

LÓGICA DE PROGRAMAÇÃO

Lucy Mari Tabuti e Rodrigo Assirati Dias

CONCEITOS BÁSICOS

Conceitos básicos de Lógica, Processamento de Dados, Algoritmo, Programa e Linguagem de Programação.

Lógica

Definição de lógica

Segundo o dicionário Aurélio¹, Lógica é:

"Inform. Forma pela qual as assertivas, pressupostos e instruções são organizadas em um algoritmo para implementação de um programa de computador".

Lógica de programação

A lógica de programação nada mais é do que a lógica usada na programação de computadores, que segue as leis do pensamento e os processos do raciocínio, cujo objetivo é desenvolver técnicas que ajudem as pessoas a realizar uma tarefa com o mínimo de esforço.

Processamento de Dados

Na informática, o processamento de dados refere-se à entrada de dados, processamento desses dados de entrada e a saída dos resultados deste processamento, numa sequência como mostra a Figura 1.

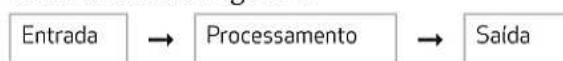


Figura 1 Processamento de dados. Fonte: Autor.

A entrada de dados, que pode ser uma coleta de dados, uma depuração de dados ou um armazenamento de dados, recebe a informação necessária que será tratada no processamento de dados. Essa entrada pode ser feita através de dispositivos de entrada, por exemplo, teclado, mouse, etc.

O processamento de dados, que pode ser aritmético ou lógico, trata a informação re-

cebida na entrada de dados, gerando resultados. Esse processamento é efetuado dentro do computador, na unidade lógica e aritmética conhecida como ULA – Unidade Lógica e Aritmética que fica dentro da CPU – Unidade Central de Processamento, o cérebro do computador.

A saída de dados, que pode ser uma coleta de resultados ou uma distribuição de resultados, é a entrega dos dados processados para o usuário. Essa saída pode ser feita através de dispositivos de saída, por exemplo, monitor, impressora, etc.

Algoritmo

Definição

Pode-se definir algoritmo como um processo que reúne um conjunto de ações que são necessárias para tratar os dados de entrada e transformá-los em resultados para um determinado objetivo.

Neste sentido, pode-se considerar que uma receita de bolo é um algoritmo, pois a partir dos ingredientes (dados de entrada), realizam-se várias ações como bater os ingredientes numa batedeira, colocar numa assadeira e depois assar por determinado tempo (processamento) para, no final, ter um bolo bem feito (resultado de saída).

Todo para o desenvolvimento de um algoritmo

Para a construção de um bom algoritmo, devem-se seguir os seguintes passos:

- ◊ ler e compreender o problema para o qual será construído um algoritmo;
- ◊ determinar qual será a entrada de dados do seu algoritmo;
- ◊ determinar quais as ações, lógicas e/ou aritméticas, que deverão ser realizadas no seu algoritmo, bem como, as restrições, se houver, para cada uma dessas ações;
- ◊ determinar qual será a saída de resultados do seu algoritmo;
- ◊ construir o algoritmo em um dos tipos de algoritmo descritos na próxima seção;

¹ Ferreira, Aurélio Buarque de Holanda. Novo Aurélio século XXI: O Dicionário da Língua Portuguesa. 3a edição. Nova Fronteira. Rio de Janeiro. 1999.

- ◊ testar o algoritmo usando o teste de mesa descrito no final deste capítulo.

Tipos de algoritmo

As formas mais importantes para a construção de algoritmos são: a descrição narrativa, o fluxograma e o pseudocódigo (ou português estruturado ou portugol).

- ◊ **Descrição narrativa.** A descrição narrativa nada mais é do que descrever, utilizando

uma linguagem natural (no nosso caso, a língua portuguesa), as ações a serem realizadas no tratamento dos dados de entrada para os resultados de saída na resolução do problema proposto.

- ◊ **Fluxograma.** O fluxograma é a forma de descrever as ações a serem realizadas no tratamento dos dados de entrada para os resultados de saída usando uma representação simbólica preestabelecida, por exemplo, como os símbolos na Figura 2.

	Símbolo utilizado para determinar o início e o término do algoritmo
	Símbolo utilizado para determinar o sentido do fluxo de dados do algoritmo. Utilizado para conectar os símbolos existentes
	Símbolo utilizado para determinar uma ação que pode ser um cálculo ou uma atribuição de valores
	Símbolo utilizado para determinar a entrada de dados do algoritmo. Esta entrada de dados pode ser via dispositivos de entrada
	Símbolo utilizado para determinar a saída de dados do algoritmo. Esta saída de dados pode ser via monitor
	Símbolo utilizado para determinar uma decisão que indicará qual caminho será seguido no algoritmo
	Símbolo utilizado para determinar uma conexão entre partes de um mesmo algoritmo

Figura 2 Fluxograma. Fonte: Autor

- ◊ **Pseudocódigo.** O pseudocódigo é a forma de descrever as ações para a resolução de um problema proposto por meio de regras preestabelecidas.

Programa

Um programa de computador é a codificação de um algoritmo numa linguagem de programação específica, por exemplo, o C e o Java.

Sempre que se codifica um algoritmo numa linguagem de programação, este programa precisa ser “traduzido” para a linguagem entendida pela máquina. A este processo chama-se interpretação ou compilação.

Linguagem de Programação C

A função main

O ponto de entrada de qualquer programa escrito em C é a função main. Ela é obrigatória e é por ela que o sistema operacional sabe por onde começar a execução do programa cada vez que este for executado.

Linguagem de Programação Java

O método main

De forma muito semelhante o C, no código fonte de qualquer programa Java há uma classe principal. Dentro desta classe principal, deve-se ter o método main que é o mé-

todo principal de qualquer programa Java e sem este método o programa não funciona. Isso por que é pelo método *main* que o programa inicia sua execução.

A sintaxe do método *main* é a seguinte: `public static void main(String <nome> [])`. `public` indica que o método *main* pode ser chamado por outro objeto, `static` indica que o método *main* é um método de classe e `void` indica que o método *main* não retorna nenhum valor. O método *main* aceita um único argumento que é um vetor que guarda os valores digitados pelo usuário após o nome do programa.

Exemplo na Linguagem C

```
/* Primeiro programa em C
```

Este código fonte será salvo no arquivo `primeiro_programa.c`.

Este programa serve para exemplificar um programa C para os alunos de Algoritmos e Lógica de Programação

```
*/
```

```
#include<stdio.h>
```

```
int main()
{
```

`printf("Primeiro Programa de Lógica de Programação");`

```
    return 0;
```

```
} // fim da função main
```

Exemplo na Linguagem Java

```
/* Primeiro programa em Java
```

Este código fonte será salvo no arquivo `primeiro_programa.java`

Este programa serve para exemplificar um programa Java para os alunos de Algoritmos e Lógica de Programação

```
*/
```

```
public class primeiro_programa
```

```
{
```

```
    public static void main(String argu-
```

```
mentos[])
{
```

```
        System.out.print("Primeiro Progra-
ma de Lógica de Programação");
        System.exit(0);
    } // fim do método main
} // fim da classe principal primeiro_pro-
grama
```

COMO CONSTRUIR UM ALGORITMO

Descrição narrativa de como construir algoritmos.

Exemplo de como fazer um café

Nesta seção, serão apresentados vários algoritmos de como fazer um café utilizando o algoritmo de descrição narrativa. Observe como fica um algoritmo sequencial para nosso exemplo:

Descrição Narrativa Sequencial

- ◊ Pegar bule.
- ◊ Colocar coador de plástico sobre o bule.
- ◊ Colocar coador de papel sobre o coador de plástico.
- ◊ Colocar café tostado e moído sobre o coador de papel.
- ◊ Colocar água sobre o café.

Descrição Narrativa de Seleção

- ◊ Pegar bule.
- ◊ Colocar coador de plástico sobre o bule.
- ◊ Colocar coador de papel sobre o coador de plástico.
- ◊ Colocar café tostado e moído sobre o coador de papel.
- ◊ se a água estiver fervente, então:
 - » Colocar água sobre o café.

Descrição Narrativa de Repetição sem a estrutura de repetição

- ◊ Pegar bule.
- ◊ Colocar coador de plástico sobre o bule.
- ◊ Colocar coador de papel sobre o coador de plástico.

- ◊ Colocar café tostado e moído sobre o coador de papel.
- ◊ se a água não estiver fervente, então
 - » Aquecer a água.
 - » se a água não estiver fervente, então
 - Continuar aquecendo a água.
 - se a água não estiver fervente, então
 - Continuar aquecendo a água
 - ...
 - continua até quando?
 - colocar água

Descrição Narrativa de Repetição usando a estrutura de repetição

- ◊ Pegar bule.
- ◊ Colocar coador de plástico sobre o bule.
- ◊ Colocar coador de papel sobre o coador de plástico.
- ◊ Colocar café tostado e moído sobre o coador de papel.
- ◊ enquanto a água não estiver fervente, faça
 - » Aquecer a água.
 - » Colocar água sobre o café.

Outro exemplo de construção de algoritmo envolvendo lógica

Problema:

Considere a Torre de Hanói com três hastes e três discos. Uma das hastes serve de suporte para três discos de tamanhos diferentes de forma que os menores estão sobre os maiores. Desenvolva um algoritmo que mova todos os discos da haste onde se encontram para uma das outras duas hastes da Torre de Hanói, seguindo as seguintes regras: pode-se mover um disco de cada vez para qualquer haste, contanto que um disco maior nunca seja colocado sobre um disco menor.

Resolução:

Considerando discos 1, 2, 3 e hastes A, B, C, tem-se o resultado descrito.

- ◊ Mover o disco 1 da haste A para a haste B.
- ◊ Mover o disco 2 da haste A para a haste C.

- ◊ Mover o disco 1 da haste B para a haste C.
- ◊ Mover o disco 3 da haste A para a haste B.
- ◊ Mover o disco 1 da haste C para a haste A.
- ◊ Mover o disco 2 da haste C para a haste B.
- ◊ Mover o disco 1 da haste A para a haste B.

CONSTANTES, VARIÁVEIS E TIPOS DE DADOS NOS ALGORITMOS

Estudo das constantes, variáveis e tipos de dados nos algoritmos.

Tipos de dados

Os tipos de dados são as características comuns dos dados a serem manipulados. Podem-se considerar cinco classificações para os tipos de dados: inteiro, real, caracter, alfanumérico e lógico.

O tipo inteiro caracteriza qualquer dado numérico que pertença ao conjunto dos números inteiros. Por exemplo: -5, 0, 32.

O tipo real caracteriza qualquer dado numérico que pertença ao conjunto dos números reais. Por exemplo: -9.0, 0, 29.45.

O tipo caracter caracteriza um único qualquer dado que pertença a um conjunto de caracteres alfanuméricos e são simbolizados por entre aspas simples (' '). Por exemplo: 't', 'E', '%'.

O tipo alfanumérico caracteriza qualquer conjunto de dados que pertençam a um conjunto de caracteres alfanuméricos e são simbolizados por entre aspas duplas (""). Por exemplo: "15", "Eu", "Pare!", "?%@".

O tipo lógico caracteriza qualquer dado que possa assumir somente duas situações: verdadeiro ou falso. Por exemplo: feminino ou masculino, loja aberta ou fechada, brasileiro ou estrangeiro.

Pseudocódigo

- ◊ Para o tipo inteiro usa-se inteiro.
- ◊ Para o tipo real usa-se real.
- ◊ Para o tipo caracter usa-se caracter.
- ◊ Para o tipo alfanumérico usa-se alfanumérico.
- ◊ Para o tipo lógico usa-se lógico.

Constantes

Definição

Diz-se que determinada informação é uma constante quando esta não sofre nenhuma alteração, ou seja, ela é fixa. Por exemplo, pode-se definir uma constante $\Pi \leftarrow 3.14$ dentro do algoritmo e neste algoritmo este valor nunca se alterará.

Pseudocódigo

A declaração de um dado constante em pseudocódigo terá a seguinte regra sintática:

declarar

 constante <nome da constante> \leftarrow <valor da constante> <tipo de dado da constante>;

Por exemplo:

declarar

 constante pi \leftarrow 3,14 real;

Pode-se também declarar vários dados constantes em pseudocódigo numa mesma linha se eles forem todos do mesmo tipo, seguindo a seguinte regra sintática, para NC \leftarrow nome da constante e VC \leftarrow valor da constante.

declarar

 constante <NC> \leftarrow <VC> , <NC> \leftarrow <VC> , ... , <NC> \leftarrow <VC> <tipo de dado da constante>;

Por exemplo:

declarar

 constante pi \leftarrow 3,14, x \leftarrow 9,3, telefone \leftarrow -7,23 real;

Variáveis

Definição

Diz-se que determinada informação é uma variável quando esta não pode sofrer alguma alteração, ou seja, ela é variável. Por exemplo, pode-se definir uma variável x dentro do

algoritmo e neste algoritmo este valor poderá se alterar.

Pseudocódigo

A declaração de uma informação variável em pseudocódigo terá a seguinte regra sintática:

declarar

 <nome da variável> <tipo de dado da variável>;

Por exemplo:

declarar

 x inteiro;

Pode-se também declarar várias informações variáveis em pseudocódigo numa mesma linha se eles forem todos do mesmo tipo, seguindo a seguinte regra sintática, para NC \leftarrow nome da constante:

declarar

 <NV> , <NV> , ... , <NV> <tipo de dado da variável>;

Por exemplo:

declarar

 telefone, idade, valor, cor alfanumérico;

No momento da declaração de um dado variável em pseudocódigo, pode-se também inicializar a variável, assim a declaração com a inicialização terá a seguinte regra sintática:

declarar

 <nome da variável> \leftarrow <valor de inicialização da variável> <tipo de dado da variável>;

Por exemplo:

declarar

 y \leftarrow 5 real;

Identificadores

Definição

Os identificadores são os nomes dados às informações de um algoritmo ou programa, por exemplo, nome da variável, nome da constante, nome do programa, etc.

Por exemplo: pi, x, num_1, valor5, etc.

Os identificadores são únicos para uma determinada informação, por exemplo, um mesmo identificador não pode ser usado para o nome de uma variável e de uma constante, ou para o nome de duas variáveis de mesmo tipo ou de tipos diferentes. Uma vez que um identificador foi usado para uma determinada informação, ele não pode ser usado novamente para identificar outra informação.

Por exemplo:

declarar

```
x numérico_inteiro;  
x numérico_real;
```

Este exemplo mostra o que não pode acontecer, ou seja, o identificador x declarado como um tipo inteiro e depois como um tipo real. Um identificador pode possuir somente um tipo de informação.

Formato

Esses identificadores devem seguir as seguintes regras para serem válidos:

- ◊ todos os caracteres devem ser letras, números ou o sublinhado (o único caractere especial permitido);
- ◊ o primeiro caractere deve ser uma letra;
- ◊ os demais caracteres podem ser letras, números ou sublinhado;
- ◊ não são permitidos caracteres especiais (com exceção do sublinhado) e o espaço em branco;
- ◊ palavras reservadas (de uma linguagem de programação ou do pseudocódigo) não devem ser usados como identificadores.

Exemplos com constantes, variáveis e identificadores em pseudocódigo

1. São identificadores válidos:

» num1, item5C, tipo_a2, A123, idade, telefone

2. São identificadores não válidos:

» inum, 5Citem, _tipoa2, _2valor, 123A, R\$, nota/2, 5*, x&y

3. Declaração de constantes:

declarar

```
constante num1 ← 9 numérico_inteiro;  
constante x_2 ← -3.28, cor ← 7.4 numérico_real;
```

4. Declaração de variáveis

declarar

```
um ← 5, dois numérico_inteiro;  
maria, vlr9 numérico_real;  
valor_4A ← “alfabeto” alfanumérico;  
sexo, X987 lógico;
```

CONSTANTES, VARIÁVEIS E TIPOS DE DADOS EM C E JAVA

Estudo das constantes, variáveis e tipos de dados nas linguagens de programação C e Java.

Tipos de dados em C

O tipo inteiro do pseudocódigo é int na linguagem C.

O tipo real do pseudocódigo é float ou double na linguagem C.

O tipo caracter do pseudocódigo é char na linguagem C.

O tipo caracter do pseudocódigo é char * na linguagem C.

O tipo alfanumérico do pseudocódigo é um conjunto de char na linguagem C.

O tipo lógico do pseudocódigo é boolean na linguagem C.

Tipos de dados em Java

O tipo inteiro do pseudocódigo é int na linguagem Java.

O tipo real do pseudocódigo é float ou double na linguagem Java.

O tipo caracter do pseudocódigo é char na linguagem Java.

O tipo alfanumérico do pseudocódigo é String na linguagem Java.

O tipo lógico do pseudocódigo é boolean na linguagem Java.

Constantes no C

A declaração de um dado constante em C, pode ser exemplificado por:

```
const double pi = 3.14;
```

Pode-se também declarar vários dados constantes em C numa mesma linha se eles forem todos do mesmo tipo.

Constantes no Java

A declaração de um dado constante em Java pode ser exemplificada por:

```
final double pi = 3.14;
```

Pode-se também declarar vários dados constantes em Java numa mesma linha se eles forem todos do mesmo tipo.

Variáveis no C

A declaração de um dado variável em C pode ser exemplificada por:

```
int x;
```

Pode-se também declarar vários dados variáveis em C numa mesma linha se eles forem todos do mesmo tipo.

Por exemplo:

```
char letra, tom, nome;
```

No momento da declaração de um dado variável em C, pode-se também inicializar a variável.

Por exemplo:

```
double y = 5;
```

Variáveis no Java

A declaração de um dado variável no Java também é muito semelhante ao C.

Por exemplo:

```
int x;
```

Pode-se também declarar vários dados variáveis em Java numa mesma linha se eles forem todos do mesmo tipo.

Por exemplo:

```
char letra, tom, nome;
```

No momento da declaração de um dado variável em Java, pode-se também inicializar a variável.

Por exemplo:

```
double y = 5;
```

EXPRESSÕES ARITMÉTICAS E LÓGICAS NOS ALGORITMOS

Estudo dos operadores aritméticos e lógicos, bem como, das expressões aritméticas e lógicas nos algoritmos.

Uma expressão aritmética é o conjunto de operadores e operandos dispostos numa determinada ordem. Neste caso, os operadores podem ser aritméticos e os operandos podem ser constantes ou variáveis inteiros ou reais. O resultado de uma expressão aritmética sempre será numérico.

Uma expressão lógica também é um conjunto de operadores e operandos dispostos numa determinada ordem. Neste caso, os operadores podem ser relacionais ou lógicos e os operandos podem ser relações, constantes ou variáveis inteiros, reais, alfanuméricas ou lógicas. O resultado de uma expressão lógica sempre será lógico, ou seja, verdadeiro ou falso.

Operadores aritméticos

Os operadores aritméticos pertencem a um conjunto de símbolos que representam as operações matemáticas básicas, que são:

Tabela 1 Operadores aritméticos básicos

Operador	Função	Exemplo
+	adição	$5 + 3$
-	subtração	$2 - a$
*	multiplicação	$b * c$
/	divisão	$d / 10$

Prioridades

Para resolver expressões aritméticas em pseudocódigo, deve-se obedecer a uma hierarquia de prioridades entre operadores aritméticos que é a seguinte:

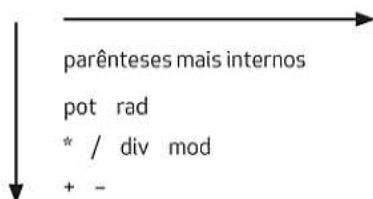


Figura 3 Prioridades entre operadores aritméticos.

Tabela 2 Operadores aritméticos auxiliares

Operador	Função	Exemplo	Resultado
pot	potenciação	pot(2,3)	$2^3 = 8$
rad	radiciação	rad(9)	$\sqrt{9} = 3$
mod	resto da divisão	7 mod 3	resto = 1
div	quociente da divisão	7 div 3	quociente = 2

Operadores relacionais

Os operadores relacionais pertencem a um conjunto de símbolos que representam as comparações possíveis entre dois valores de mesmo tipo de dados. Esses valores podem ser constantes, variáveis ou expressões aritméticas. Os operadores relacionais podem ser:

Tabela 3 Operadores relacionais

Operador	Função	Exemplo	Resultado
>	maior que	$3 > 2$	Verdadeiro
<	menor que	$3 < 2$	Falso
\geq	maior ou igual a	$5 - 2 \geq 1 + 2$	Verdadeiro
\leq	menor ou igual a	$5 - 2 \leq 1 + 2$	Verdadeiro
=	igual a	$10 = 10$	Verdadeiro
\neq	diferente de	$10 \neq 10$	Falso

Prioridades

Para resolver expressões lógicas em pseudocódigo, não há uma hierarquia de prioridades entre operadores relacionais que deve-se obedecer.

Operadores lógicos

Os operadores lógicos pertencem a um conjunto de símbolos que representam os conectivos básicos. Esses conectivos formam novas proposições compostas a partir de outras proposições simples. Os operadores lógicos podem ser:

Tabela 4 Operadores lógicos

Operador	Função
não	negação
e	conjunção
ou	disjunção

Tabela Verdade

A Tabela Verdade determina todos os resultados possíveis de uma combinação entre valores de variáveis lógicas. Esses resultados possuem somente dois valores: verdadeiro ou falso. Para exemplificar o uso dos operadores lógicos, constrói-se uma tabela verdade, que se segue:

Tabela 5 Tabela Verdade

A	B	não A	A e B	A ou B
V	V	F	V	V
V	F	F	F	V
F	V	V	F	V
F	F	V	F	F

Prioridades

Para resolver expressões lógicas em pseudocódigo, deve-se obedecer a uma hierarquia de prioridades entre operadores lógicos que é a seguinte:

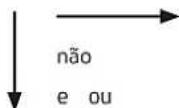


Figura 4 Prioridades entre operadores lógicos.

Prioridades entre Operadores

Para resolver expressões que contenham expressões lógicas e aritméticas, com operandos, operadores aritméticos, lógicas e relacionais existe uma prioridade entre os operadores que é a seguinte:

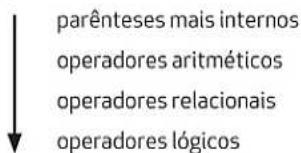


Figura 5 Prioridades entre operadores lógicos.

Exemplo com operadores

1. Expressões aritméticas

a) $5 * 2 - 3 + 14 / 2 + 9$
 $10 - 3 + 7 + 9$
 9

b) $5 - \text{pot}(2,3) + 4 - 2 * \text{rad}(4)$
 $5 - 8 + 4 - 2 * 2$
 $5 - 8 + 4 - 4$
 -3

2. Expressões lógicas com operadores relacionais e aritméticos

a) $5 * 2 = 4 + 10 / 2$
 $10 = 4 + 5$

$10 = 9$

F

b) $5 \bmod 2 + 3 < \text{pot}(3,2) * 10$

$1 + 3 < 9 * 10$

$4 < 90$

V

3. Expressões lógicas com operadores lógicos, relacionais e aritméticos

a) $3 < 7 \text{ e } 5 * 2 = 2 + 1$

$3 < 7 \text{ e } 10 = 3$

F e F

F

b) $\text{pot}(2,3) \leq \text{rad}(9) \text{ ou } 2 * 3 - 6 / 3 = 4$

$8 \leq 3 \text{ ou } 6 - 2 = 4$

F ou 4 = 4

F ou V

V

EXPRESSÕES ARITMÉTICAS E LÓGICAS EM C E JAVA

Estudo dos operadores aritméticos e lógicos, bem como, das expressões aritméticas e lógicas nas Linguagens de Programação C e Java.

Operadores aritméticos

Operadores aritméticos em C

Os operadores aritméticos básicos em C são os mesmos apresentados na Tabela 1. Além dos operadores aritméticos básicos, o C dispõe de operadores aritméticos auxiliares, como se segue:

Tabela 6 Operadores aritméticos auxiliares em C

Operador	Função	Exemplo
pow	potenciação	pow(2,3)
sqrt	radiciação	sqrt(9)
%	resto da divisão	7 % 3

Com exceção do operador que calcula o resto da divisão, as demais operações aritméticas estão organizadas em uma biblioteca matemática auxiliar. Essa biblioteca tem suas operações declaradas no arquivo math.h e portanto para utilizá-la, deve-se primeiro informar ao compilador do C que

seu programa as utiliza, para que ele possa fazer a ligação entre seu programa e essa biblioteca nos passos de compilação e ligação.

Para informar o compilador que utilizará a biblioteca auxiliar de operações matemáticas, deve-se utilizar o seguinte comando na primeira linha de seu programa:

```
#include <math.h>
```

Operadores aritméticos em Java

Assim como em C, os operadores aritméticos básicos em Java também são os mesmos apresentados na Figura 12. Os operadores aritméticos auxiliares seguem a simbologia própria do Java, como se segue:

Tabela 7 Operadores aritméticos auxiliares em Java

Operador	Função	Exemplo
Math.pow	potenciação	Math.pow(2,3)
Math.sqrt	radiciação	Math.sqrt(9)
%	resto da divisão	7 % 3

Operadores relacionais

Operadores relacionais em C e em Java

Os operadores relacionais em C e em Java têm uma pequena diferença dos operadores relacionais em pseudocódigo e são representados como se segue:

Tabela 8 Operadores relacionais em C e Java

Operador	Função	Exemplo	Resultado
>	maior que	1 > 8	Falso
<	menor que	1 < 8	Verdadeiro
>=	maior ou igual a	-2 >= 1 + 5	Falso
<=	menor ou igual a	-2 <= 7 - 2	Verdadeiro
==	igual a	10 == 9	Falso
!=	diferente de	10 != 9	Verdadeiro

Operadores lógicos em C e em Java

Os operadores lógicos em C e em Java são representados como se segue:

Tabela 9 Operadores lógicos em C e Java

Operador	Função
!	negação
&&	conjunção
	disjunção

Exemplo com operadores

1. Expressões aritméticas

a) $5 * 2 - 3 + 14 / 2 + 9$

$$10 - 3 + 7 + 9$$

9

b) $5 - \text{Math.pow}(2,3) + 4 - 2 * \text{Math.sqrt}(4)$

$$5 - 8 + 4 - 2 * 2$$

$$5 - 8 + 4 - 4$$

- 3

2. Expressões lógicas com operadores relacionais e aritméticos

a) $5 * 2 == 4 + 10 / 2$

$$10 == 4 + 5$$

$$10 == 9$$

F

b) $5 \% 2 + 3 < \text{Math.pow}(3,2) * 10$

$$1 + 3 < 9 * 10$$

$$4 < 90$$

V

3. Expressões lógicas com operadores lógicos, relacionais e aritméticos

a) $3 < 7 \&\& 5 * 2 == 2 + 1$

$$3 < 7 \&\& 10 = 3$$

$$F \&\& F$$

F

b) $\text{Math.pow}(2,3) <= \text{Math.sqrt}(9) \parallel 2 * 3$

$$- 6 / 3 == 4$$

$$8 <= 3 \parallel 6 - 2 == 4$$

$$F \parallel 4 == 4$$

$$F \parallel V$$

V

ESTRUTURA SEQUENCIAL NOS ALGORITMOS

Estudo da estrutura sequencial nos algoritmos.

