# Introdução à Lógica de Programação (online)

Cód.: TE 1669/0_EAD

# Créditos



Copyright © TechnoEdition Editora Ltda.

Todos os direitos autorais reservados. Este material de estudo não pode ser copiado, reproduzido, traduzido, baixado ou convertido em qualquer outra forma eletrônica, digital ou impressa, 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 (online)

**Coordenação Geral**
Marcia M. Rosa

**Coordenação Editorial**
Henrique Thomaz Bruscagin

**Supervisão de Desenvolvimento Digital**
Alexandre Hideki Chicaoka

**Produção, Gravação, Edição de Vídeo e Finalização**
Cássia Morais Martins (Impacta Produtora)
Diego Luiz Henrique
Ivã Lucino Camargo
Luiz Guilherme Alves Brandão Jorge (Impacta Produtora)
Mayara Camargo Firmino (Impacta Produtora)
Xandros Luiz de Oliveira Almeida (Impacta Produtora)

**Roteiro, Edição e Revisão final**
Alexandre Hideki Chicaoka
Guilherme Yuji Kinoshita
Luiz Fernando Oliveira

**Locução**
Xandros Luiz de Oliveira Almeida (Impacta Produtora)

**Diagramação**
Bruno de Oliveira Santos
Carla Cristina de Souza
Diego Luiz Henrique

**Equipe de Apoio**
Carolina A. Messias

**Edição nº 1 | 1669/0_EAD**
abril/2014

*Este material é uma nova obra derivada da seguinte obra original, produzida por TechnoEdition Editora Ltda, em Jul/2013: Introdução à Lógica de Programação*
*Autoria: André Luiz de Vasconcelos*
*Nº de registro BN: 618781*

# Sumário




Apresentação ..... 06

Capítulo 1 - Introdução à Lógica ..... 07
1.1. Lógica ..... 08
1.2. Programa ..... 10
1.2.1. Tipos de linguagem de programação ..... 10
1.3. Tradutores ..... 10
1.3.1. Tipos de tradutores ..... 11
1.4. Tipos de Programação ..... 11
1.4.1. Ocorrências de eventos ..... 13
Teste seus conhecimentos ..... 15

Capítulo 2 - Algoritmo ..... 19
2.1. Algoritmo ..... 20
2.2. Elementos de um algoritmo ..... 22
2.2.1. Ação ..... 22
2.2.2. Decisão ..... 23
2.2.3. Laço ou Loop ..... 23
2.3. Algoritmo com o comando SE encadeado ..... 25
2.4. Algoritmo com o comando CASO ..... 26
2.5. Algoritmo com o comando ENQUANTO ..... 27
Exercício 1 ..... 32
Teste seus conhecimentos ..... 29
Mãos à obra! ..... 31

Capítulo 3 - Variáveis, Operadores e Funções ..... 33
3.1. Introdução ..... 34
3.2. Utilizando variáveis ..... 34
3.2.1. Consistência de condições ..... 34
3.2.2. Controle de repetições ..... 34
3.2.3. Comparações de variáveis de memória com campos de registros ..... 35
3.3. Tipos de variáveis ..... 35
3.4. Nomes de variáveis ..... 36
3.5. Declaração de variáveis ..... 37
3.6. Comando de atribuição ..... 37
3.7. Constantes ..... 38
3.8. Operadores aritméticos ..... 38
3.8.1. Contadores e acumuladores ..... 43
3.9. Operadores relacionais ..... 44
3.10. Operadores lógicos ..... 45
3.10.1. Tabela de decisão ..... 46
3.10.2. Tabela de decisão com números binários ..... 48
3.11. Função ..... 51
3.12. Concatenação de alfanuméricos ..... 52
Teste seus conhecimentos ..... 53
Mãos à obra! ..... 59

Capítulo 4 - Fluxograma ..... 61
4.1. Introdução ..... 62
4.2. Simbologia ..... 62
4.3. Criando fluxogramas ..... 63
4.3.1. Estruturas básicas ..... 64
4.4. Teste de Mesa ..... 70
Teste seus conhecimentos ..... 71
Mãos à obra! ..... 77

Capítulo 5 - Laço ou loop e repetição ..... 79
5.1. Introdução ..... 80
5.2. Comando FOR...NEXT (PARA...PRÓXIMO) ..... 80
5.3. Comando WHILE (ENQUANTO) ..... 83
5.4. Comando DO...WHILE (FAÇA....ENQUANTO) ..... 86
Teste seus conhecimentos ..... 87
Mãos à obra! ..... 95

Capítulo 6 - Variáveis indexadas e Laços encadeados ..... 97
6.1. Vetores e matrizes ..... 98
6.2. Laços encadeados ..... 105
Teste seus conhecimentos ..... 113
Mãos à obra! ..... 121

Capítulo 7 - Processamento predefinido ..... 125
7.1. Introdução ..... 126
7.2. Construindo um processamento predefinido ..... 127
Teste seus conhecimentos ..... 131
Mãos à obra! ..... 139

Capítulo 8 - Banco de dados ..... 141
8.1. Introdução ..... 142
8.2. Banco de dados ..... 142
8.2.1. Considerações para tipos de dados ..... 143
8.2.2. Tipos de dados ..... 144
8.3. Modelo de dados ..... 145
8.3.1. Relacionamento ..... 145
8.3.1.1. Chave primária ..... 145
8.3.1.2. Chave estrangeira ..... 145

8.3.2. Modelo Entidade-Relacionamento ..... 145
8.3.3. Índice ..... 146
8.3.4. Regras de validação ..... 146
8.3.5. Texto de validação ..... 146
8.4. Criação de tabelas ..... 146
8.5. Relacionamento das tabelas ..... 147
8.6. Consistência dos campos ..... 148
8.7. Sistema de controle de cadastro ..... 149
8.7.1. Programa de inclusão ..... 150
8.7.2. Programa de consulta ..... 152
8.7.3. Programa de alteração ..... 154
8.7.4. Programa de exclusão ..... 156
8.7.5. Considerações finais ..... 158
**Teste seus conhecimentos ..... 159**

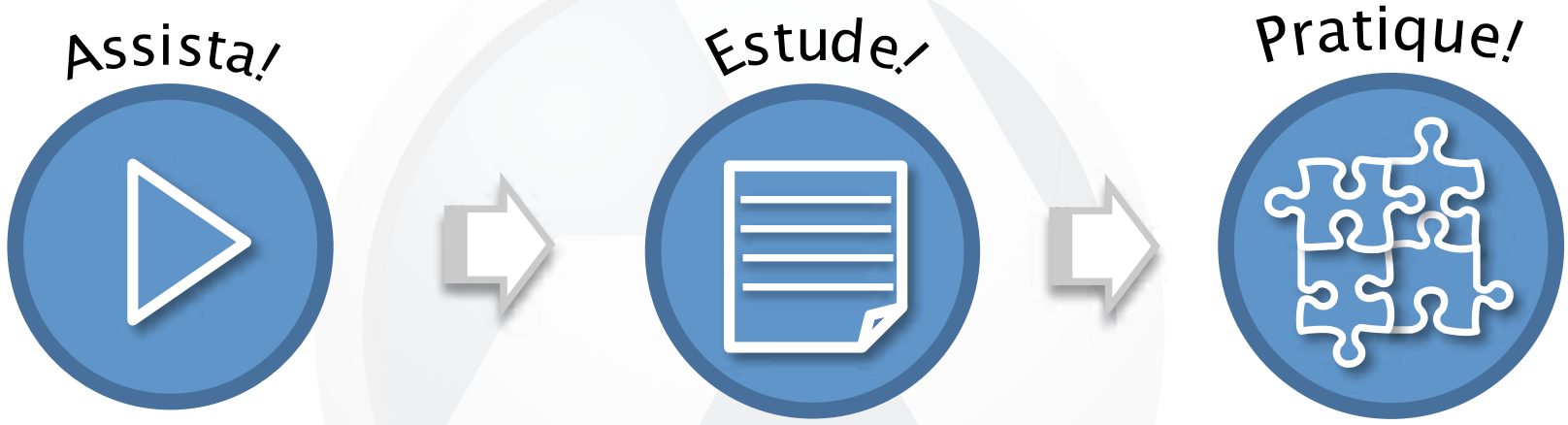
**Apêndice 1 - Linguagens de programação e exemplos de programas ..... 163**
1.1. Introdução ..... 164
1.2. Java ..... 164
1.2.1. Exemplos ..... 165
1.3. SQL ..... 168
1.3.1. Exemplos ..... 168
1.4. JavaScript ..... 170
1.4.1. Exemplos ..... 171
1.5. C# ..... 172
1.5.1. Exemplos ..... 172
1.6. PHP ..... 176
1.7. Visual Basic for Applications (VBA) ..... 177
1.7.1. Excel ..... 178
1.7.2. Access ..... 181

**Apêndice 2 - Sistemas de numeração ..... 185**
2.1. Bit e byte ..... 186
2.3. Sistemas de numeração ..... 187
2.3.1. Sistema Decimal ..... 188
2.3.2. Sistema Binário ..... 189
2.3.3. Sistema Hexadecimal ..... 189
2.4. Conversão de sistemas de numeração ..... 189
2.4.1. Conversão de Binário para Decimal ..... 189
2.4.2. Conversão de Hexadecimal para Decimal ..... 190
2.4.3. Conversão de Binário para Hexadecimal ..... 191
2.4.4. Conversão de Hexadecimal para Binário ..... 192
2.4.5. Conversão de Decimal para Binário ..... 193
2.4.6. Conversão de Decimal para Hexadecimal ..... 195
2.5. Forma rápida para conversão de sistemas de numeração ..... 196

## Bem-vindo!

É um prazer tê-lo como aluno do nosso curso online de **Introdução à Lógica de Programação**. Você vai dar os primeiros passos no universo da programação por meio de conceitos que vão ajudá-lo a analisar e resolver problemas lógicos. Algoritmos, variáveis, operadores, fluxogramas e laços de repetição são alguns dos assuntos que você vai aprender. Pronto para desenvolver seu raciocínio lógico? Para se dar bem neste curso, é importante que você esteja familiarizado com o ambiente Windows e realize as atividades propostas. Bom aprendizado!

## Como estudar?




Este curso conta com:

- Videoaulas sobre os assuntos que você precisa saber no módulo introdutório de Lógica de Programação.
- Parte teórica, com mais exemplos e detalhes para você que quer se aprofundar no assunto da videoaula.
- Exercícios de testes e laboratórios para você pôr à prova o que aprendeu.

# Introdução à Lógica

1

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

# 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 um exercício de lógica mais comum 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 três tipos básicos de tradutores: **Montador, Interpretador** 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.

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

## 1.4. Tipos de Programação

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 chamados de objetos.

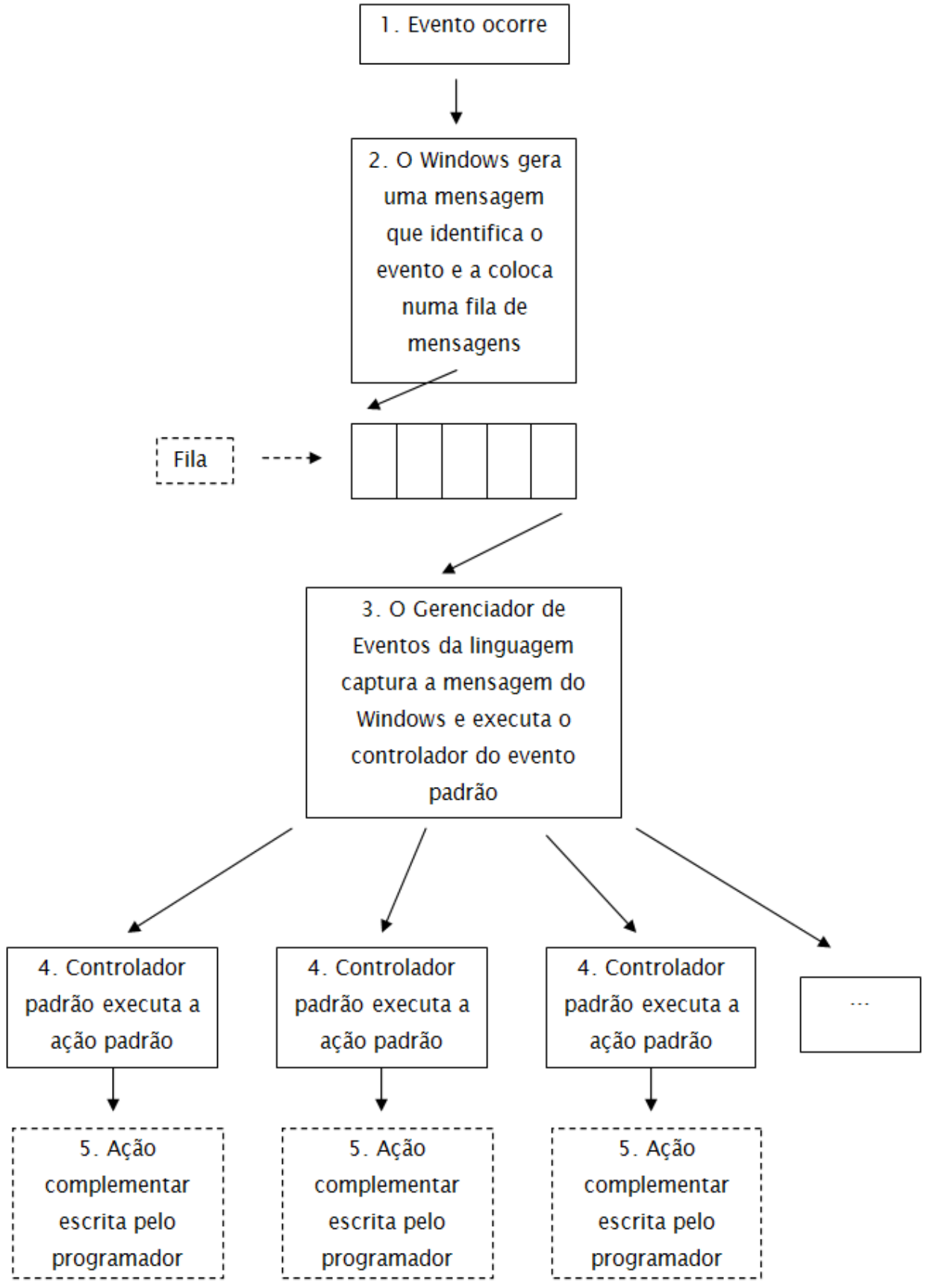
Em design-time (tempo de projeto), definimos as características (propriedades) iniciais desses objetos.

Em run-time (tempo de execução), além de alterarmos propriedades, podemos também chamar novas ações.

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**.

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.




A **programação orientada a objetos** se baseia na interação entre as unidades do software, chamadas de objetos. Nesse estilo de programação, são definidas diversas classes que determinam o comportamento e os estados de cada objeto.

## 1.4.1. 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 que depende dele para ser executado 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.

