

# **Notas de Aula de Algoritmos e Programação de Computadores**

FLÁVIO KEIDI MIYAZAWA

*com a colaboração de*

TOMASZ KOWALTOWSKI

Instituto de Computação - UNICAMP

Versão 2000.1

Estas notas de aula não devem ser usadas como única fonte de estudo. O aluno deve ler outros livros disponíveis na literatura.

Nenhuma parte destas notas pode ser reproduzida, qualquer que seja a forma ou o meio, sem a permissão dos autores.

Os autores concedem a permissão explícita para a utilização e reprodução deste material no contexto do ensino de disciplinas regulares dos cursos de graduação sob a responsabilidade do Instituto de Computação da UNICAMP.

© Copyright 2000

Instituto de Computação  
UNICAMP  
Caixa Postal 6176  
13083-970 Campinas-SP  
{fkm,tomasz}@ic.unicamp.br

## 2 Primeiros Programas em Pascal

Apesar da metodologia de fluxogramas ser antiga, ela ainda é muito usada para explicar a seqüência de instruções em programas e algoritmos. Há vários símbolos que fazem parte de um fluxograma. Para nós, este tipo de estrutura será importante apenas para representar a estrutura seqüencial dos algoritmos e programas de computador. Em um fluxograma, um *passo* ou *módulo* é representado por um retângulo. As *setas* indicam o próximo comando a ser executado. Um *losango* indica uma condição e conforme a condição seja satisfeita ou não, este pode levar a um de dois outros comandos.

Na figura 4 apresentamos alguns diagramas usados em fluxogramas.

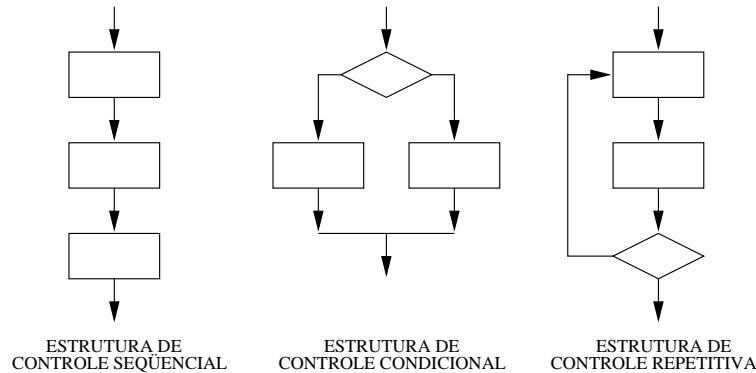


Figura 4: Exemplo de estruturas de controle usadas em programação estruturada.

Suponha que exista um curso cuja avaliação seja feita através de duas provas e um exame, sendo que o resultado final seja dado pelas seguintes regras: A primeira prova tem peso 2 e a segunda prova tem peso 3. Seja  $N$  a média ponderada destas duas provas. Caso  $N$  seja pelo menos 5.0, a nota final do aluno,  $F$ , é igual a  $N$ . Caso contrário, o aluno deve fazer o exame, digamos com nota  $E$ , e sua nota final, é a média aritmética entre  $N$  e  $E$ . Por fim, caso a nota final do aluno seja pelo menos 5.0, o aluno está aprovado, caso contrário, o aluno está reprovado. A figura 5 apresenta um exemplo de fluxograma para se avaliar um aluno de tal curso.

No início dos tempos da programação de computadores, viu-se que programas que continham quaisquer desvios, de um comando para outro, eram muito mais difíceis de se entender. Os programas mais fáceis de entender eram aqueles onde o fluxograma não tinha setas se cruzando. Provavelmente é esta liberdade que faz com que o fluxograma deva ser considerado com muito cuidado e por isso mesmo esteja em desuso. Um algoritmo descrito como um fluxograma pode ter setas levando para qualquer lugar do programa, podendo o tornar muito confuso e sem nenhuma organização.

As linguagens estruturadas fornecem um número limitado de *estruturas de controle*. Tais estruturas de controle foram desenvolvidas de tal forma que o fluxo de execução não possa ser de qualquer maneira, mas sim de forma bem organizada. Uma representação de um programa que usa estas estruturas de controle permite uma tradução para fluxogramas sem que haja linhas se cruzando. Desta forma a manutenção de um programa fica menos difícil, mesmo que isso seja feito por outra pessoa que não aquela que o implementou inicialmente. A linguagem Pascal é uma linguagem de alto nível onde as técnicas de programação estruturadas são estimuladas através de seus comandos.

Um programa em Pascal tem o seguinte formato:

```
program nome;  
  declarações {Área de Declarações}  
begin  
  Comandos {Corpo de Execução do Programa Principal}  
end.
```

As palavras **program**, **begin** e **end** são *palavras-chave* da linguagem Pascal. São palavras reservadas que fazem parte da sintaxe da linguagem e não podem ser usadas para declarar os objetos definidos pelo programador. As palavras **begin** e **end** servem para definir um bloco de instruções, no caso, definem o corpo de execução do programa principal. Na área de declarações, são definidos os objetos que iremos usar no programa, que inclui objetos que representam outros trechos de código.