Uma estrutura sequencial é aquela em que as ações de um algoritmo são executadas numa ordem sequencial, ou seja, de cima para baixo e da esquerda para a direita. Para o pseudocódigo adotado neste livro, convém-se que o uso do ponto e vírgula (;) determina o final de uma ação.

Operador de atribuição (\leftarrow) e comandos de atribuição

O comando de atribuição, que no pseudocódigo é representado pelo operador de atribuição \leftarrow , serve para atribuir um determinado valor para uma variável, tendo o cuidado de verificar se o valor que está sendo atribuído à variável tem o tipo de dado compatível, ou seja, se uma variável x foi declarada como inteiro, só é permitido atribuir valores inteiros à variável x .

Por exemplo, considere que as variáveis x , y e z foram declaradas como do tipo inteiro:

```
x ← 25;  
y ← x + 15 - 3;  
z ← y - x + rad(x) - pot(y,2);
```

Comandos de entrada e saída

Os comandos de entrada e saída são utilizados para que haja uma interação entre o algoritmo e o usuário do algoritmo. Neste caso, com o comando de entrada, é possível que o usuário forneça dados para serem processados pelo algoritmo e, com o comando de saída, é possível que o usuário veja o resultado do processamento dos dados.

Pseudocódigo – comandos de entrada

Para que o usuário possa entrar com dados num algoritmo, utiliza-se o comando de entrada. Por exemplo, considere que uma variável num foi declarada como do tipo real:

```
ler (num);
```

Pode-se também receber valores para várias variáveis com um mesmo comando de entrada. Por exemplo, considere que as variáveis $num1$, $num2$ e $num3$ foram declaradas como do tipo real:

```
ler (num1, num2, num3);
```

Pseudocódigo – comandos de saída

Para que o usuário possa ver o resultado do processamento dos dados ou alguma mensagem enviada pelo algoritmo, utiliza-se o comando de saída. Por exemplo, considere que uma variável num foi declarada como do tipo inteiro e o valor 5 lhe foi atribuído:

```
num ← 5;  
escrever (num);
```

Ao invés de mostrar para o usuário o conteúdo de uma variável, pode-se mostrar um dado alfanumérico. Por exemplo:

```
escrever ("Disciplina de Lógica de Programação");
```

Com o mesmo comando de saída, pode-se enviar uma mensagem cujo conteúdo seja a concatenação de vários dados alfanuméricos, sejam eles do tipo alfanuméricos ou da conversão de outros tipos de dados para o tipo alfanumérico. Por exemplo, considere que as variáveis $num1$ e $num2$ foram declaradas como do tipo inteiro e os valores 10 e 5 lhes foram, respectivamente, atribuídos:

```
num1 ← 10;  
num2 ← 5;  
escrever ("Maria tem ", num1, " anos e João tem ", num2, " anos");
```

Blocos

Em pseudocódigo, um bloco tem seu início com a palavra **início** e seu término com a palavra **fim**. Um algoritmo pode ser con-

siderado um bloco, no entanto, seu término é determinado com a palavra `fim_algoritmo`.

Por exemplo:

Algoritmo Exemplo

```
início_algoritmo // início do bloco ou algoritmo
Declarar
    // declaração das variáveis e/ou constantes
    // sequência dos comandos
fim_algoritmo. // fim do bloco ou algoritmo
```

Note que um algoritmo deve estar bem identado, isto é, deve estar numa formação que facilite o seu entendimento.

Estrutura sequencial

Uma estrutura sequencial é aquela em que as ações de um algoritmo são executadas numa ordem sequencial, ou seja, de cima para baixo e da esquerda para a direita. Para o pseudocódigo adotado neste livro, convenciona-se que o uso do ponto e vírgula (;) determina o final de uma ação.

Em pseudocódigo, a estrutura sequencial representa um conjunto de ações ou comandos que são executados num determinado fluxo.

Por exemplo:

Algoritmo Exemplo_Sequencial

```
início_algoritmo
Declarar
    x, y alfanumérico;
    ler(x, y);
    escrever("Você digitou primeiro o ", x);
    escrever("Você digitou logo em seguida o ", y);
fim_algoritmo.
```

Exemplos de estrutura sequencial em pseudocódigo

1. Desenvolva um algoritmo que receba, do usuário, dois números reais e exiba a soma destes dois números.

Algoritmo Soma

```
início_algoritmo
```

```
Declarar
```

```
    // declaração de variáveis
    n1, n2, soma           real;
```

```
// mensagem ao usuário
```

```
escrever ("Digite dois números reais");
```

```
// entrada de dados
```

```
ler (n1 , n2);
```

```
// processamento de dados
```

```
soma ← n1 + n2;
```

```
// saída de resultados
```

```
escrever ("A soma dos dois números reais digitados é: " , soma);
```

```
fim_algoritmo.
```

2. Desenvolva um algoritmo que receba, do usuário, dois números inteiros, calcule e exiba a média aritmética destes dois números.

Dica: a média aritmética é a soma dos dois números dividido por dois, por isso, o resultado deve ser um número real.

Algoritmo Media

```
início_algoritmo
```

```
Declarar
```

```
    // declaração de variáveis
    num1, num2           inteiro;
    media                 real;
```

```
// mensagem ao usuário
```

```
escrever ("Digite dois números inteiros");
```

```
// entrada de dados
```

```
ler (num1 , num2);
```

```
// processamento de dados
```

```
media ← (num1 + num2) / 2;
```

```
// saída de resultados
```

```
escrever ("A média dos dois números digitados é: " , media);
```

```
fim_algoritmo.
```

ESTRUTURA SEQUENCIAL EM C

Estudo da estrutura sequencial na Linguagem de Programação C.

Operador de atribuição (=) e comandos de atribuição em C

O comando de atribuição em C será o igual (=). Por exemplo, considere que as variáveis x, y e z foram declaradas como do tipo int:

```
x = 25;  
y = x + 15 - 3;  
z = y - x + sqrt(x) - pow(y,2);
```

Comandos de entrada e saída em C

Comando de entrada

Para que o usuário possa entrar com dados num programa C, utiliza-se a função scanf definida dentro da biblioteca stdio.h. Para isso, o programa deve fazer o include desta biblioteca no início do programa da seguinte forma:

```
#include <stdio.h>
```

A função scanf lê uma determinada sequência formatada de caracteres de entrada e a armazena em uma variável. A função scanf recebe como parâmetros então o formato da sequência que se quer receber e a variável onde se quer armazenar, retornando a quantidade de caracteres lidos e armazenados.

A variável precisa ser declarada previamente no programa e o formato deve corresponder ao tipo de dado que se quer ler e o tipo de dado da variável onde se quer armazenar. Os formatos mais comuns são os seguintes:

Tabela 10 Marcadores de formatação no C

Formato	Função	Exemplo
%d	Numérico inteiro	75

(continua)

Posição	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Caractere	A	l	g	o	r	i	t	m	o	\0						

A função scanf pode ainda ler mais do que um valor ao mesmo tempo. Por exemplo, considere que uma variável nome foi declarada como do tipo String e a variável idade como do tipo int.

%f	Numérico real em notação normal	75.100000
%1f	Numérico real com 1 dígito após o decimal	75.1
%s	Sequência de caracteres (string)	"algoritmos"
%c	Caractere	'A'

Por exemplo, considere que uma variável num foi declarada como do tipo inteiro:

```
printf("Digite um número: ");  
scanf("%d", &num);
```

Para ler um conjunto de caracteres, por exemplo, uma palavra, com o scanf, deve-se declarar a variável que armazenará essa palavra como um vetor² de caracteres de um tamanho definido no momento da declaração. O scanf lerá então a primeira palavra digitada pelo usuário na linha de comando e armazenará no vetor de caracteres, acrescentando um caractere de terminação de palavra (o caractere invisível '\0') logo após o último caractere da primeira palavra lida.

Por exemplo, se declarar uma variável palavra, como um vetor de caracteres de tamanho 15, pode-se armazenar uma palavra de até 14 caracteres (já que o 15º caractere será o '\0').

```
char palavra[15];  
scanf("%s", &palavra);
```

No exemplo, se o usuário digitar a palavra Algoritmo, a representação da memória que armazena o vetor palavra será a seguinte:

```
scanf("%s %d", &nome, &num);
```

Com isso pode-se ler "Paulo 18" ao mesmo tempo.

Outra forma utilizada para receber dados do usuário é usando o vetor de argumentos. Essa forma não possui uma interface gráfica e a coleta de dados é pela tela do MS-DOS. Quando se executa um determinado programa `programa.exe`, usa-se o seguinte comando no prompt do DOS:

```
c:\programa.exe
```

Pode-se também passar argumentos como parâmetro para o programa que está sendo executado da seguinte forma:

```
c:\programa.exe 5 Logica
```

O dado String "5" e o dado String "Logica" estão sendo passados como parâmetros para o método principal. Mas onde?

Como foi visto na Aula 1, a declaração completa do método principal de qualquer programa em C é:

```
int main ( int argc, char *argv[] )
```

Onde `argc` é uma variável do tipo inteiro que representa o contador de argumentos, armazenando a quantidade de argumentos passados pela linha de comando do DOS quando o programa é executado e `argv` é uma variável que representa o vetor de argumentos, ou seja, um vetor de strings que recebe o valor propriamente dito desses argumentos passados como parâmetro.

Por exemplo:

```
String n = argv[1];
```

As funções mais comuns que convertem strings são as seguintes:

Tabela 11 Funções de conversão de strings

Função	Função	Exemplo
<code>atoi</code>	Converte um valor do tipo string para inteiro	<pre>int x = atoi("1")</pre>
<code>atof</code>	Converte um valor do tipo string para real	<pre>float x = atod("1") double x = atof("1.3")</pre>

As funções de conversão de strings são definidas na biblioteca `stdlib.h`.

Comandos de saída

Para que o usuário possa ver o resultado do processamento dos dados ou alguma mensagem enviada pelo programa C, utiliza-se a função `printf`, que imprime uma mensagem formatada na saída do DOS.

A mensagem formatada contém a string que se deseja imprimir no ambiente DOS e para se imprimir uma mensagem simples, pode-se omitir a lista de argumentos. Por exemplo, para se imprimir uma mensagem simples na tela pode-se utilizar o seguinte comando:

```
printf("Algoritmos");
```

Porém, de forma muito parecida com a função `scanf`, pode-se fornecer uma mensagem formatada com marcadores de formatação, que serão substituídos por variáveis da lista de argumentos. Os marcadores de formatação são os mesmo utilizados na função `scanf` exibidos na Tabela 10.

Tabela 12 Marcadores de formatação no C

Formato	Função	Exemplo
<code>%d</code>	Numérico inteiro	<code>75</code>
<code>%f</code>	Numérico real em notação normal	<code>75.100000</code>
<code>.1f</code>	Numérico real com 1 dígito após o decimal	<code>75.1</code>
<code>%s</code>	Sequência de caracteres (string)	<code>"algoritmos"</code>
<code>%c</code>	Caractere	<code>'A'</code>

Por exemplo, considere que uma variável `num` foi declarada como tipo `int` e o valor 5 lhe foi atribuído:

```
int num = 5;
printf("%d", num);
```

Como a função substitui apenas o marcador, pode-se combinar o marcador com da-

dos alfanuméricos, fornecendo uma mensagem formatada conforme o exemplo:

```
int num = 5;
printf("Teste de inteiro: %d", num);
```

Blocos em C

Em programas C, um bloco de comandos tem seu início com a chave aberta, {, e seu término com a chave fechada, }, onde o bloco inicia-se com a chave aberta e termina com a chave fechada.

Note que um programa C também deve estar bem identado, isto é, deve estar numa formação que facilite o seu entendimento.

Estrutura sequencial em C

Em C, a estrutura sequencial não difere do pseudocódigo em relação ao comportamento, pois também representa um conjunto de ações ou comandos que são executados num determinado fluxo. Por exemplo:

```
main()
{
    char x[50], y[100];

    printf("Digite um valor:");
    scanf("%s", &x);
    printf("Digite um valor:");
    scanf("%s", &y);
    printf("Você digitou primeiro o %s",
x);
    printf("Você digitou logo em segui-
da o %s", y);
}
```

Exemplos de estrutura sequencial em C

1. Desenvolva um algoritmo que receba, do usuário, dois números reais e exiba a soma destes dois números.

```
main()
{
    float n1, n2, soma;

    /* mensagem ao usuário */

```

```
    printf("Digite dois numeros reais: ");

    /* entrada de dados */
    scanf("%f %f", &n1, &n2);

    /* processamento de dados */
    soma = n1 + n2;

    /* saída de resultados */
    printf("A soma dos dois numeros reais
digitados e %5.1f", soma);
}
```

2. Desenvolva um algoritmo que receba, do usuário, dois números inteiros, calcule e exiba a média aritmética destes dois números. Dica: a média aritmética é a soma dos dois números dividido por dois, por isso, o resultado deve ser um número real.

```
main()
{
    int num1, num2;
    float media;

    /* mensagem ao usuário */
    printf("Digite um numero inteiro: ");

    /* entrada de dados */
    scanf("%d", &num1);

    /* mensagem ao usuário */
    printf("Digite um numero inteiro: ");
    /* entrada de dados */
    scanf("%d", &num2);

    /* processamento de dados */
    media = (float)(num1 + num2) / 2;

    /* saída de resultados */
    printf("A media dos dois numeros digi-
tados e %5.1f", media);
}
```

ESTRUTURA SEQUENCIAL EM JAVA

Estudo da estrutura sequência na Linguagem de Programação Java.

Operador de atribuição (=) e comandos de atribuição

O comando de atribuição em Java será o igual (=). Por exemplo, considere que as variáveis x, y e z foram declaradas como do tipo int:

```
x = 25;  
y = x + 15 - 3;  
z = y - x + Math.sqrt(x) - Math.pow(y,2);
```

Comandos de entrada e saída

Comandos de entrada no Java

Para que o usuário possa entrar com dados num programa Java, utiliza-se o método `JOptionPane.showInputDialog` definido dentro do pacote `javax.swing`. Para isso, o programa deve fazer o import desta biblioteca no início do programa da seguinte forma:

```
import javax.swing.*;
```

Por exemplo, considere que uma variável num foi declarada como do tipo real:

```
num = Double.parseDouble(JOptionPane.  
showInputDialog("Digite um número"));
```

Outra forma utilizada para receber dados do usuário é usando o vetor de argumentos. Essa forma não possui uma interface gráfica e a coleta de dados é pela tela do MS-DOS. Quando se executa um determinado arquivo arquivo.class, usa-se o seguinte comando no prompt do DOS:

```
c:\ java arquivo
```

```
c:\ java arquivo i Logica
```

O dado String "i" e o dado String "Logica" estão sendo passados como parâmetros para o método principal. Mas onde?

O cabeçalho do método principal de qualquer programa Java é:

```
public static void main(String arg [])
```

arg é o nome dado para um vetor de Strings que recebe os argumentos passados como parâmetro quando o programa é executado. Por exemplo:

```
int num = Integer.parseInt(arg[0]);  
String n = arg[1];
```

Comandos de saída no Java

Para que o usuário possa ver o resultado do processamento dos dados ou alguma mensagem enviada pelo programa Java, utiliza-se o método `JOptionPane.showMessageDialog` definido dentro do pacote `javax.swing`. Para isso, o programa deve fazer o import desta biblioteca no início do programa da seguinte forma:

```
import javax.swing.*;
```

Por exemplo, considere que uma variável num foi declarada como do tipo inteiro e o valor 5 lhe foi atribuído:

```
num = 5;  
JOptionPane.showMessageDialog(null,  
num);
```

Ao invés de mostrar para o usuário o conteúdo de uma variável, pode-se mostrar um dado alfanumérico. Por exemplo:

```
JOptionPane.showMessageDialog(null,  
"Disciplina de Lógica de Programação");
```

Com o mesmo comando de saída, pode-se enviar uma mensagem cujo conteúdo seja a concatenação de vários dados String, sejam eles do tipo String ou da conversão de outros tipos de dados para o tipo String. Por exemplo, considere que as variáveis num1 e num2 foram declaradas como do tipo int e os valores 10 e 5 lhes foram, respectivamente, atribuídos:

```
num1 = 10;  
num2 = 5;  
JOptionPane.showMessageDialog(null,  
"Maria tem " + num1 + " anos e João tem " +  
num2 + " anos");
```

Outro método utilizado para mostrar o resultado de um processamento de dados ou alguma mensagem para o usuário é o método `System.out.print` ou `System.out.println`. Esses métodos não possuem uma interface gráfica e a mensagem é mostrada na tela do MS-DOS.

Blocos

Em programas Java, um bloco tem seu início com a chave aberta, `{`, e seu término com a chave fechada, `}`.

Note que um programa Java também deve estar bem identado, isto é, deve estar numa formação que facilite o seu entendimento.

Estrutura sequencial no Java

Em Java, a estrutura sequencial não difere do pseudocódigo em relação ao comportamento. Por exemplo:

```
class Exemplo_Sequencial
{
    public static void main(String arg[])
    {
        String x, y;
        x=JOptionPane.showInputDialog("Digite
um valor");
        y=JOptionPane.showInputDialog("Digite
um valor");
        JOptionPane.showMessageDialog(null,
"Você digitou primeiro o " + x);
        JOptionPane.showMessageDialog(null,
"Você digitou logo em seguida o " + y);
        System.exit(0);
    }
}
```

O resultado da execução deste exemplo é:

```
Digite um valor
// suponha que o usuário digitou 5 e clicou
no botão ok
Digite um valor
// suponha que o usuário digitou 10 e cli-
cou no botão ok
```

Você digitou primeiro o 5 // e o usuário clicou no botão ok

Você digitou logo em seguida o 10 // e o usuário clicou no botão ok

Exemplos de estrutura sequencial em java

1. Desenvolva um algoritmo que receba, do usuário, dois números reais e exiba a soma destes dois números.

```
class Exemplor_sequencial
{
    public static void main(String argumentos[])
    {
        double n1, n2, soma;

        // mensagem ao usuário e entrada de
        // dados
        n1 = Integer.parseInt(JOptionPane.
showInputDialog("Digite um número real:
"));

        n2 = Integer.parseInt(JOptionPane.
showInputDialog("Digite outro número
real: "));

        // processamento de dados
        soma = n1 + n2;

        // saída de resultados
        JOptionPane.showMessageDialog(null,
"A soma dos dois números reais digitados é
" + soma);
    }
}
```

2. Desenvolva um algoritmo que receba, do usuário, dois números inteiros, calcule e exiba a média aritmética destes dois números. Dica: a média aritmética é a soma dos dois números dividido por dois, por isso, o resultado deve ser um número real.

```
class Media
{
    public static void main (String argumentos[])
    {
        // declaração de variáveis
```

```

int num1, num2;
double media;

// mensagem ao usuário e entrada de
dados
num1 = Integer.parseInt(JOptionPane.
showInputDialog ("Digite um número inteiro"));
num2 = Integer.parseInt(JOptionPane.
showInputDialog ("Digite outro número inteiro"));
// processamento de dados
media = (num1 + num2) / 2;
// saída de resultados
JOptionPane.showMessageDialog
(null, "A média dos dois números digitados
é: " , media);
}

```

ESTRUTURA DE DECISÃO NOS ALGORITMOS

Estudo das estruturas de decisão simples, composta e encadeada nos algoritmos.

Quando se desenvolve um algoritmo ou programa estruturado, muitas vezes, precisa-se interferir no fluxo natural ou sequencial que o algoritmo deve seguir. No entanto, precisa-se saber controlar qual o fluxo que o algoritmo percorre do início até o fim de sua execução. Para isso, utiliza-se os fluxos de controle.

Uma estrutura de decisão é um fluxo de controle utilizado para decidir qual o fluxo que o algoritmo seguirá. Esta condição é representada por expressões lógicas e relacionais que podem ou não serem satisfeitas, isto é, podem retornar o valor verdadeiro ou falso.

Estrutura de Decisão Simples (se/então)

Uma estrutura de decisão simples pode ser utilizada quando o algoritmo precisa testar determinada condição antes de executar um conjunto de comandos. Neste caso, uma condição é avaliada e se seu resultado for verdadeiro, o conjunto de comandos dentro da estrutura se/então é executado. Por outro

lado, se o resultado da avaliação for falso, este conjunto de comandos não fará parte do fluxo de execução do algoritmo.

Na estrutura de decisão simples, utiliza-se as palavras **se** e **então** que representam as palavras principais desta estrutura e a palavra **fimse**; para determinar o fim do bloco de execução desta estrutura. Por exemplo:

```

se (var < 0)
então
    escreva("O número " , var , " é negativo");
fimse;

```

O resultado da execução deste exemplo é:

```

// se o valor de var for -5
O número -5 é negativo

```

Estrutura de Decisão Composta (se/então/senão)

Uma estrutura de decisão composta é parecida com a estrutura de decisão simples. A diferença entre as duas estruturas é que a primeira pode ser utilizada quando o algoritmo precisa testar determinada condição, cujos resultados podem ser verdadeiro ou falso, antes de executar um conjunto de comandos. Neste caso, uma condição é avaliada e se seu resultado for verdadeiro, o conjunto de comandos dentro da estrutura se/então é executado. Por outro lado, se o resultado da avaliação for falso, o conjunto de comandos dentro da estrutura senão é executado.

Na estrutura de decisão composta, utiliza-se as palavras **se**, **então** e **senão** que representam as palavras principais desta estrutura e a palavra **fimse**; para determinar o fim do bloco de execução desta estrutura. Por exemplo:

```

se (var < 0)
então
    escreva("O número " , var , " é negativo");
senão
    escreva("O número " , var , " é não negativo");

```

fimse;

O resultado da execução deste exemplo é:

```
// se o valor de var for 5  
O número 5 é não negativo
```

```
// se o valor de var for -5  
O número -5 é negativo
```

Estrutura de Decisão Encadeada (se/então ... / senão ...)

Diz-se que uma estrutura de decisão é encadeada se há estruturas de decisões dentro de outras estruturas de decisões. Não existe limite para a estrutura de decisão encadeada, pode-se ter quantas estruturas de decisão encadeadas forem necessárias. Por exemplo:

```
se (var < 0)  
então  
    escreva("O número ", var, " é negativo");  
senão  
    se (var = 0)  
        então  
            escreva("O número ", var, " é nulo");  
        senão  
            escreva("O número ", var, " é positivo");  
    fimse; // do var = 0  
fimse; // do var < 0
```

O resultado da execução deste exemplo é:

```
// se o valor de var for 5  
O número 5 é positivo  
  
// se o valor de var for -5  
O número -5 é negativo  
  
// se o valor de var for 0  
O número 0 é nulo
```

Exemplos de estrutura de decisão simples, composta e encadeada em pseudocódigo

1. Desenvolva um algoritmo que receba, do usuário, um valor alfanumérico, verifique e exiba a informação se este caractere é uma vogal.

```
Algoritmo Vogal  
início_algoritmo  
    Declarar  
        // declaração de variáveis e/ou constantes  
        vog                alfanumérico;  
        escrever ("Digite um valor alfanumérico");  
        ler (vog);  
        se (vog = 'a' ou vog = 'e' ou vog = 'i' ou  
            vog = 'o' ou vog = 'u')  
        então  
            escrever ("O caractere ", vog, " é  
            uma vogal.");  
        senão  
            escrever ("O caractere ", vog, " não  
            é uma vogal.");  
        fimse;  
    fim_algoritmo.
```

2. Desenvolva um algoritmo que receba, do usuário, dois valores reais, exiba a soma destes dois números, verifique e exiba a informação se a soma é positiva ou não positiva.

```
Algoritmo Soma  
início_algoritmo  
    Declarar  
        // declaração de variáveis e/ou constantes  
        num1, num2, soma  
        real;  
        escrever ("Digite dois números reais");  
        ler (num1, num2);  
        soma ← num1 + num2;  
        se (soma > 0)  
        então
```

```

        escrever ("A soma de “ , num1 , “
com “ , num2 , “ é igual a “ , soma , “ que é
positiva”);
    senão
        escrever (“A soma de “ , num1 , “
com “ , num2 , “ é igual a “ , soma , “ que é
não positiva”);
    fimse;
fim_algoritmo.

```

ESTRUTURA DE DECISÃO EM C

Estudo das estruturas de decisão simples, composta e encadeada na Linguagem C.

