S -> {(Declaracao|Bloco_de_comandos)} EOF

Declaracao -> {1} {2} (int|float) {3} id [{4} {5} <-[- {6}]valor {7}] {8} {9} {,id {8} {1} [{4} {5} <-[- {6}]valor {7}] {9} } ;

| {1} {2} (string|char) {10} id {8} [{4} <-valor {7}] {11} {,id {8} {1} [{4} <-valor {7}] {11} } ;

| {5} {2} const id {12} = [- {6}] {13} valor {14} ;

Bloco_de_comandos -> Comando | “{“ {Bloco_de_comandos} “}”

Comando -> id {15} ["[" Expressao {16} "]"] <- [-] {17} expressão {18} ;

| While expressão {19} {20} bloco_de_comandos {21}

| if {22} expressão {19} {23} Bloco_de_comandos {24} [else Bloco_de_comandos] {25}

| ;

| readln “(“ id {15} “)”{1} ;

| (write|writeln) {26} “(“ expressão {27} {,expressão {27} } “)”;

Expressao -> F {28} [([!]= | <[=] | >[=]) {29} F1 {30}]

F -> E {31} { (- | + | ”|””|”) {29} E1 {32} }

E -> D {33} { (* | / | div | mod | &&) {29} D1 {34} }

D -> {!} {35} C {36}

C -> B {37} | { (int | float) {38} ”(“ [-] {39} Expressao {40} ”)” }+

B -> A {41} | {“(“ Expressao {42} “)” {43} }+

A -> id {44} ["["expressão {45} "]"] {46} | valor {47}

{1}

definido = false;

{2}

Section .data

{3}

Se lexema == int então

	id.tipo = inteiro

senão

	id.tipo = real

{4}

definido = true;

{5}

Negativo = false;

```
{6}
Negativo = true;
{7}
se valor.tipo != id.tipo então
        ERRO
{8}
registro = pesquisa(id.lexema)
Se registro != null então
        ERRO
Senão
        Inserir(lexema, token, tipo, classe)
{9}
Se definido então
        “dd “
        Se negativo então
                “-“
        se id.tipo == inteiro entao
                dd valor.lex
        senão
                dd valor.lex
senão
        se id.tipo == inteiro então
                resd 1
        senão
                resd 1
memoriaDados += 4
{10}
Se lexema == char então
        id.tipo = caractere
Senão
        id.tipo = string
{11}
Se definido então
        se id.tipo == caractere entao
```

```
                db valor.lex
                memoriaDados += 1
        senão
                db valor.lex,0
                memoriaDados += valor.lex.tam - 1
senão
        se id.tipo == caractere então
                resb 1
                memoriaDados += 1
        senão
                resb 100h
                memoriaDados += 256
{12}
Registro = pesquisa(id.lexema)
If registro == null
        ERRO
Lexema = id.lexema
{13}
Inserir(lexema, id, classe-const, valor.tipo)
{14}
Se valor.tipo == inteiro entao
        “dd “
        se negativo então
                “-“
        valor.lex
        memoriaDados += 4
Senão se valor.tipo == float então
        “Dd “
        se negativo então
                “-“
        valor.lex
        memoriaDados += 4
Senão se valor.tipo == caractere então
        db valor.lex
```

memoriaDados += 1

Senão

db valor.lex,0

memoriaDados += valor.lex.tam - 1

{15}

Registro = pesquisa(id.lexema)

Se registro == null então

ERRO

senão se registro.classe == classe-const então

ERRO

Senão

Id.tipo = registro.tipo

{16}

se id.tipo != string ou expressão.tipo != inteiro então

ERRO

{17}

se reg.token == "-"

Negativo = true

senao

Negativo = false

{18}

Se (negativo && reg.tipo != inteiro && reg.tipo != real) || (se expressão.tipo != id.tipo && id.tipo != real && expressão.tipo != inteiro) então

ERRO

se expressão.tipo == inteiro entao

mov eax, [M+expressao.end]

mov id.end, [M+eax]

senão se expressão.tipo == real entao

movss rax, [M+expressão.end]

movss id.end, [M+rax]

senão se expressão.tipo == caractere entao

mov al, [M+expressão.end]

mov id.end, [M+al]