Teste seus conhecimentos

# Introdução à Lógica

1

**1. Uma tostadeira de pães suporta duas fatias de cada vez. Leva um minuto para torrar um lado de uma fatia. Qual o menor tempo gasto para torrar três fatias dos dois lados?**

- ☐ a) 3 minutos
- ☐ b) 4 minutos
- ☐ c) 5 minutos
- ☐ d) 6 minutos
- ☐ e) Nenhuma das alternativas anteriores está correta.

**2. Qual das alternativas a seguir é considerada uma linguagem de baixo  nível?**

- ☐ a) Java
- ☐ b) COBOL
- ☐ c) Assembly
- ☐ d) C#
- ☐ e) Lua

### 3. De que forma é feito o processo de tradução do código através do compilador?

- ☐ a) O verificador realiza a verificação do código, buscando por erros na sintaxe. O código então é passado para o compilador, que será responsável por gerar o módulo-objeto. Em seguida, este código ainda passa por um segundo processo de tradução, para então poder ser executado pela máquina.
- ☐ b) O código fonte é passado diretamente para o compilador, que é responsável pela tradução do código fonte na linguagem Assembly para o código de máquina. Em seguida, o programa é executado pelo computador.
- ☐ c) O verificador realiza a verificação do código, buscando por erros na sintaxe. O código então é passado para o compilador, que irá realizar a tradução para código de máquina e executar simultaneamente o programa.
- ☐ d) O código fonte é primeiramente compilado para então passar pelo verificador, que irá buscar erros na sintaxe. Em seguida, o programa é executado pelo computador.
- ☐ e) Nenhuma das alternativas anteriores está correta.

### 4. Qual das alternativas a seguir demonstra corretamente o processo geral de tradução do código fonte para linguagem de máquina?

- ☐ a) Processo computacional > Input > Output
- ☐ b) Input > Processo computacional > Output
- ☐ c) Output > Input > Processo computacional
- ☐ d) Input > Output > Processo computacional
- ☐ e) Processo computacional > Output > Input

**5. Qual das alternativas a seguir não é considerada um tipo de tradutor?**

- ☐ a) Montador
- ☐ b) Interpretador
- ☐ c) Compilador
- ☐ d) Assembly
- ☐ e) Todas as alternativas anteriores estão corretas.

# Algoritmo 2

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

# 2.1.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.

**Início**
**1. Acender o fogo**
**2. Colocar óleo na frigideira**
**3. O óleo está quente?**
**3.1. Se sim: “Próximo passo”**
**3.2. Se não: “Esperar o óleo esquentar” e “Vá para o passo 3”**
**4. Quebrar o ovo**
**5. O ovo está com uma boa consistência?**
**5.1. Se sim: “Próximo passo”**
**5.2. Se não: “Vá para o passo 10”**
**6. Colocar o ovo na frigideira**
**7. Colocar uma pitada de sal**
**8. O ovo está frito?**
**8.1. Se sim: “Próximo passo”**
**8.2. Se não: “Esperar fritar” e “Vá para o passo 8”**
**9. Pegar o ovo com a escumadeira e colocar em um prato**
**10. Desligar o fogo**
**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.

**Início**
**1. Abrir o porta-malas do carro**
**2. Tem estepe?**
**2.1. Se sim: Retirar o estepe**
**2.2. Se não: Vá para o passo 17**
**3. Tem ferramentas?**
**3.1. Se sim: Pegar as ferramentas**
**3.2. Se não: Vá para o passo 16**
**4. Desapertar os parafusos da roda**
**5. Já desapertou todos os parafusos?**
**5.1. Se sim: Vá para o passo 6**
**5.2. Se não: Vá para o passo 4**
**6. Levantar o carro com o macaco**
**7. Levantou o suficiente?**
**7.1. Se sim: Próximo passo**
**7.2. Se não: Vá para o passo 6**
**8. Retirar os parafusos**
**9. Retirar a roda**
**10. Colocar o estepe**
**11. Recolocar os parafusos**
**12. Abaixar o carro**
**13. Apertar os parafusos**
**14. Já apertou todos os parafusos?**
**14.1. Se sim: Vá para o passo 15**
**14.2. Se não: Vá para o passo 13**
**15. Guardar as ferramentas**
**16. Guardar o pneu**
**17. Fechar o porta-malas**
**Fim**

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.

# 2.2.Elementos de um algoritmo

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

## 2.2.1.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

## 2.2.2. Decisão

**Tem estepe?**
**Se sim: Retirar o estepe**
**Se não: Vá para o passo 17**

**Tem ferramentas?**
**Se sim: Pegar ferramentas**
**Se não: Vá para o passo 16**

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

## 2.2.3. Laço ou Loop

**4. Desapertar os parafusos da roda**
**5. Já desapertou todos os parafusos?**
**5.1. Se sim: Vá para o passo 6**
**5.2. Se não: Vá para o passo 4**

**Observação:** Note que, se não foram desapertados todos os parafusos, a execução do programa volta para a linha 4, fazendo um loop, que são trechos de uma lógica que podem ser executados várias vezes.

**6. Levantar o carro com o macaco**
**7. Levantou o suficiente?**
**7.1. Se sim: Próximo passo**
**7.2. Se não: Vá para o passo 6**

**Observação:** Note que, se o carro não foi suficientemente levantado, a execução do programa volta para a linha 6, fazendo um loop. Observe também que, em caso positivo, a execução do programa foi para o **Próximo passo**, que seria a mesma coisa que **Ir para o passo 8**, que é o próximo passo.

**13. Apertar os parafusos**
**14. Já apertou todos os parafusos?**
**14.1. Se sim: Vá para o passo 15**
**14.2. Se não: Vá para o passo 13**

**Observação:** Note que, se não foram apertados todos os parafusos, a execução do programa volta para a linha 13, fazendo um loop.

# 2.3.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**

**Início**
**1. Digite um número de 1 a 6**
**2. O número digitado é 1?**
**2.1. Se sim: Abrir o programa de Inclusão**
**2.2. Se não: O número digitado é 2?**
**2.2.1. Se sim: Abrir o programa de Exclusão**
**2.2.2. Se não: O número digitado é 3?**
**2.2.2.1. Se sim: Abrir o programa de Consulta**
**2.2.2.2. Se não: O número digitado é 4?**
**2.2.2.2.1. Se sim: Abrir o programa de Alteração**
**2.2.2.2.2. Se não: O número digitado é 5?**
**2.2.2.2.2.1. Se sim: Abrir o programa de Impressão**
**2.2.2.2.2.2. Se não: O número digitado é 6?**
**2.2.2.2.2.2.1. Se sim: Fechar Menu**
**2.2.2.2.2.2.2. Se não: Exibir "Número Inválido" e "Ir para o passo 1"**
**Fim**

# 2.4.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**

**Início**
**1. Digite um número de 1 a 6**
**2. Caso**
**2.1. Caso digitou 1: Abrir o programa de Inclusão**
**2.2. Caso digitou 2: Abrir o programa de Exclusão**
**2.3. Caso digitou 3: Abrir o programa de Consulta**
**2.4. Caso digitou 4: Abrir o programa de Alteração**
**2.5. Caso digitou 5: Abrir o programa de Impressão**
**2.6. Caso digitou 6: Fechar Menu**
**2.7. Caso contrário: Exibir “Número Inválido” e “Ir para o passo 1”**
**3. Fim do Caso**
**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.

# 2.5.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**
**1. Enquanto o contador de bolas azuis for menor que 03**
**1.1 Retirar uma bola**
**1.2 Caso**
**1.2.1 Caso a bola seja verde: Somar 1 no contador de bolas verdes**
**1.2.2 Caso a bola seja amarela: Somar 1 no contador de bolas amarelas**
**1.2.3 Caso contrário: Somar 1 no contador de bolas azuis**
**1.3 Fim do Caso**
**2. Fim do Enquanto**
**3. 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.

Teste seus conhecimentos

# Algoritimo

2

**1. Qual das situações a seguir melhor representa a utilização do comando CASO?**

- ☐ a) Uma decisão para escolher entre duas possíveis alternativas.
- ☐ b) Um menu de usuário para realizar ações específicas em um programa.
- ☐ c) Um conjunto de ações para realizar uma tarefa.
- ☐ d) Um laço para repetir determinada ação até que ocorra uma mudança de estado.
- ☐ e) Nenhuma das alternativas anteriores está correta.

**2. Qual comando a seguir representa um loop?**

- ☐ a) SE
- ☐ b) CASO
- ☐ c) ENQUANTO
- ☐ d) SE NÃO
- ☐ e) Nenhuma das alternativas anteriores está correta.

2

Mãos à obra!

# Algoritmo

# Exercício 1

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

# Variáveis, Operadores e Funções

3

- ✓ 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.

# 3.1.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.

# 3.2.Utilizando variáveis

A seguir, apresentamos as situações em que utilizamos variáveis.

## 3.2.1.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, desejamos escolher uma alternativa, podemos comparar a escolha com uma variável previamente definida e fazer o programa executar uma ou outra determinada sequência de comandos.

## 3.2.2.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.

### 3.2.3.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.

## 3.3.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.

# 3.4.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 | Tudo% |
| Nota_Final | Qualquer$valor |
| Valor_1 | Valor 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.

# 3.5.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
```

# 3.6.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.”
```

# 3.7.Constantes

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

# 3.8.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.

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

- **Exemplo 1**

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

- **Exemplo 2**

$$4 + 5^2 - (2\ X\ 3) \quad = \quad 4 + 5\text{^}2 - 2 * 3$$

- **Exemplo 3**

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

- **Exemplo 4**

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

- **Exemplo 5**

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

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

- **Exemplo 1**

  A = 2 * (5 + 3) / 2
  A = 2 * 8 / 2
  A = 16 / 2
  A = 8

- **Exemplo 2**

  B = 4 + 5 ^ 2 - 2 * 3
  B = 4 + 25 - 2 * 3
  B = 4 + 25 - 6
  B = 29 - 6
  B = 23

- **Exemplo 3**

  C = 25 // 2 - 125 // 3 + 5
  C = 5 - 125 // 3 + 5
  C = 5 - 5 + 5
  C = 5

- **Exemplo 4**

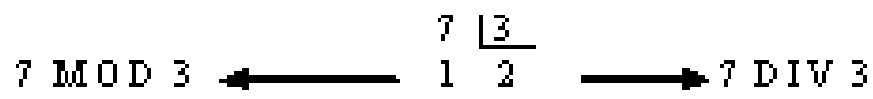
  D = 3 * 4 + 5 - 125 / 5 ** 3
  D = 3 * 4 + 5 - 125 / 125
  D = 12 + 5 - 125 / 125
  D = 12 + 5 - 1
  D = 17 - 1
  D = 16

- **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**




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**

## 3.8.1. 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**.

# 3.9.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:

```
IF Cargo = “Gerente”
    Salario = Salario * 1.10
ELSE
    Salario = Salario * 1.05
ENDIF
```

# 3.10.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: 1ª**

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

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

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: 3ª**

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

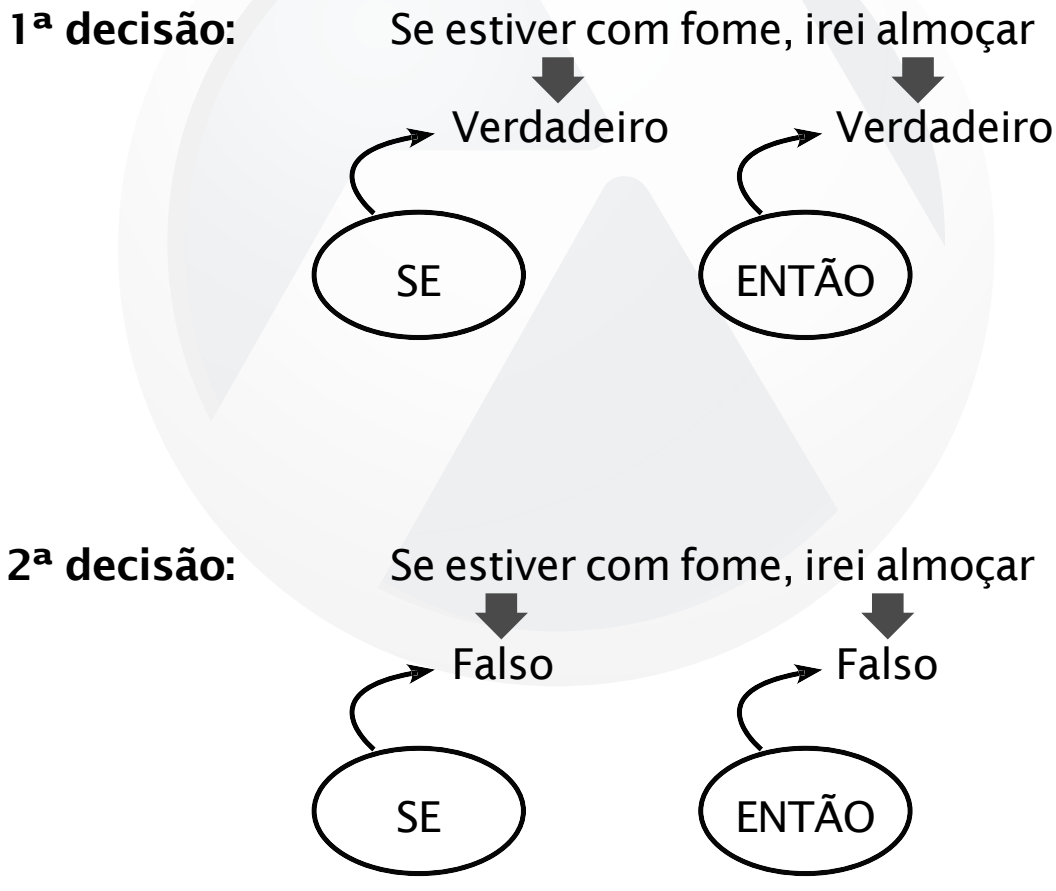
```
Se o cargo for igual a Gerente OU a Idade for maior ou igual a 50 anos
      Então conceder um aumento de 10% no salário
      Senão conceder um aumento de 5% no salário
Fim Se
```

## 3.10.1. Tabela de decisão

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

Exemplo:

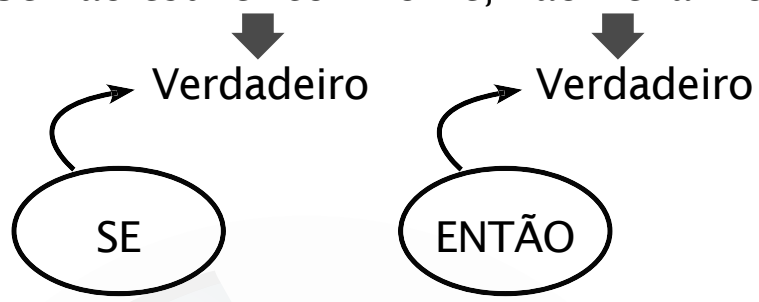
> **Se estiver com fome irei almoçar.**
> **Senão não irei almoçar.**



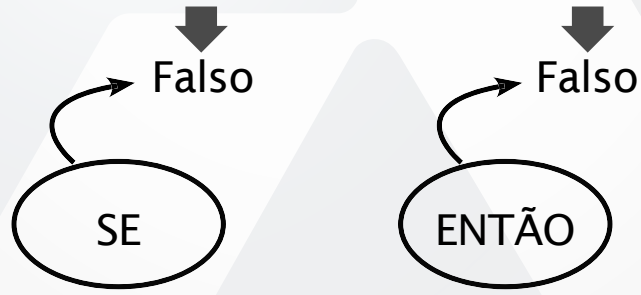
Observe que podemos escrever na negativa e o resultado será o mesmo.

