



# A Linguagem do Pseudo-Código

versão 3.0

Fabiano Oliveira  
fabiano.oliveira@ime.uerj.br

# O que são?

- Pseudo-código é uma linguagem com formato livre para descrever um algoritmo cujo objetivo é evidenciar o método de solução sendo empregado, sem se preocupar com os comandos específicos que serão necessários para implementar tal método em uma linguagem de programação

# Por que usar?

- Vantagens:
  - Como o foco é o método de solução, podemos evitar usar comandos que são importantes para o mecanismo de funcionamento de uma linguagem, mas não para o problema diretamente (ex: cabeçalhos de fontes, preparação de ambientes, etc.)

# Por que usar?

- Vantagens (continuação):
  - Podemos evitar detalhar passos que são necessários para uma linguagem, mas não para o entendimento do método de solução
    - Exemplos:
      - $A[1..N] \leftarrow 10$
      - Ordenar  $A[1..N]$

# Como usar?

- É necessário que a transformação de cada comando para uma linguagem de programação seja considerada uma tarefa fácil para o público-alvo
  - Portanto, o pseudo-código "ordenar  $V[1..N]$ " pode ser usado em algoritmos para leitores com um pouco de experiência, mas não para programadores iniciantes

# Exemplos

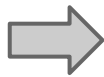
- A seguir, exemplos de tradução de pseudo-código em:
  - C
  - C++
  - Python
  - JavaScript
  - Java
  - TuPy

# Linguagem C

# IMPORTANTE

Se a sua conversão, seguindo os exemplos a seguir, não estiver funcionando, volte a comparar minuciosamente o exemplo dado com seu código convertido. Em 90% dos casos, alguma diferença ou alguma adaptação inexistente está causando o problema. Por exemplo, considere a conversão:

```
var N: ^Inteiro  
alocar(N)
```



```
int * N;  
N = (int *) malloc(sizeof(int));
```

Se ao invés, N for do tipo ponteiro para uma certa estrutura Aluno, a adaptação necessária mais natural seria trocar todas as ocorrências de "int" ("Inteiro") para o nome da estrutura Aluno, assim:

```
var N: ^Aluno  
alocar(N)
```



```
Aluno * N;  
N = (Aluno *) malloc(sizeof(Aluno));
```



# Comandos de pseudo-códigos utilizados (e sua transformação)

## O programa:

| Pseudo-código | programa CalculaAlgo()<br><código do programa>   |
|---------------|--|
| C             | <pre>// Arquivo CalculaAlgo.c #include&lt;stdio.h&gt; #include&lt;stdlib.h&gt; // incluir cabeçalhos de bibliotecas // usadas no código do programa int main(){     &lt;código do programa&gt;     return 0; }</pre> |

Linhas adicionais ou alterações destas podem ser requeridas pela versão do compilador de C!

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Compilando o programa:

### Linux:

No terminal:

```
> gcc -o <nome_executável> <nome_arquivo.c> -lm
```

Ex:

```
> gcc -o CalculaAlgo CalculaAlgo.c -lm
```

### Windows:

Em geral, há um botão ou menu de compilação no software que se está usando que possui o compilador de C

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Executando o programa:

Depois de gerado o executável do programa, a execução depende do sistema operacional.

## Windows:

Seja <DIR\_PROG> o diretório onde o compilador gera o executável

```
C:\<DIR_PROG>> CalculaAlgo.exe
```

ou

```
C:\<DIR_PROG>> CalculaAlgo.exe <entrada.txt
```

```
>saida.txt
```

(entradas do teclado são obtidas do arquivo entrada.txt, e saídas para a tela escritas em saida.txt)

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Executando o programa:

Depois de gerado o executável do programa, a execução depende do sistema operacional.

### Linux:

```
> ./<nome_executável>
```

Ex:

```
> ./CalculaAlgo
```

ou

```
> ./CalculaAlgo <entrada.txt >saida.txt
```

(entradas do teclado são obtidas do arquivo entrada.txt, e saídas para a tela são escritas no arquivo saida.txt)

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Comentários:

| Pseudo-código | <code>// este é um comentário</code>                                    |
|---------------|---|
| <b>C</b>      | <pre>// este é // um comentário  /* este é    outro comentário */</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis escalares:

|               |   |
|---------------|---|
| Pseudo-código | <pre>var A: Inteiro, B: Real, C: Lógico, D: Caractere A ← 10 B ← 20.2 C ← V D ← 'A'</pre>   |
| C             | <pre>#include &lt;stdbool.h&gt; // para uso do "bool" em C99 // para compiladores anteriores, use // #define bool char // #define true 1 // #define false 0 int A; double B; bool C; char D; A = 10; B = 20.2; C = true; D = 'A';</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores Aritméticos Comuns:

| Operador         | Pseudo-código                          | C  |
|------------------|--|--|
| Soma             | $a + b$                                | $a + b$  |
| Subtração        | $a - b$                                | $a - b$  |
| Multiplicação    | $a * b$                                | $a * b$  |
| Divisão          | $a / b$                                | $a / b$  |
| Divisão Inteira  | $a \div b$<br>(ou $a \text{ div } b$ ) | //a e b inteiros<br>$a / b$                                    |
| Resto da Divisão | $a \text{ mod } b$                     | $a \% b$   |
| Exponenciação    | $a ^ b$                                | <code>#include &lt;math.h&gt;</code><br><code>pow(a, b)</code> |
| Raiz quadrada    | $\sqrt{a}$<br>(ou $\text{sqrt}(a)$ )   | <code>#include &lt;math.h&gt;</code><br><code>sqrt(a)</code>   |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores em Cadeias:

| Operador            | Pseudo-código                                     | C  |
|---------------------|---|--|
| Declaração          | var S: Cadeia<br>ou<br>var S[1..N]:<br>Caractere  | char * S;<br>ou<br>char S[N];  |
| Comprimento         | S   | <code>#include &lt;string.h&gt;</code><br>strlen(S)  |
| Caractere           | S[5]  | S[4]   |
| Subcadeia           | R ← S[ini..fim]                                   | <code>#include &lt;string.h&gt;</code><br>strncpy(R, S+ini-1, fim-ini+1);  |
| Encontrar Subcadeia | pos ← encontrar<br>(S, SEnc,<br>aPartirDePosicao) | <code>#include &lt;string.h&gt;</code><br>char *ini = strstr(S,<br>SEnc+aPartirDePosicao-1);<br>pos = (inicio == NULL ? 0 :<br>(inicio - s)/sizeof(char)+1); |



# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores em Cadeias:

| Operador                           | Pseudo-código                         | C  |
|------------------------------------|---------------------------------------|--|
| Concatenação                       | $S3 \leftarrow S1 + S2$               | <pre>#include &lt;string.h&gt; S3 = (char *) malloc( sizeof(char)*strlen(S1)+strlen(S2)+1) ; S3[0] = '\0'; strcat(S3, S1); strcat(S3, S2);</pre> |
| Criar cadeia de outro tipo de dado | $S \leftarrow \text{para\_cadeia}(N)$ | <pre>sprintf(S, "%d", N); sprintf(S, "%0.1f", N);</pre>  |
| Criar número de cadeia             | $N \leftarrow \text{de\_cadeia}(S)$   | <pre>N = atoi(S) N = atof(S)</pre>   |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Atribuição múltipla:

| Pseudo-código | <code>var A: Inteiro, B: Real, C: Lógico, D: Caractere<br/>A, B, C, D ← 10, 20.2, V, 'A'</code>    |
|---------------|--|
| <b>C</b>      | <code>int A; double B; bool C; char D;<br/>A = 10;<br/>B = 20.2;<br/>C = true;<br/>D = 'A';</code> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Atribuição múltipla:

| Pseudo-código | <code>//X, Y inteiros</code><br><code>X, Y ← Y, X</code>       |
|---------------|--|
| <b>C</b>      | <pre>int Temp1 = Y; int Temp2 = X; X = Temp1; Y = Temp2;</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis vetores:

| Pseudo-código   | <pre>var A[1..100]: Inteiro A[1..100] ← 0</pre>   |
|---|---|
| <b>C</b><br><b>(versão 1)</b><br>Alocação Estática (limite para número de elementos relativamente baixo)                      | <pre>int A[100]; int i; for (i = 0; i &lt; 100; i++) {     A[i] = 0; }</pre>                                |
| <b>C</b><br><b>(versão 2)</b><br>Alocação Dinâmica (limite para número de elementos é a quantidade de memória contígua livre) | <pre>int * A = (int *) malloc(100*sizeof(int)); int i; for (i = 0; i &lt; 100; i++) {     A[i] = 0; }</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis vetores (#2):

| Pseudo-código   | <pre>var A[1..100]: Inteiro A[1..100] ← 0</pre>  |
|---|--|
| <b>C (versão 1)</b><br>Alocação Estática (limite para número de elementos relativamente baixo)                      | <pre>int A[101]; <b>//o elemento A[0] ficará sem uso</b> int i; for (i = 1; i &lt;= 100; i++) {     A[i] = 0; }</pre>                                |
| <b>C (versão 2)</b><br>Alocação Dinâmica (limite para número de elementos é a quantidade de memória contígua livre) | <pre>int * A = (int *) malloc(101*sizeof(int)); <b>//o elemento A[0] ficará sem uso</b> int i; for (i = 1; i &lt;= 100; i++) {     A[i] = 0; }</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis matrizes:

| Pseudo-código   | <pre>var A[1..N, 1..N]: Inteiro A[i, j] ← 0, para todo <math>1 \leq i, j \leq N</math></pre>  |
|---|---|
| <b>C</b><br><b>(versão 1)</b><br>Alocação Estática (limite para número de elementos relativamente baixo)                      | <pre>int A[N][N]; int i; int j; for (i = 0; i &lt; N; i++) {     for (j = 0; j &lt; N; j++) {         A[i][j] = 0;     } }</pre>  |
| <b>C</b><br><b>(versão 2)</b><br>Alocação Dinâmica (limite para número de elementos é a quantidade de memória contígua livre) | <pre>int i; int j; int ** A = (int **) malloc(N*sizeof(int *)); for (i = 0; i &lt; N; i++) {     A[i] = (int *) malloc(N*sizeof(int)); } for (i = 0; i &lt; N; i++) {     for (j = 0; j &lt; N; j++) {         A[i][j] = 0;     } }</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis matrizes (#2):

| Pseudo-código   | <pre>var A[1..N, 1..N]: Inteiro A[i, j] ← 0, para todo 1 ≤ i, j ≤ N</pre>  |
|---|--|
| <b>C</b><br><b>(versão 1)</b><br>Alocação Estática (limite para número de elementos relativamente baixo)                      | <pre>int A[N+1][N+1]; //A[0,0..N] e A[0..N,0] ficarão sem uso int i; int j; for (i = 1; i &lt;= N; i++) {     for (j = 1; j &lt;= N; j++) {         A[i][j] = 0;     } }</pre>   |
| <b>C</b><br><b>(versão 2)</b><br>Alocação Dinâmica (limite para número de elementos é a quantidade de memória contígua livre) | <pre>int i; int j; int ** A = (int **) malloc((N+1)*sizeof(int *)); for (i = 1; i &lt;= N; i++) {     A[i] = (int *) malloc((N+1)*sizeof(int)); } //A[0,0..N] e A[0..N,0] ficarão sem uso for (i = 1; i &lt;= N; i++) {     for (j = 1; j &lt;= N; j++) {         A[i][j] = 0;     } }</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis ponteiros:

|               |   |
|---------------|---|
| Pseudo-código | <pre>var N, M: ^Inteiro, i: Inteiro alocar(N) N^ ← 10 M ← @i M^ ← 11 desalocar(N)</pre>             |
| C             | <pre>int * N, * M, i; N = (int *) malloc(sizeof(int)); *N = 10; M = &amp;i; *M = 11; free(N);</pre> |



# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis estruturas:

|   |  |
|---|--|
| <b>Pseudo-código</b>  | <pre>estrutura Aluno:     Notas[1..N], Matricula: Inteiro     Ouvinte: Lógico     Prox: ^Aluno  N ← 100000 var a1: Aluno, a2: ^Aluno a1.Ouvinte ← F alocar(a2) a1.Prox ← a2</pre>                              |
| <b>C</b><br><b>(versão 1)</b><br>o tamanho do<br>vetor Notas é fixo,<br>não varia registro<br>a estrutura | <pre>#define N 100000 typedef struct Aluno {     int Notas[N];     bool Ouvinte;     struct Aluno *Prox; } Aluno; Aluno a1, *a2; a1.Ouvinte = false; a2 = (Aluno *) malloc(sizeof(Aluno)); a1.Prox = a2;</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis estruturas:

|  |   |
|--|---|
| <b>Pseudo-código</b>   | <pre>estrutura Aluno:     Notas[1..N], Matricula: Inteiro     Ouvinte: Lógico     Prox: ^Aluno  N ← 100 var a1: Aluno a1.Ouvinte ← F N ← 200 var a2: ^Aluno alocar(a2) a1.Prox ← a2</pre>   |
| <b>C</b><br><b>(versão 2)</b><br>o tamanho do<br>vetor Notas pode<br>variar regisro a<br>estrutura | <pre>typedef struct Aluno {     int * Notas;     bool Ouvinte;     struct Aluno *Prox; } Aluno; int N = 100; Aluno a1; a1.Notas = (int *) malloc(N * sizeof(int)); a1.Ouvinte = false; N = 200; Aluno *a2; a2 = (Aluno *) malloc(sizeof(Aluno)); (*a2).Notas = (int *) malloc(N * sizeof(int)); a1.Prox = a2;</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

