



INSTITUTO FEDERAL

Paraná

Campus Assis Chateaubriand

Algoritmos e Lógica de Programação

Tipos de Dados, Variáveis e Estruturas de Controle

Tipos de Dados, Variáveis e Estruturas de Controle

Conteúdo

- Tipos de dados primitivos: *inteiro, real, character (cadeia ou string) e lógico*;
- Variáveis e constantes;
- Atribuição e Operadores *relacionais, lógicos e aritméticos*;
- Estruturas de controle: *sequencia, decisão (ou seleção) e repetição*.

Variáveis (1)

Variáveis – Por Quê?

Do modelo EPS (Entrada, Processamento e Saída), sabemos que:

1. A **entrada de dados** consiste na captura de certos valores que o programa precisa para realizar o processamento (por exemplo, lendo algo que o usuário digitou);
2. Os **dados de saída** são os valores gerados pelo programa que são exibidos ao usuário (por exemplo, mostrados na tela do computador).

Os dados de entrada são armazenados em determinadas **posições de memória** acessadas por meio das **variáveis** do programa. Da mesma maneira, toda vez que o programa produz algum valor intermediário, também o armazena numa variável e, frequentemente, os próprios dados de saída são armazenados em algumas variáveis antes de serem exibidos aos usuários. O conteúdo de uma variável pode ser modificado ao longo do tempo, de acordo com a lógica do algoritmo.

As variáveis são utilizadas para armazenar valores que precisam ser referenciados (lidos ou escritos) durante a execução do programa. Além disso, as variáveis são sempre rotuladas com um identificador amigável (o *nome* da variável), que facilitam a leitura e compreensão do programa (ou algoritmo). Uma variável serve tão somente para identificar um valor e armazená-lo na memória.

Variáveis (2)

Analogia da Caixinha

Uma variável serve para armazenar “coisas” (valores, para ser mais preciso), da mesma forma que utilizamos caixas em casa para guardar coisas. Isso nos permite criar uma analogia válida para ilustrar o conceito.

Tipo, nome e valor de uma Variável

Existem diferentes **tipos** de caixas, específicas para guardar coisas diferentes, da mesma maneira que existem **tipos de variáveis** diferentes que se prestam a guardar **valores** de naturezas diferentes. Por exemplo, uma variável do tipo “inteiro” não pode armazenar o nome de uma pessoa, nem tampouco uma variável do tipo “caractere” pode armazenar a idade de alguém. Para não confundirmos as caixas no momento de procurar por alguma coisa que guardamos, podemos colar **rótulos** nelas; isto é, podemos dar **nomes** às caixas. Da mesma forma, atribuímos **nomes (identificadores)** às variáveis para poder utilizá-las.

A analogia da caixinha é útil para entender os conceitos de **variável**, **identificador**, **tipo** e **valor**.



Podemos pensar numa variável como uma espécie de caixa que precisamos utilizar para guardar algo (um **valor**) de nosso interesse. Para conveniência, podemos colar um rótulo na caixa (um **identificador**); assim, se precisarmos de seu conteúdo mais tarde, saberemos muito bem onde procurar.

inteiro é o **tipo** do conteúdo guardado na caixinha. Essa caixinha **só** armazena valores inteiros.
idade é o rótulo (**identificador** ou **nome**) da caixinha
21 é o **conteúdo** da caixinha (**valor** da variável)

inteiro **idade** ← **21**;

Variáveis (3)

Tipos de Variáveis

Uma variável armazena somente valores de um determinado **tipo**

- Dependendo da linguagem de programação e do hardware, existem muitos tipos de variáveis disponíveis para o programador:
 - Em qualquer linguagem, existem dois tipos fundamentais: **numérico** e **string**;
 - Na maior parte das linguagens, não existe somente um tipo numérico, mas vários; em Java, por exemplo, existem os tipos numéricos inteiros **byte**, **short**, **int** e **long**, e os tipos numéricos decimais **float** e **double**;
 - Além da **string**, é comum que se defina um tipo **caractere** (como o **char** do Java), destinado a armazenar um único caractere;
 - Também é comum que se defina um tipo lógico **booleano** (**boolean**, no java) que pode assumir somente dois valores (“verdadeiro” ou “falso”).
- Nos algoritmos construídos em nosso estudo, geralmente basta que sejam definidos os tipos **inteiro**, **real**, **string** e, quando necessário, **booleano**.