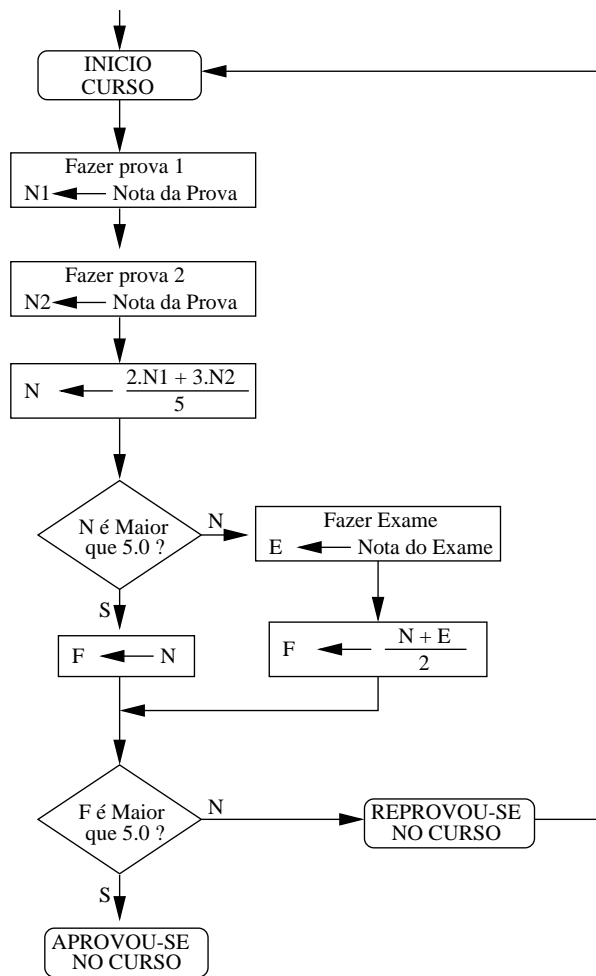


Figura 5: Fluxograma de avaliação de um curso.

Na figura 6 apresentamos um programa bem simples com apenas um comando de execução.

```

program BomDia;
begin
  writeln('Bom Dia!');
end.
  
```

Figura 6:

O Comando **writeln** na figura 6 escreve o texto 'Bom Dia!' no dispositivo de saída. Depois de escrever o texto, o programa ainda pula uma linha.

Existe também um outro comando, **write**, para imprimir um texto sem pular linha. Assim, um programa equivalente ao programa *BomDia* pode ser dado na figura 7:

```

program BomDia2;
begin
  write('Bom ');
  writeln('Dia!')
end.
  
```

Figura 7:

Note que há um espaço em branco depois de *Bom*, pois caso não tivesse, as palavras *Bom* e *Dia* seriam impressas juntas (*BomDia*).

```

program
BomDia1;
begin
write (
'Bom '
)
;
writeln
(
'Dia!');
end.

```

Figura 8: Programa desorganizado

## Observações

1. O ; (ponto e vírgula) serve para separar comandos.  
Obs.: Dois ponto e vírgula seguidos separam um comando vazio.
2. A execução do programa está definida pelos comandos entre **begin** e **end**. Note que logo após a palavra **end** segue um . (ponto final).
3. Comandos que são separados por espaços em branco não fazem diferença. Na verdade, cada item do programa pode ser separado: Assim, o programa BomDia poderia ter sido escrito como dado na figura 8. É claro que o programa fica bem mais legível na primeira forma.

## 2.1 Comentários

Dentro de um programa, descrito em Pascal podemos ter textos que não são considerados para a geração do código de máquina e servem para ajudar a compreensão do programa ou apenas como comentários.

Comentários em Pascal podem ser inseridos colocando-os entre “(\*)” e “(\*)” ou colocando-os entre “{” e “}”.

Assim, o programa *BomDia2* (figura 9) abaixo é equivalente aos dois programas acima.

```

{ Programa: BomDia.Pas }
{ Aluno: Fulano de Tal RA: 999999 }
{ Curso: MC102 }
{ Data: 01/01/01 }
{ Descrição: Programa para Imprimir “Bom Dia!” }
program BomDia2; { Programa para Imprimir “Bom Dia!” }
begin
write('Bom '); { Eu sou um comentário }
writeln('Dia!'); (* Eu também sou um comentário *)
end.

```

Figura 9:

A seguir apresentamos o fluxograma do programa BomDia2.

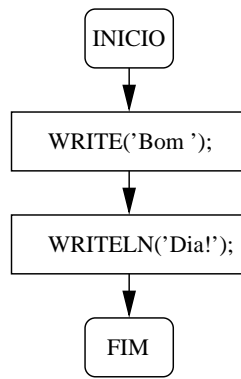


Figura 10: Fluxograma do programa hello3.