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

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




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



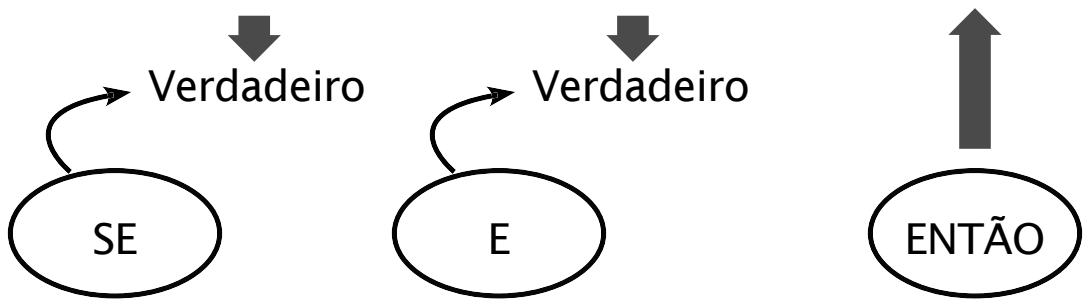

Analisamos, até agora, somente uma condição para que tomássemos uma ou outra decisão.

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:

Se estiver com fome e for meio-dia, irei almoçar



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 |

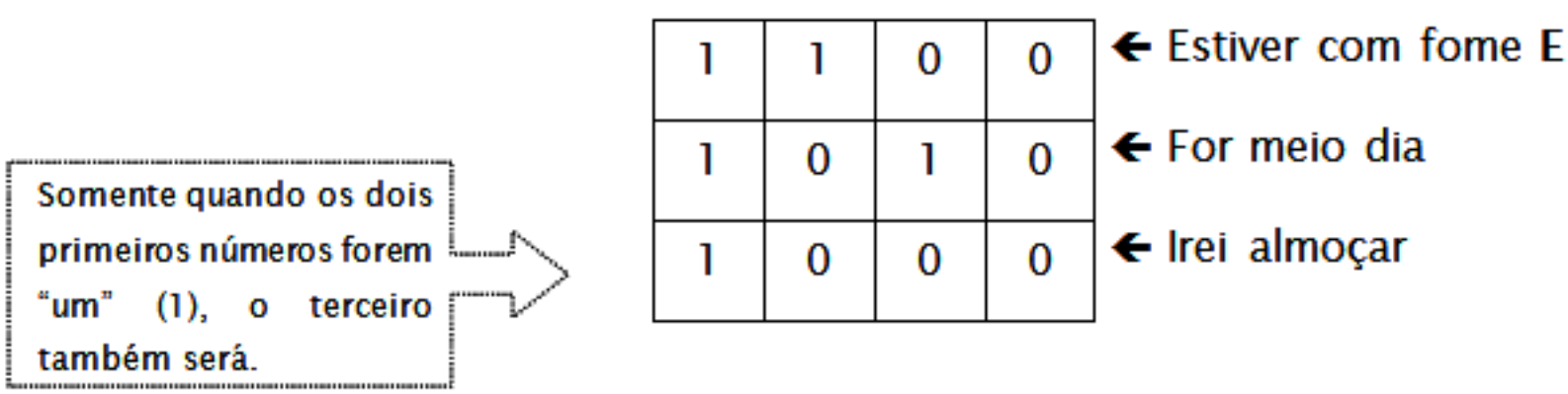
Usando o operador **OU**, basta que apenas uma das duas primeiras proposições seja verdadeira para que a terceira também seja.

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

## 3.10.2. 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.



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 ← Estiver com fome **OU** |
| 1 | 0 | 1 | 0 ← For meio dia |
| 1 | 1 | 1 | 0 ← 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 ← A |
| 1 | 1 | 1 | 1 ← B |
| 0 | 1 | 0 | 1 ← A E B |
| 1 | 0 | 1 | 0 ← **NÃO** (A E B) |

- **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 **E** A < C * 6
VERDADEIRO **E** VERDADEIRO
Resposta: VERDADEIRO

g) **NÃO** X **E** Y
**NÃO** FALSO **E** VERDADEIRO
VERDADEIRO **E** VERDADEIRO
Resposta: VERDADEIRO

h) **NÃO** Y **OU** X **E** C < A / B
**NÃO** VERDADEIRO **OU** FALSO **E** FALSO
FALSO **OU** FALSO **E** FALSO
FALSO **OU** FALSO
Resposta: FALSO

i) Y **OU** X **E** C < A / B
VERDADEIRO **OU** FALSO **E** FALSO
VERDADEIRO **OU** FALSO
Resposta: VERDADEIRO

j) ( X **OU** Y ) **E** C < A / B
(FALSO **OU** VERDADEIRO) **E** FALSO
VERDADEIRO **E** FALSO
Resposta: FALSO

# 3.11.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) ➔ B = “10”**
**C = STR(A * 5) ➔ C = “50”**
**D = STR(A - 6) ➔ D = “4”**
**E = STR(2000) ➔ E = “2000”**

**TOT = ”123”**
**N1 = VAL ( TOT ) ➔ N1 = 123**

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

# 3.12.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)
```

Teste seus conhecimentos

# Variáveis, Operadores e Funções

3

**1. Dados os seguintes valores para as variáveis:**

**A = 5**
**B = 7**
**C = 11**
**D = 8**
**E = 6**

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

- ☐ a) 30
- ☐ b) 35
- ☐ c) 40
- ☐ d) 45
- ☐ e) Nenhuma das alternativas anteriores está correta.

**2. Dados os seguintes valores para as variáveis:**

**A = 3**
**B = 5**
**C = 9**
**D = 1**
**E = 4**

**O valor de V2 na equação V2 = 30 / 15 + E ** D + 3 * C – D é:**

- ☐ a) 63
- ☐ b) 29
- ☐ c) 44
- ☐ d) 32
- ☐ e) 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:**

- ☐ a) 1
- ☐ b) 2
- ☐ c) 3
- ☐ d) 4
- ☐ e) 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 OU C > B) E 2 * C < B OU NÃO X**

**e**

**C > A 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.

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

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

3

Mãos à obra!

# Variáveis, Operadores e Funções

# Exercício 1

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

$$Y = A + 3 \; x \; B^2$$

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

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

# Exercício 2

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

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

# Fluxograma

4

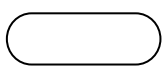
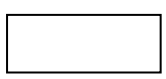
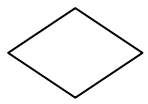
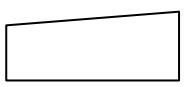
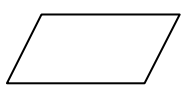
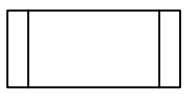
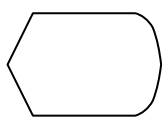
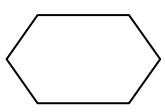
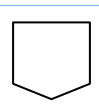
- ✓ Simbologia;
- ✓ Criação de fluxogramas;
- ✓ Estruturas básicas de fluxogramas;
- ✓ Teste de mesa.

# 4.1.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.

# 4.2.Simbologia

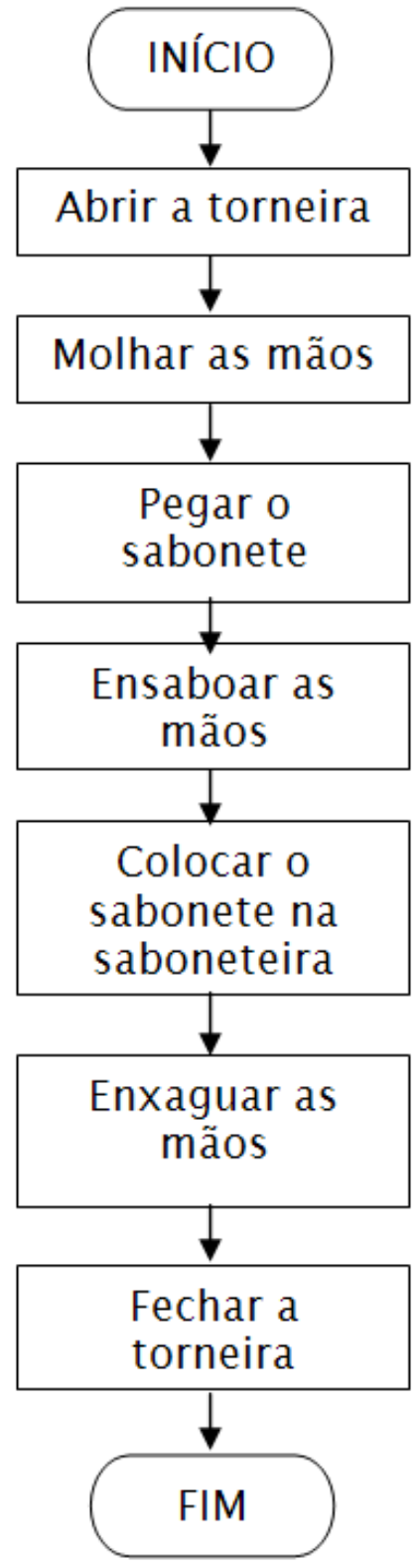
O fluxograma utiliza símbolos que permitem a descrição da sequência dos passos a serem executados de forma clara e objetiva. A tabela a seguir descreve a simbologia utilizada em fluxogramas:

| 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. |
|  | **Entrada / Saída** | Utilizado para indicar operações de leitura e gravação de registros. |
|  | **Processamento Predefinido ou Módulo** | Utilizado para indicar uma chamada a uma sub-rotina. |
| | **Documento** | Utilizado para indicar a impressão de dados. |
|  | **Exibir** | Utilizado para exibir dados na tela. |
|  | **Preparação** | Utilizado para a execução de looping. |
| | **Conector** | Utilizado para continuar o fluxograma em outra parte na mesma página. |
|  | **Conector de Página** | Utilizado para continuar o fluxograma em outra página. |
|  | **Seta de Fluxo de Dados** | Utilizado para indicar o sentido do fluxo de dados conectando os símbolos existentes. |

# 4.3.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




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.

## 4.3.1. Estruturas básicas

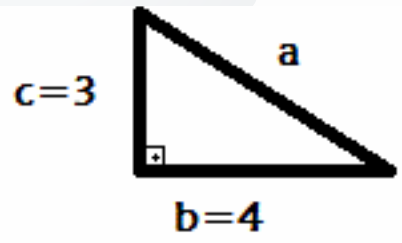
Vejamos, a seguir, as estruturas básicas de um fluxograma.

- **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 90º. 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.

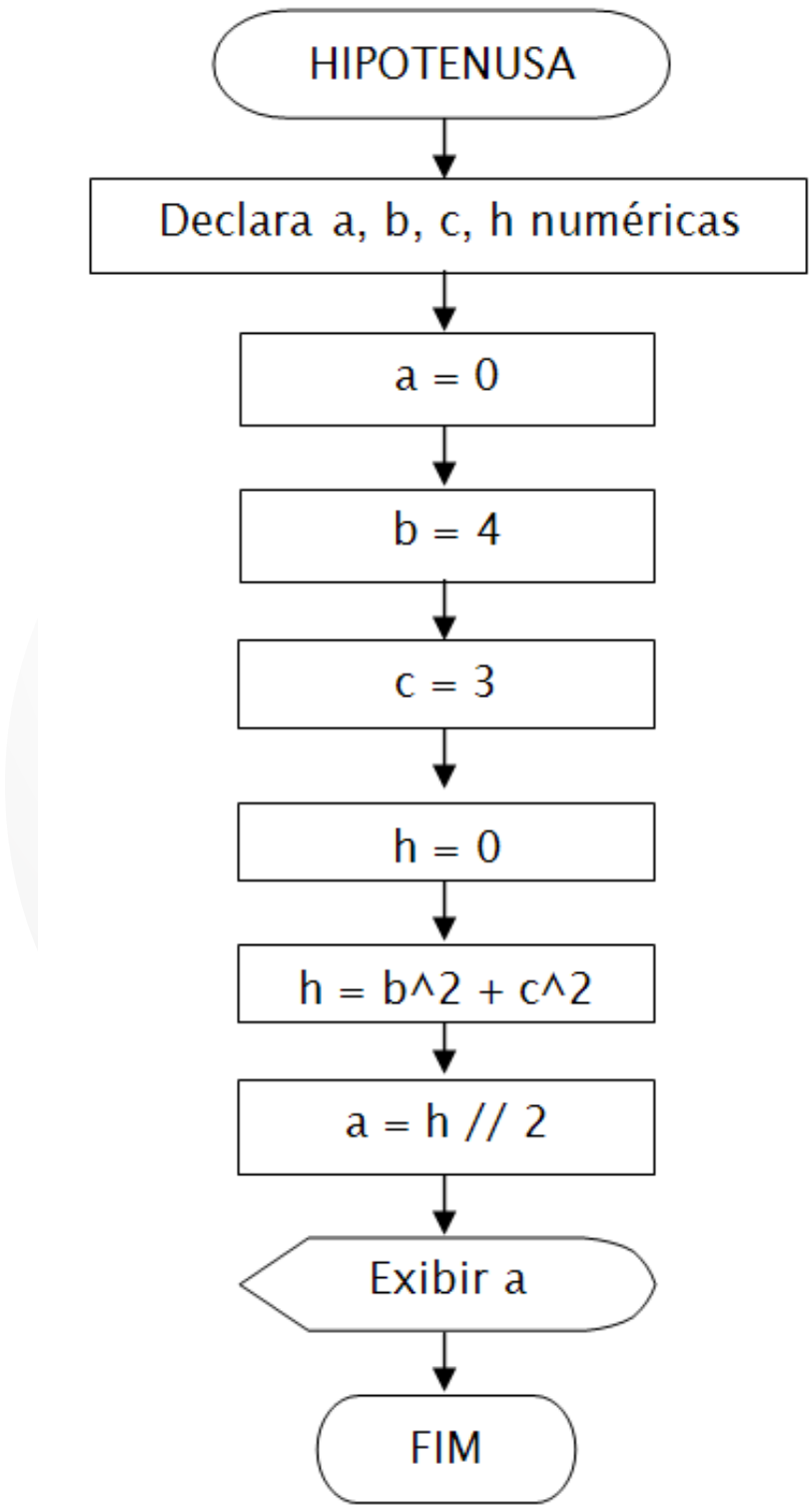
No triângulo a seguir, a hipotenusa está representada pela variável **a**, um cateto pela variável **b** que tem o valor 4 e o outro cateto pela variável **c**, que tem o valor 3.




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

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

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:

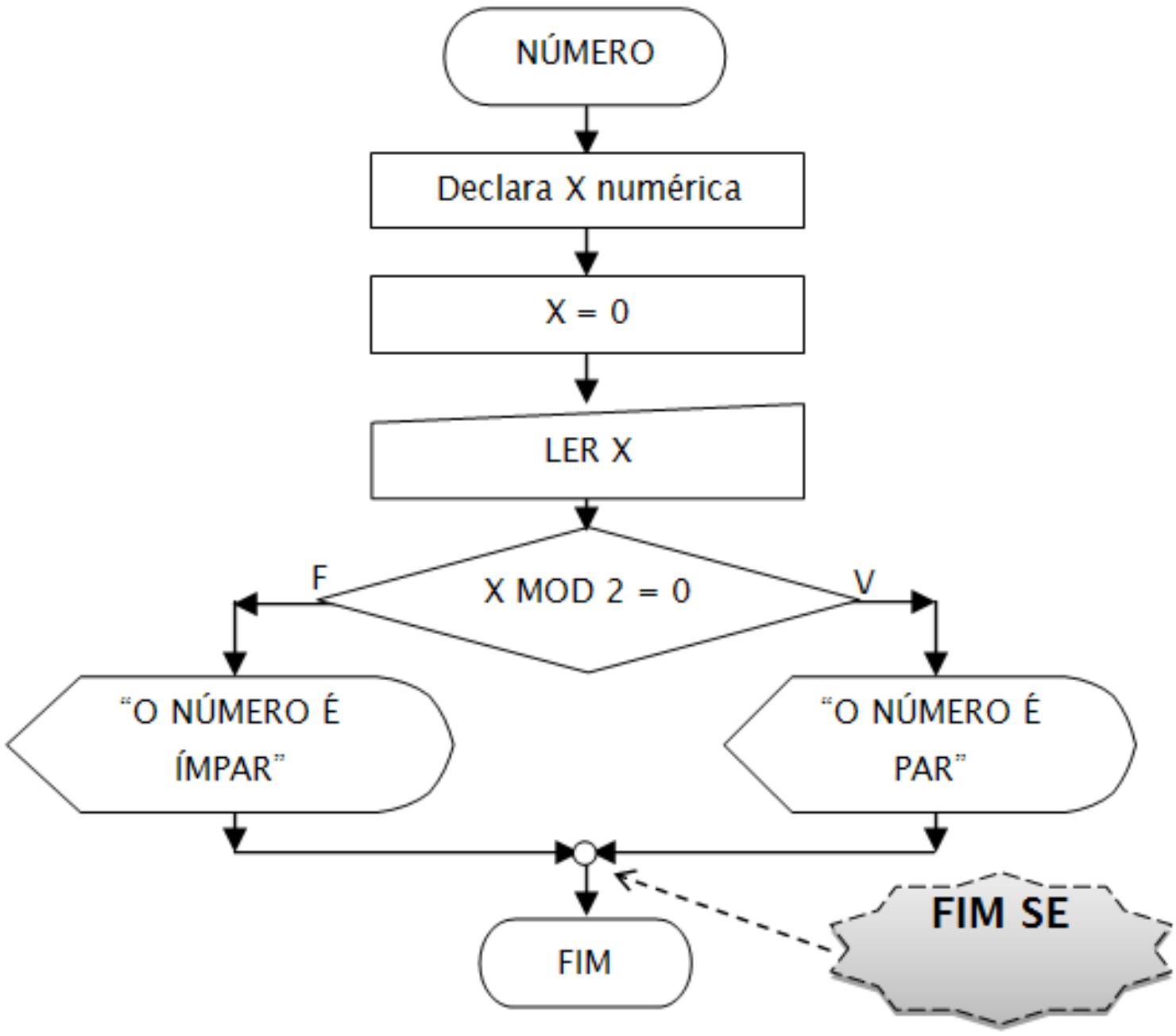
```
a = 0
b = 4
c = 3
h = 0
h = b^2 + c^2
h = 4^2 + 3^2
h = 16 + 9
h = 25
a = 25 // 2
a = 5
Exibir a  ← Exibe o valor 5
```

- **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
```



