
UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO ARARANGUÁ - ARA
DEPARTAMENTO DE COMPUTAÇÃO - DEC

# Construção de Compiladores

## Manual da Gramática

Graduandos Felipe Tomao, Matheus Akio e Rafael Canal
Prof. Marlon Oliveira

2019

## 1. Introdução

O presente documento serve como manual de utilização da GRAMÁTICA - 2019.2 disponibilizada pelo professor Marlon Oliveira para a disciplina Construção de Compiladores. Foi seguida a ordem da gramática para estruturação dos tópicos.

## 2. Criação de um programa

Para o compilador entender que o código digitado é um programa, o mesmo precisa estar delimitado por um início e um fim, para iniciar um programa é necessário a seguinte linha de comando:

program nome_do_programa;

O ponto e vírgula ( ; ) delimita a linha de inicialização, o que vem após, é um novo bloco de código, o ponto final ( . ) delimita o fim do programa. A estrutura do programa será apresentada nos próximos tópicos.

## 3. Composição do programa

Um programa é composto por quatro estruturas: procedimentos, constantes, variáveis e corpo. Cada uma delas possuem tipos e devem ser declaradas. Tais procedimentos serão descritos a seguir.

### 3.1 Identificadores

Representa o nome escolhido para rotular uma variável, constante ou o próprio algoritmo. É um nome único e não deve ser repetido durante o algoritmo. As regras são mencionadas a seguir:

1. Devem começar com letras;
2. Se houver mais de um caractere, pode ser letras ou underline _;
3. Só aceita letras e underline _;
4. Não pode ser uma palavra reservada;
5. Não deve conter caracteres especiais;

### 3.2 Literais

Literais são as representações do código fonte de um valor fixo, e são representados diretamente no código (literalmente) sem necessidade de computação. Podem ser números, caracteres ou strings. Por exemplo:

Em *x = 3, x* é uma variável e *3* é um literal.

### 3.2 Tipos de variáveis

Variáveis servem para armazenar dados, e estes dados podem ser de diferentes tipos, sendo estes *integer, char, string, real.*

**Tipo *integer:*** Representa o conjunto de números inteiros, podendo estes serem: zero, positivos e negativos, porém não decimais. O tamanho é de 2 bytes e consequentemente varia de -32768 a 32768. Por exemplo:

*int (identificador) = 5;*

Neste caso definimos uma variável (*identificador*) com tipo *integer* e valor 5.

**Tipo *char:*** Representa caracteres em geral, sendo estes números, letras, etc. O tamanho é de 1 byte. Deve ser delimitado por aspas simples, e entre o texto e as aspas, deve-se conter espaço, caso contrário, o analisador léxico retornará um erro. Por exemplo:

*char (identificador) = ‘ a ’;*

Neste caso definimos que a variável (*identificador*) do tipo *char* recebe o valor de ‘a’.

**Tipo *string:*** Representa uma sequência de caracteres (*char)* delimitados por aspas duplas, e entre o texto e as aspas, deve-se conter espaço, caso contrário, o analisador léxico retornará um erro. O mesmo, deve ser declarada do tipo *char* e seguida de seu tamanho. Por exemplo:

***char** str[6] = “batata”;*

Neste caso definimos uma *string* de 6 caracteres e em seguida atribuímos seu valor.

**Tipo *real*:** Este, diferentemente do tipo *integer* representa números decimais. A parte decimal desse número, é separada por um ponto (.)

**Estrutura *Array:*** o *Array* pode ser entendido como uma coleção de valores do mesmo tipo, sendo este um dos 4 citados anteriormente. Seus dados são acessados através de um índice. Matematicamente pode ser entendido como uma matriz. Deve ser iniciado por abertura de colchetes ([), finalizados por fechamento de colchetes (]) e deve conter entre estes seu tamanho. Seus valores devem ser inseridos entre aspas duplas e separados entre si por dois pontos finais (..) e seguido as regras prévias de tipos da variável. Por exemplo:

*int (identificador)[2] = [“1”..”2”]*

## 3.2 Operadores

Os operadores são sinais que fazem operações específicas com os valores armazenados nas variáveis. Os operadores podem ser divididos em aritméticos, comparativos e lógicos. Os operadores serão mostrados a seguir:

*Aritméticos:* “**+**” (soma), “**-**” (subtração), “*****” (multiplicação) e “*/*” (divisão).
*Comparativos:* “**>=**” (maior e igual), “**>**” (maior), “**=**” (igual), “**<=**” (menor e igual), “**<>**” (diferente) e “**<**” (menor).
*Lógicos:* “**or**” (ou) e “**and**” (e).

Em uma expressão com operadores, deve-se dar espaço entre o identificador ou número em questão e o operador em si, conforme o exemplo abaixo:

*identificador < 5*

Nota-se uma exceção quando tratamos de **sinais de menos**, pois entre o sinal e o número não deve haver espaço: Ex *-7*.

## 3.3 Comandos

Comandos são instruções que devem ser executadas pelo programa, e devem ser finalizadas por ponto-e-vírgula (;).