Estrutura de Decisão Simples (se/então)

Na estrutura de decisão simples, utiliza-se a palavra `if` para representar a palavra principal desta estrutura e chave aberta, `{`, e chave fechada, `}`, para representar o conjunto de comandos que fazem parte desta estrutura. Por exemplo:

```

if (var < 0)
{
    printf("O número %d é negativo", var);
}

```

O resultado da execução deste exemplo é:

```
// se o valor de var for -5
O número -5 é negativo
```

Estrutura de Decisão Composta (se/então/senão)

Na estrutura de decisão composta, utiliza-se a palavra `if` e `else` para representar as palavras principais desta estrutura e chave aberta, `{`, e chave fechada, `}`, para representar o conjunto de comandos que fazem parte desta estrutura. Por exemplo:

```

if (var < 0)
{
    printf("O número %d é negativo", var);
}

```

```

else
{
    printf("O número %d é não negativo",
var);
}

```

O resultado da execução deste exemplo é:

```
// se o valor de var for 5
O número 5 é não negativo
```

```
// se o valor de var for -5
O número -5 é negativo
```

Estrutura de Decisão Encadeada (se/então ... / senão ...)

Na estrutura de decisão encadeada, utiliza-se a palavra `if` e `else` para representar as palavras principais desta estrutura e chave aberta, `{`, e chave fechada, `}`, para representar o conjunto de comandos que fazem parte desta estrutura. Por exemplo:

```

if (var < 0)
{
    printf("O número %d é negativo", var);
}
else
{
    if (var == 0)
    {
        printf("O número %d é zero", var);
    }
    else
    {
        printf("O número %d é positivo",
var);
    } // do var = 0
} // do var < 0

```

O resultado da execução deste exemplo é:

```
// se o valor de var for 5
O número 5 é positivo
```

```
// se o valor de var for -5
O número -5 é negativo
```

```
// se o valor de var for 0
```

O número 0 é zero

Exemplos de estrutura de decisão simples, composta e encadeada em C

1. Desenvolva um algoritmo que receba, do usuário, um valor alfanumérico, verifica e exiba a informação se este caractere é uma vogal.

```
#include <stdio.h>

main()
{
    char vogal;

    printf("Digite um caractere: ");
    scanf("%c", &vogal);

    if ( vogal == 'a' || vogal == 'e' || vogal
        == 'i' || vogal == 'o' || vogal == 'u' || vogal ==
        'A' || vogal == 'E' || vogal == 'I' || vogal == 'O'
        || vogal == 'U' )
    {
        printf("O caractere \'%c\' é uma vogal.", vogal);
    }
    else
    {
        printf("O caractere \'%c\' não é uma vogal.", vogal);
    }
}
```

2. Desenvolva um algoritmo que receba, do usuário, dois valores reais, exiba a soma destes dois números, verifique e exiba a informação se a soma é positiva ou não positiva.

```
#include <stdio.h>

main()
{
    float num1, num2, soma;

    printf("Digite dois números reais: ");
    scanf("%f %f", &num1, &num2);
```

```
soma = num1 + num2;
```

```
if (soma > 0)
{
    printf("A soma de %f com %f é
igual a %f que é positiva", num1, num2,
soma);
}
else
{
    printf("A soma de %f com %f é
igual a %f que é não positiva", num1, num2,
soma);
}
```

ESTRUTURA DE DECISÃO EM JAVA

Estudo das estruturas de decisão simples, composta e encadeada na Linguagem Java.

Estrutura de Decisão Simples (se/então)

Na estrutura de decisão simples, utiliza-se a palavra `if` para representar a palavra principal desta estrutura e chave aberta, `{`, e chave fechada, `}`, para representar o conjunto de comandos que fazem parte desta estrutura. Por exemplo:

```
if (var < 0)
{
    System.out.print("O número " + var +
" é negativo");
}
```

O resultado da execução deste exemplo é:

```
// se o valor de var for -5
O número -5 é negativo
```

Estrutura de Decisão Composta (se/então/senão)

Na estrutura de decisão composta, utiliza-se a palavra `if` e `else` para representar as palavras principais desta estrutura e chave aberta, `{`, e chave fechada, `}`, para represen-

tar o conjunto de comandos que fazem parte desta estrutura. Por exemplo:

```
if (var < 0)
{
    System.out.print("O número " + var +
" é negativo");
}
else
{
    System.out.print("O número " + var +
" é não negativo");
}
```

O resultado da execução deste exemplo é:

```
// se o valor de var for 5
O número 5 é não negativo
```

```
// se o valor de var for -5
O número -5 é negativo
```

Estrutura de Decisão Encadeada (se/então ... / senão ...)

Na estrutura de decisão encadeada, utiliza-se a palavra **if** e **else** para representar as palavras principais desta estrutura e chave aberta, **{**, e chave fechada, **}**, para representar o conjunto de comandos que fazem parte desta estrutura. Por exemplo:

```
if (var < 0)
{
    System.out.print("O número " + var +
" é negativo");
}
else
{
    if (var == 0)
    {
        System.out.print("O número " +
var + " é nulo");
    }
    else
    {
        System.out.print("O número " + var +
" é positivo");
    } // do var = 0
} // do var < 0
```

O resultado da execução deste exemplo é:

```
// se o valor de var for 5
O número 5 é positivo
```

```
// se o valor de var for -5
O número -5 é negativo
```

```
// se o valor de var for 0
O número 0 é nulo
```

Exemplos de estrutura de decisão simples, composta e encadeada em Java

1. Desenvolva um algoritmo que receba, do usuário, um valor alfanumérico, verifica e exiba a informação se este caracter é uma vogal.

```
import javax.swing.*;
class Vogal
{
    public static void main (String arg [])
    {
        char vog;
        vog = (JOptionPane.
showInputDialog("Digite um valor alfanu-
mérico")).charAt(0);
        if (vog == 'a' || vog == 'e' || vog == 'i' ||
vog == 'o' || vog == 'u' ||
vog == 'A' || vog == 'E' || vog == 'I'
|| vog == 'O' || vog == 'U')
        {
            JOptionPane.showMessageDialog(null, "O caractere \' " +
vog + " \' é uma
vogal.");
        }
        else
        {
            JOptionPane.showMessageDialog(null, "O caractere \' " +
vog + " \' não é uma
vogal.");
        }
        System.exit(0);
    }
}
```

2. Desenvolva um algoritmo que receba, do usuário, dois valores reais, exiba a soma destes dois números, verifique e exiba a informação se a soma é positiva ou não positiva.

```

import javax.swing.*;

class Soma
{
    public static void main (String arg [])
    {
        double      num1,      num2,
soma;

        num1=Double.parseDouble(JOptionPane.
showInputDialog("Digite um número
real"));

        num2 = Double.
parseDouble (JOptionPane.
showInputDialog("Digite um número
real"));

        soma = num1 + num2;

        if (soma > 0)
        {
            JOptionPane.showMessageDialog(null, "A soma de " + num1 + " com " +
num2 + " é igual a " + soma + " que é positiva");
        }
        else
        {
            JOptionPane.showMessageDialog(null, "A soma de " + num1 + " com " +
num2 + " é igual a " + soma + " que é não
positiva");
        }
        System.exit(0);
    }
}

```

ESTRUTURA DE MÚLTIPLA ESCOLHA NOS ALGORITMOS

Estudo da estrutura de múltipla escolha simples e encadeada nos algoritmos.

Uma estrutura de múltipla escolha também é uma estrutura de decisão em que um

fluxo de controle é utilizado para decidir qual o fluxo que o algoritmo seguirá. Uma estrutura de múltipla escolha determina qual conjunto de comandos ou bloco será executado após uma opção ser avaliada e um caso para essa opção for detectada. Esta opção é representada por valores de quaisquer tipos de dados conhecidos. Note que somente um caso é executado, dentre todos os casos da estrutura de múltipla escolha.

Estrutura de Múltipla Escolha Simples

Uma estrutura de decisão de múltipla escolha pode ser utilizada quando o algoritmo precisa testar um conjunto de valores diferentes antes de executar um conjunto de comandos associado a esses valores. Neste caso, uma opção é avaliada, seu resultado é verificado dentro de um conjunto de casos e o conjunto de comandos dentro do caso relacionado à opção avaliada é executada como parte do fluxo de execução do algoritmo. Numa estrutura de múltipla escolha, o tipo de dado da opção precisa, necessariamente, ser do mesmo tipo dos casos relacionados.

Na estrutura de múltipla escolha simples, utiliza-se as palavras **escolha**, **caso** e **caso contrário** que representam as palavras principais desta estrutura e a palavra **fimescolha**; para determinar o fim do bloco de execução desta estrutura. Por exemplo:

```

escolha (var)
    caso 1 : escrever ("O valor da variável var
é 1");
    caso 2 : escrever ("O valor da variável var
é 2");
    caso 3 : escrever ("O valor da variável var
é 3");
    caso contrário : escrever ("O valor da
variável var não é nem 1, nem 2, nem 3");
fimescolha;

```

Note que o uso do comando **caso contrário** serve para os valores não previstos que a variável **var** possa assumir.

O resultado da execução deste exemplo é:

```
// se o valor de var for 1
O valor da variável var é 1

// se o valor de var for 2
O valor da variável var é 2

// se o valor de var for 3
O valor da variável var é 3

// se o valor de var for 4
O valor da variável var não é nem 1, nem
2, nem 3
```

Numa estrutura de múltipla escolha, os valores escolhidos para cada caso não precisam ser únicos, isto é, cada caso da estrutura pode ser representado por um conjunto de valores específicos. Por exemplo, um caso pode ter o valor 1, outro caso, os valores 2, 3 e 4, outro caso, os valores 7, 15 e 25 e outro caso com os valores 8, 11, 12, 13 e 14:

```
escolha (var)
caso 1 : escrever ("O valor da variável
var é 1");
caso 2 .. 4 : escrever ("O valor da variável
var pode ser 2, 3 ou 4");
caso 7, 15, 25 : escrever ("O valor da variável
var pode ser 7, 15 ou 25");
caso 8, 11 .. 14 : escrever ("O valor da variável
var pode ser 8, 11, 12, 13 ou 14");
caso contrário : escrever ("opção inválida.");
fimescolha;
```

O resultado da execução deste exemplo é:

```
// se o valor de var for 1
O valor da variável var é 1

// se o valor de var for 2
O valor da variável var pode ser 2, 3 ou 4

// se o valor de var for 7
O valor da variável var pode ser 7, 15 ou 25

// se o valor de var for 8
O valor da variável var pode ser 8, 11, 12,
13 ou 14

// se o valor de var for 100
Opção Inválida
```

Estrutura de Múltipla Escolha Encadeada

Uma estrutura de múltipla escolha encadeada pode ser utilizada quando o algoritmo precisa testar um conjunto de valores diferentes antes de executar um conjunto de comandos associados a esses valores. Por exemplo:

```
escolha (var)
caso 1 .. 3 : escolha (var)
caso 1 , 2 : escolha (var)
caso 1 : escrever
("o valor da variável var é 1");
caso 2 : escrever
("o valor da variável var é 2");
caso contrário
: escrever ("opção inválida.");
fimescolha;
caso 3 : escrever ("o valor da variável var é 3");
caso contrário : escrever
("opção inválida.");
fimescolha;
caso 4 , 5 : escolha (var)
caso 4 : escrever ("o valor da variável var é 4");
caso 5 : escrever ("o valor da variável var é 5");
caso contrário : escrever ("opção inválida.");
fimescolha;
caso 6 : escrever ("o valor da variável var é 6");
caso contrário : escrever ("opção inválida.");
fimescolha;
```

O resultado da execução deste exemplo é:

```
// se o valor de var for 1
O valor da variável var é 1

// se o valor de var for 2
O valor da variável var é 2

// se o valor de var for 3
O valor da variável var é 3

// se o valor de var for 4
O valor da variável var é 4
```

```
// se o valor de var for 5  
O valor da variável var é 5
```

```
// se o valor de var for 6  
O valor da variável var é 6
```

```
// se o valor de var for 7  
Opção Inválida
```

Exemplos de Estrutura de Múltipla Escolha em pseudocódigo

1. Desenvolva um algoritmo que recebe quatro notas bimestrais, calcula e mostra a média aritmética destas quatro notas, bem como, se o aluno foi aprovado (média ≥ 7), reprovado (média < 3), em exame (média ≥ 3 ou média < 7) ou aprovado com louvor (média = 10)

Algoritmo MediaAritmetica

 início_algoritmo

 Declarar // declaração de variáveis e/ou constantes

 n1, n2, n3, n4, media real;

 escrever ("Digite as quatro notas bimestrais"); // mensagem ao usuário

 ler (n1 , n2 , n3 , n4); // entrada de dados

 media (n1 + n2 + n3 + n4) / 4; // processamento de dados

 escrever ("A média é: " , media); // saída de resultados

 // processamento de dados

 escolha (media)

 caso 7 .. 10 : se (media = 10)

 então

 escrever("Aluno aprovado com louvor"); // saída de resultados

 senão

 escrever ("Aluno

 aprovado"); // saída de resultados

 fimse;

 caso 0 .. 2.9 : escrever ("Aluno reprovado"); // saída de resultados

 caso 3 .. 6.9 : escrever ("Aluno em exame"); // saída de resultados

 caso contrário : escrever ("média inválida"); // saída de resultados

```
fimescolha;  
fim_algoritmo.
```

2. Desenvolva um algoritmo que recebe o preço de um produto e seu código de origem e mostre o preço do produto junto de sua procedência, conforme tabela abaixo:

código de origem	região de procedência
1	Norte
2, 5, 9	Sul
3, 10 até 15	Leste
7 ou 20	Oeste
qualquer outro	Importado

Algoritmo Produto
 início_algoritmo

 Declarar // declaração de variáveis e/ou constantes

 preco real;

 codigo inteiro;

 escrever ("Digite o preço e o código de origem do produto"); // mensagem ao usuário

 ler (preco, codigo); // entrada de dados

 // processamento de dados

 escolha (codigo)

 caso 1 : escrever (preco , " - Norte"); // saída de resultados

 caso 2, 5, 9 : escrever (preco , " - Sul"); // saída de resultados

 caso 3, 10 .. 15 : escrever (preco , " - Leste"); // saída de resultados

 caso 7 , 20 : escrever (preco , " - Oeste"); // saída de resultados

 caso contrário : escrever (preco , " - Importado"); // saída de resultados

 fimescolha;

 fim_algoritmo.

§ ESTRUTURA DE MÚLTIPLA ESCOLHA EM C

Estudo da estrutura de múltipla escolha simples e encadeada na Linguagem C.

Estrutura de Múltipla Escolha Simples

Na estrutura de múltipla escolha simples, utiliza-se as palavras `switch`, `case`, `default` e `break` que representam as palavras principais desta estrutura e a chave aberta, `{`, e a chave fechada, `}`, para representar o conjunto de comandos que fazem parte desta estrutura. Por exemplo:

```
switch (var)
{
    case 1 : printf("O valor da variável var
é 1");
        break;
    case 2 : printf("O valor da variável var
é 2");
        break;
    case 3 : printf("O valor da variável var
é 3");
        break;
    default : printf ("O valor da variável var
não é nem 1, nem 2, nem 3");
}
```

O resultado da execução deste exemplo é:

```
// se o valor de var for 1
O valor da variável var é 1
```

```
// se o valor de var for 2
O valor da variável var é 2
```

```
// se o valor de var for 7
O valor da variável var é 3
```

```
// se o valor de var for 4
```

O valor da variável var não é nem 1, nem 2, nem 3

Numa estrutura de múltipla escolha, os valores escolhidos para cada caso não precisam ser únicos, isto é, cada caso da estrutura pode ser representado por um conjunto de valores específicos. Por exemplo, um caso pode ter o valor 1, outro caso, os valores 2, 3 e 4, outro caso, os valores 7, 15 e 25 e outro caso com os valores 8, 11, 12, 13 e 14:

```
switch (var)
{
```

```
    case 1 : printf("O valor da variável var
é 1");
        break;
    case 2 : case 3 : case 4 : printf("O valor
da variável var pode ser 2, 3 ou 4");
        break;
    case 7 : case 15 : case 25 : printf("O
valor da variável var pode ser 7, 15 ou 25");
        break;
    case 8 : case 11 : case 12 : case 13 : case
14 : printf("O valor da variável var pode ser 8
, 11, 12, 13 ou 14");
        break;
    default : printf("opção inválida.");
}
```

O resultado da execução deste exemplo é:

```
// se o valor de var for 1
O valor da variável var é 1
```

```
// se o valor de var for 2
O valor da variável var pode ser 2, 3 ou 4
```

```
// se o valor de var for 7
O valor da variável var pode ser 7, 15 ou 25
```

```
// se o valor de var for 8
O valor da variável var pode ser 8 , 11, 12,
13 ou 14
```

```
// se o valor de var for 100
Opção Inválida
```

Estrutura de Múltipla Escolha Encadeada

Na estrutura de múltipla escolha encadeada, utiliza-se as palavras `switch`, `case`, `default` e `break` que representam as palavras principais desta estrutura e a chave aberta, `{`, e a chave fechada, `}`, para representar o conjunto de comandos que fazem parte desta estrutura. Por exemplo:

```
switch (var)
{
    case 1 : case 2 : case 3 : switch (var)
    {
        case 1 : case 2 : switch (var)
        {

```

```

        case 1 : printf
("o valor da variável var é 1");
                break;
        case 2 : printf
("o valor da variável var é 2");
                break;
        default: printf
("opção inválida");
                }
                break;
        case 3 : printf ("o valor da
variável var é 3");
                break;
        default : printf ("opção
inválida");
                }
                break;
        case 4 : case 5 : switch (var)
{
            case 4 : printf ("o valor da
variável var é 4");
                    break;
            case 5 : printf ("o valor da
variável var é 5");
                    break;
            default : printf ("opção
inválida");
                    }
                    break;
        case 6 : printf ("o valor da variável
var é 6");
                    break;
        default : printf ("opção inválida");
    }

```

O resultado da execução deste exemplo é:

```

// se o valor de var for 1
O valor da variável var é 1

// se o valor de var for 2
O valor da variável var é 2

// se o valor de var for 3
O valor da variável var é 3

// se o valor de var for 4
O valor da variável var é 4

// se o valor de var for 5

```

<pre> case 1 : printf ("o valor da variável var é 1"); break; case 2 : printf ("o valor da variável var é 2"); break; default: printf ("opção inválida"); } break; case 3 : printf ("o valor da variável var é 3"); break; default : printf ("opção inválida"); } break; case 4 : case 5 : switch (var) { case 4 : printf ("o valor da variável var é 4"); break; case 5 : printf ("o valor da variável var é 5"); break; default : printf ("opção inválida"); } break; case 6 : printf ("o valor da variável var é 6"); break; default : printf ("opção inválida"); } </pre>	O valor da variável var é 5 // se o valor de var for 6 O valor da variável var é 6 // se o valor de var for 7 Opção Inválida
---	--

Exemplos de Estrutura de Múltipla Escolha em C

- Desenvolva um algoritmo que recebe quatro notas bimestrais, calcula e mostra a média aritmética destas quatro notas, bem como, se o aluno foi aprovado (média ≥ 7), reprovado (média < 3), em exame (média ≥ 3 ou média < 7) ou aprovado com louvor (média = 10)

```

#include <stdio.h>

main()
{
    // declaração de variáveis e/ou constantes
    float n1, n2, n3, n4, media;
    int op;
    // mensagem ao usuário e entrada de dados
    printf("Digite uma nota bimestral: ");
    scanf("%f", &n1);
    printf("Digite uma nota bimestral: ");
    scanf("%f", &n2);
    printf("Digite uma nota bimestral: ");
    scanf("%f", &n3);
    printf("Digite uma nota bimestral: ");
    scanf("%f", &n4);

    media = (n1 + n2 + n3 + n4) / 4; // processamento de dados

    printf("A média é: %f", media); // saída de resultados
}

```

```

// processamento de dados
if (media >= 7 && media <= 10)
{

```

```

        op = 1;
    }
else
{
    if (media < 7 && media >= 3)
    {
        op = 3;
    }
else
{
    if (media >= 0 && media < 3)
    {
        op = 2;
    }
}
}

switch (op)
{
case 1 : if (media == 10)
{
    printf("Aluno aprovado com louvor"); // saída de resultados
}
else
{
    printf ("Aluno aprovado"); // saída de resultados
}
break;
case 2 : printf ("Aluno reprovado"); // saída de resultados
break;
case 3 : printf ("Aluno em exame"); // saída de resultados
break;
default : printf ("média inválida"); // saída de resultados
} // fim do switch
}

```

2. Desenvolva um algoritmo que recebe o preço de um produto e seu código de origem e mostre o preço do produto junto de sua procedência, conforme tabela ao lado:

código de origem	região de procedência
1	Norte
2, 5, 9	Sul
3, 10 até 15	Leste
7 ou 20	Oeste
qualquer outro	Importado

```

main()
{
    // declaração de variáveis e/ou constantes
}

```

```

float preco;
int codigo;

// mensagem ao usuário e entrada de dados
printf("Digite o preço do produto: ");
scanf("%f", &preco);
printf("Digite o código de origem do produto: ");
scanf("%d", &codigo);

switch (codigo)
{
case 1 : printf ("%f - Norte", preco); // saída de resultados
}

```

```

        break;
    case 2 : case 5 : case 9 : printf ("%f - Sul", preco); // saída de resultados
        break;
    case 3 : case 10 : case 11 : case 12 :
    case 13 : case 14 : case 15 : printf ("%f - Leste", preco); // saída de resultados
        break;
    case 7 : case 20 : printf ("%f - Oeste", preco); // saída de resultados
        break;
    default : printf ("%f - Importado", preco); // saída de resultados
}
} // fim do switch
}

```

ESTRUTURA DE MÚLTIPLA ESCOLHA EM JAVA

Estudo da estrutura de múltipla escolha simples e encadeada na Linguagem Java.

Estrutura de Múltipla Escolha Simples

Na estrutura de múltipla escolha simples, utiliza-se as palavras `switch`, `case`, `default` e `break` que representam as palavras principais desta estrutura e a chave aberta, `{`, e a chave fechada, `}`, para representar o conjunto de comandos que fazem parte desta estrutura. Por exemplo:

```

switch (var)
{
    case 1 : System.out.print ("O valor da variável var é 1");
        break;
    case 2 : System.out.print ("O valor da variável var é 2");
        break;
    case 3 : System.out.print ("O valor da variável var é 3");
        break;
    default : System.out.print ("O valor da variável var não é nem 1, nem 2, nem 3");
}

```

O resultado da execução deste exemplo é:

```
// se o valor de var for 1
```

O valor da variável var é 1

```
// se o valor de var for 2
```

O valor da variável var é 2

```
// se o valor de var for 7
```

O valor da variável var é 3

```
// se o valor de var for 4
```

O valor da variável var não é nem 1, nem 2, nem 3

Numa estrutura de múltipla escolha, os valores escolhidos para cada caso não precisam ser únicos, isto é, cada caso da estrutura pode ser representado por um conjunto de valores específicos. Por exemplo, um caso pode ter o valor 1, outro caso, os valores 2, 3 e 4, outro caso, os valores 7, 15 e 25 e outro caso com os valores 8, 11, 12, 13 e 14:

```

switch (var)
{
    case 1 : System.out.println ("O valor da variável var é 1");
        break;
    case 2 : case 3 : case 4 : System.out.println ("O valor da variável var pode ser 2, 3 ou 4");
        break;
    case 7 : case 15 : case 25 : System.out.println ("O valor da variável var pode ser 7, 15 ou 25");
        break;
    case 8 : case 11 : case 12 : case 13 : case 14 : System.out.println ("O valor da variável var pode ser 8, 11, 12, 13 ou 14");
        break;
    default : System.out.println ("opção inválida.");
}

```

O resultado da execução deste exemplo é:

```
// se o valor de var for 1
```

O valor da variável var é 1

```
// se o valor de var for 2
```

O valor da variável var pode ser 2, 3 ou 4

```
// se o valor de var for 7  
O valor da variável var pode ser 7, 15 ou 25
```

O valor da variável var pode ser 8 , 11, 12,
13 ou 14

```
// se o valor de var for 8
```

// se o valor de var for 100
Opção Inválida

Estrutura de Múltipla Escolha Encadeada

Na estrutura de múltipla escolha encadeada, utiliza-se as palavras `switch`, `case`, `default` e `break` que representam as palavras principais desta estrutura e a chave aberta, `{`, e a chave fechada, `}`, para representar o conjunto de comandos que fazem parte desta estrutura. Por exemplo:

```
switch (var)  
{  
    case 1 : case 2 : case 3 : switch (var)  
    {  
        case 1 : case 2 : switch (var)  
        {  
            case 1 : System.out.print ("o valor da variável var é 1");  
            break;  
            case 2 : System.out.print ("o valor da variável var é 2");  
            break;  
            default : System.out.print ("opção inválida");  
        }  
        break;  
        case 3 : System.out.print ("o valor da variável var é 3");  
        break;  
        default : System.out.print ("opção inválida");  
    }  
    break;  
    case 4 : case 5 : switch (var)  
    {  
        case 4 : System.out.print ("o valor da variável var é 4");  
        break;  
        case 5 : System.out.print ("o valor da variável var é 5");  
        break;  
        default : System.out.print ("opção inválida");  
    }  
    break;  
    case 6 : System.out.print ("o valor da variável var é 6");  
    break;  
    default : System.out.print ("opção inválida");  
}
```

O resultado da execução deste exemplo é:

```
// se o valor de var for 1  
O valor da variável var é 1
```

```
// se o valor de var for 2  
O valor da variável var é 2
```

```
// se o valor de var for 3  
O valor da variável var é 3
```

// se o valor de var for 4
O valor da variável var é 4

// se o valor de var for 6
O valor da variável var é 6

// se o valor de var for 5
O valor da variável var é 5

// se o valor de var for 7
Opção Inválida

Exemplos de Estrutura de Múltipla Escolha em Java

1. Desenvolva um algoritmo que recebe quatro notas bimestrais, calcula e mostra a média aritmética destas quatro notas, bem como, se o aluno foi aprovado ($\text{média} \geq 7$), reprovado ($\text{média} < 3$), em exame ($\text{média} \geq 3$ ou $\text{média} < 7$) ou aprovado com louvor ($\text{média} = 10$)

```
class MediaAritmetica
{
    public static void main(String arg [])
    {
        // declaração de variáveis e/ou constantes
        double n1, n2, n3, n4, media;
        int op=0;
        // mensagem ao usuário e entrada de dados
        n1 = Double.parseDouble(JOptionPane.showInputDialog("Digite uma nota bimestral"));
        n2 = Double.parseDouble(JOptionPane.showInputDialog("Digite uma nota bimestral"));
        n3 = Double.parseDouble(JOptionPane.showInputDialog("Digite uma nota bimestral"));
        n4 = Double.parseDouble(JOptionPane.showInputDialog("Digite uma nota bimestral"));

        media = (n1 + n2 + n3 + n4) / 4; // processamento de dados
        System.out.println ("A média é: " + media); // saída de resultados
        // processamento de dados
        if (media>=7 && media<=10)
        {
            op = 1;
        }
        else
        {
            if (media<7 && media >=3)
            {
                op = 3;
            }
            else
            {
                if (media>=0 && media<3)
                {
                    op = 2;
                }
            }
        }
        switch (op)
        {
            case 1 : if (media == 10)
```

```

        {
            System.out.println("Aluno aprovado com louvor"); // saída de resultados
        }
    else
    {
        System.out.println ("Aluno aprovado"); // saída de resultados
    }
    break;
case 2 : System.out.println ("Aluno reprovado"); // saída de resultados
break;
case 3 : System.out.println ("Aluno em exame"); // saída de resultados
break;
default : System.out.println ("média inválida"); // saída de resultados
} // fim do switch
System.exit(0);
} // fim do void main
} // fim da classe

```

2. Desenvolva um algoritmo que recebe o preço de um produto e seu código de origem e mostre o preço do produto junto de sua procedência, conforme tabela ao lado:

código de origem	região de procedência
1	Norte
2,5,9	Sul
3,10 até 15	Leste
7 ou 20	Oeste
qualquer outro	Importado

```

class Produto
{
    public static void main(String arg [])
    {
        // declaração de variáveis e/ou
        constantes
        double preco;
        int codigo;

        // mensagem ao usuário e entrada
        de dados
        preco=Double.parseDouble(JOptionPane.
        showInputDialog("Digite o preço do produ-
        to"));
        codigo=Integer.parseInt(JOptionPane.
        showInputDialog("Digite o código de ori-
        gem do produto"));
        // processamento de dados
        switch (codigo)
        {
            case 1 : System.out.println (preco
            + " - Norte"); // saída de resultados

```

```

            break;
        case 2 : case 5 : case 9 : System.
        out.println (preco + " - Sul"); // saída de re-
        sultados
        break;
        case 3 : case 10 : case 11 : case 12 :
        case 13 : case 14 : case 15 : Sys-
        tem.out.println (preco + " - Leste"); // saída
        de resultados
        break;
        case 7 : case 20 : System.out.prin-
        tln (preco + "- Oeste"); // saída de resultados
        break;
        default : System.out.println (pre-
        co + "- Importado"); // saída de resultados
    } // fim do switch
    System.exit(0);
} // fim do void main
} // fim da classe

```

ESTRUTURAS DE REPETIÇÃO DO ENQUANTO NOS ALGORITMOS

Estudo das estruturas de repetição do enquanto simples e encadeada nos algoritmos.