senão se expressão.tipo == string entao

rotInicio = novoRot

```
        rotFim = novoRot
        mov al, 0
        mov bl, expressão.tamanho
        mov rdi, M+id.end
        mov rsi, M+Expressao.end
        rotInicio:
        cmp al, bl
        je rotFim
        mov cl, [rsi]
        mov [rdi], cl
        add al, 1
        add rdi, 1
        add rsi, 1
        jmp rotInicio
        rotFim:
{19}
se expressão.tipo != logica então
        ERRO
{20}
rotInicio = novoRot
rotFim = novoRot
rotInicio:
mov eax, [M+expressão.end]
cmp eax, 0
je rotFim
{21}
Jmp rotInicio
rotFim:
{22}
rotFalso = novoRot
rotFim = novoRot
{23}
Mov eax, [M+expressão.end]
Cmp eax, 0
```

```
Je rotFalso
{24}
Jmp rotFim
rotFalso:
{25}
rotFim:
{26}
Se reg.lex == write entao
        novaLinha = false
senao
        novaLinha = true
{27}
Se expressao.tipo == inteiro ou expressão.tipo == logica então
Senão se expressão.tipo == real então
Senão se expressão.tipo == caractere então
Senão se expressão.tipo == string então
Senão // logica
{28}
Expressao.end = F.end
Expressao.tam = F.tam
{29}
Operador = reg.token
{30}
se ( F.tipo == string ou F1.tipo == string )  então
        se operador != = entao
                ERRO
        Senão
                rotInicio = novoRot
                rotVerdadeiro = novoRot
                rotFalso = novoRot
                rotFim = novoRot
                mov al, 0
                mov bl, F.tam
                mov rdi, M+F.end
```

```
                mov rsi, M+F1.end
                rotInicio:
                cmp al, bl
                je rotVerdadeiro
                mov cl, [rsi]
                mov dl, [rdi]
                cmp cl, dl
                jne rotFalso
                add al, 1
                add rdi, 1
                add rsi, 1
                jmp rotInicio
                rotVerdadeiro:
                mov eax, 1
                jmp rotFim
                rotFalso:
                mov eax, 0
                rotFim:
                mov [M+Expressao.end], eax
                Expressao.tipo = logica
Senão se F.tipo == “” ou F1.tipo == “” então
        ERRO
Senao  // tipos inteiros, reais ou caracteres
        Se F.tipo == caractere ou F1.tipo == caractere entao
                Se F.tipo != F1.tipo então
                        ERRO
                Senao
                        Mov al, [M+F.endereco]
                        Mov bl, [M+e1.endereco]
                        Cmp al, bl
        Se F.tipo == real ou F1.tipo == real entao
                Se F.tipo == real então
                        Movss xmm0, [M+F.endereco]
                senao
```

```
                Mov eax, [M+f.endereco]
                cdqe
                Cvtsi2ss xmm0, rax
        Se e1.tipo == real então
                Movss xmm1, [M+e1.endereco]
        senao
                Mov eax, [M+e1.endereco]
                Cdqe
                Cvtsi2ss xmm1, rax
        Comiss xmm0, xmm1
Senao // comparação entre inteiros
        Mov eax, [M+F.endereco]
        Mov ebx, [M+F1.endereco]
        Cmp eax, ebx
rotVerdadeiro = novoRot
se operador == = então
        je rotVerdadeiro
Senao se operador == != então
        Jne rotVerdadeiro
Senão se operador == < então
        Jb rotVerdadeiro
Senão se operador == <= então
        Jbe rotVerdadeiro
Senão se operador == > então
        Jg rotVerdadeiro
Senão // operador >=
        Jge rotVerdadeiro
Mov eax, 0
rotFim = novoRot
jmp rotFim
rotVerdadeiro:
mov eax, 1
rotFim:
expressão.end = novoTemp
```

```
        expressão.tipo = logico
        mov [qword M+expressão.end], eax
        Expressão.tipo = logica

{31}
F.end = E.end
F.tam = E.tam
{32}
se operador == || então
        se ( E.tipo != logica ou E1.tipo != logica ) então
                ERRO
        senao
                F.tipo = logica
                Mov eax, [M+E.end]
                Mov ebx, [M+E1.end]
                Add eax, ebx
                Cmp eax, 0
                rotFalso = novoRot
                rotFim = novoRot
                Je rotFalso
                Mov eax, 1
                Jmp rotFim
                rotFalso:
                mov eax, 0
                rotFim:
Senão se ( E.tipo != inteiro e E.tipo != real ) ou ( E1.tipo != inteiro e E1.tipo != real ) então
        ERRO
Senao se E.tipo == real ou E1.tipo == real
        Se F.tipo == real então
                Movss xmm0, [M+F.endereco]
        senao
                Mov eax, [M+f.endereco]
                cdqe
                Cvtsi2ss xmm0, rax
```

```
        Se e1.tipo == real então
                Movss xmm1, [M+e1.endereco]
        senao
                Mov eax, [M+e1.endereco]
                Cdqe
                Cvtsi2ss xmm1, rax
        Se operador == “+” então
                Addss xmm0, xmm1
        Senão // operador –
                Subss xmm0, xmm1
        Movss [M+f.endereco], xmm0
        F.tipo = real
Senao
        F.tipo = inteiro
        Mov eax, [M+f.endereco]
        Mov ebx, [M+e1.endereco]
        Se operador == “+” então
                Add eax, ebx