## 2.2 Identificadores e Constantes

No programa *BomDia* imprimimos textos usando os comandos **write** e **writeln**. Além disso, estes comandos também podem ser usados para imprimir expressões com tipos básicos do pascal. Considere o seguinte programa na figura 11:

```

program idade1;
begin
  writeln('Paulo tem ',5,' anos');
  writeln('Pedro tem ',8,' anos');
  writeln('A soma das idades de Paulo e Pedro é:',5+8);
  writeln('A media das idades de Paulo e Pedro é:',(5+8)/2);
end.
  
```

Figura 11:

O programa acima deve imprimir as linhas:

```

Paulo tem 5 anos
Pedro tem 8 anos
A soma das idades de Paulo e Pedro é: 13
A media das idades de Paulo e Pedro é: 6.5
  
```

Observe que o comando **writeln** pode manipular tanto números inteiros como números reais. Note também que se a idade de Pedro passar de 8 para 9 anos, então todos os lugares onde aparece o número 8, correspondente à idade de Pedro, devemos alterá-lo para 9.

Para facilitar este tipo de mudança podemos fazer uso de um identificador associado à idade de Pedro, igual a 8 anos. Assim, sempre que precisamos referenciar sua idade referenciamos este identificador em vez do valor 8. Isto facilita a atualização da idade de Pedro, que se formos mudar de 8 anos para 9 anos, atualizamos apenas na definição do valor do identificador associado à idade de Pedro. Desta maneira, não precisaremos nos preocupar em procurar todas as ocorrências do número 8, além de verificar se tal 8 é correspondente à idade de Pedro.

*Identificadores* são nomes simbólicos para os objetos referenciados nos programas em Pascal. Identificadores podem ser formados por uma letra ou caracter sublinhado seguido por qualquer combinação de letras, dígitos ou sublinhados. Na tabela a seguir, damos alguns exemplos de identificadores válidos e inválidos.

Identificadores válidos	Identificadores inválidos
Media	2X
Meu_Primeiro_Programa	A:B:C
Idade_Paulo	1*5
RA_999999	X(9)
Nota_MC102	Nota(102)

Na linguagem Pascal, não há diferenças com as letras maiúsculas e minúsculas usadas no identificador. Assim, as seguintes seqüências de caracteres representam o mesmo identificador:

Nota_MC102	NOTA_MC102	nota_mc102	noTA_mC102
------------	------------	------------	------------

Obs.: Em algumas linguagens, como a *linguagem C*, há distinção entre maiúsculas e minúsculas usadas em identificadores.

*Constantes* são objetos cujos valores não mudam durante a execução do programa. A definição das constantes deve ser feita na área de declarações, usando a palavra reservada **const** seguida das definições de cada constante. Cada constante é definida como:

*IdentificadorDeConstante* = constante;

No programa da figura 12, apresentamos o programa da figura 11 declarando as idades de Paulo e Pedro por constantes.

```
program idade2;
const Idade_Paulo = 5;
      Idade_Pedro = 9;
begin
  writeln('Paulo tem ',Idade_Paulo,' anos');
  write('Pedro tem ',Idade_Pedro,' anos');
  writeln('A soma das idades de Paulo e Pedro é:',Idade_Paulo+Idade_Pedro);
  writeln('A media das idades de Paulo e Pedro é:',(Idade_Paulo+Idade_Pedro)/2);
end.
```

Figura 12:

Outros exemplos de declaração de constantes:

```
program constantes;
const
  Min = 100;
  Max = 999;
  Versao = '5.2.3';
  LetraPrincipal = 'X';
  Aniversario = '01/01/2000';
  :
begin
  writeln(Min,'...',Max); {Imprime: 100...999 }
  writeln('Versão do Programa é ',Versao); {Imprime: Versão do Programa é 5.2.3 }
  :
end.
```

Figura 13:

## 2.3 Variáveis e Tipos Básicos

O uso de constantes permite que possamos trabalhar com valores previamente fixados e seu uso é feito através de um identificador que está associado àquela constante. Agora se quisermos que um determinado identificador esteja associado à diferentes valores durante a execução do programa, devemos usar o conceito de *variável*.

*Variáveis* são objetos que podem ter diferentes valores durante a execução do programa. Cada variável corresponde a uma posição de memória. Embora uma variável possa assumir diferentes valores, ela só pode armazenar apenas um valor a cada instante. Cada variável é identificada por um identificador e contém valores de apenas um tipo.

Os *tipos básicos* em Pascal são: **char**, **integer**, **boolean**, **real**, **string**. A tabela a seguir apresenta exemplos de cada um destes tipos.