- Declarações de estruturas devem vir antes do seu uso!

```
typedef struct Aluno {  
    bool Ouvinte;  
} Aluno;
```

```
Aluno a1, *a2;  
a1.Ouvinte = false;
```

```
Aluno a1, *a2;  
a1.Ouvinte = false;
```

```
typedef struct Aluno {  
    bool Ouvinte;  
} Aluno;
```

# Comandos de pseudo-códigos utilizados (e sua transformação)

- A navegação "ponteiro, campo de estrutura, ponteiro, campo de estrutura, etc." tem uma notação especial por conveniência:

|               |  |
|---------------|--|
| Pseudo-código | <pre>estrutura Aluno:     Matricula: Inteiro     Prox: ^Aluno  var L: ^Aluno ... // escrever a 3.a matrícula da lista escrever(L^.Prox^.Prox^.Matricula)</pre>   |
| C             | <pre>typedef struct Aluno {     int Matricula;     struct Aluno *Prox; } Aluno; Aluno *L; ... // escrever a 3.a matrícula da lista printf("%d\n", L-&gt;Prox-&gt;Prox-&gt;Matricula); // mais conveniente, embora equivalente, a // printf("%d\n", (*(*(L).Prox).Prox).Matricula);</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Escrita de variáveis:

| Pseudo-código | escrever ("O maior valor é ", valormax)   |
|---------------|---|
| <b>C</b>      | <pre>#include &lt;stdio.h&gt; printf("O maior valor é %d\n", valormax);  ou  FILE *fp; fp = fopen("resultados.txt", "w"); fprintf(fp, "O maior valor é %d\n", valormax); fclose(fp);  ou qualquer outra maneira de guardar a saída!</pre> |

Note que o tipo do dado e outros detalhes (como quebra de linha) devem ser considerados, assim como a formatação estética da saída.

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Escrita de variáveis:

| Pseudo-código | escrever(C[1..N])   |
|---------------|---|
| <b>C</b>      | <pre>#include &lt;stdio.h&gt;  int i; for (i=0; i &lt; N; i++) {     printf("C[%d] = %d\n", i, C[i]); }</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Leitura de variáveis:

| Pseudo-código | ler(N, C[1..N])  |
|---------------|--|
| <b>C</b>      | <pre>#include &lt;stdio.h&gt;  printf("Entre com o tamanho do vetor: "); scanf("%d", &amp;N);  int i; for (i = 0; i &lt; N; i++) {     printf("Entre com valor C[%d]: ", i);     scanf("%d", &amp;(C[i])); }</pre> |

Note que o tipo do dado e outros detalhes (como a formatação estética da entrada) devem ser levados em conta.

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Leitura de variáveis:

| Pseudo-código          | <code>var Nome[1..TAM_MAX]: Caractere<br/>ler(Nome[1..TAM_MAX])</code>  |
|------------------------|---|
| <b>C</b><br>(versão 1) | <pre>#include &lt;stdio.h&gt;  char Nome[TAM_MAX]; printf("Entre com o Nome: "); int i = 0; while ((scanf("%c", &amp;(Nome[i])) == 1) &amp;&amp;       (i &lt; TAM_MAX - 1) &amp;&amp;       (Nome[i] != '\n')) {     i++; } Nome[i] = '\0'; //marcando final de cadeia</pre> |
| <b>C</b><br>(versão 2) | <pre>#include &lt;stdio.h&gt;  char Nome[TAM_MAX]; printf("Entre com o Nome: "); scanf("%s", Nome); //final de cadeia já é colocado com "%s"</pre>  |



# Comandos de pseudo-códigos utilizados (e sua transformação)

## Condicional:

|               |   |
|---------------|---|
| Pseudo-código | <pre>se condição1 então     &lt;bloco-então&gt; senão se condição2 então     &lt;bloco-senão-se&gt; senão     &lt;bloco-senão&gt;</pre>   |
| C             | <pre>if (condição1) {     &lt;bloco-então&gt; } else if {condição2} {     &lt;bloco-senão-se&gt; } else {     &lt;bloco-senão&gt; }</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores Lógicos Comuns:

| Operador       | Pseudo-código      | C  |
|----------------|--------------------|----|
| Igual          | =                  | == |
| Diferente      | $\neq$             | != |
| Menor          | <                  | <  |
| Maior          | >                  | >  |
| Menor ou Igual | $\leq$             | <= |
| Maior ou Igual | $\geq$             | >= |
| E              | E ou $\wedge$      | && |
| OU             | OU ou $\vee$       |    |
| NÃO            | NÃO ou ! ou $\neg$ | !  |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Repetição (para):

| Pseudo-código | para $i \leftarrow 1$ até $N$ passo 2 faça<br><bloco-para>                          |
|---------------|---|
| <b>C</b>      | <pre>for (int i = 1; i &lt;= N; i = i + 2) {<br/>    &lt;bloco-para&gt;<br/>}</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Repetição (enquanto):

| Pseudo-código | enquanto condição faça<br><bloco-enquanto>                         |
|---------------|--|
| <b>C</b>      | <pre>while (condição) {<br/>    &lt;bloco-enquanto&gt;<br/>}</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Procedimentos:

|               |  |
|---------------|--|
| Pseudo-código | <pre>procedimento Calcula(val p1: Inteiro, ref p2: Real) //Supõe: Condição 1 //Garante: Condição 2     &lt;bloco-procedimento&gt;  var valor: Real ← 10 Calcula(10, valor)</pre> |
| C             | <pre>void Calcula(int p1, double *p2) {     &lt;bloco-procedimento&gt; }  double valor = 10.0; Calcula(10, &amp;valor);</pre>  |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Funções:

|               |   |
|---------------|---|
| Pseudo-código | <pre>função Calcula(val p1: Inteiro, ref p2: Real): Caractere //Supõe: Condição 1 //Garante: Condição 2, onde 'retorno' é o valor de retorno     &lt;bloco-função&gt;     retornar (&lt;valor&gt;)  var valor: Real ← 10 escrever(Calcula(10, valor))</pre> |
| C             | <pre>char Calcula(int p1, double *p2) {     &lt;bloco-função&gt;     return &lt;valor&gt;; }  double valor = 10.0; printf("%c\n", Calcula(10, &amp;valor));</pre>   |

# Comandos de pseudo-códigos utilizados (e sua transformação)

- Declarações de procedimentos e funções devem vir antes do seu uso!

```
char Calcula(int p1, double *p2) {  
    <bloco-função>  
    return <valor>;  
}  
  
double valor = 10.0;  
printf("%c\n", Calcula(10, &valor));
```

```
double valor = 10.0;  
printf("%c\n", Calcula(10, &valor));  
  
char Calcula(int p1, double *p2) {  
    <bloco-função>  
    return <valor>;  
}
```

# Comandos de pseudo-códigos utilizados (e sua transformação)

- Note os comentários `//Supõe` e `//Garante` dos Procedimentos e Funções; apesar de não se transformarem em código, são importantes para especificar como usar a rotina apropriadamente
  - `//Supõe: Condição 1`

Condição 1 é uma expressão lógica que deve ser satisfeita pelo estado da computação quando o procedimento/função for invocado. Representa, portanto, o que o procedimento/função exige como pré-requisito para ser usado
  - `//Garante: Condição 2`

Condição 2 é uma expressão lógica que será satisfeita pelo estado da computação quando o procedimento ou função for finalizado. Representa, portanto, o serviço que o procedimento/função provê. No caso de funções, a palavra 'retorno' será usado como variável em Condição 2 representando o valor de retorno da função.



# Comandos de pseudo-códigos utilizados (e sua transformação)

## Vetores por Referência:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <pre>procedimento Carregar(ref A[]: Inteiro, N: Inteiro)     &lt;bloco-procedimento&gt;  var B[1..100]: Inteiro Carregar(B, 100)</pre> |
| <b>C</b>             | <pre>void Carregar(int A[], int N) {     &lt;bloco-procedimento&gt; }  int B[100]; Carregar(B, 100);</pre>                             |

O nome de um vetor representa um ponteiro para o primeiro elemento do vetor.

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Vetores por Valor:

|               |  |
|---------------|--|
| Pseudo-código | <pre>procedimento Carregar(val A[]: Inteiro, N: Inteiro)     &lt;bloco-procedimento&gt;  var B[1..100]: Inteiro ler (B[1..100]) Carregar(B, 100)</pre>   |
| C             | <pre>void Carregar(int A[], int N) {     &lt;bloco-procedimento&gt; }  int B[100]; int i; for (i = 0; i &lt; 100; i++) {     scanf("%d", &amp;(B[i])); } //copia-se o vetor para um outro auxiliar, pois, em C, //não há possibilidade de passar um vetor por valor int Baux[100]; for (i = 0; i &lt; 100; i++) {     Baux[i] = B[i]; } Carregar(Baux, 100);</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operações agrupadas:

| Pseudo-código | $A[i] \leftarrow \text{máx}\{ A[j] \mid 1 \leq j \leq N \}$   |
|---------------|---|
| <b>C</b>      | <pre>int max = A[0]; int j; for (j = 1; j &lt; N; j++) {     if (A[j] &gt; max) {         max = A[j];     } } A[i] = max;</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

|               |  |
|---------------|--|
| Pseudo-código | <pre>// sortear uniformemente um número entre min e max<br/>a ← Sortear(min, max)</pre>  |
| C             | <pre>#include &lt;stdlib.h&gt;<br/>#include &lt;time.h&gt;<br/><br/>//somente na inicialização do programa<br/>srand(time(NULL));<br/><br/>//quando precisar do número aleatório<br/>a = (rand() % (max - min + 1)) + min;</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

|   |  |
|---|--|
| Pseudo-código   | <pre>// medir o tempo decorrido para executar uma operação t ← ObterDataHora() // fazer alguma coisa escrever(ObterDataHora() - t)</pre>   |
| <b>C</b><br>versão 1<br>obtendo o tempo decorrido de CPU apenas para o processo | <pre>#include &lt;time.h&gt; #include &lt;stdio.h&gt; #include &lt;math.h&gt;  clock_t ticks1, ticks2;  ticks1 = clock();  // fazer alguma coisa  ticks2 = clock(); printf("%0.3fs\n", (double) (ticks2-ticks1)/CLOCKS_PER_SEC);</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

|   |  |
|---|--|
| Pseudo-código                                     | <pre>// medir o tempo decorrido para executar uma operação t ← ObterDataHora() // fazer alguma coisa escrever(ObterDataHora() - t)</pre>   |
| <b>C</b><br>versão 2<br>obtendo o tempo decorrido | <pre>#include &lt;time.h&gt; #include &lt;stdio.h&gt; #include &lt;math.h&gt;  time_t start, stop;  time(&amp;start);  // fazer alguma coisa  time(&amp;stop); printf("%.0fs\n", difftime(stop, start));</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

| Pseudo-código | <pre>// erro durante processamento Exceção("Mensagem")</pre>             |
|---------------|--|
| <b>C</b>      | <pre>#include &lt;stdlib.h&gt;  ...  printf("Mensagem"); exit(-1);</pre> |

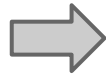
# Linguagem C++



# IMPORTANTE

Se a sua conversão, seguindo os exemplos a seguir, não estiver funcionando, volte a comparar minuciosamente o exemplo dado com seu código convertido. Em 90% dos casos, alguma diferença ou alguma adaptação inexistente está causando o problema. Por exemplo, considere a conversão:

```
var N: ^Inteiro  
alocar(N)
```



```
int * N;  
N = (int *) malloc(sizeof(int));
```

Se ao invés, N for do tipo ponteiro para uma certa estrutura Aluno, a adaptação necessária mais natural seria trocar todas as ocorrências de "int" ("Inteiro") para o nome da estrutura Aluno, assim:

```
var N: ^Aluno  
alocar(N)
```



```
Aluno * N;  
N = (Aluno *) malloc(sizeof(Aluno));
```

# Comandos de pseudo-códigos utilizados (e sua transformação)

## O programa:

| Pseudo-código | programa CalculaAlgo()<br><código do programa>  |
|---------------|---|
| <b>C++</b>    | <pre>// Arquivo CalculaAlgo.cpp #include&lt;stdio.h&gt; #include&lt;stdlib.h&gt; using namespace std; // incluir cabeçalhos de bibliotecas // usadas no código do programa int main(){     &lt;código do programa&gt;     return 0; }</pre> |

Linhas adicionais ou alterações destas podem ser requeridas pela versão do compilador de C!

# Comandos de pseudo-códigos utilizados (e sua transformação)

## **Compilando o programa:**

### **Linux:**

No terminal:

```
> g++ -o <nome_executável> <nome_arquivo.cpp>
```

**Ex:**

```
> g++ -o CalculaAlgo CalculaAlgo.cpp
```

### **Windows:**

Em geral, há um botão ou menu de compilação no editor que se integra ao compilador de C++

# Comandos de pseudo-códigos utilizados (e sua transformação)

## **Executando o programa:**

Depois de gerado o executável do programa, a execução depende do sistema operacional.