        Senão // operador –
                Sub eax, ebx
        Mov [M+f.endereco], eax
{33}
E.tipo = D.tipo
E.end = D.end
E.tam = D.tam
{34}
se operador == && então
        se ( E.tipo != logica ou D1.tipo != logica ) então
                ERRO
        Senão
                E.tipo = logica
                &&
                Mov eax, [M+D.end]
                Mov ebx, [M+D1.end]
```

```
                Add eax, ebx
                Cmp eax, 2
                rotFalso = novoRot
                rotFim = novoRot
                Jne rotFalso
                Mov eax, 1
                Jmp rotFim
                rotFalso:
                mov eax, 0
                rotFim:
Senão se operador == div ou operador == mod então
        se ( E.tipo != inteiro ou  D1.tipo != inteiro) então
                ERRO
        Senão
                E.tipo = inteiro
                Mov eax, [M+e.endereco]
                Mov ebx, [M+d1.endereco]
                cdq
                Idiv ebx
                Se operador == div então
                        Mov [M+e.endereco], eax
                Senão
                        Mov [M+e.endereco], edx
Senão se ( E.tipo != inteiro e E.tipo != real ) ou ( D1.tipo != inteiro ou D1.tipo != real ) então
        ERRO
Senao se E.tipo == real ou D1.tipo == real entao
        Se e.tipo == real então
                Movss xmm0, [M+e.endereco]
        senao
                Mov eax, [M+e.endereco]
                cdqe
                Cvtsi2ss xmm0, rax
        Se d1.tipo == real então
                Movss xmm1, [M+d1.endereco]
```

```
        senao
                Mov eax, [M+d1.endereco]
                Cdqe
                Cvtsi2ss xmm1, rax
        Se operador == * então
                mulss xmm0, xmm1
                Movss [M+e.endereco], xmm0
        Senão // operador /
                Divss xmm0, xmm1
                movss [M+e.endereco], xmm0
        E.tipo = real
Senao
        E.tipo = inteiro
        Se operador == * então
                Mov eax, [M+e.endereco]
                Mov ebx, [M+d1.endereco]
                Imul ebx
                Mov [M+e.endereco], eax
        Senão
                Mov eax, [M+e.endereco]
                Cdqe
                Cvtsi2ss xmm0, rax
                Mov eax, [M+d1.endereco]
                Cdqe
                Cvtsi2ss xmm1, rax
                Divss xmm0, xmm1
                movss [M+e.endereco], xmm0
{35}
Se reg.token == ! entao
        Negado = true
Senão
        Negado = false
{36}
se operador == ! então
```

```
        se C.tipo != logica então
                ERRO
        Senão
                D.tipo = logica
Senão
        D.tipo = C.tipo
Se negado então
        d.end = novoTemp
        mov eax, [qword M+C.end]
        neg eax
        add eax, 1
        mov [qword M+d.end], eax
senao
        D.end = C.end
        D.tam = C.tam
{37}
C.tipo = B.tipo
C.end = B.end
C.tam = B.tam
{38}
Se lexema == int então
        C.tipo = inteiro
Senão
        C.tipo = real
{39}
Se reg.lex == - entao
        Negado = true
Senao
        Negado = false
{40}
se expressão != inteiro e expressão != real então
        ERRO
Se c.tipo != expressão.tipo
        Se expressao.tipo == inteiro então
```

```
                Mov rax, [M+expressão.end]
                Se negado então
                        Neg rax
                Cvtsi2ss xmm0, rax
                C.end = novotemp(real)
                Movss [M+c.end], xmm0
        Senão // real
                Movss xmm0, [M+expressão.end]
                Cvtss2si rax, xmm0
                c.end = novotemp(inteiro)
                se negado então
                        neg rax
                mov [M+c.end], rax
Senão
c.end = expressão.end
{41}
B.tipo = A.tipo
B.end = A.end
B.tam = A.tam
{42}
B.tipo = expressão.tipo
{43} Fatores p.11
B.end = Expressao.end
B.tam = Expressao.tam
{44}
Registro = pesquisa(id.lexema)
Se registro == null então
        ERRO
Senão
        A.tipo = registro.tipo
acessoVetor = false
{45}
acessoVetor = true
se A.tipo != string ou expressao.tipo != inteiro então
```

```
        ERRO
Senão A.tipo = caractere
{46} Fatores F -> 12 p.12    /    Acesso a elementos de um vetor p.12
Se acessovetor então
        A.end = novotemp(id.tipo)
        mov rax, [M+exp.end]
        add rax, M+id.end
        mov rbx, [M+rax]
        mov [M+A.end], rbx
Senão
        A.end = id.end
        A.tam = id.tam
{47}
Se valor.tipo == string então
        Section .data
        Db valor.lex,0
        Section .text
        A.end = memoriaDados
        memoriaDados += valor.tam-1
        A.tam = valor.tam-1
Senão se valor.tipo == real então
        Section .data
        Db valor.lex
        Section .text
        A.end = memoriaDados
        memoriaDados += 4
        A.tam = valor.tam
Senão se valor.tipo == inteiro entao
        A.end = novoTemp(inteiro)
        Mov eax, valor.lex
        Mov [M+A.end], eax
Senao // caractere
        A.end = novoTemp(caractere)
        Mov al, valor.lex
```

Mov [M+A.end], al

A.tipo = valor.tipo