<i>Objeto</i>	<i>tipo de dado</i>
1	integer
-2488	integer
1.0	real
-13.6	real
0.653	real
-1.23456789E+12	real expresso em notação exponencial
'Algoritmos e Programação de Computadores'	string
'123456'	string
true	boolean
false	boolean
'A'	char
'B'	char
'6'	char
'\$'	char

Vamos ver mais sobre cada um destes tipos:

**integer**: Representa um subconjunto dos números inteiros. O tipo integer no padrão Turbo Pascal representa os inteiros de  $-32768$  a  $32767$ . Já no Padrão Gnu Pascal (GPC), o tipo integer representa inteiros de  $-2147483648$  a  $2147483647$ .

**real**: Em geral, um número real é representado por duas partes: a *mantissa* e o *expoente*. A precisão dos números reais é determinada pela quantidade de bits usada em cada uma destas partes. Exemplo: o número 10,45 pode ser representado por  $1.045E + 01$ . O número 0,00056993 pode ser representado por  $5.6993E - 04$ . Este tipo não permite representar todos os números reais, mesmo que este número seja pequeno. Um exemplo simples disso é o número  $\frac{1}{3} = 0,33333\dots$ , que certamente deve ter sua precisão computada corretamente até um certo número de casas decimais.

**char**: Representa um caracter. Ex.: 'A','B','\$',' ' (espaço em branco),...

**string**: É uma seqüência de caracteres. Uma string pode ser dada por uma seqüência de caracteres entre apóstrofes. Ex: 'Joao Paulo'. A maioria dos compiladores tem a restrição que uma string pode ter até 255 caracteres. Atualmente há compiladores que admitem strings com número bem maior de caracteres. A quantidade de caracteres (bytes) usados pela string é definido com um número entre colchetes logo após a palavra string. Ex.: **string[50]**;

O tipo acima corresponde a uma cadeia de caracteres com capacidade máxima de 50 caracteres. Obs.: uma cadeia de caracteres a ser armazenada neste tipo de string não necessariamente precisa ter 50 caracteres. Para saber mais sobre este tipo, veja a seção 8.

**boolean**: Uma variável do tipo boolean pode ter dois valores. True (verdadeiro) ou False (falso).

Além destes tipos, vamos considerar como tipo básico o tipo **byte**, que usa apenas um byte de memória e pode armazenar um número de 0 a 255.

A declaração de variáveis é feita usando se a seguinte sintaxe:

```

var
  ListaDeIdentificadoresDoTipo1  : Tipo1;
  ListaDeIdentificadoresDoTipo2  : Tipo2;
                                  :
  ListaDeIdentificadoresDoTipoN  : TipoN;

```

A palavra **var** é uma palavra reservada da linguagem Pascal que indica que vai começar uma declaração de *variáveis*.

**Exemplo 2.1** No quadro seguinte apresentamos a declaração de algumas variáveis.

```
var    x, y, z: real;
      Nome, Sobrenome: string[50];
      i, n, idade: integer;
      Sexo: char;
```

## 2.4 Comando de Atribuição

O comando de atribuição `:=` atribui um valor que está a direita de `:=`, que pode ser uma expressão, para uma variável (única) que está na parte esquerda do comando, representada pelo seu identificador.

Sempre que houver um comando de atribuição devemos observar os seguintes itens:

- Primeiro é avaliada a parte direita, obtendo-se um valor único. A parte da direita do comando de atribuição pode ser uma expressão bem complicada.
- O valor obtido da expressão da parte direita é atribuído à variável (única) que está na parte esquerda.
- O tipo do valor avaliado da expressão na parte direita deve ser compatível com o tipo da variável que está na parte esquerda. Obs.: um número inteiro também pode ser visto como um número real.
- Se um valor é armazenado em uma variável, o valor anterior que estava nesta variável é perdido.

**Exemplo 2.2** Considere a declaração das variáveis *x, y, z e t* dadas a seguir:

```
var nome: string[50];
    x: integer;
    y: real;
    z, t: char;
```

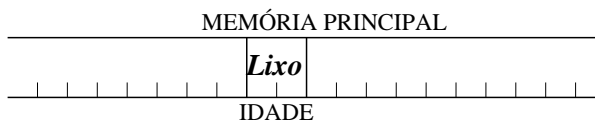
Na tabela seguinte apresentamos alguns exemplos de atribuições válidas e inválidas para estas variáveis.

Atribuições válidas	Atribuições inválidas
$x := 10$	$x := 3.14$
$y := x + 1.5$	$x := y$
$z := 'a'$	$z := 'Computador'$
$z := t$	$t := t + z$
$nome := 'Carlos'$	$nome := 100;$

Para esclarecer melhor estes conceitos, suponha que você tenha uma variável, chamada *idade*, definida como sendo inteira:

**var idade: integer;**

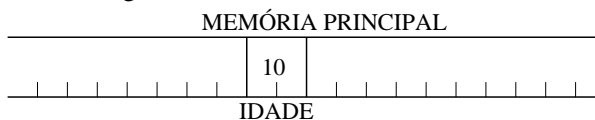
No momento da execução, o programa deve definir uma certa porção da memória, em bytes, para esta variável. A maioria das linguagens não inicializa as variáveis logo após a definição de sua memória em bytes. Assim, vamos considerar que o valor inicial em uma variável é sempre um valor desconhecido e portanto iremos considerá-lo como um *lixo*.