## **Windows:**

Seja <DIR\_PROG> o diretório onde o compilador gera o executável

```
C:\<DIR_PROG>> CalculaAlgo.exe
```

ou

```
C:\<DIR_PROG>> CalculaAlgo.exe <entrada.txt
```

```
>saida.txt
```

(entradas do teclado são obtidas do arquivo entrada.txt, e saídas para a tela escritas em saida.txt)

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Executando o programa:

Depois de gerado o executável do programa, a execução depende do sistema operacional.

### Linux:

```
> ./<nome_executável>
```

Ex:

```
> ./CalculaAlgo
```

ou

```
> ./CalculaAlgo <entrada.txt >saida.txt
```

(entradas do teclado são obtidas do arquivo entrada.txt, e saídas para a tela são escritas no arquivo saida.txt)

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Comentários:

| Pseudo-código | // este é um comentário   |
|---------------|---|
| <b>C++</b>    | // este é<br>// um comentário<br><br>/* este é<br>outro comentário */ |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis escalares:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <code>var A: Inteiro, B: Real, C: Lógico, D: Caractere<br/>A ← 10<br/>B ← 20.2<br/>C ← V<br/>D ← 'A'</code> |
| <b>C++</b>           | <code>int A; double B; bool C; char D;<br/>A = 10;<br/>B = 20.2;<br/>C = true;<br/>D = 'A';</code>          |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores Aritméticos Comuns:

| Operador         | Pseudo-código                          | C++   |
|------------------|--|---|
| Soma             | $a + b$                                | $a + b$   |
| Subtração        | $a - b$                                | $a - b$   |
| Multiplicação    | $a * b$                                | $a * b$   |
| Divisão          | $a / b$                                | $a / b$   |
| Divisão Inteira  | $a \div b$<br>(ou $a \text{ div } b$ ) | //a e b inteiros<br>$a / b$                                   |
| Resto da Divisão | $a \bmod b$                            | $a \% b$  |
| Exponenciação    | $a ^ b$                                | <code>#include &lt;cmath&gt;</code><br><code>pow(a, b)</code> |
| Raiz quadrada    | $\sqrt{a}$<br>(ou $\text{sqrt}(a)$ )   | <code>#include &lt;cmath&gt;</code><br><code>sqrt(a)</code>   |



# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores em Cadeias:

| Operador            | Pseudo-código   | C++  |
|---------------------|---|--|
| Declaração          | var S: Cadeia<br>ou<br>var S[1..N]:<br>Caractere                                  | <pre>#include &lt;string&gt; string S;</pre>         |
| Comprimento         | S   | <code>S.length()</code>                              |
| Caractere           | S[i]  | <code>S[i-1]</code>                                  |
| Subcadeia           | $R \leftarrow S[\text{ini}..\text{fim}]$  | <code>R = S.substr(ini-1, fim-ini+1);</code>         |
| Encontrar Subcadeia | $\text{pos} \leftarrow \text{encontrar}(S, \text{SEnc}, \text{aPartirDePosicao})$ | <code>pos = S.find(SEnd, aPartirDePosicao-1);</code> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores em Cadeias:

| Operador                           | Pseudo-código                         | C++   |
|------------------------------------|---------------------------------------|---|
| Concatenação                       | $S3 \leftarrow S1 + S2$               | $S3 = S1 + S2;$   |
| Criar cadeia de outro tipo de dado | $S \leftarrow \text{para\_cadeia}(N)$ | $S = \text{to\_string}(N);$   |
| Criar número de cadeia             | $N \leftarrow \text{de\_cadeia}(S)$   | <pre>#include&lt;sstream&gt; stringstream SS(S); SS &gt;&gt; N;</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Atribuição múltipla:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <code>var A: Inteiro, B: Real, C: Lógico, D: Caractere<br/>A, B, C, D ← 10, 20.2, V, 'A'</code>    |
| <b>C++</b>           | <code>int A; double B; bool C; char D;<br/>A = 10;<br/>B = 20.2;<br/>C = true;<br/>D = 'A';</code> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Atribuição múltipla:

|               |  |
|---------------|--|
| Pseudo-código | //X, Y inteiros<br>$X, Y \leftarrow Y, X$                    |
| C++           | int Temp1 = Y;<br>int Temp2 = X;<br>X = Temp1;<br>Y = Temp2; |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis vetores:

|   |   |
|---|---|
| <b>Pseudo-código</b>  | <code>var A[1..100]: Inteiro<br/>A[1..100] ← 0</code>   |
| <b>C++<br/>(versão 1)</b><br>Alocação Estática (limite para número de elementos relativamente baixo)                      | <code>int A[100];<br/>int i;<br/>for (i = 0; i &lt; 100; i++) {<br/>    A[i] = 0;<br/>}</code>                                |
| <b>C++<br/>(versão 2)</b><br>Alocação Dinâmica (limite para número de elementos é a quantidade de memória contígua livre) | <code>int * A = (int *) malloc(100*sizeof(int));<br/>int i;<br/>for (i = 0; i &lt; 100; i++) {<br/>    A[i] = 0;<br/>}</code> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis vetores (#2):

|   |   |
|---|---|
| Pseudo-código   | <pre>var A[1..100]: Inteiro A[1..100] ← 0</pre>   |
| <b>C++<br/>(versão 1)</b><br>Alocação Estática (limite para número de elementos relativamente baixo)                      | <pre>int A[101]; //o elemento A[0] ficará sem uso int i; for (i = 1; i &lt;= 100; i++) {     A[i] = 0; }</pre>                                |
| <b>C++<br/>(versão 2)</b><br>Alocação Dinâmica (limite para número de elementos é a quantidade de memória contígua livre) | <pre>int * A = (int *) malloc(101*sizeof(int)); //o elemento A[0] ficará sem uso int i; for (i = 1; i &lt;= 100; i++) {     A[i] = 0; }</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis matrizes:

|   |   |
|---|---|
| <b>Pseudo-código</b>  | <b>var A[1..N, 1..N]: Inteiro</b><br><b>A[i, j] ← 0, para todo <math>1 \leq i, j \leq N</math></b>  |
| <b>C++<br/>(versão 1)</b><br>Alocação Estática (limite para número de elementos relativamente baixo)                      | <pre>int A[N][N]; int i; int j; for (i = 0; i &lt; N; i++) {     for (j = 0; j &lt; N; j++) {         A[i][j] = 0;     } }</pre>  |
| <b>C++<br/>(versão 2)</b><br>Alocação Dinâmica (limite para número de elementos é a quantidade de memória contígua livre) | <pre>int i; int j; int ** A = (int **) malloc(N*sizeof(int *)); for (i = 0; i &lt; N; i++) {     A[i] = (int *) malloc(N*sizeof(int)); } for (i = 0; i &lt; N; i++) {     for (j = 0; j &lt; N; j++) {         A[i][j] = 0;     } }</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis matrizes (#2):

|   |  |
|---|--|
| Pseudo-código   | <pre>var A[1..N, 1..N]: Inteiro A[i, j] ← 0, para todo <math>1 \leq i, j \leq N</math></pre>   |
| <b>C++</b><br><b>(versão 1)</b><br>Alocação Estática (limite para número de elementos relativamente baixo)                      | <pre>int A[N+1][N+1]; //A[0,0..N] e A[0..N,0] ficarão sem uso int i; int j; for (i = 1; i &lt;= N; i++) {     for (j = 1; j &lt;= N; j++) {         A[i][j] = 0;     } }</pre>   |
| <b>C++</b><br><b>(versão 2)</b><br>Alocação Dinâmica (limite para número de elementos é a quantidade de memória contígua livre) | <pre>int i; int j; int ** A = (int **) malloc((N+1)*sizeof(int *)); for (i = 1; i &lt;= N; i++) {     A[i] = (int *) malloc((N+1)*sizeof(int)); } //A[0,0..N] e A[0..N,0] ficarão sem uso for (i = 1; i &lt;= N; i++) {     for (j = 1; j &lt;= N; j++) {         A[i][j] = 0;     } }</pre> |



# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis ponteiros:

|               |   |
|---------------|---|
| Pseudo-código | <pre>var N, M: ^Inteiro, i: Inteiro alocar(N) N^ ← 10 M ← @i M^ ← 11 desalocar(N)</pre>             |
| C++           | <pre>int * N, * M, i; N = (int *) malloc(sizeof(int)); *N = 10; M = &amp;i; *M = 11; free(N);</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis estruturas:

|   |  |
|---|--|
| <b>Pseudo-código</b>  | <pre>estrutura Aluno:     Notas[1..N], Matricula: Inteiro     Ouvinte: Lógico     Prox: ^Aluno  N ← 100000 var a1: Aluno, a2: ^Aluno a1.Ouvinte ← F alocar(a2) a1.Prox ← a2</pre>                              |
| <b>C++<br/>(versão 1)</b><br>o tamanho do<br>vetor Notas é fixo,<br>não varia<br>estrutura a<br>estrutura | <pre>#define N 100000 typedef struct Aluno {     int Notas[N];     bool Ouvinte;     struct Aluno *Prox; } Aluno; Aluno a1, *a2; a1.Ouvinte = false; a2 = (Aluno *) malloc(sizeof(Aluno)); a1.Prox = a2;</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis estruturas:

|  |   |
|--|---|
| <b>Pseudo-código</b>   | <pre>estrutura Aluno:     Notas[1..N], Matricula: Inteiro     Ouvinte: Lógico     Prox: ^Aluno  N ← 100 var a1: Aluno a1.Ouvinte ← F N ← 200 var a2: ^Aluno alocar(a2) a1.Prox ← a2</pre>   |
| <b>C++<br/>(versão 2)</b><br>o tamanho do<br>vetor Notas pode<br>variar estrutura a<br>estrutura | <pre>typedef struct Aluno {     int * Notas;     bool Ouvinte;     struct Aluno *Prox; } Aluno; int N = 100; Aluno a1; a1.Notas = (int *) malloc(N * sizeof(int)); a1.Ouvinte = false; N = 200; Aluno *a2; a2 = (Aluno *) malloc(sizeof(Aluno)); (*a2).Notas = (int *) malloc(N * sizeof(int)); a1.Prox = a2;</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

- Declarações de estruturas devem vir antes do seu uso!

```
typedef struct Aluno {  
    bool Ouvinte;  
} Aluno;
```

```
Aluno a1, *a2;  
a1.Ouvinte = false;
```

```
Aluno a1, *a2;  
a1.Ouvinte = false;
```

```
typedef struct Aluno {  
    bool Ouvinte;  
} Aluno;
```

# Comandos de pseudo-códigos utilizados (e sua transformação)

- A navegação "ponteiro, campo de estrutura, ponteiro, campo de estrutura, etc." tem uma notação especial por conveniência:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>estrutura Aluno:     Matricula: Inteiro     Prox: ^Aluno  var L: ^Aluno ... // escrever a 3.a matrícula da lista escrever(L^.Prox^.Prox^.Matricula)</pre>  |
| <b>C++</b>           | <pre>typedef struct Aluno {     int Matricula;     struct Aluno *Prox; } Aluno; Aluno *L; ... // escrever a 3.a matrícula da lista cout &lt;&lt; L-&gt;Prox-&gt;Prox-&gt;Matricula &lt;&lt; "\n"; // mais conveniente, embora equivalente, a // cout &lt;&lt; (*( *(*L).Prox).Prox).Matricula) &lt;&lt; "\n";</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Escrita de variáveis:

|               |  |
|---------------|--|
| Pseudo-código | escrever ("O maior valor é ", valormax)  |
| C++           | <pre>#include&lt;iostream&gt;  cout &lt;&lt; "O maior valor é " &lt;&lt; valormax &lt;&lt; "\n";</pre> |

Note que o tipo do dado e outros detalhes (como quebra de linha) devem ser considerados, assim como a formatação estética da saída.

