

Introdução à Lógica de Programação

Amanda Paganini Lima
384.846.348-25



IMPACTA
EDITORAS

COD.: TE 1584/1

Introdução à Lógica de Programação

Créditos

Copyright © TechnoEdition Editora Ltda.

Todos os direitos autorais reservados. Este manual não pode ser copiado, fotocopiado, reproduzido, traduzido ou convertido em qualquer forma eletrônica, ou legível por qualquer meio, em parte ou no todo, sem a aprovação prévia, por escrito, da TechnoEdition Editora Ltda., estando o contrafator sujeito a responder por crime de Violação de Direito Autoral, conforme o art.184 do Código Penal Brasileiro, além de responder por Perdas e Danos. Todos os logotipos e marcas utilizados neste material pertencem às suas respectivas empresas.

"As marcas registradas e os nomes comerciais citados nesta obra, mesmo que não sejam assim identificados, pertencem aos seus respectivos proprietários nos termos das leis, convenções e diretrizes nacionais e internacionais."

Introdução à Lógica de Programação

Coordenação Geral

Marcia M. Rosa

Coordenação Editorial

Henrique Thomaz Bruscagin

Supervisão Editorial

Simone Rocha Araújo Pereira

Atualização

André Luís de Vasconcelos

Criação de exemplos -

Apêndice

Bruno Bove Barbosa

João Henrique Cunha Rangel

Ricardo Azzi Silva

Revisão Ortográfica e Gramatical

Alexandre Hideki Chicaoka

Diagramação

Ivâ Lucino Camargo

Edição nº 1 | Cód.: 1584/1
julho/ 2013

Sumário

Informações sobre o treinamento	8
Capítulo 1 – Introdução à Lógica.....	9
1.1. Lógica	10
1.2. Programa	12
1.2.1. Tipos de linguagem de programação.....	12
1.3. Tradutores	12
1.3.1. Tipos de tradutores	13
Pontos principais	14
Teste seus conhecimentos	15
Mãos à obra	19
Capítulo 2 – Sistemas de numeração	23
2.1. Bit e byte	24
2.2. Sistemas de numeração	25
2.2.1. Sistema Decimal	27
2.2.2. Sistema Binário.....	27
2.2.3. Sistema Hexadecimal.....	27
2.3. Conversão de sistemas de numeração	28
2.3.1. Conversão de Binário para Decimal.....	28
2.3.2. Conversão de Hexadecimal para Decimal.....	29
2.3.3. Conversão de Binário para Hexadecimal	30
2.3.4. Conversão de Hexadecimal para Binário	31
2.3.5. Conversão de Decimal para Binário.....	32
2.3.6. Conversão de Decimal para Hexadecimal	34
2.4. Forma rápida para conversão de sistemas de numeração.....	35
Pontos principais	38
Teste seus conhecimentos	39
Mãos à obra	43
Capítulo 3 – Algoritmo.....	47
3.1. Algoritmo	48
3.2. Elementos de um algoritmo	50
3.2.1. Ação	50
3.2.2. Decisão	51
3.2.3. Laço ou Loop	51
3.3. Algoritmo com o comando SE encadeado	53
3.4. Algoritmo com o comando CASO	54
3.5. Algoritmo com o comando ENQUANTO.....	55
Pontos principais	56
Mãos à obra	57
Capítulo 4 – Variáveis, Operadores e Funções.....	59
4.1. Introdução.....	60
4.2. Utilizando variáveis	60
4.2.1. Consistência de condições.....	60
4.2.2. Controle de repetições	60
4.2.3. Comparações de variáveis de memória com campos de registros	61
4.3. Tipos de variáveis.....	61
4.4. Nomes de variáveis.....	62

Introdução à Lógica de Programação

4.5.	Declaração de variáveis	63
4.6.	Comando de atribuição	64
4.7.	Constantes	64
4.8.	Operadores aritméticos	65
4.8.1.	Contadores e acumuladores	70
4.9.	Operadores relacionais	71
4.10.	Operadores lógicos	72
4.10.1.	Tabela de decisão	73
4.10.2.	Tabela de decisão com números binários	75
4.11.	Função	78
4.12.	Concatenação de alfanuméricos	79
	Pontos principais	80
	Teste seus conhecimentos	83
	Mãos à obra	87
	Capítulo 5 – Fluxograma	95
5.1.	Introdução.....	96
5.2.	Simbologia	96
5.3.	Criando fluxogramas	98
5.3.1.	Estruturas básicas	99
5.4.	Teste de Mesa	105
	Pontos Principais	106
	Teste seus conhecimentos	107
	Mãos à obra	113
	Capítulo 6 – Processamento predefinido	117
6.1.	Introdução.....	118
6.2.	Construindo um processamento predefinido	119
	Pontos principais	123
	Teste seus conhecimentos	125
	Mãos à obra	131
	Capítulo 7 – Laço ou loop e repetição	133
7.1.	Introdução.....	134
7.2.	Comando FOR...NEXT (PARA...PRÓXIMO)	134
7.3.	Comando WHILE (ENQUANTO)	137
7.4.	Comando IF...THEN...ELSE (SE...ENTÃO...SENÃO)	140
	Pontos principais	142
	Teste seus conhecimentos	143
	Mãos à obra	149
	Capítulo 8 – Variáveis indexadas e Laços encadeados	151
8.1.	Vetores e matrizes.....	152
8.2.	Laços encadeados	159
	Pontos principais	167
	Teste seus conhecimentos	169
	Mãos à obra	175

Sumário

Capítulo 9 – Banco de dados e tipos de programação	185
9.1. Introdução.....	186
9.2. Banco de dados	186
9.2.1. Considerações para tipos de dados	188
9.2.2. Tipos de dados.....	188
9.3. Modelo de dados	189
9.3.1. Relacionamento.....	189
9.3.1.1.Chave primária	189
9.3.1.2.Chave estrangeira.....	189
9.3.2. Modelo Entidade–Relacionamento	190
9.3.3. Índice	190
9.3.4. Regras de validação.....	190
9.3.5. Texto de validação	190
9.4. Objeto	191
9.4.1. Elementos da interface de um objeto.....	191
9.5. Tipos de Programação	192
9.5.1. Ocorrências de eventos	194
9.5.2. Mensagens do Windows.....	194
9.5.3. Gerenciador de eventos	194
9.5.4. Controlador de evento padrão	195
9.5.5. Procedure complementar.....	195
9.6. Criação de tabelas	196
9.7. Relacionamento das tabelas	197
9.8. Consistência dos campos	198
9.9. Sistema de controle de cadastro	200
9.9.1. Programa de inclusão	201
9.9.2. Programa de consulta.....	203
9.9.3. Programa de alteração.....	205
9.9.4. Programa de exclusão	207
9.9.5. Considerações finais.....	209
Pontos principais	210
Mãos à obra	213

Apêndice - Linguagens de programação e exemplos de programas	219
1.1. Introdução.....	220
1.2. Java	220
1.2.1. Exemplos	220
1.3. SQL	224
1.3.1. Exemplos	224
1.4. JavaScript	226
1.4.1. Exemplos	227
1.5. C#	228
1.5.1. Exemplos	228
1.6. PHP	232
1.7. Visual Basic for Applications (VBA).....	234
1.7.1. Excel	234
1.7.2. Access.....	239

Informações sobre o treinamento

Para que os alunos possam obter um bom aproveitamento deste **curso de Introdução à Lógica de Programação**, é importante que eles tenham participado do nosso curso de Ambiente Windows, ou possuam conhecimentos equivalentes.

Introdução à Lógica

1

- ✓ Conceitos de Lógica;
- ✓ Programa;
- ✓ Tradutores.



IMPACTA
EDITORA

1.1. Lógica

Lógica é a maneira de raciocinar particular a um indivíduo ou a um grupo, gerando uma sequência coerente, regular e necessária de acontecimentos ou métodos, com a finalidade de obter uma solução prática e eficaz para um problema.

Se observarmos o nosso dia a dia, usamos a lógica frequentemente, em nossas casas, no trabalho, nas compras, no trânsito, ou seja, em tudo. Como exemplo, podemos citar a lógica que fazemos desde a hora que acordamos até quando chegamos ao trabalho:

- **Acordar no horário programado**
- **Tomar banho**
- **Vestir a roupa adequada para trabalhar**
- **Tomar café**
- **Sair de casa**
- **Chegar ao local de trabalho dentro do horário previsto**

Como podemos observar, já acordamos pensando no que temos que fazer, portanto, já acordamos utilizando a lógica e, se a lógica inicial não for a ideal, nós a modificamos para que ela nos leve à melhor solução do problema. Por exemplo, se alguém já estiver tomando banho, então, podemos tomar café antes para que não nos atrasemos para o trabalho.

- Acordar no horário programado
- Verificar se o banheiro está livre
 - Se sim:
 - ✓ Tomar banho
 - ✓ Vestir a roupa adequada para trabalhar
 - ✓ Tomar café
 - Se não:
 - ✓ Tomar café
 - ✓ Tomar banho
 - ✓ Vestir a roupa adequada para trabalhar
- Sair de casa
- Chegar ao local de trabalho dentro do horário previsto

Existem diversos tipos de exercícios e jogos que servem para desenvolver o raciocínio lógico. Vamos começar com exercícios de lógica mais comuns e depois focaremos a programação.

- Exemplo

Vamos supor que exista uma caixa com 15 bolas, sendo 5 verdes, 5 amarelas e 5 azuis. Quantas bolas devem ser retiradas da caixa, sem olhar, para termos certeza de que saíram 2 bolas de cor azul?

- Resposta

- 5 bolas verdes
- 5 bolas amarelas
- 2 bolas azuis

Total: 12 bolas

- Conclusão

Temos que ter certeza que saíram 2 bolas azuis, portanto consideramos a pior hipótese no caso de retirar bolas aleatoriamente. Foram retiradas as 10 bolas entre verdes e amarelas e depois saíram as bolas azuis.

1.2. Programa

Programa é uma sequência lógica de instruções escritas em uma linguagem de programação, para serem executadas passo a passo, com a finalidade de atingir um determinado objetivo.

1.2.1. Tipos de linguagem de programação

Uma linguagem de programação é um método padronizado para comunicar instruções para um computador.

As linguagens de programação podem ser de **baixo nível** e de **alto nível**.

As linguagens de baixo nível são aquelas capazes de compreender a arquitetura do computador e que utilizam somente instruções do processador. Exemplos: Linguagem de máquina e Assembly.

As linguagens de alto nível são aquelas com a escrita mais próxima da linguagem humana. Exemplos: Objective-C, C++, C#, Delphi, Java, VB, MATLAB e ASP.

1.3. Tradutores

Os **tradutores** foram criados para tornar mais fácil a interface entre o usuário e a máquina. Como já vimos anteriormente, as linguagens são divididas em baixo e alto nível, cada uma refletindo uma proximidade com a linguagem natural do usuário.

O computador só executa instruções em linguagem de máquina, a qual é composta por dígitos binários. Logo, para que o computador execute instruções escritas em linguagens com estruturas diferentes, é preciso que essas instruções sejam traduzidas para a linguagem de máquina.

O tipo da tradução depende da complexidade da estrutura das linguagens.

1.3.1. Tipos de tradutores

Existem quatro tipos básicos de tradutores: **Montador**, **Interpretador**, **Run-time** e **Compilador**.

O **Montador** traduz a linguagem Assembly para a linguagem de máquina. Sua estrutura é relativamente simples e depende diretamente do processador utilizado, pois cada processador tem seu set de instruções característico.

Os outros tradutores são mais complexos, pois necessitam fazer análises mais sofisticadas da estrutura da linguagem para realizar a tradução.

O **Interpretador** realiza a tradução e a execução simultaneamente, não gerando o código-objeto (linguagem de máquina) em disco.

A geração de código em disco é observada no **Run-time** e no **Compilador**. A diferença entre eles é que o **Run-time** trabalha com um código intermediário (pseudocompilado).

O **Compilador** é um programa que traduz uma linguagem de programação de alto nível para linguagem de máquina, gerando um código-objeto independente.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- **Lógica** é a maneira de raciocinar particular a um indivíduo ou a um grupo, gerando uma sequência coerente, regular e necessária de acontecimentos ou métodos, com a finalidade de obter uma solução prática e eficaz para um problema;
- **Programa** é uma sequência lógica de instruções escritas em uma linguagem de programação, para serem executadas passo a passo, com a finalidade de atingir um determinado objetivo;
- Os **tradutores** foram criados para tornar mais fácil a interface entre o usuário e a máquina. Como já vimos anteriormente, as linguagens são divididas em baixo e alto nível, cada uma refletindo uma proximidade com a linguagem natural do usuário.

Introdução à Lógica

Teste seus conhecimentos

1

Amanhã pagam 25.
384.046.



IMPACTA
EDITORA

Introdução à Lógica de Programação

1. Uma tostadeira de pães suporta duas fatias de cada vez. Para torrar um lado de uma fatia demora 1 minuto. Qual o menor tempo para torrar 3 fatias dos dois lados?

- 3 minutos
- 4 minutos
- 5 minutos
- 6 minutos
- Nenhuma das alternativas anteriores está correta.

2. Um tijolo pesa 1 kg mais $\frac{1}{2}$ tijolo. Quanto pesa um tijolo e meio?

- 2 Kg
- 2,5 Kg
- 3 Kg
- 4 Kg
- Nenhuma das alternativas anteriores está correta.

3. Em uma caixa existem 40 bolas, sendo 10 verdes, 10 amarelas, 10 azuis e 10 brancas ou pretas (há pelo menos uma branca e uma preta). Quantas bolas devem ser retiradas da caixa, sem olhar, para termos certeza de que saíram 5 bolas verdes?

- 5 bolas
- 30 bolas
- 35 bolas
- 40 bolas
- Nenhuma das alternativas anteriores está correta.

4. Em uma caixa existem 40 bolas, sendo 10 verdes, 10 amarelas, 10 azuis e 10 brancas ou pretas (há pelo menos uma branca e uma preta). Quantas bolas devem ser retiradas da caixa, sem olhar, para termos certeza de que saíram 2 da mesma cor?

- 5 bolas
- 6 bolas
- 10 bolas
- 40 bolas
- Nenhuma das alternativas anteriores está correta.

Introdução à Lógica de Programação

5. Em uma caixa existem 39 bolas, sendo 10 verdes, 10 amarelas, 10 azuis e 9 brancas ou pretas (há pelo menos uma branca e uma preta). Quantas bolas devem ser retiradas da caixa, sem olhar, para termos certeza de que saíram 9 da mesma cor?

- 9 bolas
- 18 bolas
- 27 bolas
- 34 bolas
- Nenhuma das alternativas anteriores está correta.

Introdução à Lógica

Mãos à obra!

1



IMPACTA
EDITORIA

Introdução à Lógica de Programação

Laboratório 1

Exercício 1

Érica, Priscila e Débora são estudantes, mas cada uma frequenta um curso diferente: Enfermagem, Educação Física e Música. As notas delas na prova teórica foram 9,0 , 9,5 e 7,0, e as notas na prova prática foram 8,0 , 8,5 e 10,0. Sabendo-se que:

- Quem tirou 9,0 na prova teórica tirou 8,0 na prática;
- A estudante de Enfermagem obteve 9,5 na prova teórica;
- Débora ficou chateada com o 7,0 que obteve na prova teórica;
- Priscila conseguiu a maior nota na prova prática;
- Quem estuda música não tirou 8,5.

Qual é o curso e as notas de cada uma das estudantes?

Exercício 2

No vagão de um trem do metrô com destino para Jabaquara, estão três alunos: um de Lógica, um de Flash e um de V.B. Seus nomes, por ordem alfabética, são: Antônio, João e Pedro. No vagão, havia também 3 outros passageiros com os mesmos nomes: Sr. Antônio, Sr. João e Sr. Pedro.

- Pedro sempre conversa com o aluno de Flash;
- O Sr. João desceu na estação Vila Mariana;
- O aluno de Lógica desceu na estação Praça da Árvore;
- O senhor que vive no Jabaquara tem o mesmo nome do aluno de Lógica;
- O Sr. Antônio ganha exatamente R\$ 8.000,00 por mês;

- O Sr. João ganha exatamente R\$ 7.500,00 por mês;
- O vizinho do aluno de Lógica é um dos três senhores mencionados e ganha exatamente o triplo do que ganha o aluno de Lógica.

De acordo com as informações anteriores, qual é o nome do aluno de V.B.?

Exercício 3

A cabeça de um peixe mede 8 cm. A cauda do peixe mede o tamanho da cabeça mais $\frac{1}{2}$ corpo. O corpo mede o tamanho da cabeça mais a cauda. Quanto mede o peixe?

Exercício 4

Há 8 bolas idênticas, sendo uma delas mais pesada que as outras 7 bolas, que têm o mesmo peso. Utilizando uma balança de dois braços, descubra qual é a mais pesada, nas situações a seguir:

- Utilizando a balança 3 vezes;
- Utilizando a balança 2 vezes.

Exercício 5

Temos 3 bolas (A, B e C) de 3 cores (vermelha, branca e amarela). Dadas as frases:

- A é vermelha;
- B não é vermelha;
- C não é amarela.

E sabendo-se que uma frase é verdadeira e duas são falsas, descubra a cor de cada bola.

Introdução à Lógica de Programação

Exercício 6

Há 3 pessoas (Ana, Bia e Maria) e 3 tortas (chocolate, limão e morango). Dadas as frases:

- Bia comeu torta de limão;
- Bia não comeu torta de chocolate;
- Ana não comeu torta de chocolate;
- Ana não comeu torta de morango.

E sabendo-se que uma frase é verdadeira e 3 são falsas, descubra que torta cada uma das pessoas comeu.

Exercício 7

André Luis, Luis Carlos e Luiz Antônio saíram para passear de moto. Após uma parada, eles trocaram as motos e os capacetes entre si. Quer dizer, cada um passeia agora com a moto de um segundo e com o capacete de um terceiro. O rapaz que está com o capacete de Luiz Antônio está com a moto de Luis Carlos. Quem está com a moto de André Luis?

Exercício 8

Ao passar no hall do edifício, um rapaz perguntou onde estava o zelador a algumas pessoas que estavam sentadas por ali. O rapaz obteve as seguintes respostas, informando que ele estava em 4 andares diferentes:

- Ele está no último andar.
- Não, está no 10º.
- Acho que está no 13º.
- Não, está no 16º andar.

Afinal, onde está o zelador? É fácil saber onde ele está, pois alguém errou por um andar, outro por dois, outro por três e outro por quatro.

Sistemas de numeração

2

- ✓ Bit e byte;
- ✓ Sistemas de numeração;
- ✓ Conversão de sistemas de numeração.



IMPACTA
EDITORA

2.1. Bit e byte

BIT é a contração de **BInary digiT**, que significa dígito binário. Bit é a menor unidade de informação do computador.

A seguir podemos observar a correspondência entre as unidades de informação, ou seja, como se mede a informação:

Unidade de medida	Prefixo*	Valor	Quantidade de caracteres	Base binária
1 byte	B	8 bits	1	2^0
1 Kilobyte	kB	1024 B	1.024	2^{10}
1 Megabyte	MB	1024 kB	1.048.576	2^{20}
1 Gigabyte	GB	1024 MB	1.073.741,824	2^{30}
1 Terabyte	TB	1024 GB	1.099.511.627.776	2^{40}
1 Petabyte	PB	1024 TB	1.125.899.906.842.620	2^{50}

* Os prefixos estão de acordo com o SI – SISTEMA INTERNACIONAL DE UNIDADES

Um quilo, nas medidas que usamos cotidianamente, representa o número 1000, que é resultado de uma potência de base 10 (decimal), ou seja, $10^3 = 1000$. Sendo assim, por exemplo, um quilo de queijo é igual a 1000 gramas de queijo.

Um **byte** é uma estrutura fundamentada no código binário, ou seja, na base 2 (binário). Portanto, quando queremos um quilo de bytes, temos que elevar a base 2 a algum número inteiro até conseguir atingir a milhar. Como não há um número inteiro possível que atinja exatamente o valor 1000, então pegamos o próximo número superior a 1000, que é o 1024, ou seja, $2^{10} = 1024$. O número 2 elevado à nona potência é inferior a 1000 ($2^9 = 512$).

2.2. Sistemas de numeração

Este ponto do nosso estudo é importante, pois a utilização de códigos em sistemas com bases diferentes de 10 (decimal) é comum no ambiente de processamento de dados.

Nós estamos acostumados com o sistema decimal (base 10) de numeração, onde temos dez números. Vai de 0 a 9 e depois aumentamos uma casa e repetimos os dois primeiros números, que é o 10. Depois continuamos a repetir o número 1 e trocamos o 0 por 1 até 9, que são os números 11, 12, 13, 14, 15, 16, 17, 18 e 19. Depois vem o 20 e assim por diante.

Com o sistema binário (base 2) é a mesma coisa, só que temos apenas dois números, o zero (0) e o um (1). Vai de 0 a 1 e depois aumentamos uma casa e repetimos os dois primeiros números, resultando no 10. Depois vem o número 11 e como não tem outros algarismos, depois vem o 100, 101, 110, 111, 1000 e assim por diante.

Com o sistema hexadecimal (base 16) é a mesma coisa, só que temos 16 números. Vai de 0 a 9, de A a F e depois aumentamos uma casa e repetimos os dois primeiros números, que é o 10. Depois continuamos a repetir o número 1 e trocamos o 0 por 1 até 9, e depois de A a F, que são os números 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E e 1F. Depois vem o 20 e assim por diante.

Introdução à Lógica de Programação

A seguir temos a tabela de conversão dos sistemas de numeração:

Decimal	Binário	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

2.2.1. Sistema Decimal

Para entender este sistema, basta decompor qualquer número inteiro em potência de base dez. Vejamos um exemplo:

$$\begin{aligned}1024_{(10)} &= 1*(10)^3 + 0*(10)^2 + 2*(10)^1 + 4*(10)^0 \\&= 1*1000 + 0*100 + 2*10 + 4*1 \\&= 1000 + 0 + 20 + 4\end{aligned}$$

Logo, pode-se concluir que o número nada mais é que o conjunto de coeficientes das potências de 10. É importante observar que, por ser base decimal, os dígitos disponíveis são: 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9.

2.2.2. Sistema Binário

Neste sistema, a base é 2 e os dígitos disponíveis são 0 e 1. Este sistema de numeração constitui o alfabeto interno dos computadores.

2.2.3. Sistema Hexadecimal

A base deste sistema é 16 e os dígitos disponíveis são: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E e F. Este sistema simplifica a representação dos números, porque diminui a quantidade de algarismos.

2.3. Conversão de sistemas de numeração

A seguir, veremos como fazer a conversão de Binário para Decimal, de Hexadecimal para Decimal, de Binário para Hexadecimal, de Hexadecimal para Binário, de Decimal para Binário e de Decimal para Hexadecimal.

2.3.1. Conversão de Binário para Decimal

Para fazer esta conversão, utilizaremos a soma das multiplicações das potências de base 2.

Vejamos um exemplo:

$$\begin{aligned} & 10.000.000.000_{(2)} \\ & = 1*(2)^{10} + 0*(2)^9 + 0*(2)^8 + 0*(2)^7 + 0*(2)^6 + 0*(2)^5 + 0*(2)^4 + 0*(2)^3 + 0*(2)^2 + 0*(2)^1 + 0*(2)^0 \\ & = 1*1024 + 0*512 + 0*256 + 0*128 + 0*64 + 0*32 + 0*16 + 0*8 + 0*4 + 0*2 + 0*1 \\ & = 1024 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 \\ & = 1024_{(10)} \end{aligned}$$

Como se pode observar, cada um dos algarismos em binário foi multiplicado por 2 (base 2 = binário) e depois, da direita para a esquerda, o número 2 foi elevado a partir de zero (0), sendo acrescido de um (1).

Vejamos outro exemplo:

$$\begin{aligned} & 110110_{(2)} \\ & = 1*(2)^5 + 1*(2)^4 + 0*(2)^3 + 1*(2)^2 + 1*(2)^1 + 0*(2)^0 \\ & = 1*32 + 1*16 + 0*8 + 1*4 + 1*2 + 0*1 \\ & = 32 + 16 + 0 + 4 + 2 + 0 \\ & = 54_{(10)} \end{aligned}$$

2.3.2. Conversão de Hexadecimal para Decimal

Para fazer esta conversão, utilizaremos a soma das multiplicações das potências de base 16.

Vejamos um exemplo:

$$\begin{aligned}400_{(16)} &= 4*(16)^2 + 0*(16)^1 + 0*(16)^0 \\&= 4*256 + 0*16 + 0*1 \\&= 1024 + 0 + 0 \\&= 1024_{(10)}\end{aligned}$$

Como se pode observar, cada um dos algarismos em hexadecimal foi multiplicado por 16 (base 16 = hexadecimal) e depois, da direita para a esquerda, o número 16 foi elevado a partir de zero (0), sendo acrescido de um (1).

Vejamos outro exemplo:

$$\begin{aligned}17A5_{(16)} &= 1*(16)^3 + 7*(16)^2 + A*(16)^1 + 5*(16)^0 \\&= 1 + 4096 + 7*256 + 10*16 + 5*1 \\&= 4096 + 1792 + 160 + 5 \\&= 6053_{(10)}\end{aligned}$$

Note, na conversão acima, que o número A em hexadecimal foi substituído pelo número 10 em decimal para que a conta fosse feita. Foi utilizado a tabela de conversão dos sistemas de numeração.

2.3.3. Conversão de Binário para Hexadecimal

Para fazer esta conversão, divide-se o número binário em grupos de quatro algarismos, começando da direita para a esquerda. Depois, utilizando a tabela de conversão de sistemas de numeração, substitui-se, então, cada um dos grupos de quatro algarismos por seu correspondente hexadecimal.

Vejamos um exemplo:

1101	0100	1111	0011 ₍₂₎
D	4	F	3 ₍₁₆₎

Caso o último grupo da esquerda não contenha quatro algarismos, basta complementar com zeros. Um número em decimal corresponde a quatro algarismos em binário. Lembre-se de que zeros à esquerda não são significativos.

Vejamos outro exemplo:

O número 1111001111 em binário possui dez algarismos, portanto receberá dois zeros à esquerda para que possamos separar em grupos de quatro algarismos:

0011
3

1100
C

1111 ₍₂₎
F ₍₁₆₎

←utilizando a tabela de conversão de sistemas de numeração

2.3.4. Conversão de Hexadecimal para Binário

Nesta conversão, basta fazer o inverso do que fizemos na conversão de Binário para Hexadecimal. Utilizando a tabela de conversão de sistemas de numeração, substitui-se, então, cada número hexadecimal pelo seu correspondente em binário.

Vejamos um exemplo:

E	7	5	A ₍₁₆₎
1110	0111	0101	1010 ₍₂₎

Vejamos outro exemplo:

1	B	2 ₍₁₆₎
0001	1011	0010 ₍₂₎

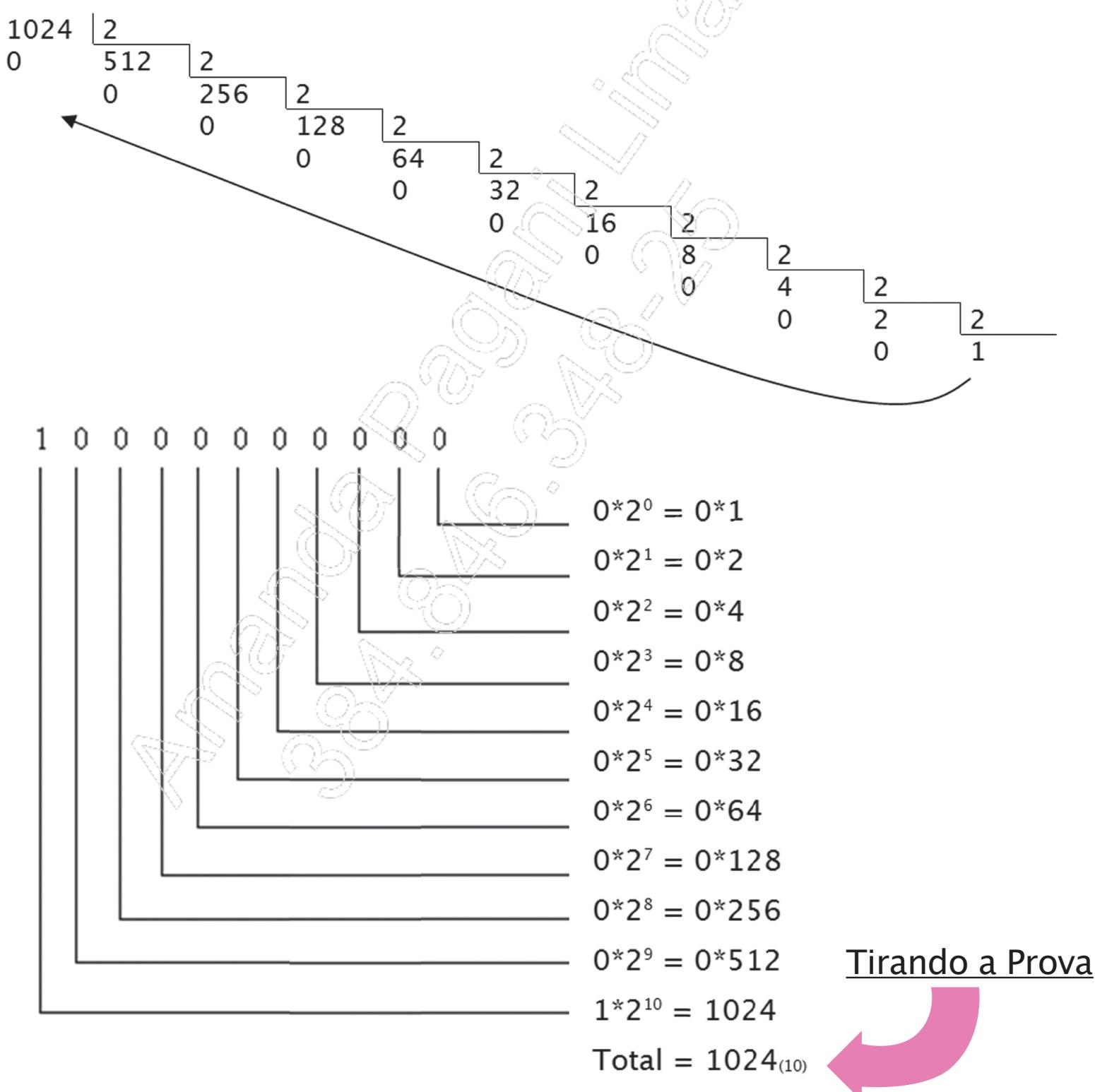
Decimal	Binário	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

2.3.5. Conversão de Decimal para Binário

Para fazer esta conversão, basta realizar sucessivas divisões por dois e, a partir daí, devemos pegar sempre o último quociente, que será sempre 1, e ele será o primeiro número binário, e depois pegamos todos os restos na ordem da última divisão para a primeira.

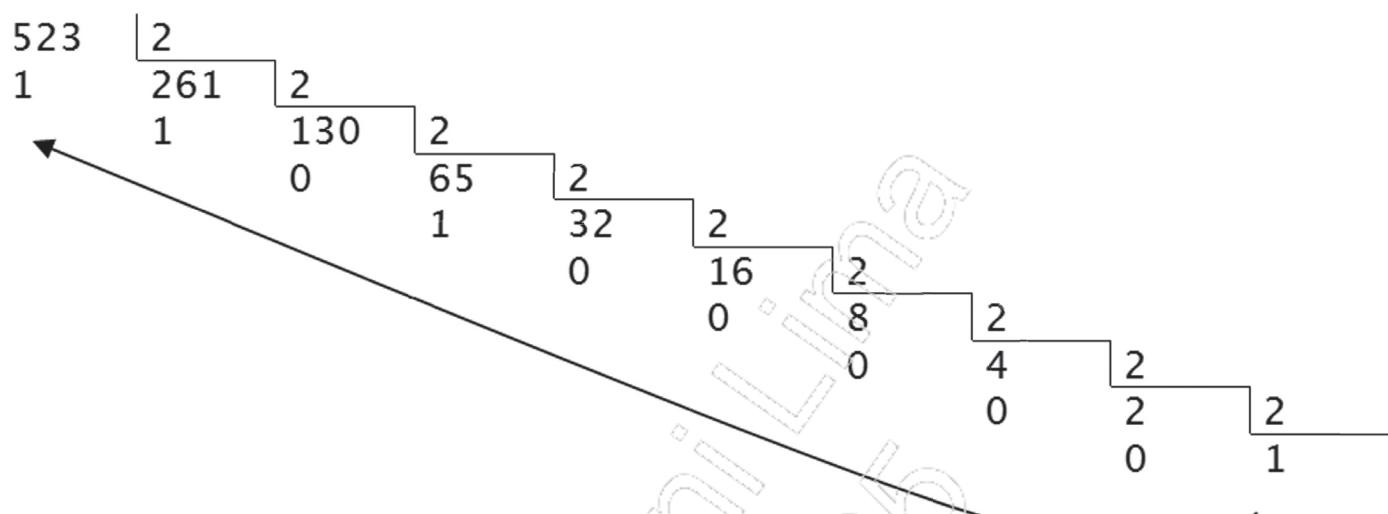
- **Exemplo 1**

$$1.024_{(10)} = 100\ 0000\ 0000_{(2)}$$



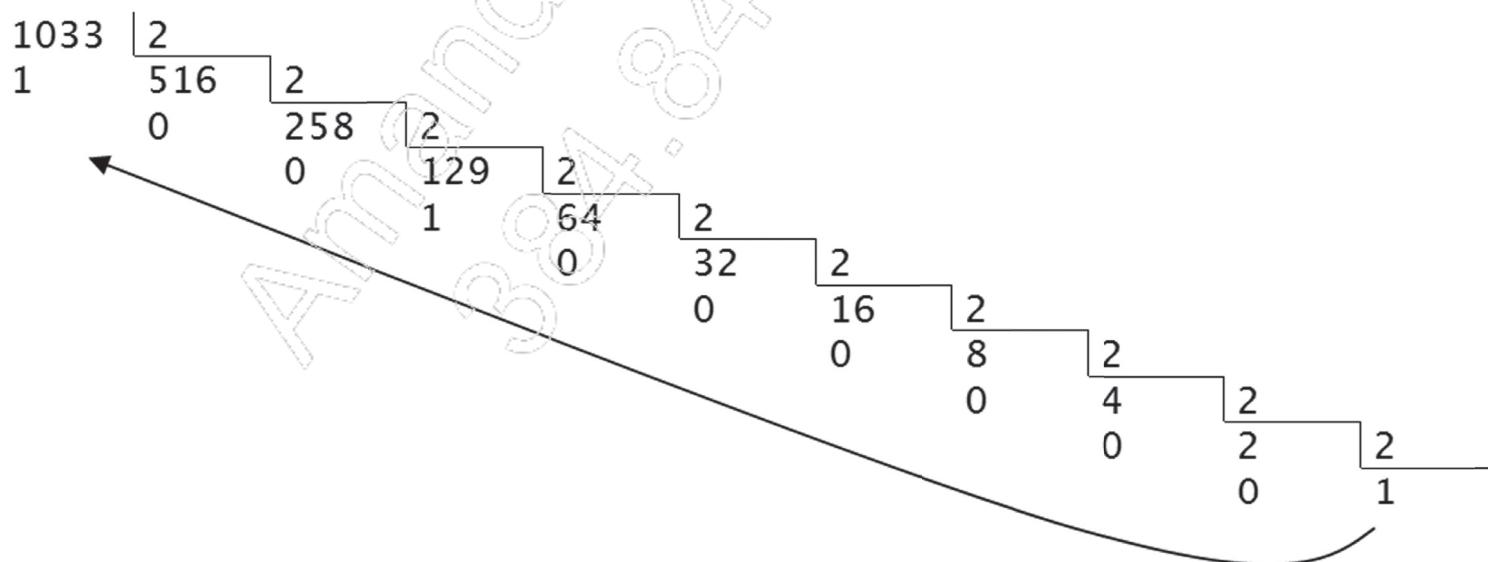
- Exemplo 2

$$523_{(10)} = 10\ 0000\ 1011_{(2)}$$



- Exemplo 3

$$1.033_{(10)} = 100\ 0000\ 1001_{(2)}$$

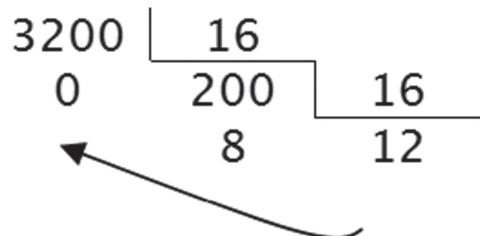


2.3.6. Conversão de Decimal para Hexadecimal

Para converter de decimal para hexadecimal, procede-se do mesmo modo que na conversão decimal para binário. Basta agora dividir por 16 e não mais por 2.