### 3.3.1) Comandos de entrada e saída:

Servem para receber e mostrar valores ao usuário. Temos *write* que serve para exibir expressões ou valores de variáveis e *read* que serve para ler valores inseridos pelo usuário para variáveis. Abaixo mostramos possíveis utilizações de ambos comandos:

> ***write*** *(identificador),* exibe o valor da variável identificada na tela.
> ***write*** *(1 + 1),* tal comando exibirá o valor "2" na tela do usuário;
> ***read*** *(identificador),* lê um valor inserido pelo usuário e armazena-o na variável identificada.

### 3.3.2) Comando de chama de procedimento:

Serve para executar um procedimento previamente criado no programa. É utilizado conforme demonstrado a seguir:

***chamaprocedure*** seguido de um identificador do procedimento seguido de parâmetros opcionais. Por exemplo:

> ***chamaprocedure*** *procExemplo (parametroExemplo1, parametroExemplo2)*

### 3.3.3) Comando condicional:

Comandos que são executados caso a condição estabelecida seja cumprida.

```
if [ Condição a ser respeitada ] then
    begin
        Comandos a serem executados
end;
```

Tal condicional pode ou não ser seguido com um comando *else* em formato semelhante ao *if* e que é executado quando a condição estipulada previamente não é respeitada.

### 3.3.4) Comandos de repetição

Executa certa parte do código repetidas vezes até que a condição de parada estipulada seja alcançada. Temos três comandos diferentes nesta categoria, e estes serão explicados abaixo:

***while*** *[* Expressão condicional *]* ***do***
***begin***
Comandos a serem executados
***end;***

Neste caso, enquanto a expressão condicional for respeitada, os comandos são executados repetidamente.

***repeat -*** Comandos a serem executados
***until*** *- [* Expressão condicional *];*

Neste outro caso, os comandos são executados repetidamente até que a condição estipulada seja atingida.

***for*** *[* identificador = Expressão de início *]* ***to*** *[* Expressão de parada *]* ***do***
***begin***
Comandos a serem executados
***end;***

Neste último caso, a partir da condição de início estipulada para um identificador arbitrário e até a condição de parada ser alcançada os comandos são executados repetidamente.

## 3.3 Declaração de constantes

Uma constante, propriamente dita, é um identificador ao qual é atribuído um valor que não pode ser alterado. Para declaração, esta deve receber um nome e um tipo, previamente declarado como const. O ponto e vírgula separa a declaração de inúmeras constantes numa mesma linha. Por Exemplo:

**const** nome_da_constante_A = tipo_constante_A; nome_da_constante_B = tipo_constante_B;

## 3.4 Declaração de variáveis

Uma variável, propriamente dita, é um identificador ao qual é atribuído um valor que pode ser alterado durante no decorrer do código. Para declaração, esta deve receber um nome e um tipo, previamente declarado como declaravariaveis. Por Exemplo:

**declaravariaveis** nome_da_variavel_A : tipo_variavel_A; nome_da_variavel_B : tipo_variavel_B;

### 3.5 Declaração de procedimentos

Para declarar um procedimento uma sequência de informações deve ser passadas: nome do procedimento, parâmetros e seus tipo, iniciando a linha com procedure. Por Exemplo:
**procedure** nome_do_procedimento(parametro_A, parametro_B : tipo_parametro)

### 3.6 Declaração de corpo

Um corpo é um bloco de código o qual pode ser formado por procedimento, por um ou vários comandos, etc. As palavras begin e end delimitam seu tamanho e tudo que estiver dentro delas fazem parte do corpo. Por exemplo:

```
declaravariaveis nota : integer;
begin
        nota  10;
        write (nota);
end
```

## 4. Comentários

Nem todo texto é código, à trechos em que o programador apenas faz comentários para documentar o código e auxiliar na interpretação de outras pessoas. Pode-se separar os comentário em dois tipo: de linha, ou conjunto de bloco e estes não devem ser compilados. Comentários só podem ser realizados dentro do corpo do programa.

### 4.1 Comentário de Linha

Para identificar a linha como um comentário a mesma deve iniciar com cerquilha (#) e após o texto em si. Por exemplo:

```
#Esta linha é um comentário
```

### 4.2 Comentário de Bloco

Para identificar o bloco como um comentário o mesmo deve estar contido entre cerquilha e asterisco (#*) de abertura e asterisco e cerquilha (*#) após o texto em si para fechamento. Por exemplo:

```
#*Este bloco
        é
um comentário*#
```

## 5. Erros Léxicos

Análise Léxica é um termo utilizado na construção de compiladores para referir-se, ao processo de analisar caracteres, e reproduzir símbolos presentes na gramática. Contudo, se este conjunto de símbolos não estiver presente na gramática, logo, consideramos isto, como um Erro Léxico.
Ou seja, um erro léxico, pode ser considerado tudo aquilo de errado referente a sintaxe. Como por exemplo, a extrapolação de caracteres em string, a extrapolação na atribuição de um número inteiro, entre outras.