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Escrita de variáveis:

| Pseudo-código | escrever(C[1..N])  |
|---------------|--|
| <b>C++</b>    | <pre>#include&lt;iostream&gt;  int i; for (i=0; i &lt; N; i++) {     cout &lt;&lt; C[i]; } cout &lt;&lt; "\n";</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Leitura de variáveis:

| Pseudo-código | ler(N, C[1..N])  |
|---------------|--|
| <b>C++</b>    | <pre>#include&lt;iostream&gt;  cin &gt;&gt; N;  int i; for (i = 0; i &lt; N; i++) {     cin &gt;&gt; C[i]; }</pre> |



# Comandos de pseudo-códigos utilizados (e sua transformação)

## Leitura de variáveis:

| Pseudo-código | <pre>var Nome[1..TAM_MAX]: Caractere<br/>ler(Nome[1..TAM_MAX])</pre>  |
|---------------|---|
| <b>C++</b>    | <pre>#include&lt;iostream&gt;<br/>#include&lt;string&gt;<br/><br/>string Nome;<br/>cin &gt;&gt; Nome;</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Condicional:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>se condição1 então     &lt;bloco-então&gt; senão se condição2 então     &lt;bloco-senão-se&gt; senão     &lt;bloco-senão&gt;</pre>   |
| <b>C++</b>           | <pre>if (condição1) {     &lt;bloco-então&gt; } else if {condição2) {     &lt;bloco-senão-se&gt; } else {     &lt;bloco-senão&gt; }</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores Lógicos Comuns:

| Operador       | Pseudo-código      | C++ |
|----------------|--------------------|-----|
| Igual          | =                  | ==  |
| Diferente      | ≠                  | !=  |
| Menor          | <                  | <   |
| Maior          | >                  | >   |
| Menor ou Igual | ≤                  | <=  |
| Maior ou Igual | ≥                  | >=  |
| E              | E ou $\wedge$      | &&  |
| OU             | OU ou $\vee$       |     |
| NÃO            | NÃO ou ! ou $\neg$ | !   |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Repetição (para):

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <code>para i ← 1 até N passo 2 faça<br/>  &lt;bloco-para&gt;</code>                 |
| <b>C++</b>           | <code>for (int i = 1; i &lt;= N; i = i + 2) {<br/>  &lt;bloco-para&gt;<br/>}</code> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Repetição (enquanto):

| Pseudo-código | enquanto condição faça<br><bloco-enquanto>                         |
|---------------|--|
| <b>C++</b>    | <pre>while (condição) {<br/>    &lt;bloco-enquanto&gt;<br/>}</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Procedimentos:

|               |  |
|---------------|--|
| Pseudo-código | <pre>procedimento Calcula(val p1: Inteiro, ref p2: Real) //Supõe: Condição 1 //Garante: Condição 2     &lt;bloco-procedimento&gt;  var valor: Real ← 10 Calcula(10, valor)</pre> |
| C++           | <pre>void Calcula(int p1, double &amp;p2) {     &lt;bloco-procedimento&gt; }  double valor = 10.0; Calcula(10, valor);</pre>   |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Funções:

|               |   |
|---------------|---|
| Pseudo-código | <pre>função Calcula(val p1: Inteiro, ref p2: Real): Caractere //Supõe: Condição 1 //Garante: Condição 2, onde 'retorno' é o valor de retorno     &lt;bloco-função&gt;     retornar (&lt;valor&gt;)  var valor: Real ← 10 escrever(Calcula(10, valor))</pre> |
| C++           | <pre>char Calcula(int p1, double &amp;p2) {     &lt;bloco-função&gt;     return &lt;valor&gt;; }  double valor = 10.0; printf("%c\n", Calcula(10, valor));</pre>  |

# Comandos de pseudo-códigos utilizados (e sua transformação)

- Declarações de procedimentos e funções devem vir antes do seu uso!

```
char Calcula(int p1, double &p2) {  
    <bloco-função>  
    return <valor>;  
}  
  
double valor = 10.0;  
printf("%c\n", Calcula(10, valor));
```

```
double valor = 10.0;  
printf("%c\n", Calcula(10, valor));  
  
char Calcula(int p1, double &p2) {  
    <bloco-função>  
    return <valor>;  
}
```



# Comandos de pseudo-códigos utilizados (e sua transformação)

## Vetores por Referência:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>procedimento Carregar(ref A[: Inteiro, N: Inteiro)     &lt;bloco-procedimento&gt;  var B[1..100]: Inteiro Carregar(B, 100)</pre> |
| <b>C++</b>           | <pre>void Carregar(int A[], int N) {     &lt;bloco-procedimento&gt; }  int B[100]; Carregar(B, 100);</pre>                            |

O nome de um vetor representa um ponteiro para o primeiro elemento do vetor.

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Vetores por Valor:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>procedimento Carregar(val A[]: Inteiro, N: Inteiro)     &lt;bloco-procedimento&gt;  var B[1..100]: Inteiro ler (B[1..100]) Carregar(B, 100)</pre>  |
| <b>C++</b>           | <pre>void Carregar(int A[], int N) {     &lt;bloco-procedimento&gt; }  int B[100]; int i; for (i = 0; i &lt; 100; i++) {     cin &gt;&gt; B[i]; } //copia-se o vetor para um outro auxiliar, pois, em C++, //não há possibilidade de passar um vetor por valor int Baux[100]; for (i = 0; i &lt; 100; i++) {     Baux[i] = B[i]; } Carregar(Baux, 100);</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operações agrupadas:

| Pseudo-código | $A[i] \leftarrow \max\{ A[j] \mid 1 \leq j \leq N \}$   |
|---------------|---|
| <b>C++</b>    | <pre>#include&lt;iostream&gt;  int vmax = A[0]; for (int j = 1; j &lt; N; j++) {     vmax = max(vmax, A[j]); } A[i] = vmax;</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

| Pseudo-código | <pre>// sortear uniformemente um número entre min e max<br/>a ← Sortear(min, max)</pre>  |
|---------------|--|
| C             | <pre>#include &lt;stdlib.h&gt;<br/>#include &lt;time.h&gt;<br/><br/>//somente na inicialização do programa<br/>srand(time(NULL));<br/><br/>//quando precisar do número aleatório<br/>a = (rand() % (max - min + 1)) + min;</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

|   |  |
|---|--|
| Pseudo-código   | <pre>// medir o tempo decorrido para executar uma operação t ← ObterDataHora() // fazer alguma coisa escrever(ObterDataHora() - t)</pre>   |
| <b>C</b><br>versão 1<br>obtendo o tempo decorrido de CPU apenas para o processo | <pre>#include &lt;time.h&gt; #include &lt;stdio.h&gt; #include &lt;math.h&gt;  clock_t ticks1, ticks2;  ticks1 = clock();  // fazer alguma coisa  ticks2 = clock(); printf("%0.3fs\n", (double) (ticks2-ticks1)/CLOCKS_PER_SEC);</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

|   |  |
|---|--|
| Pseudo-código                                     | <pre>// medir o tempo decorrido para executar uma operação t ← ObterDataHora() // fazer alguma coisa escrever(ObterDataHora() - t)</pre>   |
| <b>C</b><br>versão 2<br>obtendo o tempo decorrido | <pre>#include &lt;time.h&gt; #include &lt;stdio.h&gt; #include &lt;math.h&gt;  time_t start, stop;  time(&amp;start);  // fazer alguma coisa  time(&amp;stop); printf("%.0fs\n", difftime(stop, start));</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

|               |  |
|---------------|--|
| Pseudo-código | // erro durante processamento<br>Exceção("Mensagem")   |
| C++           | <pre>#include &lt;stdlib.h&gt;  ...  cout &lt;&lt; "Mensagem" &lt;&lt; "\n"; exit(-1);</pre> |

# **Linguagem Python**



# Comandos de pseudo-códigos utilizados (e sua transformação)

## O programa:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <code>programa CalculaAlgo()<br/>    &lt;código do programa&gt;</code> |
| <b>Python</b>        | <code>// Arquivo CalculaAlgo.py<br/>&lt;código do programa&gt;</code>  |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## **Executando o programa:**

Depois de gerado o arquivo de script do programa (digamos, CalculaAlgo.py), a execução depende do sistema operacional.

## **Windows:**

Seja <DIR\_PYTHON> o diretório onde o compilador está instalado

```
C:\<DIR_PYTHON>\python.exe CalculaAlgo.py
```

ou

```
C:\<DIR_PYTHON>\python.exe CalculaAlgo.py
```

```
<entrada.txt >saida.txt
```

(entradas do teclado são obtidas do arquivo entrada.txt, e saídas para a tela escritas em saida.txt)

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Executando o programa:

Depois de gerado o arquivo de script do programa (digamos, `CalculaAlgo.py`), a execução depende do sistema operacional.

## Linux:

```
$ python CalculaAlgo.py
```

ou

```
$ python CalculaAlgo.py <entrada.txt >saida.txt
```

(entradas do teclado são obtidas do arquivo `entrada.txt`, e saídas para a tela escritas em `saida.txt`)

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Comentários:

| Pseudo-código | <code>// este é um comentário</code>   |
|---------------|--|
| Python        | <pre># este é<br/># um comentário<br/><br/>""" este é outro<br/>comentário """</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis escalares:

|               |   |
|---------------|---|
| Pseudo-código | <pre>var A: Inteiro, B: Real, C: Lógico, D: Caractere<br/>A ← 10<br/>B ← 20.2<br/>C ← V<br/>D ← 'A'</pre> |
| Python        | <pre>A = 10<br/>B = 20.2<br/>C = True<br/>D = "A"</pre>   |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores Aritméticos Comuns:

| Operador         | Pseudo-código                            | Python               |
|------------------|--|----------------------|
| Soma             | $a + b$                                  | <code>a + b</code>   |
| Subtração        | $a - b$                                  | <code>a - b</code>   |
| Multiplicação    | $a * b$                                  | <code>a * b</code>   |
| Divisão          | $a / b$                                  | <code>a / b</code>   |
| Divisão Inteira  | $a \div b$<br>(ou <code>a div b</code> ) | <code>a // b</code>  |
| Resto da Divisão | $a \bmod b$                              | <code>a % b</code>   |
| Exponenciação    | $a ^ b$                                  | <code>a ** b</code>  |
| Raiz quadrada    | $\sqrt{a}$<br>(ou <code>sqrt(a)</code> ) | <code>sqrt(a)</code> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores em Cadeias:

| Operador                           | Pseudo-código  | Python   |
|------------------------------------|--|--|
| Declaração                         | <code>var S: Cadeia</code><br>ou<br><code>var S[1..N]:</code><br>Caractere | não existe declaração antes da atribuição            |
| Comprimento                        | <code> S </code>   | <code>len(S)</code>                                  |
| Caractere                          | <code>S[5]</code>  | <code>S[4]</code>                                    |
| Subcadeia                          | <code>S[5..9]</code>   | <code>S[5:10]</code>                                 |
| Encontrar Subcadeia                | <code>encontrar(S, SEnc,</code><br><code>aPartirDePosicao)</code>          | <code>S.index(SEnc, aPartirDePosicao - 1)</code>     |
| Concatenação                       | <code>S1 + S2</code>   | <code>S1 + S2</code>                                 |
| Criar cadeia de outro tipo de dado | <code>S ← para_cadeia(N)</code>  | <code>S = str(N)</code>                              |
| Criar número de cadeia             | <code>N ← de_cadeia(S)</code>  | <code>N = int(S)</code><br><code>N = float(S)</code> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Atribuição múltipla:

|               |   |
|---------------|---|
| Pseudo-código | <code>var A: Inteiro, B: Real, C: Lógico, D: Caractere<br/>A, B, C, D ← 10, 20.2, V, 'A'</code> |
| Python        | <code>A, B, C, D = 10, 20.2, True, "A"</code>   |



# Comandos de pseudo-códigos utilizados (e sua transformação)

## Atribuição múltipla:

|               |                        |
|---------------|------------------------|
| Pseudo-código | $X, Y \leftarrow Y, X$ |
| Python        | $X, Y = Y, X$          |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis vetores:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <code>var A[1..100]: Inteiro</code><br><code>A[1..100] ← 0</code>                            |
| <b>Python</b>        | <code>A = [0 for i in range(100)]</code><br><code>#A[0] = 0, A[1] = 0, ..., A[99] = 0</code> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis matrizes:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <code>var A[1..N, 1..N]: Inteiro</code><br><code>A[i, j] ← 0, para todo <math>1 \leq i, j \leq N</math></code>  |
| <b>Python</b>        | <pre>A = [[0 for i in range(N)] for j in range(N)]<br/>#A[0][0] = 0, A[0][1] = 0, ..., A[0][N-1] = 0<br/>#A[1][0] = 0, A[1][1] = 0, ..., A[1][N-1] = 0<br/>#...<br/>#A[N-1][0] = 0, A[N-1][1] = 0, ..., A[N-1][N-1] = 0</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis ponteiros:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <pre>var N, M: ^Inteiro, i: Inteiro alocar(N) N^ ← 10 M ← @i M^ ← 11 desalocar(N)</pre>  |
| <b>Python</b>        | <pre>class Ponteiro(object):     def __init__(self, valor):         self.A = valor #A representa o valor apontado N, M, i = Ponteiro(None), Ponteiro(None), Ponteiro(None) N.A = 10 M.A = i M.A.A = 11 N = None #Garbage Collector devolverá a memória</pre> |

Um ponteiro em Python é sempre uma referência a um objeto. Portanto, todo dado referenciado por um ponteiro deve ser transformado em um objeto.