- **Exemplo**

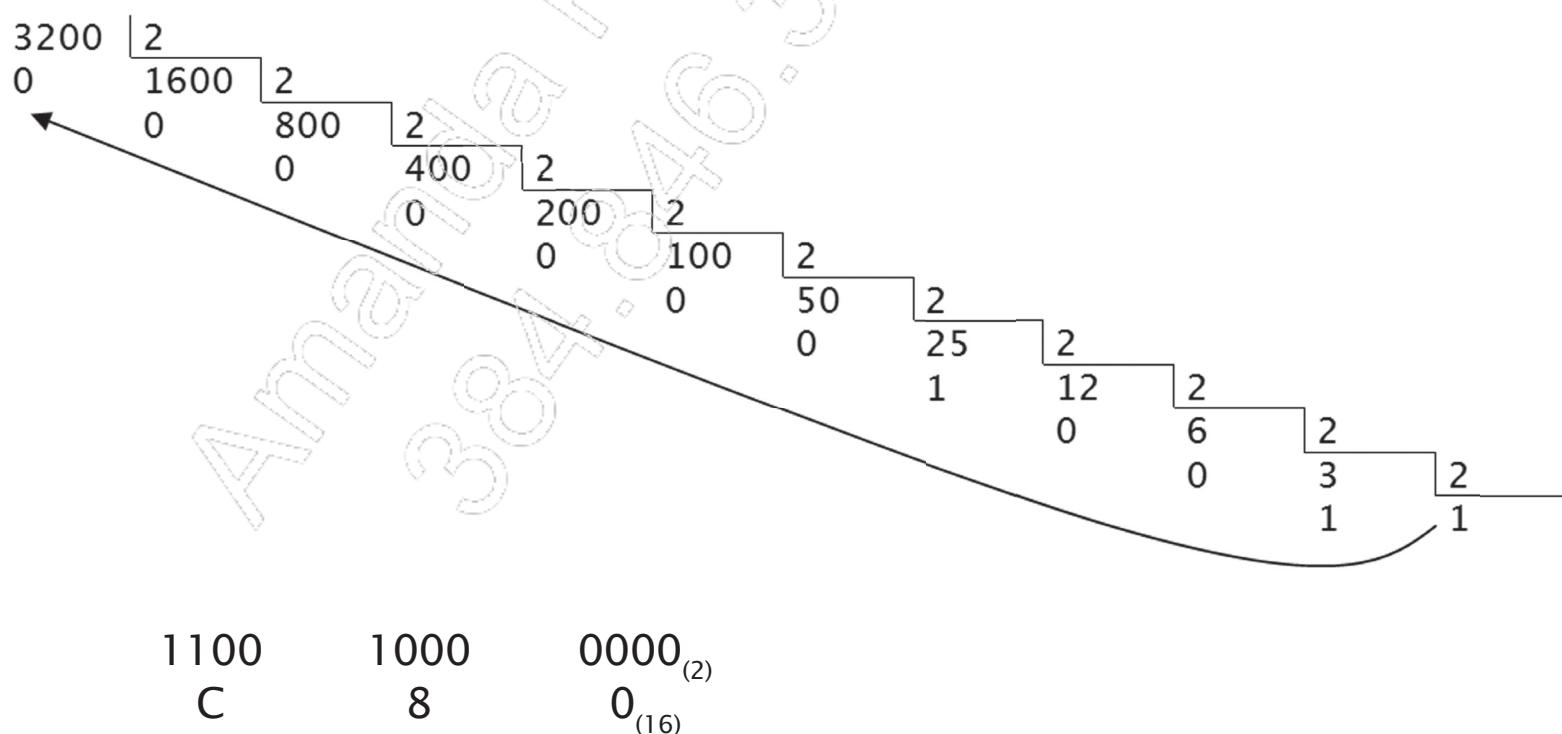
$$3200_{(10)} = C80_{(16)}$$



Na tabela de conversão, $12_{(10)} = C_{(16)}$, portanto $3200_{(10)} = C80_{(16)}$.

Pegaremos sempre o último quociente, pois ele será o primeiro número hexadecimal, e depois pegamos todos os restos na ordem da última divisão para a primeira.

Outra forma de fazer esta conversão é converter primeiro de decimal para binário e, depois, converter de binário para hexadecimal utilizando a tabela de conversão dos sistemas de numeração.



2.4. Forma rápida para conversão de sistemas de numeração

Para converter um número Decimal em Binário, usamos o método de sucessivas divisões por dois, pegando sempre o resto.

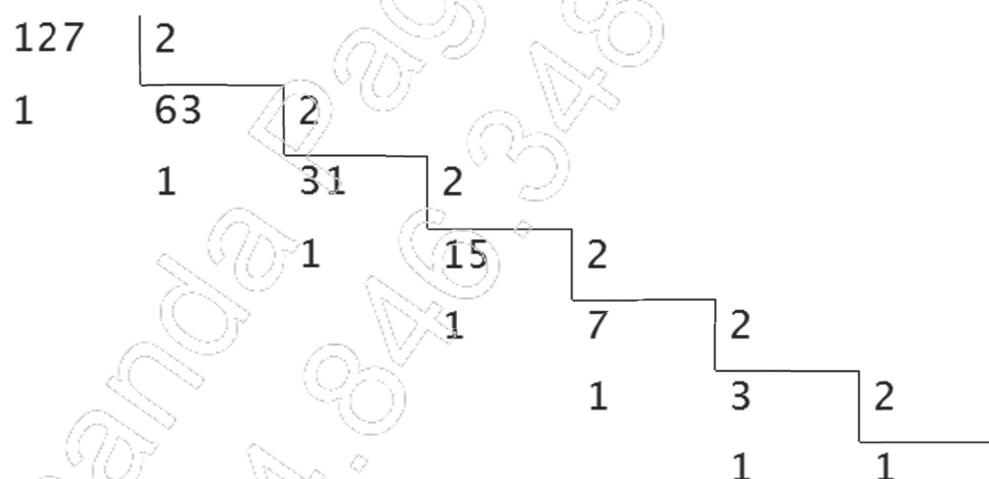
A forma rápida de conversão consiste em escrever numa linha os números da direita para a esquerda a partir do número 1 e, depois, sempre o dobro, ou seja, são as potências de 2.

Depois, na linha de baixo, coloque o número 1 abaixo do mesmo número decimal a ser convertido e preencha os demais números com 0.

- **Exemplo 1**

Qual a correspondência do número $127_{(10)}$ em binário?

$$127_{(10)} = 111\ 1111_{(2)}$$



Usando o esquema seguinte, encontramos uma forma rápida para fazer a conversão. Quando não houver o número a ser convertido na primeira linha, pegue o imediatamente menor, da esquerda para a direita, e complemente na mesma ordem até chegar no valor desejado.

128	64	32	16	8	4	2	1
0	1	1	1	1	1	1	1
	64	32	16	8	4	2	1
	+	+	+				

$= 127$

$$0111\ 1111_{(2)} = 127_{(10)}$$

Introdução à Lógica de Programação

- **Exemplo 2**

Qual a correspondência do número $4100_{(10)}$ em binário?

32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0

4096 +
6

$$4096 + 4 = 4100$$

$$4100_{(10)} = 0001\ 0000\ 0000\ 0100_{(2)} = 1004_{(16)}$$

- **Exemplo 3**

Quanto corresponde o número decimal 4 em binário?

128	64	32	16	8	4	2	1
0	0	0	0	0	1	0	0

$$0100_{(2)} = 4_{(10)}$$

Ou seja, o número 4 em decimal corresponde ao número 100 em binário.

- **Exemplo 4**

Quanto corresponde o número decimal 108 em binário?

128	64	32	16	8	4	2	1
0	1	1	0	1	1	0	0
64	32	0	+ 8	+ 4	+ 0	+ 0	= 108

+ +

$$0110\ 1100_{(2)} = 108_{(10)}$$

Ou seja, o número 108 em decimal corresponde ao número 01101100 em binário.

Como não há exatamente o número 108 na linha de cima, pegamos o 64 e colocamos 1 abaixo.

Vemos quanto sobrou e continuamos a fazer a mesma coisa: $108 - 64 = 44$. Como não há exatamente o número 44 na linha de cima, pegamos o 32 e colocamos 1 abaixo.

Vemos quanto sobrou e continuamos a fazer a mesma coisa: $44 - 32 = 12$. Como não há exatamente o número 12 na linha de cima, pegamos o 8 e colocamos 1 abaixo.

Vemos quanto sobrou e continuamos a fazer a mesma coisa: $12 - 8 = 4$. Como há exatamente o número 4 na linha de cima, colocamos 1 abaixo e terminamos.

A seguir, temos a tabela com as potências de 2 até o expoente 20 e com as potências de 16 até o expoente 5:

Potência de 2 = Valor	Potência de 16 = Valor
$2^0 = 1$	$16^0 = 1$
$2^1 = 2$	
$2^2 = 4$	
$2^3 = 8$	
$2^4 = 16$	$16^1 = 16$
$2^5 = 32$	
$2^6 = 64$	
$2^7 = 128$	
$2^8 = 256$	$16^2 = 256$
$2^9 = 512$	
$2^{10} = 1024$	
$2^{11} = 2048$	
$2^{12} = 4096$	$16^3 = 4096$
$2^{13} = 8192$	
$2^{14} = 16\,384$	
$2^{15} = 32\,768$	
$2^{16} = 65.536$	$16^4 = 65.536$
$2^{17} = 131.072$	
$2^{18} = 262.144$	
$2^{19} = 524.288$	
$2^{20} = 1.048.576$	$16^5 = 1.048.576$

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- O **bit** é a menor unidade de informação do computador;
- É importante termos a tabela de conversão dos sistemas de numeração, pois a utilização de códigos em sistemas com bases diferentes de 10 (decimal) é comum no ambiente de processamento de dados;
- Os sistemas de numeração mais usados são: **Decimal, Binário e Hexadecimal**;
- Para entender o sistema **Decimal**, basta decompor qualquer número inteiro em potência de base dez;
- No sistema **Binário**, a base é 2 e os dígitos disponíveis são 0 e 1. Este sistema de numeração constitui o alfabeto interno dos computadores;
- A base do sistema **Hexadecimal** é 16 e os dígitos disponíveis são: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E e F. Este sistema simplifica a representação dos números, porque diminui a quantidade de algarismos;
- Podemos fazer conversões de Binário para Decimal, de Hexadecimal para Decimal, de Binário para Hexadecimal, de Hexadecimal para Binário, de Decimal para Binário e de Decimal para Hexadecimal.

Sistemas de numeração

Teste seus conhecimentos

2

Amanhã p/ amanhã
384.846.48-25



IMPACTA
EDITORA

Introdução à Lógica de Programação

1. Qual a representação decimal do número $101011_{(2)}$?

- $40_{(10)}$
- $42_{(10)}$
- $43_{(10)}$
- $47_{(10)}$
- Nenhuma das alternativas anteriores está correta.

2. Qual a representação decimal do número $A0C_{(16)}$?

- $2560_{(10)}$
- $2572_{(10)}$
- $2584_{(10)}$
- $2588_{(10)}$
- Nenhuma das alternativas anteriores está correta.

3. Qual a representação hexadecimal do número $1011011110_{(2)}$?

- 1DE₍₁₆₎
- 2EF₍₁₆₎
- 2DE₍₁₆₎
- 189₍₁₆₎
- Nenhuma das alternativas anteriores está correta.

4. Qual a representação binária do número 8BF₍₁₆₎?

- 1000 1011 1111₍₂₎
- 1001 1011 1110₍₂₎
- 1001 1010 1100₍₂₎
- 1000 1010 1111₍₂₎
- Nenhuma das alternativas anteriores está correta.

Introdução à Lógica de Programação

5. Qual a representação binária do número $333_{(10)}$?

- $1\ 1011\ 1111_{(2)}$
- $1\ 0011\ 1101_{(2)}$
- $1\ 1010\ 1100_{(2)}$
- $1\ 0100\ 1101_{(2)}$
- Nenhuma das alternativas anteriores está correta.

Sistemas de numeração

Mãos à obra!

2

Amanhã
Bragagni Lima
384.000-25



IMPACTA
EDITORA

Laboratório 1

Exercício 1

Qual a representação decimal dos números em binário a seguir?

- $11000111_{(2)}$
- $101011000_{(2)}$

Exercício 2

Qual a representação decimal dos números em hexadecimal a seguir?

- $2E_{(16)}$
- $12F_{(16)}$

Exercício 3

Qual a representação hexadecimal dos números em binário a seguir?

- $1011101000111100_{(2)}$
- $1001011100100000101_{(2)}$

Exercício 4

Qual a representação binária dos números em hexadecimal a seguir?

- 9 C E $0_{(16)}$
- 6 0 D 3 5 $_{(16)}$

Exercício 5

Escreva a representação binária dos números em decimal a seguir:

- $221_{(10)}$
- $1000_{(10)}$

Exercício 6

Qual a representação hexadecimal dos números em decimal a seguir?

- $485_{(10)}$
- $1600_{(10)}$

Exercício 7

Converta os seguintes números decimais para binários:

	256	128	64	32	16	8	4	2	1
a)	16								
b)	26								
c)	31								
d)	100								
e)	167								
f)	200								
g)	221								
h)	256								
i)	281								
j)	300								

Introdução à Lógica de Programação

Exercício 8

Converta os seguintes números hexadecimais para binários e para decimais:

- a) \$05
 - b) \$18
 - c) \$29
 - d) \$3F
 - e) \$FF
 - f) \$ED
 - g) \$FB
 - h) \$2A
 - i) \$81
 - j) \$170
 - k)\$44C
 - l)\$10A

3

Algoritmo Mãos à obra!

Amanda Paganini Lima
384.846.348-25



IMPACTA
EDITORA

Laboratório 1

Exercício 1

Escreva um algoritmo para fazer uma ligação telefônica a partir de um telefone fixo que já está devidamente ligado e conectado.

Exercício 2

O processo do algoritmo a seguir tem o objetivo de retirar bolas de uma caixa, sem precisar olhar para elas, até que se consiga 20 bolas de uma mesma cor. Copie o algoritmo a seguir utilizando decisões encadeadas, substituindo o comando **CASO** pelo comando **SE**.

Início

Enquanto todos os contadores de bolas forem menores que 20

 Retirar uma bola

 Caso

 Caso a bola seja verde: Somar 1 no contador de bolas verdes

 Caso a bola seja amarela: Somar 1 no contador de bolas amarelas

 Caso a bola seja azul: Somar 1 no contador de bolas azuis

 Caso a bola seja branca: Somar 1 no contador de bolas brancas

 Caso contrário: Somar 1 no contador de bolas pretas

 Fim do caso

Enquanto

Totalizar os contadores

Exibir valores

FIM

Algoritmo

3

- ✓ Algoritmo;
- ✓ Elementos de um algoritmo;
- ✓ Algoritmo com o comando SE;
- ✓ Algoritmo com o comando CASO;
- ✓ Algoritmo com o comando ENQUANTO.



IMPACTA
EDITORA

3.1. Algoritmo

Algoritmo é a descrição sequencial ordenada dos passos que devem ser executados, de forma lógica e clara, com a finalidade de facilitar a resolução de um problema.

Você viu anteriormente o passo a passo da rotina do início do dia de uma pessoa. Então, é o algoritmo que ajuda a resolver o que fazer desde a hora de acordar até chegar ao local de trabalho.

A seguir temos um algoritmo, com numeração nas linhas, de **como fritar um ovo**.

Detalhes: Um ovo, uma lata de óleo, um saleiro, um prato, uma colher e uma caixa de fósforos estão ao lado do fogão, em cima de uma pia. Tem gás e a frigideira já está em cima de uma boca do fogão.

1. Início
2. Acender o fogo
3. Colocar óleo na frigideira
4. O óleo está quente?
 - 4.1. Se sim: “Próximo passo”
 - 4.2. Se não: “Esperar o óleo esquentar” e
“Vá para o passo 4”
5. Quebrar o ovo
6. O ovo está bom?
 - 6.1. Se sim: “Próximo passo”
 - 6.2. Se não: “Vá para o passo 10”
7. Colocar o ovo na frigideira
8. Colocar uma pitada de sal
9. O ovo está frito?
 - 9.1. Se sim: “Próximo passo”
 - 9.2. Se não: “Esperar fritar” e
“Vá para o passo 9”
10. Pegar o ovo com a colher e colocar no prato
11. Desligar o fogo
12. Fim

A seguir temos um algoritmo de **como trocar um pneu**.

Detalhes: O carro está estacionado num lugar seguro e quem vai trocar o pneu já está ao lado do porta-malas com as chaves do carro.

1. Início
2. Abrir o porta-malas do carro
3. Tem estepe?
 - 3.1. Se sim: Retirar o estepe
 - 3.2. Se não: Vá para o passo 18
4. Tem ferramentas?
 - 4.1. Se sim: Pegar as ferramentas
 - 4.2. Se não: Vá para o passo 17
5. Desapertar os parafusos da roda
6. Já desapertou todos os parafusos?
 - 6.1. Se sim: Vá para o passo 7
 - 6.2. Se não: Vá para o passo 5
7. Levantar o carro com o macaco
8. Levantou o suficiente?
 - 8.1. Se sim: Próximo passo
 - 8.2. Se não: Vá para o passo 7
9. Retirar os parafusos
10. Retirar a roda
11. Colocar o estepe
12. Recolocar os parafusos
13. Abaixar o carro
14. Apertar os parafusos
15. Já apertou todos os parafusos?
 - 15.1. Se sim: Vá para o passo 16
 - 15.2. Se não: Vá para o passo 14
16. Guardar as ferramentas
17. Guardar o pneu
18. Fechar o porta-malas
19. Fim

Introdução à Lógica de Programação

Os algoritmos de ações cotidianas geralmente permitem que uma pessoa o execute mesmo que ela nunca tenha feito tal coisa.

Uma receita de um bolo é um exemplo. Você pode fazer o bolo se seguir corretamente a receita. Agora, se a receita não estiver detalhada o suficiente, já não se pode garantir o resultado.

Um algoritmo detalhado mostra ações importantes como **Desligar o fogo**, no primeiro algoritmo, ou **Guardar o pneu**, no segundo algoritmo.

3.2. Elementos de um algoritmo

Veremos, a seguir, os elementos que compõem um algoritmo.

3.2.1. Ação

- Abrir o porta-malas do carro
- Retirar os parafusos
- Retirar a roda
- Colocar o estepe
- Recolocar os parafusos
- Abaixar o carro
- Guardar as ferramentas
- Guardar o pneu
- Fechar o porta-malas

3.2.2. Decisão

Tem estepe?

Se sim: Retirar o estepe

Se não: Vá para o passo 18

Tem ferramentas?

Se sim: Pegar ferramentas

Se não: Vá para o passo 17

Observação: Note que, numa decisão, existem duas respostas ou caminhos a seguir: o lado verdadeiro e o lado falso da indagação.

3.2.3. Laço ou Loop

5. Desapertar os parafusos da roda

6. Já desapertou todos os parafusos?

6.1. Se sim: Vá para o passo 7

6.2. Se não: Vá para o passo 5

Observação: Note que, se não foram desapertados todos os parafusos, a execução do programa volta para a linha 5, fazendo um loop, que são trechos de uma lógica que podem ser executados várias vezes.

Introdução à Lógica de Programação

7. Levantar o carro com o macaco
8. Levantou o suficiente?
 - 8.1. Se sim: Próximo passo
 - 8.2. Se não: Vá para o passo 7

Observação: Note que, se o carro não foi suficientemente levantado, a execução do programa volta para a linha 7, fazendo um loop. Observe também que, em caso positivo, a execução do programa vai para o **Próximo passo**, que seria a mesma coisa que **Ir para o passo 9**, que é o próximo passo.

14. Apertar os parafusos
15. Já apertou todos os parafusos?
 - 15.1. Se sim: Vá para o passo 16
 - 15.2. Se não: Vá para o passo 14

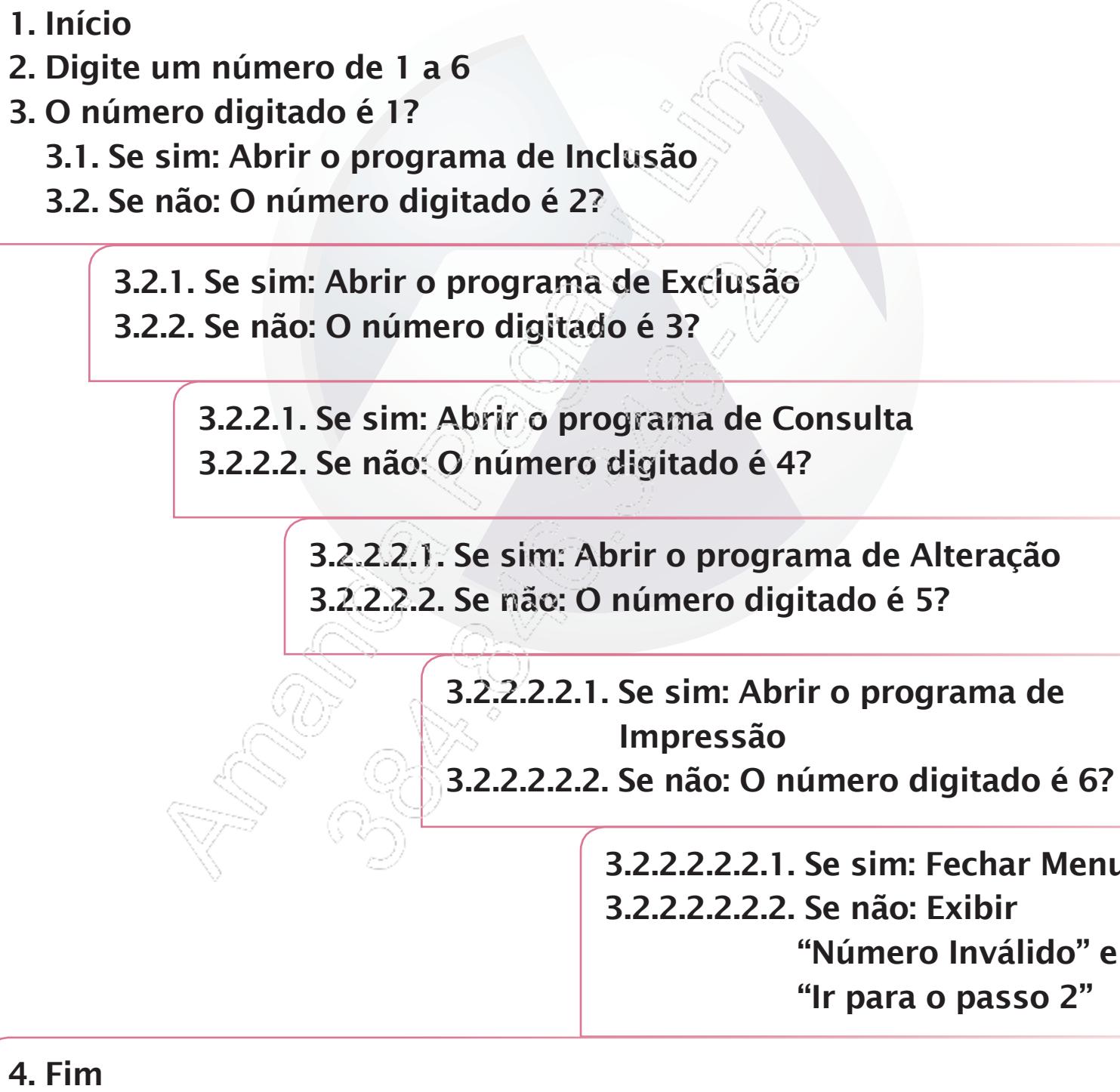
Observação: Note que, se não foram apertados todos os parafusos, a execução do programa volta para a linha 14, fazendo um loop.

3.3. Algoritmo com o comando SE encadeado

Já vimos que o comando **SE** tem duas respostas, **SE SIM** e **SE NÃO**, mas pode ser que você precise de mais alternativas como resposta.

A solução é **encadear** o comando **SE**, colocando um dentro do outro.

- Exemplo: Menu principal de um programa



3.4. Algoritmo com o comando CASO

Já vimos que o comando **SE** tem duas respostas, **SE SIM** e **SE NÃO**. O comando **CASO** pode ter quantas respostas você precisar. Ele é usado como alternativa ao comando **SE** encadeado.

Observação: Nem todas as linguagens tem o comando **CASO**, e somente no comando **SE** existe **SE SIM** e **SE NÃO**.

- **Exemplo: Menu principal de um programa**

1. Início
2. Digite um número de 1 a 6
3. Caso
 - 3.1. Caso digitou 1: Abrir o programa de Inclusão
 - 3.2. Caso digitou 2: Abrir o programa de Exclusão
 - 3.3. Caso digitou 3: Abrir o programa de Consulta
 - 3.4. Caso digitou 4: Abrir o programa de Alteração
 - 3.5. Caso digitou 5: Abrir o programa de Impressão
 - 3.6. Caso digitou 6: Fechar Menu
 - 3.7. Caso contrário: Exibir “Número Inválido” e “Ir para o passo 2”
4. Fim do Caso
5. Fim

Observação: Note que na linha 4, o comando **Fim do Caso** serve pra indicar onde termina o comando **CASO**. Depois o programa continua normalmente a partir da próxima linha, que, neste caso, é o comando **Fim**, que serve para identificar o final do algoritmo.

3.5. Algoritmo com o comando ENQUANTO

O comando **ENQUANTO**, que em inglês significa **WHILE**, serve para que uma parte do programa seja repetida enquanto uma situação for satisfeita. O loop que repete as linhas de programação só é interrompido quando a condição que o faz repetir os comandos não for mais verdadeira.

Exemplo: Vamos supor que exista uma caixa com 30 bolas, sendo 10 verdes, 10 amarelas e 10 azuis. O objetivo é retirar bolas da caixa, sem olhar, até ter certeza de que saíram 3 bolas de cor azul. Para facilitar utilizaremos contadores de bolas, que inicialmente têm o valor zero (0), porque ainda nenhuma bola foi retirada.

Início

Enquanto o contador de bolas azuis for menor que 03

Retirar uma bola

Caso

Caso a bola seja verde: Somar 1 no contador de bolas verdes

Caso a bola seja amarela: Somar 1 no contador de bolas amarelas

Caso contrário: Somar 1 no contador de bolas azuis

Fim do Caso

Fim do Enquanto

Exibir a mensagem “Três bolas azuis foram retiradas da caixa”.

Fim

Note que, dentro do comando **ENQUANTO**, foram colocados os comandos de retirar uma bola e o comando **CASO** que verifica a cor das bolas somando 1 no contador da respectiva cor. Após o comando **Caso contrário: Somar 1 no contador de bolas azuis**, a execução volta diretamente para o comando **Enquanto o contador de bolas azuis for menor que 03**.

Após esta verificação, caso o contador de bolas azuis não seja então menor que 3, a execução vai para **Exibir a mensagem Três bolas azuis foram retiradas da caixa**. Caso contrário, a execução entra novamente no loop e retira mais uma bola.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- **Algoritmo** é a descrição sequencial ordenada dos passos que devem ser executados, de forma lógica e clara, com a finalidade de facilitar a resolução de um problema;
- Os algoritmos são compostos por ação, decisão e laço ou loop;
- O comando **ENQUANTO (WHILE)** serve para que uma parte do programa seja repetida enquanto uma situação for satisfeita.

Variáveis, Operadores e Funções

4

- ✓ Utilização de variáveis;
- ✓ Tipos de variáveis;
- ✓ Nome de variáveis;
- ✓ Declaração de variáveis;
- ✓ Comando de atribuição;
- ✓ Constantes;
- ✓ Operadores aritméticos;
- ✓ Operadores relacionais;
- ✓ Operadores lógicos;
- ✓ Função;
- ✓ Concatenação de alfanuméricos.



IMPACTA
EDITORA

4.1. Introdução

Variáveis são áreas na memória, utilizadas em programação, que servem para armazenar dados. O conteúdo de uma variável pode ser alterado, mas uma variável só pode conter um dado por vez.

As áreas na memória são divisões na memória. O computador identifica cada divisão por meio de um endereço no formato hexadecimal, ou seja, para facilitar a localização dos dados, as variáveis são encontradas pelos endereços de memória, assim como uma casa é encontrada pelo seu endereço. Utilizamos variáveis frequentemente em programação.

4.2. Utilizando variáveis

A seguir, apresentamos as situações em que utilizamos variáveis.

4.2.1. Consistência de condições

Com as variáveis podemos verificar a veracidade, ou não, de uma condição para assim obtermos um ou outro resultado. Se, entre várias opções, desejarmos escolher uma alternativa, podemos comparar a escolha com uma variável previamente definida e fazer o programa executar uma ou outra determinada sequência de comandos.

4.2.2. Controle de repetições

As variáveis de memória podem ser usadas para o controle de repetições. Se escrevermos uma sequência de comandos dentro de um programa e desejarmos que esta sequência seja repetida um certo número de vezes, podemos utilizar uma variável numérica e, a cada passagem, aumentar ou diminuir seu valor, de modo que, quando atingido um determinado valor, a sequência de comandos pare de ser executada.

4.2.3. Comparações de variáveis de memória com campos de registros

Quando trabalhamos com arquivos de banco de dados, podemos fazer comparações, trocas e procuras de registros através das variáveis de memória. As variáveis ficam armazenadas na memória RAM do computador, enquanto **campos** de um **registro** pertencem ao banco de dados correspondente e, portanto, estão gravados no disco (ou em outro meio de armazenamento) do computador.

4.3. Tipos de variáveis

Os tipos e valores dos conteúdos das variáveis podem variar de linguagem para linguagem. Vejamos:

- **Alfanumérica:** Atribuímos qualquer tipo de caractere a elas, ou seja, letras, números ou sinais;
- **Numérica:** Atribuímos somente números a elas, pois podemos usá-las para efetuar cálculos;
- **Data:** Atribuímos somente datas a elas;
- **Lógica:** Atribuímos somente valores verdadeiros ou falsos (V/F). São utilizadas para testes lógicos;
- **Objeto:** Atribuímos uma referência a um objeto.

4.4. Nomes de variáveis

Para atribuir um nome a uma variável, devemos observar alguns aspectos:

- Não é possível começar um nome de variável com número;

Certo	Errado
X	
Teste	
ABC2	1ABC
J34CD	2Parcela
Texto3	3Fase
VAR1	
VNome	

- Só é permitido o uso de underline (_) no nome. Espaço ou qualquer outro sinal é proibido;

Certo	Errado
Salariofixo	
Nota_Final	Tudo% Qualquer\$valor Valor
Valor_1	da Compra

- Para maior facilidade, devemos usar sempre nomes autoexplicativos;
- Devemos, no entanto, cuidar para que as variáveis não tenham nomes de comandos, funções ou campos de um banco de dados, pois isto pode causar problemas durante a execução do programa.

4.5. Declaração de variáveis

As variáveis precisam ser **declaradas**. Para declarar uma variável, devemos informar o nome e o tipo da variável.

Cada um dos tipos de variáveis ocupa espaços diferentes de memória, de acordo com os dados que irão armazenar. A definição do tipo de variável otimiza a utilização do espaço de memória.

Normalmente a variável é declarada no início do programa para que possa ser utilizada no programa inteiro, sendo assim, a declaramos no início do algoritmo.

Veja, a seguir, exemplos de como iremos declarar as variáveis nos algoritmos:

Declara A, B, C numéricas, D, E alfanuméricas

Declara VALORX numérica, TEXTOY alfanumérica

Declara NOTA_BIMESTRAL1, NOTA_BIMESTRAL2 numéricas

Declara MENSAGEM5, MENSAGEM9 alfanuméricas, HOJE data

4.6. Comando de atribuição

O **comando de atribuição** serve para armazenar um valor numa variável. As variáveis numéricas que serão utilizadas no programa e não têm um valor inicial predefinido geralmente recebem o valor zero (0).

Utilizaremos o sinal de igual (=) para representar o comando de atribuição. Onde tiver =, lê-se **recebe**:

```
A = 10  
X = 0  
Valor = 0  
Nota = 7,5  
Nome_Aluno = "Leandro"  
MensagemFinal = "Boa noite."
```

4.7. Constantes

Ao contrário das variáveis, as **constantes** possuem valor fixo e não sofrem alteração durante o processamento.

4.8. Operadores aritméticos

Em programação, usamos os mesmos **operadores aritméticos** que são usados na matemática, e com as mesmas prioridades.

Em primeiro lugar são executadas as exponenciações e as radiciações, em segundo as multiplicações e as divisões, e depois as adições e as subtrações.

OPERAÇÃO	OPERADOR USADO	PRIORIDADE
Radiciação	//	1º
Exponenciação	^ ou **	1º
Multiplicação	*	2º
Divisão	/	2º
Adição	+	3º
Subtração	-	3º

Usamos também os seguintes operadores para operações matemáticas não-convencionais:

OPERAÇÃO	OPERADOR USADO	PRIORIDADE
Resto da divisão	Mod	2º
Quociente da divisão inteira	Div	2º

Normalmente, as operações são executadas da esquerda para a direita, a não ser que mudemos sua ordem usando parênteses.

Para a construção de algoritmos que contêm fórmulas matemáticas, todas as expressões aritméticas devem ser linearizadas, ou seja, colocadas em linha.

Introdução à Lógica de Programação

Veja, a seguir, exemplos de como é uma fórmula na matemática tradicional e como ela fica linearizada.

- **Exemplo 1**

$$2 \times \frac{5 + 3}{2} = 2 * (5 + 3)/2$$

- **Exemplo 2**

$$4 + 5^2 - (2 \times 3) = 4 + 5^2 - 2 * 3$$

- **Exemplo 3**

$$\sqrt{25} - \sqrt[3]{125} + 5 = 25//2 - 125//3 + 5$$

- **Exemplo 4**

$$3 \times 4 + 5 - \frac{125}{5^3} = 3 * 4 + 5 - 125 / 5^{**3}$$

- **Exemplo 5**

$$3 \times \left((4 + 5) - \left(\frac{25}{5} \right) \right)^2 = 3 * ((4+5)-(25/5)) ** 2$$

Veja, a seguir, o valor de cada variável com a resolução das fórmulas matemáticas respeitando a prioridade de execução dos operadores.

- **Exemplo 1**

$$\begin{aligned} A &= 2 * (5 + 3) / 2 \\ A &= 2 * 8 / 2 \\ A &= 16 / 2 \\ A &= 8 \end{aligned}$$

- **Exemplo 2**

$$\begin{aligned} B &= 4 + 5 ^ 2 - 2 * 3 \\ B &= 4 + 25 - 2 * 3 \\ B &= 4 + 25 - 6 \\ B &= 29 - 6 \\ B &= 23 \end{aligned}$$

- **Exemplo 3**

$$\begin{aligned} C &= 25 // 2 - 125 // 3 + 5 \\ C &= 5 - 125 // 3 + 5 \\ C &= 5 - 5 + 5 \\ C &= 5 \end{aligned}$$

- **Exemplo 4**

$$\begin{aligned} D &= 3 * 4 + 5 - 125 / 5 ** 3 \\ D &= 3 * 4 + 5 - 125 / 125 \\ D &= 12 + 5 - 125 / 125 \\ D &= 12 + 5 - 1 \\ D &= 17 - 1 \\ D &= 16 \end{aligned}$$

Introdução à Lógica de Programação

- **Exemplo 5**

```
E = 3 * ((4 + 5) - (25/5)) ** 2  
E = 3 * (9 - (25/5)) ** 2  
E = 3 * (9 - 5) ** 2  
E = 3 * 4 ** 2  
E = 3 * 16  
E = 48
```

- **Exemplo 6**

F = 7 / 3	→	F = 2,3333...3
G = 7 DIV 3	→	G = 2
H = 7 MOD 3	→	H = 1

$$\begin{array}{ccc} 7 & | & 3 \\ 7 \text{MOD}3 & \leftarrow & 1 \quad 2 \quad \rightarrow 7\text{DIV}3 \end{array}$$

Portanto:

- 7 MOD 3 mostra como resultado o número 1;
- 7 DIV 3 mostra como resultado o número 2.

- Exemplo 7

```
J = 1234 / 10      → J = 123,4  
K = 1234 / 100     → K = 12,34  
L = 1234 / 1000    → L = 1,234  
  
M = 1234 DIV 10    → M = 123  
N = 1234 DIV 100   → N = 12  
P = 1234 DIV 1000  → P = 1  
  
Q = 1234 MOD 10    → Q = 4  
R = 1234 MOD 100   → R = 34  
S = 1234 MOD 1000  → S = 234  
  
T = 1234 DIV 10 DIV 10 → T = 12  
U = 1234 MOD 10 * 10 → U = 40  
  
V = 1234 MOD 1000 DIV 100 → V = 2  
W = 1234 DIV 100 / 6 DIV 2 → W = 1
```

- Exemplo 8

```
NRO = 457  
UNIDADE = NRO MOD 10    → UNIDADE = 7  
DEZENA = NRO DIV 10 MOD 10 → DEZENA = 5  
CENTENA = NRO DIV 100    → CENTENA = 4
```

4.8.1. Contadores e acumuladores

- **Contador**

Um **contador** é construído a partir de uma variável que recebe o valor dela mesma mais outro valor.

A linha que descreve um contador A é:

$$A = A + 1$$

Nesse exemplo a variável A está recebendo o valor dela mesma mais 1, ou seja, está contando de 1 em 1. O valor 1 é uma constante.

- **Acumulador**

Um **acumulador** é uma variável que recebe o valor dela mesma mais o valor de outra variável.

A linha que descreve um acumulador B de A é:

$$B = B + A$$

Nesse exemplo, a variável B está recebendo o valor dela mesma mais o valor da variável A. A variável A representa o valor a ser somado, acumulado na variável B.

- **Acumulador de Conta**

A linha que descreve um acumulador C do dobro de B é:

$$C = C + 2 * B$$

Nesse exemplo, a variável C está recebendo o valor dela mesma mais duas vezes o valor da variável B.

4.9. Operadores relacionais

Quando encontramos situações nas quais é necessário usar condições, adotamos os seguintes operadores para estabelecer relações:

OPERAÇÃO	OPERADOR USADO
Igual a	=
Maior que	>
Menor que	<
Maior ou igual a	>=
Menor ou igual a	<=
Diferente de	<>

- Exemplos

Num algoritmo:

```

Se o Cargo for igual a Gerente
  Então conceder um aumento de 10% no salário
  Senão conceder um aumento de 5% no salário
Fim Se

```

Observação: Note que o comando **FIM SE** acima foi colocado pra identificar o fim do comando **SE**.

Numa linguagem de programação:

```

F Cargo = "Gerente"
  Salario = Salario * 1.10
ELSE
  Salario = Salario * 1.05
ENDIF

```

4.10. Operadores lógicos

São usados quando existem duas ou mais condições em um teste lógico, e assim como os operadores de comparação, retornam um resultado **VERDADEIRO** ou **FALSO**:

- **NÃO (NOT)** – Ordem de Prioridade: 1^a

Operador lógico que inverte a lógica de uma expressão. Se ela for verdadeira, torna-se falsa, e vice-versa.

- **E (AND)** – Ordem de Prioridade: 2^a

Operador lógico em que a resposta da operação é verdadeira somente se as duas condições forem verdadeiras.

```
Se o Cargo for igual a Gerente E a Idade for maior ou igual a 50 anos
    Então conceder um aumento de 10% no salário
    Senão conceder um aumento de 5% no salário
Fim Se
```

- **OU (OR)** – Ordem de Prioridade: 3^a

Operador lógico em que a resposta da operação é verdadeira se pelo menos uma das condições for verdadeira.

```
Se o cargo for igual a Gerente OU a Idade for maior ou igual a 50 anos
    Então conceder um aumento de 10% no salário
    Senão conceder um aumento de 5% no salário
Fim Se
```

4.10.1. Tabela de decisão

Partindo de uma condição, podemos tomar uma ou outra decisão.