O tipo **string** denota uma **cadeia de caracteres**; ou seja, vários caracteres agrupados. Assim, temos que a letra “a” é um caractere, mas que “aaaaaa” é uma **string** (ou **cadeia**).

Variáveis (4)

Tipos de Variáveis

Um **tipo de variável** ou simplesmente **tipo** é um atributo que informa ao computador qual é a natureza dos valores que serão armazenados numa determinada variável. Por exemplo, os comandos abaixo

```
string nomePessoa;  
inteiro idadePessoa;
```

“nomePessoa” está declarada como string e só pode armazenar valores alfanuméricos

“idadePessoa” é uma variável inteira e só pode armazenar valores inteiros

declaram duas variáveis, *nomePessoa* e *idadePessoa* que são, respectivamente, dos tipos string e inteiro. Portanto, *nomePessoa* somente pode receber valores alfanuméricos e *idadePessoa*, valores numéricos inteiros. Isso nos indica que os comandos abaixo não têm sentido algum:

```
nomePessoa ← 21;  
idadePessoa ← “José”;
```

Uma variável string não pode armazenar um valor numérico!

Uma variável inteira não pode armazenar uma string!



Sabemos que o tipo de uma variável restringe a natureza dos valores que ela pode armazenar (eis por quê uma variável numérica não pode armazenar o nome de uma pessoa). Como uma analogia, pense num jogo de encaixe: da mesma forma que somente os blocos de um certo formato podem ser encaixados em um dos moldes vazados, somente os valores de um certo tipo podem ser armazenados numa variável. O tipo também define quais operações podem ser feitas com uma variável (ex.: não é possível somar dois nomes).

Variáveis (5)

Tipos de Variáveis

Considerações sobre o tipo de uma variável

1. Tipo **string**:

- O tipo **string** é o mais flexível pois ele se presta a guardar dados **alfanuméricos** (ou seja, caracteres de qualquer espécie, incluindo dígitos numéricos). Contudo, um valor **string** vai ser *sempre* interpretado como um conjunto de caracteres:

O comando **string** `strTeste ← "15"` **NÃO** atribui o valor numérico 15 à variável `strTeste`, mas sim o valor alfanumérico "15" (note que o valor dessa variável está entre aspas – por convenção, os valores alfanuméricos são **sempre** delimitados por aspas).
Pelo mesmo raciocínio, a expressão "15" + 1 não tem sentido algum: qual é o resultado da soma entre um número e uma palavra? Qual seria o resultado de "José da Silva" + 1??

- Muitos algoritmos utilizam o recurso de **concatenação de strings**:

```
string n ← "José"; escreva (n+ " da Silva") ; (mostra "José da Silva" na tela)  
escreva ("1"+ "2"+ "3"+ "4") ; (mostra "1234" na tela, ao invés do número 10)
```

Variáveis (6)

Tipos de Variáveis

Considerações sobre o tipo de uma variável

2. Tipo **inteiro**:

- Tipos **inteiros** são utilizados para representar *valores enumeráveis* (ou seja, que podem ser contados), como o número de vezes que um certo comando deve ser executado ou uma quantidade de objetos que não podem ser fracionados (por exemplo, assentos num avião, quartos num hotel, parcelas de um pagamento, etc);
- Tentar armazenar um valor decimal numa variável inteira frequentemente denota um erro de lógica. Por exemplo, a média aritmética de um conjunto de números inteiros não é, conceitualmente, um número **inteiro**, mas sim um número **decimal**.

3. Tipo **real**:

- É o tipo numérico mais abrangente, pois todo número **inteiro** é um número **real**, embora o contrário não seja verdadeiro: 3 não somente é um número **inteiro**, mas é **real** também; 2.5 é **real**, mas certamente não é **inteiro**;
- Embora toda variável inteira possa ser substituída por uma real, é sensato utilizar cada tipo conforme apropriado. Em caso de dúvida, use o tipo **real**.