# Comandos de pseudo-códigos utilizados (e sua transformação)

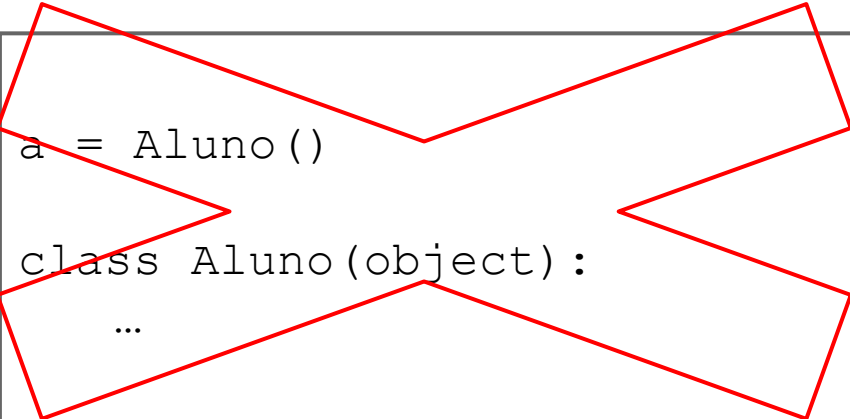
## Declaração/Atribuição de variáveis estruturas:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>estrutura Aluno:     Notas[1..N], Matricula: Inteiro     Ouvinte: Lógico     Prox: ^Aluno  var a1: Aluno, a2: ^Aluno a1.Ouvinte ← F alocar(a2) a1.Prox ← a2</pre>  |
| <b>Python</b>        | <pre>N = 1000 class Aluno(object):     def __init__(self):         self.Notas = [0 for i in range(N)]         self.Ouvinte = True         self.Prox = None a1, a2 = Aluno(), Aluno() # não há suporte em Python para a1 ser o objeto em si, # apenas uma referência a ele a1.Ouvinte = False a1.Prox = a2</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

- Declarações de estruturas devem vir antes do seu uso!

```
class Aluno(object):  
    ...  
a = Aluno()
```



```
a = Aluno()  
class Aluno(object):  
    ...
```

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Escrita de variáveis:

| Pseudo-código | <code>escrever ("O maior valor é ", valormax)</code>   |
|---------------|--|
| Python        | <pre>print "O maior valor é", valormax  ou  print "O maior valor é " + str(valormax)  ou  print "O maior valor é %s" % (str(valormax))</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Escrita de variáveis:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <pre>var C[1..N]: Inteiro ... escrever (C[1..N])</pre>   |
| <b>Python</b>        | <pre>C = [0 for i in range(N)] ... for i in range(N):     print "C(%s) = %s\n" % (str(i+1), str(C[i]))</pre> |



# Comandos de pseudo-códigos utilizados (e sua transformação)

## Leitura de variáveis:

| Pseudo-código | ler(N, C[1..N])  |
|---------------|--|
| Python        | <pre>N = int(raw_input("Entre com o tamanho do vetor:")) C = [0 for i in range(N)] for i in range(N):     C[i] = int(raw_input("Entre com C("+str(i+1)+"): "))</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Condicional:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>se condição1 então     &lt;bloco-então&gt; senão se condição2 então     &lt;bloco-senão-se&gt; senão     &lt;bloco-senão&gt;</pre> |
| <b>Python</b>        | <pre>if condição1:     &lt;bloco-então&gt; elif condição2:     &lt;bloco-senão-se&gt; else:     &lt;bloco-senão&gt;</pre>               |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores Lógicos Comuns:

| Operador       | Pseudo-código      | Python |
|----------------|--------------------|--------|
| Igual          | =                  | ==     |
| Diferente      | ≠                  | !=     |
| Menor          | <                  | <      |
| Maior          | >                  | >      |
| Menor ou Igual | ≤                  | <=     |
| Maior ou Igual | ≥                  | >=     |
| E              | E ou $\wedge$      | and    |
| OU             | OU ou $\vee$       | or     |
| NÃO            | NÃO ou ! ou $\neg$ | not    |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Repetição (para):

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <code>para i ← 1 até N passo 2 faça</code><br><code>&lt;bloco-para&gt;</code> |
| <b>Python</b>        | <code>for i in range(1, N+1, 2):</code><br><code>&lt;bloco-para&gt;</code>    |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Repetição (enquanto):

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <code>enquanto condição faça</code><br><code>&lt;bloco-enquanto&gt;</code> |
| <b>Python</b>        | <code>while condição:</code><br><code>&lt;bloco-enquanto&gt;</code>        |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Procedimentos:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>procedimento Calcula(val p1: Inteiro, ref p2: Real) //Supõe: Condição 1 //Garante: Condição 2     &lt;bloco-procedimento&gt;  var valor: Real valor ← 10.0 Calcula(10, valor) escrever (valor)</pre> |
| <b>Python</b>        | <pre>def Calcula(p1, p2):     #p2[0] deve ser usado no lugar de p2 abaixo     &lt;bloco-procedimento&gt; valor = [10.0] Calcula(10, valor) print valor[0]</pre>   |

# Comandos de pseudo-códigos utilizados (e sua transformação)

- Em Python, há distinção de valores classificados pelo sistema de tipos do Python como mutáveis e imutáveis. Valores mutáveis modificados dentro de um procedimento/função refletem a modificação no procedimento/função que fez a chamada, valores imutáveis não refletem. Portanto:
  - valores imutáveis passados por referência devem ser convertidos pelo cliente para um valor mutável, por exemplo, encapsulando o valor dentro de um vetor
  - valores mutáveis passados por valor devem ser copiados dentro do procedimento/função para que modificações no valor não reflitam no cliente
- Em linhas gerais, tipos primitivos são imutáveis; vetores, objetos (ponteiros a estruturas) são mutáveis

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Funções:

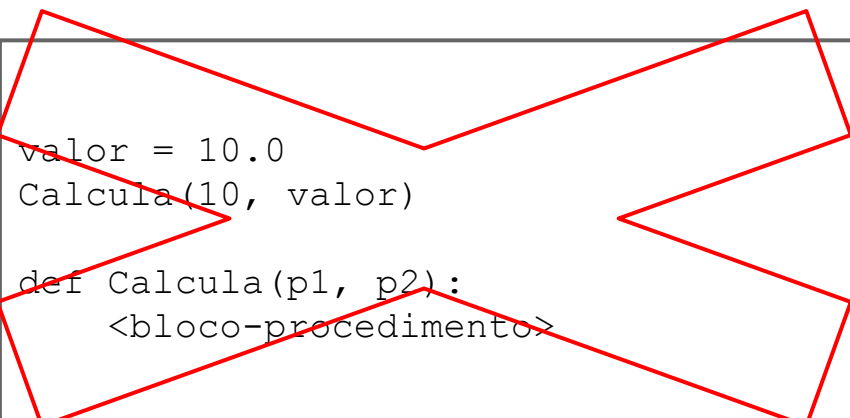
|               |  |
|---------------|--|
| Pseudo-código | <pre>função Calcula(val p1: Inteiro, ref p2: Real): Caractere //Supõe: Condição 1 //Garante: Condição 2     &lt;bloco-função&gt;     retornar (&lt;valor&gt;)  var valor: Real ← 10 escrever(Calcula(10, valor))</pre> |
| Python        | <pre>def Calcula(p1c, p2):     p1 = p1c #p1 será usado por Calcula     &lt;bloco-função&gt;     return &lt;valor&gt; valor = 10.0 print "%s\n" % Calcula(10, valor)</pre>  |



# Comandos de pseudo-códigos utilizados (e sua transformação)

- Declarações de procedimentos e funções devem vir antes do seu uso!

```
def Calcula(p1, p2):  
    <bloco-procedimento>  
  
valor = 10.0  
Calcula(10, valor)
```



```
valor = 10.0  
Calcula(10, valor)  
  
def Calcula(p1, p2):  
    <bloco-procedimento>
```

# Comandos de pseudo-códigos utilizados (e sua transformação)

- Note os comentários `//Supõe` e `//Garante` dos Procedimentos e Funções; apesar de não se transformarem em código, são importantes para especificar como usar a rotina apropriadamente

- `//Supõe: Condição 1`

Condição 1 é uma expressão lógica que deve ser satisfeita pelo estado da computação quando o procedimento/função for invocado. Representa, portanto, o que o procedimento/função exige como pré-requisito para ser usado

- `//Garante: Condição 2`

Condição 2 é uma expressão lógica que será satisfeita pelo estado da computação quando o procedimento ou função for finalizado. Representa, portanto, o serviço que o procedimento/função provê. No caso de funções, a palavra 'retorno' será usado como variável em Condição 2 representando o valor de retorno da função.

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Vetores por Referência:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>procedimento Carregar(ref A[: Inteiro, N: Inteiro)     &lt;bloco-procedimento&gt;  var B[1..100]: Inteiro Carregar(B, 100)</pre> |
| <b>Python</b>        | <pre>def Carregar(A, N):     &lt;bloco-procedimento&gt;  B = [0 for i in range(100)] Carregar(B, 100)</pre>                           |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Vetores por Valor:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <pre>procedimento Carregar(val A[]: Inteiro, N: Inteiro)     &lt;bloco-procedimento&gt;  var B[1..100]: Inteiro ler (B[1..100]) Carregar(B, 100)</pre>   |
| <b>Python</b>        | <pre>def Carregar(Ac, N):     A = [Ac[i] for i in range(N)]     #A será usado no bloco abaixo     &lt;bloco-procedimento&gt;  B = [0 for i in range(100)] for i in range(N):     B[i] = int(raw_input("Entre com B(" + str(i) + "): ")) Carregar(B, 100)</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operações agrupadas:

|               |  |
|---------------|--|
| Pseudo-código | <pre>// vetor A(1..N)<br/>A[i] ← máx{ A[j]   1 ≤ j ≤ N }</pre> |
| Python        | <pre>A[i] = max([A[j] for j in range(N)])</pre>                |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

|               |   |
|---------------|---|
| Pseudo-código | <pre>// sortear uniformemente um número entre min e max<br/>a ← Sortear(min, max)</pre> |
| Python        | <pre>from random import randint<br/><br/>a = randint(min, max)</pre>                    |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

|               |   |
|---------------|---|
| Pseudo-código | <pre>// medir o tempo decorrido para executar uma operação<br/>t ← ObterDataHora()<br/>// fazer alguma coisa<br/>escrever(ObterDataHora() - t)</pre>  |
| Python        | <pre>from datetime import datetime<br/><br/>to = datetime.now()<br/><br/>// fazer alguma coisa<br/><br/>dt = datetime.now() - to<br/>print "%s" % (dt.days*24*60*60 + dt.seconds)*1000 +<br/>            dt.microseconds/1000.0</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

|               |  |
|---------------|--|
| Pseudo-código | // erro durante processamento<br>Exceção("Mensagem") |
| Python        | <pre>import sys<br/><br/>sys.exit("Mensagem")</pre>  |



# **Linguagem JavaScript**

# Comandos de pseudo-códigos utilizados (e sua transformação)

## O programa:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <code>programa CalculaAlgo()<br/>  &lt;código do programa&gt;</code>   |
| <b>JavaScript</b>    | <code>// Arquivo CalculaAlgo.html<br/>&lt;script type="text/javascript" &gt;<br/>  &lt;código do programa&gt;<br/>&lt;/script&gt;</code> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## **Executando o programa:**

Depois de gerado o arquivo do programa (digamos, `CalculaAlgo.html`), abrir em um navegador (browser).

NOTA: É necessário que a execução de JavaScript esteja habilitada no browser (normalmente, estará).

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Comentários:

| Pseudo-código | // este é um comentário   |
|---------------|---|
| JavaScript    | // este é<br>// um comentário<br><br>/* este é outro<br>comentário */ |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis escalares:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <code>var A: Inteiro, B: Real, C: Lógico, D: Caractere</code><br><code>A ← 10</code><br><code>B ← 20.2</code><br><code>C ← V</code><br><code>D ← 'A'</code> |
| <b>JavaScript</b>    | <code>var A, B, C, D;</code><br><code>A = 10;</code><br><code>B = 20.2;</code><br><code>C = true;</code><br><code>D = "A";</code>                           |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores Aritméticos Comuns:

| Operador         | Pseudo-código                            | JavaScript                     |
|------------------|--|--------------------------------|
| Soma             | $a + b$                                  | <code>a + b</code>             |
| Subtração        | $a - b$                                  | <code>a - b</code>             |
| Multiplicação    | $a * b$                                  | <code>a * b</code>             |
| Divisão          | $a / b$                                  | <code>a / b</code>             |
| Divisão Inteira  | $a \div b$<br>(ou <code>a div b</code> ) | <code>Math.floor(a / b)</code> |
| Resto da Divisão | $a \bmod b$                              | <code>a % b</code>             |
| Exponenciação    | $a ^ b$                                  | <code>Math.pow(a, b)</code>    |
| Raiz quadrada    | $\sqrt{a}$<br>(ou <code>sqrt(a)</code> ) | <code>Math.sqrt(a)</code>      |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores em Cadeias:

| Operador                           | Pseudo-código  | JavaScript   |
|------------------------------------|--|--|
| Declaração                         | <code>var S: Cadeia<br/>ou<br/>var S(1..N):<br/>Caractere</code> | <code>var S;</code>                                |
| Comprimento                        | <code> S </code>   | <code>S.length</code>                              |
| Caractere                          | <code>S(5)</code>  | <code>S[4]</code>                                  |
| Subcadeia                          | <code>S(5..9)</code>   | <code>S.substring(5, 10)</code>                    |
| Encontrar Subcadeia                | <code>encontrar(S, SEnc,<br/>aPartirDePosicao)</code>            | <code>S.indexOf(SEnc, aPartirDePosicao - 1)</code> |
| Concatenação                       | <code>S1 + S2</code>   | <code>S1 + S2</code>                               |
| Criar cadeia de outro tipo de dado | <code>S ← para_cadeia(N)</code>                                  | <code>S = String(N)</code>                         |
| Criar número de cadeia             | <code>N ← de_cadeia(S)</code>                                    | <code>N = (+S)</code>                              |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Atribuição múltipla:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <code>var A: Inteiro, B: Real, C: Lógico, D: Caractere<br/>A, B, C, D ← 10, 20.2, V, 'A'</code> |
| <b>JavaScript</b>    | <code>var A, B, C, D;<br/>A = 10;<br/>B = 20.2;<br/>C = true;<br/>D = "A";</code>               |