Exemplo:

**Se estiver com fome irei almoçar.
Senão não irei almoçar.**

1^a decisão: Se estiver com fome irei almoçar



2^a decisão: Se estiver com fome irei almoçar



Observe que podemos escrever na negativa e o resultado será o mesmo.

**Se NÃO estiver com fome NÃO irei almoçar.
Senão irei almoçar.**

1^a decisão: Se não estiver com fome, não irei almoçar



2^a decisão: Se não estiver com fome, não irei almoçar



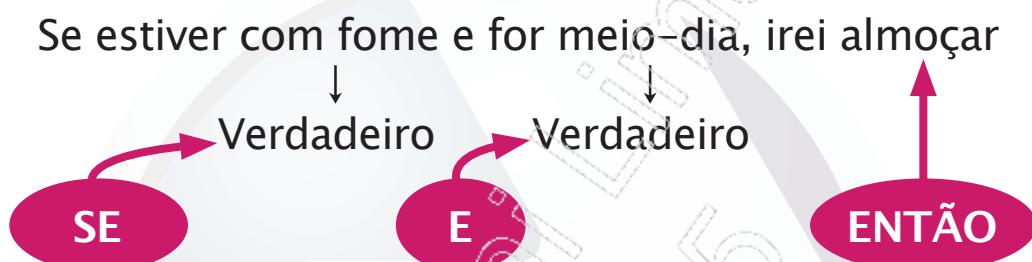
Introdução à Lógica de Programação

Analisamos, até agora, somente uma condição para que tomássemos uma ou outra decisão.

A **TABELA DE DECISÃO** também é conhecida como **TABELA VERDADE**.

Vamos criar uma tabela unindo duas condições, usando o operador **E**.

Vamos supor, então, que o almoço só é servido após o meio-dia:



Somente quando as duas primeiras proposições forem verdadeiras, a terceira também será.

Se	estiver com fome	e	for meio-dia	então	irei almoçar
	V	+	V	=	V
	V	+	F	=	F
	F	+	V	=	F
	F	+	F	=	F

Usando o operador **OU**, basta que apenas uma das duas primeiras proposições seja verdadeira para que a terceira também seja.

Se	estiver com fome	ou	for meio-dia	então	irei almoçar
	V	+	V	=	V
	V	+	F	=	V
	F	+	V	=	V
	F	+	F	=	F

4.10.2. Tabela de decisão com números binários

Neste caso o número zero (0) tem valor **FALSO** e o número um (1) tem valor **VERDADEIRO**.

Usando o operador **E**, somente quando as duas primeiras proposições forem verdadeiras, a terceira também será.

Somente quando os dois primeiros números forem “um” (1), o terceiro também será.

1	1	0	0
1	0	1	0
1	0	0	0

→ Estiver com fome E
→ For meio dia
→ Irei almoçar

Usando o operador **OU**, basta que apenas uma das duas primeiras proposições seja verdadeira para que a terceira também seja.

Basta que apenas um dos dois primeiros números seja “um” (1), para que o terceiro também seja.

1	1	0	0
1	0	1	0
1	1	1	0

→ Estiver com fome OU
→ For meio dia
→ Irei almoçar

Podemos efetuar operações binárias com variáveis. Como exemplo, vamos supor que $A = 0101$ e $B = 1111$.

NÃO é um operador que inverte a lógica. Se for verdade, torna-se falsa, e vice-versa.

0	1	0	1
1	1	1	1
0	1	0	1
1	0	1	0

→ A
→ B
→ A E B
→ NÃO (A E B)

Introdução à Lógica de Programação

- **Exemplos:**

Dadas as variáveis:

- **Numéricas:** A = 10, B = 5, C = 2
- **Lógicas:** X = FALSO, Y = VERDADEIRO

Vejamos os resultados das expressões:

- a) A < B E C >= 2
FALSO E VERDADEIRO
Resposta: FALSO
- b) A - 5 < B E C <= A / 2
FALSO E VERDADEIRO
Resposta: FALSO
- c) A < B OU C < B
FALSO OU VERDADEIRO
Resposta: VERDADEIRO
- d) B < A OU C > B E 2 * C < B
VERDADEIRO OU FALSO E VERDADEIRO
VERDADEIRO OU FALSO
Resposta: VERDADEIRO
- e) B < A E C > B E 2 * C < B
VERDADEIRO E FALSO E VERDADEIRO
FALSO E VERDADEIRO
Resposta: FALSO

f) $C < A \wedge A < C * 6$

VERDADEIRO E VERDADEIRO

Resposta: VERDADEIRO

g) $\neg X \wedge Y$

\neg FALSO E VERDADEIRO

VERDADEIRO E VERDADEIRO

Resposta: VERDADEIRO

h) $\neg Y \vee X \wedge C < A / B$

\neg VERDADEIRO OU FALSO E FALSO

FALSO OU FALSO E FALSO

FALSO OU FALSO

Resposta: FALSO

i) $Y \vee X \wedge C < A / B$

VERDADEIRO OU FALSO E FALSO

VERDADEIRO OU FALSO

Resposta: VERDADEIRO

j) $(X \vee Y) \wedge C < A / B$

(FALSO OU VERDADEIRO) E FALSO

VERDADEIRO E FALSO

Resposta: FALSO

4.11. Função

Função é uma rotina que retorna um valor específico.

- **STR()**: Transforma número em caracteres numéricos;
- **VAL()**: Transforma caracteres numéricos em número;
- **LEN()**: Retorna um inteiro contendo o número de caracteres de uma sequência de caracteres, ou seja, conta os caracteres.

Exemplos:

A = 10
B = STR(A) è B = “10”
C = STR(A * 5) è C = “50”
D = STR(A - 6) è D = “4”
E = STR(2000) è E = “2000”

TOT = “123”
N1 = VAL (TOT) è N1 = 123

NOME = “JOAQUIM JOSE DA SILVA XAVIER”
N2 = LEN (NOME) è N2 = 28

4.12. Concatenação de alfanuméricos

Os símbolos para concatenação de alfanuméricos são: & ou + .

Exemplo:

DIA = 19
MES = "abril"
ANO = 2013
CIDADE = "São Paulo"

A seguir, temos um exemplo de como fica a criação da variável DATA utilizando as variáveis anteriores para resultar no texto **São Paulo, 19 de abril de 2013**:

DATA = CIDADE & ", " & STR(DIA) & " de " & MES & " de " & STR(ANO)

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- **Variáveis** são áreas na memória, utilizadas em programação, que servem para armazenar dados. O conteúdo de uma variável pode ser alterado, mas uma variável só pode conter um dado por vez;
- Com as variáveis podemos verificar a veracidade, ou não, de uma condição para assim obtermos um ou outro resultado;
- As variáveis de memória podem ser usadas para o controle de repetições de comandos;
- Quando trabalhamos com arquivos de bancos de dados, podemos fazer comparações, trocas e procuras de registros através das variáveis de memória;
- Para declarar uma variável, devemos informar o nome e o tipo da variável;
- O nome de uma variável não pode começar por um número e só é permitido o uso de underline (_) no nome, e nenhum outro sinal. Evite utilizar nomes de comandos, funções ou campos de um banco de dados nas variáveis;

- O **comando de atribuição** serve para guardar uma informação na área de memória do computador. Utilizaremos o sinal de igual (=) para representar o comando de atribuição;
- Diferente das variáveis, as **constantes** possuem valor fixo e não sofrem alteração durante o processamento;
- Em programação, utilizamos os mesmos **operadores aritméticos** da matemática, com as mesmas prioridades. Em primeiro lugar, são executadas as exponenciações e as radiciações, em segundo as multiplicações e as divisões, e depois as adições e as subtrações;
- Os **operadores lógicos** são usados quando existem duas ou mais condições em um teste lógico;
- **Função** é uma rotina que retorna um valor específico;
- Os símbolos utilizados para concatenação de alfanuméricos são & ou +.

4

Variáveis, Operadores e Funções

Teste seus conhecimentos



IMPACTA
EDITORIA

Introdução à Lógica de Programação

1. Dados os seguintes valores para as variáveis:

$$A = 5 \quad B = 7 \quad C = 11 \quad D = 8 \quad E = 6$$

O valor de V1 na equação $V1 = C + 4 + A * D / 2$ é:

- 30
- 35
- 40
- 45
- Nenhuma das alternativas anteriores está correta.

2. Dados os seguintes valores para as variáveis:

$$A = 3 \quad B = 5 \quad C = 9 \quad D = 1 \quad E = 4$$

O valor de V2 na equação $V2 = 30 / 15 + E ^\star D + 3 * C - D$ é:

- 63
- 29
- 44
- 32
- Nenhuma das alternativas anteriores está correta.

3. Dados os seguintes valores para as variáveis:

A = 2 B = 3 C = 4 D = 5 E = 6

O valor de V3 na equação $V3 = 3^{**} A - E - 3 * (D - (B + B)/B^5) - 2$ é:

- 1
- 2
- 3
- 4
- Nenhuma das alternativas anteriores está correta.

4. Dadas as variáveis:

Numéricas: A = 10, B = 5, C = 2

Lógicas: X = FALSO, Y = VERDADEIRO

O resultado das equações:

$(B < A \text{ OU } C > B) \text{ E } 2 * C < B \text{ OU } \text{NÃO } X$

e

$C > A \text{ E } A < C * 5$

é:

- a) VERDADEIRO e VERDADEIRO
- b) FALSO e VERDADEIRO
- c) VERDADEIRO e FALSO
- d) FALSO e FALSO
- e) Nenhuma das alternativas anteriores está correta.

Introdução à Lógica de Programação

5. Dadas as variáveis VAR1 = “LOGICA”, VAR2 = “20” e VAR3 = 7, o resultado da equação $(5 + \text{LEN}(\text{VAR1}) - \text{VAR3})^{} 2 - \text{VAL}(\text{STR}(\text{VAL}(\text{VAR2}) + 1) + \text{STR}(\text{VAR3}))$ é:**

- a) 201
- b) 209
- c) - 202
- d) - 201
- e) Nenhuma das alternativas anteriores está correta.

4

Variáveis, Operadores e Funções

Mãos à obra!

Amanhã é dia de programar! 384.04.380-25



IMPACTA
EDITORA

Laboratório 1

Exercício 1

Dados os seguintes valores para as variáveis:

$$A = 4 \quad B = 6 \quad C = 10 \quad D = 2 \quad E = 5$$

Resolva as seguintes equações:

$$X1 = C + A * D$$

$$X2 = (C - E) * B$$

$$X3 = A - B * (C - (A + B) / E * 2)$$

$$X4 = E ** D + 3 * C$$

$$X5 = (B + C) // 2 - D$$

$$X6 = (E + A * (E ^ D / (C - E)) - A) / (D + E)$$

Exercício 2

Escreva as fórmulas a seguir como linhas de programação, ou seja, deixe-as linearizadas:

$$X = 2 \times A + 5$$

$$Y = A + 3 \times B^2$$

$$Z = \frac{A + B}{5 \times C}$$

$$W = \sqrt[3]{A + 2 \times B}$$

$$R = 2^4 - \sqrt[4]{\frac{A}{C}}$$

$$S = 5^2 \times \left((2 \times 5) + \left(\frac{15}{3} \right)^3 \right)$$

Introdução à Lógica de Programação

Exercício 3

Resolva as seguintes fórmulas:

$$D1 = 257 / 10$$

$$D2 = 257 / 100$$

$$D3 = 257 / 1000$$

$$Q1 = 257 \text{ DIV } 10$$

$$Q2 = 257 \text{ DIV } 100$$

$$Q3 = 257 \text{ DIV } 1000$$

$$R1 = 257 \text{ MOD } 10$$

$$R2 = 257 \text{ MOD } 100$$

$$R3 = 257 \text{ MOD } 1000$$

$$M1 = 257 \text{ DIV } 10 \text{ MOD } 10$$

$$M2 = 257 \text{ MOD } 100 \text{ DIV } 10$$

Exercício 4

Crie um algoritmo que realize as seguintes tarefas enquanto números são digitados em uma variável de nome NRO:

- Conte os números digitados;
- Acumule os valores dos números digitados;
- Ao final, exiba quantos números foram digitados e a soma dos valores digitados.

Exercício 5

Crie um algoritmo que realize as seguintes tarefas enquanto números são digitados em uma variável de nome **NRO**:

- Conte quantos números 5 foram digitados;
- Acumule o valor de todos os números diferentes de 5;
- Ao final, exiba quantos números 5 foram digitados e a soma dos valores diferentes de 5.

Exercício 6

Crie um algoritmo que realize as seguintes tarefas enquanto idades são digitadas em uma variável de nome **IDADE**:

- Conte quantas pessoas informaram a idade maior que 40;
- Conte quantas pessoas informaram a idade menor ou igual a 40;
- Ao final, exiba quantas pessoas informaram a idade maior que 40 e quantas pessoas informaram a idade menor ou igual a 40.

Exercício 7

Crie um algoritmo que realize as seguintes tarefas enquanto números são digitados em uma variável de nome **NRO**:

- Conte quantos números que foram digitados terminam em zero;
- Conte quantos números digitados NÃO terminam em zero;
- Ao final, exiba quantos números digitados terminam em zero e quantos números digitados NÃO terminam em zero.

Introdução à Lógica de Programação

Exercício 8

Observe os registros cadastrados, contendo informações sobre o sexo e cor dos olhos de pessoas:

QTDE	SEXO	COR DOS OLHOS
8	FEMININO	AZUL
6	FEMININO	VERDE
5	FEMININO	PRETO

QTDE	SEXO	COR DOS OLHOS
3	MASCULINO	AZUL
5	MASCULINO	VERDE
7	MASCULINO	PRETO

Quantas pessoas serão selecionadas se:

- a) Olhos = "PRETO" E Sexo = "MASCULINO"
- b) Olhos = "PRETO" OU Sexo = "MASCULINO"
- c) Olhos = "AZUL" OU Olhos = "VERDE" E Sexo = "FEMININO"
- d) (Olhos = "AZUL" OU Olhos = "VERDE") E Sexo = "FEMININO"
- e) Olhos <> "AZUL" E Sexo = "FEMININO" OU Olhos <> "VERDE"
- f) Olhos <> "AZUL" E (Sexo = "FEMININO" OU Olhos <> "VERDE")
- g) Olhos <> "AZUL" E Sexo = "FEMININO" E Olhos <> "VERDE"
- h) Olhos = "AZUL" E Sexo = "MASCULINO" E Olhos = "VERDE"

Exercício 9

Dado o seguinte trecho de um algoritmo:

```
Se A = B OU A <= C - 2 E B > C então
    Comando1
Senão
    Comando2
Fim Se
```

Para os valores das questões a seguir, será executado o **Comando1** ou o **Comando2**?

- | | |
|----------------|-------|
| a) A = 5 B = 5 | C = 5 |
| b) A = 5 B = 3 | C = 4 |
| c) A = 5 B = 8 | C = 7 |
| d) A = 5 B = 6 | C = 1 |

Exercício 10

Dados A = "IMPACTA", B = "10" e N = 15, resolva as questões a seguir:

- a) STR(N) + B
- b) VAL(B) + LEN(A)
- c) N + VAL(B)
- d) LEN(B) + N
- e) STR (LEN (A)) + STR (N)
- f) STR (VAL (B) +7)
- g) VAL (STR (VAL(B) - 3) + STR (N))

Introdução à Lógica de Programação

Exercício 11

Dadas as variáveis **NOME** = “João” e **IDADE** = 17, defina as variáveis **MSG1** E **MSG2** concatenando **NOME** e **IDADE** de forma que o conteúdo resulte nas seguintes mensagens:

- a) **MSG1** = “João tem 17 anos”
- b) **MSG2** = “João vai fazer 18 anos”

Fluxograma

5

- ✓ Simbologia;
- ✓ Criação de fluxogramas;
- ✓ Estruturas básicas de fluxogramas;
- ✓ Teste de mesa.



IMPACTA
EDITORA

5.1. Introdução

Fluxograma, ou Diagrama de Blocos, é a representação gráfica de um algoritmo, sendo constituído de blocos funcionais que mostram o fluxo dos dados e as operações efetuadas com eles.

5.2. Símbologia

O fluxograma utiliza símbolos que permitem a descrição da sequência dos passos a serem executados de forma clara e objetiva. A tabela a seguir descreve a simbologia utilizada em fluxogramas:

Símbolo	Significado	Descrição
	Terminação	Utilizado para indicar o início, fim ou saída de um fluxograma.
	Processamento	Utilizado para indicar uma ação.
	Decisão	Utilizado para comparar dados e desviar o fluxo conforme o resultado seja verdadeiro ou falso.
	Entrada manual	Utilizado para a entrada de dados através do teclado.

Símbolo	Significado	Descrição
A paralelogramo.	Entrada / Saída	Utilizado para indicar operações de leitura e gravação de registros.
Um retângulo dividido verticalmente em duas partes iguais.	Processamento Predefinido ou Módulo	Utilizado para indicar uma chamada a uma sub-rotina.
Um retângulo com bordas desiguais.	Documento	Utilizado para indicar a impressão de dados.
Um octágono.	Exibir	Utilizado para exibir dados na tela.
Um hexágono.	Preparação	Utilizado para a execução de looping.
Um círculo.	Conector	Utilizado para continuar o fluxograma em outra parte na mesma página.
Um pentágono.	Conector de Página	Utilizado para continuar o fluxograma em outra página.
Uma seta apontando para a direita.	Seta de Fluxo de Dados	Utilizado para indicar o sentido do fluxo de dados conectando os símbolos existentes.

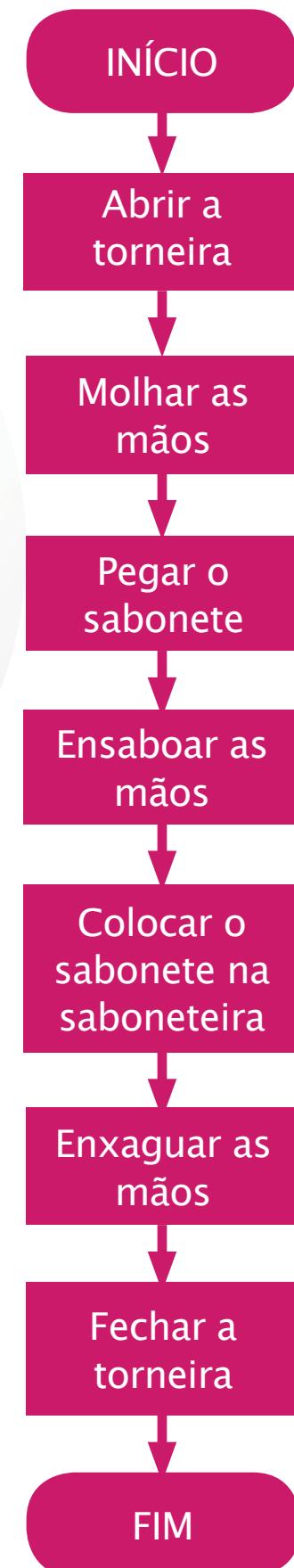
5.3. Criando fluxogramas

A seguir, temos um algoritmo e o fluxograma de **como lavar as mãos com sabonete de barra**. Detalhes: Suponha que você já está em frente à pia e não falta nada para que possa lavar as mãos:

INÍCIO
Abrir a torneira Molhar as mãos
Pegar o sabonete
Ensaboar as mãos
Colocar o sabonete na saboneteira
Enxaguar as mãos
Fechar a torneira
FIM

Observação: O algoritmo anterior recebeu na primeira linha o texto INÍCIO, porém é comum usar o nome do programa. Por exemplo: Neste caso poderia ser **LAVAR AS MÃOS**.

O fluxograma teria, então, no primeiro símbolo, que é o símbolo de inicialização, o nome **LAVAR AS MÃOS**. Veja, também, que o algoritmo não recebeu numeração, ou seja, ela não é obrigatória, servindo apenas para facilitar o entendimento.



5.3.1. Estruturas básicas

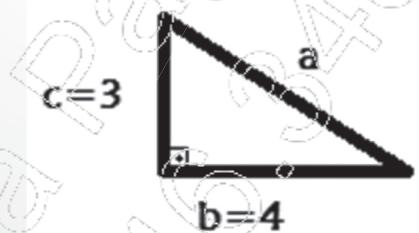
Vejamos, a seguir, as estruturas básicas de um fluxograma.

- **Sequência**

No exemplo anterior, de **LAVAR AS MÃOS**, houve uma ação após a outra, portanto chamamos essa estrutura de **SEQUÊNCIA**.

Vejamos outro exemplo. De acordo com o Teorema de Pitágoras, em qualquer triângulo retângulo, o quadrado do comprimento da hipotenusa é igual à soma dos quadrados dos comprimentos dos catetos. O triângulo retângulo pode ser identificado pela existência de um ângulo reto, ou seja, um ângulo medindo 90° . O triângulo retângulo é formado pela hipotenusa, que constitui o maior segmento do triângulo e é localizada opostamente ao ângulo reto, e por dois catetos.

No triângulo a seguir, a hipotenusa está representada pela variável **a**, um cateto pela variável **b** que tem o valor 4 e o outro cateto pela variável **c**, que tem o valor 3.



Veja o algoritmo para efetuar o cálculo $a^2 = b^2 + c^2$ e exibir o valor da hipotenusa:

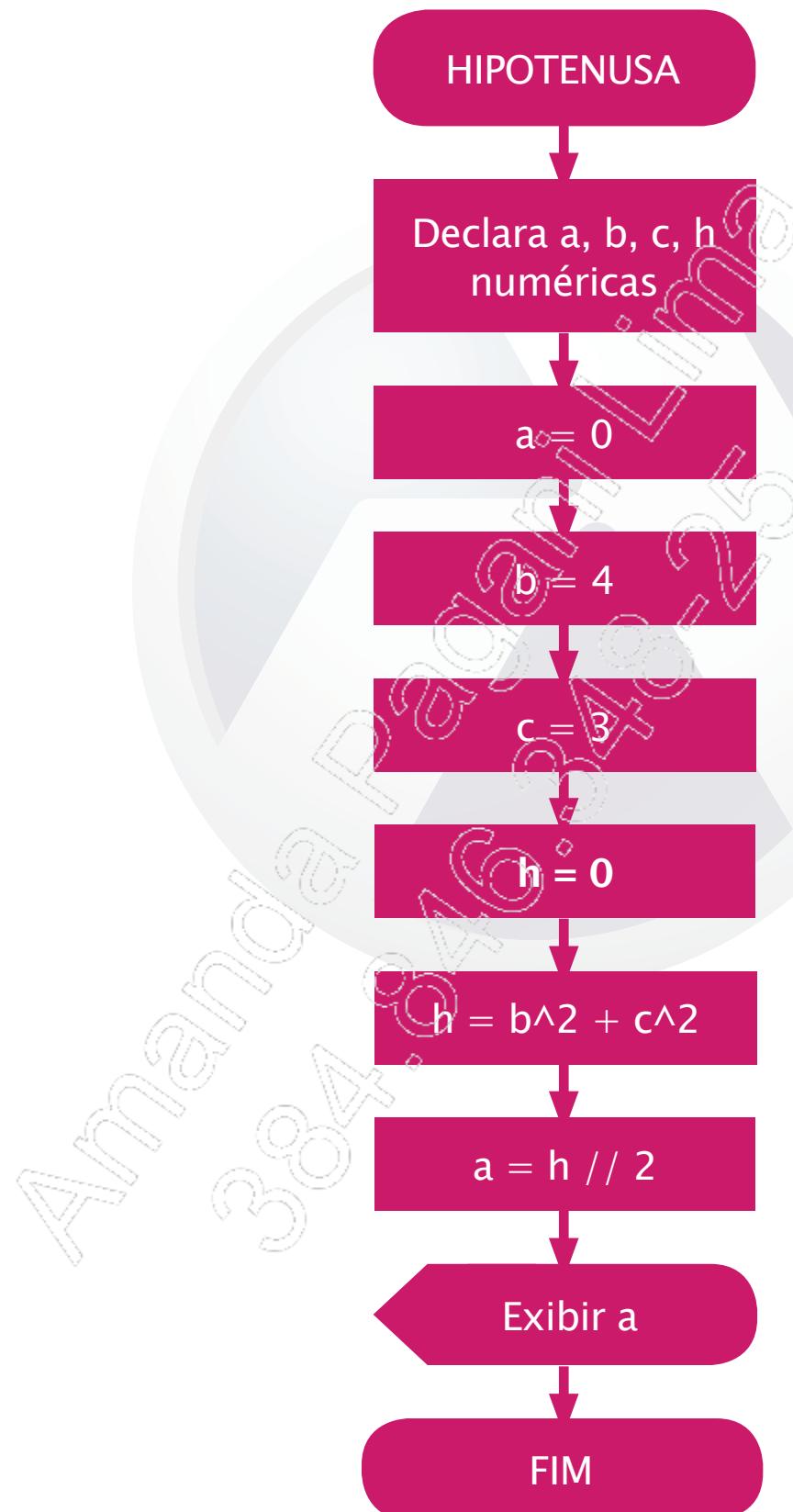
```

HIPOTENUSA
Declara a, b, c, h numéricas
a = 0
b = 4
c = 3
h = 0
h = b2 + c2
a = h // 2 → Raiz quadrada de h para
descobrir o valor da hipotenusa
Exibir a
FIM

```

Introdução à Lógica de Programação

Note, no algoritmo, que foi executada uma sequência de ações. Veja, a seguir, como fica seu fluxograma:



A seguir, temos a simulação da execução do fluxo para descobrir o valor da hipotenusa:

```

a = 0
b = 4
c = 3
h = 0
h = b^2 + c^2
h = 4^2 + 3^2
h = 16 + 9
h = 25
a = 25 // 2
a = 5
Exibir a → Exibe o valor 5
```

- **Condição/Seleção**

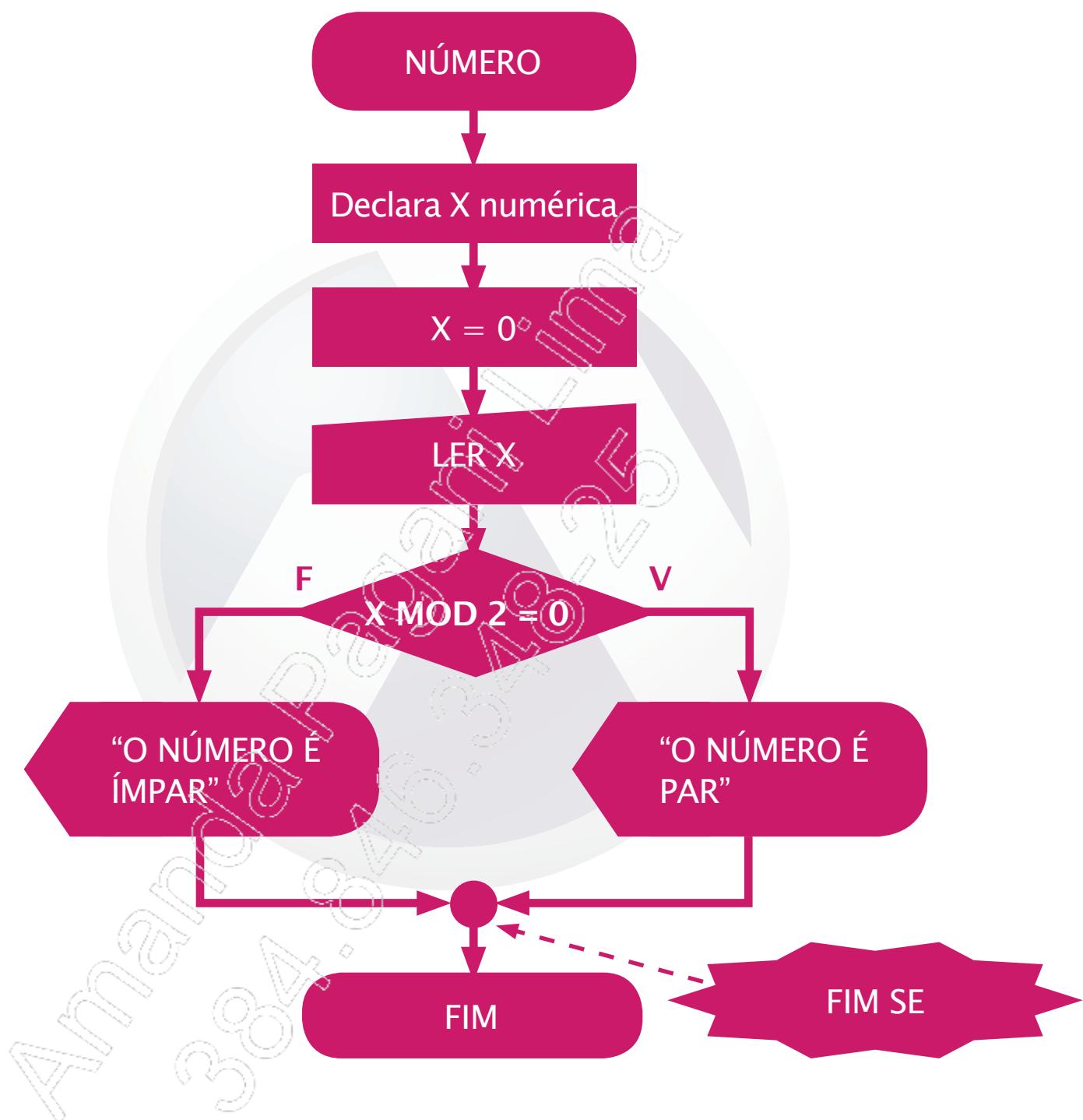
Esta estrutura permite representar uma condição e selecionar o fluxo a seguir dependendo do resultado da condição, se a condição é verdadeira ou falsa, podendo assim executar diferentes instruções. Para isso usaremos o comando **SE** e suas duas respostas, **ENTÃO** e **SENÃO** (IF, THEN, ELSE). O **ENTÃO** é a saída verdadeira e o **SENÃO** é a saída falsa.

A seguir, temos um algoritmo e o fluxograma para exibir se um número digitado é par ou ímpar:

```

NÚMERO
Declara X numérica
X = 0
Ler X
Se X MOD 2 = 0 → Verifica se o resto da divisão
de X por 2 é igual a zero
    Então exibir mensagem “O número é par”
    Senão exibir mensagem “O número é ímpar”
Fim Se
FIM
```

Introdução à Lógica de Programação



Note que depois do **FIM SE** só existe uma seta fluxo, ou seja, só existe um caminho a seguir. O comando **FIM SE** é a união das setas dos fluxos que vêm do lado verdadeiro e do lado falso da decisão.

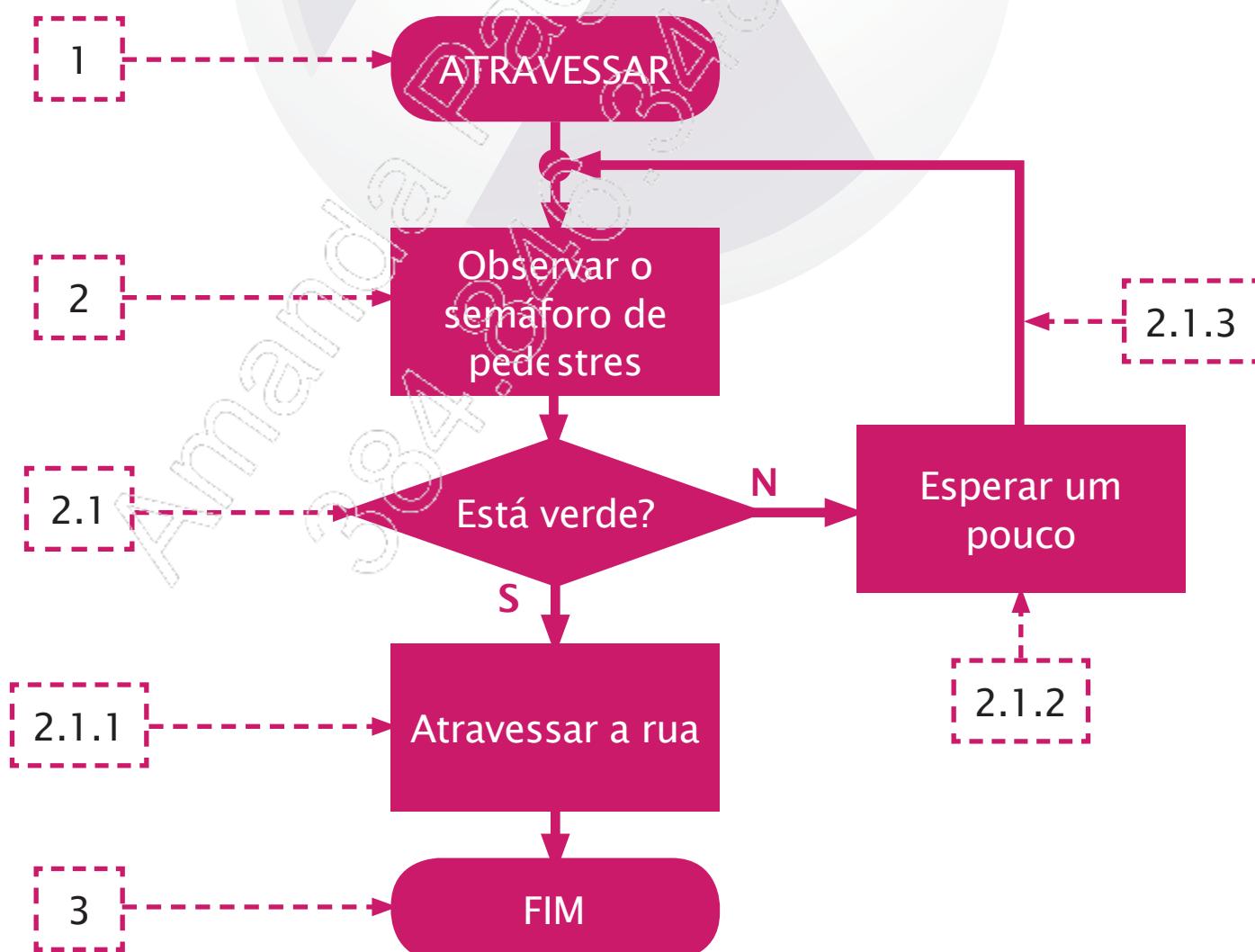
- Repetição condicional

Esta estrutura permite representar uma condição e, dependendo do resultado da mesma, pode-se executar novamente algumas instruções.

Vejamos um exemplo de um algoritmo e de um fluxograma de como atravessar uma rua num semáforo de pedestres.

- 1. ATRAVESSAR**
- 2. Observar o semáforo de pedestres**
 - 2.1. Verifique se está verde**
 - 2.1.1. Se sim: Atravessar a rua**
 - 2.1.2. Senão: Esperar um pouco**
 - 2.1.3. Vá para o passo 2**
- 3. FIM**

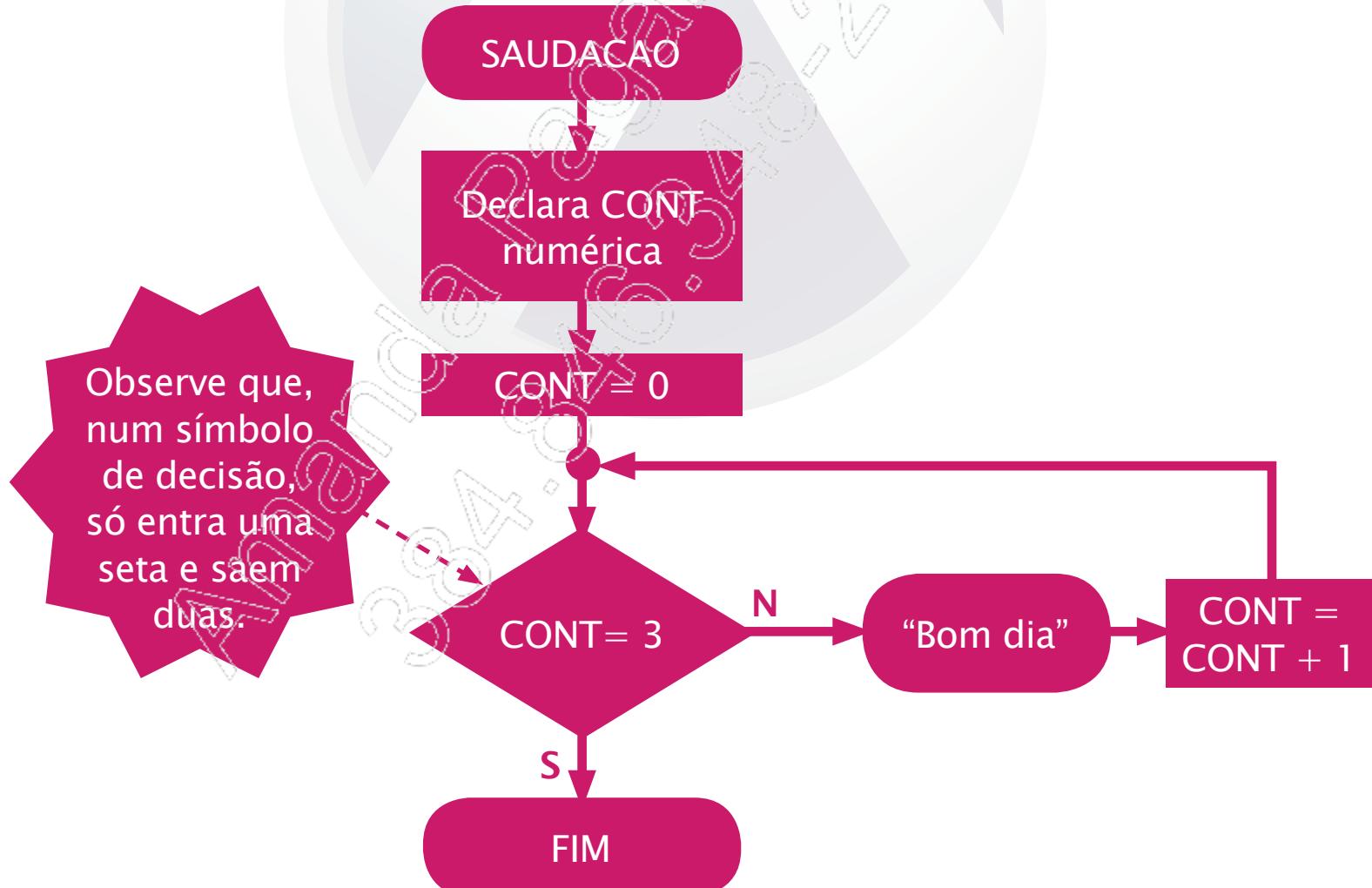
Note que o passo 2.1.3, **Vá para o passo 2**, é representado apenas pela seta de fluxo. A seta terminou no caminho que leva ao passo 2, porém poderia ter entrado diretamente no símbolo do passo 2.