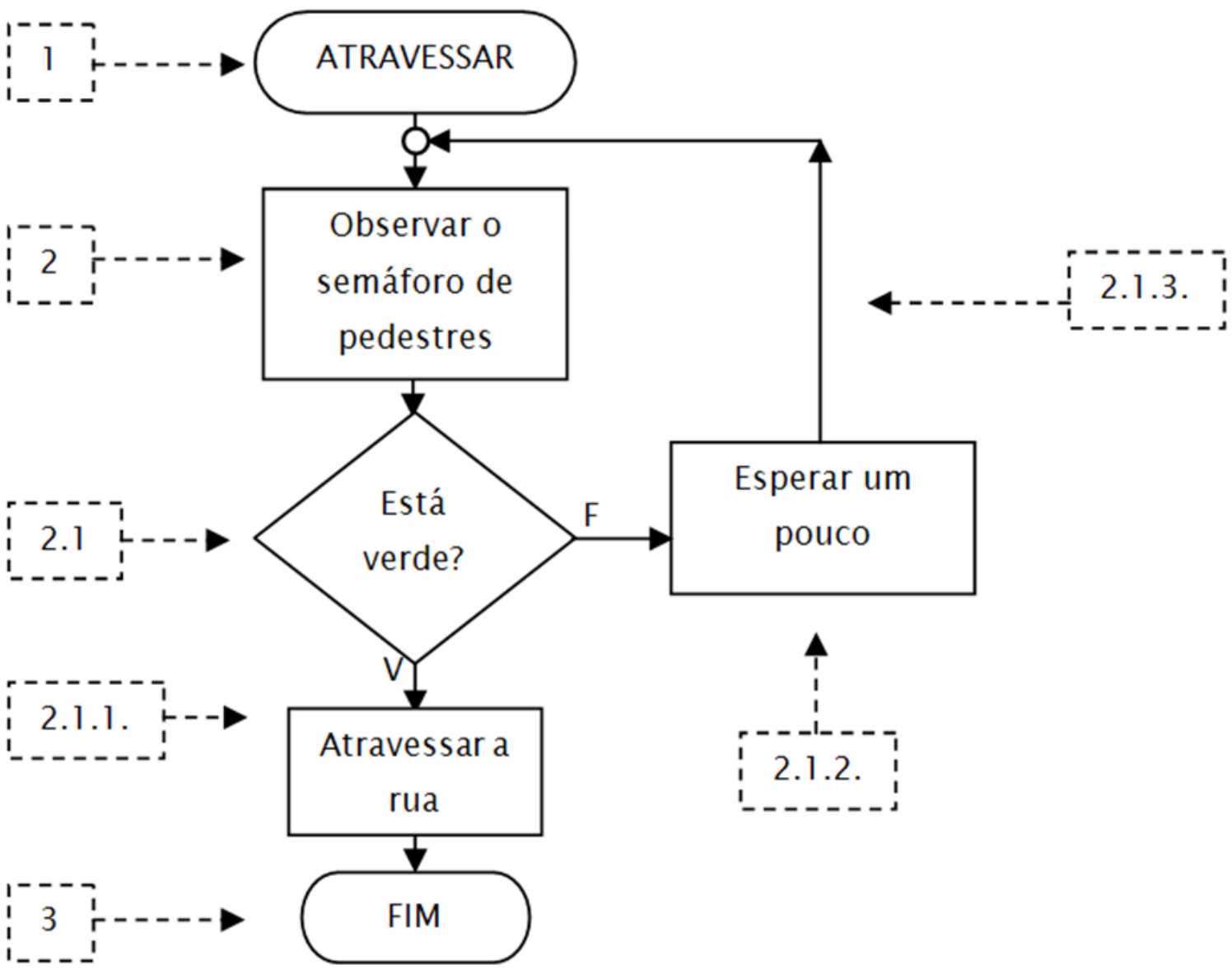

Note que depois do **FIM SE** só existe uma seta fluxo, ou seja, só existe um caminho a seguir. O comando **FIM SE** é a união das setas dos fluxos que vêm do lado verdadeiro e do lado falso da decisão.

- **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.

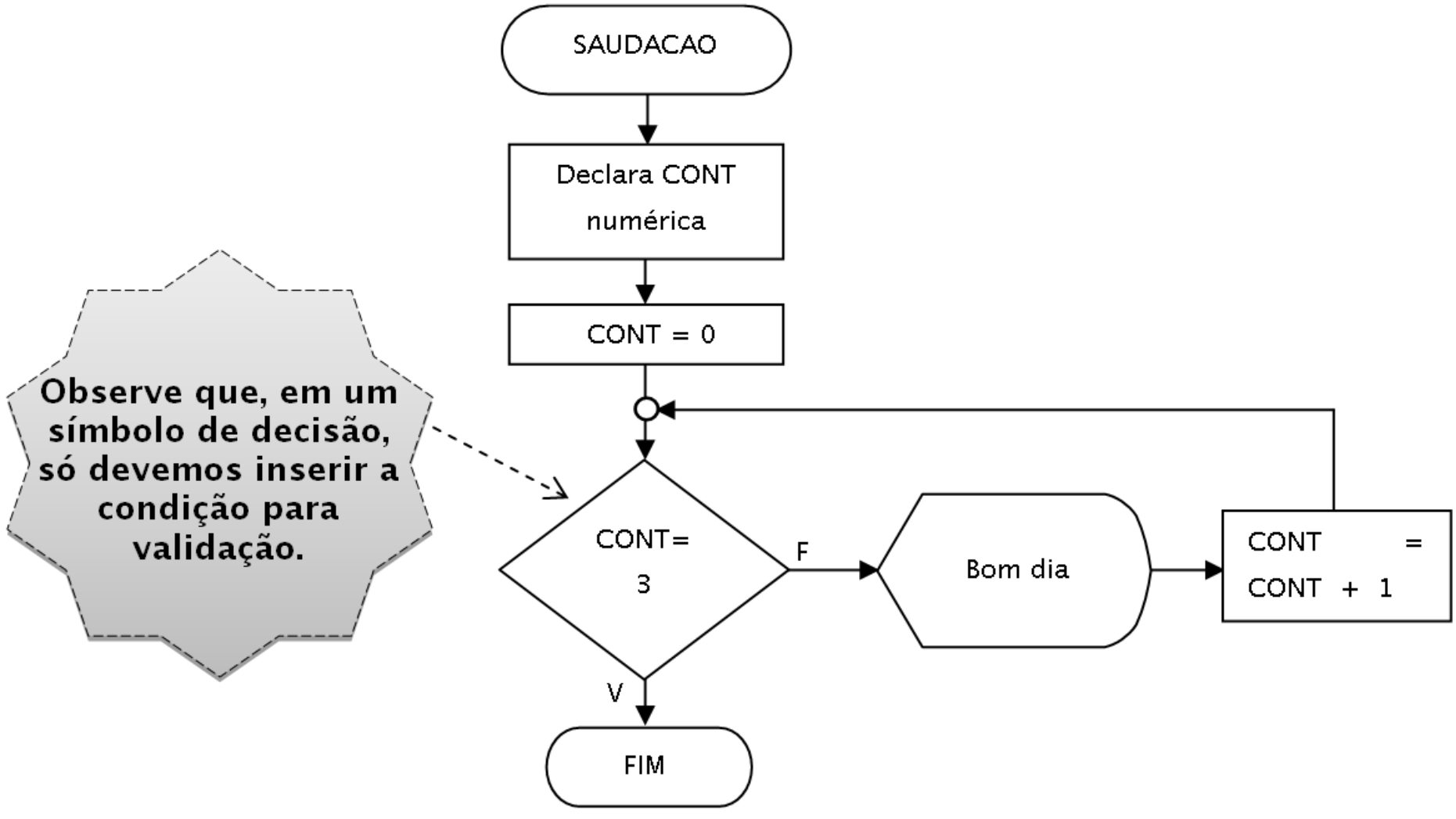
1. ATRAVESSAR
2. Observar o semáforo de pedestres
   2.1. Verifique se está verde
      2.1.1. Se sim: Atravessar a rua
      2.1.2. Senão: Esperar um pouco
      2.1.3. Vá para o passo 2
3. FIM




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.

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



# 4.4.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
```

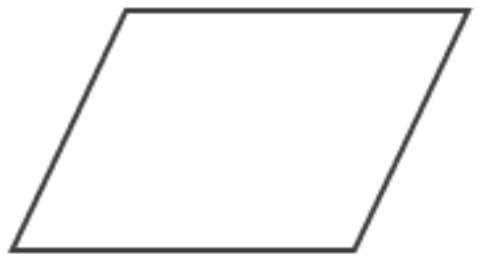
O resultado mostrado representa o **Teste de Mesa** exibindo todos os valores que a variável **CONT** recebeu.

Teste seus conhecimentos

# Fluxograma

4

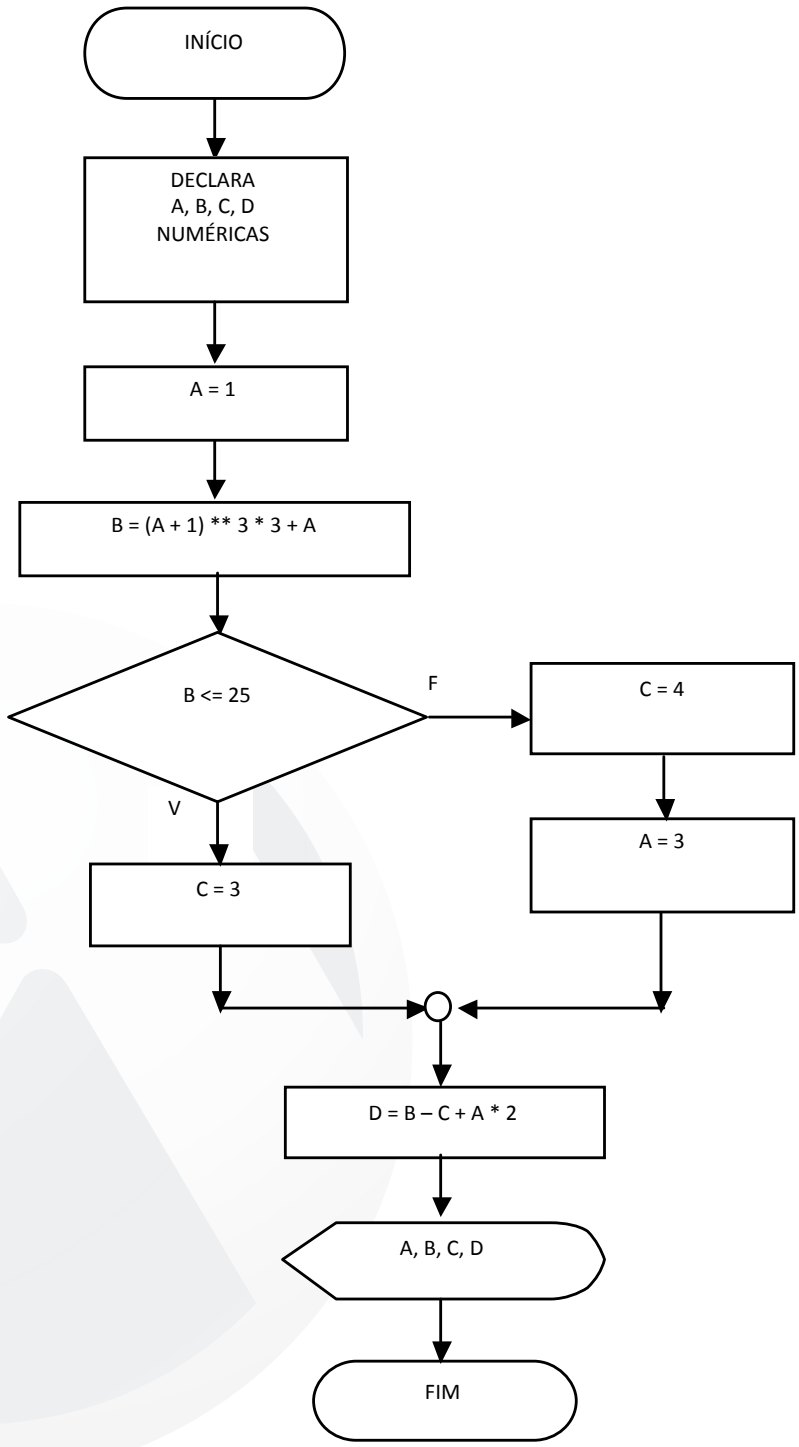
## 1. O símbolo a seguir representa qual passo de um algoritmo?



- ☐ a) Estrutura de decisão
- ☐ b) Estrutura de laço
- ☐ c) Entrada de dados
- ☐ d) Ação
- ☐ e) Exibir na tela

## 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
```