# Comandos de pseudo-códigos utilizados (e sua transformação)

## Atribuição múltipla:

| Pseudo-código | $X, Y \leftarrow Y, X$   |
|---------------|--|
| JavaScript    | <pre>var Temp1 = Y;<br/>var Temp2 = X;<br/>X = Temp1;<br/>Y = Temp2;</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis vetores:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <code>var A[1..100]: Inteiro<br/>A[1..100] ← 0</code>  |
| <b>JavaScript</b>    | <code>A = []; A[99] = undefined;<br/>for (var i = 0; i &lt; 100; i++) {<br/>    A[i] = 0;<br/>}</code> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis matrizes:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>var A[1..N, 1..N]: Inteiro A[i, j] ← 0, para todo <math>1 \leq i, j \leq N</math></pre>  |
| <b>JavaScript</b>    | <pre>A = []; A[N-1] = undefined; for (var i = 0; i &lt; N; i++) {   A[i] = []; A[i][N-1] = undefined;   for (var j = 0; j &lt; N; j++) {     A[i][j] = 0;   } }</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis ponteiros:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <pre>var N, M: ^Inteiro, i: Inteiro alocar(N) N^ ← 10 M ← @i M^ ← 11 desalocar(N)</pre>  |
| <b>JavaScript</b>    | <pre>function Ponteiro() {     this.A = undefined; //A representa o valor apontado } var N = new Ponteiro(), M = new Ponteiro(); var i = new Ponteiro(); N.A = 10; M.A = i; M.A.A = 11; N = undefined; //Garbage Collector devolverá a memória</pre> |

Um ponteiro em JavaScript é sempre uma referência a um objeto. Portanto, todo dado referenciado por um ponteiro deve ser transformado em um objeto.

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis estruturas:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>estrutura Aluno:     Notas[1..N], Matricula: Inteiro     Ouvinte: Lógico     Prox: ^Aluno  var a1: Aluno, a2: ^Aluno a1.Ouvinte ← F alocar(a2) a1.Prox ← a2</pre>  |
| <b>JavaScript</b>    | <pre>var N = 1000; function Aluno() {     this.Notas = []; this.Notas[N-1] = undefined;     this.Matricula = undefined; this.Ouvinte = undefined;     this.Prox = undefined; } var a1 = new Aluno(), a2 = new Aluno(); // não há suporte em JavaScript para a1 ser o objeto em si, // apenas uma referência a ele a1.Ouvinte = false; a1.Prox = a2;</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

- Declarações de estruturas podem vir antes ou depois do seu uso!

```
function Aluno() {  
    ...  
}  
  
a = new Aluno();
```

```
a = new Aluno();  
  
function Aluno() {  
    ...  
}
```

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Escrita de variáveis:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <code>escrever ("O maior valor é ", valormax)</code>       |
| <b>JavaScript</b>    | <code>alert("O maior valor é " + String(valormax));</code> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Escrita de variáveis:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <pre>var C[1..N]: Inteiro ... escrever (C[1..N])</pre>   |
| <b>JavaScript</b>    | <pre>var C = []; C[N-1] = undefined; ... for (var i = 0; i &lt; N; i++) {     alert("C("+String(i+1)+") = "+String(C[i])); }</pre> |



# Comandos de pseudo-códigos utilizados (e sua transformação)

## Leitura de variáveis:

| Pseudo-código | ler(N, C[1..N])  |
|---------------|--|
| JavaScript    | <pre>N = (+prompt("Entre com o tamanho do vetor:"));<br/>var C = []; C[N-1] = undefined;<br/>for (var i = 0; i &lt; N; i++) {<br/>    C[i] = (+prompt("Entre com C("+String(i+1)+") :"))<br/>}</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Condicional:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>se condição1 então   &lt;bloco-então&gt; senão se condição2 então   &lt;bloco-senão-se&gt; senão   &lt;bloco-senão&gt;</pre>   |
| <b>JavaScript</b>    | <pre>if (condição1) {   &lt;bloco-então&gt; } else if (condição2) {   &lt;bloco-senão-se&gt; } else {   &lt;bloco-senão&gt; }</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores Lógicos Comuns:

| Operador       | Pseudo-código | JavaScript  |
|----------------|---------------|---|
| Igual          | =             | === (compara tipos dos dados)<br>ou<br>== (não compara tipos dos dados) |
| Diferente      | ≠             | !== (compara tipos dos dados)<br>ou<br>!= (não compara tipos dos dados) |
| Menor          | <             | <   |
| Maior          | >             | >   |
| Menor ou Igual | ≤             | <=  |
| Maior ou Igual | ≥             | >=  |
| E              | E ou ∧        | &&  |
| OU             | OU ou ∨       |   |
| NÃO            | NÃO ou ! ou ¬ | !   |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Repetição (para):

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | para $i \leftarrow 1$ até $N$ passo 2 faça<br><bloco-para>           |
| <b>JavaScript</b>    | for (var $i = 1$ ; $i \leq N$ ; $i = i + 2$ ) {<br><bloco-para><br>} |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Repetição (enquanto):

| Pseudo-código | enquanto condição faça<br><bloco-enquanto>                         |
|---------------|--|
| JavaScript    | <pre>while (condição) {<br/>    &lt;bloco-enquanto&gt;<br/>}</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Procedimentos:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>procedimento Calcula(val p1: Inteiro, ref p2: Real) //Supõe: Condição 1 //Garante: Condição 2     &lt;bloco-procedimento&gt;  var valor: Real ← 10 Calcula(10, valor)</pre>                                    |
| <b>JavaScript</b>    | <pre>function Calcula(p1, p2ref) {     //trocar o uso de p2 em Calcula por p2ref.A     &lt;bloco-procedimento&gt; } var valor = 10.0; var valorref = {'A': valor}; Calcula(10, valorref); valor = valorref.A;</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

- Em JavaScript, todos os parâmetros são por valor. Caso um valor seja modificado dentro de um procedimento/função e a modificação deve ser refletida no procedimento/função que fez a chamada, devemos criar uma referência ao valor explicitamente antes da chamada ser feita, e passar tal referência ao procedimento/função, que pode usá-la para alterar o valor.

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Funções:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <pre>função Calcula(val p1: Inteiro, ref p2: Real): Caractere //Supõe: Condição 1 //Garante: Condição 2     &lt;bloco-função&gt;     retornar (&lt;valor&gt;)  var valor: Real ← 10 escrever(Calcula(10, valor))</pre>                               |
| <b>JavaScript</b>    | <pre>function Calcula(p1, p2ref) {     //trocar o uso de p2 em Calcula por p2ref.A     &lt;bloco-função&gt;     return &lt;valor&gt;; } var valor = 10.0; var valorref = {'A': valor}; console.log(Calcula(10, valorref)); valor = valorref.A;</pre> |



# Comandos de pseudo-códigos utilizados (e sua transformação)

- Declarações de procedimentos e funções podem vir antes ou depois do seu uso!

```
function Calcula(p1) {  
    <bloco-procedimento>  
}  
  
Calcula(10);
```

```
Calcula(10);  
  
function Calcula(p1) {  
    <bloco-procedimento>  
}
```

# Comandos de pseudo-códigos utilizados (e sua transformação)

- Note os comentários `//Supõe` e `//Garante` dos Procedimentos e Funções; apesar de não se transformarem em código, são importantes para especificar como usar a rotina apropriadamente
  - `//Supõe: Condição 1`

Condição 1 é uma expressão lógica que deve ser satisfeita pelo estado da computação quando o procedimento/função for invocado. Representa, portanto, o que o procedimento/função exige como pré-requisito para ser usado
  - `//Garante: Condição 2`

Condição 2 é uma expressão lógica que será satisfeita pelo estado da computação quando o procedimento ou função for finalizado. Representa, portanto, o serviço que o procedimento/função provê. No caso de funções, a palavra 'retorno' será usado como variável em Condição 2 representando o valor de retorno da função.

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Vetores por Referência:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>procedimento Carregar(ref A[: Inteiro, N: Inteiro)     &lt;bloco-procedimento&gt;  var B[1..100]: Inteiro Carregar(B, 100)</pre> |
| <b>JavaScript</b>    | <pre>function Carregar(A, N) {     &lt;bloco-procedimento&gt; } var B = []; B[99] = undefined; Carregar(B, 100);</pre>                |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Vetores por Valor:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>procedimento Carregar(val A[]: Inteiro, N: Inteiro)     &lt;bloco-procedimento&gt;  var B[1..100]: Inteiro ler (B[1..100]) Carregar(B, 100)</pre>  |
| <b>JavaScript</b>    | <pre>function Carregar(Ac, N) {     var A = []; A[Ac.length-1] = undefined;     for (var i = 0; i &lt; Ac.length; i++) {         A[i] = Ac[i];     }     &lt;bloco-procedimento&gt; //A será usado por Carregar } var B = []; B[99] = undefined; for (var i = 0; i &lt; 100; i++) {     B[i] =(+prompt("Entre com B(" + String(i+1) + "):")); } Carregar(B, 100);</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operações agrupadas:

|               |  |
|---------------|--|
| Pseudo-código | <pre>// vetor A(1..N) A[i] ← máx{ A[j]   1 ≤ j ≤ N }</pre>   |
| JavaScript    | <pre>var max = A[0]; for (var j = 1; j &lt; N; j++) {     if (A[j] &gt; max) {         max = A[j];     } } A[i] = max;</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <code>// sortear uniformemente um número entre min e max</code><br><code>a ← Sortear(min, max)</code> |
| <b>JavaScript</b>    | <code>a = Math.floor((Math.random() * max) + min);</code>   |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <pre>// medir o tempo decorrido para executar uma operação t ← ObterDataHora() // fazer alguma coisa escrever(ObterDataHora() - t)</pre> |
| <b>JavaScript</b>    | <pre>var t = (new Date()).getTime();  // fazer alguma coisa  alert(+((new Date()).getTime() - t) + "ms");</pre>                          |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <code>// erro durante processamento</code><br><code>Exceção("Mensagem")</code> |
| <b>JavaScript</b>    | <code>throw new Error("Mensagem");</code>                                      |



# **Linguagem Java**

# Comandos de pseudo-códigos utilizados (e sua transformação)

## O programa:

| Pseudo-código | programa CalculaAlgo()<br><código do programa>   |
|---------------|--|
| Java          | // Arquivo CalculaAlgo.java<br>public class CalculaAlgo {<br>public static void main(String[] args)<br>{<br><código do programa><br>}<br>} |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## **Executando o programa:**

Depois de compilado o programa (digamos, `CalculaAlgo.class`), a partir do diretório onde o compilador gerou o `.class`, executar:

```
> java CalculaAlgo
```

ou

```
> java CalculaAlgo <entrada.txt >saida.txt
```

(entradas do teclado são obtidas do arquivo `entrada.txt`, e saídas para a tela escritas em `saida.txt`)

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Comentários:

| Pseudo-código | <code>// este é um comentário</code>                                 |
|---------------|--|
| Java          | <pre>// este é // um comentário  /* este é outro comentário */</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis escalares:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>var A: Inteiro, B: Real, C: Lógico, D: Caractere<br/>A ← 10<br/>B ← 20.2<br/>C ← V<br/>D ← 'A'</pre> |
| <b>JavaScript</b>    | <pre>int A; double B; boolean C; char D;<br/>A = 10;<br/>B = 20.2;<br/>C = true;<br/>D = 'A';</pre>       |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores Aritméticos Comuns:

| Operador         | Pseudo-código                            | Java                           |
|------------------|--|--------------------------------|
| Soma             | $a + b$                                  | <code>a + b</code>             |
| Subtração        | $a - b$                                  | <code>a - b</code>             |
| Multiplicação    | $a * b$                                  | <code>a * b</code>             |
| Divisão          | $a / b$                                  | <code>a / b</code>             |
| Divisão Inteira  | $a \div b$<br>(ou <code>a div b</code> ) | <code>Math.floor(a / b)</code> |
| Resto da Divisão | $a \bmod b$                              | <code>a % b</code>             |
| Exponenciação    | $a ^ b$                                  | <code>Math.pow(a, b)</code>    |
| Raiz quadrada    | $\sqrt{a}$<br>(ou <code>sqrt(a)</code> ) | <code>Math.sqrt(a)</code>      |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores em Cadeias:

| Operador                              | Pseudo-código                                    | Java                                  |
|---------------------------------------|--|---------------------------------------|
| Declaração                            | var S: Cadeia<br>ou<br>var S(1..N):<br>Caractere | String S;                             |
| Comprimento                           | S  | S.length()                            |
| Caractere                             | S(5)   | S.charAt(4)                           |
| Subcadeia                             | S(5..9)  | S.substring(5, 10)                    |
| Encontrar Subcadeia                   | encontrar(S, SEnc,<br>aPartirDePosicao)          | S.indexOf(SEnc, aPartirDePosicao - 1) |
| Concatenação                          | S1 + S2  | S1 + S2                               |
| Criar cadeia de outro<br>tipo de dado | S ← para_cadeia(N)                               | S = Integer.toString(N)               |
| Criar número de cadeia                | N ← de_cadeia(S)                                 | N = Integer.valueOf(S)                |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Atribuição múltipla:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <code>var A: Inteiro, B: Real, C: Lógico, D: Caractere<br/>A, B, C, D ← 10, 20.2, V, 'A'</code>       |
| <b>Java</b>          | <code>int A; double B; boolean C; char D;<br/>A = 10;<br/>B = 20.2;<br/>C = true;<br/>D = 'A';</code> |