Introdução à Lógica de Programação

A seguir, temos um exemplo de um algoritmo e de um fluxograma com um contador, cujo objetivo é exibir 3 (três) vezes a mensagem “Bom dia”:

1. SAUDACAO
2. Declara CONT numérica
3. CONT = 0
4. Se CONT = 3
 4.1. Então Vá para o passo 5
 4.2. Senão Exibir “Bom dia”
 4.3. CONT = CONT + 1
 4.4. Vá para o passo 4
5. FIM



5.4. Teste de Mesa

Teste de Mesa é a simulação da execução de um algoritmo, programa ou fluxograma, sem utilizar o computador, empregando apenas lápis e papel.

A simulação da execução do fluxo para descobrir qual o valor da hipotenusa, conforme vimos anteriormente, representa o **Teste de Mesa** daquele fluxo.

Note no fluxo anterior que, inicialmente, o contador **CONT** recebeu o valor zero (0) e depois, na execução, recebeu os valores 1, 2 e 3.

```
CONT = 0  
CONT = CONT + 1 → CONT = 1  
CONT = CONT + 1 → CONT = 2  
CONT = CONT + 1 → CONT = 3
```

O resultado mostrado representa o **Teste de Mesa** exibindo todos os valores que a variável **CONT** recebeu.

Pontos Principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- **Fluxograma**, ou **Diagrama de Blocos**, é a representação gráfica de um algoritmo, sendo constituído de blocos funcionais que mostram o fluxo dos dados e as operações efetuadas com eles;
- O fluxograma utiliza símbolos que permitem a descrição, de forma clara e objetiva, da sequência dos passos a serem executados;
- Numa programação, para efetuarmos uma decisão, podemos utilizar o comando **SE**, **ENTÃO**, **SENÃO** (IF, THEN, ELSE). O **ENTÃO** é a saída verdadeira e o **SENÃO** é a saída falsa do comando **SE**;
- O comando **FIM SE** é a união das setas dos fluxos que vêm do lado verdadeiro (**ENTÃO**) e do lado falso (**SENÃO**) da decisão. A partir deste ponto só existe uma seta fluxo, ou seja, só existe um caminho a seguir;
- **Teste de mesa** é a simulação da execução de um algoritmo, programa ou fluxograma.

5

Fluxograma

Teste seus conhecimentos

Amanda Paolini Lima
384.846-25



IMPACTA
EDITORA

Introdução à Lógica de Programação

1. Dado o seguinte algoritmo e seu respectivo fluxograma:

INÍCIO

 DECLARA A, B, C, D NUMÉRICAS

 A = 1

 B = (A + 4) ** 2

 SE B <= 25

 ENTÃO A = 3

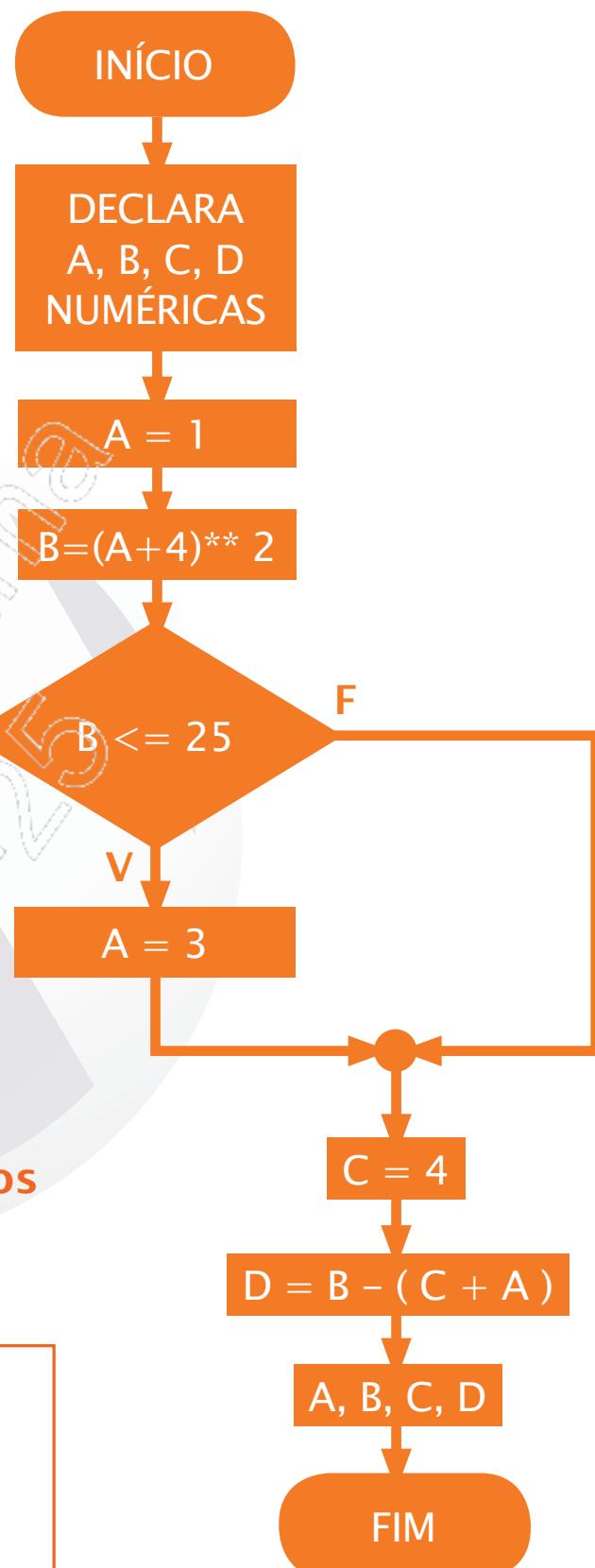
 FIM SE

 C = 4

 D = B - (C + A)

 EXIBIR A, B, C, D

FIM



Após o teste de mesa, os valores exibidos das variáveis A, B, C, D são:

- a) 1, 25, 4, 20
- b) 2, 20, 8, 10
- c) 3, 25, 4, 18
- d) 3, 17, 4, 14
- e) Nenhuma das alternativas anteriores está correta.

2. Dado o seguinte algoritmo e seu respectivo fluxograma:

INÍCIO

 DECLARA A, B, C, D NUMÉRICAS

 A = 1

 B = (A + 1) ** 3 * 3 + A

 SE B <= 25

 ENTÃO C = 3

 SENÃO C = 4

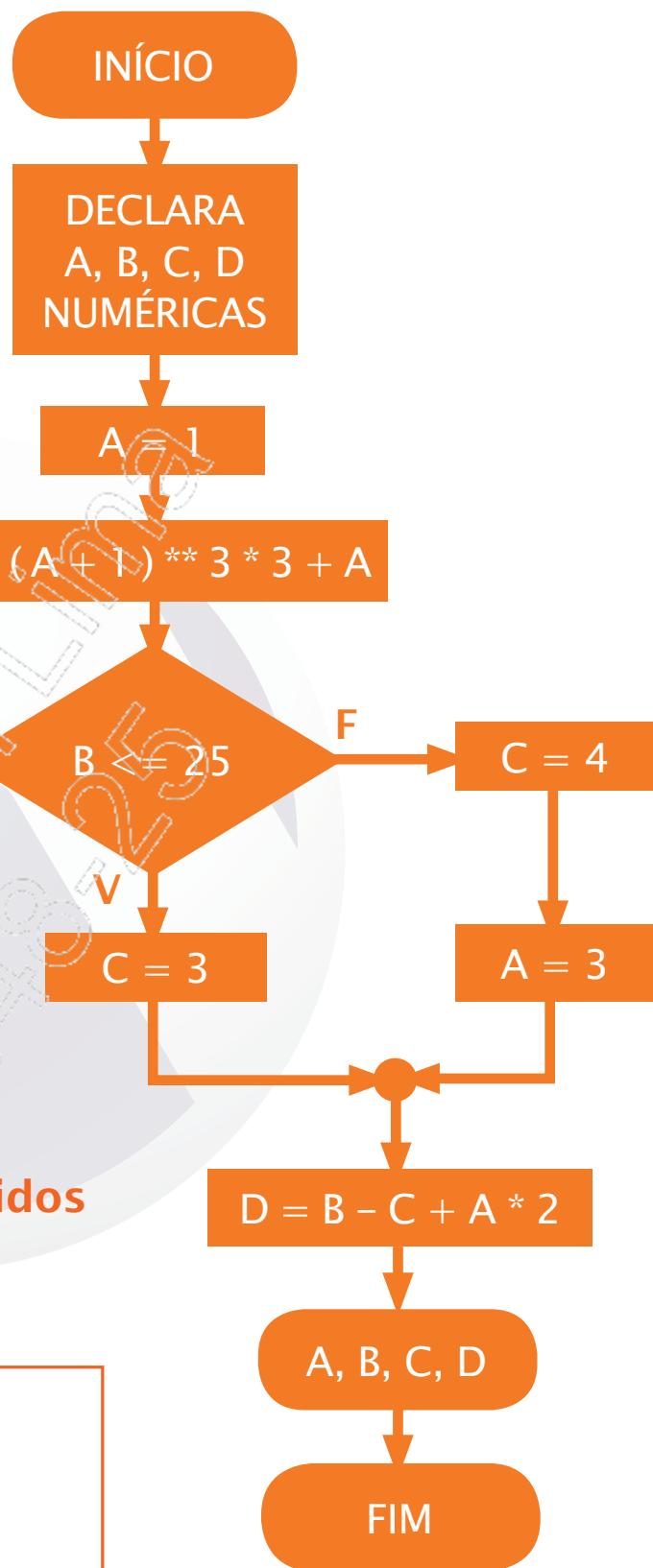
 A = 3

 FIM SE

 D = B - C + A * 2

 EXIBIR A, B, C, D

FIM



Após o teste de mesa, os valores exibidos das variáveis A, B, C, D são:

- a) 1, 32, 3, 22
- b) 1, 25, 3, 24
- c) 3, 25, 4, 22
- d) 3, 32, 4, 34
- e) Nenhuma das alternativas anteriores está correta.

Introdução à Lógica de Programação

3. Dado o seguinte algoritmo e seu respectivo fluxograma:

INÍCIO

DECLARA H, J, K, L, M NUMÉRICAS

H = 3

J = 9 // 2 + H

SE J < 5

ENTÃO K = 3

L = 4

SENÃO

ENTÃO

SE J < 10

K = 6

L = 7

H = 8

K = 9

L = 10

SENÃO

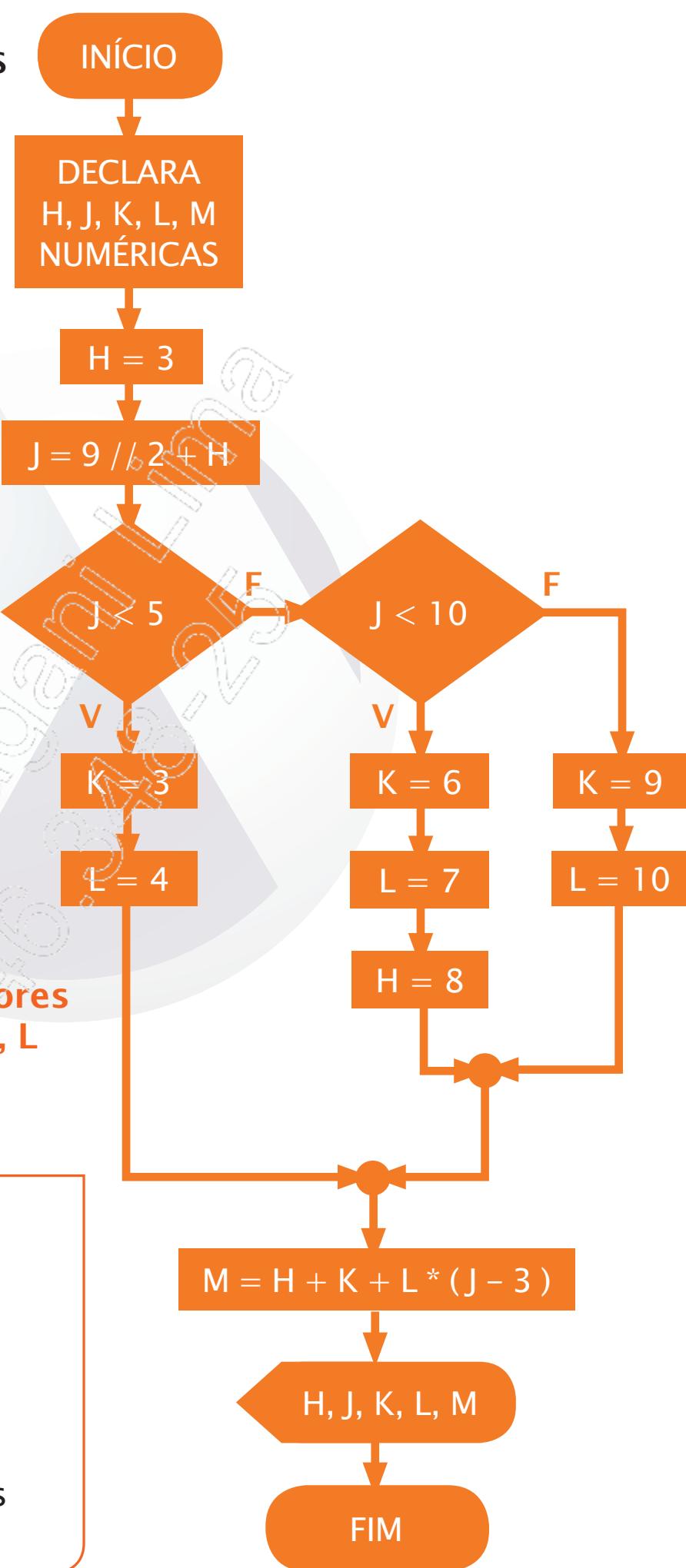
FIM SE

FIM SE

M = H + K + L * (J - 3)

EXIBIR H, J, K, L, M

FIM



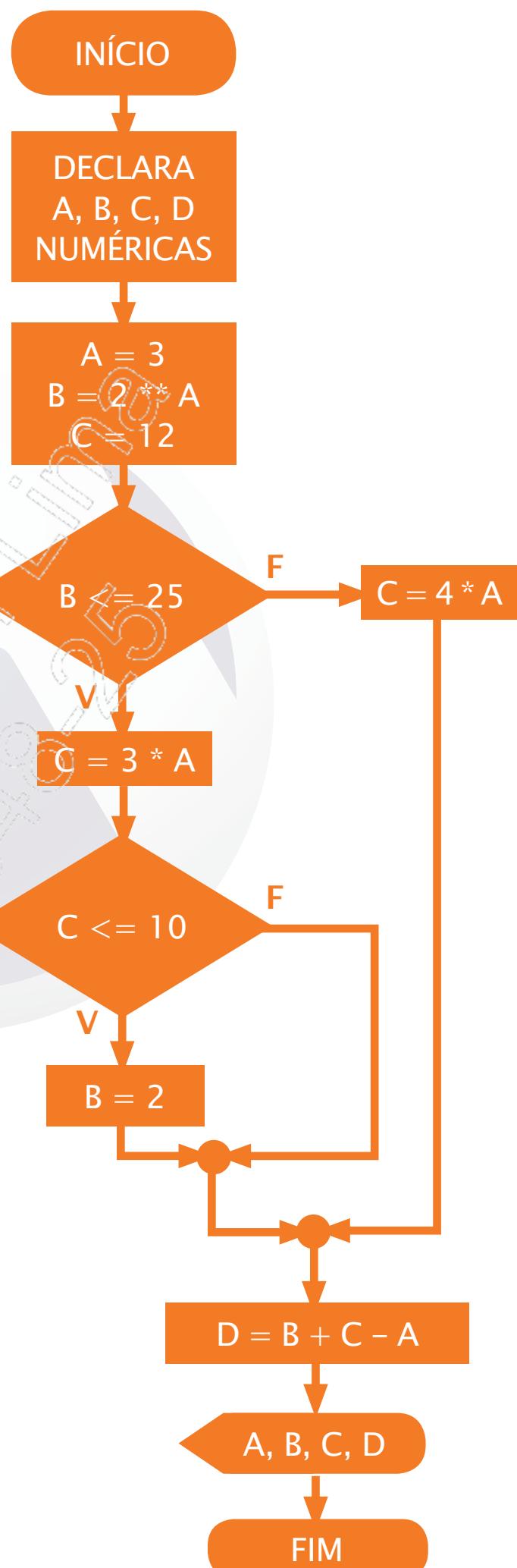
Após o teste de mesa, os valores exibidos das variáveis H, J, K, L e M são:

- a) 8, 6, 9, 10, 47
- b) 3, 3, 3, 4, 6
- c) 3, 21, 9, 10, 192
- d) 8, 6, 6, 7, 35
- e) Nenhuma das alternativas anteriores está correta.

4. Dado o seguinte algoritmo e seu respectivo fluxograma:

```

INÍCIO
    DECLARA A, B, C, D NUMÉRICAS
    A = 3
    B = 2 ** A
    C = 12
    SE B <= 25
        ENTÃO  C = 3 * A
        SE    C <= 10
            B = 2
        FIM SE
        SENÃO  C = 4 * A
    FIM SE
    D = B + C - A
    EXIBIR A, B, C, D
FIM
  
```



Após o teste de mesa, os valores exibidos das variáveis A, B, C, D são:

- a) 3, 25, 12, 34
- b) 2, 3, 9, 8
- c) 3, 2, 9, 8
- d) 3, 8, 12, 8
- e) Nenhuma das alternativas anteriores está correta.

Introdução à Lógica de Programação

5. Dado o seguinte algoritmo e seu respectivo fluxograma:

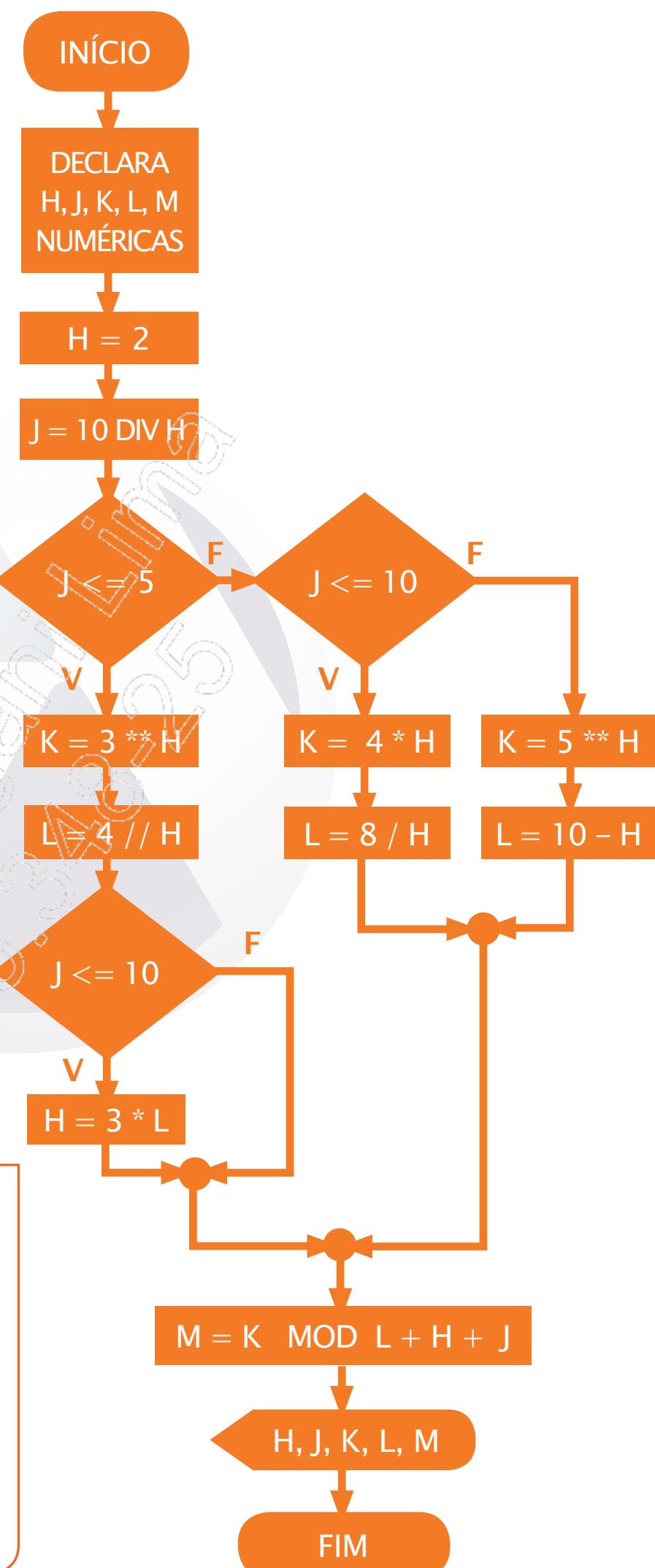
INÍCIO

```
DECLARA H, J, K, L, M NUMÉRICAS  
H = 2  
J = 10 DIV H  
SE J <= 5  
ENTÃO K = 3 ** H  
L = 4 // H  
SE J <= 10  
ENTÃO H = 3 * L  
FIM SE  
SENÃO SE J <= 10  
ENTÃO K = 4 * H  
L = 8 / H  
SENÃO K = 5 * H  
L = 10 - H  
FIM SE  
FIM SE  
M = K MOD L + H + J  
EXIBIR H, J, K, L, M
```

FIM

Após o teste de mesa,
os valores exibidos das
variáveis H, J, K, L e M são:

- a) 6, 5, 9, 2, 12
- b) 2, 10, 8, 4, 12
- c) 2, 12, 10, 8, 16
- d) 2, 5, 9, 2, 8
- e) Nenhuma das alternativas anteriores está correta.



5

Fluxograma

Mãos à obra!

Amanda Paixão Lima
384.846.349-25



IMPACTA
EDITORA

Laboratório 1

Exercício 1

Faça o teste de mesa dos algoritmos a seguir:

- **Algoritmo 1**

INÍCIO

 DECLARA A, B, C NÚMERICAS

 A = 10

 B = 5

 C = 1

 A = A + 1

 B = B + A

 C = C + 2 * B

 EXIBIR A, B, C

FIM

- **Algoritmo 2**

INÍCIO

 DECLARA A, B NUMÉRICAS

 A = 8

 B = 5

 B = B + A

 A = B - A

 B = B - A

 EXIBIR A, B

FIM

Exercício 2

Faça o fluxograma para os algoritmos a seguir:

- **Algoritmo 1**

1. INÍCIO
2. Sente-se
3. Aguarde
4. Escute o nome chamado
5. É o seu nome?
 - 5.1. Se sim: Entrar no consultório
 - 5.2. Senão: Vá para o passo 3
6. FIM

- **Algoritmo 2**

```
INÍCIO
DECLARA N, P, Q NUMÉRICAS
N = 1
SE N < 10
    ENTÃO N = N + 1
        P = N * 2
        SE N < 10
            ENTÃO N = N + 1
                P = N * 2
                Q = P + N
                SE N < 10
                    ENTÃO N = N + 1
                        SE P < 10
                            ENTÃO SE N < 10
                                ENTÃO N = N + 1
                                FIM SE
                            FIM SE
                        FIM SE
                    FIM SE
                FIM SE
            FIM SE
        FIM SE
    FIM SE
EXIBIR N, P, Q
FIM
```

Introdução à Lógica de Programação

- **Algoritmo 3**

INÍCIO

 DECLARA H, J, K, L, M NUMÉRICAS

 H = 3

 L = 9

 J = L // 2 + H

 SE J < 5

 ENTÃO H = 8

 SENÃO H = 4

 FIM SE

 SE J < 6

 ENTÃO K = 3

 SE J < 10

 ENTÃO L = 11

 FIM SE

 SENÃO K = 6

 SE J < 10

 ENTÃO L = 10

 FIM SE

 H = 7

 FIM SE

 M = H + K + L

 EXIBIR H, J, K, L, M

FIM

Processamento predefinido

6

- ✓ Processamento predefinido;
- ✓ Construção de processamento predefinido.



IMPACTA
EDITORA

6.1. Introdução

Processamento Predefinido é um programa que pode ser usado em outro programa. No contexto de linguagens de programação, um subprograma, sub-rotina, função ou procedimento consiste numa parte do programa que resolve um problema específico.

O conceito de **função** difere de **procedimento** porque ela retorna um valor, sendo que, em algumas linguagens, esta distinção não existe.

Uma **sub-rotina** pode ser usada em várias partes do programa ou sistema e assim podemos ter aproveitamento de uma determinada ação em diversos momentos diferentes, com a vantagem do gerenciamento da lógica dessa ação estar centralizado. Esta sub-rotina também pode ser reaproveitada em outros programas ou sistemas.

Os parâmetros são os argumentos da sub-rotina. Eles são a comunicação da sub-rotina com os demais programas que a chamarão em algum momento e, através deles, a sub-rotina pode receber e retornar valores que serão processados para um objetivo final.

Sendo assim, no momento de se declarar a sub-rotina, é necessário declarar de que tipo, quais e quantos serão os parâmetros de entrada e de saída para que ela funcione corretamente.

Concluindo, quando notamos que, em um sistema que vamos desenvolver, será necessário usar várias vezes uma mesma rotina, criamos um programa só com ela e, quando necessário, o programa principal chama essa rotina, que será executada e depois retornará ao lugar de onde partiu com o resultado obtido e, então, o programa principal continua a partir da próxima linha de programação.

6.2. Construindo um processamento predefinido

Para construir um processamento predefinido, considera-se um trecho de programa. Vejamos um exemplo de um algoritmo para:

- Receber 2 números nas variáveis A e B;
- Trocar os conteúdos das 2 variáveis, ou seja, o conteúdo da variável A deve ser transferido para a variável B, e o da variável B, para a variável A;
- Mostrar na tela os valores de A e B, para conferir se os conteúdos foram realmente trocados.

PROGRAMA1

Declara A, B, C numéricas

A = 0

B = 0

C = 0

Ler A

Ler B

TROCAR(A,C)

TROCAR(B,A)

TROCAR(C,B)

Exibir A, B

FIM

← Programa principal

TROCAR(X,Y numéricas)

← Processamento predefinido

Y=X

SAÍDA

Introdução à Lógica de Programação

Vejamos os detalhes do trecho de programa anterior:

- **TROCAR(X,Y numéricas)**: Aqui está o nome da rotina e entre os parênteses está sendo definido 2 variáveis numéricas (**X,Y**) e, portanto, quando algum programa principal chamar essa rotina, deverá enviar 2 valores para que ela seja executada;
- **Y = X**: A variável **Y** está recebendo o mesmo valor da variável **X**;
- **SAÍDA** é Saída da rotina. Retorna a execução para a próxima linha do programa que a chamou.

No programa principal, as variáveis **A**, **B** e **C** foram definidas inicialmente com valor igual a zero (0), depois as variáveis **A** e **B** receberam os valores que foram fornecidos pelo usuário e depois a rotina **TROCAR()** foi chamada levando esses valores.

Para entendimento, vamos supor que os valores digitados pelo usuário foram 1 e 2, ou seja, **A=1** e **B=2**.

Na rotina **TROCAR()**, as variáveis **X** e **Y** recebem os valores das variáveis na ordem em que foram enviadas, portanto:

- **TROCAR(A,C)**: Chama a rotina e faz com que **X** receba o valor de **A**, e **Y** receba o valor de **C**:

$X=1$
 $Y=0$

Depois é executado o cálculo existente:

$Y=X$
 $Y=1$

Ao retornar ao programa principal, **A** continua com 1, **B** continua com 2, e **C**, que recebeu o valor de **Y**, fica com 1 após o cálculo.

- **TROCAR(B,A)**: Chama a rotina e faz com que **X** receba o valor de **B**, e **Y** receba o valor de **A**:

$X=2$
 $Y=1$

Depois é executado o cálculo existente:

$Y=X$

$Y=2$

Ao retornar ao programa principal, **B** continua com 2, **C** continua com 1, e **A**, que recebeu o valor de **Y**, fica com 2 após o cálculo.

- **TROCAR(C,B)**: Chama a rotina e faz com que **X** receba o valor de **C**, e **Y** receba o valor de **B**:

$X=1$

$Y=2$

Depois é executado o cálculo existente:

$Y=X$

$Y=1$

Ao retornar ao programa principal, **C** continua com 1, **A** continua com 2, e **B**, que recebeu o valor de **Y**, fica com 1 após o cálculo.

Após a execução do programa, o valor atual de **A** é o valor antigo de **B** e vice-versa. A variável **C** foi utilizada somente para auxiliar a operação.

Sendo assim, durante o decorrer do programa, as variáveis receberam os seguintes valores:

A: 0, 1 e 2

B: 0, 2 e 1

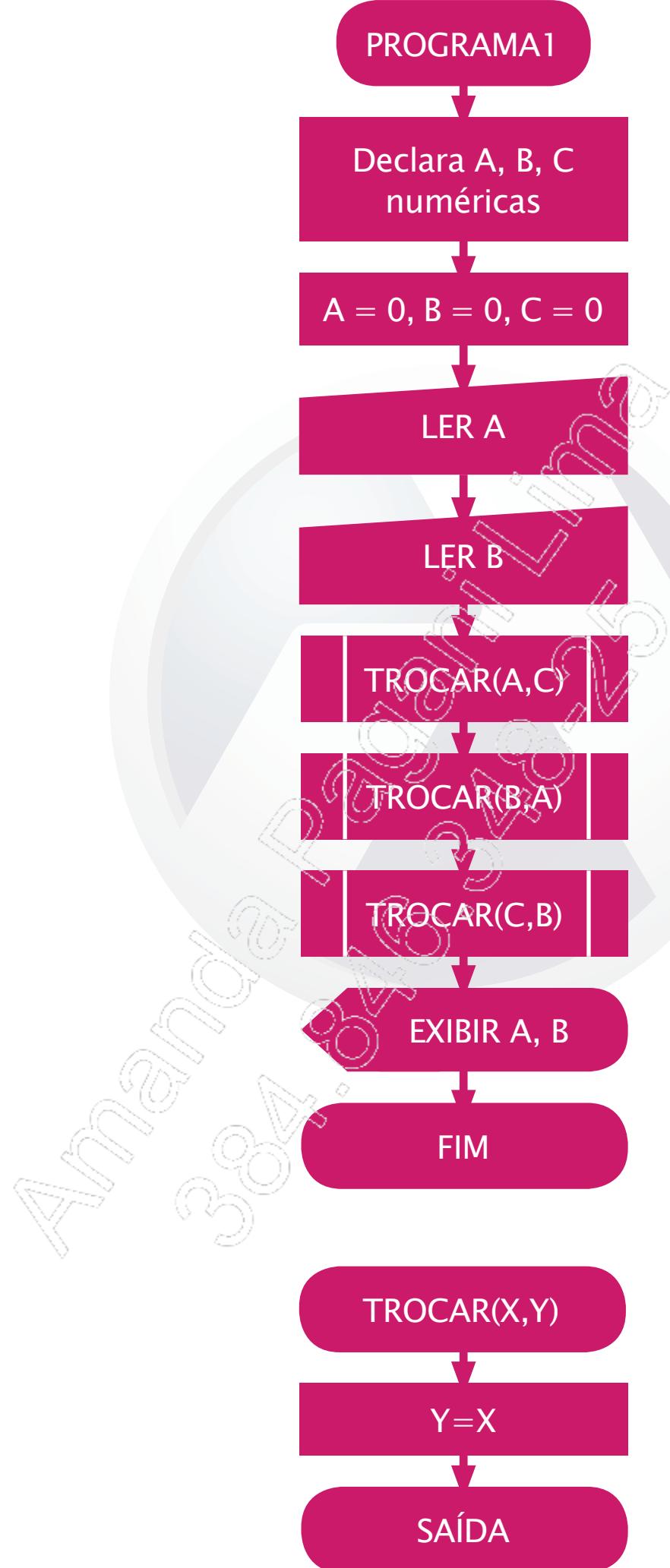
C: 0 e 1

- **Exibir A, B**: Exibe na tela os valores atuais de **A** e **B** (2 e 1), mostrando que os conteúdos de **A** e **B** foram realmente trocados.

O resultado mostrado representa o **teste de mesa**, ou seja, a simulação, passo a passo, da execução de um algoritmo.

Introdução à Lógica de Programação

A seguir, temos o fluxograma do programa principal e da rotina:



Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- **Processamento predefinido** é um programa que pode ser usado em outro programa;
- Uma **sub-rotina** pode ser usada em várias partes do programa ou sistema em questão, e também pode ser reaproveitada em outros programas ou sistemas;
- Os **parâmetros** são os argumentos da sub-rotina. Eles são a comunicação da sub-rotina com os demais programas que a chamarão em algum momento e, através deles, a sub-rotina pode receber e retornar valores que serão processados para um objetivo final;
- Para construir um processamento predefinido, consideramos um trecho de programa.

6

Processamento predefinido

Teste seus conhecimentos

Amanda Paolini
384.846-2010-23
Amanda Paolini



IMPACTA
EDITORA

Introdução à Lógica de Programação

1. Dado o processamento predefinido:

SOMAR (X , Y , Z NUMÉRICAS)

$$Z = X + Y$$

SAÍDA

Faça o teste de mesa para o programa a seguir:

PROGRAMA1

 DECLARA A, B, C NUMÉRICAS

 A = 1

 B = 2

 C = 0

 SOMAR (A , B , C)

 SOMAR (B , C , A)

 SOMAR (C , A , B)

 SOMAR (A , B , C)

 SOMAR (B , C , A)

 EXIBIR A, B, C

FIM

Após o teste de mesa, os valores exibidos das variáveis A, B e C são:

- a) 5, 8, 3
- b) 5, 8, 13
- c) 5, 2, 3
- d) 21, 8, 13
- e) Nenhuma das alternativas anteriores está correta.

2. Dado o seguinte processamento predefinido:

```
TROCAR ( X , Y NUMÉRICAS)
    DECLARA AUX NUMÉRICA
    SE X > Y ENTÃO
        AUX = X
        X = Y
        Y = AUX
    FIM SE
SAÍDA
```

Faça o teste de mesa para o programa a seguir:

PROGRAMA2

```
    DECLARA A, B, C, D NUMÉRICAS
    A = 10
    B = 5
    C = 7
    D = 1
    TROCAR ( A , B )
    TROCAR ( B , C )
    TROCAR ( C , D )
    TROCAR ( A , B )
    TROCAR ( B , C )
    TROCAR ( A , B )
    EXIBIR A , B , C , D
FIM
```

Após o teste de mesa, os valores exibidos das variáveis A, B e C são:

- a) 1, 7, 1, 10
- b) 5, 1, 7, 10
- c) 1, 5, 7, 10
- d) 7, 1, 5, 10
- e) Nenhuma das alternativas anteriores está correta.

Introdução à Lógica de Programação

3. Dado os seguintes processamentos predefinidos:

SOMAR (X , Y , Z NUMÉRICAS)

$$Z = X + Y$$

SAÍDA

e

TROCAR (X , Y NUMÉRICAS)

DECLARA AUX NUMÉRICA

SE X > Y ENTÃO

$$\text{AUX}=X$$

$$X = Y$$

$$Y = \text{AUX}$$

FIM SE

SAÍDA

Faça o teste de mesa para o programa a seguir:

PROGRAMA3

DECLARA A, B, C, D NUMÉRICAS

$$A = 5$$

$$B = 2$$

$$C = 0$$

$$D = 0$$

SOMAR (A , B , C)

TROCAR (A , B)

SOMAR (B , C , D)

TROCAR (C , B)

TROCAR (A , D)

SOMAR (D , A , B)

EXIBIR A , B , C , D

FIM

Após o teste de mesa, os valores exibidos das variáveis A, B e C são:

- a) 2, 14, 7, 12
- b) 2, 14, 5, 12
- c) 5, 14, 7, 9
- d) 2, 12, 5, 9
- e) Nenhuma das alternativas anteriores está correta.

4. Dado os seguintes processamentos predefinidos:**SOMAR (X , Y , Z NUMÉRICAS)**

$$Z = X + Y$$

SAÍDA**e****MÉDIA (M , N , P NUMÉRICAS)**

$$M = (N + P) / 2$$

SAÍDA**Faça o teste de mesa para o programa a seguir:****PROGRAMA4****DECLARA A, B, C NUMÉRICAS**

$$A = 0$$

$$B = 2$$

$$C = 4$$

MÉDIA (A , B , C)**SOMAR (B , C , A)****MÉDIA (B , C , A)****SOMAR (C , A , B)****MÉDIA (C , A , B)****MÉDIA (A , B , C)****SOMAR (C , A , B)****MÉDIA (C , A , B)****EXIBIR A, B, C****FIM****Após o teste de mesa, os valores exibidos das variáveis A, B e C são:**

- a) 12, 14, 8
- b) 6, 14, 13
- c) 9, 17, 13
- d) 12, 17, 8
- e) Nenhuma das alternativas anteriores está correta.