Variáveis (7)

Tipos de Variáveis

Considerações sobre o tipo de uma variável

4. Tipo **booleano** (ou **lógico**):

- Tipos booleanos armazenam somente dois valores (**Verdadeiro** ou **Falso**);
- Variáveis booleanas são primariamente associadas a **comparações** (também chamados de **testes lógicos**), que empregam **operadores relacionais** (ou seja, os operadores **<**, **>**, **=**, **<>**, **<=** e **>=**) ou **lógicos** (**E**, **OU** e **NÃO**):

```
booleano teste;
```

```
inteiro idade ← 15;
```

```
string nome ← "João";
```

```
teste ← (idade > 18);
```

```
teste ← (idade < 30);
```

```
teste ← (idade = 15);
```

```
teste ← (nome = "Pedro");
```

```
teste ← (nome <> "Pedro");
```

```
teste ← (idade = 16) E (nome <> "Pedro");
```

```
teste ← (idade = 16) OU (nome <> "Pedro");
```

Uma expressão relacional utiliza operadores relacionais para comparar valores, e também é conhecida como teste lógico porque retorna um valor booleano!

/* a variável teste recebe o valor Falso */

/* teste recebe Verdadeiro */

/* teste recebe Verdadeiro */

/* teste recebe Falso */

/* teste recebe Verdadeiro */

/* teste recebe Falso */

/* teste recebe Verdadeiro */

Os operadores relacionais comparam valores diferentes entre si, e o resultado dessa comparação só pode ser "Verdadeiro" ou "Falso".

Uma Palavra Sobre Operadores Relacionais

O conceito de **operadores** é oriundo da matemática, onde são utilizados para obter certos valores a partir de outros, chamados de **operandos**

Operando Valores

Os operadores mais conhecidos são aqueles utilizados nas operações aritméticas básicas (+, -, * e /). Como sabemos, *um **operador aritmético** produz um resultado numérico a partir de dois **operandos numéricos***; por exemplo, o número 3 é obtido quando somamos 1 e 2. Por sua vez, um **operador relacional** (<, >, <=, >=, = e <>) produz um ***resultado booleano*** (**VERDADEIRO** ou **FALSO**) ao comparar dois **operandos**, que podem ser **numéricos**, **caracteres**, **strings** ou **booleanos**.

Junto com as **estruturas de seleção** (**SE... ENTÃO... SENÃO...**) que estudaremos oportunamente, as **expressões relacionais** permitem a um algoritmo tomar decisões e desviar o *fluxo de processamento* apropriadamente. Por exemplo, se um valor digitado pelo usuário for incorreto (digamos, um ano com 5 dígitos ou um preço negativo), o algoritmo pode rejeitar a entrada e finalizar com uma mensagem de erro.

As chamadas **estruturas de repetição** (**ENQUANTO FAÇA**, **REPITA ATÉ** e **PARA FAÇA**) também dependem do uso das expressões relacionais.

Essas variáveis levam às expressões relacionais abaixo

```
numA ← 5; numB ← 10;  
nmPessA ← "Pedro"; nmPessB ← "João";  
vlFlag1 ← FALSO; vlFlag2 ← VERDADEIRO;
```

Expressão Relacional	Resultado
numA > numB	FALSO
numB > numA	VERDADEIRO
numA = numB	FALSO
numB <> numA	VERDADEIRO
numA <= numB	VERDADEIRO
numA >= numB	FALSO
vlFlag1 <> vlFlag2	VERDADEIRO
nmPessA = nmPessB	FALSO
nmPessA = "João"	FALSO
nmPessA <> "João"	VERDADEIRO

Alguns Exemplos com Operadores Relacionais

Assumindo que o valor das variáveis A, B e C sejam, respectivamente, 3, 7 e 4 determine o valor das expressões relacionais abaixo. Faça o mesmo considerando que o valor dessas variáveis sejam, respectivamente, 5, 2 e 10.

a) $(A + C) > B$

b) $(A + 2 * C) - B < 5$

c) $B * C \leq 5 * A$

d) $B \geq (A + 2)$

e) $A * C \leq 2 * B$

f) $(B - A) \geq 3 * C$

g) $(C + A) \neq B$

h) $(B + A) \leq C$

i) $2 * C - 1 < B$

j) $A * C \leq B *$

k) $C = B - A$

l) $2 * B \neq 1 + (3 * A + C)$

Uma Palavra Sobre Operadores Lógicos (1)

Combinando Valores Booleanos

Os **operadores lógicos** são aqueles que produzem valores booleanos a partir de operandos também booleanos. Existem vários desses operadores, mas os mais utilizados são três: **E**, **OU** e **NÃO**.