# Comandos de pseudo-códigos utilizados (e sua transformação)

## Atribuição múltipla:

| Pseudo-código | $X, Y \leftarrow Y, X$  |
|---------------|---|
| <b>Java</b>   | <pre>int Temp1 = Y; //supondo X,Y inteiros int Temp2 = X; X = Temp1; Y = Temp2;</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis vetores:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <code>var A[1..100]: Inteiro<br/>A[1..100] ← 0</code>   |
| <b>Java</b>          | <pre>int A[] = new int[100];<br/>for (int i = 0; i &lt; 100; i++) {<br/>    A[i] = 0;<br/>}</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis matrizes:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <code>var A[1..N, 1..N]: Inteiro<br/>A[i, j] ← 0, para todo <math>1 \leq i, j \leq N</math></code>   |
| <b>Java</b>          | <pre>int A[][] = new int[N][N];<br/>for (int i = 0; i &lt; N; i++) {<br/>    for (int j = 0; j &lt; N; j++) {<br/>        A[i][j] = 0;<br/>    }<br/>}</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis ponteiros:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>var N, M: ^Inteiro, i: Inteiro alocar(N) N^ ← 10 M ← @i M^ ← 11 desalocar(N)</pre>   |
| <b>Java</b>          | <pre>class Ponteiro {     public Object A; //A representa o valor apontado     public Ponteiro() {}; } Ponteiro N = new Ponteiro(); Ponteiro M = new Ponteiro(); Ponteiro i = new Ponteiro(); N.A = 10; M.A = i; ((Ponteiro) M.A).A = 11; N = null; //Garbage Collector devolverá a memória</pre> |

Um ponteiro em Java é sempre uma referência a um objeto. Portanto, todo dado referenciado por um ponteiro deve ser transformado em um objeto.

# Comandos de pseudo-códigos utilizados (e sua transformação)

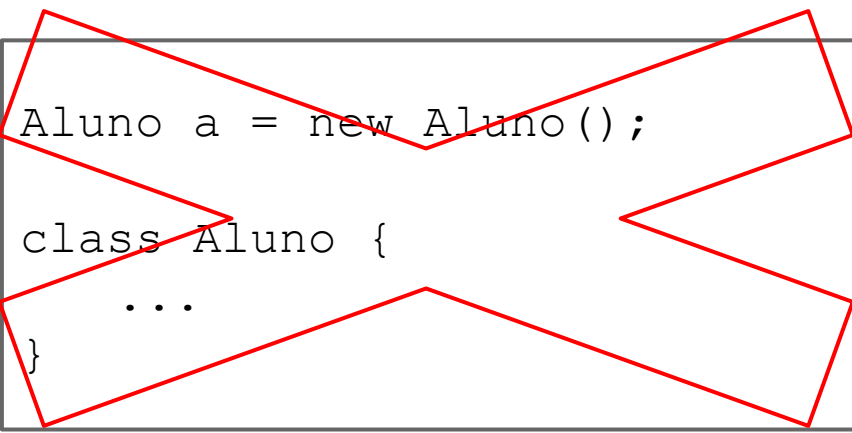
## Declaração/Atribuição de variáveis estruturas:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <pre>estrutura Aluno:     Notas[1..N], Matricula: Inteiro     Ouvinte: Lógico     Prox: ^Aluno  var a1: Aluno, a2: ^Aluno a1.Ouvinte ← F alocar(a2) a1.Prox ← a2</pre>   |
| <b>Java</b>          | <pre>class Aluno {     double Notas[];     int Matricula;     boolean Ouvinte;     Aluno Prox;     public Aluno(int N) {         this.Notas = new double[N];         this.Matricula = -1;         this.Ouvinte = false;         this.Prox = null;     } }</pre> <pre>Aluno a1 = new Aluno(N); Aluno a2 = new Aluno(N); // não há suporte em Java para a1 ser o objeto em si, apenas uma referência a ele a1.Ouvinte = false; a1.Prox = a2;</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

- Declarações de estruturas devem vir antes do seu uso!

```
class Aluno {  
    ...  
}  
  
Aluno a = new Aluno();
```



```
Aluno a = new Aluno();  
  
class Aluno {  
    ...  
}
```

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Escrita de variáveis:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <code>escrever ("O maior valor é ", valormax)</code>                                  |
| <b>Java</b>          | <code>System.out.println("O maior valor é " +<br/>Integer.toString(valormax));</code> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Escrita de variáveis:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>var C[1..N]: Inteiro ... escrever (C[1..N])</pre>  |
| <b>Java</b>          | <pre>int C[] = new int[N]; ... for (int i = 0; i &lt; N; i++) {     System.out.println(C[i]); }</pre> |



# Comandos de pseudo-códigos utilizados (e sua transformação)

## Leitura de variáveis:

| Pseudo-código | ler(N, C[1..N])  |
|---------------|--|
| Java          | <pre>java.util.Scanner s = new java.util.Scanner(System.in); N = s.nextInt(); for (int i = 0; i &lt; N; i++) {     C[i] = s.nextInt(); }</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Condicional:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>se condição1 então     &lt;bloco-então&gt; senão se condição2 então     &lt;bloco-senão-se&gt; senão     &lt;bloco-senão&gt;</pre>   |
| <b>Java</b>          | <pre>if (condição1) {     &lt;bloco-então&gt; } else if (condição2) {     &lt;bloco-senão-se&gt; } else {     &lt;bloco-senão&gt; }</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores Lógicos Comuns:

| Operador       | Pseudo-código     | Java |
|----------------|-------------------|------|
| Igual          | =                 | ==   |
| Diferente      | ≠                 | !=   |
| Menor          | <                 | <    |
| Maior          | >                 | >    |
| Menor ou Igual | ≤                 | <=   |
| Maior ou Igual | ≥                 | >=   |
| E              | e. $\wedge$       | &&   |
| OU             | ou, $\vee$        |      |
| NÃO            | não, $!$ , $\neg$ | !    |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Repetição (para):

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <code>para <math>i \leftarrow 1</math> até <math>N</math> passo 2 faça</code><br><code>&lt;bloco-para&gt;</code>      |
| <b>Java</b>          | <code>for (int i = 1; i <math>\leq</math> N; i = i + 2) {</code><br><code>&lt;bloco-para&gt;</code><br><code>}</code> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Repetição (enquanto):

| Pseudo-código | enquanto condição faça<br><bloco-enquanto>                         |
|---------------|--|
| Java          | <pre>while (condição) {<br/>    &lt;bloco-enquanto&gt;<br/>}</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Procedimentos:

|               |  |
|---------------|--|
| Pseudo-código | <pre>procedimento Calcula(val p1: Inteiro, ref p2: Real)     &lt;bloco-procedimento&gt;  var valor: Real ← 10 Calcula(10, valor)</pre>   |
| Java          | <pre>static class Ponteiro {     public Object A;     public Ponteiro() {}; }  static void Calcula(int p1, Ponteiro p2ref) {     //trocar o uso de p2 em Calcula por p2ref.A     ... }  public static void main(String[] args) {     ...     Ponteiro valorref = new Ponteiro();     double valor = 10.0;     valorref.A = valor;     Calcula(10, valorref);     valor = (double)valorref.A;     ... }</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

- Em Java, todos os parâmetros são por valor. Caso um valor seja modificado dentro de um procedimento/função e a modificação deve ser refletida no procedimento/função que fez a chamada, devemos criar uma referência ao valor explicitamente antes da chamada ser feita, e passar tal referência ao procedimento/função, que pode usá-la para alterar o valor.

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Funções:

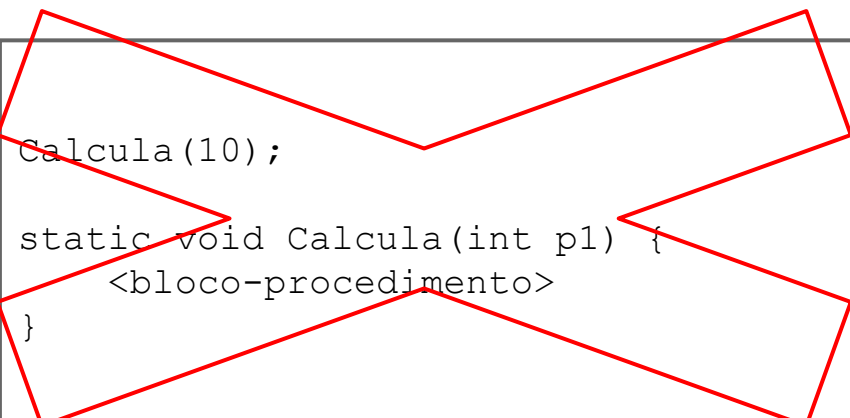
|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>função Calcula(val p1: Inteiro, ref p2: Real): Caractere     &lt;bloco-função&gt;     retornar (&lt;valor&gt;)  var valor: Real ← 10 escrever(Calcula(10, valor))</pre>  |
| <b>Java</b>          | <pre>static class Ponteiro {     public Object A;     public Ponteiro() {} }  static char Calcula(int p1, Ponteiro p2ref) {     //trocar o uso de p2 em Calcula por p2ref.A     ... }  public static void main(String[] args) {     ...     Ponteiro valorref = new Ponteiro();     double valor = 10.0;     valorref.A = valor;     System.out.println(Calcula(10, valorref));     valor = (double)valorref.A;     ... }</pre> |



# Comandos de pseudo-códigos utilizados (e sua transformação)

- Declarações de procedimentos e funções podem vir antes ou depois do seu uso!

```
static void Calcula(int p1) {  
    <bloco-procedimento>  
}  
  
Calcula(10);
```



```
Calcula(10);  
  
static void Calcula(int p1) {  
    <bloco-procedimento>  
}
```

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Vetores por Referência:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <pre>procedimento Carregar(ref A[]: Inteiro, N: Inteiro)     &lt;bloco-procedimento&gt;  var B[1..100]: Inteiro Carregar(B, 100)</pre> |
| <b>Java</b>          | <pre>static void Carregar(int[] A, int N) {     &lt;bloco-procedimento&gt; } int B[] = new int[100]; Carregar(B, 100);</pre>           |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Vetores por Valor:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>procedimento Carregar(val A[]: Inteiro, N: Inteiro)     &lt;bloco-procedimento&gt;  var B[1..100]: Inteiro ler (B[1..100]) Carregar(B, 100)</pre>  |
| <b>Java</b>          | <pre>static void Carregar(int[] Aorig, int N) {     int A[N];     for (int i=0; i &lt; N; i++) {         A[i] = Aorig[i];     }     &lt;bloco-procedimento&gt; //A será usado por Carregar } int B[] = new int[100]; java.util.Scanner s = new java.util.Scanner(System.in); for (int i = 0; i &lt; 100; i++) {     B[i] = s.nextInt(); } Carregar(B, N);</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operações agrupadas:

|               |  |
|---------------|--|
| Pseudo-código | <pre>// vetor A(1..N) A[i] ← máx{ A[j]   1 ≤ j ≤ N }</pre>   |
| Java          | <pre>int max = A[0]; for (int j = 1; j &lt; N; j++) {     if (A[j] &gt; max) {         max = A[j];     } } A[i] = max;</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <code>// sortear uniformemente um número entre min e max</code><br><code>a ← Sortear(min, max)</code> |
| <b>Java</b>          | <code>a = (int) Math.floor((Math.random() * max) + min);</code>                                       |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <pre>// medir o tempo decorrido para executar uma operação<br/>t ← ObterDataHora()<br/>// fazer alguma coisa<br/>escrever(ObterDataHora() - t)</pre>               |
| <b>Java</b>          | <pre>long t = System.currentTimeMillis();<br/><br/>// fazer alguma coisa<br/><br/>System.out.println(Long.toString(System.currentTimeMillis() - t) + " ms");</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <code>// erro durante processamento</code><br><code>Exceção("Mensagem")</code> |
| <b>Java</b>          | <code>System.out.println("Mensagem");</code><br><code>System.exit(-1);</code>  |

