inteiro || byte)) == ERRO -> tipos incompatíveis

[12] - {switch()
case + : se(E_2.tipo == (boolean)) == ERRO
break;
case - : se(E_2.tipo == (string || boolean)) == ERRO
break;
case or : se(E_2.tipo == (string) == ERRO
break;} -> tipos incompatíveis

[13] - {switch()

case * : se(E_2.tipo == (string || boolean)) == ERRO
break;
case / : se(E_2.tipo == (string || boolean)) == ERRO
break;
case and : se(E_2.tipo == (string) == ERRO
break;} -> tipos incompatíveis

[14] - se ($E_{3.F}$.tipo != boolean) == ERRO -> tipos incompatíveis

[1] - { id.val = valor lido}

[2] - { write.val += (string)E.val}

[3] - { Imprime write.val }

[4] - { Imprime quebra de linha }

[5] - { flag para realizar ações semânticas = E.val}

[6] - { flag para realizar ações semânticas = true }

[6.1] - { se !(flag para realizar ações semânticas)
flag para realizar ações semânticas = true }

[7] - { id.val = E.val }

[8] - {switch(){
case int: id.tipo
case string: id.tipo = string;
break}

id.classe = variável}

[9] - { id.val = D_1.val }

[10] - { id.classe = constante }

[11] - { D_1.tipo = E.tipo
D_1.val = E.val }

[12] - { id.val = E.val }

[13] - { E.tipo = E_1.tipo }

[13.1] - { E_2.tipo = E_3.tipo }

[14] - { E.tipo = boolean
{switch()
case < : E.val = (E_1.val < $E_{1.F}$.val)
break;
case > : E.val = (E_1.val > $E_{1.F}$.val)
break;
case <= : E.val = (E_1.val <= $E_{1.F}$.val)
break;
case >= : E.val = (E_1.val >= $E_{1.F}$.val)
break;
case ==: E.val = (E_1.val == $E_{1.F}$.val)
break;
case !=: E.val = (E_1.val != $E_{1.F}$.val)
break;}}

[15] - { E_1.tipo = inteiro }

[15.1] - {flag_sub = true}

[15.2] - { flag_sub = false}

[16] - {se (E_1.tipo == inteiro) E_2.tipo = inteiro }
[18] - { switch()
case + : E_1.val = (E_2.val + $E_{2.F}$.val)
break;
case - : E_1.val = (E_2.val - $E_{2.F}$.val)
break;
case or : E_1.val = (E_2.val || $E_{2.F}$.val)
break;

if ((E_2.tipo == inteiro && $E_{2.F}$.tipo == byte) ||
(E_2.tipo == inteiro && $E_{2.F}$.tipo == byte)){
E_1.tipo = inteiro
}
else E_1.tipo = $E_{2.F}$.tipo }

[20] - { switch()
case * : E_2.val = (E_3.val * $E_{3.F}$.val)
break;
case / : E_2.val = (int)(E_3.val / $E_{3.F}$.val)
break;
case and : E_2.val = (E_3.val && $E_{3.F}$.val)
break; }

E_2.val = E_3.val;

[21] - {E_3.tipo = $E_{3.F}$.tipo
E_3.val = !($E_{3.F}$.val) }

[21.1] - { E_3.tipo = E.tipo
E_3.val = E.val }

[21.2] - { E_3.tipo = id.tipo
E_3.val = id.val }

[21.3] - { E_3.tipo = constante_valor.tipo
E_3.val = constante_valor.val }

[21.4] - { D = constante_valor.tipo
D = constante_valor.val]

[22] - { id.tipo = constante_valor.tipo
id.val = constante_valor.val}