Uma estrutura de repetição é um fluxo de controle utilizado para decidir quantas vezes determinado conjunto de comandos se repetirá dentro do algoritmo.

Tem-se três tipos de estrutura de repetição: enquanto, faça/enquanto e para. Cada uma

dessas estruturas de repetição pode ser simples ou encadeada.

Uma estrutura de repetição enquanto pode ser utilizada quando o algoritmo precisa testar determinada condição antes de executar um conjunto de comandos repetidas vezes. Neste caso, uma condição é avaliada e se seu resultado for verdadeiro, o conjunto de comandos dentro da estrutura de repetição enquanto é executado e após esta execução, a condição é novamente avaliada. Por outro lado, se o resultado da avaliação for falso, este conjunto de comandos não será executado e o fluxo do algoritmo segue normalmente.

Na estrutura de repetição enquanto, utiliza-se as palavras enquanto e faça que repre-

sentam as palavras principais desta estrutura e a palavra fimenquanto; para determinar o fim do bloco de execução desta estrutura. Por exemplo:

```
x ← 0;  
enquanto (x<3) faça  
    escreva ("O valor de x é: " , x);  
    x ← x + 1;  
fimenquanto;  
O resultado da execução deste exemplo é:  
  
O valor de x é 0  
O valor de x é 1  
O valor de x é 2
```

Encadeamento

Na estrutura de repetição enquanto, utiliza-se as palavras enquanto e faça que representam as palavras principais desta estrutura e a palavra fimenquanto; para determinar o fim do bloco de execução desta estrutura. Por exemplo:

Desenvolva um algoritmo que calcula o fatorial dos números inteiros de 5 até 15.

```
Algoritmo ExemploEnquanto  
início_algoritmo  
    Declarar  
        i, num, fat inteiro;  
  
        num ← 5; // entrada de dados  
        // processamento de dados  
        enquanto (num <= 15) faça  
            fat ← num;  
            i ← num - 1;  
            enquanto (i > 1) faça  
                fat ← fat * i;  
                i ← i - 1;  
            fimenquanto;  
            escreva ("O valor do fatorial de " , num , " é " , fat); // saída de dados  
            num ← num + 1;  
        fimenquanto;  
    fim_algoritmo.
```

Exemplo de Estrutura de Repetição Enquanto em pseudocódigo

1. Desenvolva um algoritmo que calcula a soma de todos os números compreendidos entre 1 e 100.

```
// na estrutura de repetição enquanto  
Algoritmo Soma
```

Declarar

i, s \leftarrow o inteiro;

```
i  $\leftarrow$  1;  
enquanto (i  $\leq$  100) faça  
    s  $\leftarrow$  s + i;  
    i  $\leftarrow$  i + 1;  
fimenquanto;  
escreva ("A somatória de 1 até 100 é " , s);  
fim_algoritmo.
```

2. Desenvolva um algoritmo que calcula a soma de todos os números pares compreendidos entre 1 e 100.

```
// na estrutura de repetição enquanto  
Algoritmo Soma  
Declarar  
    i, s  $\leftarrow$  o inteiro;  
  
    i  $\leftarrow$  1;  
    enquanto (i  $\leq$  100) faça  
        se ((i mod 2) = 0)  
            então  
                s  $\leftarrow$  s + i;  
            fimse;  
        i  $\leftarrow$  i + 1;  
    fimenquanto;  
    escreva ("A somatória dos números pares  
de 1 até 100 é " , s);  
fim_algoritmo.
```

ESTRUTURAS DE REPETIÇÃO DO FAÇA/ ENQUANTO

Estudo das estruturas de repetição do faça/enquanto simples e encadeada nos algoritmos.

Faça/Enquanto

Uma estrutura de repetição faça/enquanto pode ser utilizada quando o algoritmo precisa testar determinada condição depois de executar um conjunto de comandos, passando por esses passos repetidas vezes.

Na estrutura de repetição faça/enquanto, utiliza-se as palavras faça e enquanto que representam as palavras principais desta estru-

tura e a própria palavra enquanto determinará o fim do bloco de execução desta estrutura. Por exemplo:

```
x  $\leftarrow$  0;  
faça  
    escreva ("O valor de x é: " , x);  
    x  $\leftarrow$  x + 1;  
enquanto (x < 3);
```

Resultado da execução deste exemplo:

```
O valor de x é 0  
O valor de x é 1  
O valor de x é 2
```

Para um melhor entendimento do encadeamento nas estruturas de repetição faça/enquanto, observe o seguinte exemplo:

Desenvolva um algoritmo que calcula o factorial dos números inteiros de 5 até 15.

```
Algoritmo ExemploFacaEnquanto  
início_algoritmo  
Declarar  
    i, num, fat inteiro;  
  
    num  $\leftarrow$  5; // entrada de dados  
    // processamento de dados  
    faça  
        fat  $\leftarrow$  num;  
        i  $\leftarrow$  num - 1;  
        faça  
            fat  $\leftarrow$  fat * i;  
            i  $\leftarrow$  i - 1;  
        enquanto (i > 1);  
        escreva ("O valor do factorial de " , num  
, " é " , fat); // saída de dados  
        num  $\leftarrow$  num + 1;  
    enquanto (num  $\leq$  15);  
fim_algoritmo.
```

Exemplo de Estrutura de Repetição Faça/ Enquanto em pseudocódigo

1. Desenvolva um algoritmo que calcula a soma de todos os números compreendidos entre 1 e 100.

```
// na estrutura de repetição faça/enquanto
Algoritmo Soma
Declarar
    i, s ← o inteiro;

    i ← 1;
    faça
        s ← s + i;
        i ← i + 1;
    enquanto (i <= 100);
    escreva ("A somatória de 1 até 100 é " , s);
fim_algoritmo.
```

2. Desenvolva um algoritmo que calcula a soma de todos os números pares compreendidos entre 1 e 100.

```
// na estrutura de repetição faça/enquanto
Algoritmo Soma
Declarar
    i, s ← o inteiro;

    i ← 1;
    faça
        se ((i mod 2) = 0)
            então
                s ← s + i;
        fimse;
        i ← i + 1;
    enquanto (i <= 100);
    escreva ("A somatória dos números pares
de 1 até 100 é " , s);
fim_algoritmo.
```

ESTRUTURAS DE REPETIÇÃO DO WHILE E DO/WHILE EM C

Estudo das estruturas de repetição do while e do/while simples e encadeada na Linguagem C.

Estrutura de Repetição Enquanto em C

Em C, para a estrutura de repetição enquanto, utiliza-se a palavra while que representa a palavra principal desta estrutura e chave aberta, {, e chave fechada, }, para repre-

sentar o conjunto de comandos que fazem parte desta estrutura. Por exemplo:

```
x = 0;
while (x<3)
{
    printf("O valor de x é: %d \n" + x);
    x = x + 1;
}
```

O resultado da execução deste exemplo é:

```
O valor de x é 0
O valor de x é 1
O valor de x é 2
```

Encadeamento

Em C, para a estrutura de repetição enquanto, utiliza-se a palavra while que representa a palavra principal desta estrutura e chave aberta, {, e chave fechada, }, para representar o conjunto de comandos que fazem parte desta estrutura.

Para um melhor entendimento do encadeamento nas estruturas de repetição enquanto, observe o seguinte exemplo:

Desenvolva um algoritmo que calcula o factorial dos números inteiros de 5 até 15.

A solução deste exercício em C é a seguinte:

```
#include <stdio.h>

main()
{
    int i, num, fat;

    num = 5;
    while (num <= 15)
    {
        fat = num;
        i = num - 1;
        while (i > 1)
        {
            fat = fat * i;
            i = i - 1;
        }
        // segundo enquanto
    }
}
```

```

        printf("O valor do fatorial de %d
é %d \n", num, fat);
        num = num + i;
    } // primeiro enquanto
}

```

Faça/Enquanto em C

Em C, para a estrutura de repetição **faça/ enquanto**, utiliza-se as palavras **do** e **while** que representam as palavras principais desta estrutura e chave aberta, **{**, e chave fechada, **}**, para representar o conjunto de comandos que fazem parte desta estrutura. Por exemplo:

```

x = 0;
do
{
    printf("O valor de x é: %d \n", x);
    x = x + 1;
} while (x<3);

```

Resultado da execução deste exemplo:

```

O valor de x é 0
O valor de x é 1
O valor de x é 2

```

Encadeamento

Em C, para a estrutura de repetição **faça/ enquanto**, utiliza-se as palavras **do** e **while** que representam as palavras principais desta estrutura e chave aberta, **{**, e chave fechada, **}**, para representar o conjunto de comandos que fazem parte desta estrutura.

Para um melhor entendimento do encadeamento nas estruturas de repetição **faça/ enquanto**, observe o seguinte exemplo:

Desenvolva um algoritmo que calcula o fatorial dos números inteiros de 5 até 15.

A solução deste exercício em C é a seguinte:

```
#include <stdio.h>
```

```
main()
```

```

{
    int i, num, fat;

    num = 5;
    do
    {
        fat = num;
        i = num - 1;
        do
        {
            fat = fat * i;
            i = i - 1;
        } while (i > 1);
        printf("O valor do fatorial de %d é %d
\ n", num, fat);
        num = num + 1;
    } while (num <= 15);
}

```

Exemplo de Estrutura de Repetição Enquanto e Faça/Enquanto no C

1. Desenvolva um algoritmo que calcula a soma de todos os números compreendidos entre 1 e 100.

```
// na estrutura de repetição enquanto
#include <stdio.h>
```

```
main()
{
    int i, s = 0;

    i = 1;
    while (i <= 100)
    {
        s = s + i;
        i = i + 1;
    } // fim do while
    printf("A somatória de 1 até 100 é
%d", s);
} // fim do main
```

```
// na estrutura de repetição faça/enquanto
#include <stdio.h>
```

```
main()
{
    int i, s = 0;
```

```

i = 1;
do
{
    s = s + i;
    i = i + 1;
} while (i <= 100); // fim do while
printf("A somatória de 1 até 100 é
%d", s);
} // fim do void main

```

2. Desenvolva um algoritmo que calcula a soma de todos os números pares compreendidos entre 1 e 100.

```

// na estrutura de repetição enquanto
#include <stdio.h>

main()
{
    int i, s = 0;

    i = 1;
    while (i <= 100)
    {
        if ((i % 2) == 0)
        {
            s = s + i;
        } // fim do IF
        i = i + 1;
    } // fim do while
    printf("A somatória de 1 até 100 é
%d", s);
} // fim do main

```

// na estrutura de repetição faça/enquanto

```

#include <stdio.h>

main()
{
    int i, s = 0;

    i = 1;
    do
    {
        if ((i % 2) == 0)
        {
            s = s + i;
        } // fim do IF

```

```

        i = i + 1;
    } while (i <= 100); // fim do while
    printf("A somatória de 1 até 100 é
%d", s);
} // fim do void main

```

ESTRUTURAS DE REPETIÇÃO DO WHILE E DO/WHILE EM JAVA

Estudo das estruturas de repetição do while e do/while simples e encadeada na Linguagem Java.

Estrutura de Repetição Enquanto no Java

No Java, para a estrutura de repetição enquanto, utiliza-se a palavra `while` que representa a palavra principal desta estrutura e chave aberta, `{`, e chave fechada, `}`, para representar o conjunto de comandos que fazem parte desta estrutura. Por exemplo:

```

x = 0;
while (x<3)
{
    System.out.println ("O valor de x é: "
+ x);
    x = x + 1;
}

```

O resultado da execução deste exemplo é:

```

O valor de x é 0
O valor de x é 1
O valor de x é 2

```

Encadeamento

No Java, para a estrutura de repetição enquanto, utiliza-se a palavra `while` que representa a palavra principal desta estrutura e chave aberta, `{`, e chave fechada, `}`, para representar o conjunto de comandos que fazem parte desta estrutura.

Para um melhor entendimento do encadeamento nas estruturas de repetição enquanto, observe o seguinte exemplo:

Desenvolva um algoritmo que calcula o fatorial dos números inteiros de 5 até 15.

A solução deste exercício em Java é a seguinte:

```
class ExemploEnquanto
{
    public static void main(String args [])
    {
        i, num, fat intreiro;

        num = 5;
        while (num <= 15)
        {
            fat = num;
            i = num - 1;
            while (i > 1)
            {
                fat = fat * i;
                i = i - 1;
            } // segundo enquanto
            System.out.print ("O valor do
fatorial de " + num + " é " + fat);
            num = num + 1;
        } // primeiro enquanto
    } // main
} // classe
```

Faça/Enquanto no Java

No Java, para a estrutura de repetição **faça/enquanto**, utiliza-se as palavras **do** e **while** que representam as palavras principais desta estrutura e chave aberta, **{**, e chave fechada, **}**, para representar o conjunto de comandos que fazem parte desta estrutura. Por exemplo:

```
x = 0;
do
{
    System.out.println ("O valor de x é: "
+ x);
    x = x + 1;
} while (x<3);
```

Resultado da execução deste exemplo:

```
O valor de x é 0
O valor de x é 1
```

O valor de x é 2

Encadeamento

No Java, para a estrutura de repetição **faça/enquanto**, utiliza-se as palavras **do** e **while** que representam as palavras principais desta estrutura e chave aberta, **{**, e chave fechada, **}**, para representar o conjunto de comandos que fazem parte desta estrutura.

Para um melhor entendimento do encadeamento nas estruturas de repetição **faça/enquanto**, observe o seguinte exemplo:

Desenvolva um algoritmo que calcula o fatorial dos números inteiros de 5 até 15.

A solução deste exercício em Java é a seguinte:

```
class ExemploFacaEnquanto
{
    public static void main(String args [])
    {
        i, num, fat intreiro;

        num = 5;
        do
        {
            fat = num;
            i = num - 1;
            do
            {
                fat = fat * i;
                i = i - 1;
            } while (i > 1);
            System.out.print ("O valor do
fatorial de " + num + " é " + fat);
            num = num + 1;
        } while (num <= 15);
    } // main
} // classe
```

Exemplo de Estrutura de Repetição Enquanto, Faça/Enquanto e Para em Java

1. Desenvolva um algoritmo que calcula a soma de todos os números compreendidos entre 1 e 100.

```

// na estrutura de repetição enquanto
class Soma
{
    public static void main (String args [])
    {
        int i, s = 0;

        i = 1;
        while (i <= 100)
        {
            s = s + i;
            i = i + 1;
        } // fim do while
        System.out.println ("A somatória de
i até 100 é " + s);
        System.exit(0);
    } // fim do void main
} // fim da classe

// na estrutura de repetição faça/enquanto
class Soma
{
    public static void main (String args [])
    {
        int i, s = 0;

        i = 1;
        do
        {
            s = s + i;
            i = i + 1;
        } while (i <= 100);
        System.out.println ("A somatória de
i até 100 é " + s);
        System.exit(0);
    } // fim do void main
} // fim da classe

// na estrutura de repetição para
class Soma
{
    public static void main (String args [])
    {
        int i, s = 0;

        for ( i = 1 ; i <= 100 ; i++)
        {
            s = s + i;
        }
    }
}

```

```

    }

    System.out.println ("A somatória de
i até 100 é " + s);
    System.exit(0);
} // fim do void main
} // fim da classe

```

2. Desenvolva um algoritmo que calcula a soma de todos os números pares compreendidos entre 1 e 100.

```

// na estrutura de repetição enquanto
class Soma
{
    public static void main (String args [])
    {
        int i, s = 0;

        i = 1;
        while (i <= 100)
        {
            if ((i % 2) == 0)
            {
                s = s + i;
            } // fim do if
            i = i + 1;
        } // fim do while
        System.out.println ("A somatória
dos números pares de i até 100 é " + s);
        System.exit(0);
    } // fim do void main
} // fim da classe

// na estrutura de repetição faça/enquanto
class Soma
{
    public static void main (String args [])
    {
        int i, s = 0;

        i = 1;
        do
        {
            if ((i % 2) == 0)
            {
                s = s + i;
            } // fim do se
            i = i + 1;
        } while (i <= 100);
    }
}

```

```

        System.out.println ("A somatória
dos números pares de 1 até 100 é " + s);
        System.exit(0);
    } // fim do void main
} // fim da classe

// na estrutura de repetição para
class Soma
{
    public static void main (String args [])
    {
        int i, s = 0;

        for ( i = 1 ; i <= 100 ; i++ )
        {
            if ((i % 2) == 0)
            {
                s = s + i;
            } // fim do if
        } // fim do for
        System.out.println ("A somatória
dos números pares de 1 até 100 é " + s);
        System.exit(0);
    } // fim do void main
} // fim da classe

```

EXERCÍCIOS 1

1. Desenvolva um algoritmo que receba dois números inteiros e mostre a soma desses números.
2. Desenvolva um algoritmo que receba dois números inteiros e mostre o quociente e o resto da divisão do primeiro número pelo segundo.
3. Desenvolva um algoritmo que receba seu nome e sobrenome e mostre essas informações concatenadas.
4. Desenvolva um algoritmo que receba um número e verifique se ele é par ou ímpar.
5. Desenvolva um algoritmo que receba uma nota e verifique se com essa nota o aluno foi aprovado ($nota \geq 5$), reprovado ($nota < 3$) ou em recuperação ($nota < 5$ e $nota \geq 3$).
6. Desenvolva um algoritmo que receba um mês e verifique se ele corresponde ao primeiro semestre ou ao segundo semestre.
7. Desenvolva um algoritmo que receba um mês e verifique se ele corresponde ao primeiro, segundo, terceiro ou quarto trimestre.
8. Desenvolva um algoritmo que receba um dia da semana e mostre suas tarefas nesses dias.
9. Desenvolva um algoritmo que receba dez número e mostre a soma desses números.
10. Desenvolva um algoritmo que receba quinze números e mostre a soma dos números pares.
11. Desenvolva um algoritmo que receba um número e calcule o fatorial desse número.
12. Desenvolva um algoritmo que receba um número e mostre o resultado da tabuada desse número.
13. Desenvolva um algoritmo que receba vinte números e mostre o maior desses números.
14. Desenvolva um algoritmo que receba dois números e calcule a soma do quadrado desses números.
15. Desenvolva um algoritmo que receba dois números e calcule a soma do cubo desses números.
16. Desenvolva um algoritmo que receba um número e mostre a raiz quadrada desse número.
17. Desenvolva um algoritmo que receba duas notas e mostre a média aritmética dessas notas.
18. Desenvolva um algoritmo que um número inteiro e verifique se esse número é divisível por 3.
19. Desenvolva um algoritmo que receba a medida do lado de um quadrado e mostre sua área.
20. Desenvolva um algoritmo que receba o raio de uma circunferência e calcule a medida do seu comprimento.
21. Desenvolva um algoritmo que receba o raio de um círculo e mostre sua área.
22. Desenvolva um algoritmo que receba trinta número e mostre a soma desses números.

23. Desenvolva um algoritmo que calcula o quadrado dos números de 10 a 20.
24. Desenvolva um algoritmo que receba a idade de vinte pessoas e mostre a idade da pessoa mais nova.
25. Desenvolva um algoritmo que receba dez números e calcule a média desses dez números.

ESTRUTURA DE REPETIÇÃO DO PARA

Estudo das estruturas de repetição do para simples nos algoritmos.

Estrutura de repetição do Para no pseudocódigo

Uma estrutura de repetição para pode ser utilizada quando o algoritmo precisa ter definido a quantidade de vezes que um conjunto de comandos deve ser executado.

Na estrutura de repetição para, utiliza-se as palavras para, de, até, passo, faça que representam as palavras principais desta estrutura e a palavra fimpara; para determinar o fim do bloco de execução desta estrutura. Por exemplo:

```
para x de 0 até 2 passo +1 faça
    escreva ("O valor de x é: ", x);
fimpara;
```

O resultado da execução deste exemplo é:

```
O valor de x é 0
O valor de x é 1
O valor de x é 2
```

Encadeamento

Na estrutura de repetição para, utiliza-se as palavras para, de, até, passo, faça que representam as palavras principais desta estrutura e a palavra fimpara; para determinar o fim do bloco de execução desta estrutura.

Para um melhor entendimento do encadeamento nas estruturas de repetição para, observe o seguinte exemplo:

Desenvolva um algoritmo que calcula o fatorial dos números inteiros de 5 até 15.

```
Algoritmo ExemploPara
início_algoritmo
    Declarar
        i, num, fat inteiro;
    // processamento de dados
    para num de 5 até 15 passo +1 faça
        fat ← num;
        para i de (num - 1) até 2 passo -1 faça
            fat ← fat * i;
        fimpara;
        escreva ("O valor do fatorial de ", num,
        " é ", fat); // saída de dados
    fimpara;
fim_algoritmo.
```

Exemplo de Estrutura do Para em pseudocódigo

1. Desenvolva um algoritmo que calcula a soma de todos os números compreendidos entre 1 e 100.

```
// na estrutura de repetição para
Algoritmo Soma
    Declarar
        i, s ← 0 inteiro;
    para i de 1 até 100 passo +1 faça
        s ← s + i;
    fimpara;
    escreva ("A somatória de 1 até 100 é ", s);
fim_algoritmo.
```

2. Desenvolva um algoritmo que calcula a soma de todos os números pares compreendidos entre 1 e 100.

```
// na estrutura de repetição para
Algoritmo Soma
    Declarar
        i, s ← 0 inteiro;
    para i de 1 até 100 passo +1 faça
        se ((i mod 2) = 0)
            então
                s ← s + i;
        fimse;
    fimpara;
```

escreva ("A somatória dos números pares de 1 até 100 é " , s);
fim_algoritmo.

ESTRUTURA DE REPETIÇÃO DO ENQUANTO, FAÇA/ENQUANTO E PARA

Desenvolva um algoritmo que mostra um menu de opções para: calcular a soma de todos os números compreendidos entre 1 e 100; calcular a soma de todos os números pares compreendidos entre 1 e 100; e calcular a soma de todos os números divisíveis por 3.

Algoritmo Soma
Declarar
 op, i, s ← o inteiro;

 faça
 escreva ("Digite 1 para soma de todos os números entre 1 e 100\n");
 escreva ("Digite 2 para soma de todos os números pares entre 1 e 100\n");
 escreva ("Digite 3 para soma de todos os números divisíveis por 3 entre 1 e 100\n");
 escreva ("Digite 0 para sair do programa\n");
 leia (op);
 escolha (op)
 caso 1 : início
 i ← 1;
 enquanto (i <= 100) faça
 s ← s + i;
 i ← i + 1;
 fimenquanto;
 escreva ("A somatória de 1 até 100 é " , s);
 fim;
 caso 2 : início
 i ← 1;
 faça
 se ((i mod 2) = 0) então
 s ← s + i;
 fimse;
 i ← i + 1;
 enquanto (i <= 100);
 escreva ("A somatória dos números pares de 1 até 100 é " , s);

 fim;
 caso 3 : início
 para i de 1 até 100 passo 1 faça
 se ((i mod 3) = 0) então
 s ← s + i;
 fimse;
 fimpara;
 escreva ("A somatória dos números divisíveis por 3 de 1 até 100 é " , s);
 fim;
 caso 0 : início
 escreva ("Saindo do programa!");
 fim;
 caso contrário : escreva ("opção inválida, tente novamente");
 fimescolha;
 enquanto (op <> 0);
 fim_algoritmo.