Introdução à Lógica de Programação

5. Dado os seguintes processamentos predefinidos:

SOMAR (X , Y , Z NUMÉRICAS)

$$Z = X + Y$$

SAÍDA

e

MEDIA (M , N , P NUMÉRICAS)

$$M = (N + P) / 2$$

SAÍDA

e

TROCAR (X , Y NUMÉRICAS)

DECLARA AUX NUMÉRICA

SE X > Y ENTÃO

$$AUX = X$$

$$X = Y$$

$$Y = AUX$$

FIM SE

SAÍDA

Faça o teste de mesa para o programa a seguir:

PROGRAMA5

DECLARA A, B, C NUMÉRICAS

$$A = 1$$

$$B = 2$$

$$C = 3$$

MÉDIA (B , C, A)

SOMAR (B , C , A)

MÉDIA (B , C , A)

TROCAR (A, B)

SOMAR (A , B, C)

MÉDIA (A , B , C)

TROCAR (A, B)

MÉDIA (C , A , B)

EXIBIR A, B, C

FIM

Após o teste de mesa, os valores exibidos das variáveis A, B e C são:

- 7, 5, 12
- 5, 7, 6
- 5, 4, 6
- 12, 7, 9
- Nenhuma das alternativas anteriores está correta.

6

Processamento predefinido

Mãos à obra!



IMPACTA
EDITORIA

Introdução à Lógica de Programação

Laboratório 1

Exercício 1

Em um programa que verifica se um **CÓDIGO DE CARGO** é existente numa tabela com códigos que vão de 1 a 100, crie o processamento predefinido **CHECACODIGO()** para validar o código digitado. Se o código for válido, altere o valor da variável **CHECK** para que, no programa adiante, seja feita a busca da descrição do cargo, e para que esta seja exibida. Senão, exiba mensagem de erro e finalize.

```
VERIFICA_CARGO
    DECLARA CODCARGO, CHECK NUMÉRICAS
    CODCARGO = 0
    CHECK = 0
    EXIBIR "DIGITE O CÓDIGO DE UM CARGO"
    LER CODCARGO
    CHEGACODIGO( CODCARGO, CHECK )
    SE CHECK < > 0
        ENTÃO: BUSCAR A DESCRIÇÃO DO CARGO NA TABELA DE CARGOS
            EXIBIR A DESCRIÇÃO DO CARGO
    FIM SE
FIM
```

Laço ou loop e repetição

7

- ✓ Fluxograma com comando FOR...NEXT (PARA...PRÓXIMO);
- ✓ Fluxograma com comando WHILE (ENQUANTO);
- ✓ Fluxograma com comando IF...THEN...ELSE (SE...ENTÃO...SENÃO).



IMPACTA
EDITORA

7.1. Introdução

Neste capítulo, aprenderemos o conceito de laço e como escrever e utilizar os laços em programação.

Estruturas de laços e repetições são construídas para executar trechos de uma lógica várias vezes. Um laço em programação significa um retorno a linhas de programa já executadas para que sejam executadas novamente. Em inglês, laço significa **loop**, no sentido de retornar dando uma volta, fazendo um círculo.

Os laços de repetição se encarregam de repetir determinados comandos enquanto uma determinada condição for satisfeita.

Os comandos descritos a seguir são utilizados para efetuar um laço dentro de uma lógica de programação.

7.2. Comando FOR...NEXT (PARA...PRÓXIMO)

Um loop com o comando **FOR** é realizado quando uma condição com operação matemática é checada para executar um trecho de programa até o passo em que essa condição seja satisfeita (verdadeira).

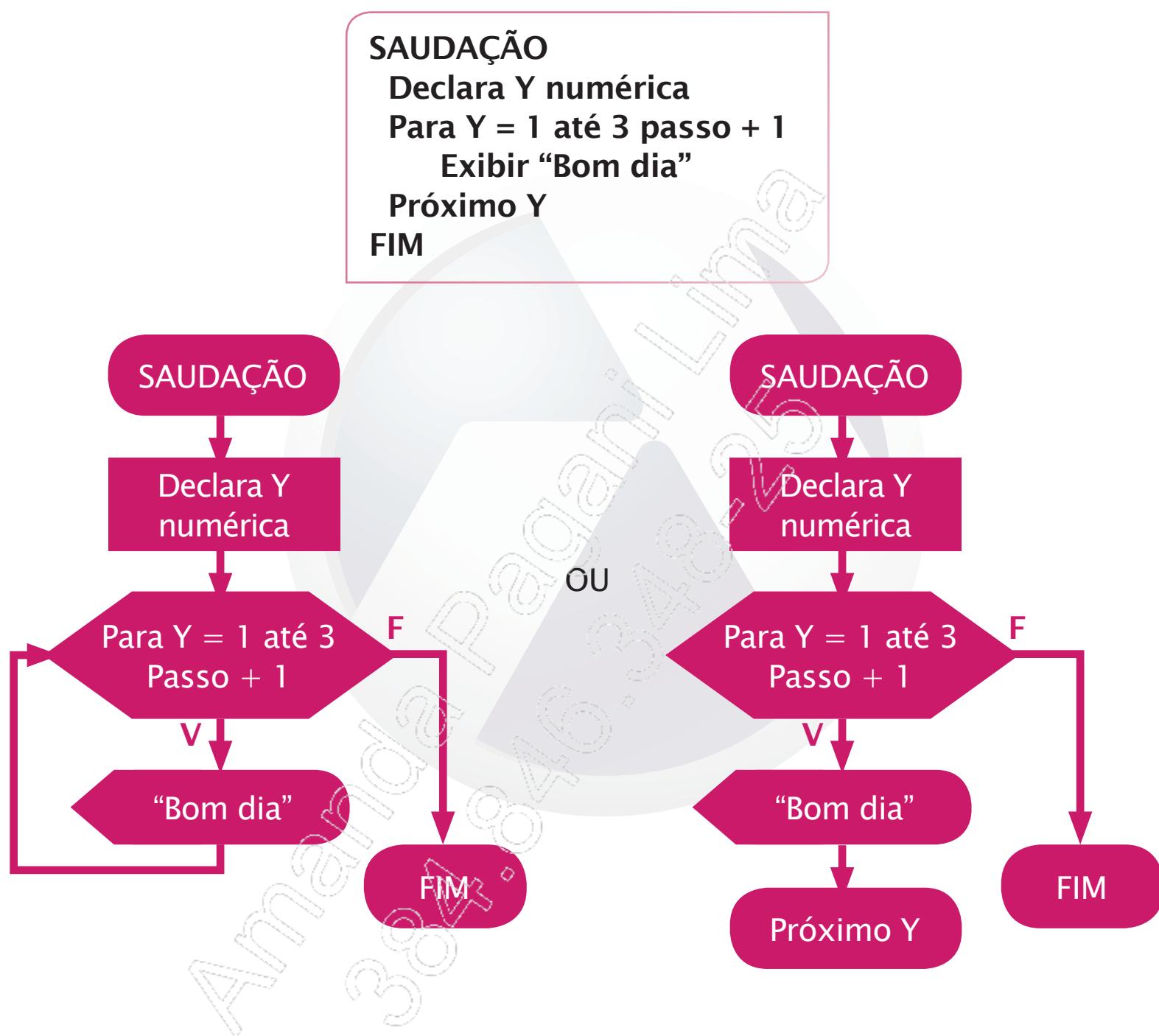
Na sintaxe do comando **FOR**, declaramos sua inicialização, sua condição e seu incremento. A cada vez que o loop é executado, a variável do comando que é utilizada como contador aumenta ou diminui de acordo com o incremento, até que a condição do comando não seja mais satisfeita. Esse tipo de loop é uma repetição contável.

FOR (PARA)

NEXT (PRÓXIMO)
(FIM PARA)

SÍMBOLO DE
PREPARAÇÃO

A seguir, temos um exemplo de algoritmo e fluxograma de um programa com o comando **PARA** que exibe três vezes a mensagem “Bom dia”.



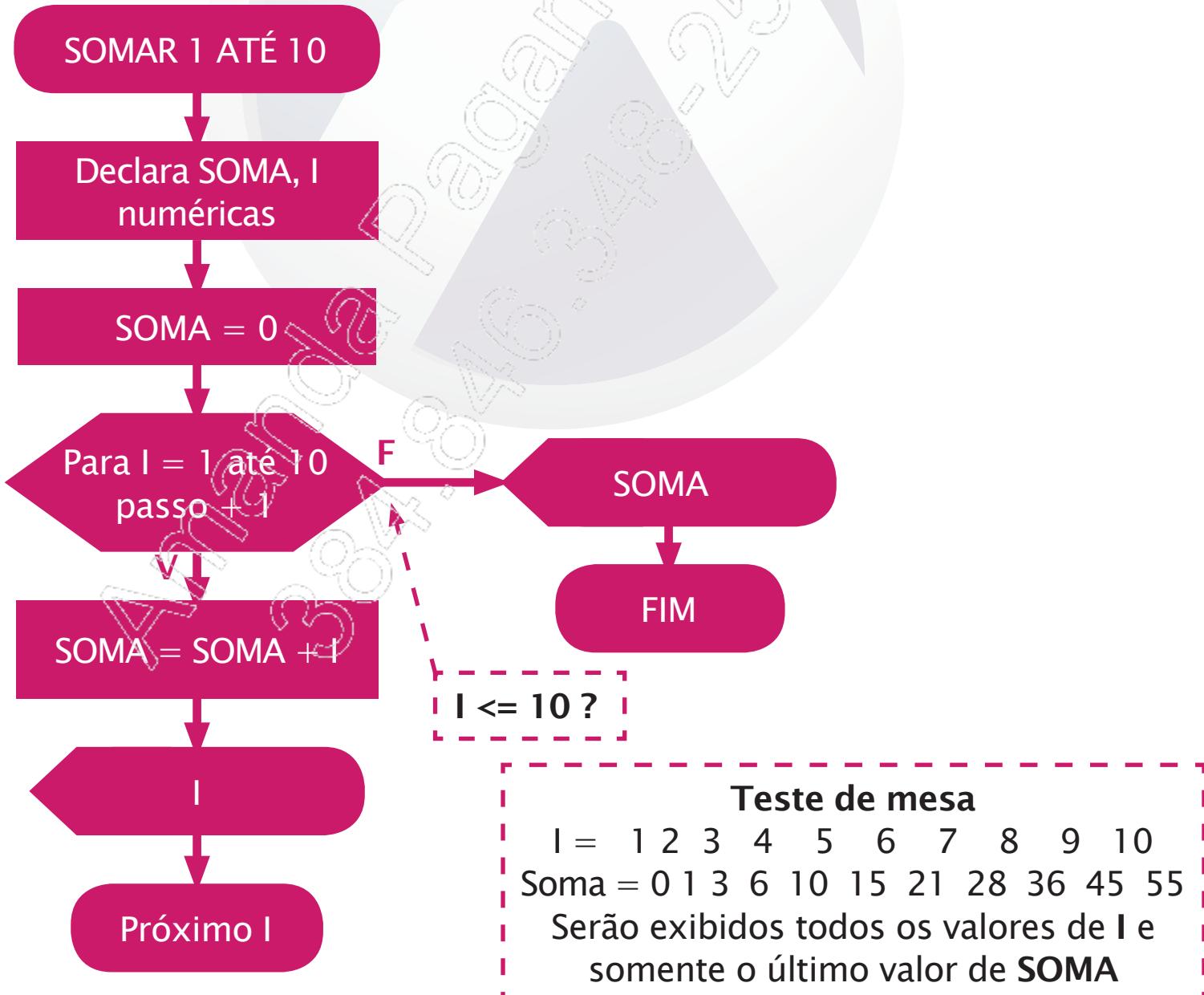
Note que no fluxo da direita está sendo utilizado um símbolo de terminação com o comando **PRÓXIMO Y**. Esse símbolo representa a seta do fluxo que retorna ao símbolo de preparação, conforme mostra o fluxo da esquerda. Depois deste símbolo não há mais símbolos. As duas estruturas são corretas.

Introdução à Lógica de Programação

A seguir, temos um exemplo de algoritmo com o comando **PARA** que soma os números de 1 (um) até 10 (dez), exibindo cada número somado e o valor total da soma:

```
SOMAR 1 ATÉ 10
Declara SOMA, I numéricas
SOMA = 0
Para I = 1 até 10 passo + 1
    SOMA = SOMA + I
    Exibir I
    Próximo I
    Exibir SOMA
FIM
```

A seguir, temos um fluxograma com o comando **PARA** que soma os números de 1 (um) até 10 (dez), exibindo cada número somado e o valor total da soma:



7.3. Comando WHILE (ENQUANTO)

Um loop com o comando **WHILE** é realizado quando uma condição é checada para executar um trecho de programa até o passo em que essa condição seja satisfeita (verdadeira). Trata-se de um loop de repetição contável.

Na sintaxe do comando **WHILE**, declaramos apenas sua condição. Sua inicialização e seu incremento são ações executadas separadamente.

O loop com o comando **WHILE** é utilizado quando não conhecemos o número de iterações que temos que realizar. Esta é a diferença do loop **WHILE** com o comando **FOR**.

WHILE (ENQUANTO)

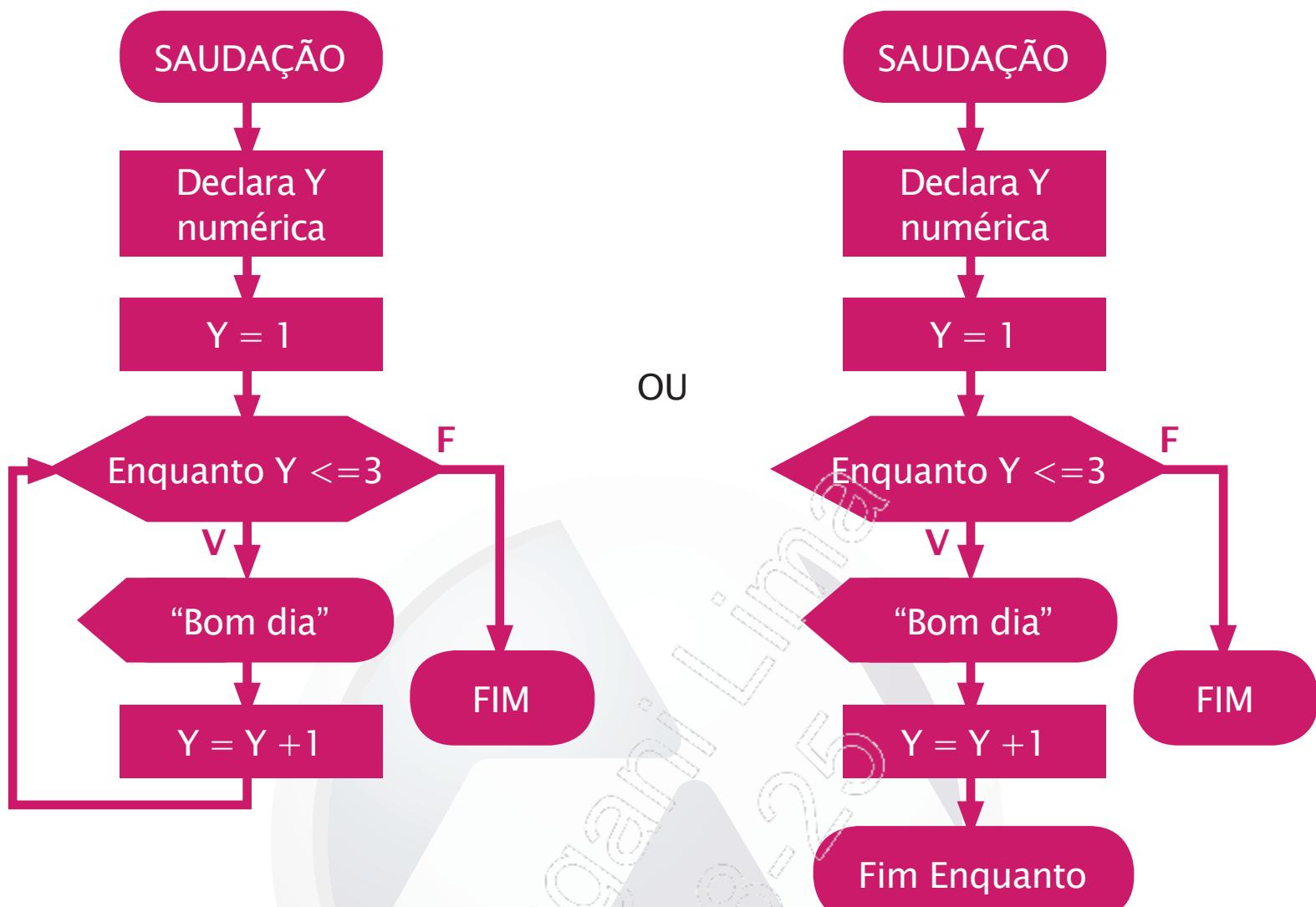
WEND (FIM ENQUANTO)

SÍMBOLO DE
PREPARAÇÃO

A seguir, temos um exemplo de algoritmo e fluxograma de um programa com o comando **ENQUANTO** que exibe três vezes a mensagem “**Bom dia**”:

```
Declarar Y numérica
Y = 1
Enquanto Y <= 3
    Exibir “Bom dia”
    Y = Y + 1
Fim Enquanto
FIM
```

Introdução à Lógica de Programação

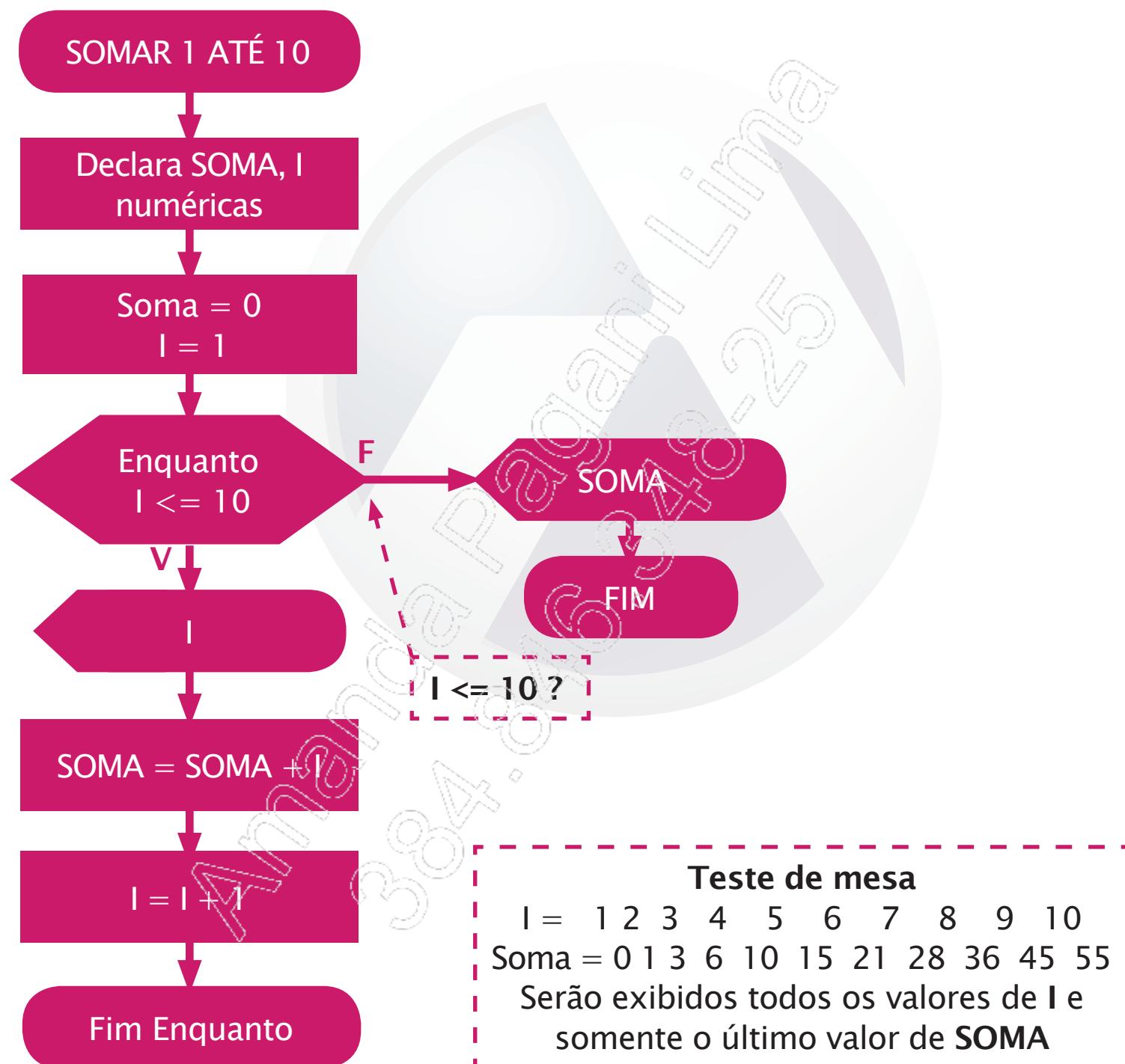


Note que no fluxo da direita está sendo utilizado um símbolo de terminação com o comando **FIM ENQUANTO** no lugar das setas. As duas estruturas são corretas.

A seguir, temos um exemplo de algoritmo com o comando **ENQUANTO** que soma os números de 1 (um) até 10 (dez), exibindo cada número somado e o valor total da soma:

```
SOMAR 1 ATÉ 10
Declara SOMA, I numéricas
SOMA = 0
I = 1
Enquanto I <= 10
    Exibir I
    SOMA = SOMA + I
    I = I + 1
Fim Enquanto
Exibir SOMA
FIM
```

A seguir, temos um fluxograma com o comando ENQUANTO que soma os números de 1 (um) até 10 (dez), exibindo cada número somado e o valor total da soma:



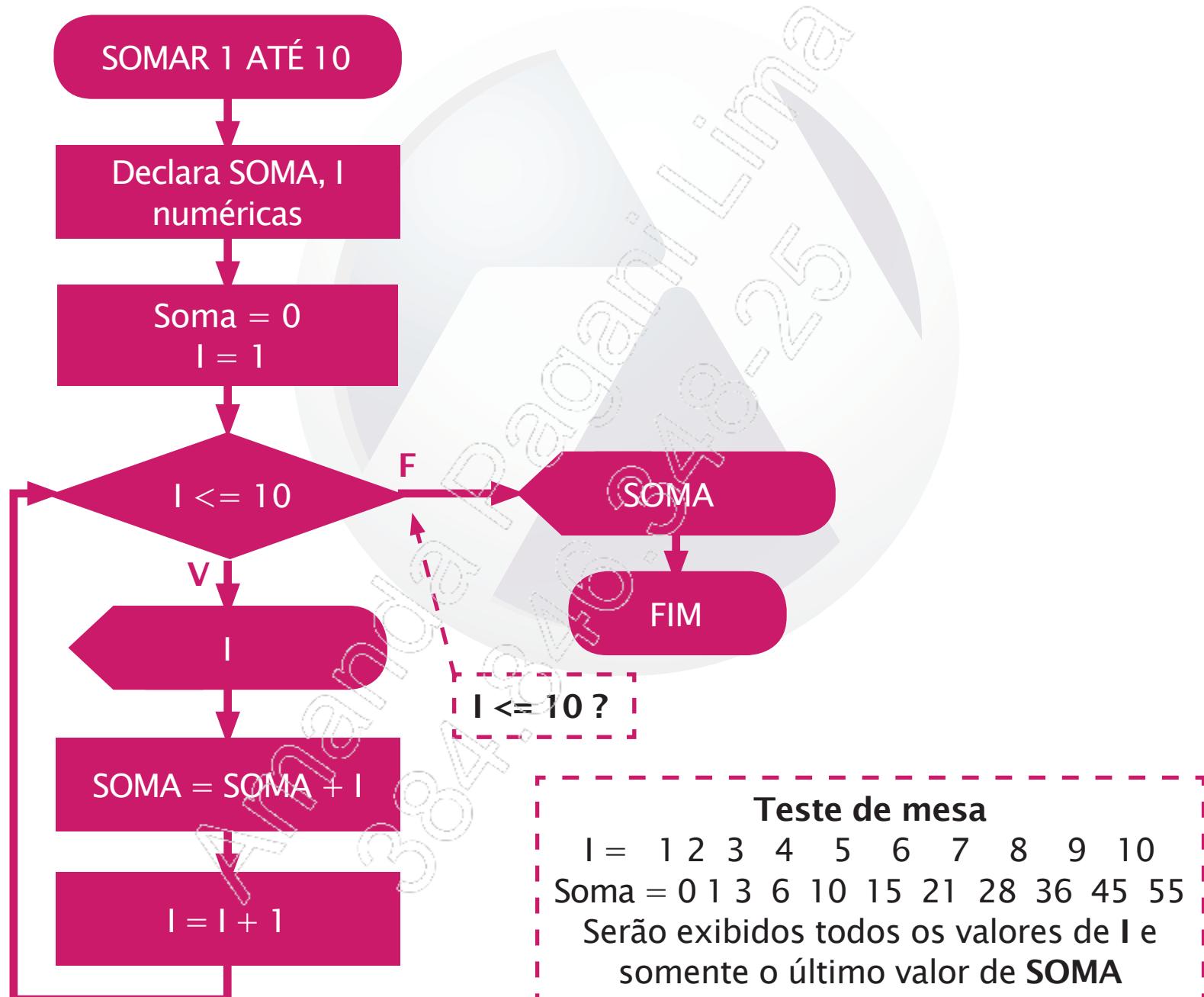
7.4. Comando IF...THEN...ELSE (SE...ENTÃO...SENÃO)

Um loop com o comando **SE** é realizado quando uma condição é checada e, dependendo do resultado dela, pode-se executar novamente um trecho de programa. Trata-se de um loop de repetição condicional.

A seguir, temos um exemplo de algoritmo com o comando **SE** que soma os números de 1 (um) até 10 (dez), exibindo cada número somado e o valor total da soma:

```
SOMAR 1 ATÉ 10
    Declara SOMA, I numéricas
    SOMA = 0
    I = 1
    Se I <= 10
        Então: Exibir I
        SOMA = SOMA + I
        I = I + 1
    Senão: Exibir SOMA
    Fim Se
FIM
```

A seguir, temos um exemplo de fluxograma com o comando **SE** que soma os números de 1 (um) até 10 (dez), exibindo cada número somado e o valor total da soma:



Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Estruturas de laços e repetições são construídas para executar trechos de uma lógica várias vezes;
- Um laço em programação significa um retorno a linhas de programa já executadas para que sejam executadas novamente;
- Em inglês, **laço** significa **loop**, no sentido de retornar dando uma volta, fazendo um círculo;
- Os laços de repetição se encarregam de repetir determinados comandos enquanto uma determinada condição for verdadeira;
- Um loop com o comando **FOR** é realizado quando uma condição com operação matemática é verificada para executar um trecho de programa até o passo em que essa condição seja satisfeita (verdadeira);
- Na sintaxe de **FOR**, declaramos sua inicialização, sua condição e seu incremento. A cada vez que o loop é executado, a variável do comando que é utilizada como contador aumenta ou diminui conforme o incremento, até que a condição do comando não seja mais satisfeita;
- Um loop com o comando **WHILE** é realizado quando uma condição é checada para executar um trecho de programa até o passo em que essa condição seja satisfeita (verdadeira);
- Na sintaxe de **WHILE**, declaramos apenas sua condição. Sua inicialização e seu incremento são ações executadas separadamente. O loop com o comando **WHILE** é usado quando não se conhece o número de iterações a serem realizadas;
- Um loop com o comando **SE** é realizado quando uma condição é verificada e, dependendo do resultado dela, pode-se executar novamente um trecho de programa.

7

Laço ou loop e repetição

Teste seus conhecimentos

Amanhã p/ amanhã
384.846.25



IMPACTA
EDITORA

Introdução à Lógica de Programação

1. Faça o teste de mesa para o algoritmo a seguir:

PROGRAMA1

```
Declara SOMA, I numéricas  
SOMA = 0  
Para I = 1 até 5 passo + 1  
    SOMA = SOMA + 2 * I  
Próximo I  
Exibir SOMA, I  
FIM
```

Após o teste de mesa, os valores exibidos das variáveis SOMA e I são:

- a) 30 e 5
- b) 12 e 4
- c) 20 e 5
- d) 30 e 6
- e) Nenhuma das alternativas anteriores está correta.

2. Faça o teste de mesa para o algoritmo a seguir:

PROGRAMA2

```
Declara SOMA, I numéricas  
SOMA = 0  
Para I = 1 até 10 passo + 2  
    SOMA = SOMA + I  
Próximo I  
Exibir SOMA, I  
FIM
```

Após o teste de mesa, os valores exibidos das variáveis SOMA e I são:

- a) 25 e 11
- b) 16 e 11
- c) 25 e 9
- d) 16 e 9
- e) Nenhuma das alternativas anteriores está correta.

Introdução à Lógica de Programação

3. Faça o teste de mesa para o algoritmo a seguir:

PROGRAMA3

Declarar SOMA, I numéricas

SOMA = 0

Para I = 1 até 5 passo + 1

SOMA = SOMA + I / 2

Próximo I

Exibir SOMA, I

FIM

Após o teste de mesa, os valores exibidos das variáveis SOMA e I são:

- a) 5 e 6
- b) 5 e 5
- c) 7,5 e 5
- d) 7,5 e 6
- e) Nenhuma das alternativas anteriores está correta.

4. Faça o teste de mesa para o algoritmo a seguir:

PROGRAMA4

Declarar A, B numéricas

A = 17

B = 0

Enquanto A >= 3

 A = A - 3

 B = B + 1

Fim Enquanto

Exibir A, B

FIM

Após o teste de mesa, os valores exibidos das variáveis A e B são:

- a) 8 e 3
- b) 5 e 5
- c) 2 e 5
- d) 5 e 4
- e) Nenhuma das alternativas anteriores está correta.

Introdução à Lógica de Programação

5. Faça o teste de mesa para o algoritmo a seguir:

PROGRAMA5

Declarar A, B numéricas

A = 42

B = 0

Enquanto A MOD 5 <> 0

 A = A - 3

 B = B + 1

Fim Enquanto

Exibir A, B

FIM

Após o teste de mesa, os valores exibidos das variáveis A e B são:

- a) 33 e 3
- b) 36 e 3
- c) 30 e 4
- d) 36 e 4
- e) Nenhuma das alternativas anteriores está correta.

Laço ou loop e repetição

Mãos à obra!

7

Amanhã é dia de
384.000.000.
Amanhã é dia de
384.000.000.



IMPACTA
EDITORA

Introdução à Lógica de Programação

Laboratório 1

Exercício 1

Faça o teste de mesa para o algoritmo a seguir:

INÍCIO

```
Declara SOMA, I Numéricas
SOMA = 0
I = 1
Enquanto I <= 8
    Se I MOD 2 = 1 então
        SOMA = SOMA + I
    Fim se
    I = I + 1
Fim Enquanto
Exibir SOMA, I
FIM
```

Exercício 2

Faça o teste de mesa para o algoritmo a seguir:

INÍCIO

```
Declara A, B, C, D, E, G, H, J Numéricas
A = 1
Para B = 1 até 3 passo +1
    A = A + 5
    C = 1
    Exibir A
Próximo B
D = 2
Se C < 3
    Então: E = D * 4
    Senão: E = D**4
Fim Se
G = 6
Para H = 2 até 6 passo + 2
    G = G - 1
    A = A + 1
Próximo H
J = G + D
Exibir A, E, J
FIM
```

Variáveis indexadas e Laços encadeados

8

- ✓ Vetores e matrizes;
- ✓ Laços encadeados.

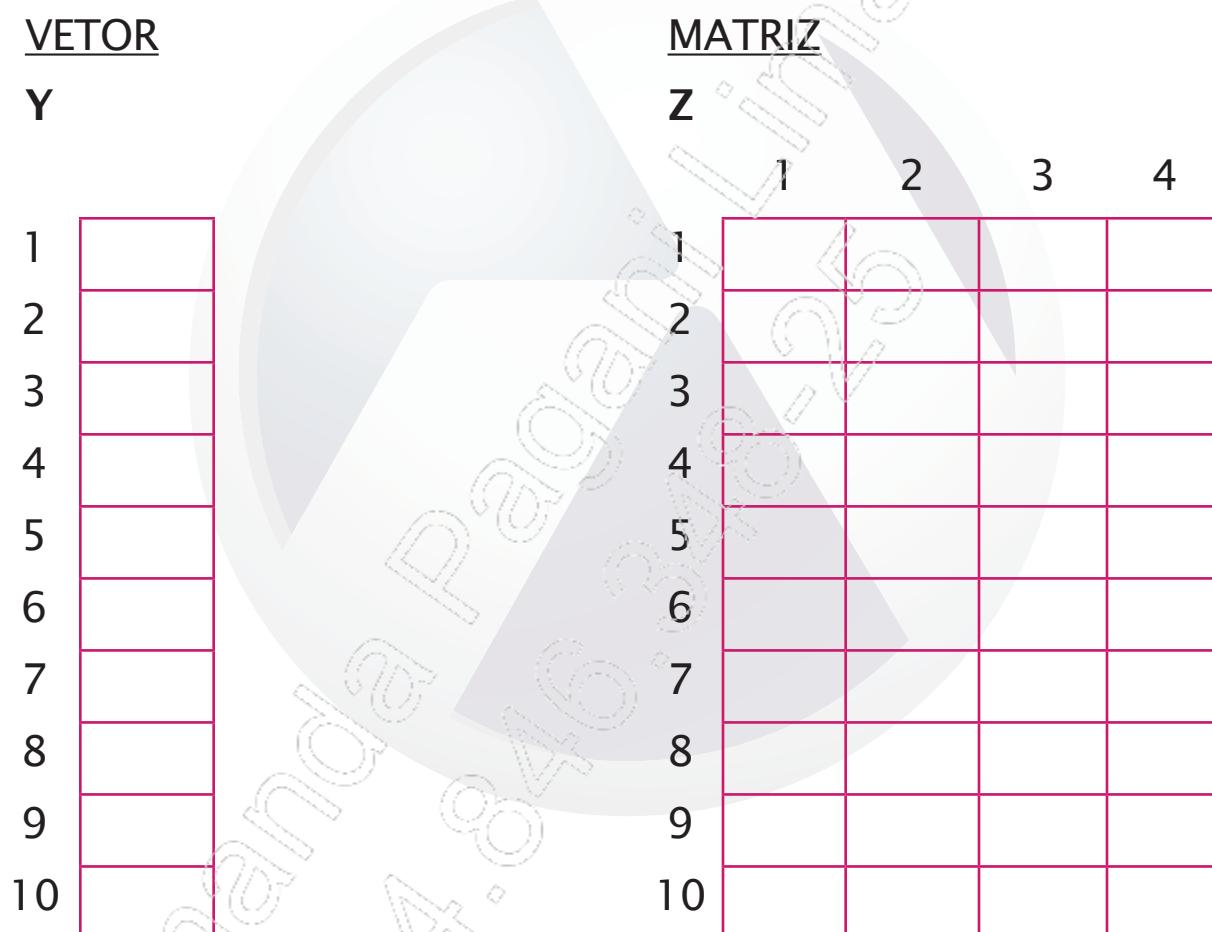


IMPACTA
EDITORA

8.1. Vetores e matrizes

Neste capítulo aprenderemos a utilizar as variáveis indexadas em programação. **Variáveis indexadas** são um conjunto de variáveis que apresentam o mesmo nome, são do mesmo tipo, mas são diferentes no valor de seu índice. As variáveis indexadas podem ter várias dimensões:

- **Vetores**: uma dimensão;
- **Matrizes**: n dimensões.



Dados dois números inteiros positivos m e n , chama-se matriz $m \times n$ a tabela formada por $m \cdot n$ números reais, dispostos em m linhas (horizontais) e n colunas (verticais).



Importante: Cada elemento é indicado por a_{ij} , em que i indica a linha e j , a coluna, às quais a_{ij} pertence.