Após o comando

Idade:=10;

a configuração da memória deve ficar da seguinte forma





**Exercício 2.1** Considere o seguinte programa:

```
program Produtos;
var Produto : integer;
begin
  Produto:=2;
  Produto:=Produto*Produto;
  Produto:=Produto*Produto;
  Produto:=Produto*Produto;
  writeln('O valor contido em Produto é: ',Produto);
end.
```

O que o programa imprime ?

Obs.: Muitas das linguagens de programação não inicializam automaticamente as variáveis. Assim, é importante inicializar as variáveis em algum momento, antes de referenciar seus valores em expressões.

## 2.5 Operadores

### Operadores Aritméticos

A seguir, apresentamos alguns operadores aritméticos presentes na linguagem Pascal.

- +, -, \*, / (adição, subtração, multiplicação e divisão de números, resp.).
- **mod** (Operador de inteiros: resto de divisão inteira).
- **div** (Operador de inteiros: parte inteira da divisão).

Operandos dos tipos **integer** e **real** podem ser combinados. O tipo do valor resultante de uma operação aritmética dependerá da operação e dos tipos de seus operadores. A tabela a seguir apresenta as operações válidas para tipos de operandos e seus respectivos resultados.

Operandos	Operador	Tipo de resultado
<b>integer e integer</b>	+, -, *, <b>div</b> , <b>mod</b>	<b>integer</b>
<b>integer e integer</b>	/	<b>real</b>
<b>real e integer</b>	+, -, *, /	<b>real</b>
<b>real e real</b>	+, -, *, /	<b>real</b>

**Exemplo 2.4** Exemplos de operações aritméticas:

```
program OpAritmetico;
var i:integer;
    x:real;
begin
  i := 4 mod 3; {resultado da operação é inteiro e igual a 1}
  x := 4/2; {resultado da operação é real e igual a 2.0}
  x := 2 * 3 + 5 * 3 - (123456 mod 123);
end.
```

## Operadores Lógicos

A linguagem Pascal oferece os seguintes operadores lógicos:

- **not** (inverte o valor booleano)
- **and** (é verdadeiro se ambos os operandos são verdadeiros)
- **or** (é verdadeiro se um dos operandos for verdadeiro)

**Exemplo 2.5** *Exemplos de operações lógicas:*

```
program OpLogico;
var a, b, c: boolean;
begin
  a := false;  b := true;
  c := ( not a and b); {c ← True (verdadeiro)}
  writeln(c);
end.
```

**Exercício 2.2** *Algumas implementações da Linguagem Pascal apresentam o operador lógico **xor**, sendo que este operador resulta verdadeiro se exatamente um dos operandos for verdadeiro. Implemente este operador usando os operadores **and**, **or** e **not**.*

## Operadores Relacionais

Os operadores relacionais podem ser usados para os tipos **real**, **integer**, **string**, **char** e **byte**. Naturalmente os dois operandos devem ser de tipos compatíveis.

O resultado de uma operação relacional é sempre lógico (**boolean**), retornando ou **true** ou **false**.

No caso do operador = (igual), também é possível se comparar valores booleanos. Na comparação de strings, os caracteres são comparados dois a dois caracteres nas mesmas posições, até que um caracter seja menor que outro ou até que uma string termine antes que outra. Assim, 'Ave Maria' é maior que 'Ave Cesar'. Pois o primeiro caracter a diferir nas duas strings é o quinto caracter. Na codificação ASCII, 'M' é maior que 'C'. Portanto o resultado da relação

('Ave Maria' > 'Ave Cesar')

é true (verdadeiro). Como a codificação de 'A' em ASCII é menor que 'a', então o resultado da relação

('AAA' >= 'aaa')

é false (falso).

**Exemplo 2.6** *Exemplos de operações relacionais:*

```
program OpRelacionais;
var a, b, c: boolean;
    x, y: real;
begin
  x := 10.0;  y := 30.0;  a := false;
  b := not a and ((x >= y) or ('paulo' <= 'pedro'));
  c := ('paulo' <= 'Paulo'); {resultado é falso, já que 'p' > 'P'}
  writeln(b);
  writeln(c);
end.
```

## Operadores em Strings

O único operador para seqüência de caracteres é o operador de concatenação de strings, +, que concatena duas strings.