# **Linguagem TuPy**



# Comandos de pseudo-códigos utilizados (e sua transformação)

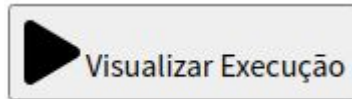
## O programa:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <code>programa CalculaAlgo()<br/>    &lt;código do programa&gt;</code>  |
| <b>TuPy</b>          | <code>// acesse <a href="https://tupy.herokuapp.com/">https://tupy.herokuapp.com/</a><br/>    &lt;código do programa&gt;</code> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

**Executando o programa:**

Clicar no botão



# Comandos de pseudo-códigos utilizados (e sua transformação)

## Comentários:

| Pseudo-código | <code>// este é um comentário</code>      |
|---------------|---|
| TuPy          | <code># este é<br/># um comentário</code> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis escalares:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>var A: Inteiro, B: Real, C: Lógico, D: Caractere<br/>A ← 10<br/>B ← 20.2<br/>C ← V<br/>D ← 'A'</pre>                   |
| <b>TuPy</b>          | <pre>inteiro A; real B; lógico C; Caractere D;<br/>A &lt;- 10<br/>B &lt;- 20.2<br/>C &lt;- verdadeiro<br/>D &lt;- 'A'</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores Aritméticos Comuns:

| Operador         | Pseudo-código                            | TuPy                 |
|------------------|--|----------------------|
| Soma             | $a + b$                                  | <code>a + b</code>   |
| Subtração        | $a - b$                                  | <code>a - b</code>   |
| Multiplicação    | $a * b$                                  | <code>a * b</code>   |
| Divisão          | $a / b$                                  | <code>a / b</code>   |
| Divisão Inteira  | $a \div b$<br>(ou $a \text{ div } b$ )   | <code>a div b</code> |
| Resto da Divisão | $a \bmod b$                              | <code>a mod b</code> |
| Exponenciação    | $a ^ b$                                  | <code>a ^ b</code>   |
| Raiz quadrada    | $\sqrt{a}$<br>(ou <code>sqrt(a)</code> ) | <code>raiz(a)</code> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores em Cadeias:

| Operador                              | Pseudo-código                                    | TuPy                            |
|---------------------------------------|--|---------------------------------|
| Declaração                            | var S: Cadeia<br>ou<br>var S[1..N]:<br>Caractere | cadeia S                        |
| Comprimento                           | S  | S                               |
| Caractere                             | S[5]   | S[4]                            |
| Subcadeia                             | S[5..9]  | S[4..8]                         |
| Encontrar Subcadeia                   | encontrar(S, SEnc,<br>aPartirDePosicao)          | --                              |
| Concatenação                          | S1 + S2  | S1 + S2                         |
| Criar cadeia de outro<br>tipo de dado | S ← para_cadeia(N)                               | S <- cadeia(N)                  |
| Criar número de<br>cadeia             | N ← de_cadeia(S)                                 | N <- inteiro(S)<br>N <- real(S) |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Atribuição múltipla:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <code>var A: Inteiro, B: Real, C: Lógico, D: Caractere<br/>A, B, C, D ← 10, 20.2, V, 'A'</code>      |
| <b>TuPy</b>          | <code>inteiro A; real B; lógico C; Caractere D<br/>A, B, C, D &lt;- 10, 20.2, verdadeiro, 'A'</code> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Atribuição múltipla:

|               |                        |
|---------------|------------------------|
| Pseudo-código | $X, Y \leftarrow Y, X$ |
| TuPy          | $X, Y <- Y, X$         |



# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis vetores:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <code>var A[1..100]: Inteiro</code><br><code>A[1..100] ← 0</code> |
| <b>TuPy</b>          | <code>inteiro A[100]</code><br><code>A[0..99] &lt;- 0</code>      |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis matrizes:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <code>var A[1..N, 1..N]: Inteiro</code><br><code>A[i, j] ← 0, para todo <math>1 \leq i, j \leq N</math></code> |
| <b>TuPy</b>          | <code>inteiro A[N,N];</code><br><code>A[*,*] &lt;- 0</code>  |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Declaração/Atribuição de variáveis ponteiros:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <pre>var N, M: ^Inteiro, i: Inteiro alocar(N) N^ ← 10 M ← @i M^ ← 11 desalocar(N)</pre>  |
| <b>TuPy</b>          | <p>Não há uma conversão direta. Um ponteiro em TuPy é sempre uma referência a um objeto. Portanto, todo dado referenciado por um ponteiro não pode ser um tipo escalar, devendo ser transformado em um objeto.</p> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

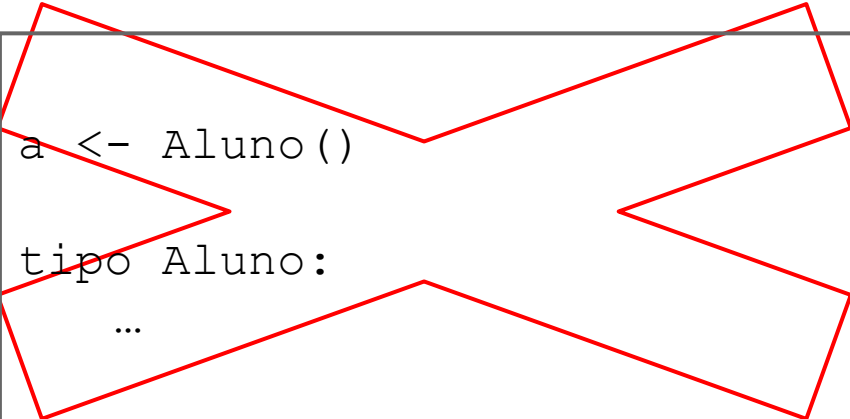
## Declaração/Atribuição de variáveis estruturas:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <pre>estrutura Aluno:     Notas[1..N], Matricula: Inteiro     Ouvinte: Lógico     Prox: ^Aluno  var a1: Aluno, a2: ^Aluno a1.Ouvinte ← F alocar(a2) a1.Prox ← a2</pre>   |
| <b>TuPy</b>          | <pre>inteiro N &lt;- 1000 tipo Aluno:     inteiro Notas[N]     lógico Ouvinte     Aluno Prox Aluno a1, a2; a1,a2 &lt;- Aluno(), Aluno() # não há suporte em TyPy para a1 ser o objeto em si, # apenas uma referência a ele a1.Ouvinte &lt;- falso a1.Prox &lt;- a2</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

- Declarações de tipos devem vir antes do seu uso!

```
tipo Aluno:  
  ...  
a <- Aluno()
```



```
a <- Aluno()  
tipo Aluno:  
  ...
```

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Escrita de variáveis:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <code>escrever ("O maior valor é ", valormax)</code> |
| <b>TuPy</b>          | <code>escrever ("O maior valor é", valormax)</code>  |




# Comandos de pseudo-códigos utilizados (e sua transformação)

## Escrita de variáveis:

|               |  |
|---------------|--|
| Pseudo-código | <pre>var C[1..N]: Inteiro ... escrever (C[1..N])</pre> |
| TuPy          | <pre>inteiro C[N] ... escrever (C)</pre>               |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Leitura de variáveis:

| Pseudo-código | ler(N, C[1..N])   |
|---------------|---|
| <b>TuPy</b>   | <pre>ler(N) para i &lt;- 0 até N:     ler(C[i])</pre> <p>Os valores de entrada devem ser previamente carregados na caixa de texto:</p> <div><div>3 1 2 3</div><div></div></div> <div><div> Visualizar Execução</div><div> Salvar programa</div><div> Carregar programa</div></div> |



# Comandos de pseudo-códigos utilizados (e sua transformação)

## Condicional:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>se condição1 então     &lt;bloco-então&gt; senão se condição2 então     &lt;bloco-senão-se&gt; senão     &lt;bloco-senão&gt;</pre> |
| <b>TuPy</b>          | <pre>se condição1:     &lt;bloco-então&gt; senão se condição2:     &lt;bloco-senão-se&gt; senão:     &lt;bloco-senão&gt;</pre>          |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operadores Lógicos Comuns:

| Operador       | Pseudo-código      | TuPy |
|----------------|--------------------|------|
| Igual          | =                  | =    |
| Diferente      | ≠                  | !=   |
| Menor          | <                  | <    |
| Maior          | >                  | >    |
| Menor ou Igual | ≤                  | <=   |
| Maior ou Igual | ≥                  | >=   |
| E              | E ou $\wedge$      | e    |
| OU             | OU ou $\vee$       | ou   |
| NÃO            | NÃO ou ! ou $\neg$ | não  |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Repetição (para):

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <code>para i ← 1 até N passo 2 faça<br/>  &lt;bloco-para&gt;</code>       |
| <b>TuPy</b>          | <code>para i &lt;- 1 até N incl. passo 2:<br/>  &lt;bloco-para&gt;</code> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Repetição (enquanto):

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <code>enquanto condição faça</code><br><code>&lt;bloco-enquanto&gt;</code> |
| <b>TuPy</b>          | <code>enquanto condição:</code><br><code>&lt;bloco-enquanto&gt;</code>     |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Procedimentos:

|               |   |
|---------------|---|
| Pseudo-código | <pre>procedimento Calcula(val p1: Inteiro, ref p2: Real) //Supõe: Condição 1 //Garante: Condição 2     &lt;bloco-procedimento&gt;  var valor: Real valor ← 10.0 Calcula(10, valor) escrever (valor)</pre> |
| Python        | <pre>Calcula(val inteiro p1, ref real p2): #Supõe: Condição 1 #Garante: Condição 2     &lt;bloco-procedimento&gt;  real valor valor &lt;- 10.0 Calcula(10, valor) escrever (valor)</pre>                  |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Funções:

|               |  |
|---------------|--|
| Pseudo-código | <pre>função Calcula(val p1: Inteiro, ref p2: Real): Caractere //Supõe: Condição 1 //Garante: Condição 2     &lt;bloco-função&gt;     retornar (&lt;valor&gt;)  var valor: Real ← 10 escrever(Calcula(10, valor))</pre> |
| TuPy          | <pre>character Calcula(val inteiro p1, ref real p2): #Supõe: Condição 1 #Garante: Condição 2     &lt;bloco-função&gt;     retornar &lt;valor&gt;  real valor valor &lt;- 10.0 escrever (Calcula(10, valor))</pre>      |

# Comandos de pseudo-códigos utilizados (e sua transformação)

- Declarações de procedimentos e funções podem vir antes ou depois do seu uso!

```
inteiro Calcula(inteiro v):  
    ...  
Calcula(10)
```

```
Calcula(10)  
  
inteiro Calcula(inteiro v):  
    ...
```

# Comandos de pseudo-códigos utilizados (e sua transformação)

- Note os comentários `//Supõe` e `//Garante` dos Procedimentos e Funções; apesar de não se transformarem em código, são importantes para especificar como usar a rotina apropriadamente
  - `//Supõe: Condição 1`

Condição 1 é uma expressão lógica que deve ser satisfeita pelo estado da computação quando o procedimento/função for invocado. Representa, portanto, o que o procedimento/função exige como pré-requisito para ser usado
  - `//Garante: Condição 2`

Condição 2 é uma expressão lógica que será satisfeita pelo estado da computação quando o procedimento ou função for finalizado. Representa, portanto, o serviço que o procedimento/função provê. No caso de funções, a palavra 'retorno' será usado como variável em Condição 2 representando o valor de retorno da função.



# Comandos de pseudo-códigos utilizados (e sua transformação)

## Vetores por Referência:

|                      |   |
|----------------------|---|
| <b>Pseudo-código</b> | <pre>procedimento Carregar(ref A[: Inteiro, N: Inteiro)     &lt;bloco-procedimento&gt;  var B[1..100]: Inteiro Carregar(B, 100)</pre> |
| <b>TuPy</b>          | <pre>Carregar(ref inteiro[] A, inteiro N):     &lt;bloco-procedimento&gt;  inteiro B[100] Carregar(B, 100)</pre>                      |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Vetores por Valor:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <pre>procedimento Carregar(val A[]: Inteiro, N: Inteiro)     &lt;bloco-procedimento&gt;  var B[1..100]: Inteiro ... Carregar(B, 100)</pre> |
| <b>TuPy</b>          | <pre>Carregar(val inteiro[] A, inteiro N):     &lt;bloco-procedimento&gt;  inteiro B[100] ... Carregar(B, 100)</pre>                       |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Operações agrupadas:

|               |   |
|---------------|---|
| Pseudo-código | <pre>// vetor A[1..N] A[i] ← máx{ A[j]   1 ≤ j ≤ N }</pre>  |
| TuPy          | <pre>inteiro inf &lt;- 100000000000 inteiro m &lt;- -inf; para j &lt;- 0 até N:     m &lt;- máx(A[j], m) A[i] &lt;- m</pre> |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

|               |   |
|---------------|---|
| Pseudo-código | <pre>// sortear uniformemente um número entre min e max<br/>a ← Sortear(min, max)</pre> |
| TuPy          | <pre>a &lt;- inteiro_aleatório(min, max)</pre>  |

# Comandos de pseudo-códigos utilizados (e sua transformação)

## Outras operações úteis:

|                      |  |
|----------------------|--|
| <b>Pseudo-código</b> | <code>// erro durante processamento</code><br><code>Exceção("Mensagem")</code> |
| <b>TuPy</b>          | <code>escrever ("Mensagem")</code><br><code>parar</code>                       |