GRAMÁTICA LINGUAGEM L (TRADUÇÃO DE TIPOS)

S → { D } "main" { C } "end"

D → ("integer" [1] | "boolean" [2] | "byte" [3]
| "string" [4]) "id" [5] ["=" ["-" [6]] "valor" [7]]
{ "," "id" [5] ["=" ["-" [6]] "valor" [7]] } ";" | "const" "id" "=" ["-" [6]] "valor" [8] ";"

[1] { tipo = inteiro }

[2] { tipo = boolean)

[3] { tipo = byte)

[4] { tipo = string)

[5] { se id.classe == vazio então

id.tipo = tipo

id.classe = classe-var

senão ERRO)

[6] { flag_negativo = true }

[7]{ se valor.valor >= 256 e valor.valor < 0 então

valor.tipo = inteiro

se valor.valor < 256 então

valor.tipo = byte

se id.tipo != valor.tipo então

se id.tipo == byte e valor.tipo == inteiro ERRO

senão (flag_negativo e (id.tipo == inteiro) e (valor.tipo == inteiro ou valor.tipo == byte

id.tipo = inteiro

senão se flag_negativo e (valor.tipo == byte) então

ERRO

senão ERRO }

[8] { se id.classe == vazio então

se flag_negativo e (id.tipo == inteiro ou idr.tipo == byte)

e (valor.tipo == inteiro ou valor.tipo == byte) então

id.classe = classe-const

senão ERRO)

senão ERRO)

C → A | B | "readln" "(" "id"[9] ")" ";"
| ("write" | "writeln") "(" EXP [10] [, EXP [10]] ")" ";"
| "id" [11] "=" EXP [12] ";" | ";"

[9] { se id.classe == vazio então ERRO

senão id.classe == classe-const || id.tipo == booleano então ERRO)

[10] { se EXP.tipo == booleano então ERRO)

[11] { se id.classe == vazio então ERRO

senão id.classe == classe-const então ERRO)

[12] { ! (id.tipo == inteiro e EXP.tipo == byte)

se id.tipo != EXP.tipo então

ERRO)

A → "while" "(" EXP [13] ")" (C | "begin" { C } "end")

B → "if" "(" EXP [13] ")" "then" (C | "begin" { C } "end") ["else" (C | "begin" { C } "end")]

[13] { se EXP.tipo != booleano então ERRO}

EXP → E [14] [("<" | ">" | "==" [15] | "!=" | "<=" | ">=") E1 [16]]

[14] { EXP.tipo = E.tipo }

[15] { condcao_igualdade = verdadeiro }

[16] { se E.tipo != E1.tipo

se ((E.tipo == inteiro ou E.tipo == byte) e (E1.tipo == inteiro ou E1.tipo == byte)

E.tipo = booleano

ERRO senão se E.tipo != inteiro ou E.tipo != byte então

se E.tipo == string e condicao_igualdade ou (E.tipo == inteiro ou E.tipo == byte)

então EXP.tipo = booleano

senão ERRO

}

E → ["+" [17] | "-" [17]] T [18] { ("+" [19] | "-" [20] | "or" [21]) T1 [22] }

[17] { condicao_sinal = verdadeiro }

[18] { se condicao_sinal então

se T.tipo != byte E T.tipo != inteiro então

ERRO

senão E.tipo = T.tipo}

[19] { condicao_soma_concatenacao = verdadeiro }

[20] { condicao_subtracao = verdadeiro }

[21] { condicao_ou = verdadeiro }

[22] { se (condicao_soma_concatenacao) então

se (T.tipo == booleano ou T1.tipo == booleano) então ERRO

senão se (condicao_subtracao) então

se (T.tipo != inteiro e T.tipo != byte ou T1.tipo != inteiro e T1.tipo != byte) então ERRO

senão T.tipo = T1.tipo = inteiro

senão se (condicao_ou) então

se (T.tipo != booleano ou T1.tipo != booleano) então ERRO

}

T → F [23] { ("*" [24] | "/" [25] | "and" [26]) F1 [27] }

[21] { T.tipo = F.tipo }

[22] { condicao_multiplicacao = verdadeiro }

[23] { condicao_divisao = verdadeiro }

[24] { condicao_and = verdadeiro }

[25] { se F.tipo == F1.tipo então

se condicao_multiplicacao então

se T.tipo != inteiro e T.tipo != byte então ERRO

senão T.tipo = inteiro

senão se condicao_divisao então

se T.tipo != inteiro ou T.tipo != byte então ERRO

senão se T.tipo != byte então

T.tipo = inteiro

T1.tipo = inteiro

senão se condicao_and então

se T.tipo != booleano então ERRO

}

F → "not" F1 [28] | "(" EXP [29] ")" | "id" [30] | "valor" [31]

[26] { se F1.tipo != booleano então ERRO }

[27] { F.tipo = EXP.tipo }

[28] { se id.classe != vazia então F.tipo = id.tipo senão ERRO }

[29] { F.tipo = valor.tipo }