Variáveis indexadas e Laços encadeados

Um elemento de uma tabela pode ser referenciado de duas formas: **Implícita** e **Explícita**. Vejamos a seguinte tabela:



- **Referência implícita:** Usamos o índice para nos referenciar a um certo elemento da tabela. A seguir, temos um exemplo de referência implícita, considerando a tabela de dias da semana mostrada anteriormente:

$$A = 2 \rightarrow DIA[A] = \text{Terça-feira}$$

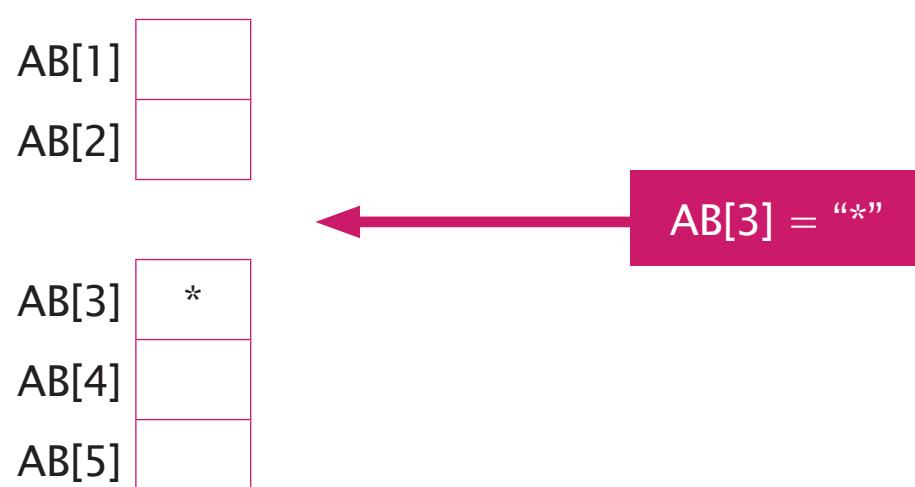
- **Referência explícita:** Referenciamos-nos diretamente ao elemento desejado. A seguir, temos um exemplo de referência explícita, considerando nossa tabela de dias da semana:

$$DIA[2] = \text{Terça-feira}$$

Vejamos estes outros exemplos:

- **Exemplo 1**

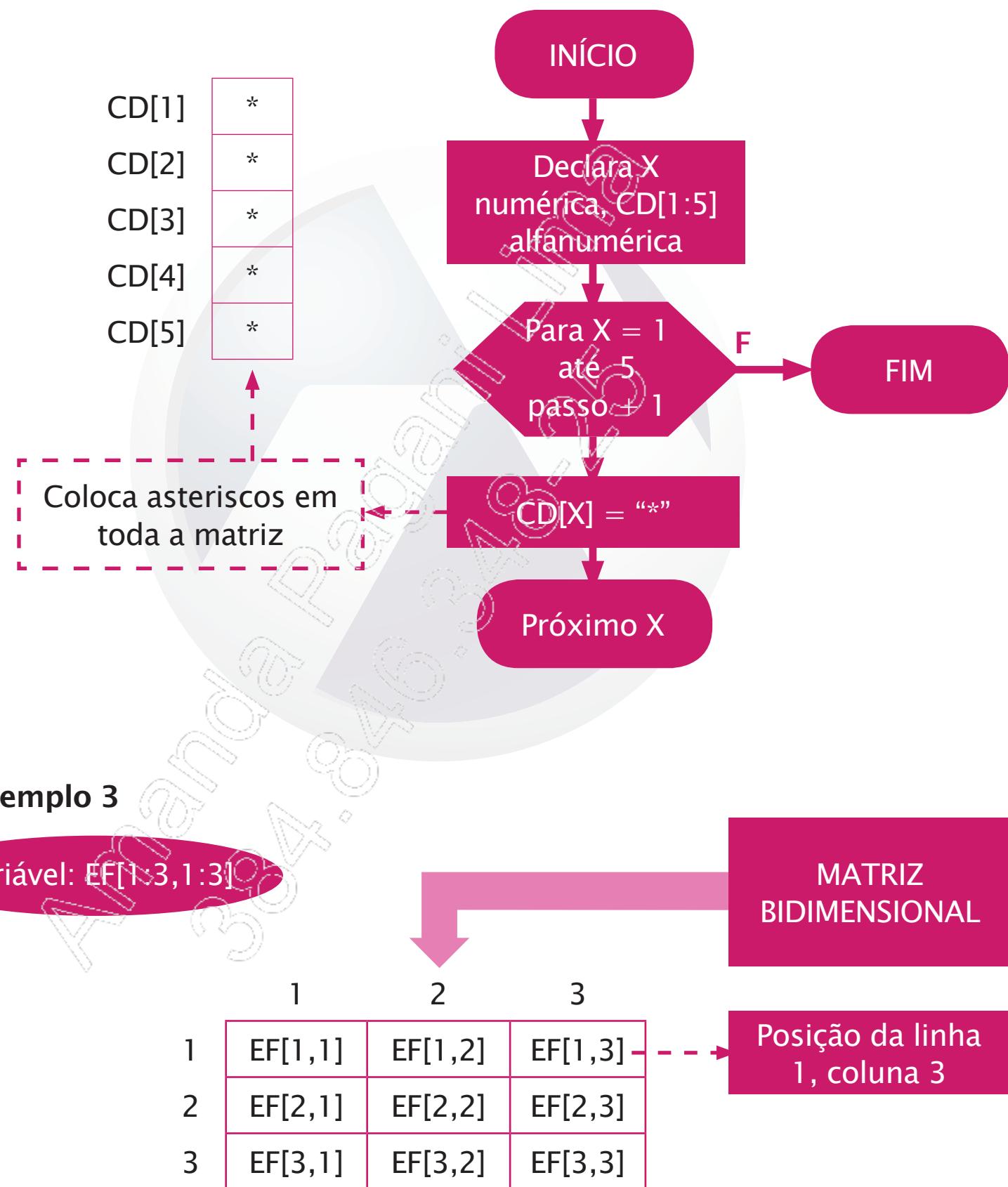
Variável: AB[1:5]



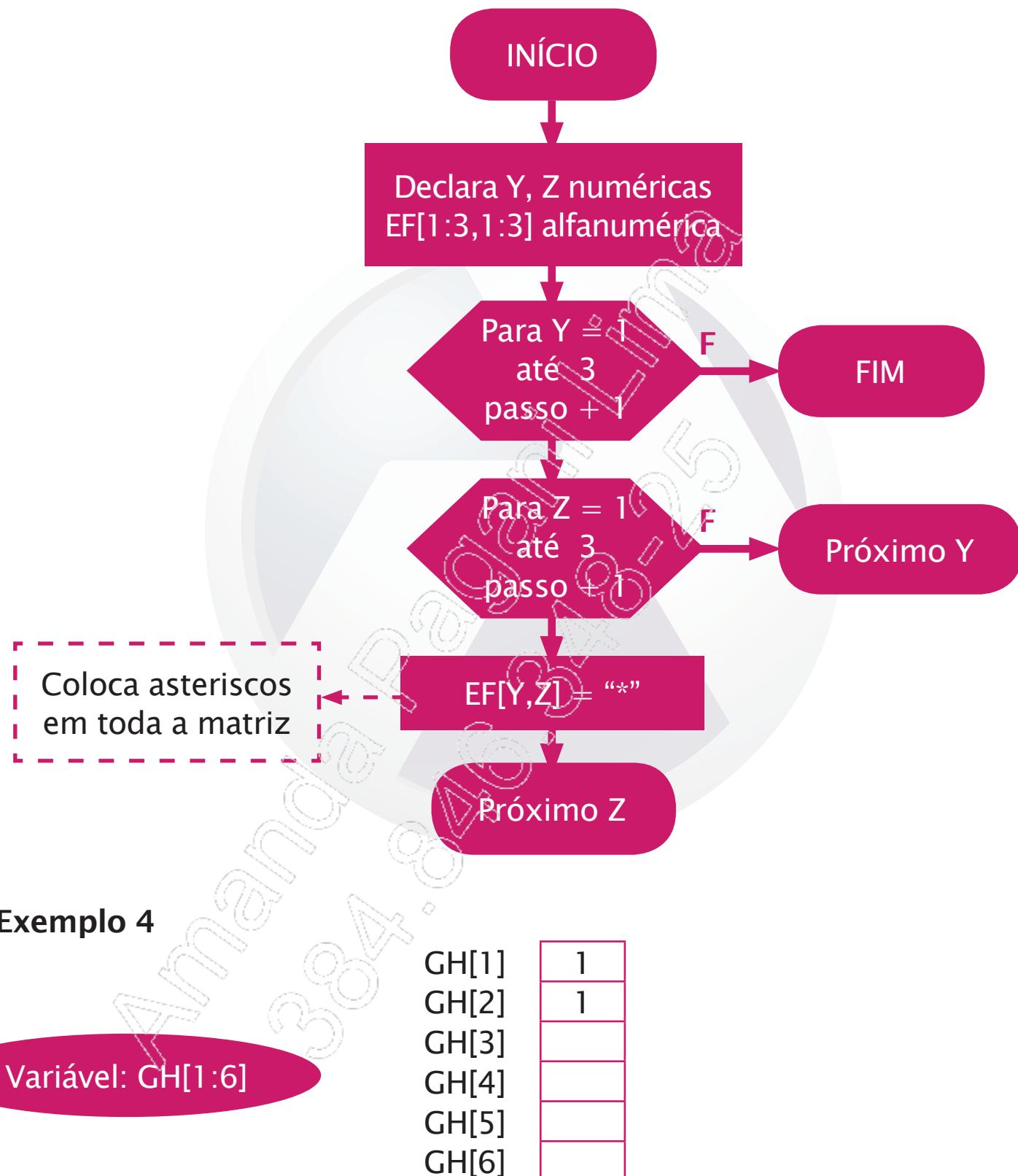
Introdução à Lógica de Programação

- **Exemplo 2**

Variável: CD[1:5]



Variáveis indexadas e Laços encadeados



Introdução à Lógica de Programação

A seguir, temos o algoritmo e o teste de mesa do exemplo 4:

- **Algoritmo**

```
INÍCIO
    Declara GH[1:6], J numéricas
    GH[1] = 1
    GH[2] = 1
    Para J = 3 até 6 passo + 1
        GH[J] = GH[J - 1] + GH[J - 2]
        Próximo J
    FIM
```

- **Teste de Mesa**

```
GH[ 3 ] = GH[ 3 - 1 ] + GH[ 3 - 2 ]
GH[ 2 ] + GH[ 1 ]
2

GH[ 4 ] = GH[ 4 - 1 ] + GH[ 4 - 2 ]
GH[ 3 ] + GH[ 2 ]
3

GH[ 5 ] = GH[ 5 - 1 ] + GH[ 5 - 2 ]
GH[ 4 ] + GH[ 3 ]
5

GH[ 6 ] = GH[ 6 - 1 ] + GH[ 6 - 2 ]
GH[ 5 ] + GH[ 4 ]
8
```

Variáveis indexadas e Laços encadeados

A seguir, temos um algoritmo e o fluxograma correspondente que verifica se um número digitado é encontrado num vetor. Será exibida uma mensagem informando se o número foi encontrado ou não. Como vetor, vamos considerar **VET [1:10]**.

INÍCIO

Declara VET [1:10], NUM, L numéricas, MSG alfanumérica

MSG = “Não encontrou”

NUM = 0

Ler NUM

Para L = 1 até 10 passo + 1

Se VET [L] = NUM

Então MSG = “Número encontrado”

L = 10

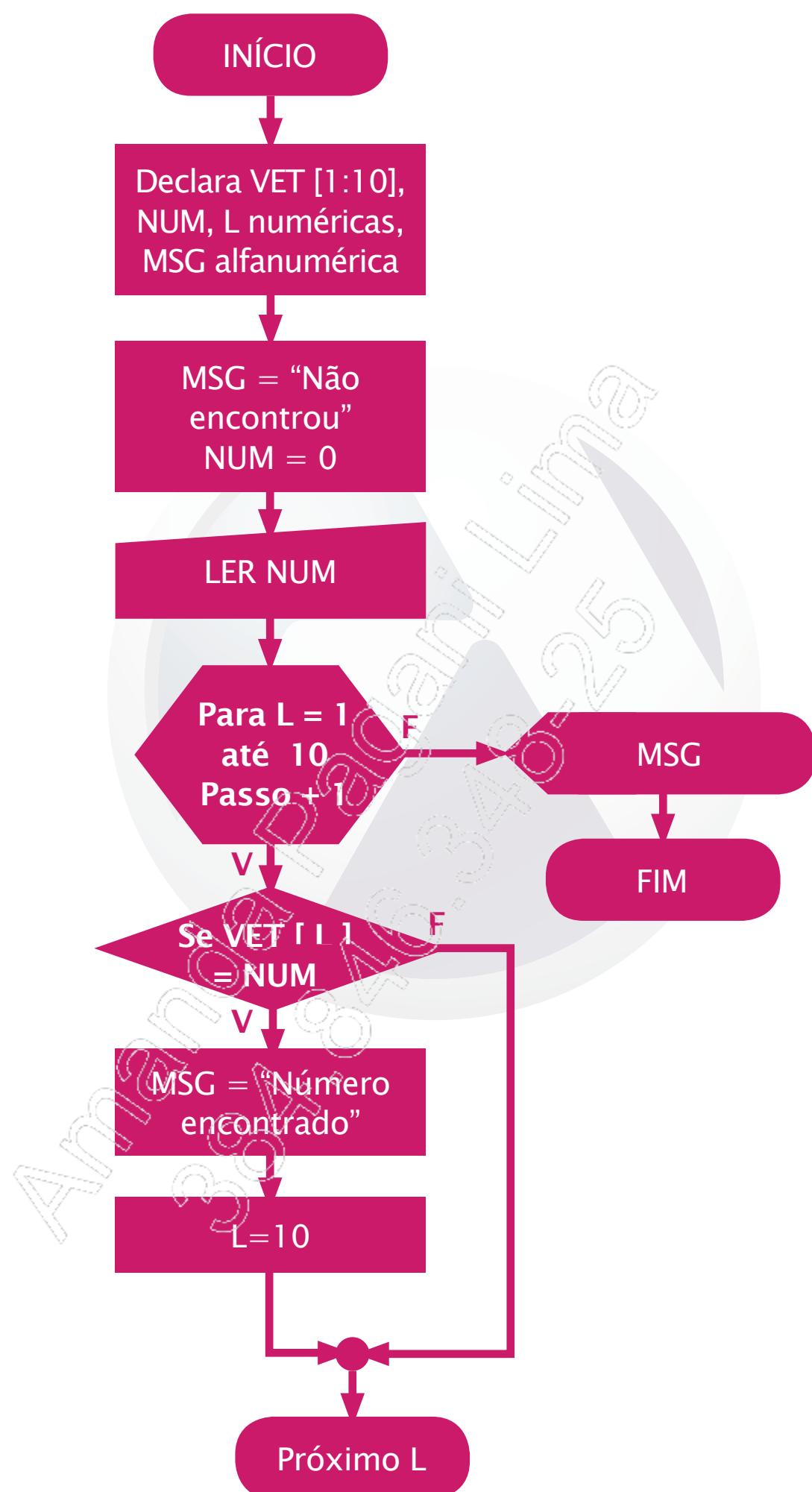
Fim Se

Próximo L

Exibir MSG

FIM

Introdução à Lógica de Programação

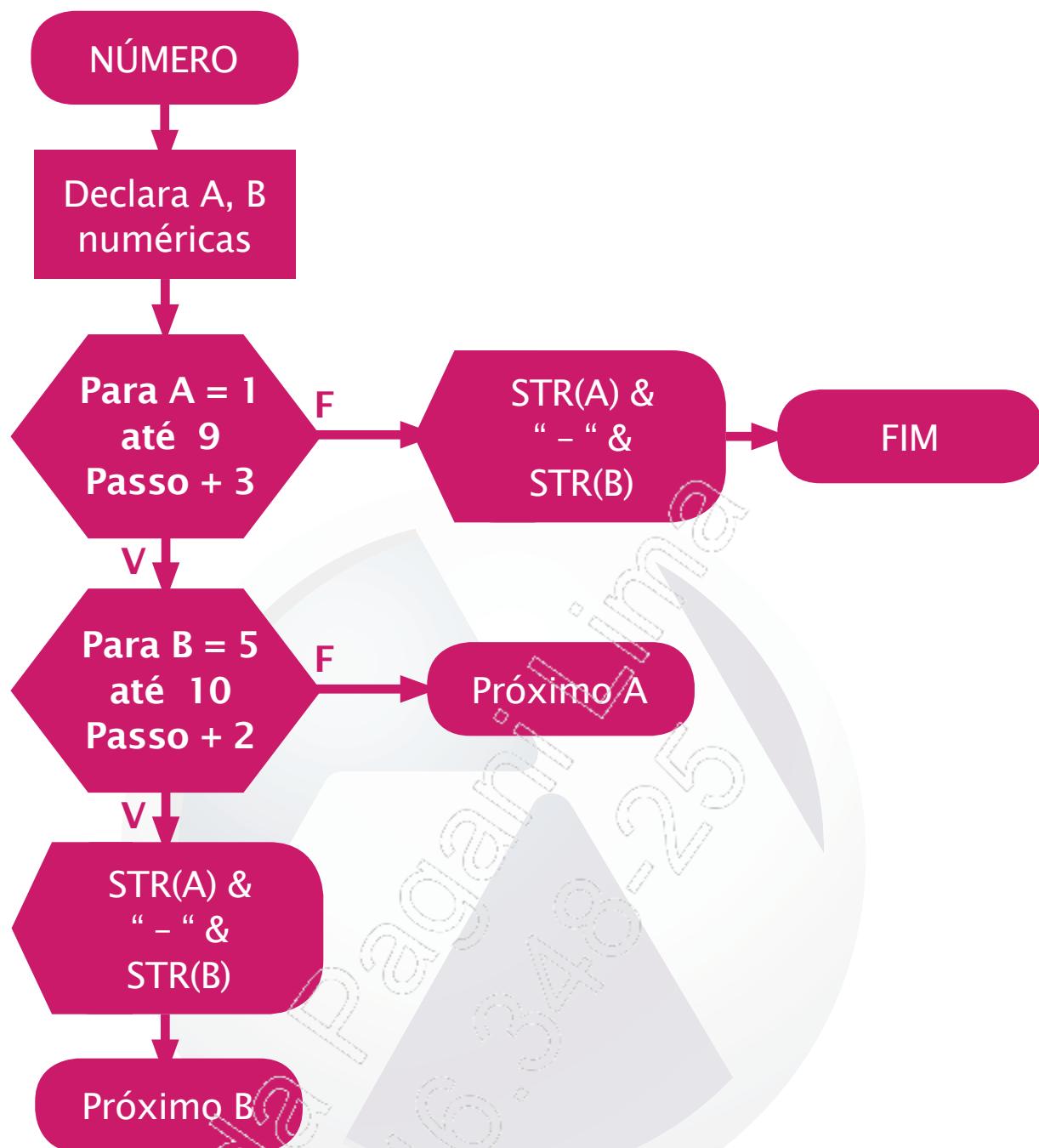


8.2. Laços encadeados

Laços ou Loops encadeados são laços executados dentro de outros laços. No caso do comando PARA, o primeiro a ser criado é o último a ser fechado. O comando PRÓXIMO fecha o laço. O último comando PARA aberto é o primeiro a ser fechado com o comando PRÓXIMO, ou seja, o último loop criado é o primeiro a ser fechado. Vejamos um exemplo de laço encadeado a seguir:

```
INÍCIO
    Declara A, B numéricas
    Para A = 1 até 9 passo + 3
        Para B = 5 até 10 passo + 2
            Exibir A, “ - “, B
            Próximo B
        Próximo A
        Exibir A , “ - “, B
    FIM
```

Introdução à Lógica de Programação



Teste de mesa		
A	-	B
1	-	5
1	-	7
1	-	9
4	-	5
4	-	7
4	-	9
7	-	5
7	-	7
7	-	9
10	-	11

Variáveis indexadas e Laços encadeados

Neste outro exemplo, temos um algoritmo que preenche a matriz conforme a figura utilizando apenas um laço (loop):

Matriz
LIN

	1	2	3	4	5	6	7	8	9	0
1										
2										
3										
4										
5		X	X	X	X	X	X	X	X	
6										
7										
8										
9										
0										

LINHA

Declara C numérica, LIN[1:10,1:10] alfanumérica

Para C = 2 até 9 passo + 1

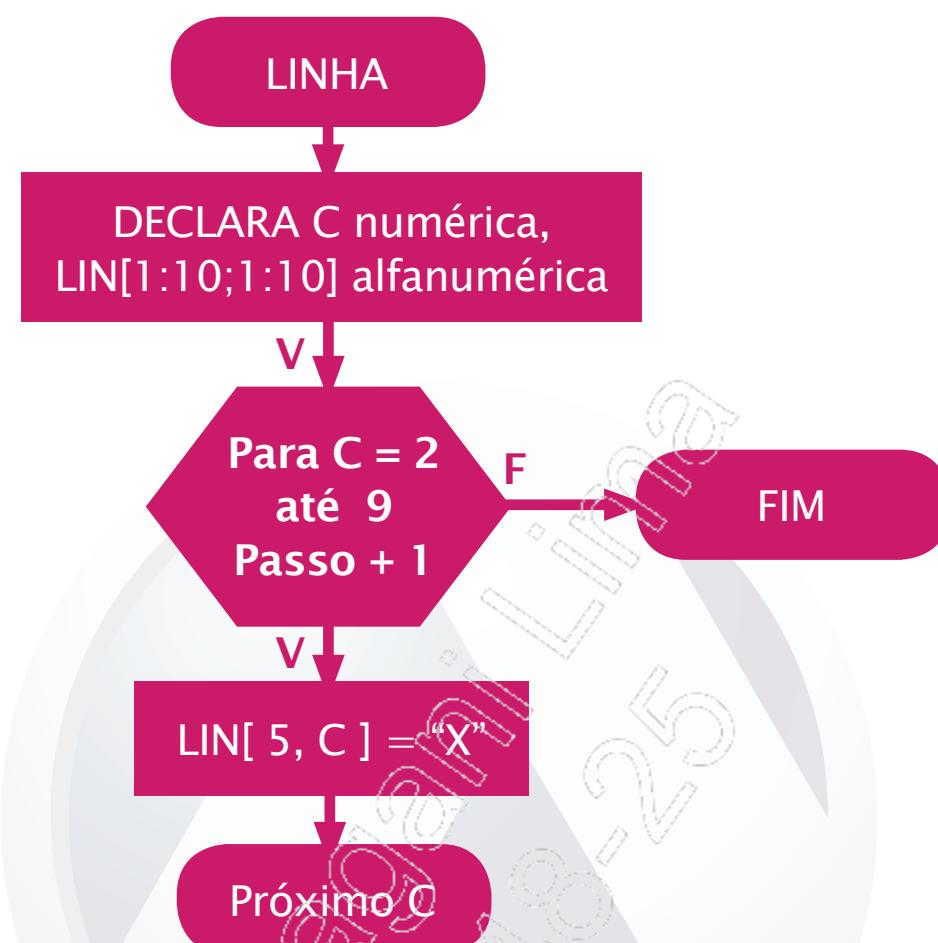
LIN[5, C] = "X"

Próximo C

FIM

Note que foram preenchidas as posições, na linha 5, da coluna 2 até a 9.

Introdução à Lógica de Programação



A seguir, temos um algoritmo que preenche a matriz conforme a figura utilizando laço (loop) encadeado:

Matriz Q	1	2	3	4	5	6	7	8	9	0
1										
2	X	X	X	X	X	X	X	X	X	
3	X	X	X	X	X	X	X	X	X	
4	X	X	X	X	X	X	X	X	X	
5	X	X	X	X	X	X	X	X	X	
6	X	X	X	X	X	X	X	X	X	
7	X	X	X	X	X	X	X	X	X	
8	X	X	X	X	X	X	X	X	X	
9										
0										

Variáveis indexadas e Laços encadeados

QUADRADO

Declara L, C numéricas, Q[1:10,1:10] alfanumérica

Para L = 2 até 8 passo + 1

Para C = 2 até 8 passo + 1

Q[L, C] = "X"

Próximo C

Próximo L

FIM

Vejamos o teste de mesa do último exemplo. Considerando que a variável **L** está sendo usada para a posição da linha e a variável **C** está sendo usada para a posição da coluna, veja, na tabela a seguir, que foi preenchida toda a matriz, desde a posição [2,2] até a posição [8,8].

L	C	L	C	L	C	L	C	L	C	L	C
2	2	3	2	4	2	5	2	6	2	7	2
2	3	3	3	4	3	5	3	6	3	7	3
2	4	3	4	4	4	5	4	6	4	7	4
2	5	3	5	4	5	5	5	6	5	7	5
2	6	3	6	4	6	5	6	6	6	7	6
2	7	3	7	4	7	5	7	6	7	7	8
2	8	3	8	4	8	5	8	6	8	7	8

Introdução à Lógica de Programação

A seguir, temos um algoritmo que preenche a matriz conforme a figura utilizando dois laços:

Matriz
TR

	1	2	3	4	5	6	7	8	9	0
1										
2										
3					X					
4				X			X			
5			X					X		
6	X	X	X	X	X	X	X	X	X	
7										
8										
9										
0										

TRIANGULO

Declara L, C, D numéricas, TR[1:10,1:10]
alfanumérica

C = 4

D = 6

Para L = 4 até 6 passo + 1

 TR[L, C] = "X"

 TR[L, D] = "X"

 C = C - 1

 D = D + 1

Próximo L

Para C = 3 até 7 passo + 1

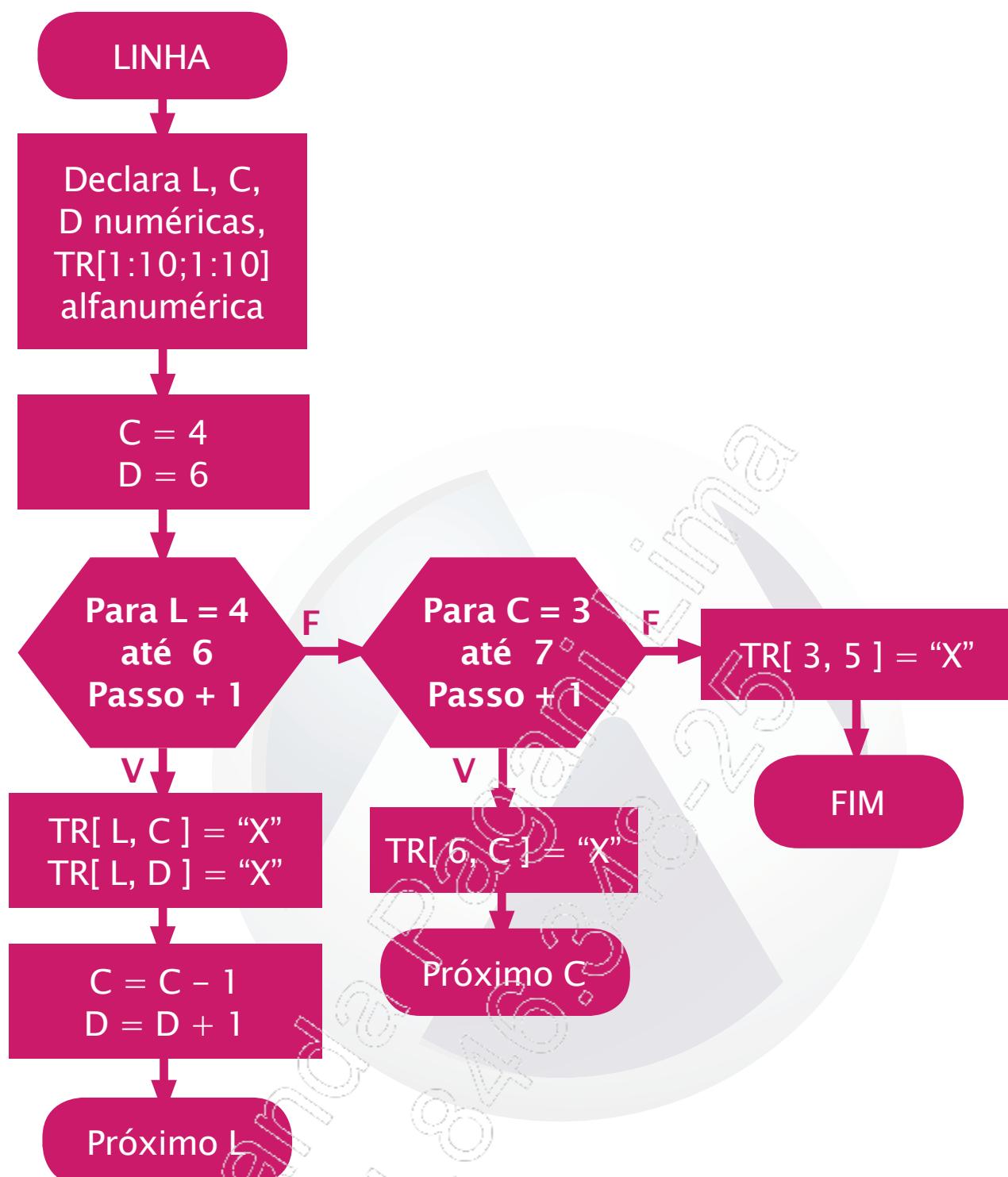
 TR[6, C] = "X"

Próximo C

TR[3, 5] = "X"

FIM

Variáveis indexadas e Laços encadeados



Considerando o exemplo do triângulo na matriz **TR**, observe, a seguir, as linhas e colunas preenchidas pelos valores das variáveis **L**, **C** e **D**:

LINHA (L)	COLUNA (C)
4	4
5	3
6	2

LINHA (L)	COLUNA (D)
4	6
5	7
6	8

Introdução à Lógica de Programação

Note que, se somarmos os valores das variáveis **L** e **C**, o resultado é sempre 8 (oito), portanto **C = 8 - L**. Após encontrarmos essa solução matemática, não será mais preciso utilizar a variável **C** no primeiro loop.

Perceba, também, que, se subtrairmos dos valores da variável **D** os valores da variável **L**, o resultado é sempre 2 (dois), portanto **D = 2 + L**. Após encontrarmos essa solução matemática, não será mais preciso utilizar a variável **D**.

L	+	C	=	8
4	+	4	=	8
5	+	3	=	8
6	+	2	=	8

D	-	L	=	2
6	-	4	=	2
7	-	5	=	2
8	-	6	=	2

Veja, a seguir, como ficou o algoritmo que preenche a matriz **TR**:

```
TRIANGULO
Declara L,C numéricas, TR[ 1:10,1:10 ] alfanumérica
Para L = 4 até 6 passo + 1
    TR[ L, 8 - L ] = "X"
    TR[ L, 2 + L ] = "X"
Próximo L
Para C = 3 até 7 passo + 1
    TR[ 6, C ] = "X"
Próximo C
TR[ 3, 5 ] = "X"
FIM
```

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- **Variáveis indexadas** são um conjunto de variáveis que apresentam o mesmo nome, são do mesmo tipo, mas são diferentes no valor de seu índice;
- As variáveis indexadas podem ter várias dimensões. **Vetores** apresentam uma dimensão, enquanto **matrizes** possuem n dimensões;
- **Laços ou Loops encadeados** são laços executados dentro de outros laços;
- Num encadeamento de laços, o primeiro loop criado é o último a ser fechado.

8

Variáveis indexadas e Laços encadeados

Teste seus conhecimentos

Amanhã parâmetros
384.646.



IMPACTA
EDITORA

Introdução à Lógica de Programação

1. Faça o teste de mesa para o algoritmo a seguir:

PROGRAMA1

Declarar A, B, C numéricas

A = 0

Para B = 1 até 4 passo + 1

 A = A + B

 Para C = 1 até 3 passo + 1

 B = B + 1

 Próximo C

Próximo B

Exibir A, B, C

FIM

Após o teste de mesa, os valores exibidos das variáveis A, B e C são:

- a) 1, 4, 4
- b) 1, 5, 4
- c) 2, 5, 4
- d) 2, 4, 3
- e) Nenhuma das alternativas anteriores está correta.

2. Faça o teste de mesa para o algoritmo a seguir:**PROGRAMA2****Declarar A, B, C numéricas****A = 5****Para B = 2 até 4 passo + 2** **A = A - B** **Enquanto A > 2** **C = B + 1** **A = A - 1** **Fim Enquanto****Próximo B****Exibir A, B, C****FIM****Após o teste de mesa, os valores exibidos das variáveis A, B e C são:**

- a) 2, 4, 3
- b) 2, 6, 2
- c) -2, 6, 3
- d) -2, 4, 3
- e) Nenhuma das alternativas anteriores está correta.

Introdução à Lógica de Programação

3. Faça o teste de mesa para o algoritmo a seguir:

PROGRAMA3

Declara A, B, C numéricas

A = 5

B = 1

Enquanto A > 2

A = A - B

Para B = 2 até 3 passo + 1

C = A + B

Próximo B

Fim Enquanto

Exibir A, B, C

FIM

Após o teste de mesa, os valores exibidos das variáveis A, B e C são:

- a) 0, 3, 2
- b) 4, 3, 2
- c) 4, 4, 3
- d) 0, 4, 3
- e) Nenhuma das alternativas anteriores está correta.

4. Faça o teste de mesa para o algoritmo a seguir:

PROGRAMA4

Declarar A, B, C numéricas

A = 5

B = 1

Enquanto A > 2

 A = A - B

 Enquanto A > 2

 C = A + B

 A = A - B

 Fim Enquanto

Fim Enquanto

Exibir A, B, C

FIM

Após o teste de mesa, os valores exibidos das variáveis A, B e C são:

- a) 3, 1, 1
- b) 2, 2, 4
- c) 2, 1, 4
- d) 3, 1, 2
- e) Nenhuma das alternativas anteriores está correta.

Introdução à Lógica de Programação

5. Faça o teste de mesa para o algoritmo a seguir:

PROGRAMA5

Declarar A, B, C numéricas

A = 5

Para B = 2 até 4 passo + 1

 A = A - B

 Enquanto A > 2

 C = B + A

 Se C < 6

 A = 2

 Fim Se

 Fim Enquanto

Próximo B

Exibir A, B, C

FIM

Após o teste de mesa, os valores exibidos das variáveis A, B e C são:

- a) -5, 5, 5
- b) -5, 4, 5
- c) 5, 4, -5
- d) 5, 5, 5
- e) Nenhuma das alternativas anteriores está correta.

8

Variáveis indexadas e Laços encadeados

Mãos à obra!

Amanhãs Paganini Lima
384.000
25



IMPACTA
EDITORA

Laboratório 1

Exercício 1

Dada a matriz M [14x13] e i = 7, preencher a matriz M com os valores a seguir:

M [3 , 4] = "A"

M [13 , i] = "B"

M [i , 8] = "C"

M [i , i] = "D"

M [i + 5 , 14] = "E"

M [9 , i - 4] = "F"

M [15 - i , i] = "G"

M [i + 3 , i - 6] = "H"

M	1	2	3	4	5	6	7	8	9	10	11	12	13
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													

Exercício 2

Dada a matriz **MAT [12x13]**, escreva um algoritmo usando laço para preencher a matriz conforme a figura a seguir:

MAT

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	V												
2	V	D											
3	V		D										
4	V			D									
5	V				D								
6	V					D							
7	V						D						
8	V							D					
9	V								D				
10	V									D			
11	V										D		
12	V	H	H	H	H	H	H	H	H	H	H	H	

Introdução à Lógica de Programação

Exercício 3

Dada a matriz T1 [14x20], escreva um algoritmo usando laço para preencher a matriz conforme a figura a seguir:

T1		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1																					
2				V																	
3				V	D																
4				V		D															
5				V			D														
6				V				D													
7				V					D												
8				V						D											
9				V							D										
10				V								D									
11				V									D								
12				V										D							
13					V	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	
14																					

Exercício 4

Dada a matriz **TELA** [14x20], escreva um algoritmo usando laço para preencher a matriz conforme a figura que segue:

TELA

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1																				
2																				
3																				
4																				
5																				
6																				
7																				
8																				
9																				
10																				
11																				
12	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
13																				
14																				

Introdução à Lógica de Programação

Exercício 5

Dada a matriz **TEL2** [8x11], escreva um algoritmo usando laço encadeado para preencher a matriz conforme a figura que segue:

TEL2

	1	2	3	4	5	6	7	8	9	10	11
1											
2											
3						X					
4					X	X	X				
5				X	X	X	X	X			
6		X	X	X	X	X	X	X	X		
7	X	X	X	X	X	X	X	X	X	X	
8											

Exercício 6

Dada a matriz **TEL3** [8x11], escreva um algoritmo usando laço encadeado para preencher a matriz conforme a figura que segue:

TEL3

	1	2	3	4	5	6	7	8	9	10	11
1											
2							8				
3							8	9	10		
4						8	9	10	11	12	
5					8	9	10	11	12	13	14
6			8	9	10	11	12	13	14	15	16
7	8	9	10	11	12	13	14	15	16	17	18
8											

Variáveis indexadas e Laços encadeados

Exercício 7

Preencha a matriz M [5x5] de acordo com o algoritmo a seguir:

PROGRAMA1

Declara L, C, M [1:5, 1:5] numéricas

Para L = 1 até 3 passo +1

Para C = 2 até 5 passo +1

M[L,C] = C

Próximo C

Próximo L

FIM

M

	1	2	3	4	5
1					
2					
3					
4					
5					

Exercício 8

Preencha a matriz MM [5x5] de acordo com o algoritmo a seguir:

PROGRAMA2

Declara L, C, MM [1:5, 1:5] numéricas

Para L = 1 até 5 passo +1

Para C = 6 - L até 5 passo + 1

MM [L , C] = "X"

Próximo C

Próximo L

FIM

MM

	1	2	3	4	5
1					
2					
3					
4					
5					

Introdução à Lógica de Programação

Exercício 9

Dada a matriz **RETANGULO** [14x11], escreva um algoritmo usando laço encadeado para preencher a matriz conforme a figura que segue:

RETANGULO

	1	2	3	4	5	6	7	8	9	10	11
1											
2											
3		R	R	R	R	R	R				
4		R	R	R	R	R	R				
5		R	R	R	R	R	R				
6		R	R	R	R	R	R				
7		R	R	R	R	R	R	R			
8		R	R	R	R	R	R	R			
9		R	R	R	R	R	R	R			
10		R	R	R	R	R	R	R			
11		R	R	R	R	R	R	R			
12											
13											
14											

Exercício 10

Dada a matriz **XADREZ [8x8]**, escreva um algoritmo usando laço encadeado para preencher a matriz conforme a figura que segue:

XADREZ

	1	2	3	4	5	6	7	8
1	X		X		X		X	
2		X		X		X		X
3	X		X		X		X	
4		X		X		X		X
5	X		X		X		X	
6		X		X		X		X
7	X		X		X		X	
8		X		X		X		X

Exercício 11

Crie um algoritmo e o fluxograma correspondente que conte quantas vezes um número digitado em um vetor é encontrado. Considere **Y [1:10]**.

Exercício 12

Crie um algoritmo e o fluxograma correspondente que verifica se, em um vetor, existem números duplicados. Considere **Y [1:10]** como vetor.

Banco de dados e tipos de programação

9

- ✓ Banco de dados;
- ✓ Modelo de dados;
- ✓ Objeto;
- ✓ Tipos de programação;
- ✓ Criação de tabelas;
- ✓ Relacionamento das tabelas;
- ✓ Consistência dos campos;
- ✓ Sistema de controle de cadastro.



IMPACTA
EDITORA

9.1. Introdução

O objetivo deste capítulo é apresentar o conceito de **banco de dados**, **objetos** e de **tipos de programação**.

9.2. Banco de dados

Um **banco de dados** é uma coleção de informações relacionadas a um determinado assunto ou finalidade, como um cadastro de fornecedores, um cadastro de produtos no estoque de uma empresa ou uma agenda.

As informações armazenadas em um banco de dados podem ser consultadas, comparadas, alteradas, impressas ou excluídas.

Ao criar um banco de dados, é fundamental que haja um planejamento voltado para o objetivo e forma de utilização desse banco, ou seja, é necessário considerar que tipos de informações ele deve conter.

A estrutura de dados é uma área na qual definimos quais são os dados que queremos armazenar:

- **Campo:** É composto por um caractere ou um conjunto de caracteres. Os campos correspondem às colunas da tabela;
- **Registro:** Um campo ou um conjunto de campos. Os registros correspondem às linhas da tabela;
- **Tabela:** É composta por um registro ou um conjunto de registros, e um campo ou um conjunto de campos;
- **Banco de dados:** É composto por uma tabela ou um conjunto de tabelas.