O operador **E** (**AND** ou **conjunção**) é um operador **binário** (ou seja, utiliza dois operandos) e caracteriza-se por retornar VERDADEIRO somente se ambos os operandos forem verdadeiros; caso contrário, retorna FALSO.

O operador **OU** (**OR** ou **disjunção**) também é um operador **binário**, mas retorna VERDADEIRO se qualquer um dos operandos forem verdadeiros; caso contrário, retorna FALSO.

Por fim, o operador **NÃO** (**NOT** ou negação) é **unário** (possui um único operando) e inverte um valor booleano (transforma VERDADEIRO em FALSO e vice-versa).

```
numA ← 5; numB ← 10; nmPessA ← "Pedro"; nmPessB ← "João";  
vlFlag1 ← FALSO; vlFlag2 ← VERDADEIRO;
```

(numB > numA) E vlFlag2	VERDADEIRO
(nmPessA = "João") E (numB = 10)	FALSO
(nmPessA = "João") OU (numB = 10)	VERDADEIRO
(numB > numA) E (nmPessB <> "Pedro")	VERDADEIRO
(numB > numA) E NÃO (nmPessB = "Pedro")	VERDADEIRO
(vlFlag1 E vlFlag2) OU (numB < 0)	FALSO
NÃO ((vlFlag1 E vlFlag2) OU (numB > 0))	FALSO

Testes lógicos (comparam dois valores e retorna "Verdadeiro" ou "Falso")

Um operador lógico combina dois valores booleanos e produz um terceiro valor (com exceção do operador "NÃO", que é unário)

Uma Palavra Sobre Operadores Lógicos (2)

Talvez a melhor maneira de se compreender como funcionam os operadores lógicos seja utilizando uma **tabela da verdade**. Essa tabela é usada em lógica booleana para descrever o funcionamento de uma ou mais funções lógicas tais como os operadores **E**, **OU** e **NÃO**. Do lado esquerdo, se coloca o valor dos operandos e do lado direito, o resultado da expressão. Em nossos estudos, não utilizaremos as tabelas da verdade - elas são mostradas aqui simplesmente para sintetizar o comportamento de cada um dos operadores lógicos que abordamos.

Tabela da Verdade

Valor1	Valor2	Valor1 E Valor2	Valor1 OU Valor2	NÃO Valor2
FALSO	FALSO	FALSO	FALSO	VERDADEIRO
FALSO	VERDADEIRO	FALSO	VERDADEIRO	FALSO
VERDADEIRO	FALSO	FALSO	VERDADEIRO	VERDADEIRO
VERDADEIRO	VERDADEIRO	VERDADEIRO	VERDADEIRO	FALSO

Valor dos operandos

Resultado da operação

Em nossos algoritmos, o valor de cada um dos operandos é o resultado de uma expressão lógica ou relacional. Por exemplo, Valor1 poderia ser o resultado de uma expressão tal como "vlClienteEspecial = VERDADEIRO" e Valor2 o resultado de "vlCompra > 1.000,00". Então, a expressão "Valor1 E Valor2" seria lido como "(vlClienteEspecial = VERDADEIRO) E (vlCompra > 1.000,00)" (em português, "se o cliente for especial e o valor da compra for superior a R\$ 1.000,00 reais").