ESTRUTURA DE REPETIÇÃO DO FOR EM C

Estudo das estruturas de repetição do for simples e encadeado na Linguagem C.

Estrutura de repetição do Para no C

Em C, para a estrutura de repetição para, utiliza-se a palavra for que representa a palavra principal desta estrutura e a chave aberta, {, e chave fechada, }, para representar o conjunto de comandos que fazem parte desta estrutura. Por exemplo:

```
for (x=0 ; x<3 ; x++)  
{  
    printf ("O valor de x é: %d \n" , x);  
}
```

O resultado da execução deste exemplo é:

```
O valor de x é 0  
O valor de x é 1  
O valor de x é 2
```

Encadeamento

Em C, para a estrutura de repetição para, utiliza-se a palavra **for** que representa a palavra principal desta estrutura e a chave aberta, **{**, e chave fechada, **}**, para representar o conjunto de comandos que fazem parte desta estrutura.

Para um melhor entendimento do encadeamento nas estruturas de repetição para, observe o seguinte exemplo:

Desenvolva um algoritmo que calcula o factorial dos números inteiros de 5 até 15.

A solução deste exercício em C é a seguinte:

```
#include <stdio.h>

main()
{
    int i, num, fat;

    for (num = 5 ; num <= 15 ; num++)
    {
        fat = num;
        for (i = (num - 1) ; i > 1 ; i--)
        {
            fat = fat * i;
        } // segundo for
        printf("O valor do fatorial de %d é %d\n", num, fat);
    } // primeiro for
} // fim do main
```

Para um melhor entendimento do encadeamento nas estruturas de repetição para, observe o seguinte exemplo:

Desenvolva um algoritmo que calcula o factorial dos números inteiros de 5 até 15.

A solução deste exercício em C é a seguinte:

```
#include <stdio.h>

main()
{
    int i, num, fat;

    for (num = 5 ; num <= 15 ; num++)
    {
        fat = num;
```

```
        for (i = (num - 1) ; i > 1 ; i--)
        {
            fat = fat * i;
        } // segundo for
        printf("O valor do fatorial de %d é %d\n", num, fat);
    } // primeiro for
} // fim do main
```

Exemplo de Estrutura de Repetição do Para em C

1. Desenvolva um algoritmo que calcula a soma de todos os números compreendidos entre 1 e 100.

```
// na estrutura de repetição para
#include <stdio.h>
```

```
main()
{
    int i, s = 0;

    for ( i = 1 ; i <= 100 ; i++ )
    {
        s = s + i;
    }
    printf("A somatória de 1 até 100 é %d",
s);
} // fim do main
```

2. Desenvolva um algoritmo que calcula a soma de todos os números pares compreendidos entre 1 e 100.

```
// na estrutura de repetição para
#include <stdio.h>
```

```
main()
{
    int i, s = 0;

    for ( i = 1 ; i <= 100 ; i++ )
    {
        if ((i % 2) == 0)
        {
            s = s + i;
        } // fim do IF
    }
```

```

    printf("A somatória de 1 até 100 é %d",
s);
} // fim do main

```

3. Desenvolva um algoritmo que mostra um menu de opções para: calcular a soma de todos os números compreendidos entre 1 e 100; calcular a soma de todos os números pares compreendidos entre 1 e 100; e calcular a soma de todos os números divisíveis por 3.

```

// na estrutura de repetição para
#include <stdio.h>

main()

```

```

    // declaração de variáveis e/ou constantes
    int op, i, s;

    do
    {
        s = 0;
        // mensagem ao usuário e
        // entrada de dados
        printf("Digite 1 para soma
de todos os números entre 1 e 100 \nDigite 2 para soma de todos os números pares
entre 1 e 100 \nDigite 3 para soma de todos
os números divisíveis por 3 entre 1 e 100 \
nDigite 0 para sair do programa\n");

```

```

        scanf("%d", &op);

```

```

        switch (op)
        {
            case 1 :
                for ( i = 1 ; i <= 100 ; i++ )
                {
                    s = s + i;
                }
                printf("A somatória de 1 até 100 é %d \n", s);
                break;
            case 2 :
                for ( i = 1 ; i <= 100 ; i++ )
                {
                    if ((i % 2) == 0)
                    {
                        s = s + i;
                    } // fim do IF
                }
                printf("A somatória dos pares de 1 até 100 é %d \n", s);
                break;
            case 3 :
                for ( i = 1 ; i <= 100 ; i++ )
                {
                    if ((i % 3) == 0)
                    {
                        s = s + i;
                    } // fim do IF
                }
                printf("A somatória dos múltiplos de 3 de 1 até 100 é %d \n", s);
                break;
        }
    }
}

```

```

        case o :
            printf("Saindo do programa \n");
            break;
        default : printf ("Opção inválida \n"); // saída de resultados
    } // fim do switch
} while (op != o); // fim do do
} // fim do main

```

ESTRUTURA DE REPETIÇÃO DO FOR EM JAVA

Estudo das estruturas de repetição do for simples e encadeado na Linguagem Java.

Estrutura de Repetição do Para no Java

No Java, para a estrutura de repetição para, utiliza-se a palavra `for` que representa a palavra principal desta estrutura e a chave aberta, `{`, e chave fechada, `}`, para representar o conjunto de comandos que fazem parte desta estrutura. Por exemplo:

```

for (x=0 ; x<3 ; x++)
{
    System.out.println ("O valor de x é: "
+ x);
}

```

O resultado da execução deste exemplo é:

```

O valor de x é 0
O valor de x é 1
O valor de x é 2

```

Encadeamento

No Java, para a estrutura de repetição para, utiliza-se a palavra `for` que representa a palavra principal desta estrutura e a chave aberta, `{`, e chave fechada, `}`, para representar o conjunto de comandos que fazem parte desta estrutura.

Para um melhor entendimento do encadeamento nas estruturas de repetição para, observe o seguinte exemplo:

Desenvolva um algoritmo que calcula o fatorial dos números inteiros de 5 até 15.

```
class ExemploPara
```

```

public static void main(String args [])
{
    int i, num, fat;

    for (num = 5 ; num <= 15 ; num++)
    {
        fat = num;
        for (i = (num - 1) ; i > 1 ; i--)
        {
            fat = fat * i;
        } // segundo for
        System.out.print ("O valor do
fatorial de " + num + " é " + fat);
    } // primeiro for
} // main
} // classe

```

Exemplo de Estrutura de Repetição Enquanto, Faça/Enquanto e Para em Java

1. Desenvolva um algoritmo que calcula a soma de todos os números compreendidos entre 1 e 100.

```

// na estrutura de repetição enquanto
class Soma
{
    public static void main (String args [])
    {
        int i, s = 0;

        i = 1;
        while (i <= 100)
        {
            s = s + i;
            i = i + 1;
        } // fim do while
        System.out.println ("A somatória de
1 até 100 é " + s);
        System.exit(0);
    } // fim do void main
}

```

```

} // fim da classe

// na estrutura de repetição faça/enquanto
to
class Soma
{
    public static void main (String args [])
    {
        int i, s = 0;

        i = 1;
        do
        {
            s = s + i;
            i = i + 1;
        } while (i <= 100);
        System.out.println ("A somatória de
1 até 100 é " + s);
        System.exit(0);
    } // fim do void main
} // fim da classe

// na estrutura de repetição para
class Soma
{
    public static void main (String args [])
    {
        int i, s = 0;

        for ( i = 1 ; i <= 100 ; i++ )
        {
            s = s + i;
        }
        System.out.println ("A somatória de
1 até 100 é " + s);
        System.exit(0);
    } // fim do void main
} // fim da classe

2. Desenvolva um algoritmo que calcula a soma de todos os números pares compreendidos entre 1 e 100.
// na estrutura de repetição enquanto
class Soma
{
    public static void main (String args [])
    {
        int i, s = 0;

        i = 1;
        while (i <=
100)
        {
            if ((i % 2) == 0)
            {
                s = s + i;
            }
            i = i + 1;
        } // fim do while
        System.out.println ("A somatória dos números pares de 1 até 100 é " + s);
        System.exit(0);
    } // fim do void main
} // fim da classe

// na estrutura de repetição faça/enquanto
to
class Soma
{
    public static void main (String args [])
    {
        int i, s = 0;

        i = 1;
        do
        {
            if ((i % 2) == 0)
            {
                s = s + i;
            }
            i = i + 1;
        } // fim do se
        i = i + 1;
    } while (i <= 100);
    System.out.println ("A somatória dos números pares de 1 até 100 é " + s);
    System.exit(0);
} // fim do void main
} // fim da classe

// na estrutura de repetição para
class Soma
{
    public static void main (String args [])
    {
        int i, s = 0;

        for ( i = 1 ; i <= 100 ; i++ )
        {
            if ((i % 2) == 0)
            {
                s = s + i;
            }
        } // fim do if
    } // fim do for
}

```

```

        System.out.println ("A somatória
dos números pares de 1 até 100 é " + s);
        System.exit(0);
    } // fim do void main
} // fim da classe

```

ESTRUTURA DE VETORES

Estudo das estruturas de vetores nos algoritmos.

Uma estrutura de dados homogênea é uma estrutura capaz de armazenar várias informações de um mesmo tipo de dado. Assim, com um único nome declarado para esta estrutura, pode-se manipular várias informações.

Vet	0	1	2	3	4
	carteira1	carteira2	carteira3	carteira4	carteira5

Declaração de vetor

Quando se declara um vetor, precisa-se saber qual o tipo de dados das informações que serão armazenadas no vetor e o número de elementos desse vetor. Por exemplo,

Declarar

Vet_exemplo [100] inteiro;

notas	0	1	2	3	4	5	6	7	8	9
	9.5	10.0	8.5	5	8	7.5	6	9	8.5	7

Atribuindo valores ao vetor

Quando se atribui valores ao vetor, precisa-se saber qual o tipo de dados das informações que serão armazenadas no vetor e o número de elementos desse vetor. Por exemplo:

notas [0] ← 9.5;

Pode-se também atribuir valores no momento da declaração do vetor, por exemplo:

Declarar

notas [3] ← { 9.5 , 10.0 , 8.5 } real;

Existem dois tipos de estrutura de dados homogênea: vetores e matrizes.

Definição de vetor

Um vetor é um conjunto de informações de um mesmo tipo de dado. Note que vetor é um nome particular dado à matriz unidimensional. Por exemplo, considere uma fileira de 5 carteiras de uma sala de aula representando uma estrutura de dados e cada uma das carteiras representando partições dessa estrutura. Em outras palavras, a fileira de carteiras é um vetor e cada carteira um elemento deste vetor. Pode-se representar, graficamente, este vetor da seguinte forma:

Exemplo de vetor

Para exemplificar, graficamente, pense num vetor chamado notas, de 10 posições, contendo notas finais de 10 alunos. Como está se considerando notas, então determina-se que essas informações são do tipo real.

Mostrando os elementos de um vetor

Num vetor, os valores são mostrados para cada posição do vetor e a forma de mostrar os elementos na tela é a mesma de qualquer variável ou constante. Por exemplo:

escrever (notas [0]);

ESTRUTURA DE VETORES

Estudo das estruturas de vetores nos algoritmos.

1. Desenvolva um algoritmo que recebe 25 valores numéricos inteiros e mostra esses números.

Algoritmo Mostrar
início_algoritmo

Declarar // declaração de variáveis e/ou constantes

Vet [25] , i inteiro;

// processamento de dados
para i de 0 até 24 passo + 1 faça
 escrever (“Digite um valor inteiro”);
// mensagem ao usuário
 ler (Vet[i]); // entrada de dados
 escrever (Vet[i]); // saída de resultados
fimpara;
fim_algoritmo.

2. Desenvolva um algoritmo que recebe 100 valores numéricos inteiros e mostre a soma destes 100 números.

Algoritmo Somar
início_algoritmo

Declarar // declaração de variáveis e/ou constantes

VetSoma [100] , i , soma o inteiro;

// processamento de dados
para i de 0 até 99 passo + 1 faça
 escrever (“Digite um valor inteiro”);
// mensagem ao usuário
 ler (VetSoma[i]); // entrada de dados
 soma soma + VetSoma[i];
fimpara;
escrever (“A soma dos 100 valores digitados é: ”, soma); // saída de resultados
fim_algoritmo.

3. Desenvolva um algoritmo que recebe 50 notas bimestrais, calcule e mostre a média aritmética destas 50 notas.

Algoritmo MediaAritmetica

início_algoritmo

Declarar // declaração de variáveis e/ou constantes

i inteiro;

Notas [50] , media, soma ← 0 real;

// processamento de dados
para i de 0 até 49 passo + 1 faça
 escrever (“Digite uma nota”); // mensagem ao usuário
 ler (Notas[i]); // entrada de dados
 soma ← soma + Notas[i];
fimpara;
 media ← soma/50; // processamento de dados
 escrever (“A média das 50 notas digitadas é: ”, media); // saída de resultados
fim_algoritmo.

ESTRUTURA DE VETORES EM C

Estudo das estruturas de vetores na Linguagem C.

Declaração de vetor

A declaração de um vetor em C fica como no exemplo:

```
int vet_exemplo [100];
```

Atribuindo valores ao vetor

A atribuição de valores para cada posição do vetor em C fica como no exemplo:

```
notas [0] = 9.5;
```

Pode-se também atribuir valores no momento da declaração do vetor. Por exemplo:

```
float notas [] = { 9.5, 10.0, 8.5};
```

Observação: Quando se utiliza uma posição do vetor como parâmetro para algumas funções que atribuem valores a variáveis, como o caso da função `scanf` por exemplo, deve-se preceder o elemento do vetor com o operador unário `&`. No caso do `scanf`, a sintaxe correta para se usar um elemento do vetor `notas` como parâmetro é:

```
scanf("%f", &notas[0]);
```

Mostrando os elementos de um vetor

Para mostrar os valores de cada posição do vetor em C segue o exemplo:

```
printf("%f", notas [0]);
```

Note que a mensagem dentro do comando `printf` pode vir formada pelos elementos de formatação e por outras `Strings`. Por exemplo:

```
printf("A nota é %f", notas [0]);
```

Exemplos de um vetor em C

1. Desenvolva um algoritmo que recebe 25 valores numéricos inteiros e mostra esses números.

```
#include <stdio.h>

main()
{
    // declaração de variáveis e/ou constantes
    int Vet [25] , i;

    // processamento de dados
    for (i = 0 ; i < 25 ; i++)
    {
        // mensagem ao usuário e entrada de dados
        printf("Digite um valor inteiro: ");
        scanf("%d", &Vet[i]);
        printf("%d", Vet[i]); // saída de resultados
    } // for
} // fim do main
```

2. Desenvolva um algoritmo que recebe 100 valores numéricos inteiros e mostre a soma destes 100 números.

```
#include <stdio.h>

main()
{
    // declaração de variáveis e/ou constantes
```

```
int VetSoma [100] , i , soma = 0;

// processamento de dados
for ( i = 0 ; i <= 99 ; i++ )
{
    // mensagem ao usuário e entrada de dados
    printf("Digite um valor inteiro: ");
    scanf("%d", &VetSoma[i]);
    soma = soma + VetSoma[i];
} // for
printf("A soma dos 100 valores digitados é: %d", soma); // saída de resultados
} // fim do main
```

```
3. Desenvolva um algoritmo que recebe 50 notas bimestrais, calcule e mostre a média aritmética destas 50 notas.

#include <stdio.h>

main()
{
    // declaração de variáveis e/ou constantes
    int i;
    float Notas[50] , media , soma = 0;

    // processamento de dados
    for ( i = 0 ; i <= 49 ; i++ )
    {
        // mensagem ao usuário e entrada de dados
        printf("Digite uma nota: ");
        scanf("%f", &Notas[i]);
        soma = soma + Notas[i];
    } // for
    media = soma/50;
    printf("A média das 50 notas digitadas é: %d", media); // saída de resultados
} // fim do main
```

ESTRUTURA DE VETORES EM JAVA

Estudo das estruturas de vetores na Linguagem Java.

Declaração de vetor

A declaração de um vetor em Java segue o exemplo:

```
int Vet_exemplo [ ] = new int [100];
```

Pode-se declarar um vetor em Java e depois determinar seu tamanho, por exemplo:

```
int Vet_exemplo [ ];
Vet_exemplo = new int [100];
```

Atribuindo valores ao vetor

A atribuição de valores para cada posição do vetor em Java segue o exemplo:

```
notas [0] = 9.5;
```

Pode-se também atribuir valores no momento da declaração do vetor, por exemplo:

```
double notas [ ] = { 9.5 , 10.0 , 8.5};
```

Mostrando os elementos de um vetor

Para mostrar os valores de cada posição do vetor em Java, siga o exemplo:

```
System.out.println(notas [0]);
```

Note que a mensagem dentro do comando `System.out.println` pode vir concatenado de outras Strings. Por exemplo:

```
System.out.println("A nota é: " + notas[0]);
```

Exemplos de um vetor no Java

1. Desenvolva um algoritmo que recebe 25 valores numéricos inteiros e mostra esses números.

```
class Mostrar
{
    public static void main(String args [])
    {
        // declaração de variáveis e/ou
        // constantes
        int Vet [ ] , i;
```

```
Vet = new int [25];

// processamento de dados
for ( i = 0 ; i < 25 ; i++ )
{
    // mensagem ao usuário e en-
    // trada de dados
    Vet[i]=Integer.parseInt(JOptionPane.
    showInputDialog ("Digite um valor inteiro"));

    System.out.println (Vet[i]);///
    saída de resultados
} // for
System.exit(0);
} // void main
} // classe Mostrar
```

2. Desenvolva um algoritmo que recebe 100 valores numéricos inteiros e mostre a soma destes 100 números.

```
class Somar
{
    public static void main(String args [])
    {
        // declaração de variáveis e/ou
        // constantes
        int VetSoma [ ] , i , soma = 0;
        VetSoma = new int [100];

        // processamento de dados
        for ( i = 0 ; i <= 99 ; i++ )
        {
            // mensagem ao usuário e en-
            // trada de dados
            VetSoma[i] = Integer.
            parseInt(JOptionPane.showInputDialog
            ("Digite um valor inteiro"));

            soma = soma + VetSoma[i];
        } // for
        System.out.println ("A soma dos
        100 valores digitados é: " + soma); // saída
        de resultados
        System.exit(0);
    } // void main
} // classe Somar
```

3. Desenvolva um algoritmo que recebe 50 notas bimestrais, calcule e mostre a média aritmética destas 50 notas.

```

class MediaAritmetica
{
    public static void main(String args [])
    {
        // declaração de variáveis e/ou
        constantes
        int i;
        double Notas[ ] , media , soma = 0;
        Notas = new double [50];

        // processamento de dados
        for( i = 0 ; i <= 49 ; i++ )
        {
            // mensagem ao usuário e en-
            // traída de dados
            Notas[i] = Double.
            parseDouble(JOptionPane.showInputDialog ("Digite uma nota"));
            soma = soma + Notas[i];
        } // for
        media = soma/50;
        System.out.println ("A média das
        50 notas digitadas é: " + media); // saída de
        resultados
        System.exit(0);
    } // void main
} // classe MediaAritmetica

```

Mat	0	1	2	3	4
0	carteira1	carteira2	carteira3	carteira4	carteira5
1	carteira6	carteira7	carteira8	carteira9	carteira10
2	carteira11	carteira12	carteira13	carteira14	carteira15
3	carteira16	carteira17	carteira18	carteira19	carteira20
4	carteira21	carteira22	carteira23	carteira24	carteira25
5	carteira26	carteira27	carteira28	carteira29	carteira30
6	carteira31	carteira32	carteira33	carteira34	carteira35
7	carteira36	carteira37	carteira38	carteira39	carteira40
8	carteira41	carteira42	carteira43	carteira44	carteira45
9	carteira46	carteira47	carteira48	carteira49	carteira50

Declaração de matriz

Quando se declara uma matriz, precisa-se saber qual o tipo de dados das informações que serão armazenadas na matriz e quais informações, além do número de elementos dessa matriz, ou seja, o número de linhas e colunas. O nome atribuído para a declaração

ESTRUTURA DE MATRIZES

Estudo das estruturas de matrizes nos algoritmos.

A seguir observe como e quando trabalhar com uma matriz, bem como declarar e manipular uma matriz.

Definição de matriz

Uma matriz é um conjunto de informações de um mesmo tipo de dado, podendo conter informações diferentes. Note que matriz é um nome particular dado à matriz multidimensional, seja ela de dimensão dois ou mais.

Para um exemplo de uma matriz bidimensional, considere dez fileiras de 5 carteiras de uma sala de aula representando uma estrutura de dados, cada uma das fileiras e cada uma das carteiras representando partições dessa estrutura, em outras palavras, as dez fileiras de 5 carteiras são uma matriz bidimensional e cada carteira um elemento desta matriz. Pode-se representar, graficamente, esta matriz da seguinte forma:

da matriz segue as mesmas regras da declaração de qualquer identificador. Por exemplo,

Declarar

Matt_exemplo [100][5] inteiro;

Exemplo de matriz

Pode-se exemplificar, graficamente, uma matriz, chamada **notas**, de 10 linhas e 5 colunas, ou seja, 50 posições, contendo na primeira coluna as notas do primeiro bimestre de 10 alunos, na segunda coluna as notas do

	notas	1º Bim	2º Bim	3º Bim	4º Bim	Média Final
		0	1	2	3	4
aluno 1	0	9.0	8.0	8.5	9.0	8.5
aluno 2	1	5.0	6.0	7.0	8.0	6.5
aluno 3	2	7.5	7.5	7.5	7.5	7.5
aluno 4	3	5.0	9.0	3.0	7.0	6.0
aluno 5	4	8.0	6.0	6.0	7.0	7.0
aluno 6	5	9.0	8.0	9.0	10.0	9.0
aluno 7	6	8.0	8.0	8.0	8.0	8.0
aluno 8	7	8.5	6.0	7.5	6.0	7.0
aluno 9	8	9.5	8.5	8.5	9.5	9.0
aluno 10	9	5.0	6.5	7.0	7.5	6.5

Atribuindo valores à matriz

Quando se atribui valores à matriz, precisa-se saber qual o tipo de dados das informações que serão armazenadas na matriz e o número de linhas e colunas dessa matriz, ou seja, o número de elementos dessa matriz. Por exemplo:

```
notas [0][0] ← 9.0;
```

Pode-se também atribuir valores no momento da declaração da matriz, por exemplo:

Declarar

```
notas [2][3] ← {{ 9.0 , 10.0 , 8.5} , { 5.5 , 5.0 , 7.5}} real;
```

Mostrando os elementos de uma matriz

Os valores de uma matriz são mostrados para cada posição da matriz e a forma de mostrar os elementos na tela é a mesma de qualquer variável ou constante. Por exemplo:

```
escrever (notas [0][0]);
```

segundo bimestre de 10 alunos, na terceira coluna as notas do terceiro bimestre de 10 alunos, na quarta coluna as notas do quarto bimestre de 10 alunos e na quinta coluna as médias finais dos 10 alunos. Como está se considerando notas, então se determina que essas informações são do tipo real.

Note que a mensagem dentro do comando escrever pode vir concatenado de outros valores alfanuméricos. Por exemplo:

```
escrever ("A nota é: " , notas[0][0]);
```

Exemplos de matrizes em pseudocódigo

- Desenvolva um algoritmo que recebe 25 valores numéricos inteiros numa matriz 5x5 e mostra esses números.

Algoritmo Mostrar

 início_algoritmo

 Declarar // declaração de variáveis e/ou constantes

```
        Mat [5][5] , i , j inteiro;
```

 // processamento de dados

 para i de 0 até 4 passo + 1 faça

 para j de 0 até 4 passo + 1 faça

 escrever ("Digite um valor inteiro");

 // mensagem ao usuário

 ler (Mat[i][j]); // entrada de dados

 escrever (Mat[i][j]); // saída de resultados

 fimpara;

 fimpara;

```
fim_algoritmo.
```

2. Desenvolva um algoritmo que recebe 120 valores numéricos inteiros em uma matriz 10x12 e mostre a soma destes 120 números.

Algoritmo Somar

```
início_algoritmo
```

```
    Declarar // declaração de variáveis e/  
ou constantes
```

```
        MatSoma [10][12] , a , b ,  
        soma ← 0 inteiro;
```

```
        // processamento de dados  
        para a de 0 até 9 passo + 1 faça  
            para b de 0 até 11 passo + 1 faça  
                escrever ("Digite um valor inteiro");  
        // mensagem ao usuário  
        ler (MatSoma[a][b]); // entrada de  
dados  
        soma ← soma + MatSoma[a][b];  
        fimpara;  
        fimpara;  
        escrever ("A soma dos 120 valores digi-  
tados é: " , soma); // saída de resultados  
    fim_algoritmo.
```

```
double notas [2][3] = {{ 9.0 , 10.0 , 8.5} , {  
5.5 , 5.0 , 7.5}};
```

Observação: Quando se utiliza uma posição de uma matriz como parâmetro para algumas funções que atribuem valores a variáveis, como o caso da função `scanf` por exemplo, deve-se preceder o elemento da matriz com o operador unário `&`. No caso do `scanf`, a sintaxe correta para se usar um elemento do vetor `notas` como parâmetro é:

```
scanf("%f", &notas[0][0]);
```

Mostrando os elementos de uma matriz

Para mostrar os valores de cada posição da matriz em C segue o exemplo:

```
printf("%f", notas [0][0]);
```

Note que a mensagem dentro do comando `printf` pode vir formada pelos elementos de formatação e por outras Strings. Por exemplo:

```
System.out.println("A nota é: " + notas[0]  
[0]);
```

Exemplos de matrizes em C

1. Desenvolva um algoritmo que recebe 25 valores numéricos inteiros numa matriz 5x5 e mostre esses números.

```
#include <stdio.h>
```

```
main()  
{  
    // declaração de variáveis e/ou cons-  
stantes  
    int Mat [5][5] , i , j ;  
  
    // processamento de dados  
    for ( i = 0 ; i < 5 ; i++ )  
    {  
        for ( j = 0 ; j < 5 ; j++ )
```

ESTRUTURA DE MATRIZES EM C

Estudo das estruturas de matrizes na Linguagem C.