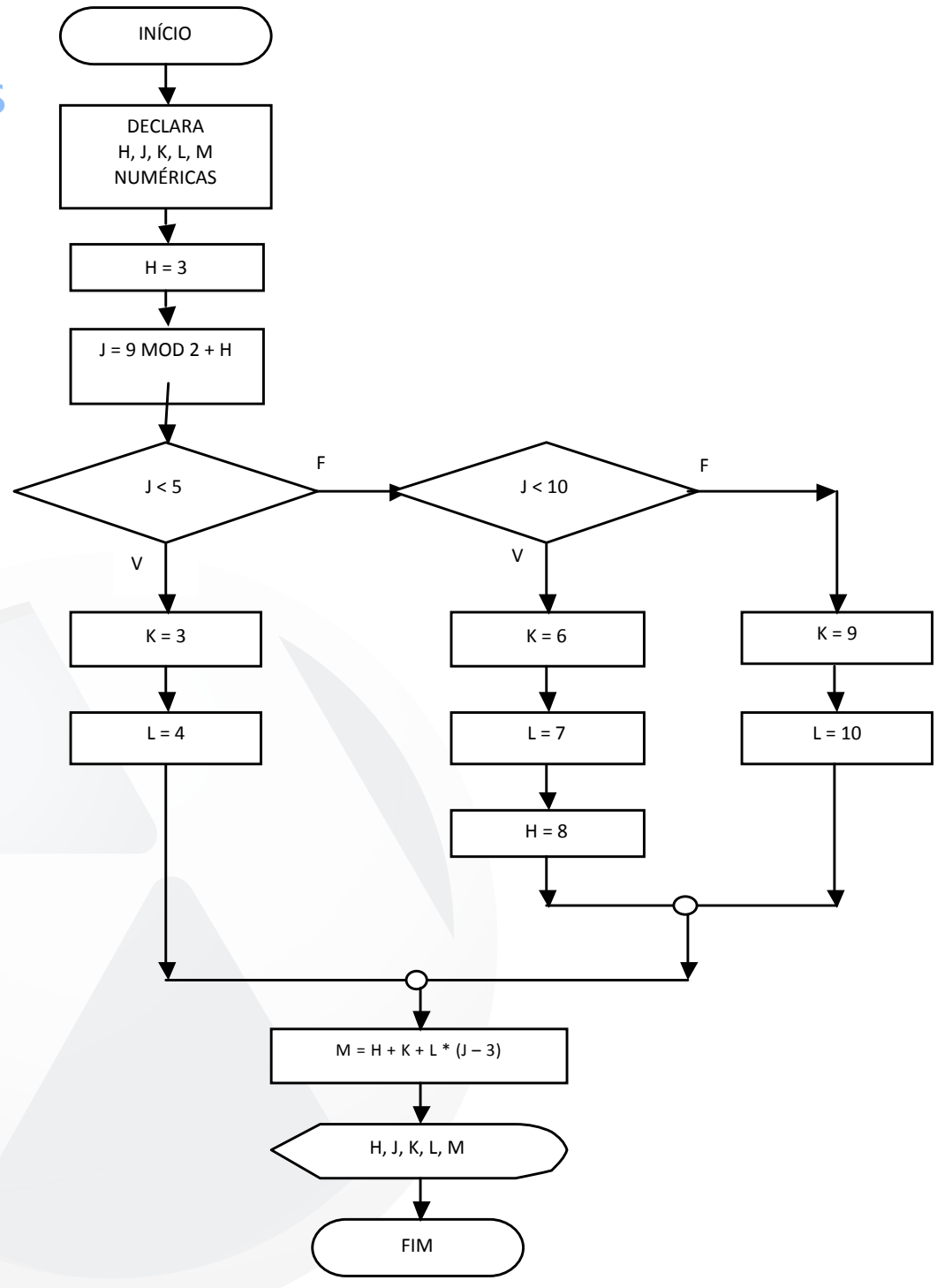



**Após o teste de mesa, os valores exibidos das variáveis A, B, C e D são:**

- ☐ 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.

## 3. Dado o seguinte algoritmo e seu respectivo fluxograma:

```
INÍCIO
DECLARA H, J, K, L, M NUMÉRICAS
H = 3
J = 9 MOD 2 + H
SE J < 5
ENTÃO K = 3
                L = 4
SENÃO     SE J < 10
          ENTÃO  K = 6
                 L = 7
                 H = 8
          SENÃO  K = 9
                 L = 10
      FIM SE
FIM SE
M = H + K + L * (J – 3)
EXIBIR H, J, K, L, M
FIM
```

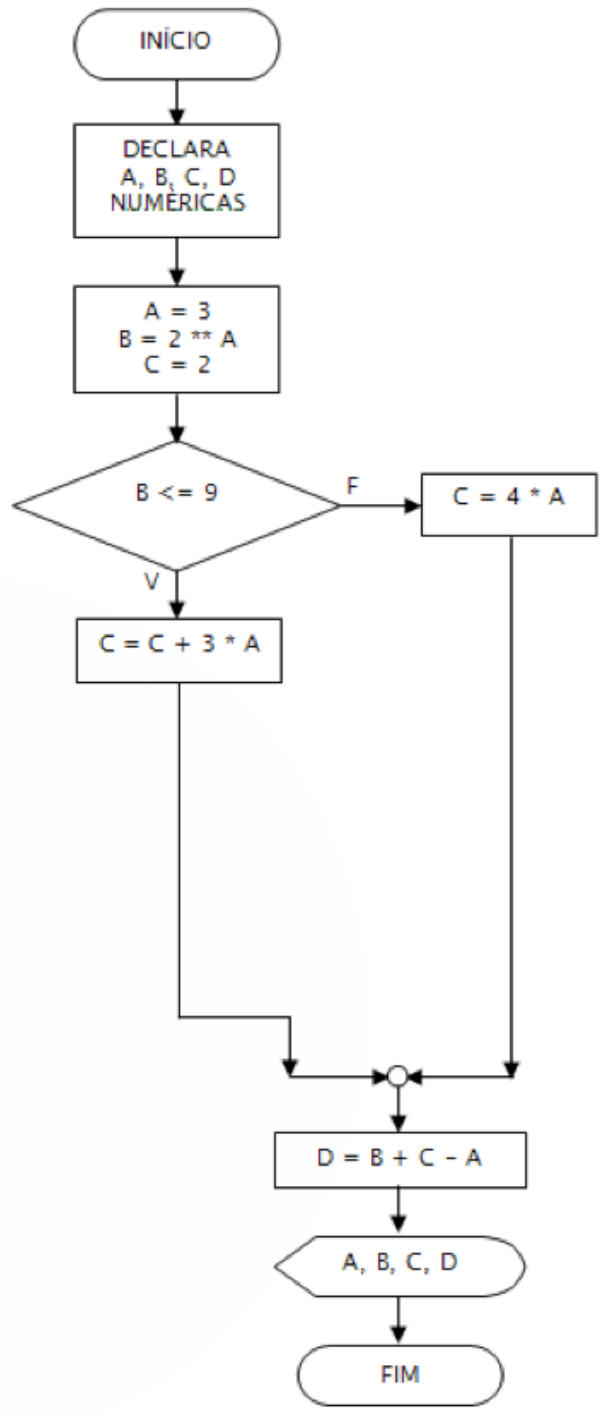



**Após o teste de mesa, os valores exibidos das variáveis H, J, K, L e M são:**

- ☐ 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 = 2
SE B <= 9
     ENTÃO C = C + (3 * A)
     SENÃO  B = 2
                 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 e D são:**

- ☐ a) 3, 25, 12, 34
- ☐ b) 2, 3, 9, 8
- ☐ c) 3, 8, 11, 16
- ☐ d) 3, 8, 12, 8
- ☐ e) Nenhuma das alternativas anteriores está correta.

**5. Dado o seguinte algoritmo:**

```
INÍCIO
 DECLARA H, J, K, L, M NUMÉRICAS
 H = 3
 J = 10 DIV H
 SE J <= 5
  ENTÃO K = 3 ** H
           L = 5 - H
  SENÃO  SE J <=10
                ENTÃO K = 4 * H
                         L = 8 / H
                         H = 8
           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) 3, 3, 27, 2, 7
- ☐ 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.

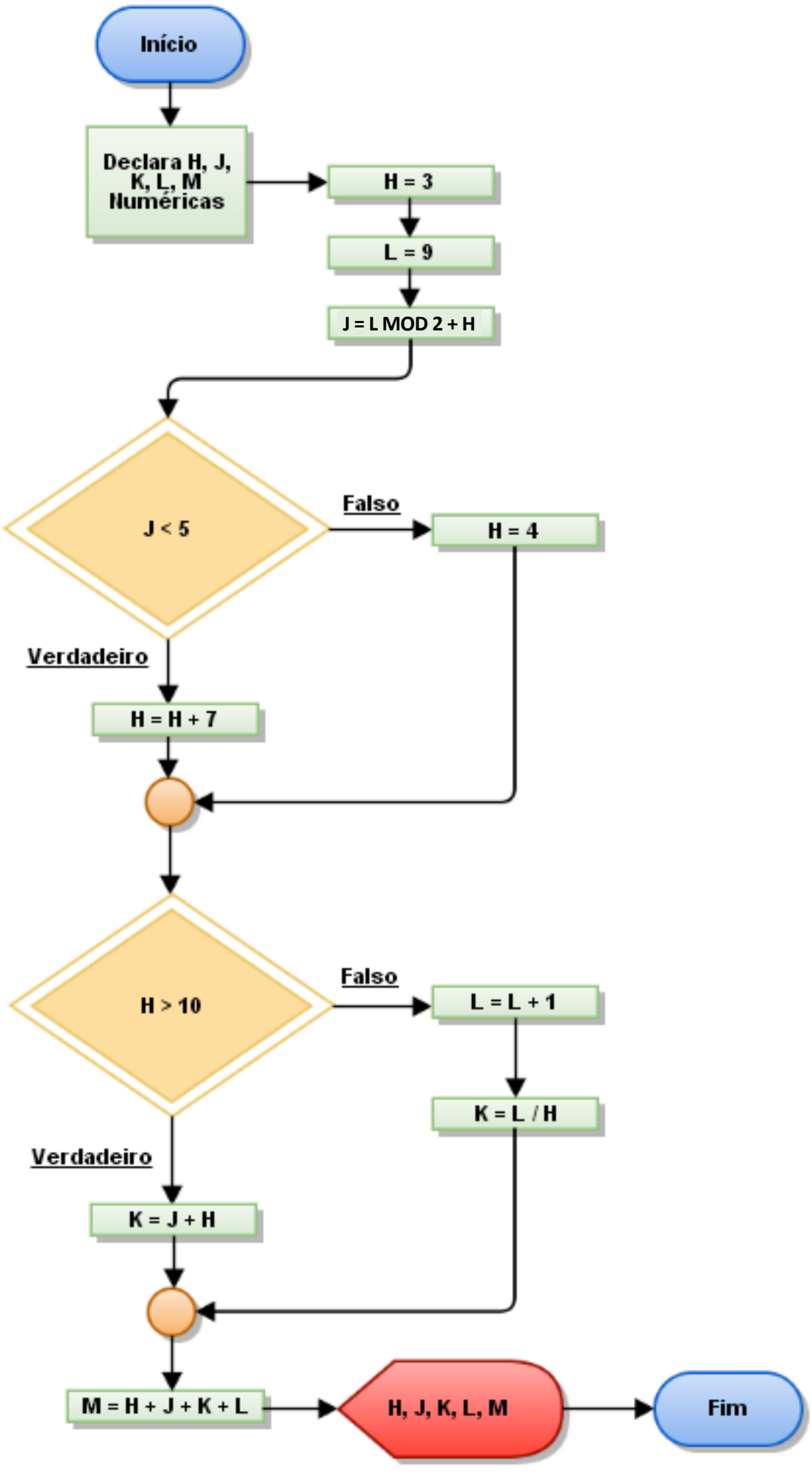
4

Mãos à obra!

# Fluxograma

# Exercício 1

Faça o teste de mesa do fluxograma a seguir:



# Laço ou loop e repetição

5

- ✓ Estruturas de repetição com comando FOR...NEXT (PARA...PRÓXIMO);
- ✓ Estruturas de repetição com comando WHILE (ENQUANTO);
- ✓ Estruturas de repetição com comando DO...WHILE (FAÇA...ENQUANTO).

# 5.1.Introdução

Nesta aula, 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.

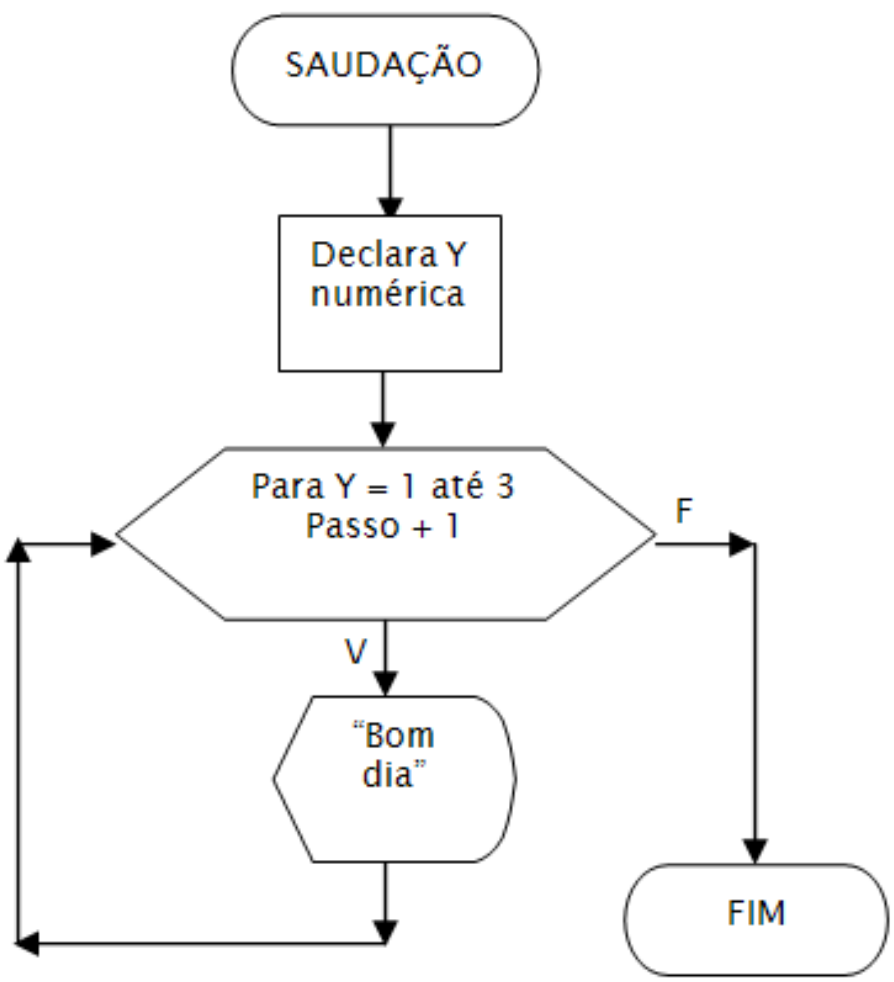
# 5.2.Comando FOR…NEXT (PARA...PRÓXIMO)

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).

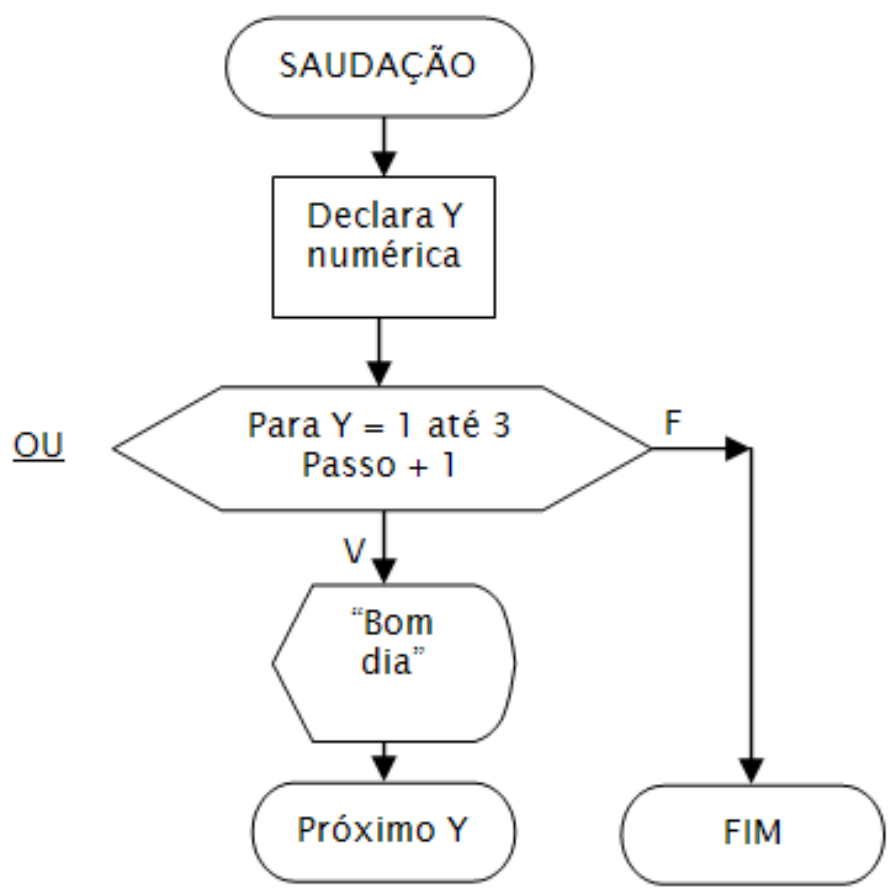
Na sintaxe do comando **FOR**, declaramos sua inicialização, sua condição e seu incremento. A cada vez que o loop é executado, a variável do comando que é utilizada como contador aumenta ou diminui de acordo com o incremento, até que a condição do comando não seja mais satisfeita. Esse tipo de loop é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

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