Exemplo:

Tabela: AGENDA

ID	NOME	ENDEREÇO	TELEFONE	UF
01	André	Rua Presidente, 07	99-99999-9199	SP
02	Rute	Al. das Árvore, 32	99-99999-9992	RJ
03	Solange	Rua da Praça, 454	99-99999-9939	SP
04	Sandra	Rua Alegre, 23	99-9999-4999	BA
05	Luis	Al. das Fontes, 602	99-9999-5999	SC

Coluna = Campo

Registro

Observando a tabela anterior, podemos notar que os tipos de dados podem ser diferentes para cada campo.

No planejamento de uma tabela, é importante que sejam analisadas as necessidades dos formulários, das consultas e dos relatórios do banco de dados.

Os bancos de dados que possuem tabelas relacionadas entre si são chamados de **bancos de dados relacionais**.

Uma tabela deve armazenar apenas informações sobre o mesmo assunto.

9.2.1. Considerações para tipos de dados

Para decidir a espécie de tipo de dados a ser utilizada para um campo, podemos tomar como base as seguintes considerações:

- A espécie de valores que desejamos armazenar no campo. Por exemplo, não é possível armazenar texto em um campo com um tipo de dados **DATA**;
- O espaço de armazenamento que desejamos utilizar para os valores neste campo;
- Os tipos de operações que desejamos efetuar com os valores do campo. Por exemplo, não é possível somar valores em campo com tipo de dados **TEXTO**, mas em campos do tipo **NÚMERO** é possível;
- A classificação dos valores de um campo. Os números são classificados como sequências de caracteres em um campo do tipo **TEXTO** (1, 10, 2, 20, 3, 30, e assim por diante) e não como valores numéricos. Para classificar números como valores numéricos utilizamos um campo do tipo **NÚMERO**.

9.2.2. Tipos de dados

A seguir, descrevemos os tipos de dados mais utilizados nas linguagens de programação:

- **Texto:** Texto ou combinações de textos e números, como endereços ou números que não exijam cálculos, como números de telefone ou códigos postais;
- **Número:** Dados numéricos a serem utilizados em cálculos matemáticos;
- **Moeda:** Valores monetários. Evita o arredondamento durante os cálculos;
- **Data/Hora:** Datas e horas;

- **Lógico (Booleano):** Campos que irão conter somente um entre dois valores, como Sim/Não, Verdadeiro/Falso ou Ativado/Desativado;
- **Objeto:** Objetos criados em outros programas (como documentos do Microsoft Word, planilhas do Microsoft Excel, figuras, sons ou outros dados binários).



Os nomes dos tipos de dados e as particularidades de cada um podem variar de linguagem para linguagem.

9.3. Modelo de dados

É o diagrama contendo as estruturas de dados e os relacionamentos.

9.3.1. Relacionamento

O **relacionamento** é um componente que define como duas tabelas se relacionam. As duas tabelas que se deseja ligar devem, obrigatoriamente, ter um campo em comum. Este campo recebe o nome de **chave**.

9.3.1.1. Chave primária

O campo **CHAVE PRIMÁRIA** determina de forma exclusiva cada registro armazenado. Não existem dois registros com o mesmo dado em um campo **CHAVE PRIMÁRIA** de uma mesma tabela.

9.3.1.2. Chave estrangeira

Chamamos de **CHAVE ESTRANGEIRA** o campo que possui um relacionamento com uma **CHAVE PRIMÁRIA** de outra tabela. Esse tipo de chave, que pode ocorrer repetidas vezes, estabelece um relacionamento entre a tabela em que ela está localizada e a tabela que contém a **CHAVE PRIMÁRIA**.

9.3.2. Modelo Entidade-Relacionamento

O diagrama de um sistema que contém todas as suas tabelas e seus relacionamentos recebe o nome de **Modelo Entidade-Relacionamento**. Esse diagrama deve ser desenvolvido enquanto estamos fazendo a **modelagem de dados**, tarefa esta que consiste em definir e estruturar os dados que serão manipulados e/ou gerados no sistema em questão. Esse modelo físico deve apresentar todos os detalhes das tabelas e seus relacionamentos. Nas tabelas, definimos os campos, seus tipos de dados e os índices.

9.3.3. Índice

Definimos um campo como índice para auxiliar na ordenação de dados e para agilizar processos de busca.

9.3.4. Regras de validação

Estas regras são usadas para garantir a consistência dos dados nos campos. Obrigatoriamente, os dados que serão digitados em um determinado campo devem obedecer às regras especificadas na consistência, ou seja, elas são uma expressão lógica para aceitação do dado.

9.3.5. Texto de validação

Trata-se da mensagem a ser exibida quando é quebrada a regra de validação.

9.4. Objeto

Um objeto pode existir no mundo real ou pode ser uma derivação de objetos do mundo real. Em programação, um **objeto** é um elemento computacional que representa alguma entidade abstrata ou concreta do domínio de interesse do problema em questão.

Cada objeto possui uma identidade e é distingível de outro objeto, independentemente de seus atributos serem idênticos.

O agrupamento de objetos com atributos e funções similares é chamado de **classe**. Um objeto, então, é considerado como uma instância de uma classe. Um **programa orientado a objetos** é composto por um conjunto de objetos que se interagem.

9.4.1. Elementos da interface de um objeto

São elementos da interface de um objeto:

- **Propriedades:** São as características de um objeto;
- **Evento:** Um objeto qualquer pode reconhecer alguma ação do usuário e, a partir dela, disparar um evento, ou seja, é uma propriedade que armazena o nome de uma rotina executando-a automaticamente quando ocorrer alguma ação com o objeto;
- **Métodos:** São ações executadas por um objeto.

Ou seja, um objeto tem propriedades, sofre eventos e reage com métodos.

9.5. Tipos de Programação

O advento das interfaces gráficas de usuário deu origem a uma nova forma de programação chamada **Programação Visual**.

O ponto de partida do processo de codificação é a montagem de uma tela de interface com o usuário. Montamos uma tela arrastando controles para uma janela. Esses controles são objetos, conforme definidos anteriormente.

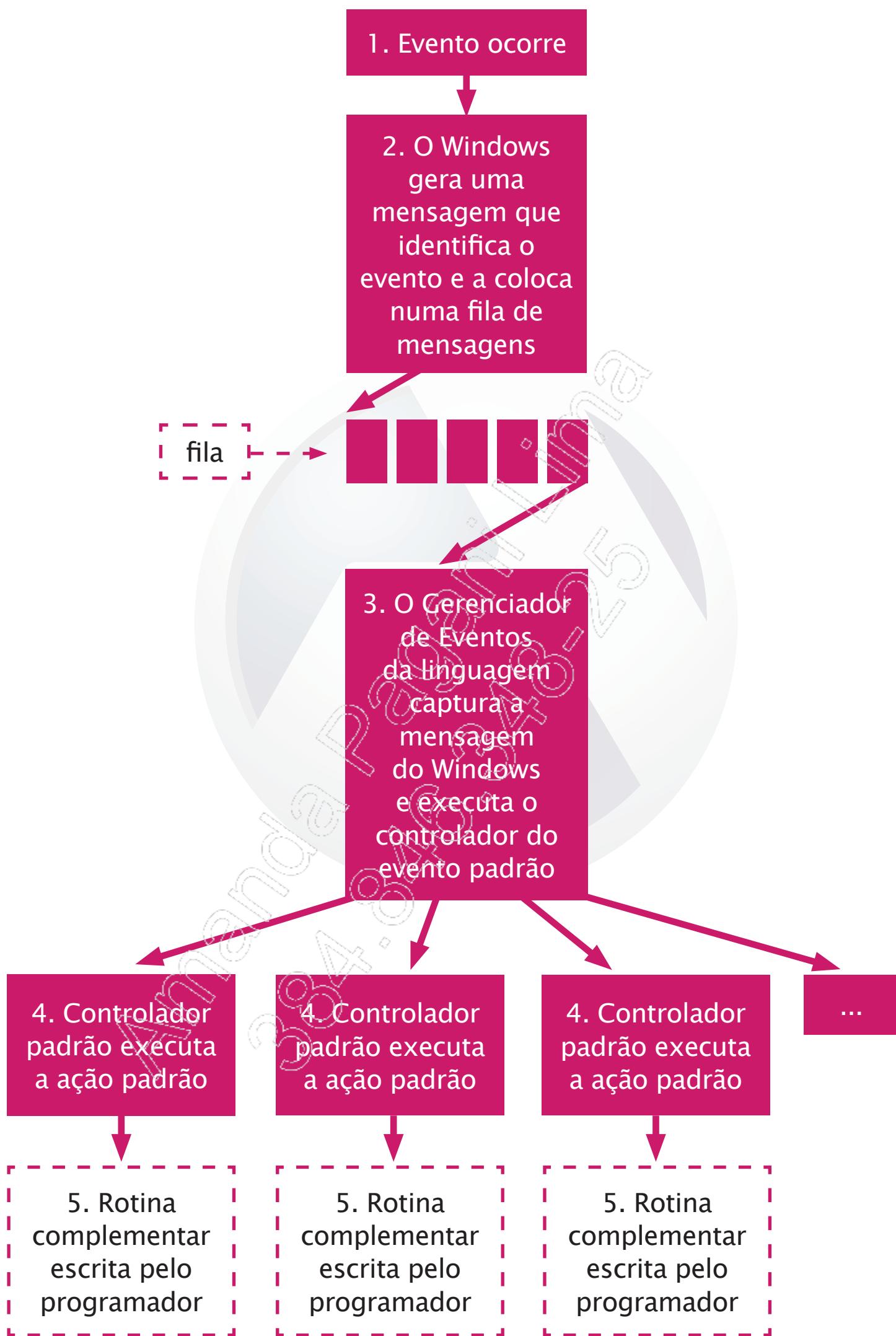
Em design-time (tempo de projeto), definimos as características (propriedades) iniciais desses objetos.

Em run-time (tempo de execução), além de alterarmos propriedades, podemos também invocar métodos.

Realizamos tais tarefas na medida em que o usuário interage com a tela e seus objetos, disparando eventos. Capturamos tais eventos para que nosso código possa ser executado, por isso chamamos este estilo de **programação orientada a eventos**, que é conhecida como **Programação Orientada a Objetos (POO)**.

Na programação tradicional (**programação procedural**), toda a lógica é descrita em sequência e executada nesta ordem. As ações do usuário são “engessadas”, já que a ordem não pode ser alterada por ele.

Na **programação orientada a eventos**, nossa lógica é “quebrada” em vários módulos (procedimentos), que são executados na ordem determinada pelas ações do usuário.



9.5.1. Ocorrências de eventos

Um **evento** consiste em um fato que possa influenciar na execução de um programa. Sem a ocorrência de um evento, o programa ficaria em situação de espera eternamente.

Existem três tipos básicos de eventos:

- **Evento de Mouse**

Ocorre quando executamos qualquer tipo de ação com o mouse, como mover o mouse sobre um objeto, pressionar um dos botões do mouse sobre um objeto, soltar um dos botões do mouse, clicar (pressionar e soltar) com o botão esquerdo do mouse sobre um objeto ou aplicar um duplo-clique com o botão esquerdo do mouse sobre um objeto.

- **Evento de teclado**

Ocorre quando executamos qualquer tipo de ação com o teclado, como quando pressionamos uma tecla quando o foco está sobre um objeto, quando soltamos uma tecla quando o foco está sobre um objeto ou quando pressionamos e soltamos uma tecla quando o foco está sobre um objeto.

- **Evento de clock**

É gerado a pedido do programador, em intervalos de tempo regulares, cuja unidade de medida é milissegundos.

9.5.2. Mensagens do Windows

Uma mensagem do Windows identifica o tipo de evento ocorrido e fornece parâmetros adicionais para o seu tratamento.

9.5.3. Gerenciador de eventos

É um conjunto de rotinas que captura a primeira mensagem da fila, detecta de que objeto ela partiu e desvia o processamento para a rotina de tratamento padrão.

9.5.4. Controlador de evento padrão

Não precisamos construir rotina alguma para que um formulário de uma aplicação de uma linguagem como Delphi, Visual Basic ou C# possa ser movido, redimensionado ou fechado.

Um botão, por exemplo, quando clicado, tem sua aparência modificada (é redesenhado). Quando uma caixa de entrada de texto tem foco e digitamos algo nela, os caracteres que digitamos são inseridos automaticamente na caixa de texto. Essas são ações padrão desses objetos. Não precisamos programá-las, pois já existe uma rotina (**Controlador de evento padrão**) que as executa.

9.5.5. Procedure complementar

Se quisermos implementar ações ao controlador padrão, devemos, então, criar nossa procedure (rotina) e ligá-la ao controlador padrão, que a executará em complemento à sua ação.

Exemplos:

- Quando clicar num botão, exibir uma mensagem;
- Quando digitar numa caixa de texto, emitir um “beep” para cada caractere digitado;
- Quando clicar numa barra de rolagem, aumentar ou diminuir o tamanho da fonte de uma etiqueta (label).

9.6. Criação de tabelas

A seguir, veremos um exemplo de uma **AGENDA**.

PLANEJAMENTO

Banco de Dados: **DBAGENDA**

Tabela **TABAGENDA**

Campo chave primária = **CODAGENDA**

Descrição	Campos	Tipo	Tamanho
Código	CODAGENDA	NÚMERO	5
Nome do contato	NOME	TEXTO	35
Telefone	TELEFONE	TEXTO	20
Endereço	ENDERECO	TEXTO	40
Código da cidade	CODCIDADEAGENDA	NÚMERO	4

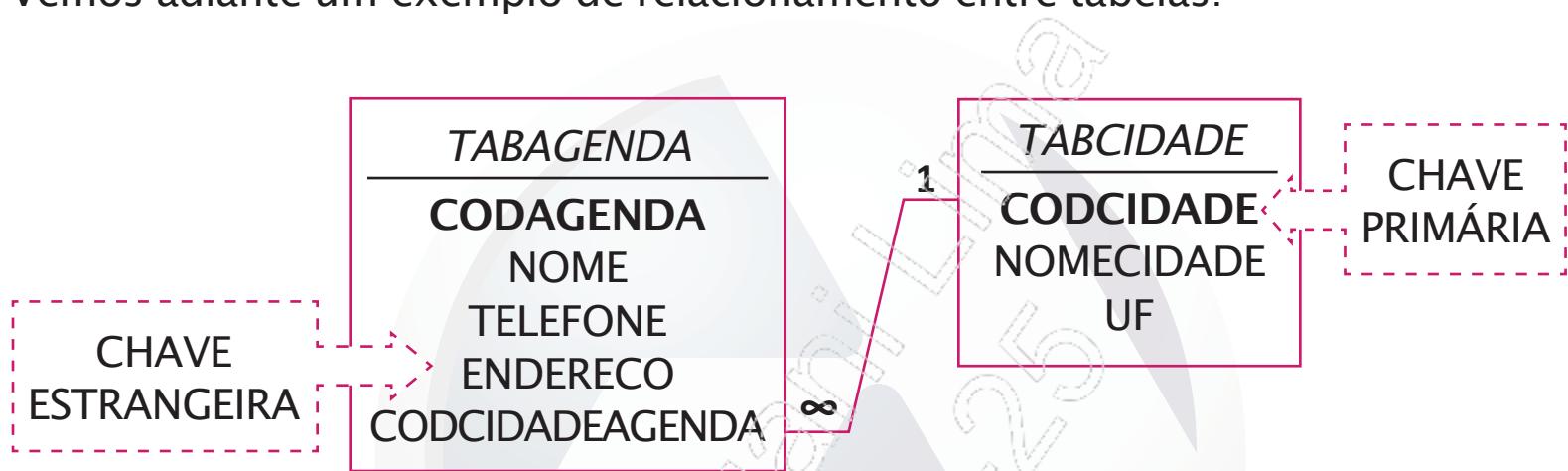
Tabela **TABCIDADE** (Tabela de Cidades)

Campo chave primária = **CODCIDADE**

Descrição	Campos	Tipo	Tamanho
Código da Cidade	CODCIDADE	NÚMERO	4
Nome da Cidade	NOME CIDADE	TEXTO	40
UF	UF	TEXTO	2

9.7. Relacionamento das tabelas

Vemos adiante um exemplo de relacionamento entre tabelas:



- Os campos denominados **CHAVE PRIMÁRIA** estão em negrito;
- Um campo **CHAVE PRIMÁRIA** não pode ter valores duplicados, por isso neste caso não podemos utilizar o campo **NOME**, pois pode haver mais de um registro com o mesmo nome;
- Os nomes dos campos relacionados, ou seja, da chave primária e da chave estrangeira, não precisam ser os mesmos, mas os dados nos campos precisam coincidir;
- Note que na **TABCIDADE** só pode haver um registro com o mesmo código, já na **TABAGENDA** pode haver muitos registros com o mesmo código de cidade. Relacionamento de um para muitos.

9.8. Consistência dos campos

A seguir, temos um exemplo de elaboração de regras de consistências, em que somente o preenchimento do telefone não é obrigatório na tabela da agenda.

Tabela TABAGENDA

Campo chave primária = CODAGENDA

Descrição	campo	Consistência
Código	CODAGENDA	É requerido que seja preenchido, o número tem que ser maior que zero, senão apresentar mensagem de código inválido. Não pode ter códigos duplicados, senão exibir mensagem de código existente.
Nome do contato	NOME	É requerido que seja preenchido, senão apresentar mensagem de que o preenchimento é obrigatório.
Telefone	TELEFONE	-
Endereço	ENDERECO	É requerido que seja preenchido, senão apresentar mensagem de que o preenchimento é obrigatório.
Código da cidade	CODCIDADEAGENDA	É requerido que seja preenchido, senão apresentar mensagem de que o preenchimento é obrigatório. Deve existir na tabela de cidades.

A seguir, temos a tabela relacionada:

Tabela **TABCIDADE** (Tabela de Cidades)
Campo chave primária = **CODCIDADE**

Descrição	campo	Consistência
Código da Cidade	CODCIDADE	É requerido que seja preenchido, o número tem que ser maior que zero, senão apresentar mensagem de código inválido. Não pode ter códigos duplicados, senão exibir mensagem de código existente.
Nome da Cidade	NOMECIDADE	É requerido que seja preenchido, senão apresentar mensagem de que o preenchimento é obrigatório.
UF	UF	É requerido que seja preenchido, senão apresentar mensagem de que o preenchimento é obrigatório.

No sistema a seguir, considere que as consistências são efetuadas nas janelas das propriedades dos campos, não sendo necessário expressá-las no algoritmo.

9.9. Sistema de controle de cadastro

Independente da linguagem a ser utilizada, a lógica principal para atingir o objetivo é a mesma. A seguir, apresentamos um exemplo de tela inicial para o sistema de controle do cadastro da AGENDA.



- **Objetivo**

Os botões da tela anterior tem a finalidade de chamar os programas de inclusão, consulta, alteração e exclusão dos registros da tabela da agenda, e também chamar o programa principal da tabela de registros de cidades.

- **Processo**

Banco de dados utilizado: **DBAGENDA**

Tabela: **TABAGENDA**

- Botão **INCLUIR**: Chama o programa de **INCLUSÃO** de registros;
- Botão **CONSULTAR**: Chama o programa de **CONSULTA** de registros;

- Botão **ALTERAR**: Chama o programa de ALTERAÇÃO de registros;
- Botão **EXCLUIR**: Chama o programa de EXCLUSÃO de registros;
- Botão **TABELA DE CIDADES**: Chama a tela principal da tabela de cidades;
- Botão **SAIR**: Fecha o banco de dados e a tela, saindo do sistema.

9.9.1. Programa de inclusão

A seguir apresentamos o planejamento para o programa de inclusão de registros.

- **Objetivo**

Incluir registros na tabela **TABAGENDA**.

Descrição	Campo	Tipo	Tamanho
Código	CODAGENDA	NÚMERO	5
Nome do contato	NOME	TEXTO	35
Telefone	TELEFONE	TEXTO	20
Endereço	ENDERECO	TEXTO	40
Código da cidade	CODCIDADEAGENDA	NÚMERO	4

- **Processo**

Para que os campos não sejam acessados diretamente, os dados a serem digitados nas telas serão armazenados em variáveis e depois gravados num registro da tabela.

Serão utilizadas as variáveis a seguir para receber os respectivos dados:

- **VARCODAGENDA = Código do registro**
- **VARNOME = Nome do contato**
- **VARTELEFONE= Telefone do contato**

Introdução à Lógica de Programação

- VARENDERECO = Endereço do contato
- VARCODCID = Código da cidade
- Algoritmo

1. INCLUSÃO
2. Declara VARCODAGENDA, VARCODCID numéricas e VARNAME, VARTELEFONE, VARENTERECO alfanuméricas
3. VARCODAGENDA = 0
4. VARNAME = “ ”
5. VARTELEFONE = “ ”
6. VARENTERECO = “ ”
7. VARCODCID = 0
8. Abrir tabelas TABAGENDA e TABCIDADE
9. Abrir tela de inclusão de registros na tabela da TABAGENDA
10. Ler VARCODAGENDA
11. Buscar na tabela TABAGENDA no campo CODAGENDA = VARCODAGENDA
12. Encontrou o código?
 - 12.1. Se sim: Exibir “Código já existente”
 - 12.2. Vá para o passo 23
 - 12.3. Se não: Próximo passo
13. Ler VARNAME, VARTELEFONE, VARENTERECO, VARCODCID
14. Buscar na tabela TABCIDADE no campo CODCIDADE = VARCODCID
15. Encontrou?
 - 15.1. Se sim: Próximo passo
 - 15.2. Se não: Exibir “Código inválido. Cidade não Existente.”
 - 15.3. Vá para o passo 23
16. *** Comentário: Fazer os campos receberem o conteúdo das variáveis e depois gravar o registro
17. CODAGENDA = VARCODAGENDA
18. NOME = VARNAME
19. TELEFONE = VARTELEFONE
20. ENDERECO = VARENTERECO
21. CODCIDADEAGENDA = VARCODCID
22. Gravar o registro na tabela TABAGENDA
23. Fechar tela de inclusão de registros da TABAGENDA
24. Fechar tabelas TABAGENDA e TABCIDADE
25. SAÍDA

9.9.2. Programa de consulta

A seguir, apresentamos o planejamento para o programa de consulta de registros.

- **Objetivo**

Consultar registros na tabela TABAGENDA.

Descrição	Campo	Tipo	Tamanho
Código	CODAGENDA	NÚMERO	5
Nome do contato	NOME	TEXTO	35
Telefone	TELEFONE	TEXTO	20
Endereço	ENDERECO	TEXTO	40
Código da cidade	CODCIDADEAGENDA	NÚMERO	4

- **Processo**

A busca será feita pelo nome e, caso exista mais de um contato com o nome procurado, exibir na sequência um registro do outro. Após encontrar o registro na tabela, copiar os dados dos campos para as variáveis. Os dados serão exibidos nas telas em variáveis. Exibir o nome da cidade em vez do código.

Serão utilizadas as variáveis a seguir para receber os respectivos dados:

- **VARCODAGENDA** = Código do registro
- **VARNOME** = Nome do contato
- **VARTELEFONE** = Telefone do contato
- **VARENDERECO** = Endereço do contato
- **VARCODOCID** = Código da cidade
- **VARNOMECIDADE** = Nome da cidade

Introdução à Lógica de Programação

- Algoritmo

1. CONSULTA
2. Declara VARCODAGENDA numérica e VARNAME, VARTELEFONE, VARENDERECO, VARNOMECEIDADE alfanuméricas
3. VARCODAGENDA = 0
4. VARNAME = “ ”
5. VARTELEFONE = “ ”
6. VARENDERECO = “ ”
7. VARNOMECEIDADE = “ ”
8. Abrir tabelas TABAGENDA e TABCIDADE
9. Abrir tela de consulta de registros na tabela da TABAGENDA
10. Ler VARNAME
11. Ordenar a tabela TABAGENDA pelo campo NOME
12. Buscar na tabela TABAGENDA no campo NOME = VARNAME
13. Encontrou o nome?
 - 13.1. Se sim: Próximo passo
 - 13.2. Se não: Exibir “Contato não Existente.”
 - 13.3. Vá para o passo 22
14. Buscar na tabela TABCIDADE no campo CODCIDADE = CODCIDADEAGENDA
15. VARCODAGENDA = CODAGENDA
16. VARTELEFONE = TELEFONE
17. VARENDERECO = ENDERECO
18. VARNOMECEIDADE = NOMECEIDADE
19. Exibir na tela as variáveis VARCODAGENDA, VARNAME, VARTELEFONE, VARENDERECO e VARNOMECEIDADE
20. *** Comentário: Após a visualização do registro, verificar se existe outro registro com o mesmo nome, senão fechar a tela.
21. O próximo registro tem o mesmo nome?
 - 21.1. Se sim: Exibir os dados do outro registro indo para o passo 13
 - 21.2. Se não: Próximo passo
22. Fechar tela de consulta de registros da TABAGENDA
23. Fechar tabelas TABAGENDA e TABCIDADE
24. SAÍDA

9.9.3. Programa de alteração

A seguir, apresentamos o planejamento para o programa de alteração de registros.

- **Objetivo**

Alterar registros na tabela TABAGENDA.

Descrição	Campo	Tipo	Tamanho
Código	CODAGENDA	NÚMERO	5
Nome do contato	NOME	TEXTO	35
Telefone	TELEFONE	TEXTO	20
Endereço	ENDERECO	TEXTO	40
Código da cidade	CODCIDADEAGENDA	NÚMERO	4

- **Processo**

A busca será feita pelo código do registro. Após encontrar o registro na tabela, copiar os dados dos campos para as variáveis. Os dados serão exibidos nas telas em variáveis. Exibir o código e o nome da cidade. Será permitida a alteração dos dados em algumas variáveis.

Serão utilizadas as variáveis a seguir para receber os respectivos dados:

- **VARCODAGENDA** = Código do registro
- **VARNOME** = Nome do contato
- **VARTELEFONE** = Telefone do contato
- **VARENDERECO** = Endereço do contato
- **VARCODOCID** = Código da cidade
- **VARNOMECIDADE** = Nome da cidade

Introdução à Lógica de Programação

- Algoritmo

1. ALTERAÇÃO
2. Declara VARCODAGENDA, VARCODOCID numéricas e VARNAME, VARTELEFONE, VARENDERECO, VARNOMECEIDADE alfanuméricas
3. VARCODAGENDA = 0
4. VARNAME = “ ”
5. VARTELEFONE = “ ”
6. VARENDERECO = “ ”
7. VARCODOCID = 0
8. VARNOMECEIDADE = “ ”
9. Abrir tabelas TABAGENDA e TABCIDADE
10. Abrir tela de alteração de registros da TABAGENDA
11. Ler VARCODAGENDA
12. Buscar na tabela TABAGENDA no campo CODAGENDA = VARCODAGENDA
13. Encontrou o código?
 - 13.1. Se sim: Próximo passo
 - 13.2. Se não: Exibir “Código não existente”
 - 13.3. Vá para o passo 30
14. Buscar na tabela TABCIDADE no campo CODCIDADE = CODCIDADEAGENDA
15. VARCODAGENDA = CODAGENDA
16. VARNAME = NOME
17. VARTELEFONE = TELEFONE
18. VARENDERECO = ENDERECO
19. VARCODOCID = CODCIDADEAGENDA
20. VARNOMECEIDADE = NOMECEIDADE
21. Exibir na tela as variáveis VARCODAGENDA, VARNAME, VARTELEFONE, VARENDERECO, VARCODOCID e VARNOMECEIDADE
22. Desproteger permitindo alteração as variáveis VARNAME, VARTELEFONE, VARENDERECO e VARCODOCID
23. *** Comentário: Fazer os campos receberem o conteúdo das variáveis e depois gravar o registro
24. CODAGENDA = VARCODAGENDA
25. NOME = VARNAME
26. TELEFONE = VARTELEFONE
27. ENDERECO = VARENDERECO
28. CODCIDADEAGENDA = VARCODOCID
29. Gravar o registro na tabela TABAGENDA
30. Fechar tela de alteração de registros da TABAGENDA
31. Fechar tabelas TABAGENDA e TABCIDADE
32. SAÍDA

9.9.4. Programa de exclusão

A seguir apresentamos o planejamento para o programa de exclusão de registros.

- **Objetivo**

Excluir registros na tabela **TABAGENDA**.

Descrição	Campo	Tipo	Tamanho
Código	CODAGENDA	NÚMERO	5
Nome do contato	NOME	TEXTO	35
Telefone	TELEFONE	TEXTO	20
Endereço	ENDERECO	TEXTO	40
Código da cidade	CODCIDADEAGENDA	NÚMERO	4

- **Processo**

A busca será feita pelo código do registro. Os dados serão exibidos nas telas em variáveis. Após encontrar o registro na tabela, copiar os dados dos campos para as variáveis. Os dados serão exibidos nas telas em variáveis. Exibir o código e o nome da cidade. Permitir a exclusão do registro exibido.

Serão utilizadas as variáveis a seguir para receber os respectivos dados:

- **VARCODAGENDA** = Código do registro
- **VARNOME** = Nome do contato
- **VARTELEFONE** = Telefone do contato
- **VARENDERECO** = Endereço do contato
- **VARCODOCID** = Código da cidade
- **VARNOMECIDADE** = Nome da cidade

Introdução à Lógica de Programação

- Algoritmo

1. EXCLUSÃO
2. Declara VARCODAGENDA, VARCODOCID numéricas e VARNAME, VARTELEFONE, VARENDERECO, VARNOMECIDADE alfanuméricas
3. VARCODAGENDA = 0
4. VARNAME = “ ”
5. VARTELEFONE = “ ”
6. VARENDERECO = “ ”
7. VARCODOCID = 0
8. VARNOMECIDADE = “ ”
9. Abrir tabelas TABAGENDA e TABCIDADE
10. Abrir tela de exclusão de registros da TABAGENDA
11. Ler VARCODAGENDA
12. Buscar na tabela TABAGENDA no campo CODAGENDA = VARCODAGENDA
13. Encontrou o código?
 - 13.1. Se sim: Próximo passo
 - 13.2. Se não: Exibir “Código não existente”
 - 13.3. Vá para o passo 24
14. Buscar na tabela TABCIDADE no campo CODCIDADE = CODCIDADEAGENDA
15. VARCODAGENDA = CODAGENDA
16. VARNAME = NOME
17. VARTELEFONE = TELEFONE
18. VARENDERECO = ENDERECO
19. VARCODOCID = CODCIDADEAGENDA
20. VARNOMECIDADE = NOMECIDADE
21. Exibir na tela as variáveis VARCODAGENDA, VARNAME, VARTELEFONE, VARENDERECO, VARCODOCID e VARNOMECIDADE
22. *** Comentário: Após exibir o registro perguntar se deseja excluí-lo e caso queira limpar o conteúdo dos campos do registro
23. Deseja excluir o registro?
 - 23.1. Se sim: Apagar o registro da tabela TABAGENDA
 - 23.2. CODAGENDA = 0
 - 23.3. NOME = “ ”
 - 23.4. TELEFONE = “ ”
 - 23.5. ENDERECO = “ ”
 - 23.6. CODCIDADEAGENDA = 0
- 23.7. Se não: Próximo passo
24. Fechar tela de exclusão de registros da TABAGENDA
25. Fechar tabelas TABAGENDA e TABCIDADE
26. SAÍDA

9.9.5. Considerações finais

A tela do MENU PRINCIPAL DA AGENDA possui o botão **TABELA DE CIDADES**, que dá acesso à tela do MENU PRINCIPAL DO CADASTRO DE CIDADES.

Este menu contém os botões de inclusão, consulta, alteração e exclusão de registros do cadastro de cidades. As funcionalidades dos botões são as mesmas dos botões do menu da agenda, porém se referem aos campos e registros da tabela **TABCIDADE**.

A tela do MENU PRINCIPAL DA AGENDA também possui o botão **SAIR**, que sai do sistema, fechando a tela principal e o banco de dados **DBAGENDA**.

O programa de **CONSULTA** exibe os dados na tela, o programa de **ALTERAÇÃO** exibe os dados na tela, permitindo alteração, e o programa de **EXCLUSÃO** exibe os dados na tela, permitindo a exclusão. Poderíamos aproveitar o mesmo programa e, de acordo com o botão selecionado, desviar o programa para a funcionalidade específica.

Nesses algoritmos, quando um código não é encontrado, a tela é fechada, retornando à tela do menu principal da agenda. Também pode ser programado para que fosse solicitado novamente um código.

A lógica dos passos para atingir um objetivo pode ser descrita de várias maneiras. Se escrevermos o programa em uma **linguagem procedural**, teremos que escrever os passos em um programa principal que chama os outros programas, mas tudo numa forma “engessada”, em que o usuário deve interagir com o sistema na ordem em que ele foi programado.

Se escrevermos o programa em uma linguagem de **programação orientada a objetos**, além de simplificar, no sentido de podermos programar “por partes”, ou seja, objeto por objeto, botão por botão, o usuário pode interagir mais livremente com o sistema.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Um **banco de dados** é uma coleção de informações relacionadas a um determinado assunto ou finalidade, como, por exemplo, uma agenda;
- A estrutura é uma área na qual definimos quais são os dados que queremos armazenar;
- São tipos de dados utilizados em linguagens de programação: Texto, Número, Moeda, Data/Hora, Lógico e Objeto;
- O **relacionamento** é um componente que define como duas tabelas se relacionam. As duas tabelas que se deseja ligar devem, obrigatoriamente, ter um campo em comum. Este campo recebe o nome de **chave**;
- O campo **CHAVE PRIMÁRIA** determina, de forma exclusiva, cada registro armazenado. Ou seja, não existem dois registros com o mesmo dado em um campo **CHAVE PRIMÁRIA** de uma mesma tabela;
- Chamamos de **CHAVE ESTRANGEIRA** aquela que possui um relacionamento com uma chave primária de outra tabela;
- O diagrama de um sistema que contém todas as suas tabelas e seus relacionamentos recebe o nome de **Modelo Entidade-Relacionamento**;
- Definimos um campo como índice para auxiliar na ordenação de dados e para agilizar processos de busca;
- As **regras de validação** são usadas para garantir a consistência dos dados nos campos. Obrigatoriamente, os dados que serão digitados em um determinado campo devem obedecer às regras especificadas na consistência, ou seja, elas são uma expressão lógica para aceitação do dado;
- **Texto de validação** é a mensagem a ser exibida quando uma regra de validação é quebrada;

- Em programação, um **objeto** é um elemento computacional que representa alguma entidade abstrata ou concreta do domínio de interesse do problema em questão;
- **Propriedades** são as características de um objeto. **Evento** é uma propriedade que armazena o nome de uma rotina, executando-a automaticamente quando ocorrer alguma ação com o objeto. **Métodos** são ações executadas por um objeto;
- Um **objeto** tem propriedades, sofre eventos e reage com métodos;
- Na **programação visual**, o ponto de partida do processo de codificação é a montagem de uma tela de interface com o usuário, composta por controles, os quais são objetos. Em design-time, definimos as características (propriedades) iniciais desses objetos. Em run-time, além de alterarmos propriedades, podemos também invocar métodos. Realizamos tais tarefas na medida em que o usuário interage com a tela e seus objetos, disparando eventos. Capturamos tais eventos para que nosso código possa ser executado, por isso chamamos este estilo de **programação orientada a eventos**, que é conhecida como **Programação Orientada a Objetos (POO)**;
- Um **evento** consiste em um fato que possa influenciar na execução de um programa. Sem a ocorrência de um evento, o programa ficaria em situação de espera eternamente. Existem 3 tipos básicos de eventos: evento de mouse, evento de teclado e evento de clock.

9

Banco de dados e tipos de programação

Mãos à obra!



IMPACTA
EDITORIA

Laboratório 1

Exercício 1

Escreva um algoritmo para um programa calcular a média dos valores de quatro variáveis numéricas.

Objetivo: Receber e calcular a média aritmética de quatro notas bimestrais e exibi-la na tela.

Processo: Serão utilizadas as variáveis a seguir para receber os respectivos dados:

- NB1 = Primeira nota bimestral
- NB2 = Segunda nota bimestral
- NB3 = Terceira nota bimestral
- NB4 = Quarta nota bimestral
- MSG = Mensagem
- MEDIA = Média das quatro notas

Calcular a média aritmética de acordo com a fórmula **MEDIA = (NB1 + NB2 + NB3 + NB4) / 4** e exibir o resultado com uma das respectivas mensagens:

- Se a média for menor que 5, exibir a mensagem "**Aluno reprovado**";
- Se a média for maior ou igual a 5 e menor que 7, exibir a mensagem "**Aluno em recuperação**";
- Se a média for maior ou igual a 7, exibir a mensagem "**Aluno aprovado**".

Após o resultado, perguntar se deseja fazer um novo cálculo. Em caso afirmativo, retornar ao início do programa, senão encerrar o programa.

Consistência das variáveis: Considere que as consistências são efetuadas nas propriedades dos objetos que representam as variáveis, não sendo necessário expressá-las no algoritmo.

VARIÁVEL	CONSISTÊNCIA
NB1	É requerido que seja preenchida, deve ser do tipo numérica, o número tem que ser maior ou igual a zero e menor ou igual a dez, senão apresentar a mensagem “A nota deve estar entre 0 e 10”.
NB2	É requerido que seja preenchida, deve ser do tipo numérica, o número tem que ser maior ou igual a zero e menor ou igual a dez, senão apresentar a mensagem “A nota deve estar entre 0 e 10”.
NB3	É requerido que seja preenchida, deve ser do tipo numérica, o número tem que ser maior ou igual a zero e menor ou igual a dez, senão apresentar a mensagem “A nota deve estar entre 0 e 10”.
NB4	É requerido que seja preenchida, deve ser do tipo numérica, o número tem que ser maior ou igual a zero e menor ou igual a dez, senão apresentar a mensagem “A nota deve estar entre 0 e 10”.
MEDIA	Deve ser do tipo numérica.
MSG	Deve ser do tipo alfanumérica.

Introdução à Lógica de Programação

Exercício 2

Escreva um algoritmo para um programa comparar os valores de duas variáveis numéricas.

Objetivo: Receber e comparar duas notas e exibir na tela a relação existente entre elas.