Declaração de matriz

A declaração de uma matriz em C segue o exemplo:

```
int Mat_exemplo [100][5];
```

Atribuindo valores à matriz

A atribuição de valores para cada posição da matriz em C segue o exemplo:

```
notas [0][0] = 9.0;
```

Pode-se também atribuir valores no momento da declaração da matriz, por exemplo:

```

    {
        // mensagem ao usuário e
        // entrada de dados
        printf("Digite um valor inteiro: ");
        scanf("%d", &Mat[i][j]);
        printf("%d", Mat[i][j]); // saída
        de resultados
        } // for usando a variável j
    } // for usando a variável i
} // fim do main
2. Desenvolva um algoritmo que recebe 120
valores numéricos inteiros numa matriz
10x12 e mostre a soma destes 120 números.

#include <stdio.h>

int FuncSoma()
{
    // declaração de variáveis e/ou constantes
    int MatSoma[10][12], a, b, soma = 0;

    // processamento de dados
    for (a = 0; a <= 9; a++)
    {
        for (b = 0; b <= 11; b++)
        {
            // mensagem ao usuário e entrada
            // de dados
            printf("Digite um valor inteiro: ");
            scanf("%d", &MatSoma[a][b]);
            soma = soma + MatSoma[a][b];
        } // for usando a variável b
    } // for usando a variável a
    return soma; // saída de resultados
} // FuncSoma

main()
{
    // declaração de variáveis e/ou constantes
    int s = 0;

    // processamento de dados
    s = FuncSoma();
    printf("A soma dos 120 valores digitados é: %d", s); // saída de resultados
}

```

} // fim do main

ESTRUTURA DE MATRIZES EM JAVA

Estudo das estruturas de matrizes na Linguagem Java.

Declaração de matriz

A declaração de uma matriz em Java terá como o exemplo:

```
int Mat_exemplo[] = new int [100][5];
```

Pode-se declarar uma matriz em Java e depois determinar seu tamanho, por exemplo:

```
int Mat_exemplo[];
Mat_exemplo = new int [100][5];
```

Atribuindo valores à matriz

A atribuição de valores para cada posição da matriz em Java segue o exemplo:

```
notas[0][0] = 9.0;
```

Pode-se também atribuir valores no momento da declaração da matriz, conforme o exemplo:

```
double notas[][] = {{9.0, 10.0, 8.5}, {5.5, 5.0, 7.5}};
```

Mostrando os elementos de uma matriz

Para mostrar os valores de cada posição da matriz em Java segue o exemplo:

```
System.out.println(notas[0][0]);
```

Note que a mensagem dentro do comando `System.out.println` pode vir concatenado de outras Strings. Por exemplo:

```
System.out.println("A nota é: " + notas[0][0]);
```

Exemplos de matrizes em Java

- Desenvolva um algoritmo que recebe 25 valores numéricos inteiros numa matriz 5x5 e mostre esses números.

```
class Mostrar
{
    public static void main(String args [])
    {
        // declaração de variáveis e/ou
        constantes
        int Mat [ ][ ] , i , j ;
        Mat = new int [5][5];

        // processamento de dados
        for ( i = 0 ; i < 5 ; i++ )
        {
            for ( j = 0 ; j < 5 ; j++ )
            {
                // mensagem ao usuário e
                // entrada de dados
                Mat[i][j] = Integer.
                parseInt(JOptionPane.showInputDialog
                ("Digite um valor inteiro"));

                System.out.println (Mat[i]
                [j]); // saída de resultados
            } // for usando a variável j
        } // for usando a variável i
        System.exit(0);
    } // void main
} // classe Mostrar
```

- Desenvolva um algoritmo que recebe 120 valores numéricos inteiros numa matriz 10x12 e mostre a soma destes 120 números.

```
class Somar
{

    public static int FuncSoma( )
    {
        // declaração de variáveis e/ou
        constantes
        int MatSoma [ ][ ] , a , b , soma = 0;
        MatSoma = new int [10][12];

        // processamento de dados
        for ( a = 0 ; a <= 9 ; a++ )
        {
            for ( b = 0 ; b <= 11 ; b++ )
```

```
    }
    // mensagem ao usuário e
    // entrada de dados
```

```
    MatSoma[a][b] = Integer.
    parseInt(JOptionPane.showInputDialog
    ("Digite um valor inteiro"));

    soma = soma + MatSoma[a][b];
} // for usando a variável b
} // for usando a variável a
return soma; // saída de resultados
} // FuncSoma
```

```
public static void main(String args [])
{
    // declaração de variáveis e/ou
    constantes
    int s = 0;
```

```
    // processamento de dados
    s = FuncSoma( );
    System.out.println ("A soma dos
    120 valores digitados é: " + s); // saída de re-
    sultados
    System.exit(0);
} // void main
} // classe Somar
```

CONCEITOS DE MODULARIZAÇÃO

Estudo dos conceitos introdutórios de modularização nos algoritmos.

Necessidade da modularização em programas

Um dos pontos principais para o uso da modularização é a divisão do algoritmo em módulos para que ele seja melhor interpretado e desenvolvido. A divisão em módulos facilita o entendimento parcial do problema e entendendo todos os problemas parciais, no final, tem-se entendido o problema maior.

Outro ponto principal para o uso de modularização é a reutilização de código. Neste caso, basta desenvolver um módulo com esse pedaço do algoritmo e sempre que precisa-se dele no algoritmo, basta fazer a chamada do módulo, evitando reescrever o mesmo código.

Construção de módulos

Para construir os módulos, primeiro precisa-se analisar o problema e dividi-lo em partes principais, que são os módulos. Por fim, precisa-se analisar todos os módulos e o problema geral para garantir que o algoritmo mais os módulos resolvem o problema de forma simples.

Por exemplo, suponha que se quer desenvolver um algoritmo que recebe dois valores numéricos e uma opção do usuário que decide qual entre as quatro operações básicas da matemática (soma, subtração, divisão e multiplicação) ele quer resolver.

Analizando este problema, pode-se dividí-lo em subproblemas, por exemplo, um menu de opções para que o usuário possa escolher uma operação matemática, o cálculo da soma, o cálculo da subtração, o cálculo da divisão e o cálculo da multiplicação. Tem-se assim cinco módulos para resolver o nosso problema.

Escopo de variáveis

O escopo de uma variável é onde, dentro do algoritmo, uma variável é válida ou pode ser reconhecida. Uma variável pode ter o escopo dividido em dois tipos: variável global ou variável local.

Uma variável global é aquela declarada no início do algoritmo e pode ser utilizada por qualquer parte deste algoritmo, seja nos comandos do próprio algoritmo bem como dentro de qualquer módulo que pertença ao algoritmo. Neste caso, sua declaração é feita apenas uma única vez, não sendo permitido que o mesmo nome de variável seja declarado dentro de qualquer outra parte do algoritmo, por exemplo, dentro de qualquer outro módulo.

Uma variável local é aquela declarada dentro de algum bloco, por exemplo, dentro de um módulo. Neste caso, esta variável é válida e reconhecida somente dentro do bloco em que foi declarada. Assim, o mesmo nome de variável pode ser declarado dentro de diferentes blocos, pois serão reconhecidas como uma nova variável.

PROCEDIMENTOS E FUNÇÕES

Estudo dos procedimentos e funções nos algoritmos.

Módulos Procedimento

Os módulos procedimento são os módulos que não retornam valor para o módulo ou algoritmo que os tenham chamado. Neste sentido, esse tipo de módulo é utilizado quando o algoritmo principal ou módulo que chama o módulo procedimento não necessita de retorno de qualquer dado do módulo.

Nos módulos procedimento, utiliza-se as palavras `início_algoritmo` e `fimmódulo` que representam as palavras principais deste módulo, ou seja, o início e o fim do módulo, respectivamente.

Por exemplo, suponha que se quer desenvolver um módulo que imprime o nome da disciplina e o nome da instituição de ensino:

```
Dados ()  
    início  
        Declarar  
            disc, fac alfanumérico;  
  
            disc ← “Lógica de Programação”;  
            fac ← “IBTA”;  
            escreva (“nome da disciplina => ” ,  
            disc);  
            escreva (“nome da instituição de ensino  
=> ” , fac);  
        fimmódulo;
```

Módulos Função

Os módulos função são os módulos que retornam algum valor para o módulo ou algoritmo que os tenham chamado. Neste sentido, esse tipo de módulo é utilizado quando o algoritmo principal ou módulo que chama o módulo função necessita de retorno de qualquer dado do módulo.

Nos módulos função, utilizam-se as palavras `início` e `fimmódulo` que representam as palavras principais deste módulo, que representam o início e o fim do módulo, respectivamente. Além dessas palavras, utiliza-se a palavra `retornar` que é a principal palavra do

módulo função, pois é a partir deste comando que um valor é retornado para o algoritmo ou módulo que o chamou.

Note que os comandos dentro do módulo são comuns àqueles de qualquer algoritmo, ou seja, com entrada de dados, processamento desses dados e saída de resultados. Note ainda que o valor de retorno enviado pelo comando retornar precisa ser do mesmo tipo de dado declarado para o módulo função.

Por exemplo, suponha que se quer desenvolver um módulo que imprime o nome da disciplina e o nome da instituição de ensino:

```
alfanumérico Dados ()  
    início  
        Declarar  
            disc, fac, mens alfanumérico;  
  
            disc ← “Lógica de Programação”;  
            fac ← “IBTA”;  
            mens ← “nome da disciplina => “ +  
            disc + “\n nome da instituição de ensino =>”  
            + fac;  
            retornar mens;  
        fimmódulo;
```

PROCEDIMENTOS E FUNÇÕES PARAMETRIZADOS

Estudo dos procedimentos e funções parametrizados nos algoritmos.

Parametrização de módulos

Cada módulo interage com o módulo principal ou mesmo com outros módulos e, muitas vezes, quando os módulos são chamados há a necessidade de enviar dados para que esses módulos possam resolver o seu subproblema. Esses dados que são enviados para os módulos são chamados de argumentos que são passados como parâmetros aos módulos.

É dentro dos parênteses que os argumentos são passados como parâmetros aos módulos.

Quando se passa argumentos como parâmetros, precisa-se identificar qual o tipo de

dado será passado como parâmetro. Neste caso, cada argumento deverá ser declarado no momento que o módulo for desenvolvido.

Nos módulos procedimento, utilizarem-se as palavras **início** e **fimmódulo** que representam as palavras principais deste módulo, ou seja, o início e o fim do módulo, respectivamente.

Por exemplo, suponha que se quer desenvolver um módulo que imprime o nome da disciplina e o nome da instituição de ensino:

```
Dados (alfanumérico disc , alfanumérico  
fac)  
    início  
        escreva («nome da disciplina => « ,  
        disc);  
        escreva («nome da instituição de ensi-  
        no =>» , fac);  
    fimmódulo;
```

Chamada dos módulos

Para o módulo procedimento, basta fazer a chamada deste módulo pelo nome do módulo, passando os respectivos dados como argumento se for necessário.

Para o módulo função, a chamada deste módulo precisa estar vinculada a uma variável, ou seja, o retorno de dado da chamada do módulo função deve ser atribuído a uma variável cujo tipo de dado deve ser o mesmo que o dado que está sendo retornado. Neste caso, a passagem de dados como argumentos, se for necessário, não se difere do módulo procedimento.

Módulo procedimento sem parametrização

A chamada dos módulos procedimento sem parametrização em pseudocódigo segue a seguinte regra sintática:

```
<nome do módulo> ();
```

Por exemplo, a chamada do módulo que imprime o nome da disciplina e o nome da instituição de ensino desenvolvido anteriormente:

Dados ();

Módulo função sem parametrização

A chamada dos módulos procedimento sem parametrização em pseudocódigo segue o seguinte exemplo:

A chamada do módulo que imprime o nome da disciplina e o nome da instituição de ensino desenvolvido anteriormente:

```
alfanumérico m;  
m ← Dados ( );
```

Módulo procedimento com parametrização

A chamada dos módulos procedimentos com parametrização em pseudocódigo segue o exemplo:

A chamada do módulo que imprime o nome da disciplina e o nome da instituição de ensino desenvolvido anteriormente:

```
Dados ("Lógica de Programação", "Técnico");
```

Módulo função com parametrização

A chamada dos módulos função com parametrização em pseudocódigo segue o seguinte exemplo:

A chamada do módulo que imprime o nome da disciplina e o nome da instituição de ensino desenvolvido anteriormente:

```
alfanumérico m;  
m ← Dados ("Lógica de Programação",  
"Técnico");
```

EXEMPLOS DE MODULARIZAÇÃO EM ALGORITMOS

Exemplos de funções e procedimentos em pseudocódigo.

1. Desenvolva um algoritmo que recebe dois valores inteiro e o símbolo da operação

conforme tabela abaixo, calcula e mostra a operação efetuada:

Símbolo da operação	Nome da operação
+	adição
-	subtração
*	multiplicação
/	divisão

```
// módulo função Mais que recebe dois valores inteiros e retorna um valor inteiro  
inteiro Mais (n1 inteiro , n2 inteiro)
```

```
início
```

```
Declarar // declaração de variáveis que só podem ser usados dentro deste módulo  
res inteiro;
```

```
res n1 + n2; // processamento de dados
```

```
retornar res; // retorno do módulo  
fimMódulo; // Mais
```

```
// módulo procedimento Menos que recebe dois valores inteiros
```

```
Menos (n1 inteiro, n2 inteiro)
```

```
início
```

```
Declarar // declaração de variáveis que só podem ser usados dentro deste módulo  
res inteiro;
```

```
res n1 - n2; // processamento de dados
```

```
escrever ("A diferença de " , n1 , " com "  
, n2 , " é " , res); // saída de resultados
```

```
fimMódulo; // Menos
```

```
// módulo procedimento Vezes que recebe dois valores inteiros
```

```
Vezes (n1 inteiro, n2 inteiro)
```

```
início
```

```
Declarar // declaração de variáveis que só podem ser usados dentro deste módulo  
res inteiro;
```

```
res n1 * n2; // processamento de dados
```

```
escrever ("O produto de " , n1 , " com "  
, n2 , " é " , res); // saída de resultados
```

```
fimMódulo; // Vezes
```

```

// módulo função Dividido que recebe dois valores inteiros e retorna um valor real
real Dividido (n1 inteiro, n2 inteiro)
início
    Declarar // declaração de variáveis que só podem ser usados dentro deste módulo
    res real;
    res n1 / n2; // processamento de dados
    retornar res; // retorno do módulo
fimMódulo; // Dividido

// módulo principal
Algoritmo Calculo
início_algoritmo
    Declarar // declaração de variáveis e/ou constantes que podem ser usados no algoritmo
Calculo
    div real;
    soma, num1, num2 inteiro;
    oper alfanumérico;

    escrever ("Digite dois inteiros e a operação"); // mensagem ao usuário
    ler (num1, num2, oper); // entrada de dados
    // processamento de dados
    escolha (oper)
        caso '+' : /* chamada do módulo função Mais, passando num1 e num2 como parâmetro
e atribuindo o retorno do módulo na variável soma */
            soma ← Mais(num1 , num2);
            escrever ("A soma de " , num1 , " com " , num2 , " é " , soma); // saída de re-
sultados
        caso '-' : /* chamada domódulo procedimento Menos, passando num1 e num2 como
parâmetro*/
            Menos (num1 , num2);
        caso '*' : /* chamada do módulo procedimento vezes, passando num1 e num2 como
parâmetro */
            Vezes (num1 , num2);
        caso '/' : /* chamada do módulo função Dividido, passando num1 e num2 como parâ-
metro e atribuindo o retorno do módulo na variável div */
            div Dividido (num1 , num2);
            escrever ("A divisão de " , num1 , " com " , num2 , " é " , div); // saída de resultados
        caso contrário : escrever ("operação inválida"); // saída de resultados
    fimEscolha;
fim_algoritmo.

```

2. Desenvolva um algoritmo que mostre um menu de opções para: calcular a soma de todos os números compreendidos entre 1 e 100; calcular a soma de todos os números

pares compreendidos entre 1 e 100; e calcular a soma de todos os números divisíveis por 3.

```

// módulo procedimento Menu
Menu()
início

```

```

escreva ("Digite 1 para soma de todos os números entre 1 e 100\n");
escreva ("Digite 2 para soma de todos os números pares entre 1 e 100\n");
escreva ("Digite 3 para soma de todos os números divisíveis por 3 entre 1 e 100\n");
escreva ("Digite 0 para sair do programa\n");
fimMódulo; // Menu

// módulo função SomaTudo que retorna um valor inteiro
íntero SomaTudo ()
início
    Declarar // declaração de variáveis que só podem ser usados dentro deste módulo
        s→o, i íntero;

        i → 1;
        // processamento de dados
        enquanto (i <= 100) faça
            s → s + i;
            i → i + 1;
        fimEnquanto;
        retornar s; // retorno do módulo
fimMódulo; // Soma Tudo

// módulo procedimento SomaPares que recebe um valor inteiro
SomaPares (par íntero)
início
    Declarar // declaração de variáveis que só podem ser usados dentro deste módulo
        s→o, i íntero;

        i → 1;
        // processamento de dados
        faça
            se ((i mod par) = 0)
                então
                    s → s + i;
            fimSe;
            i → i + 1;
        enquanto (i <= 100);
        escreva ("A somatória dos números pares de 1 até 100 é " , s); // saída de resultados
fimMódulo; // SomaPares

// módulo função SomaDivisivel3 que recebe um valor inteiro e retorna um valor inteiro
íntero SomaDivisivel3 (d íntero)
início
    Declarar // declaração de variáveis que só podem ser usados dentro deste módulo
        i, s → o íntero;

        // processamento de dados
        para i de 1 até 100 passo +1 faça
            se ((i mod d) = 0)
                então
                    s → s + i;
            fimSe;

```

```

fimpara;
    retornar s; // retorno da função
fimMódulo; // SomaDivisivel3

// módulo principal
Algoritmo Somador
    Declarar // declaração de variáveis que só podem ser usados dentro do algoritmo Somador
        op, soma, div inteiro;

    // processamento de dados
    faça
        /* chamada do módulo procedimento Menu, sem parametrização */
        Menu();
        leia (op); // entrada de dados
        escolha (op)
            caso 1 : /* chamada do módulo função SomaTudo, sem parâmetrização e atribuindo o
            retorno do módulo na variável soma */
                soma SomaTudo();
                escreva ("A somatória de 1 até 100 é " , soma); // saída de resultados
            caso 2 : /* chamada do módulo procedimento SomaPares, passando o número 2 como
            parâmetro*/
                SomaPares(2);
            caso 3 : /* chamada do módulo função SomaDivisivel3, passando o número 3 como pa-
            râmetro e atribuindo o retorno do módulo na variável div */
                div SomaDivisivel3(3);
                // saída de resultados
                escreva ("A somatória dos números divisíveis por 3 de 1 até 100 é " , div);
            caso o : escreva ("Saindo do programa!");
            caso contrário : escreva ("opção inválida, tente novamente");
        fimEscolha;
        enquanto (op <> o);
    fim_algoritmo.

```

MODULARIZAÇÃO EM C

Estudos de funções e procedimentos em C.

Módulos Procedimento

Nos módulos procedimento em C, utiliza-se a palavra `void` que representa a palavra principal desta estrutura e a chave aberta, `{`, e chave fechada, `}`, para representar o conjunto de comandos que fazem parte deste módulo.

Por exemplo, suponha que se quer desenvolver um módulo que peça ao usuário que informe dois inteiros e imprima a soma dos dois:

```
void Soma ()
```

```
{
    int a, b, soma;

    printf("Digite um inteiro: ");
    scanf("%d", &a);
    printf("Digite um inteiro: ");
    scanf("%d", &b);

    soma = a + b;

    printf("A soma é: %d", soma);
}
```

Perceba o uso da palavra `void`, ela é utilizada no C para identificar um módulo procedimento.

Módulos Função

Nos módulos função em C, utiliza-se a palavra **return** que representa a palavra principal desta estrutura e a chave aberta, {, e chave fechada, }, para representar o conjunto de comandos que fazem parte deste módulo. É a partir do comando **return** que um valor é retornado para o programa ou função que o chamou.

Por exemplo, suponha que se quer desenvolver um módulo que peça ao usuário que informe dois inteiros e devolva a soma dos dois:

```
int Soma ()  
{  
    int a, b, soma;  
  
    printf("Digite um inteiro: ");  
    scanf("%d", &a);  
    printf("Digite um inteiro: ");  
    scanf("%d", &b);  
  
    soma = a + b;  
  
    return soma;  
}
```

Parametrização de módulos

Os argumentos num programa C são passados como parâmetros. Por exemplo, suponha que se quer desenvolver um módulo que peça ao usuário que informe dois inteiros e devolva a soma dos dois:

```
int Soma (int a, int b)  
{  
    int soma;  
  
    soma = a + b;  
  
    return soma;  
}
```

Chamada dos módulos

Módulo procedimento sem parametrização em C

A chamada dos módulos procedimento sem parametrização em C segue o exemplo:

A chamada do módulo que solicita ao usuário dois inteiros e imprime sua soma desenvolvido anteriormente:

```
Soma();
```

Módulo procedimento com parametrização em C

A chamada dos módulos procedimento com parametrização em C segue o exemplo:

A chamada do módulo que solicita ao usuário dois inteiros e imprime sua soma desenvolvido anteriormente:

```
Soma(4, 9);
```

Note que a quantidade de argumentos que foi passada como parâmetro é a mesma do módulo quando ele foi declarado. No nosso exemplo, a quantidade de argumentos é dois.

Módulo função sem parametrização em C

A chamada dos módulos função sem parametrização em C segue o exemplo:

A chamada do módulo que solicita ao usuário dois inteiros e imprime sua soma desenvolvido anteriormente:

```
int s;  
s = Soma();
```

Note que o tipo de retorno do módulo função **Soma** é **int** e a variável **s** que recebe o valor de retorno da função também é do tipo **int**, ou seja, do mesmo tipo de dados. Isso é obrigatório.

Módulo função com parametrização em C

A chamada dos módulos função com parametrização em C segue o exemplo:

A chamada do módulo que solicita ao usuário dois inteiros e imprime sua soma desenvolvido anteriormente:

```
int s;
```

s = Dados (4 , 9);

Note que a quantidade de argumentos que foi passada como parâmetro é a mesma do módulo quando ele foi declarado. No nosso exemplo, a quantidade de argumentos é dois.

EXEMPLOS DE MODULARIZAÇÃO EM C

Estudo das funções nos algoritmos.