```
SAUDAÇÃO
  Declara Y numérica
  Para Y = 1 até 3 passo + 1
    Exibir "Bom dia"
  Próximo Y
FIM
```




OU

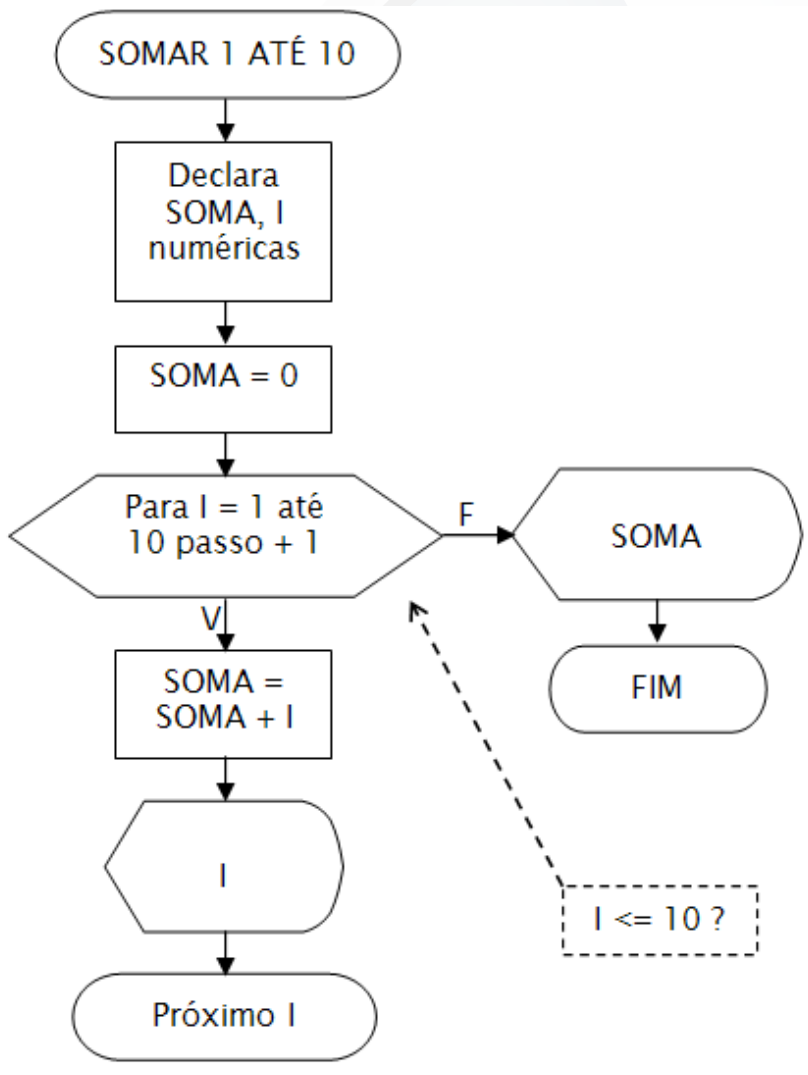



Note que no fluxo da direita está sendo utilizado um símbolo de terminação com o comando **PRÓXIMO Y**. Esse símbolo representa a seta do fluxo que retorna ao símbolo de preparação, conforme mostra o fluxo da esquerda. Depois deste símbolo não há mais símbolos. As duas estruturas são corretas.

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:




**Teste de mesa**

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| I = | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Soma = | 0 | 1 | 3 | 6 | 10 | 15 | 21 | 28 | 36 | 45 | 55 |

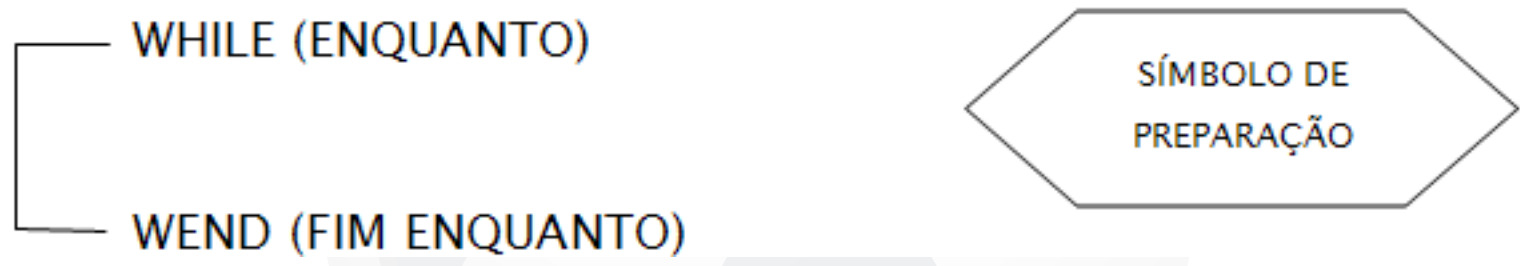
Serão exibidos todos os valores de **I** e somente o último valor de **SOMA**

# 5.3.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.

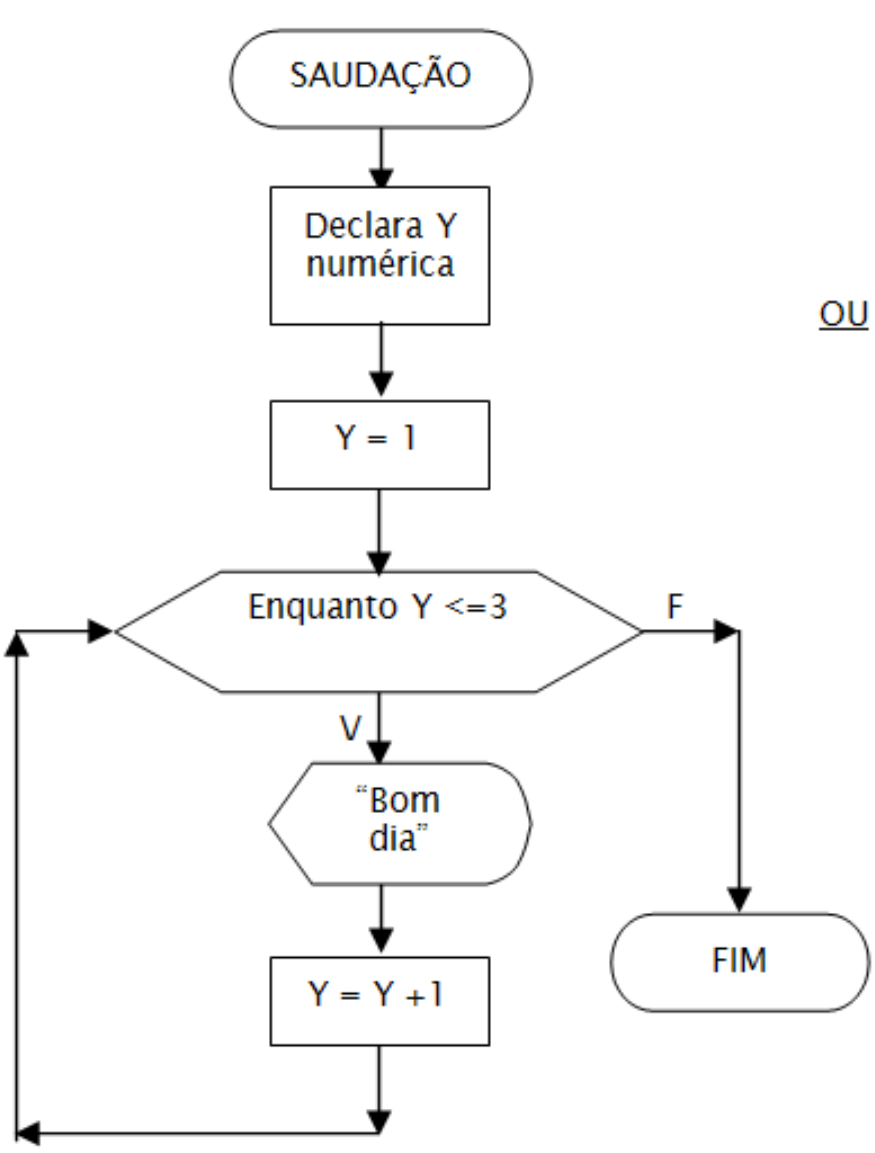
Na sintaxe do comando **WHILE**, declaramos apenas sua condição. Sua inicialização e seu incremento são ações executadas separadamente.

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**.

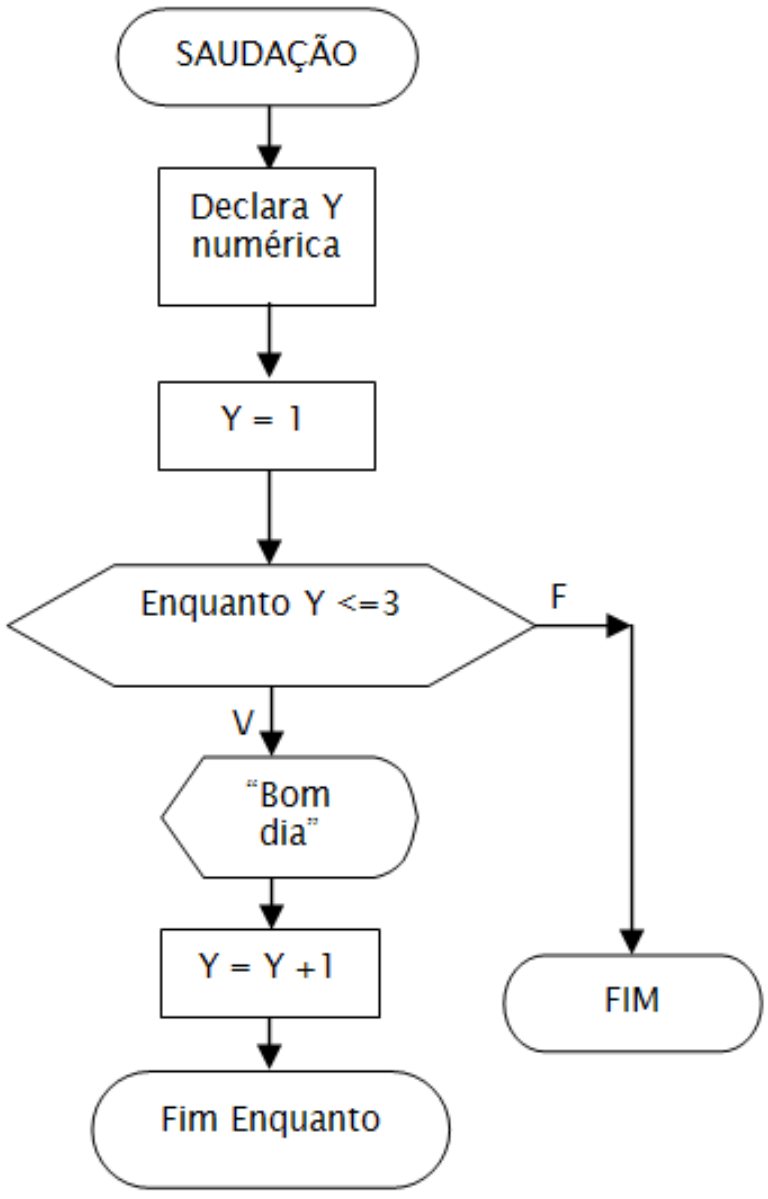



A seguir, temos um exemplo de algoritmo e fluxograma de um programa com o comando **ENQUANTO** que exibe três vezes a mensagem "**Bom dia**":