**Exemplo 2.7** *Exemplo da operação de concatenação de strings:*

```
program OpConcatenaString;  
var a: string[50];  
begin  
  a := 'Algoritmos';  
  a := a + 'e';  
  a := a + 'Programação';  
  a := a + 'de';  
  a := a + 'Computadores';  
  writeln(a); {Deve imprimir: AlgoritmoseProgramaçãodeComputadores}  
end.
```

## Precedência

Ao se avaliar uma expressão, os operadores seguem uma ordem de precedência. Estas regras de precedência são familiares às usadas em operadores algébricos. Se dois operadores possuem o mesmo nível de precedência, então a expressão é avaliada da esquerda para a direita. Parte de expressões que estão contidas entre parênteses são avaliadas antes da expressão que a engloba. A seguinte tabela apresenta a ordem de prioridade dos operadores (primeiro os de maior precedência) na linguagem Pascal.

Expressões dentro de parênteses
Operador unário (positivo ou negativo)
Operador <b>not</b>
Operadores multiplicativos: *, /, <b>div</b> , <b>mod</b> , <b>and</b>
Operadores aditivos: +, -, <b>or</b> , <b>xor</b>
Operadores relacionais: =, <>, <, >, <=, >=, <b>in</b>

Sempre que quisermos quebrar a ordem natural das operações, podemos usar parênteses para especificar a nova ordem. Assim, uma atribuição como  $Exp := (1 + (3 + 5) * (9 - 20 / (3 + 7)))$ , pode ter a seguinte ordem de operações

$$Exp := (1 + (3 + 5) * (9 - 20 / (3 + 7)))$$

$$Exp := (1 + 8 * (9 - 20 / (3 + 7)))$$

$$Exp := (1 + 8 * (9 - 20 / 10))$$

$$Exp := (1 + 8 * (9 - 2))$$

$$Exp := (1 + 8 * 7)$$

$$Exp := (1 + 56)$$

$$Exp := 57.$$

## 2.6 Algumas Funções Pré-Definidas

A maioria das linguagens de programação já nos oferece um subconjunto de funções básicas que podemos usar para construir expressões mais complexas. A tabela a seguir apresenta algumas destas funções.

Função	Tipo do parâmetro	Tipo do Resultado	Resultado
<b>ArcTan</b>	real	real	arco tangente
<b>Cos</b>	real	real	coseno
<b>Sin</b>	real	real	Seno
<b>Abs</b>	integer	integer	valor absoluto
<b>Abs</b>	real	real	valor absoluto
<b>Exp</b>	real	real	exponencial
<b>Frac</b>	real	real	parte fracionária
<b>Int</b>	real	real	parte inteira
<b>Ln</b>	real	real	Logaritmo Natural
<b>Random</b>	$N$ : integer	integer	Número pseudo aleatório em $[0, \dots, N - 1]$
<b>Random</b>	sem parâmetro	real	Número pseudo aleatório em $(0, \dots, 1)$
<b>Round</b>	real	integer	Arredondamento
<b>Sqr</b>	integer	integer	quadrado
<b>Sqr</b>	real	real	quadrado
<b>Sqrt</b>	real	real	raiz quadrado
<b>Trunc</b>	real	integer	Truncamento
<b>Chr</b>	integer	char	ordinal para caracter
<b>Ord</b>	tipo escalar, exceto reais	integer	número ordinal do tipo escalar
<b>Length</b>	string	integer	Quantidade de caracteres da string

**Exemplo 2.8** Uma maneira muito comum de se fazer a potência de um número  $x$ ,  $x > 0$ , elevado a outro número  $y$ , é através das funções **Exp** e **Ln**, usando a seguinte fórmula:  $x^y = \mathbf{Exp}(y * \mathbf{Ln}(x))$ . O programa a seguir lê dois números,  $x$  e  $y$ , e imprime  $x^y$ .

```

program Expoente;
var x,y,z : real;
begin
  write('Entre com o valor de x: ');
  readln(x);
  write('Entre com o valor de y: ');
  readln(y);
  z:= Exp(y*ln(x));
  writeln('O valor de ',x,' elevado a ',y,' é igual a ',z);
end.

```

Além destas funções, existem alguns comandos úteis para atualizar o valor de algumas variáveis. Algumas destas são:

Comando	Tipo da variável de parâmetro	Resultado
<b>Inc</b>	inteiro	Incrementa o valor da variável de 1.
<b>Dec</b>	inteiro	Decrementa o valor da variável de 1.

## 2.7 Comandos de Escrita

Como vimos, podemos escrever os dados na tela através do comando **writeln**. Este comando tem como parâmetros uma lista de objetos, não necessariamente do mesmo tipo. Os parâmetros devem ser separados por vírgulas. Cada parâmetro é impresso logo após a impressão do parâmetro anterior. A linguagem Pascal tem dois comandos básicos para escrita: o comando **write** e o comando **writeln** (de *write+line*). Ambos imprimem os valores de seus respectivos argumentos, com a diferença que o comando **write** mantém a posição da próxima impressão/leitura logo após a impressão de seu último parâmetro e o comando **writeln** atualiza a posição da próxima impressão/leitura para o início da próxima linha. O comando **writeln** sem argumentos apenas atualiza a posição da próxima impressão/leitura para o início da próxima linha.