1. Desenvolva um algoritmo que recebe dois valores inteiro e o símbolo da operação conforme tabela da página seguinte, calcula e mostra a operação efetuada:

Símbolo da operação	Nome da operação
+	adição
-	subtração
*	multiplicação
/	divisão

```
#include <stdio.h>

// módulo função Mais que recebe dois valores inteiros e retorna um valor inteiro
int Mais (int n1 , int n2)
{
    // declaração de variáveis que só podem ser usados dentro deste módulo
    int res;

    res = n1 + n2; // processamento de dados
    return res; // retorno do módulo
} // fim do módulo Mais

// módulo procedimento Menos que recebe dois valores inteiros
void Menos (int n1 , int n2)
{
    // declaração de variáveis que só podem ser usados dentro deste módulo
    int res;

    res = n1 - n2; // processamento de dados
    printf("A diferença de %d com %d é %d", n1, n2, res); // saída de resultados
```

} // fim do módulo Menos

```
// módulo procedimento Vezes que recebe dois valores inteiros
```

```
void Vezes (int n1 , int n2)
```

```
{
```

```
// declaração de variáveis que só podem ser usados dentro deste módulo
```

```
int res;
```

```
res = n1 * n2; // processamento de dados
```

```
printf("O produto de %d com %d é %d", n1, n2, res); // saída de resultados
```

```
} // fim do módulo Vezes
```

```
// módulo função Dividido que recebe dois valores inteiros e retorna um valor real
```

```
double Dividido (int n1 , int n2)
```

```
{
```

```
// declaração de variáveis que só podem ser usados dentro deste módulo
```

```
float res;
```

```
res = (float)n1 / n2; // processamento de dados
```

```
return res; // retorno do módulo
```

```
} // fimd o módulo Dividido
```

```
main()
```

```
{
```

```
// declaração de variáveis e/ou constantes que podem ser usados no algoritmo Calculo
```

```
float div ;
```

```
int oper, soma, num1, num2;
```

```
// mensagem ao usuário e entrada de dados
```

```
printf("Digite um número inteiro: ");
```

```
scanf("%d", &num1);
```

```
printf("Digite um número inteiro: ");
```

```
scanf("%d", &num2);
```

```
printf("Digite a operação: \n1 para somar \n2 para subtrair \n3 para multiplicar \n4 para dividir: ");
```

```

scanf("%d", &oper);

// processamento de dados
switch (oper)
{
    case 1: /* chamada do módulo função Mais, passando num1 e num2 como parâmetro
e atribuindo o retorno do módulo na variável soma */
        soma = Mais(num1 , num2);
        // saída de resultados
        printf("A soma de %d com %d é %d", num1, num2, soma);
        break;
    case 2 : /* chamada domódulo procedimento Menos, passando num1 e num2 como
parâmetro*/
        Menos (num1 , num2);
        break;
    case 3 : /* chamada do módulo procedimento vezes, passando num1 e num2 como
parâmetro */
        Vezes (num1 , num2);
        break;
    case 4 : /* chamada do módulo função Dividido, passando num1 e num2 como parâ-
metro e atribuindo o retorno do módulo na variável div */
        div = Dividido (num1 , num2);
        // saída de resultados
        printf("A divissão de %d com %d é %f", num1, num2, div);
        break;
    default : printf("operação inválida"); // saída de resultados
}
} // fim do switch
} // fim do main

```

2. Desenvolva um algoritmo que mostre um menu de opções para: calcular a soma de todos os números compreendidos entre 1 e 100; calcular a soma de todos os números pares compreendidos entre 1 e 100; e calcular a soma de todos os números divisíveis por 3.

```

// na estrutura de repetição para
#include <stdio.h>

int Menu()
{
    int op;

    // mensagem ao usuário e entrada de dados
    printf("Digite 1 para soma de todos os números entre 1 e 100 \nDigite 2 para
soma de todos os números pares entre 1 e 100 \nDigite 3 para soma de todos os números divi-
síveis por 3 entre 1 e 100 \nDigite 0 para sair do programa\n");

    scanf("%d", &op);
    return op;
} // fim do módulo Menu

// módulo função SomaTudo que retorna um valor inteiro

```

```

int SomaTudo ()
{
    // declaração de variáveis que só podem ser usados dentro deste módulo
    int s=0, i;

    i = 1;
    // processamento de dados
    while (i <= 100)
    {
        s = s + i;
        i = i + 1;
    } // fim do while
    return s; // retorno do módulo
} // fim do módulo SomaTudo

// módulo procedimento SomaPares que recebe um valor inteiro
void SomaPares (int par)
{
    // declaração de variáveis que só podem ser usados dentro deste módulo
    int s=0, i;

    i = 1;
    // processamento de dados
    do
    {
        if ((i % par) == 0)
        {
            s = s + i;
        } // fim do if
        i = i + 1;
    } while (i <= 100);
    printf("A somatória dos números pares de 1 até 100 é %d \n", s); // saída de resultados
} // fim do móduloSomaPares

// módulo função SomaDivisivel3 que recebe um valor inteiro e retorna um valor inteiro
int SomaDivisivel3 (int d)
{
    // declaração de variáveis que só podem ser usados dentro deste módulo
    int i, s = 0;

    // processamento de dados
    for ( i = 1 ; i <= 100 ; i++ )
    {
        if ((i % d) == 0)
        {
            s = s + i;
        } // fim do if
    } // fim do for
    return s; // retorno da função
}// fim do módulo SomaDivisivel3

```

```

// módulo principal
main()
{
    // declaração de variáveis e/ou constantes
    int op, soma, div;

    do
    {
        /* chamada do módulo procedimento Menu, sem parametrização */
        op = Menu();

        switch (op)
        {
            case 1 :
                /* chamada do módulo função SomaTudo, sem parâmetrização e atribuindo o retorno do módulo na variável soma */
                soma = SomaTudo();
                printf("A somatória de 1 até 100 é %d \n", soma);
                break;
            case 2 :
                /* chamada do módulo procedimento SomaPares, passando o número 2 como parâmetro*/
                SomaPares(2);
                break;
            case 3 :
                /* chamada do módulo função SomaDivisivel3, passando o número 3 como parâmetro e atribuindo o retorno do módulo na variável div */
                div = SomaDivisivel3(3);
                // saída de resultados
                printf("A somatória dos múltiplos de 3 de 1 até 100 é %d \n", div);

                break;
            case 0 :
                printf("Saindo do programa \n");
                break;
            default : printf ("opção inválida, tente novamente \n"); // saída de resultados
        } // fim do switch
    } while (op != 0); // fim do do
} // fim do main

```

MODULARIZAÇÃO EM JAVA

Estudo das funções e Procedimentos na Linguagem Java.

Módulos Procedimento

Nos módulos procedimento em Java, utiliza-se a palavra **void** que representa a palavra

principal desta estrutura e a chave aberta, {, e chave fechada, }, para representar o conjunto de comandos que fazem parte deste módulo.

Por exemplo, suponha que se quer desenvolver um módulo que imprime o nome da disciplina e o nome da instituição de ensino:

```

void Dados ( )
{

```

```

String disc, fac;

disc = "Lógica de Programação";
fac = "Técnico";
System.out.println ("nome da disciplina => " + disc);
System.out.println ("nome da instituição de ensino =>" + fac);
}

```

Perceba o uso da palavra `void`, ela é utilizada no Java para identificar um módulo procedimento.

Módulos Função

Os comandos dentro do módulo são comuns àqueles de qualquer programa e que o valor de retorno enviado pelo comando `return` precisa ser do mesmo tipo de dado declarado para o módulo função.

Por exemplo, suponha que se quer desenvolver um módulo que imprime o nome da disciplina e o nome da instituição de ensino:

```

String Dados ()
{
    String disc, fac, mens;

    disc = "Lógica de Programação";
    fac = "Técnico";
    mens = "nome da disciplina => " +
    disc + "\nnome da instituição de ensino =>" +
    fac;
    return mens;
}

String mens;

mens = "nome da disciplina => " +
disc + "\nnome da instituição de ensino =>" +
fac;
retornar mens;
}

```

Note que o tipo de retorno do módulo função `Dados` é `String` e o retorno do módulo é um `String mens`, ou seja, do mesmo tipo de dados. Isso é obrigatório.

Parametrização de módulos em Java

A quantidade de argumentos que pode ser passada como parâmetro não é determinada, pode ser um único argumento ou uma quantidade finita de argumentos.

Por exemplo, suponha que se quer desenvolver um módulo que imprime o nome da disciplina e o nome da instituição de ensino:

```

String Dados (String disc , String fac)
{

```

Chamada dos módulos

Módulo procedimento sem parametrização em Java

A chamada dos módulos procedimento sem parametrização em Java segue o exemplo:

A chamada do módulo que imprime o nome da disciplina e o nome da instituição de ensino desenvolvido anteriormente:

`Dados ();`
Módulo procedimento com parametrização em Java

A chamada dos módulos procedimento com parametrização em Java segue o exemplo:

A chamada do módulo que solicita ao usuário dois inteiros e imprime sua soma desenvolvido anteriormente:

`Dados("Algoritmos", "Técnico");`

Módulo função sem parametrização em Java

A chamada dos módulos função sem parametrização em Java segue o exemplo:

A chamada do módulo que imprime o nome da disciplina e o nome da instituição de ensino desenvolvido anteriormente:

```
String m;  
m = Dados();
```

Módulo função com parametrização em Java

A chamada dos módulos função com parametrização em Java segue o exemplo:

A chamada do módulo que imprime o nome da disciplina e o nome da instituição de ensino desenvolvido anteriormente:

```
String m;  
m = Dados("Lógica de Programação",  
"Técnico");
```

EXEMPLOS DE MODULARIZAÇÃO EM JAVA

Estudo das funções e procedimentos na Linguagem Java.

1. Desenvolva um algoritmo que recebe dois valores inteiro e o símbolo da operação conforme tabela abaixo, calcula e mostra a operação efetuada:

Símbolo da operação	Nome da operação
+	adição
-	subtração
*	multiplicação
/	divisão

```
class Calculo {  
  
    // módulo função Mais que recebe dois  
    // valores inteiros e retorna um valor inteiro  
    static int Mais (int n1 , int n2)  
    {  
        // declaração de variáveis que só podem  
        // ser usados dentro deste módulo  
        int res;  
  
        res = n1 + n2; // processamento de
```

dados

```
        return res; // retorno do módulo  
    } // fim do módulo Mais
```

```
    // módulo procedimento Menos que recebe  
    // dois valores inteiros
```

```
    static void Menos (int n1 , int n2)  
    {  
        // declaração de variáveis que só podem  
        // ser usados dentro deste módulo  
        int res;
```

```
        res = n1 - n2; // processamento de  
        dados
```

```
        System.out.println ("A diferença de "  
        + n1 + " com " + n2 + " é " + res); // saída de  
        resultados
```

```
    } // fim do módulo Menos
```

```
    // módulo procedimento Vezes que recebe  
    // dois valores inteiros
```

```
    static void Vezes (int n1 , int n2)  
    {  
        // declaração de variáveis que só podem  
        // ser usados dentro deste módulo  
        int res;
```

```
        res = n1 * n2; // processamento de  
        dados
```

```
        System.out.println ("O produto de "  
        + n1 + " com " + n2 + " é " + res); // saída de  
        resultados
```

```
    } // fim do módulo Vezes
```

```
    // módulo função Dividido que recebe  
    // dois valores inteiros e retorna um valor real
```

```
    static double Dividido (int n1 , int n2)  
    {  
        // declaração de variáveis que só podem  
        // ser usados dentro deste módulo  
        double res;
```

```
        res = n1 / n2; // processamento de  
        dados
```

```
        return res; // retorno do módulo  
    } // fimd o módulo Dividido
```

```
    // módulo principal  
    public static void main (String args [ ])  
    {
```

```

// declaração de variáveis e/ou constantes que podem ser usados no algoritmo Calculo
double div ;
int oper, soma, num1, num2;

// mensagem ao usuário e entrada de dados
num1 = Integer.parseInt(JOptionPane.showInputDialog("Digite um número inteiro"));
num2 = Integer.parseInt(JOptionPane.showInputDialog("Digite um número inteiro"));
oper = Integer.parseInt(JOptionPane.showInputDialog("Digite a operação: \n1 para somar
\n2 para subtrair \n3 para multiplicar \n4 para dividir"));

// processamento de dados
switch (oper)
{
    case 1 :/* chamada do módulo função Mais, passando num1 e num2 como parâmetro
e atribuindo o retorno do módulo na variável soma */
        soma = Mais(num1 , num2);
        // saída de resultados
        System.out.println ("A soma de " + num1 + " com " + num2 + " é " + soma);
        break;
    case 2 :/* chamada domódulo procedimento Menos, passando num1 e num2 como
parâmetro*/
        Menos (num1 , num2);
        break;
    case 3 :/* chamada do módulo procedimento vezes, passando num1 e num2 como
parâmetro */
        Vezes (num1 , num2);
        break;
    case 4 :/* chamada do módulo função Dividido, passando num1 e num2 como parâ-
metro e atribuindo o retorno do módulo na variável div */
        div = Dividido (num1 , num2);
        // saída de resultados
        System.out.println ("A divisão de " + num1 + " com " + num2 + " é " + div);
        break;
    default : System.out.println ("operação inválida"); // saída de resultados
}
// fim do switch
System.exit(0);
} // fim do void main
} // fim da classe

```

2. Desenvolva um algoritmo que mostre um menu de opções para: calcular a soma de todos os números compreendidos entre 1 e 100; calcular a soma de todos os números pares compreendidos entre 1 e 100; e calcular a soma de todos os números divisíveis por 3.

```
class Somador {
```

```

// módulo função Menu
static int Menu()
{
    int op;
```

```

op = Integer.parseInt(JOptionPane.showInputDialog("Digite 1 para soma de todos os números entre 1 e 100\nDigite 2 para soma de todos os números pares entre 1 e 100\nDigite 3 para soma de todos os números divisíveis por 3 entre 1 e 100\nDigite 0 para sair do programa\n"));
return op;
} // fim do módulo Menu

// módulo função SomaTudo que retorna um valor inteiro
static int SomaTudo ()
{
    // declaração de variáveis que só podem ser usados dentro deste módulo
    int s=0, i;

    i = 1;
    // processamento de dados
    while (i <= 100)
    {
        s = s + i;
        i = i + 1;
    } // fim do while
    return s; // retorno do módulo
} // fim do módulo SomaTudo

// módulo procedimento SomaPares que recebe um valor inteiro
static void SomaPares (int par)
{
    // declaração de variáveis que só podem ser usados dentro deste módulo
    int s=0, i;

    i = 1;
    // processamento de dados
    do
    {
        if ((i % par) == 0)
        {
            s = s + i;
        } // fim do if
        i = i + 1;
    } while (i <= 100);
    System.out.println ("A somatória dos números pares de 1 até 100 é " + s); // saída de resultados
} // fim do módulo SomaPares

// módulo função SomaDivisivel3 que recebe um valor inteiro e retorna um valor inteiro
static int SomaDivisivel3 (int d)
{
    // declaração de variáveis que só podem ser usados dentro deste módulo
    int i, s = 0;

    // processamento de dados
    for (i = 1 ; i <= 100 ; i++)

```

```

{
    if ((i % d) == 0)
    {
        s = s + i;
    } // fim do if
} // fim do for
return s; // retorno da função
}// fim do módulo SomaDivisivel3

// módulo principal
public static void main (String args [])
{
    // declaração de variáveis que só podem ser usados dentro do algoritmo Somador
    int op, soma, div;

    // processamento de dados
    do
    {
        /* chamada do módulo procedimento Menu, sem parametrização */
        op = Menu(); // entrada de dados
        switch (op)
        {
            case 1 : /* chamada do módulo função SomaTudo, sem parâmetrização e atribuindo
o retorno do módulo na variável soma */
                soma = SomaTudo();
                System.out.println ("A somatória de 1 até 100 é " + soma); // saída de resultados
                break;
            case 2 : /* chamada do módulo procedimento SomaPares, passando o número 2
como parâmetro*/
                SomaPares(2);
                break;
            case 3 : /* chamada do módulo função SomaDivisivel3, passando o número 3 como
parâmetro e atribuindo o retorno do módulo na variável div */
                div = SomaDivisivel3(3);
                // saída de resultados
                System.out.println ("A somatória dos números divisíveis por 3 de 1 até 100
é " + div);
                break;
            case 0 : System.out.println ("Saindo do programa!");
                System.exit(0);
            default : System.out.println ("opção inválida, tente novamente");
        } // fim do switch
    } while (op != 0);
} // fim do void main
} // fim da classe

```

AVALIAÇÃO DE APRENDIZAGEM 2

Estrutura de repetição do para, vetores, matrizes, procedimentos e funções em algoritmos, Linguagem C e Linguagem Java.

Você foi encarregado de desenvolver um programa para uma empresa logística responsável por entregar encomendas entre cidades do Brasil. Seu programa deverá armazenar as distâncias entre as 5 cidades mais atendidas pela transportadora. Você deverá utilizar os códigos das cidades (1, 2, 3, 4 ou 5) e as distâncias deverão ser armazenadas em uma tabela de distâncias, conforme o exemplo abaixo:

	1	2	3	4	5
1	0	39,5	73,2	58	62
2	39,5	0	36	25,4	86,8
3	73,2	36	0	55,1	85,7
4	58	25,4	55,1	0	23
5	62	86,8	85,7	23	0

O programa desenvolvido por você deverá inicialmente apresentar um menu com três opções: 1) Cadastramento de distâncias, 2) Consulta de distâncias e 3) Sair do programa. No cadastramento de distâncias, seu programa deverá pedir para o usuário informar a distância entre a primeira e a segunda cidade e armazená-la na tabela, em seguida fazer o mesmo entre a primeira e a terceira, e assim por diante até todas as distâncias estarem cadastradas. Após isso seu programa deve retornar ao menu inicial; Na consulta de distâncias, seu programa deverá pedir para o usuário informar a cidade de origem e em seguida a cidade de destino. Caso os códigos das cidades sejam válidos, seu programa deverá retornar a distância entre as duas cidades armazenadas na tabela e em seguida retornar ao menu inicial. Caso o usuário digite um código incorreto, o sistema deve informar e pedir para que o usuário insira novamente um código de cidade; Caso o usuário escolha a terceira opção, o programa deverá ser encerrado.

Seu programa deve ser modularizado, utilizando um módulo para apresentar o menu (módulo procedimento sem parâmetros), um módulo para o cadastramento de distâncias(módulo função que devolve a tabela de distâncias) e outro módulo para a consulta de distâncias (módulo função parametrizado, que recebe como parâmetro a

tabela de distância e devolve a distância encontrada).

OBS: Diferente do Java, o módulo de cadastramento de usuários em C deverá ser um módulo procedimento que recebe a tabela de distâncias. Para que a tabela informada seja modificada para acrescentar as distâncias, ao utilizarmos o módulo deveremos informar a tabela precedida do operador &. Ex:

```
float distancias[5][5];
Cadastro(&distancias);
```

Solução em C

```
#include <stdio.h>
#include<stdlib.h>
```

// módulo função Menu que retorna a opção digitada pelo usuário

```
int Menu()
{
    int op;

    // mensagem ao usuário e
    // entrada de dados
    printf("Digite 1 para cadastramento de distâncias\nDigite 2 para consulta de distâncias\nDigite 3 para sair do programa\n:");
    scanf("%d", &op);
    return op;
} // fim do módulo Menu
```

//Módulo procedimento que recebe a tabela de distâncias e insere as distâncias informadas pelo usuário

```
void Cadastro(float distancias[5][5])
{
```

```
    int i, j;
```

```
    for (i = 0; i < 5; i++)
    {
```

```
        for (j = 0; j <= i; j++)
        {
```

```
            if (i == j)
            {
```

```
                distancias[i][j] = 0;
```

```
}
```

```

        else
        {
            printf("Digite a distancia da cidade %d ate %d: ", j + 1, i + 1);
            scanf("%f", &distancias[i][j]);
            distancias[j][i] = distancias[i][j];
        }
    }
}

// Módulo função parametrizado, que recebe como parâmetro a tabela de distância e devolve
a distância encontrada
float Consulta(float distancias[5][5])
{
    int origem, destino;

    do {
        printf("Digite o número da cidade de origem: ");
        scanf("%d", &origem);

        if (origem <= 0 || origem > 5)
        {
            printf("Número inválido!\n");
        }
    } while (origem <= 0 || origem > 5);

    do {
        printf("Digite o número da cidade de destino: ");
        scanf("%d", &destino);

        if (destino <= 0 || destino > 5)
        {
            printf("Número inválido!\n");
        }
    } while (destino <= 0 || destino > 5);

    return distancias[origem-1][destino-1];
}

// módulo principal
main()
{
    // declaração de variáveis e/ou constantes
    int op;
    float distancias[5][5];
    float distancia;
}

```

```

do
{
    op = Menu();

    switch (op)
    {
        case 1 :
            Cadastro(&distancias);
            break;
        case 2 :
            distancia = Consulta(distancias);
            printf("A distância é %f\n", distancia);
            break;
        case 3 :
            printf("Saindo do programa \n");
            break;
        default : printf ("opção inválida, tente novamente \n"); // saída de resultados
    } // fim do switch
} while (op != 3); // fim do do
} // fim do main

```

Solução em Java

```

import javax.swing.*;

public class Distancias {

    // módulo função Menu que retorna a opção digitada pelo usuário
    static int Menu()
    {
        int op;

        // mensagem ao usuário e entrada de dados
        op = Integer.parseInt(JOptionPane.showInputDialog("Digite 1 para cadastramento de distancias\nDigite 2 para consulta de distancias\nDigite 3 para sair do programa"));
        return op;
    } // fim do módulo Menu

    //Módulo função que devolve a tabela de distâncias informadas pelo usuário
    static double[][][] Cadastro()
    {
        int i, j;
        double distancias[][][] = new double[5][5][5];

        for (i = 0; i < 5; i++)
        {
            for (j = 0; j <= i; j++)
            {

```

```

        if (i == j)
        {
            distancias[i][j] = 0;
        }
        else
        {
            distancias[i][j] = Double.parseDouble(JOptionPane.
showInputDialog("Digite a distancia da cidade " + (j + 1) + " ate " + (i + 1) + ":"));

            distancias[j][i] = distancias[i][j];
        }
    }

    return distancias;
}

// Módulo função parametrizado, que recebe como parâmetro a tabela de distância e
devolve a distância encontrada
static double Consulta(double distancias[][])
{
    int origem, destino;

    do {
        origem = Integer.parseInt(JOptionPane.showInputDialog("Digite o
número da cidade de origem"));
        if (origem <= 0 || origem > 5)
        {
            System.out.println("Número inválido!");
        }
    } while (origem <= 0 || origem > 5);

    do {
        destino = Integer.parseInt(JOptionPane.showInputDialog("Digite o
número da cidade de destino"));

        if (destino <= 0 || destino > 5)
        {
            System.out.println("Número inválido!");
        }
    } while (destino <= 0 || destino > 5);

    return distancias[origem-1][destino-1];
}

public static void main (String args [ ])
{

```

```

// declaração de variáveis e/ou constantes
int op;
double distancias[][] = new double[5][5];
double distancia;

do
{
    op = Menu();

    switch (op)
    {
        case 1 :
            distancias = Cadastro();
            break;
        case 2 :
            distancia = Consulta(distancias);
            System.out.println("A distância é " + distancia);
            break;
        case 3 :
            System.out.println("Saindo do programa!");
            System.exit(0);
            break;
        default : System.out.println ("opção inválida, tente novamente");
    } // fim do switch
} while (op != 3);
} // fim do void main
} //fim da classe

```

EXERCÍCIOS 2

Estrutura de repetição do para, vetores, matrizes, procedimentos e funções em algoritmos, Linguagem C e Linguagem Java.

1. Escreva uma função que receba dois números inteiros e retorna a soma desses dois números.
2. Escreva uma função que receba três números reais e retorna o produto desses números.
3. Escreva uma função que receba um número e retorna verdadeiro se ele for positivo e falso caso contrário.
4. Escreva um procedimento que receba dois números inteiros e mostre a soma desses dois números.
5. Escreva um procedimento que receba três números reais e mostre o produto desses números.

6. Escreva um que receba um número e mostre a informação se esse número é par ou não.
7. Desenvolva um algoritmo que recebe dez números num vetor e mostre esses números.
8. Escreva uma função que recebe um vetor e retorna o maior número desse vetor.
9. Escreva um procedimento que recebe um vetor e mostre a soma dos números desse vetor.
10. Escreva um procedimento que recebe um vetor e mostre os números pares desse vetor.
11. Escreva uma função que recebe um vetor e retornar a soma dos números positivos desse vetor.
12. Escreva uma função que recebe uma matriz e retorna o maior número dessa matriz.

13. Desenvolva um algoritmo que receba 50 números e calcule a soma desses números.
14. Desenvolva um algoritmo que mostra o produto dos vinte números de 1 a 20.
15. Desenvolva um algoritmo que recebe uma opção inteira e sai da repetição somente quando o usuário digitar 0.
16. Desenvolva um algoritmo que mostra o resultado da série $1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{N}$.
17. Desenvolva um algoritmo que recebe dois números a e b e mostre o resultado de a elevado a b sem utilizado o operador da potência.
18. Escreva uma função que receba um vetor de números e calcule e retorna a média desses números.
19. Escreva um procedimento que receba um vetor de números e calcule e retorna a média desses números.
20. Escreva uma função que receba um vetor e retorna o maior valor desse vetor.
21. Escreva um procedimento que receba um vetor e escreva o menor valor desse vetor.
22. Desenvolva uma função que recebe um vetor de dez números e inverta os valores desse vetor em outro vetor.
23. Escreva uma função que receba dois números e retorna a potência desses dois números.
24. Escreva um procedimento que receba dois números e mostre a potência desses dois números.
25. Escreva um procedimento que recebe uma matriz e mostre a média dos números dessa matriz.

ATIVIDADE COMPLEMENTAR 1

Desenvolva um programa em C e um programa em Java para a seguinte situação-problema.

Alguns motoristas registram os quilômetros dirigidos e os litros consumidos a cada abastecimento de seus carros. Sua função é escrever um programa que ajude esses mo-

toristas a obter o consumo (em quilômetros por litro) de seus carros. Ao ser executado, seu programa solicitará que o usuário informe o número de quilômetros dirigidos e a quantidade de litros de combustível colocada naquele abastecimento. Após receber esta entrada, seu programa deverá calcular e mostrar a quantidade de quilômetros por litro obtida neste abastecimento e em seguida a quantidade média dos abastecimentos digitados até então e em seguida solicitar os dados do próximo abastecimento. Caso o usuário digite 0 no número de quilômetros e 0 na quantidade de litros de combustível consumidos, o programa deverá ser encerrado.

Um exemplo de funcionamento do programa poderia ser:

- ◊ Digite a quantidade de quilômetros percorridos (ou 0 para sair): 320
- ◊ Digite a quantidade de litros abastecidos percorridos (ou 0 para sair): 39
- ◊ Seu consumo neste abastecimento é: 8.20 km/l
- ◊ Sua média de consumo é: 8.20 km/l
- ◊ Digite a quantidade de quilômetros percorridos (ou 0 para sair): 360
- ◊ Digite a quantidade de litros abastecidos percorridos (ou 0 para sair): 37
- ◊ Seu consumo neste abastecimento é: 9.73 km/l
- ◊ Sua média de consumo é: 8.97 km/l
- ◊ Digite a quantidade de quilômetros percorridos (ou 0 para sair): 0
- ◊ Digite a quantidade de litros abastecidos percorridos (ou 0 para sair): 0
- ◊ Saindo...

ATIVIDADE COMPLEMENTAR 2

Desenvolva um programa em C e um programa em Java para a seguinte situação-problema.

Você foi encarregado de desenvolver um programa para uma empresa logística responsável por entregar encomendas entre cidades do Brasil. Seu programa deverá arma-

zenar as distâncias entre as 5 cidades mais atendidas pela transportadora. Você deverá utilizar os códigos das cidades (1, 2, 3, 4 ou 5) e as distâncias deverão ser armazenadas em uma tabela de distâncias, conforme o exemplo abaixo:

	1	2	3	4	5
1	0	39,5	73,2	58	62
2	39,5	0	36	25,4	86,8
3	73,2	36	0	55,1	85,7
4	58	25,4	55,1	0	23
5	62	86,8	85,7	23	0

O programa desenvolvido por você deverá inicialmente apresentar um menu com três opções: 1) Cadastramento de distâncias, 2) Consulta de distâncias e 3) Sair do programa. No cadastramento de distâncias, seu programa deverá pedir para o usuário informar a distância entre a primeira e a segunda cidade e armazená-la na tabela, em seguida fazer o mesmo entre a primeira e a terceira, e assim por diante até todas as distâncias estarem cadastradas. Após isso seu programa deve retornar ao menu inicial; Na consulta de distâncias, seu programa deverá pedir para o usuário informar a cidade de origem e em seguida a cidade de destino. Caso os códigos das cidades sejam válidos, seu programa deverá retornar a distância entre as duas cidades armazenadas na tabela e em seguida retornar ao menu inicial. Caso o usuário digite um código incorreto, o sistema deve informar e pedir para que o usuário insira novamente um código de cidade; Caso o usuário escolha a terceira opção, o programa deverá ser encerrado.