Processo: Serão utilizadas as variáveis a seguir para receber os respectivos dados:

- NOTA1 = Primeira nota
- NOTA2 = Segunda nota
- MSG = Mensagem

Comparar as duas notas e exibir o resultado com uma das respectivas mensagens:

- Se a 1^a nota for igual à 2^a nota, exibir “As duas notas são iguais”;
- Se a 1^a nota for maior que a 2^a nota, exibir “A 1^a nota é a maior”;
- Se a 2^a nota for maior que a 1^a nota, exibir “A 2^a nota é a maior”.

Após o resultado, perguntar se deseja fazer uma nova comparação. Em caso afirmativo, retornar ao início do programa, senão encerrar o programa.

Consistência das variáveis: Considere que as consistências são efetuadas nas propriedades dos objetos que representam as variáveis, não sendo necessário expressá-las no algoritmo.

VARIÁVEL	CONSISTÊNCIA
NOTA1	É requerido que seja preenchida, deve ser do tipo numérica, o número tem que ser maior ou igual a zero e menor ou igual a dez, senão apresentar a mensagem “A nota deve estar entre 0 e 10”.
NOTA2	É requerido que seja preenchida, deve ser do tipo numérica, o número tem que ser maior ou igual a zero e menor ou igual a dez, senão apresentar a mensagem “A nota deve estar entre 0 e 10”.
MSG	Deve ser do tipo alfanumérica.

Exercício 3

Escreva um algoritmo para um programa comparar os valores de três variáveis numéricas.

Objetivo: Receber e comparar três notas e exibir na tela a relação existente entre elas.

Processo: Serão utilizadas as variáveis a seguir para receber os respectivos dados:

- NOTA1 = Primeira nota
- NOTA2 = Segunda nota
- NOTA3 = Terceira nota
- MSG = Mensagem

Comparar as três notas e exibir o resultado com uma das respectivas mensagens:

- Se a 1^a nota for maior que as outras notas, exibir “A 1^a nota é a maior”;
- Se a 2^a nota for maior que as outras notas, exibir “A 2^a nota é a maior”;
- Se a 3^a nota for maior que as outras notas, exibir “A 3^a nota é a maior”;
- Se a 1^a e a 2^a nota forem iguais e maiores que a 3^a nota, exibir “A 1^a e a 2^a nota são iguais e maiores”;
- Se a 1^a e a 3^a nota forem iguais e maiores que a 2^a nota, exibir “A 1^a e a 3^a nota são iguais e maiores”;
- Se a 2^a e a 3^a nota forem iguais e maiores que a 1^a nota, exibir “A 2^a e a 3^a nota são iguais e maiores”;
- Se as três notas forem iguais, exibir “As três notas são iguais”.

Após o resultado, perguntar se deseja fazer uma nova comparação. Em caso afirmativo, retornar ao início do programa, senão encerrar o programa.

Introdução à Lógica de Programação

Consistência das variáveis: Considere que as consistências são efetuadas nas propriedades dos objetos que representam as variáveis, não sendo necessário expressá-las no algoritmo.

VARIÁVEL	CONSISTÊNCIA
NOTA1	É requerido que seja preenchida, deve ser do tipo numérica, o número tem que ser maior ou igual a zero e menor ou igual a dez, senão apresentar a mensagem “A nota deve estar entre 0 e 10”.
NOTA2	É requerido que seja preenchida, deve ser do tipo numérica, o número tem que ser maior ou igual a zero e menor ou igual a dez, senão apresentar a mensagem “A nota deve estar entre 0 e 10”.
NOTA3	É requerido que seja preenchida, deve ser do tipo numérica, o número tem que ser maior ou igual a zero e menor ou igual a dez, senão apresentar a mensagem “A nota deve estar entre 0 e 10”.
MSG	Deve ser do tipo alfanumérica.

Apêndice

Linguagens de programação e exemplos de programas

Amâncio Palagni Lima
384.40-25



IMPACTA
EDITORA

Introdução à Lógica de Programação

1.1. Introdução

Os conceitos de lógica e de programação abordados neste curso são aplicáveis a diversas linguagens de programação, como Java, SQL, JavaScript, C#, PHP e Visual Basic for Applications (VBA).

Neste apêndice apresentamos uma breve descrição dessas linguagens, seguida de exemplos de declaração de variáveis, uso de operadores e de estruturas de decisão e repetição em simples programas desenvolvidos com cada uma delas.

1.2. Java

Criada na década de 90 por profissionais da Sun Microsystems, a Java é uma das linguagens de programação orientada a objetos mais utilizadas no mundo, popularizando-se no desenvolvimento de aplicações para Web.

Pelo fato de utilizar boa parte da estrutura da linguagem C++ e conceitos de segurança da linguagem SmallTalk, a Java é tida como uma linguagem de fácil aprendizado, relativamente familiar aos programadores e bastante eficaz, que possibilita o desenvolvimento maciço de aplicações, applets e sistemas embutidos.

1.2.1. Exemplos

A seguir, temos um exemplo de declaração de variáveis em um programa em Java. Note que há linhas que se iniciam com barras duplas (//), e linhas entre /** e */. Utilizados para comentar as linhas de código, esses caracteres são inseridos pelo programador para facilitar o entendimento do código. Os comentários são ignorados pelo compilador ao executar o programa.

Linguagens de programação e exemplos de programas

```
/** Classe de exemplo de variáveis */
public class Variaveis {

    /** Método principal Main, o primeiro método à ser executado */
    public static void main(String[] args) {

        //Tipos de dados primitivos
        byte A = 127;
        System.out.println("O valor de A é:");
        System.out.println(A);

        short B = 32767;
        System.out.println("O valor de B é:");
        System.out.println(B);

        int C = 2147483647;
        System.out.println("O valor de C é:");
        System.out.println(C);

        long D = 9223372036854775807L;
        System.out.println("O valor de D é:");
        System.out.println(D);

        float E = 1.12345678F;
        System.out.println("O valor de E é:");
        System.out.println(E);

        double F = 1.123456789098765D;
        System.out.println("O valor de F é:");
        System.out.println(F);

        boolean G = true;
        System.out.println("O valor de G é:");
        System.out.println(G);

        char H = 'J';
        System.out.println("O valor de H é:");
        System.out.println(H);

    }

}
```

Introdução à Lógica de Programação

Já no código a seguir, temos a utilização dos comandos de fluxo em um programa desenvolvido em Java:

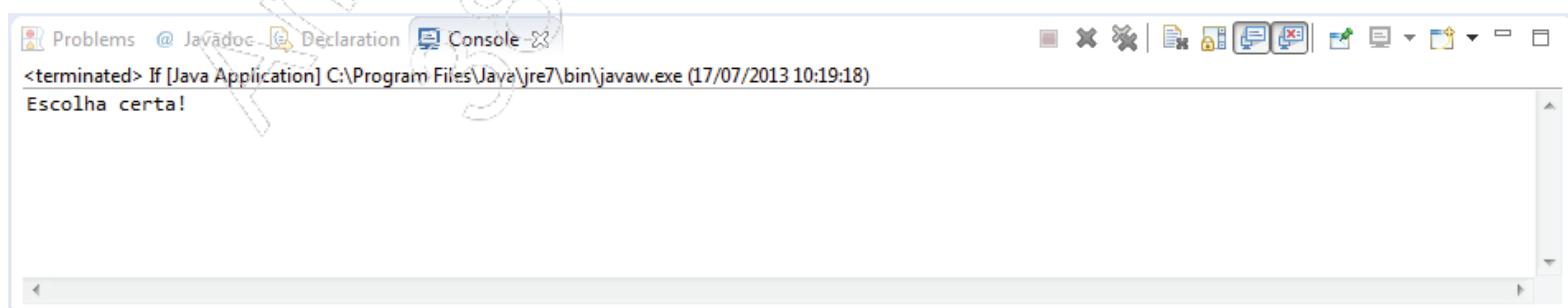
```
/** Classe de exemplo de IF */
public class If {

    /** Método principal Main, o primeiro método à ser executado */
    public static void main(String[] args) {

        //Declarando uma variável com o valor TRUE
        boolean alunoImpacta = true;

        //Usando a condição IF para validar o valor da variável
        if(alunoImpacta==true){
            //Resultado se a condição for verdadeira
            System.out.println("Escolha certa!");
        }else{
            //Resultado se a condição não for verdadeira
            System.out.println("Continue tentando!");
        }
    }
}
```

O resultado do código anterior é o seguinte:



Linguagens de programação e exemplos de programas

Este outro exemplo demonstra o uso de um laço de repetição **FOR** em um programa feito em Java:

```
/** Classe de exemplo de laço de repetição */
public class For {

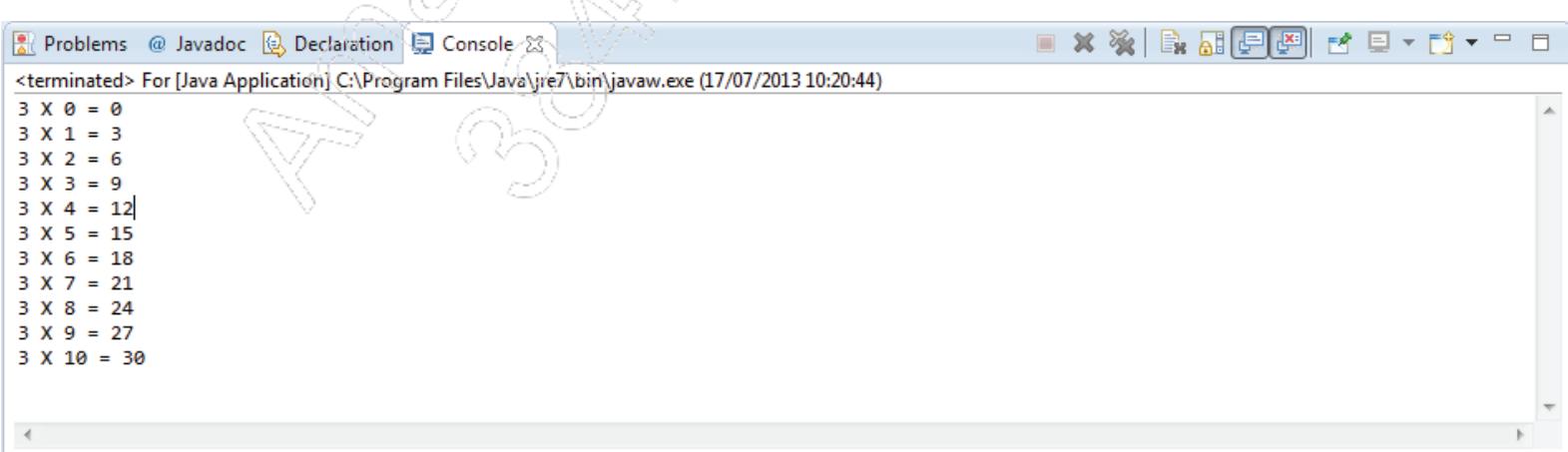
    /** Método principal Main, o primeiro método à ser executado */
    public static void main(String[] args) {

        //Declarando variável a com o valor 3
        int a = 3;

        //Iniciando o valor de i em 0 e repetindo o processo até 10
        for(int i=0; i<=10; i++) {

            //Imprimindo mensagem no console
            System.out.println(a+" X "+i+" = "+(a*i));
        }
    }
}
```

O resultado da execução do código que acabamos de ver é o seguinte:



```
Problems @ Javadoc Declaration Console
<terminated> For [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (17/07/2013 10:20:44)
3 X 0 = 0
3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
3 X 4 = 12
3 X 5 = 15
3 X 6 = 18
3 X 7 = 21
3 X 8 = 24
3 X 9 = 27
3 X 10 = 30
```

Introdução à Lógica de Programação

1.3. SQL

Desenvolvida no início da década de 70, a **SQL (Structure Query Language)** é uma linguagem de programação utilizada para manipulação de banco de dados. É caracterizada por sua exigência sintática rígida e por basear-se no conceito de banco de dados relacional. Na década de 80, a SQL foi adotada como linguagem padrão pela ANSI (American National Standard Institute) e ISO (International Organization for Standardization).

A Microsoft desenvolveu uma implementação para a linguagem SQL padrão ANSI denominada **T-SQL (Transact-SQL)**, que oferece opções adicionais para os comandos SQL, além de novos comandos que permitem o recurso de programação, como os de controle de fluxo, variáveis de memória, entre outros.

1.3.1. Exemplos

O código a seguir exemplifica, em linguagem de banco de dados SQL, a declaração de tipos de dados, a definição de valores para os tipos declarados e a exibição em tela desses tipos. Os trechos precedidos por hífen duplo (--) são comentários, e serão ignorados ao executar o script.

```
--Declarando um tipo de dado
DECLARE @id int --Tipo de dado inteiro
DECLARE @nome varchar(20) --Tipo de dado caractere com tamanho no
maximo de 20 caracteres
DECLARE @data datetime
DECLARE @dinheiro money

--Definindo o valor do dado
SET @id = 1
SET @nome = 'BRUNO'
SET @data = GETDATE() --Função GetDate() recupera a data e hora
atual.
SET @dinheiro = 11.35

--Mostrando o resultado na tela
PRINT @id
PRINT @Nome
PRINT @data
PRINT @dinheiro
```

Linguagens de programação e exemplos de programas

No exemplo a seguir, temos a criação de um banco de dados, a criação de uma tabela para o banco de dados e a inserção de dados nessa tabela. Na sequência, temos o trecho de código que realiza uma filtragem para exibição de dados do banco utilizando operadores lógicos estudados durante o curso:

```
--Criando um banco de dados
CREATE DATABASE DatabaseName

--Criando uma tabela
--A propriedade identity define que a coluna somente terá valores
unicos
--É definido o nome da coluna e depois o tipo na construção.
CREATE TABLE tb_tablename (id int identity, nome varchar(200))

--Inserindo dados em uma tabela
INSERT INTO tb_tablename(id,nome) VALUES(1,'Bruno')
INSERT INTO tb_tablename(id,nome) VALUES(1,'Tassia')

--Listando os dados
--O caractere * corresponde a todos as colunas da tabela que
serão listadas.
SELECT * FROM tb_tablename
SELECT id FROM tb_tablename --irá listar apenas a coluna ID

--Filtrando uma lista utilizando operadores lógicos. (=, <=. <,
>, >=, !=)
SELECT * FROM tb_tablename WHERE id = 1 --Irá listar o registro
com ID igual a 1
SELECT * FROM tb_tablename WHERE id != 1 --Irá listar o registro
com ID diferente que 1
SELECT * FROM tb_tablename WHERE id <= 1 --Irá listar o registro
com ID menor ou igual a 1
SELECT * FROM tb_tablename WHERE id < 1 --Irá listar o registro
com ID menor que 1
SELECT * FROM tb_tablename WHERE id >= 1 --Irá listar o registro
com ID maior ou igual a 1
SELECT * FROM tb_tablename WHERE id > 1 --Irá listar o registro
com ID maior que 1
```

Introdução à Lógica de Programação

Por fim, temos um código que demonstra o uso de uma estrutura condicional IF em um programa de banco dados com SQL:

```
--Utilizando estrutura de condição
DECLARE @n int
SET @n = 1
IF @n = 1
    BEGIN
        PRINT 'Condição Verdadeira'
    END
ELSE
    BEGIN
        PRINT 'Condição Falsa'
    END
```

1.4. JavaScript

Com possibilidade de execução em diversas plataformas, o **JavaScript** é uma linguagem de script orientada a objeto considerada leve, concisa e de fácil integração com outras aplicações. Vale dizer que JavaScript não possui relação qualquer com Java, sendo esta última uma linguagem mais complexa. Tem grande utilidade do lado do cliente, em um sistema cliente-servidor, como a Internet, em que é possível estender a linguagem básica por meio da disponibilização de objetos de controle para os navegadores (browsers). Do lado do servidor, um dos possíveis benefícios obtidos com a extensão básica da linguagem é a comunicação entre uma aplicação e um banco de dados.

Linguagens de programação e exemplos de programas

1.4.1. Exemplos

O exemplo de código de programa em JavaScript a seguir demonstra como é feita a declaração de variáveis nessa linguagem:

```
/* Os tipos de dados no javascript são:  
 undefined, boolean, number, string, function e object */  
  
// Toda declaração de variável no javascript deve ser precedida por 'var'  
var meuNumero = 10;  
  
// O tipo de dado no javascript é assumido de acordo com o valor inserido na variável  
var minhaString = "Olá mundo!";  
  
// Você pode declarar várias variáveis seguidas, mesmo de tipos diferentes;  
var minhaVerdade = true, meuFalso = false;  
  
// Se você não der um valor, a variável por padrão é do tipo e valor undefined  
var minhaVariavel, outraVariavel = undefined;
```

Neste outro exemplo de programa em JavaScript, temos o uso de **IF** e de loop com **FOR** e **WHILE**:

```
// O if no javascript pode ser feito com variáveis de tipo diferente  
if (false == "") {  
    console.log("Texto vazio é equivalente a falso!");  
} else if (false == 0) {  
    console.log("Número zero é equivalente a falso!");  
} else if (false == []) {  
    console.log("Um array vazio é equivalente a falso!");  
}  
  
// A ausência de um objeto é representada por null em javascript  
if (null == true || null == false) { // retorna falso  
    console.log("null não é equivalente nem a verdadeiro nem a falso.");  
} else if (undefined == true || undefined == false) { // retorna falso  
    console.log("undefined também não!");  
}  
  
for (var i = 0; i < 10; i++) {  
    // concatenação no javascript é feita com o sinal de mais  
    console.log("Girando: " + i);  
}  
  
// Soma também é exercida com o sinal de mais  
var dois = 1 + 1; // recebe dois  
// mas cuidado!!  
var resultado = "1" + 1; // aqui recebe "11" e não 2!  
  
// enquanto você não digitar "minhasenha" na janela, o while irá rodar  
var senha = "";  
while(senha != "minhasenha") {  
    senha = prompt("Digite a sua senha");  
}
```

1.5. C#

O C# é uma linguagem orientada a objeto utilizada para a criação de diversos tipos de programas, como programas cliente-servidor, programas tradicionais do Windows, programas de banco de dados, além de componentes distribuídos, XML Web Services, entre outros.

Em conjunto com o Visual Studio e outras ferramentas da plataforma .NET, mostra-se uma linguagem de grande utilidade para desenvolver, de maneira fácil e eficiente, vários tipos de programas.

1.5.1. Exemplos

O exemplo de código a seguir demonstra como é feita a declaração de variáveis em um programa desenvolvido com a linguagem C#.

Linguagens de programação e exemplos de programas

```
// Não se assuste! Essas bibliotecas básicas são criadas pelo
// Visual Studio, assim que você cria um novo projeto.
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ExemploDeLogical
{
    class Program
    {
        // TODO programa em C# é iniciado pela função Main()
        static void Main(string[] args)
        {
            // A declaração de uma variável no C# é feita definindo o tipo de variável e um nome:
            bool verdadeiro_ou_falso; // bool -> 1 bit

            // Você pode definir um valor para a variável no momento de sua criação
            byte AlgunsNumeros = 200; // byte -> 1 byte

            // C# é CASE SENSITIVE, ou seja, para ele maiúsculas e minúsculas são diferentes;
            short algunsNumeros; // short -> 2 bytes

            // Você pode declarar mais de uma variável por linha
            int primeiraVariavel, segundaVariavel; // int -> 4
bytes

            // Veja o exemplo da declaração de um array de 30
posições
            long[] meuLongo = new long[30]; // long -> 8 bytes
cada posição do array

            // Um caractere é uma frase
            char mander = 'A';
            string mensagem = "Olá mundo!";
        }
    }
}
```

Introdução à Lógica de Programação

Este outro exemplo mostra o uso de IF e loop em C#:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
// Adicionando a biblioteca de manipulação de arquivos para o
exemplo com o DO WHILE
using System.IO;

namespace ExemploDeLogica2
{
    class Program
    {
        static void Main(string[] args)
        {
            // Para você escrever um texto comum numa aplicação
            // básica:
            Console.WriteLine("Olá mundo");

            // Console.ReadLine permite que o usuário escreva um
            // valor para o programa:
            string valor_entregue_pelo_usuario = Console.ReadLine();

            // Comparações em C# devem ser feitas sempre com va-
            // riáveis do mesmo tipo, no caso abaixo, string e string
            if (valor_entregue_pelo_usuario == "Olá")
            {
                Console.WriteLine("Bem vindos à Impacta!");
            }

            // Quando o inteiro é declarado dentro do FOR, ele
            // se extingue quando o FOR é concluído:
            for (int i = 1; i <= 10; i++)
            {
```

Linguagens de programação e exemplos de programas

```
// A concatenação em C# é feita com o sinal + e
só é possível com strings, se você deseja
    // concatenar um valor numérico com uma string,
deve converte-lo antes para texto
    Console.WriteLine("Você está dentro de um loop!
Volta número " + i.ToString());
}

// Exemplo de DO WHILE, enquanto o nome do arquivo
já existir, ele tentará um novo,
// com um novo número: arquivo 1.txt, arquivo 2.txt,
arquivo 3.txt, arquivo 4.txt...
string meuArquivo; int j = 0;
do
{
    meuArquivo = "arquivo" + j.ToString() + ".txt";
    j++;
} while (File.Exists(meuArquivo));

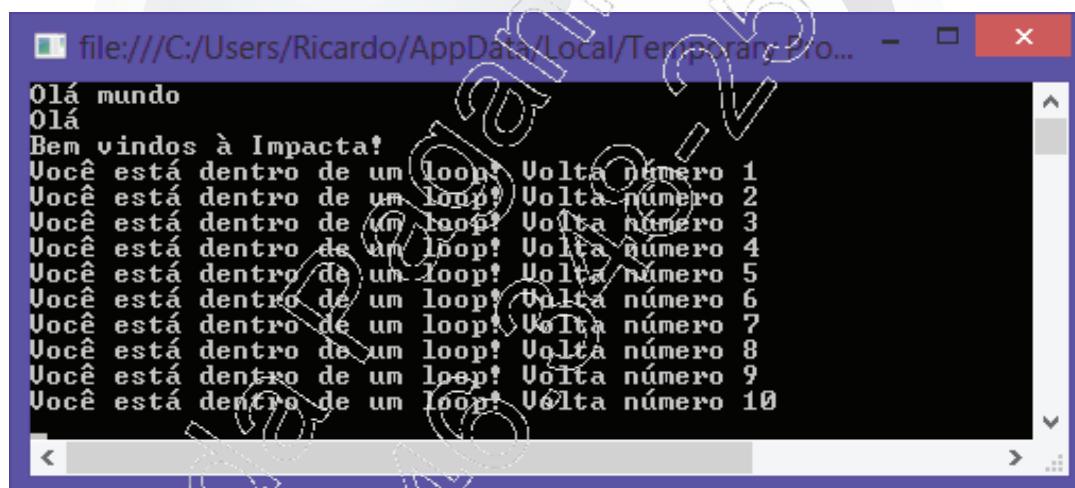
// As aplicações em Console do C# costumam ser con-
cluidas com um Console.ReadKey(),
// dessa forma o programa só irá encerrar se o usuá-
rio precionar alguma tecla,
// isso dará tempo à ele de ler o que foi escrito na
tela
    Console.ReadKey();
}
}
```

Introdução à Lógica de Programação

Temos, a seguir, o resultado da execução do código anterior. Ao executar o programa, você verá a seguinte tela, com a mensagem **Olá mundo**:



Se digitarmos **Olá** na tela anterior, a mensagem **Bem vindos à Impacta!** será exibida, e os comandos de loop são executados, conforme mostrado a seguir:



1.6. PHP

Destinado primordialmente ao desenvolvimento de scripts do lado do servidor, o **PHP (Hypertext Preprocessor)** é uma linguagem de programação open source (código aberto) muito popular na área de desenvolvimento Web, tanto para aplicações quanto para Websites. Uma das vantagens das páginas em PHP é possuir HTML com código embutido, diferentemente de linguagens como C e Perl, que exigem muitos comandos para a criação de HTML. Também é utilizado para desenvolvimento de scripts de linha de comando e aplicações GUI (interface gráfica do usuário) do lado cliente.

Linguagens de programação e exemplos de programas

No exemplo de código a seguir, podemos ver como é feita a declaração de variáveis no contexto da linguagem PHP:

```
<html>
  <head>
    <title>Conheça o PHP: Uma linguagem web</title>
  </head>
  <body>
    <p>O php trabalha de uma forma que faz com que pareça que<br />
      ele está dentro do html, mas na verdade é o html que está dentro dele!</p>
    <?php // O sinal <? ou <?php indica que iremos iniciar código em php

      // A declaração de uma variável no php é feita apenas escrevendo seu
      // nome com a opção de definir à ela um valor, de acordo com o valor
      $meubooliano = TRUE;

      // Toda variável no php é precedida pelo sinal $ e o segundo caractere
      // deve ser uma letra ou underline
      $meuinteiro = 13;

      // O tipo da variável no php é definido de acordo com o valor que é definido para ela
      $minhastring = "Olá mundo!";

      // as únicas divisões de números no php são int e float
      $meufloat = 10.01;

      // Se você declarar uma variável e não der um valor à ela, ela iniciará com o valor nulo
      $meunulo;
      $ou_meu_outro_nulo = null;

      // O php é case insensitive: ele não diferencia maiúsculas de minúsculas
      // Você pode alterar o tipo de uma variável no php se desejar
      $MeuInteiro = FalsE;

      // echo no php, permite que você escreva no documento
      echo $minhastring.", bem vindos à Impacta!"; // Php usa ponto para concatenar
    ?>
  </body>
</html>
```

Vejamos outro trecho de programa em PHP, no qual uma estrutura condicional é estabelecida:

```
<?php
/* A comparação de valores no php não
precisa ser feita com valores do mesmo tipo */
if (TRUE == 1) {
  echo "retorna verdadeiro!";
} else if ("Olá mundo!" == true) {
  echo "retorna verdadeiro também";
}

if (false == null) {
  echo "retorna verdadeiro também";
} else if ("" == false) {
  echo "Todos nós retornamos verdade!";
}
?>
```

1.7. Visual Basic for Applications (VBA)

Com características semelhantes à Visual Basic (VB), a **Visual Basic for Applications**, ou **VBA**, é uma linguagem de programação visual orientada a objetos implementada nos softwares que compõem o pacote Microsoft Office.

Sua utilização permite criar macros para aperfeiçoar tarefas que são realizadas frequentemente, fazendo com que várias ações sejam executadas em uma determinada sequência, por meio de um único atalho de teclado, por exemplo.

Vejamos, a seguir, exemplos de utilização da linguagem VBA nos softwares Excel e Access.

1.7.1. Excel

O Excel, um dos softwares de planilha eletrônica mais utilizados do mercado, permite o uso dos recursos da linguagem VBA. Vejamos um exemplo de um programa em VBA no Excel que calcula a média de quatro notas. Essas notas devem ser entre zero (0) e dez (10). Cada nota lida é verificada individualmente se está entre zero (0) e dez (10).

Linguagens de programação e exemplos de programas

```
Option Explicit
Sub CalculaMedia01()

    'Versão 1, com cada nota sendo lida e verificada individualmente
    'As linhas abaixo indicam que as variáveis serão do tipo dinâmicas,
    'isto é, serão apagadas da memória do aplicativo/computador,
    'e suas informações são do tipo STRING = Texto,
    'SINGLE = Um dos subtipos do tipo Numérico

    Dim Nota1 As String
    Dim Nota2 As String
    Dim Nota3 As String
    Dim Nota4 As String
    Dim Media As Single
    Dim Msg As String
    Dim Tit As String

    Tit = "Cálculo da Média"
    'A planilha PLANILHANOTAS é selecionada
    Sheets("PlanilhaNotas").Select
    'As variáveis são carregadas com o conteúdo das células indicadas
    Nota1 = Range("A2")
    Nota2 = Range("B2")
    Nota3 = Range("C2")
    Nota4 = Range("D2")

    If Not IsNumeric(Nota1) Then
        Msg = "A rotina aceita apenas números"
    ElseIf Nota1 < 0 Or Nota1 > 10 Then
        Msg = "A rotina aceita apenas números entre 0 e 10"
    End If
    'Se o conteúdo da variável MSG for diferente de vazio,
    'isto é, se ocorreram erros, então
```

Introdução à Lógica de Programação

```
If Msg <> "" Then
    'O conteúdo das variáveis MSG e TIT são apresentadas em tela
    'juntamente com o endereço da célula onde o erro foi encon-
trado
    MsgBox Msg, , Tit & " em A2"
    'A célula A2 é selecionada para que o operador possa corrigir
o erro
    Range("A2").Select
    'O processamento é terminado
    Exit Sub
End If

If Not IsNumeric(Nota2) Then
    Msg = "A rotina aceita apenas números"
ElseIf Nota2 < 0 Or Nota2 > 10 Then
    Msg = "A rotina aceita apenas números entre 0 e 10"
End If
If Msg <> "" Then
    MsgBox Msg, , Tit & " em B2"
    Range("B2").Select
    Exit Sub
End If

If Not IsNumeric(Nota3) Then
    Msg = "A rotina aceita apenas números"
ElseIf Nota3 < 0 Or Nota3 > 10 Then
    Msg = "A rotina aceita apenas números entre 0 e 10"
End If
If Msg <> "" Then
    MsgBox Msg, , Tit & " em C2"
    Range("C2").Select
    Exit Sub
End If
```

Linguagens de programação e exemplos de programas

```
If Not IsNumeric(Nota4) Then
    Msg = "A rotina aceita apenas números"
ElseIf Nota4 < 0 Or Nota4 > 10 Then
    Msg = "A rotina aceita apenas números entre 0 e 10"
End If
If Msg <> "" Then
    MsgBox Msg, , Tit & " em D2"
    Range("D2").Select
    Exit Sub
End If

Media = (CSng(Nota1) + CSng(Nota2) + CSng(Nota3) + CSng(Nota4)) /
4
Msg = "A média das notas é " & Media

MsgBox Msg, , Tit
End Sub
```

A seguir, temos outro exemplo de um programa em VBA no Excel, em que cada nota lida tem sua verificação efetuada através de um loop:

```
Sub CalculaMedia02()

    'Versão 2, com matriz e cada nota sendo lida e verificada dentro
    'de um loop

    Dim i As Byte
    Dim Msg As String
    Dim Tit As String
    Dim Media As Single
    Dim MediaFinal As Single
    Dim Nota(3) As String
    Tit = "Cálculo da Média "
```

Introdução à Lógica de Programação

'Conceitos

'Neste exemplo, se não houver erros de preenchimento das células, a rotina faz o cálculo da média.

'Ocorrendo um dos erros previstos, o operador é alertado sobre a natureza do erro, o local onde

'ele ocorreu é selecionado e a rotina é interrompida

```
For i = 0 To 3
    Nota(i) = Cells(2, i + 1)
    If Not IsNumeric(Nota(i)) Then
        Msg = "A rotina aceita apenas números"
    ElseIf Nota(i) < 0 Or Nota(i) > 10 Then
        Msg = "A rotina aceita apenas números entre 0 e 10"
    End If
    'Se a variável MSG tiver um conteúdo (uma das mensagens de
    erro) então
    If Msg <> "" Then
        'O texto composto pelo conteúdo das variáveis MSG, TIT e
        pelas frases indicadas abaixo
        'é apresentado ao operador
        MsgBox Msg & vbCrLf & "Faça a correção e execute novamente
        a rotina", , Tit & " Erro em Nota" & (i) + 1
        'A célula onde o erro foi identificado é selecionada
        Cells(2, i + 1).Select
        'O processamento é encerrado
        Exit Sub
    End If
    Next
    'Não foram encontrados erros, a natureza das variáveis é con-
    vertida, a média é calculada
    MediaFinal = (CSng(Nota(0)) + CSng(Nota(1)) + CSng(Nota(2)) +
    CSng(Nota(3))) / 4
    'A variável é carregada com a frase indicada e com o resultado
    do cálculo
    Msg = "A média das notas é " & MediaFinal
    'O conteúdo da variável é apresentado ao operador
    MsgBox Msg, , Tit
End Sub
```

1.7.2. Access

O Access, da Microsoft, é um software que permite construir, aplicar e distribuir banco de dados. É possível realizar muitas tarefas utilizando a interface do usuário, mas em muitos casos é necessária a programação. O Access também permite o uso dos recursos do VBA.

Vejamos um exemplo de um programa em VBA no Access. Ele calcula a média de quatro notas. Essas notas devem ser entre zero (0) e dez (10). Cada nota lida é verificada individualmente se está entre zero (0) e dez (10):

```
Sub CalculaMedia01()

    'Versão 1, com cada nota sendo lida e verificada individualmente
    'A linhas abaixo indicam que as variáveis serão do tipo dinâmi-
    'cas,
    'isto é, serão apagadas da memória do aplicativo/computador,
    'e suas informações são do tipo STRING = Texto,
    'SINGLE = Um dos subtipos do tipo Numérico

    'Declaração das variáveis
    Dim Nota1 As String
    Dim Nota2 As String
    Dim Nota3 As String
    Dim Nota4 As String
    Dim Media As Single
    Dim Msg As String
    Dim Tit As String

    'Atribuição de valores ou carregamento
    Tit = "Cálculo da Média"

    'Conceitos
    'A estrutura de decisão IF...ELSEIF...ELSE...ENDIF permite veri-
    'ficar os vários
    'tipos de erros possíveis. O processamento sairá do laço quando
    'não houver erros (ELSE).
    'O laço Do...Loop é necessário caso os erros de digitação sejam
    'repetitivos,
    'isto é, enquanto houver erros no preenchimento das variáveis o
    'processamento
    'não sai do laço
```

Introdução à Lógica de Programação

```
Do While True
    'A variável NOTA1 é preenchida por digitação do operador, na
    caixa de entrada
    Nota1 = InputBox("Digite a Nota1", Tit)
    'O conteúdo digitado é verificado em relação a sua natureza,
    'se não for de natureza numérico, então
    If Not IsNumeric(Nota1) Then
        'A variável MSG é carregada com o texto indicado
        Msg = "Digite apenas números"
        'O conteúdo digitado é verificado em relação ao seu valor,
        'se for menor do que zero ou maior do que 10, então
        ElseIf Nota1 < 0 Or Nota1 > 10 Then
            'A variável MSG é carregada com o texto indicado
            Msg = "Digite números entre 0 e 10"
            'O conteúdo digitado não é como indicado nas alternativas an-
            teriores
        Else
            'O processamento sai do laço e vai para a linha seguinte ao
        LOOP
            Exit Do
        End If
        'O conteúdo da variável MSG e da variável TIT são mostrados
        em tela
        MsgBox Msg, , Tit
    Loop

    Do While True
        Nota2 = InputBox("Digite a Nota2", Tit)
        If Not IsNumeric(Nota1) Then
            Msg = "Digite apenas números"
        ElseIf Nota2 < 0 Or Nota2 > 10 Then
            Msg = "Digite números entre 0 e 10"
        Else
            Exit Do
        End If
        MsgBox Msg, , Tit
    Loop
```

Linguagens de programação e exemplos de programas

```
Do While True
    Nota3 = InputBox("Digite a Nota3", Tit)
    If Not IsNumeric(Nota1) Then
        Msg = "Digite apenas números"
    ElseIf Nota3 < 0 Or Nota3 > 10 Then
        Msg = "Digite números entre 0 e 10"
    Else
        Exit Do
    End If
    MsgBox Msg, , Tit
Loop

Do While True
    Nota4 = InputBox("Digite a Nota4", Tit)
    If Not IsNumeric(Nota4) Then
        Msg = "Digite apenas números"
    ElseIf Nota4 < 0 Or Nota4 > 10 Then
        Msg = "Digite números entre 0 e 10"
    Else
        Exit Do
    End If
    MsgBox Msg, , Tit
Loop

'A média aritmética (variável MEDIA) é calculada após a conversão
'(CSNG = converter o conteúdo da variável para a natureza numérica, sub tipo single)
'do conteúdo da variável (Nota1, Nota2, Nota3, Nota4) para natureza número
Media = (CSng(Nota1) + CSng(Nota2) + CSng(Nota3) + CSng(Nota4)) /
4
'A variável MSG é carregada com a frase indicada
Msg = "A média das notas é " & Media
'O conteúdo da variável MSG e da variável TIT são mostrados em tela
MsgBox Msg, , Tit
End Sub
```

Introdução à Lógica de Programação

A seguir, temos outro exemplo de um programa em VBA no Access, em que cada nota lida tem sua verificação efetuada através de um loop:

```
Sub CalculaMedia02()

    'Versão 2, com matriz e cada nota sendo lida e verificada dentro
    'de um loop
    'A linha DIM NOTA(3) AS STRING indica a declaração de uma matriz
    'de uma dimensão
    'com 4 elementos numerados de zero a 3 e de natureza texto (como
    'o nome das variáveis)

    Dim i As Byte
    Dim Msg As String
    Dim Tit As String
    Dim Media As Single
    Dim Nota(3) As String
    Tit = "Cálculo da Média"
    'Laço do tipo FOR...NEXT.
    'Lê-se: Para i variando de 0 até 3, com incremento de 1
    'O processamento é automaticamente levado para fora do laço
    'quando a contagem de i for maior do que 3
    For i = 0 To 3 Step 1
        'Laço do tipo teste de condição, isto é, faça enquanto a con-
        'dição for verdadeira
        'Neste caso o processamento é levado para fora do laço pelo
        comando EXIT DO
        Do While True
            Nota(i) = InputBox("Digite a Nota " & i + 1, Tit)
            If Not IsNumeric(Nota(i)) Then
                Msg = "Digite apenas números"
            ElseIf Nota(i) < 0 Or Nota(i) > 10 Then
                Msg = "Digite números entre 0 e 10"
            Else
                Exit Do
            End If
            MsgBox Msg, , Tit
        Loop
        'Quando i for igual a 3, o laço FOR...NEXT o processamento au-
        'tomaticamente
        'passará para a linha de comando após a cláusula NEXT
    Next
    MediaFinal = (CSng(Nota(0)) + CSng(Nota(1)) + CSng(Nota(2)) +
    CSng(Nota(3)))/4
    Msg = "A média das notas é " & MediaFinal
    MsgBox Msg, , Tit
End Sub
```