**Exemplo 2.9** Considere o programa a seguir:

```
program ComandosDeEscrita;  
var  
  a: boolean;  
  b: integer;  
  c: string[50];  
begin  
  a := true;  
  b := 2 + 3 * 4 - 5;  
  c := 'Exemplo';  
  writeln('Valor de a = ',a,'. Vai para a próxima linha');  
  writeln('Valor de b = ',b,'. Vai para a próxima linha');  
  writeln('Valor de c = ',c,'. Vai para a próxima linha');  
  write('Valor de a = ',a,' ');  
  write('Valor de b = ',b,' ');  
  write('Valor de c = ',c,'');  
  writeln;  
  writeln('Fim das impressões');  
end.
```

O programa acima imprime as seguintes linhas:

```
Valor de a = True. Vai para a próxima linha  
Valor de b = 9. Vai para a próxima linha  
Valor de c = Exemplo. Vai para a próxima linha  
Valor de a = true. Valor de b = 9. Valor de c = Exemplo.  
Fim das impressões.
```

Muitas vezes queremos formatar melhor a forma de impressão dos dados no comando *write* e *writeln*. Uma maneira simples de definir a quantidade de caracteres, que uma expressão será impressa, é colocar a expressão a ser impressa seguida de : (dois pontos) e a quantidade de caracteres desejado. Isto fará com que a expressão seja impressa com a quantidade de caracteres especificada. Caso o dado a ser impresso é um número real, podemos complementar esta formatação adicionando : (dois pontos) e a quantidade de casas decimais depois do ponto decimal. No exemplo a seguir apresentamos um programa com algumas formatações de impressões e sua respectiva tela de execução.

**Exemplo 2.10** Considere o programa a seguir:

```
program ComandosDeEscrita;  
var  
  a: boolean;  
  b: integer;  
  c: string[50];  
  d: real;  
begin  
  a := true;  
  b := 4321;  
  c := 'Exemplo';  
  d := 1234.5678;  
  writeln('Valor de a = [',a:1,']');  
  writeln('Valor de b = [',b:7,']');  
  writeln('Valor de c = [',c:4,']');  
  writeln('Valor de d = [',d:9:2,']');  
end.
```

O programa acima imprime as seguintes linhas:

```
Valor de a = [T]
Valor de b = [ 4321]
Valor de c = [Exem]
Valor de d = [ 1234.57]
```

Obs.: Na formatação de valores reais, pode haver perda ou arredondamentos em algumas casas decimais.

## 2.8 Comandos de Leitura

Há dois comandos básicos de leitura: o comando **read** e o comando **readln**. Ambos os comandos tem como parâmetros variáveis, de tipos básicos diferentes de **boolean**, e permite ler do teclado os novos valores a serem atribuídos às variáveis. O comando **read**, após sua execução, mantém a posição da próxima impressão/leitura logo após a leitura das variáveis. O comando **readln** (de *read line*) atualiza a posição da próxima impressão/leitura para o início da próxima linha.

**Exemplo 2.11** Considere o programa a seguir:

```
program ComandosDeLeitura;
var
  idade: integer;
  nome: string[50];
begin
  write('Entre com sua idade: ');
  readln(idade);
  write('Entre com seu nome: ');
  readln(nome);
  writeln('O Sr. ',nome,' tem ',idade,' anos.')
end.
```

Caso o usuário entre com os dados `29` e `Fulano de Tal`, o programa deve ficar com a seguinte configuração na tela:

```
Entre com sua idade: 29
Entre com seu nome: Fulano de Tal
O Sr. Fulano de Tal tem 29 anos.
```

**Exercício 2.3** Faça um programa que lê 7 números e imprima a média destes números. Obs.: Use no máximo duas variáveis numéricas.

**Exercício 2.4** Sabendo que o valor numérico de uma letra na codificação ASCII é dada pela função **ord**, faça um programa que leia uma letra e escreva a codificação em binário desta letra.

Exemplo: Suponha que a letra a ser lida é a letra 'a'. A função **ord**('a') retorna o valor 97 e o programa deve imprimir: **01100001**. Note que  $0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 97$ .

**Exercício 2.5** Faça um programa que lê dois pontos  $P_1 = (x_1, y_1)$  e  $P_2 = (x_2, y_2)$ ; e devolve a distância entre estes dois pontos, cujo valor é impresso no programa principal. Obs.:  $dist = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ .

**Exercício 2.6** Faça um programa que lê uma temperatura em graus Fahrenheit e retorne a temperatura em graus Centígrados. Obs.:  $(C = 5/9 \cdot (F - 32))$ .