```
SAUDAÇÃO
   Declara Y numérica
   Y = 1
   Enquanto Y <= 3
    Exibir "Bom dia"
    Y = Y + 1
   Fim Enquanto
FIM
```




OU

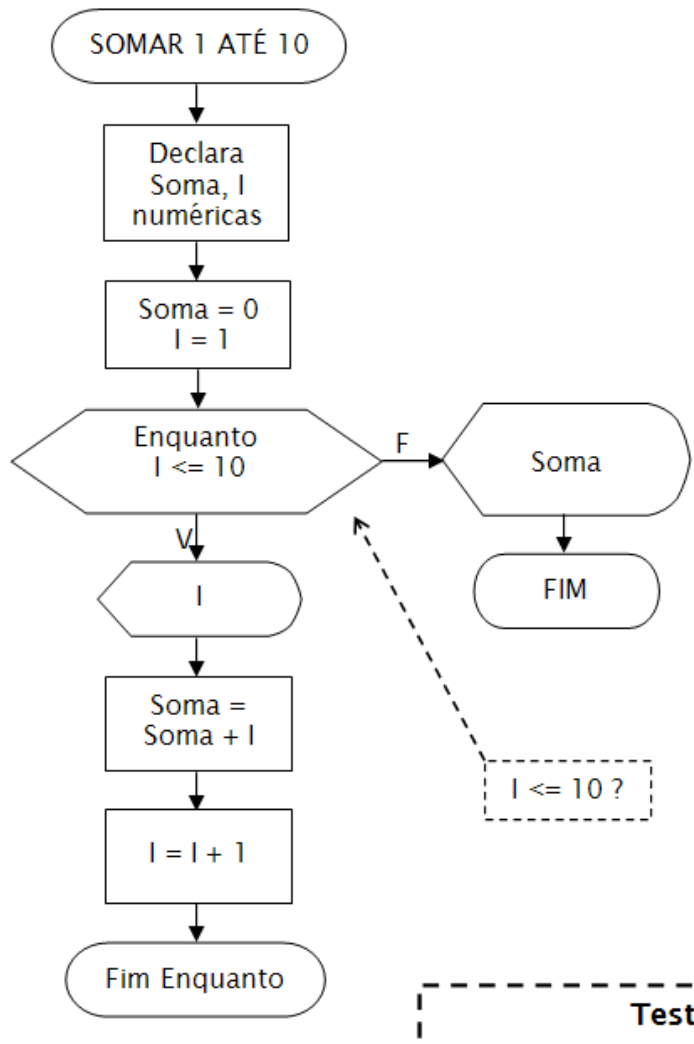



Note que no fluxo da direita está sendo utilizado um símbolo de terminação com o comando **FIM ENQUANTO** no lugar das setas. As duas estruturas são corretas.

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:




**Teste de mesa**

| I = | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Soma = 0 | 1 | 3 | 6 | 10 | 15 | 21 | 28 | 36 | 45 | 55 |

Serão exibidos todos os valores de **I** e somente o último valor de **SOMA**

# 5.4.Comando DO...WHILE (FAÇA....ENQUANTO)

Um loop com o comando **DO...WHILE** é utilizado quando é necessário executar um trecho de código ao menos uma vez, antes da condição do loop ser checada para continuar ou não a iteração.

Isso quer dizer que, mesmo quando a condição estabelecida pelo loop é falsa, quando utilizamos **DO...WHILE**, o trecho de código será executado uma vez antes de ter sua iteração negada.

Assim como o comando **WHILE**, o **DO...WHILE** é utilizado quando não conhecemos o número de iterações que iremos realizar.

A seguir, temos um exemplo de algoritmo que utiliza o comando DO...WHILE para somar números até que o usuário digite o número zero (0):

```
SOMAR NÚMEROS
  Declara SOMA, X numéricas
  SOMA = 0
  Faça:
   Ler X
   Se X != 0:
         SOMA  = SOMA + X
  Enquanto X != 0
  Exibir SOMA
FIM
```

Veja que, como já havíamos atribuído o valor zero à variável **SOMA** antes de iniciar o loop, se utilizássemos a estrutura **WHILE**, o trecho de código não seria executado nenhuma vez. Com o uso do comando DO WHILE, é possível forçar a execução do trecho de código e obter resultados muito diferentes.

Teste seus conhecimentos

# Laço ou loop e repetição

5

**1. Quais são os três tipos de laço utilizados nas linguagens de programação modernas?**

- ☐ a) PARA, ENQUANTO e PARA-ENQUANTO
- ☐ b) PARA, ENQUANTO e FAÇA-ENQUANTO
- ☐ c) FAÇA, DURANTE e FAÇA-DURANTE
- ☐ d) FAÇA, DURANTE e PARA-ENQUANTO
- ☐ e) Nenhuma das alternativas anteriores está correta.

**2. Quais as principais características do laço ENQUANTO e do laço PARA?**

- ☐ a) Quando não sabemos quantas vezes o trecho de código será repetido utilizamos o laço do tipo ENQUANTO.
- ☐ b) No laço PARA, o contador interrompe a repetição do código, ao tornar falsa a condição do laço.
- ☐ c) No laço ENQUANTO não inicializamos uma variável e nem um incremento, apenas a condição que queremos testar.
- ☐ d) Podemos utilizar o laço ENQUANTO na validação do programa.
- ☐ e) Todas as alternativas anteriores estão corretas.

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

```
PROGRAMA1
INÍCIO
   Declara SOMA, C numéricas
   SOMA = 0
   Para C = 1, C < = 5, C = C + 1
         SOMA = SOMA + 2 * C
   Próximo C
   Exibir SOMA, C
FIM
```

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

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

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

```
PROGRAMA2
INÍCIO
   Declara SOMA, C numéricas
   SOMA = 0
   Para C = 1, C < = 10, C = C + 2
                SOMA = SOMA + C
   Próximo C
   Exibir SOMA, C
FIM
```

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

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

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

```
PROGRAMA3
INÍCIO
   Declara SOMA, C numéricas
   SOMA = 0
   Para C = 1, C = 5, C = C + 1
              SOMA = SOMA + C / 2
   Próximo C
   Exibir SOMA, C
FIM
```

**Após o teste de mesa, os valores exibidos das variáveis SOMA e C 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.

**6. Faça o teste de mesa para o algoritmo a seguir:**

```
PROGRAMA4
INÍCIO
   Declara A, B numéricas
   A = 17
   B = 0
   Enquanto A maior ou igual 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.