Alguns Exemplos com Operadores Lógicos

Assumindo que o valor das variáveis A, B e C sejam, respectivamente, 3, 4 e 8 determine o valor das expressões relacionais abaixo. Faça o mesmo considerando que o valor dessas variáveis sejam, respectivamente, 10, 2 e 5.

a) $A > 3$ **E** $2 * B = C$

b) $A <> 2$ **OU** $B \leq 5$

c) $A = 3$ **OU** $B \geq 2$ **E** $B/2 < A$

d) $A = 3$ **E NÃO** $(B \leq 4)$ **E** $C = 8$

e) $A = 3$ **E** $B > 4$ **E** $C = 8$

f) $A <> 8$ **OU** $B = 4$ **E** $C > 2$

g) $A > B$ **OU** $B < 5$

h) $B > A$ **E** $C <> A$

i) $A <> B$ **E** $B = C$

j) $C > 2$ **OU** $A < B$

k) $C > 2$ **OU NÃO** $(A \geq B)$

l) $A > B$ **OU** $B > A$ **E** $C <> B$

Variáveis (8)

Tipos de Variáveis

Considerações sobre o tipo de uma variável

5. Além de determinar qual é a natureza dos valores que podem ser armazenados numa variável, *o tipo determina quais são as operações que podem ser feitas com essa variável*:

```
string    s1, s2;
```

```
inteiro   i1, i2;
```

```
booleano  b1, b2;
```

```
i1 ← 1; s1 ← "1";
```

```
i2 ← i1 OU 1; /* operador lógico OU envolvendo inteiros?? */
```

```
s2 ← s1 + "2"; /* resulta em "12", não em 3. Ou seja, é uma concatenação, não uma soma */
```

```
s2 ← (i1 > 10); /* uma string Falsa?? */
```

```
s2 ← "2" - s1; /* subtração de strings?? */
```

```
b1 ← Verdadeiro * Falso; /* multiplicação entre valores lógicos?? */
```

```
b2 ← b1 + Verdadeiro; /* adição entre valores lógicos?? */
```

```
b2 ← i1 + 2; /* 1 + 2 resulta em Verdadeiro ou Falso? Isso tem sentido?? */
```

Inicialização de Variáveis (1)

- Em muitas linguagens de programação como o Java e o Pascal, as variáveis devem ser **declaradas**:
 - Ao declarar uma variável, o programador tem que especificar o seu **tipo** e **identificador antes** que ela seja utilizada;
 - Opcionalmente, uma variável pode ser inicializada com algum valor na declaração.
- Exemplo de declaração e inicialização de variáveis:

Algoritmo TestaVars;

Declaracao de variaveis

inteiro int1 \leftarrow 3, int2 \leftarrow 9, int3, int4;

/* int3 e int4 não foram inicializadas */

real vlPi \leftarrow 3.141592654, vlRaio1, vlRaio2;

/* vlRaio1 e vlRaio2 não foram inicializadas */

string nomePessoa;

/* nomePessoa não foi inicializada */

Inicio

1. escreva(nomePessoa); /* nomePessoa não foi inicializada e não tem valor definido */

2. vlRaio1 \leftarrow 5*int1; /* Correto: vlRaio1 foi inicializada após declarada, int1 foi inicializada */

3. real vlArea1 \leftarrow 2*vlPi*vlRaio1; /* Correto: vlArea1 foi declarada nessa linha e inicializada */

4. vlArea2 \leftarrow 2*vlPi*vlRaio2; /* Ilegal: vlArea2 não foi declarada. Ademais, vlRaio2 não foi inicializada */

5. int4 \leftarrow int1 + int2 + int3; /* int3 não foi inicializada e não tem valor definido */

Fim do algoritmo.

Inicialização de Variáveis (2)

- Em nossa disciplina, a declaração de variáveis é *recomendada*, porque estimula o desenvolvedor a adotar boas práticas de programação;
- É praxe que as variáveis sejam declaradas no início de cada algoritmo, mas isso não é obrigatório. Em linguagens de programação como o Java, as variáveis podem ser declaradas em qualquer ponto do código;
- Não é recomendável utilizar uma variável que não tenha sido inicializada:
 - Em princípio, o valor de uma variável não inicializada é indefinido (diz-se que uma variável não inicializada contém “lixo”);
 - Algumas linguagens atribuem valores padronizados para variáveis não inicializadas (por exemplo, tipos numéricos não inicializados assumem automaticamente o valor zero e strings assumem o valor nulo);
 - Em java, os atributos de um objeto que não são inicializados assumem valores padrão, mas variáveis locais devem ser necessariamente inicializadas antes de serem utilizadas.