**Exercício 2.7** Faça um programa que lê uma temperatura em graus Centígrados e retorne a temperatura em graus Fahrenheit.

## 2.9 Tipos Escalares

Os tipos escalares **integer**, **byte** e **char** são tipos que representam conjuntos ordenados com valores distintos. A linguagem Pascal também permite que possamos definir nossos próprios tipos de dados escalares ordenados. Isto pode ser feito definindo o conjunto ordenado especificando cada elemento do conjunto ou aproveitando os elementos de conjunto de dados já definidos anteriormente. A seguir apresentamos estas duas maneiras de se definir tipos escalares ordenados.

Com este tipo de dados, por serem ordenados, podemos fazer comparações do tipo: igualdade (=), menor (<), menor ou igual (<=), maior (>), maior ou igual (>=) e diferença (<>).

### Especificando todos os elementos

Para definir um tipo escalar ordenado especificando todos os elementos, escrevemos seus elementos (como identificadores) entre parênteses e separados por vírgula.

Obs.: estes tipos não podem ser impressos diretamente com uso do comando `write` ou `writeln`.

*(Lista\_de\_identificadores)*

Como já vimos, a função **ord** sobre um caracter devolve o código ASCII do mesmo. A função **ord** também pode ser usada para encontrar a posição de outros elemento definidos por conjuntos escalares ordenados especificados elemento a elemento. A função **ord** retorna 0 caso seja o primeiro elemento, 1 se for o segundo elemento, 2 se for o terceiro elemento, assim por diante.

**Exemplo 2.12** *A seguir apresentamos alguns exemplos de declarações e atribuições usando os conjuntos ordenados especificados elemento a elemento.*

```
type
  TipoSexo = (Masculino,Feminino);
  TipoCor = (Vermelho,Verde,Azul);
  TipoMes = (Janeiro,Fevereiro,Marco,Abril,Maio,Junho,Julho,
            Agosto,Setembro,Outubro,Novembro,Dezembro);
  TipoEstadoSul = (RS,SC,PR);
var
  Sexo: TipoSexo;
  Estado: TipoEstadoSul;
  Cor: TipoCor;
  Categoria: (Aluno,Funcionario,Professor);
  Mes: TipoMes;
begin
  Sexo := Masculino;
  Estado := PR;
  Categoria := Funcionario;
  Mes := Abril;
  writeln(Ord(Mes)); {Deve imprimir 3}
end.
```

Muitos compiladores fazem uma otimização de maneira que se o tipo de conjunto ordenado tiver até 256 elementos, então este tipo usa apenas um byte de memória.

### Especificando uma faixa de elementos

Um tipo faixa é um tipo de dado escalar que define uma faixa de elementos. Esta faixa é especificada pelo primeiro elemento e o último elemento. Os elementos de um tipo de dado faixa devem fazer parte de um conjunto ordenado escalar. Para isso é possível usar tanto os tipos escalares já definidos (como os tipos **integer** e **char**) como também os novos tipos escalares ordenados como definidos elemento a elemento. Um tipo faixa é definido especificando o

primeiro e o último elemento na seguinte sintaxe:

*Primeiro\_Elemento..Ultimo\_Elemento*

Isto significa que uma variável deste tipo poderá receber qualquer valor que esteja no intervalo *Primeiro\_Elemento..Ultimo\_Elemento*. Se uma variável é definida como sendo de um tipo faixa onde os extremos foram tomados a partir de um conjunto ordenado especificado elemento a elemento, então os elementos que são passíveis de serem atribuídos a esta variável ficam restritos ao intervalo definido no conjunto ordenado também.

Um tipo faixa também pode ser usado para especificar os índices de vetores, mesmo que não sejam números inteiros.

Obs.: Um tipo escalar só poderá ser impresso pelo comando `write` ou `writeln` se este for do tipo `char` ou `integer`.

**Exemplo 2.13** *A seguir apresentamos alguns exemplos de declarações e atribuições usando uma faixa de elementos.*

```
type
  TipoMaiuscula = 'A'..'Z';
  TipoMinuscula = 'a'..'z';
  TipoDigitoCaracter = '0'..'9';
  TipoDigito = TipoDigitoCaracter;
  TipoDigitoNumero = 0..9;
  TipoMes = (Janeiro, Fevereiro, Marco, Abril, Maio, Junho, Julho,
            Agosto, Setembro, Outubro, Novembro, Dezembro);
  TipoMesPrimeiroSemestre = Janeiro..Junho; {Supondo a declaração de TipoMes do exemplo anterior}
  TipoString = string[100]; {Tipo auxiliar}
var
  DigC: TipoDigito;
  DigN: TipoDigitoNumero;
  IdadeAdolecencia: 12..18;
  Letra: TipoMaiuscula;
  MesEstagio: TipoMesPrimeiroSemestre;
begin
  DigC := '5';
  DigN := 5;
  Letra := 'X';
  MesEstagio := Abril;
end.
```