Seu programa deve ser modularizado, utilizando um módulo para apresentar o menu (módulo procedimento sem parâmetros), um módulo para o cadastramento de distâncias(módulo função que devolve a tabela de distâncias) e outro módulo para a consulta de distâncias (módulo função parametrizado, que recebe como parâmetro a tabela de distância e devolve a distância encontrada).

OBS: Diferente do Java, o módulo de cadastramento de usuários em C deverá ser um módulo procedimento que recebe a tabela de distâncias. Para que a tabela informada seja modificada para acrescentar as distâncias, ao utilizarmos o módulo deveremos informar a tabela precedida do operador &. Ex:

```
float distancias[5][5];
Cadastro(&distancias);
```

ATIVIDADE COMPLEMENTAR 3

Crie um programa que simule um sistema de autenticação. O programa deverá apresentar um menu com três opções. Na primeira opção, o programa deverá cadastrar os nomes e as senhas de 5 pessoas e retornar ao menu principal. Na segunda opção, o programa deverá perguntar o nome e a senha da pessoa que deseja acesso. Caso o nome entrado seja um dos cinco nomes cadastrados, o sistema deverá mostrar a mensagem “Acesso liberado”, e, caso contrário, “Acesso negado” e voltar ao menu principal. Caso o usuário escolha a terceira opção, o sistema deverá ser encerrado.

ATIVIDADE COMPLEMENTAR 4

O valor de π pode ser calculado usando a seguinte série:

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \dots$$

A precisão da aproximação do valor de π depende da quantidade de termos da série que utilizamos para fazer esse cálculo. Crie um programa que calcule a aproximação do valor de π utilizando n termos da série acima. O programa deverá pedir um número n ao usuário, e deverá imprimir o valor aproximado de π usando n termos (conte o primeiro 4 como um termo). Caso o usuário digite um número menor do que 1, o programa deverá solicitar ao usuário que entre outro número, repetindo isso até que o usuário digite um número inteiro maior ou igual a 1.

REFERÊNCIAS BIBLIOGRÁFICAS

Tabuti, L. M. Lógica de Programação. Faculdade IBTA, 3a Edição. 2a Impressão. 2005.
Tabuti, L. M. Estruturas de Dados. Faculdade IBTA, 2a Edição. 1a Impressão. 2005.

GABARITO - EXERCÍCIOS 1

1.
Algoritmo Soma
inicio

```
var  
    n1, n2, s inteiro;  
  
    ler (n1, n2);  
    s <- n1 + n2;  
    escrever (s);
```

fim.

2.
Algoritmo Divisao
inicio

```
var  
    n1, n2, q, r inteiro;  
  
    ler (n1, n2);  
    q <- n1 div n2;  
    r <- n1 mod n2;  
    escrever (q, r);
```

fim.

3.
Algoritmo NomeSobrenome
inicio

```
var  
    n, s alfanumérico;  
  
    ler(n, s);  
    escrever (n , “ “ , s);
```

fim.

4.
Algoritmo ParImpar
inicio

```
var  
    n inteiro;  
  
    ler (n);  
    se (n mod 2 = 0)  
        então  
            escrever (“é par”);
```

senão
 escrever (“é ímpar”);
fimse;
fim.

5.

Algoritmo Nota

```
inicio  
    var  
        n real;
```

```
    ler (n);  
    se (n >= 5)  
        então  
            escrever (“aprovado”);  
    senão  
        se (nota >= 3)
```

```
        então  
            escrever (“em recuperação”);  
        senão  
            escrever (“reprovado”);
```

```
    fimse;  
fimse;  
fim.
```

6.

Algoritmo Semestre

inicio

```
    var
```

m inteiro;

```
    ler (m);
```

```
    se (m >= 1 e m <= 6)  
        então  
            escrever (“primeiro semestre”);
```

```
    senão  
        se (m >= 7 e m <= 12)  
            então  
                escrever (“segundo semestre”);  
            fimse;  
        fim.
```

7.

Algoritmo trimestre

inicio

```

var
    m inteiro;
    ler (m);
    escolha (m)
        caso 1, 2, 3 : escrever ("primeiro trimestre");
        caso 4, 5, 6 : escrever ("segundo trimestre");
        caso 7, 8, 9 : escrever ("terceiro trimestre");
        caso 10, 11, 12 : escrever ("quarto trimestre");
        caso contrário: escrever ("inválido");
    fimescolha;
fim.

8.
Algoritmo tarefas
inicio
    var
        dia alfanumérico;
        ler (dia);
        escolha (dia)
            caso "segunda-feira", "sexta-feira" : escrever ("estudar algoritmos");
            caso "terça-feira" : escrever ("estudar matemática");
            caso "quarta-feira" : escrever ("estudar inglês");
            caso "quinta-feira" : escrever ("estudar português");
            caso "sábado" : escrever ("passar");
            caso "domingo" : escrever ("descansar");
            caso contrário : escrever ("inválido");
    fim.

9.
Algoritmo soma
    var
        i, n, s <- 0 inteiro;
        para i de 1 até 10 passo +1 faça
            ler (n);
            s <- s + n;
        fim.

fimpara;
escrever (s);
fim.

10.
Algoritmo SomaPares
    var
        i, n, s <- 0 inteiro;
        para i de 1 até 15 passo +1 faça
            ler (n);
            se (n mod 2 = 0)
                então
                    s <- s + n;
            fimse;
        fimpara;
        escrever (s);
    fim.

11.
Algoritmo Fatorial
    var
        i, n, fat <- 1 inteiro;
        ler (n);
        para i de 1 até n passo + 1 faça
            fat <- fat * i;
        fimpara;
        escrever (fat);
    fim.

12.
Algoritmo Tabuada
inicio
    var
        t, i inteiro;
        ler (t);
        para i de 1 até 10 passo + 1 faça
            escrever (i , " x " , t , " = " ,
            (t*i));
        fimpara;
    fim.

13.
Algoritmo MaiorNumero
inicio
    var
        n, maior, i inteiro;

```

```

    ler (n);
    maior <- n;
    para i de 1 até 19 passo +1 faça
        ler (n);
        se (n > maior)
            então
                maior <-
n);
                fimse;
            fimpara;
            escrever(maior);
        fim.

14.
Algoritmo SomaQuadrado
inicio
    var
        s, n1, n2 real;
    ler (n1, n2);
    s <- n1*n1 + n2*n2;
    escrever (s);
fim.

15.
Algoritmo SomaQuadrado
inicio
    var
        s, n1, n2 real;
    ler (n1, n2);
    s <- pot(n1,3) + pot(n2,3);
    escrever (s);
fim.

16.
Algoritmo Raiz
inicio
    var
        n, r inteiro;
    ler (n);
    r <- rad(n);
    escrever (r);
fim.

17.
Algoritmo Media
inicio
    var
        n1, n2, m real;
    ler (n1, n2);
    m <- (n1 + n2) / 2;
    escrever (m);
fim.

18.
Algoritmo Divisivel3
inicio
    var
        n inteiro;
    ler(n);
    se (n mod 3 = 0)
        então
            escrever ("é divisí-
vel por três");
        senão
            escrever ("não é di-
visível por três");
        fimse;
    fim.

19.
Algoritmo Area
inicio
    var
        l, a real;
    ler (l);
    se (l > 0)
        então
            a <- l * l;
            escrever (a);
        senão
            escrever ("valor in-
válido");
        fimse;
    fim.

20.
Algoritmo Comprimento
inicio
    var
        r, c real;
    ler (r);
    se (r > 0)
        então
            c <- 2 * 3.14 * r;
        senão
            escrever ("valor in-
válido");
    fim.

```

```

        escrever (c);
    senão
        escrever ("valor in-
válido");
    fimse;
fim.

21.
Algoritmo AreaCírculo
inicio
    var
        r, a real;
    ler (r);
    se (r > 0)
        então
            a <- 3.14 * r * r
            escrever (a);
    senão
        escrever ("valor in-
válido");
    fimse;
fim.

22.
Algoritmo Soma
inicio
    var
        i, s <- 0, n inteiro;
    para i de 1 até 30 passo +1 faça
        ler (n);
        s <- s + n;
    fimpara;
    escrever (n);
fim.

23.
Algoritmo Quadrado
inicio
    var
        n, i inteiro;
    para i de 10 até 20 passo +1 faça
        n <- i * i;
        escrever (n);
    fimpara;
fim.

24.
Algoritmo Idade
inicio
    var
        id, i, menor inteiro;
    ler (id);
    menor <- id;
    para i de 2 até 20 passo +1 faça
        ler (id);
        se (menor > id)
            então
                menor <-
id;
            fimse;
        fimpara;
        escrever (menor);
    fim.

25.
Algoritmo Media
inicio
    var
        n, i, m <- 0, i inteiro;
    para i de 1 até 10 passo +1 faça
        ler (n);
        m <- m + n;
    fimpara;
    m <- m / 10;
    escrever (m);
fim.

```

GABARITO - EXERCÍCIOS 2

1.
inteiro Soma (n1 inteiro, n2 inteiro)
inicio

var
 s inteiro;

s <- n1 + n2;
 retornar s;

fim.

2.
real Produto (n1 real, n2 real, n3 real)
inicio

var
 p real;
 p <- n1 * n2 * n3;
 retornar p;

fim.

3.

```

lógico Verifica (n inteiro)
inicio
    se (n > 0)
        então
            retornar verdadei-
ro;
        senão
            retornar falso;
        fimse;
fim.

4.
Soma (n1 inteiro, n2 inteiro)
inicio
    var
        s inteiro;
    s <- n1 + n2;
    escrever (s);
fim.

5.
Produto (n1 real, n2 real, n3 real)
inicio
    var
        p real;
    p <- n1 * n2 * n3;
    escrever (p);
fim.

6.
Verifica (n inteiro)
inicio
    se (n > 0)
        então
            escrever ("é par");
        senão
            escrever ("é ím-
par");
        fimse;
fim.

7.
Algoritmo VetorNumeros
inicio
    var
        Vet[10], n, i inteiro;
    para i de 0 até 9 passo +1 faça
        ler (n);

```

Vet[i] <- n;
escrever (Vet[i]);
fimpara;
fim.

8.
inteiro MaiorNumero (Vet[] inteiro)
inicio
 var
 maior <- Vet[0], i inteiro;

para i de 1 até (Vet.tamanho-1) passo
+1 faça
 se (Vet[i] > maior)
 então
 maior <-
 Vet[i];
 fimse;
 fimpara;
 retornar maior;
fim.

9.
SomaVetor (Vet[] real)
var
 i, s <- 0 inteiro;
 para i de 0 até (Vet.tamanho-1) pas-
so +1 faça
 s <- s + Vet[i];
 fimpara;
 escrever s;
fim.

10.
ParesVetor (Vet[] inteiro)
inicio
 var
 i inteiro;
 para i de 0 até (Vet.tamanho-1) pas-
so +1 faça
 se (Vet[i] mod 2 == 0)
 então
 escrever
 (Vet[i]);
 fimse;
 fimpara;

11.
real PositivoVetor(Vet[] real)

```

inicio
    var
        i, s <- 0 inteiro;
        para i de 0 ate (Vet.tamanho-1) passo +1 faça
            se (Vet[i] > 0)
                então
                    s <- s +
                    Vet[i];
                fimse;
            fimpara;
            retornar s;
        fim.

12.
inteiro MaiorNumero (Mat[][] inteiro)
inicio
    var
        maior <- Mat[0][0], i, j inteiro;
        para i de 0 ate (Mat.tamanho-1) passo +1 faça
            para j de 0 ate (Mat[i].tamanho-1) passo +1 faça
                se (Mat[i][j] > maior)
                    então
                        maior <- Mat[i][j];
                    fimse;
                fimpara;
            retornar maior;
        fim.

13.
Algoritmo Soma
inicio
    var
        s <- 0, i <- 1, n inteiro;
        enquanto (i <= n) faça
            ler (n);
            s <- s + n;
            i <- i + 1;
        fimenquanto;
        escrever (s);
    fim.

14.
Algoritmo Produto
inicio
    var
        p <- 1, i <- 1, n inteiro;
        enquanto (i <= n) faça
            p <- p * i;
            i <- i + 1;
        fimenquanto;
        escrever (p);
    fim.

15.
Algoritmo Opcão
inicio
    var
        op inteiro;
        faç
            ler (op);
        enquanto (op <> 0);
    fim.

16.
Algoritmo Serie
inicio
    var
        n, i <- 1, s <- 0 inteiro;
        ler (n);
        enquanto (i <= n) faça
            s <- s + 1/i;
        fimenquanto;
        escrever (s);
    fim.

17.
Algoritmo Potencia
inicio
    var
        a, b, pot <- 1 inteiro;
        ler (a, b);
        enquanto (b >= 1) faça
            pot <- pot * a;
            b <- b - 1;
        fimenquanto;
        escrever (pot);
    fim.

```

18.
real Media (Vet[] inteiro)
inicio
var m <- 0, i inteiro;
para i de 0 até (Vet.tamanho - 1) passo +1 faça
m <- m + Vet[i];
fimpara;
m <- m / Vet.tamanho;
retornar (m);
fim.

menor <- Vet[0];
para i de 1 até (Vet.tamanho - 1) passo +1 faça
se (Vet[i] < menor)
então
menor <-
Vet[i];
fimse;
fimpara;
retornar menor;
fim.

22.
Algoritmo Inverte
inicio
var
i, Vet[10], Inv[10] inteiro;
para i de 0 até 9 passo +1 faça
ler (Vet[i]);
fimpara;

para i de 0 até 9 passo +1 faça
Inv[i] <- Vet[9-i];
fimpara;
para i de 0 até 9 passo +1 faça
escrever (Vet[i], Inv[i]);
fimpara;
fim.

23.
inteiro Potencia(a inteiro, b inteiro)
inicio
var
i, pot <- 1 inteiro;
enquanto (b >= 1) faça
pot <- pot * a;
b <- b - 1;
fimenquanto;
retornar pot;
fim.

24.
Potencia(a inteiro, b inteiro)
inicio
var
i, pot <- 1 inteiro;
enquanto (b >= 1) faça

```

        pot <- pot * a;
        b <- b - 1;
fimenquanto;
escrever (pot);
fim.

25.
Media (Mat[]) inteiro
inicio
    var
        m <- 0, i, j inteiro;
        para i de 0 até (Mat.tamanho-1) passo +1 faça
            para j de 0 até (Mat[0].tamanho-1) passo +1 faça
                media <-
media + Mat[i][j];
            fimpara;
            fimpara;
            media <- media / (Mat.tamanho *
Mat[0].tamanho);
            escrever (media);
fim.

```

GABARITO - ATIVIDADE COMPLEMENTAR 1

Solução em C

```

#include <stdio.h>

// módulo principal
main()
{
    // declaração de variáveis e/ou constantes
    int contador = 0;
    float quilometros, litros;
    float consumoAtual = 0;
    float consumoAcumulado = 0;

    do
    {
        printf("Digite a quantidade
de quilometros percorridos (ou 0 para sair):
");
        scanf("%f", &quilometros);

```

```

printf("Digite a quantidade
de litros abastecidos percorridos (ou 0 para
sair): ");
scanf("%f", &litros);

if (litros != 0 && quilome-
etros != 0)
{
    contador = conta-
    dor + 1;
    consumoAtual =
    quilometros / litros;
    consumoAcumula-
    do = consumoAcumulado + consumoAtual;

    printf("Seu consu-
mo neste abastecimento é: %fkm/l \n", con-
sumoAtual);
    printf("Sua média
de consumo é: %fkm/l \n", (consumoAcu-
mulado/contador));
}
else
{
    printf("Saindo...
\n");
}
} while (litros != 0 && quilometros
!= 0); // fim do do
} // fim do main

```

Solução em Java

```

import javax.swing.*;
public class Consumo {
    public static void main(String[] args)
    {
        // declaração de variáveis e/
        ou constantes
        int contador = 0;
        double quilometros, litros;
        double consumoAtual = 0;
        double consumoAcumula-
        do = 0;
        do

```

```

    {
        quilometros
        = Double.parseDouble(JOptionPane.
        showInputDialog("Digite a quantidade de
        quilometros percorridos (ou 0 para sair)"));
        litros =
        Double.parseDouble(JOptionPane.
        showInputDialog("Digite a quantidade de
        litros abastecidos percorridos (ou 0 para
        sair)"));

        if (litros != 0 &&
        quilometros != 0)
        {
            contador =
            contador + 1;
            consumo-
            Atual = quilometros / litros;
            consumo-
            Acumulado = consumoAcumulado + consu-
            moAtual;

            JOptionPane-
            ne.showMessageDialog(null,"Seu consumo
            neste abastecimento é: " + consumoAtual +
            "km/l");

            JOptionPane-
            ne.showMessageDialog(null,"Sua média de
            consumo é: " + (consumoAcumulado/conta-
            dor) + "km/l");
        }
        else
        {
            JOptionPane-
            ne.showMessageDialog(null,"Saindo...");
        }
    } while (litros != 0 && quilo-
    metros != 0); // fim do do

} // fim do void main
} //fim da classe

```

GABARITO - ATIVIDADE COMPLEMENTAR 2

Solução em C

```

#include <stdio.h>
#include<stdlib.h>

// módulo função Menu que retorna a opção
// digitada pelo usuário

```

```

int Menu()
{
    int op;

    // mensagem ao usuário e
    entrada de dados
    printf("Digite 1 para cadastramento de distâncias\nDigite 2 para consulta de distâncias\nDigite 3 para sair do programa\n:");
    scanf("%d", &op);
    return op;
} // fim do módulo Menu

// Módulo procedimento que recebe a tabela
// de distâncias e insere as distâncias informadas
// pelo usuário
void Cadastro(float distancias[5][5])
{
    int i, j;

    for (i = 0; i < 5; i++)
    {
        for (j = 0; j <= i; j++)
        {
            if (i == j)
            {
                distancias[i][j] = 0;
            }
            else
            {

```

```

                printf("Digite a distância da cidade %d ate
                %d: ", j + 1, i + 1);

```

```

                scanf("%f", &distancias[i][j]);

```

```

                distancias[j][i] = distancias[i][j];
            }
        }
    }
}
```

```

// Módulo função parametrizado, que recebe como parâmetro a tabela de distância e
// devolve a distância encontrada
float Consulta(float distancias[5][5])
{

```

```

int origem, destino;

do {
    printf("Digite o número da cidade de origem: ");
    scanf("%d", &origem);

    if (origem <= 0 || origem > 5)
    {
        printf("Número inválido!\n");
    }

} while (origem <= 0 || origem > 5);

do {
    printf("Digite o número da cidade de destino: ");
    scanf("%d", &destino);

    if (destino <= 0 || destino > 5)
    {
        printf("Número inválido!\n");
    }

} while (destino <= 0 || destino > 5);

return distancias[origem-1][destino-1];
}

// módulo principal
main()
{
    // declaração de variáveis e/ou constantes
    int op;
    float distancias[5][5];
    float distancia;

    do
    {
        op = Menu();

        switch (op)
        {
            case 1 :
                Cadastro(&distancias);
                break;
            case 2 :
                distancia = Consulta(distancias);
                printf("A distância é %f\n", distancia);
                break;
        }
    }
}

```

```

        case 3 :
            printf("Saindo do programa \n");
            break;
        default : printf ("opção inválida, tente novamente \n"); // saída de resultados
    } // fim do switch
} while (op != 3); // fim do do
} // fim do main

```

Solução em Java

```

import javax.swing.*;

public class Distancias {

    // módulo função Menu que retorna a opção digitada pelo usuário
    static int Menu()
    {
        int op;

        // mensagem ao usuário e entrada de dados
        op = Integer.parseInt(JOptionPane.showInputDialog("Digite 1 para cadastramento de distâncias\nDigite 2 para consulta de distâncias\nDigite 3 para sair do programa"));
        return op;
    } // fim do módulo Menu

    //Módulo função que devolve a tabela de distâncias informadas pelo usuário
    static double[][] Cadastro()
    {
        int i, j;
        double distancias[][] = new double[5][5];

        for (i = 0; i < 5; i++)
        {
            for (j = 0; j <= i; j++)
            {
                if (i == j)
                {
                    distancias[i][j] = 0;
                }
                else
                {
                    distancias[i][j] = Double.parseDouble(JOptionPane.showInputDialog("Digite a distância da cidade " + (j + 1) + " até " + (i + 1) + ":"));
                    distancias[j][i] = distancias[i][j];
                }
            }
        }
        return distancias;
    }
}

```

```
}
```

```
// Módulo função parametrizado, que recebe como parâmetro a tabela de distância e  
devolve a distância encontrada
```

```
static double Consulta(double distancias[][])
```

```
{
```

```
    int origem, destino;
```

```
    do {
```

```
        origem = Integer.parseInt(JOptionPane.showInputDialog("Digite o  
número da cidade de origem"));
```

```
        if (origem <= 0 || origem > 5)
```

```
{
```

```
            System.out.println("Número inválido!");
```

```
}
```

```
} while (origem <= 0 || origem > 5);
```

```
    do {
```

```
        destino = Integer.parseInt(JOptionPane.showInputDialog("Digite o  
número da cidade de destino"));
```

```
        if (destino <= 0 || destino > 5)
```

```
{
```

```
            System.out.println("Número inválido!");
```

```
}
```

```
} while (destino <= 0 || destino > 5);
```

```
    return distancias[origem-1][destino-1];
```

```
}
```

```
public static void main (String args [ ])
```

```
{
```

```
    // declaração de variáveis e/ou constantes
```

```
    int op;
```

```
    double distancias[][] = new double[5][5];
```

```
    double distancia;
```

```
    do
```

```
{
```

```
    op = Menu();
```

```
    switch (op)
```

```
{
```

```
    case 1 :
```

```
        distancias = Cadastro();
```

```
        break;
```

```

        case 2 :
            distancia = Consulta(distancias);
            System.out.println("A distância é " + distancia);
        break;
    case 3 :
        System.out.println("Saindo do programa!");
        System.exit(0);
        break;
    default : System.out.println ("opção inválida, tente novamente");
}
} // fim do switch
} while (op != 3);
} // fim do void main
} //fim da classe

```

GABARITO - ATIVIDADE COMPLEMENTAR 3

Solução em Java

```

import javax.swing.*;
public class Acesso {

    // módulo função Menu que retorna a opção digitada pelo usuário
    static int Menu()
    {
        int op;

        // mensagem ao usuário e entrada de dados
        op = Integer.parseInt(JOptionPane.showInputDialog("Digite 1 para cadastramento dos usuários\nDigite 2 para teste de acesso\nDigite 3 para sair do programa"));
        return op;
    } // fim do módulo Menu

    //Módulo função que devolve a tabela de usuários e senhas
    static String[][] Cadastro()
    {
        int i;
        String usuarios[][] = new String[5][2];

        for (i = 0; i < 5; i++)
        {
            usuarios[i][0] = JOptionPane.showInputDialog("Digite o nome do usuário " + (i + 1) + ":");

            usuarios[i][1] = JOptionPane.showInputDialog("Digite a senha do usuário " + (i + 1) + ":");

        }

        return usuarios;
    }
}

```

```

// Módulo função parametrizado, que recebe como parâmetro a tabela de usuários e
pede um usuário e senha para teste de acesso
static void TestarAcesso(String usuarios[][])
{
    String usuarioTeste, senhaTeste;
    int i;

    usuarioTeste = JOptionPane.showInputDialog("Digite o nome do usuário:");
    senhaTeste = JOptionPane.showInputDialog("Digite a senha de acesso:");

    for (i = 0; i < 5; i++)
    {
        if(usuarioTeste.equals(usuarios[i][0]) && senhaTeste.equals(usuarios[i]
[1]))
        {
            System.out.println("Acesso liberado!");
            return;
        }
    }

    System.out.println("Acesso negado!");
}

public static void main (String args [ ])
{
    // declaração de variáveis e/ou constantes
    int op;
    String usuarios[][] = new String[5][2];

    do
    {
        op = Menu();

        switch (op)
        {
            case 1 :
                usuarios = Cadastro();
                break;
            case 2 :
                TestarAcesso(usuarios);
                break;
            case 3 :
                System.out.println("Saindo do programa!");
                System.exit(0);
                break;
            default : System.out.println ("opção inválida, tente novamente");
        } // fim do switch
}

```

```
        } while (op != 3);
    } // fim do void main
} //fim da classe
```

GABARITO - ATIVIDADE COMPLEMENTAR 4

Solução em Java

```
import javax.swing.*;
public class ValorDePi {
    public static void main (String args [])
    {
        int n, i;
        double pi = 0.0;

        do {
            n = Integer.parseInt(JOptionPane.showInputDialog("Digite o número de termos:"));
        } while (n < 1);

        for (i = 0; i < n; i++)
        {
            if (i % 2 == 0)
            {
                pi = pi + 4.0 / (2.0 * i + 1.0);
            }
            else
            {
                pi = pi - 4.0 / (2.0 * i + 1.0);
            }
        }

        System.out.println(pi);
    } // fim do void main
} //fim da classe
```

Solução em C

```
#include <stdio.h>

// módulo principal
main()
{
```

```
// declaração de variáveis e/ou constantes
int n, i;
float pi = 0.0;

do {
    printf("Digite o número de termos: ");
    scanf("%d", &n);
} while (n < 1);

for (i = 0; i < n; i++)
{
    if (i % 2 == 0)
    {
        pi = pi + 4.0 / (2.0 * i + 1.0);
    }
    else
    {
        pi = pi - 4.0 / (2.0 * i + 1.0);
    }
}

printf("%f", pi);
} // fim do main
```

SUAS ANOTAÇÕES
