




#### Legenda:

-  → Linha excluída da especificação
-  → Linha alterada na especificação
-  → Linha adicionada na especificação

A gramática abaixo foi escrita em uma versão de E-BNF seguindo as seguintes convenções:

- 1 - variáveis da gramática são escritas em letras minúsculas sem aspas.
- 2 - Tokens são escritos entre aspas simples
- 3 - símbolos escritos em letras maiúsculas representam o lexema de um *token* do tipo especificado.
- 4 - o símbolo '|' indica produções diferentes de uma mesma variável.
- 5 - o operador [ ] indica uma estrutura sintática opcional.
- 6 - o operador { } indica uma estrutura sintática que é repetida zero ou mais vezes.

#### Regras para identificadores:

- 1 - Pode-se utilizar: números, letras maiúsculas, letras minúsculas e *underscore* (sublinhado).
- 2 - O primeiro caractere deve ser sempre uma letra ou o *underscore*.
- 3 - Não são permitidos espaços em branco e caracteres especiais (ex.: @, \$, +, -, %, ! etc.).
- 4 - Não podemos usar palavras reservadas (palavras que sejam um *token* da linguagem).
- 5 - Identificadores numéricos podem ser do tipo inteiro ou real. A diferença entre eles é a parte decimal dos números reais, separada por uma vírgula.

#### Regras para comentários:

- 1 - A linguagem aceita comentários de linha indicados por //
- 2 - A linguagem aceita comentários de bloco (possivelmente de múltiplas linhas) indicados por /\* e \*/
- 3 - Os comentários de bloco podem ser aninhados.
- 4 - Os comentários de bloco devem ser aninhados e balanceados.

#### Regras para strings:

- 1 - As strings serão escritas entre aspas simples ' e '.
- 2 - Uma string não pode conter uma quebra de linha.
- 3 - Perceba que não existe um tipo string na linguagem, mas sim um tipo caractere. Logo, se quisermos declarar uma string de tamanho 3, fazemos: caractere : var\_str [3];

#### Regras para vetores:

- 1 - Os vetores são declarados da seguinte forma: tipo : nome\_vetor [tam];
- 2 - Os índices dos vetores vão de 0 a tam-1 (isso também vale para strings).
- 3 - Não existem vetores com mais de uma dimensão (e se perguntar o que significa isso perde meio ponto).

#### Regras para expressões lógicas:

- 1 - Os operadores >, <, >= e <= não podem ser usados com operandos que sejam expressões lógicas.
- 2 - Os operadores = e <> não podem ser usados em cadeia. Ex.: a = b = c
- 3 - Uma cadeia com os operadores = e <> pode ser obtida através do uso de parênteses. Ex.: (a = b) = (c = d)

A semântica da linguagem segue a semântica de C. Em caso de dúvida consulte o professor antes de fazer bobagens.

```
programa : 'programa' ID 'inicio' {declaracao} {comando} 'fim' '.'
programa : 'programa' ID 'inicio' {declaracao} {comando} 'fim.'
```

```
declaracao : tipo ':' {var ','} var ';'
            | 'const' ID valor ';'

```

```
tipo : 'real' | 'inteiro' | 'caractere'
```

```
var : ID | ID '[' N_INT '']
```

```
valor : STRING | N_INT | N_REAL
```

```
comando : var '<-' exp ';'
        | 'leia' '(' {var ','} var ')' ';'
        | 'escreva' '(' {exp ','} exp ')' ';'

```

```
        | 'se' '(' exp-logica ')' 'entao' {comando ';' } comando ';' ['senao'
{comando ';' } comando ';' ] 'fim se' ';'

```

```
        | 'se' '(' exp-logica ')' 'entao' {comando} comando ['senao' {comando}
comando] 'fim se' ';'

```

```
        | 'avalie' '(' exp ')' { 'caso' valor ':' {comando ';' } comando ';' }
['senao' ':' {comando ';' } comando ';' ] 'fim avalie' ';'

```

```
        | 'avalie' '(' exp ')' { 'caso' valor ':' {comando} comando } ['senao' ':'
{comando} comando] 'fim avalie' ';'

```

```
        | 'enquanto' '(' exp-logica ')' 'faca' {comando ';' } comando ';' 'fim
enquanto' ';'

```

```
        | 'enquanto' '(' exp-logica ')' 'faca' {comando} comando 'fim enquanto'
';'

```

```
        | 'repita' {comando ';' } comando ';' 'ate' '(' exp-logica ')' ':'
        | 'repita' {comando ';' } comando ';' 'ate' '(' exp-logica ')' ';'

```

```
        | 'repita' {comando} comando 'ate' '(' exp-logica ')' ';'

```

```
        | 'para' var 'de' N_INT 'ate' N_INT 'faca' {comando ';' } comando ';'
'fim para' ';'

```

```
        | 'para' var 'de' N_INT 'passo' N_INT 'ate' N_INT 'faca' {comando ';' }
comando ';' 'fim para' ';'

```

```
        | 'para' var 'de' N_INT 'passo' N_INT 'ate' N_INT 'faca' {comando}
comando 'fim para' ';'

```

```
exp : valor
```

```
    | var
```

```
    | '(' exp ')'
```

```
    | '-' exp
```

```
    | exp '+' exp
```

```
    | exp '-' exp
```

```
    | exp '*' exp
```

```
    | exp '/' exp
```

```
    | exp-logica
```

```
    | '(' exp-logica ')'
```

```
exp-logica:
```

```
    '(' exp-logica ')'
```

```
    | exp '=' exp
```

```
    | exp '<>' exp
```

```
    | exp '<=' exp
```

```
    | exp '>=' exp
```

```
    | exp '<' exp
```

```
    | exp '>' exp
```

```
    | 'nao' exp-logica
```

```
    | exp-logica 'e' exp-logica
```

```
    | exp-logica 'ou' exp-logica
```

```
    | exp-logica 'xor' exp-logica
```