Inicialização de Variáveis (3)

Um Exemplo em Java

```
class TestaVars {  
    int engVars;  
    int matVars;  
    int fisVars;  
    public static void main(String args[]){  
        TestaVars obj1 = new TestaVars ();  
        obj1.engVars = 50;  
        obj1.matVars = 80;  
        obj1.fisVars = 90;  
        TestaVars obj2 = new TestaVars ();  
        obj2.engVars = 80;  
        obj2.matVars = 60;  
        obj2.fisVars = 85;  
    }  
}
```

Declaração de variáveis
(no caso do Java, são
chamadas de "atributos")

Não é o que estamos estudando
nesse momento, mas nessas linhas de
código se declara e se inicializa um
"objeto", que é uma variável.

Inicialização das variáveis. Se o código
for modificado, elas poderiam ser
inicializadas assim que declaradas.

Valores literais e constantes (1)

A definição de "constante" contrasta com a de "variável"

- Diferentemente de uma variável, cujo conteúdo pode mudar ao longo do tempo, *o conteúdo de uma constante jamais se modifica depois da inicialização*;
- Literais (ou "valores literais") são valores especificados diretamente no código do programa e, portanto, são fixos;
- Embora as constantes e os literais tenham uma característica em comum (não podem ser modificados), não devem ser confundidos:
 - Uma constante é referenciada por um identificador (nome) e, exceto pelo fato de não poder ter seu conteúdo alterado, é utilizada exatamente como uma variável;
 - Os literais também são chamados de constantes anônimas (*unnamed constants*).

Valores literais e constantes (2)

- Um literal é inserido no código pelo programador (“hardcoded”), e só pode ser modificado mediante a alteração do programa;
- Uma constante pode ser inicializada por um literal ou por um valor fornecido pelo usuário – mas, naturalmente, depois de inicializada não pode mais ser modificada;
- Muitas linguagens como o Java suportam explicitamente o uso de constantes. Em nossos algoritmos, poderemos identificar as constantes como tal;
- Para ilustrar a diferença entre variáveis, constantes e literais, considere esses comandos escritos em pseudocódigo:

```
1. string nomeP ← “João”; /* declara uma variável string e a inicializa com o valor literal “João” */
2. constante string nomePC ← “Maria”; /* declara uma constante inicializada com o literal “Maria” */
3. “João” ← nomePC; /* ilegal, modificar um literal dessa maneira não faz sentido algum! */
4. nomeP ← nomePC + “Rita”; /* altera o valor da variável, de “João” para “Maria Rita” */
5. nomePC ← “Afonso”; /* ilegal, não se pode modificar uma constante! */
```

Valores literais e constantes (3)

Algoritmo Usa Constantes e Literais;

Declaração de Variáveis

string nomeP ← "João";

inteiro quant ← 20;

Declaração de Constantes

string nomePC ← "Pedro";

inteiro quantC ← 120;

Início

escreva ("Entre com um nome"); /* imprime um literal na tela */

leia (nomeP); /* lê um valor do teclado, fornecido pelo usuário, e o armazena em *nome* */

escreva ("Entre com outro nome"); /* imprime um literal na tela */

leia (nomePC); /* ilegal, pois *nomePC* foi declarado como constante */

escreva (nome + " da Silva"); /* concatena uma variável e um literal */

quant ← quantC * 2; /* atribui o valor 240 a *quant* */

quantC ← 20; /* ilegal, pois *quantC* foi declarado como constante */

Fim.

Valores literais e constantes (4)

Algoritmo Usa Constantes e Literais;

Declaração de Variáveis

string nomeP ← "João";

inteiro quant ← 20;

Declaração de Constantes

string nomePC ← "Pedro";

inteiro quantC ← 120;

Variáveis

Início

escreva ("Entre com um nome"); /* imprime um literal na tela */

leia (nomeP); /* lê um valor do teclado, fornecido pelo usuário, e o armazena em nome */

escreva ("Entre com outro nome"); /* imprime um literal na tela */

leia (nomePC); /* ilegal, pois nomePC foi declarado como constante */

escreva (nome + " da Silva"); /* concatena uma variável e um literal */

quant ← quantC * 2; /* atribui o valor 240 a quant */

quantC ← 20; /* ilegal, pois quantC foi declarado como constante */

Fim.

**Valores
literais**

Constantes