**7. Faça o teste de mesa para o algoritmo a seguir:**

```
PROGRAMA5
INÍCIO
   Declara A, B numéricas
   A = 42
   B = 0
   Enquanto A MOD 5 Diferente de  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.

5

Mãos à obra!

# Laço ou loop e repetição

# Exercício 1

Escreva um algoritmo que, utilizando o comando PARA, calcule a soma nos N primeiros números pares, e considerando que o valor de N deverá ser digitado pelo usuário.

# Variáveis indexadas e Laços encadeados

6

- ✓ Vetores e matrizes;
- ✓ Laços encadeados.

# 6.1.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.

VETOR

Y

| | |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |
| 10 | |

MATRIZ

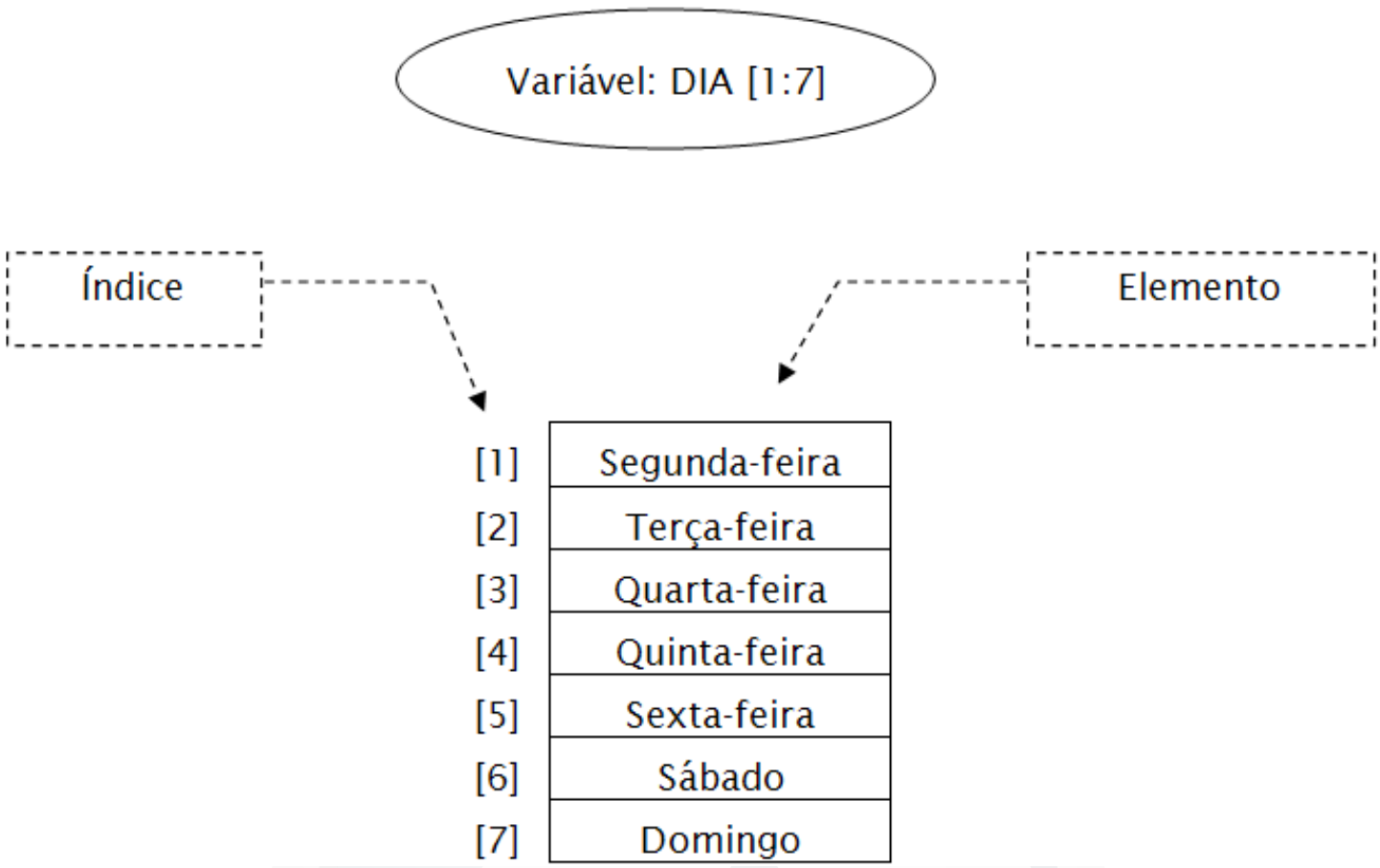
Z

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | |

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

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

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:

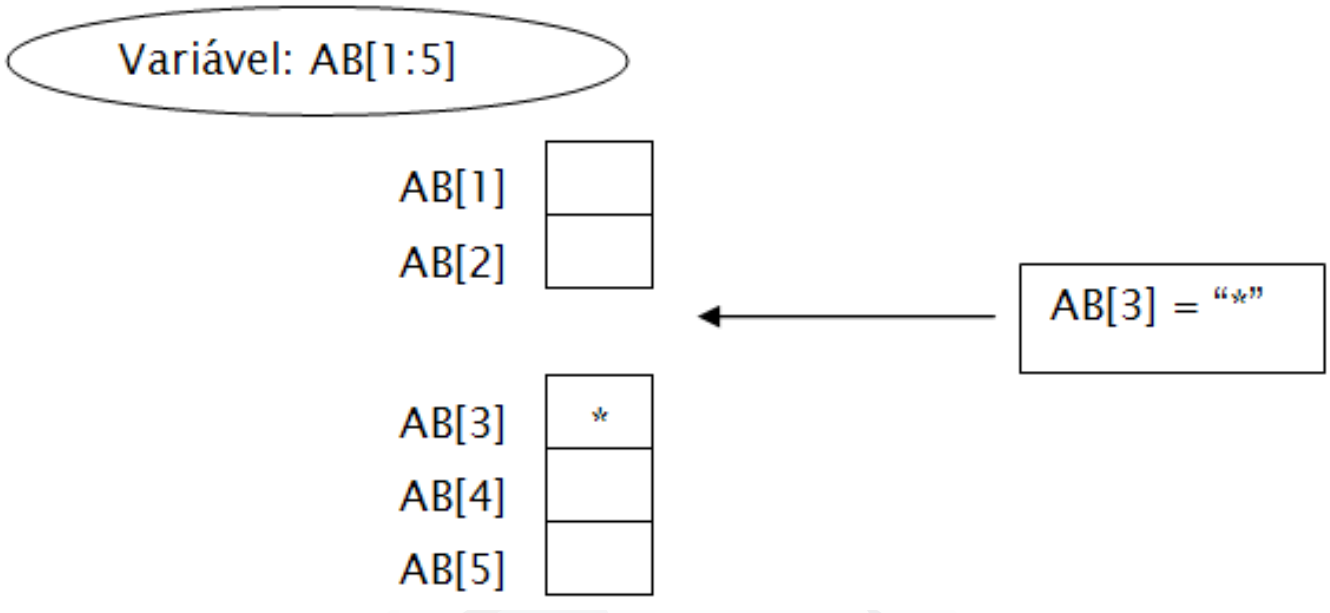
A = 2 ➔ DIA[ A ] = Terça-feira

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

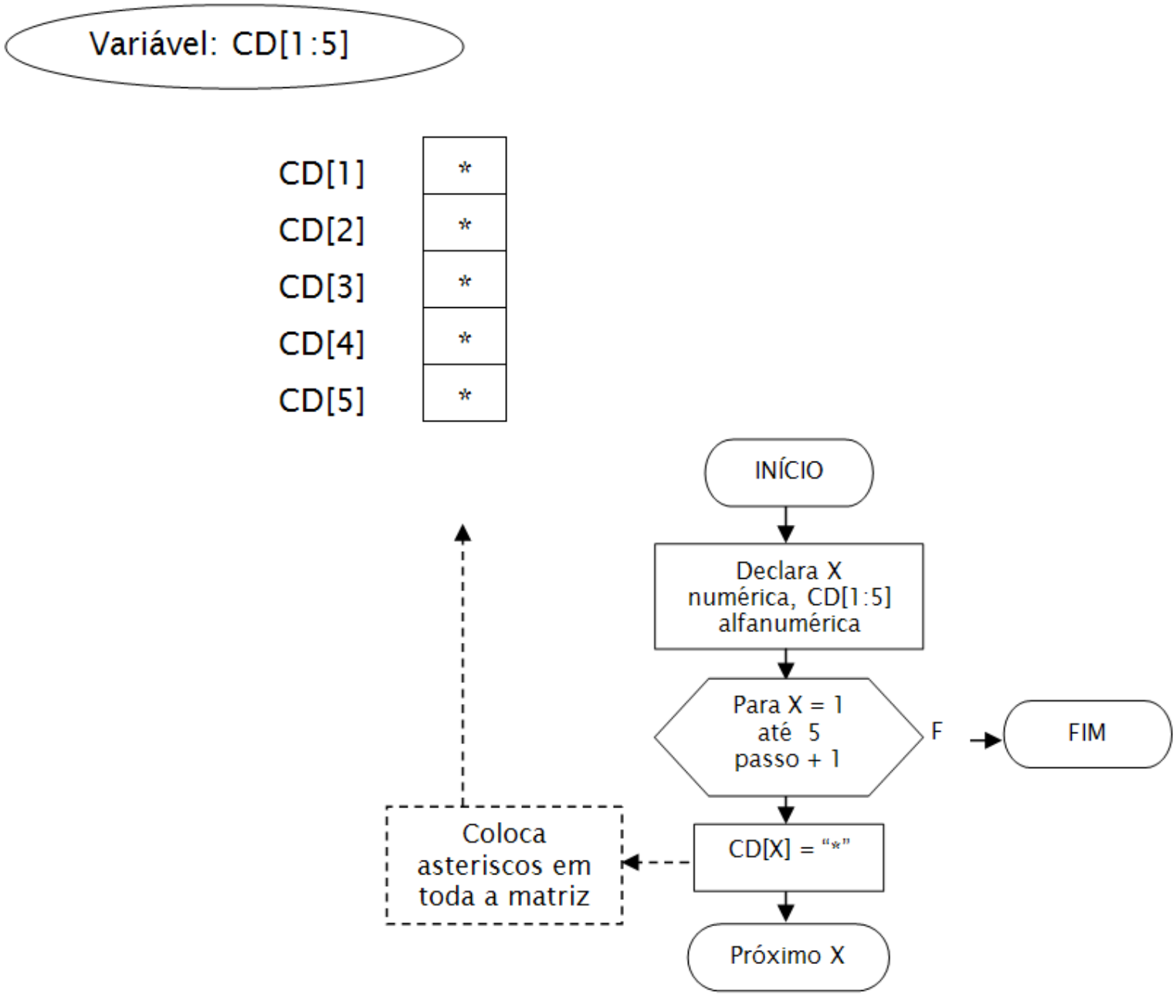
DIA[ 2 ] = Terça-feira

Vejamos estes outros exemplos:

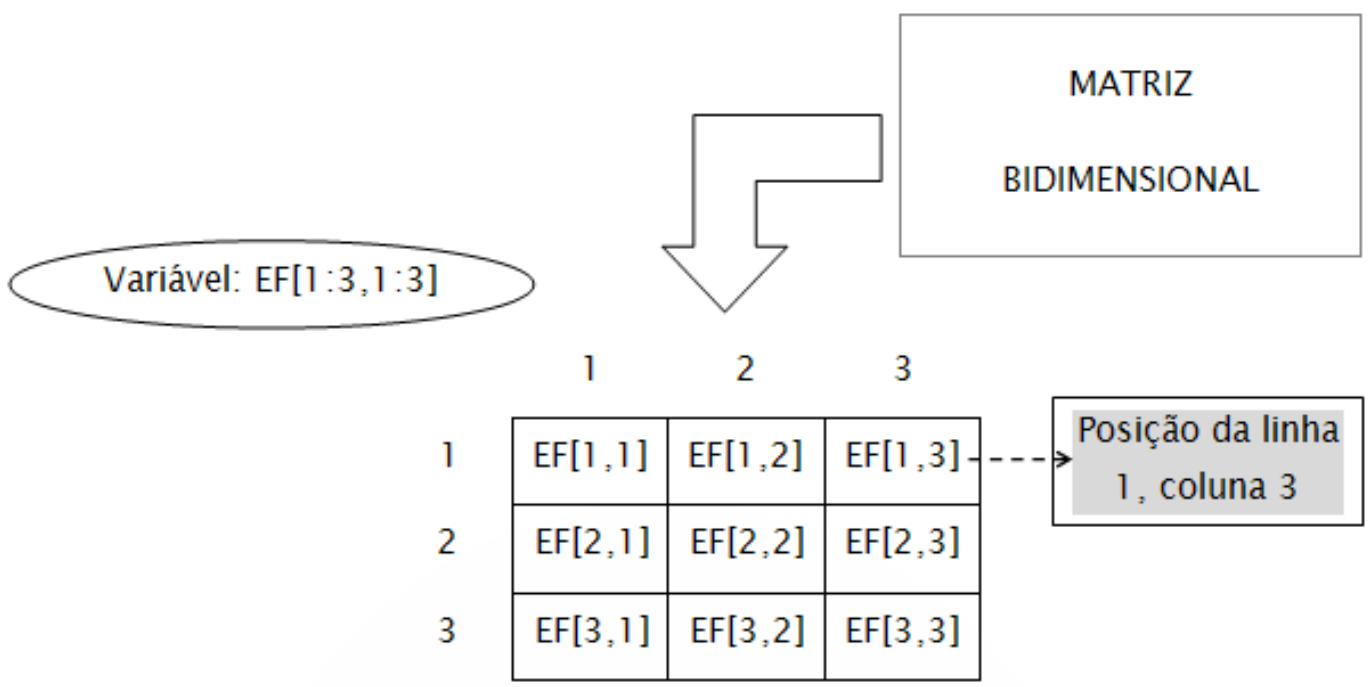
- **Exemplo 1**

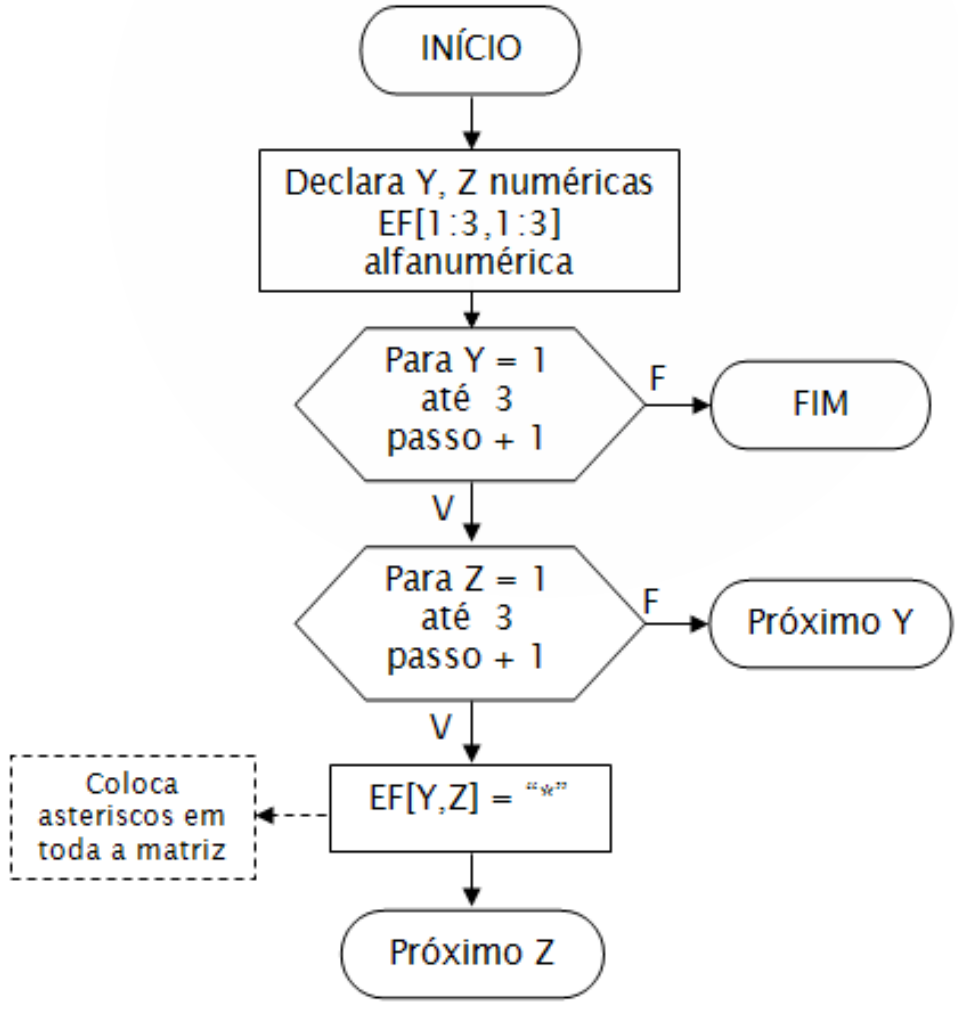



- **Exemplo 2**

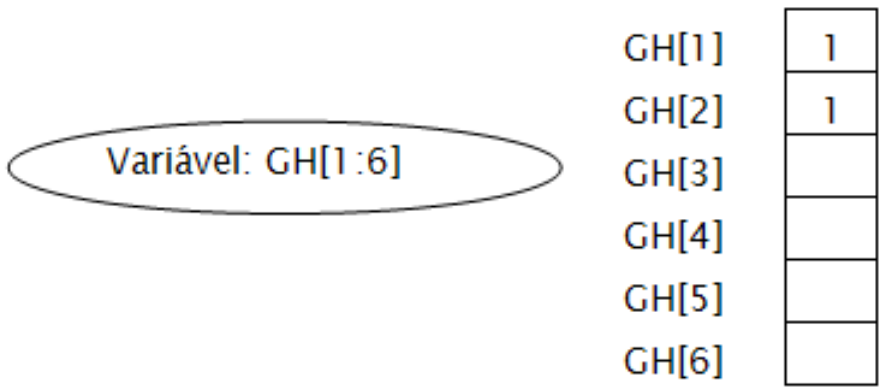


- **Exemplo 3**


MATRIZ
BIDIMENSIONAL
Variável: EF[1:3,1:3]
1
2
3
1
2
3
EF[1,1]
EF[1,2]
EF[1,3]
EF[2,1]
EF[2,2]
EF[2,3]
EF[3,1]
EF[3,2]
EF[3,3]
Posição da linha 1, coluna 3


INÍCIO
Declara Y, Z numéricas
EF[1:3,1:3]
alfanumérica
Para Y = 1
até 3
passo + 1
F
FIM
V
Para Z = 1
até 3
passo + 1
F
Próximo Y
V
EF[Y,Z] = "*"
Coloca asteriscos em toda a matriz
Próximo Z

- **Exemplo 4**




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

- **Algoritmo**

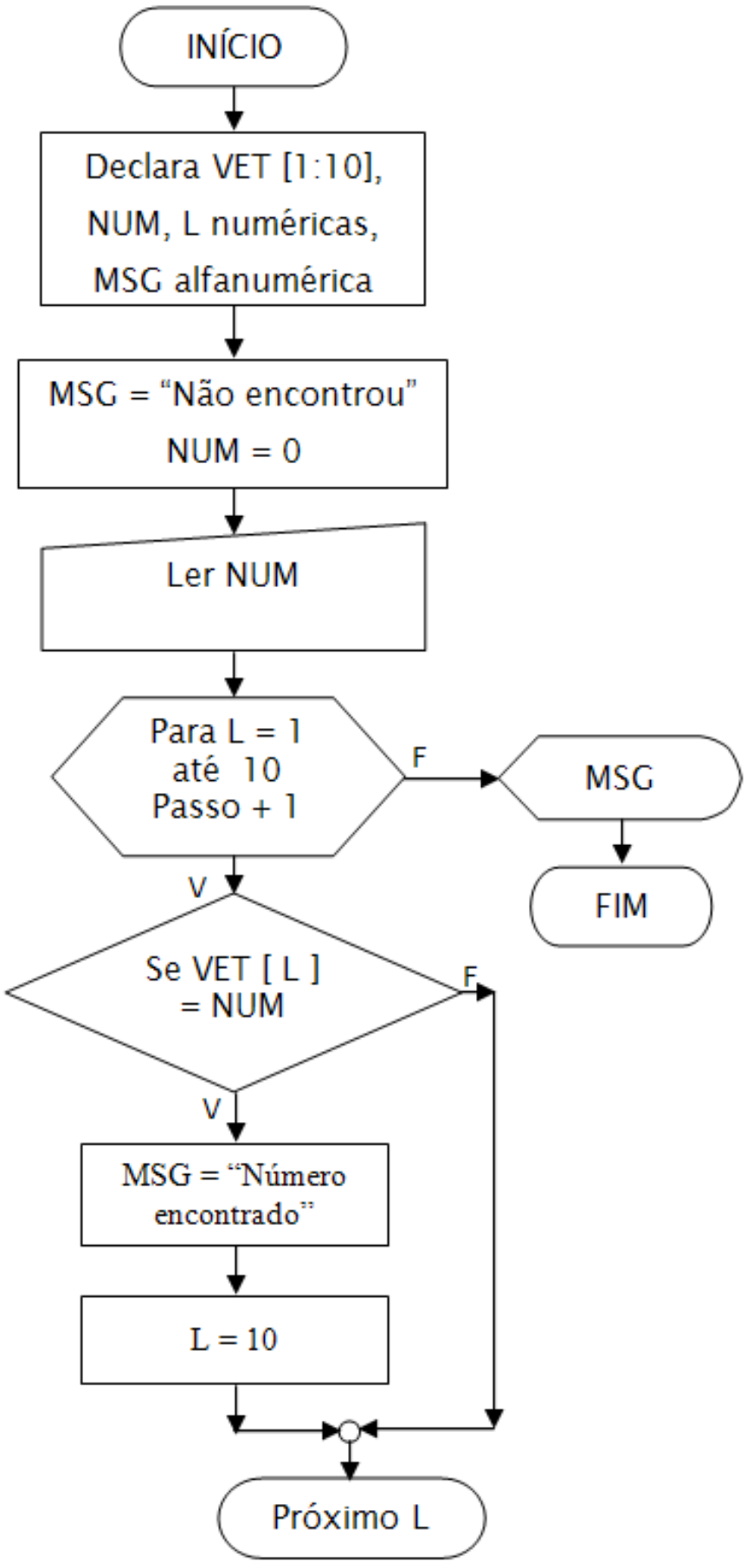
```
INÍCIO
   Declara GH[1:6], J numéricas
   GH[1] = 1
   GH[2] = 1
   Para J = 3 até 6 passo + 1
         GH[J] = GH[J – 1] + GH[J – 2]
   Próximo J
FIM
```

- **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
```

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 ]**.

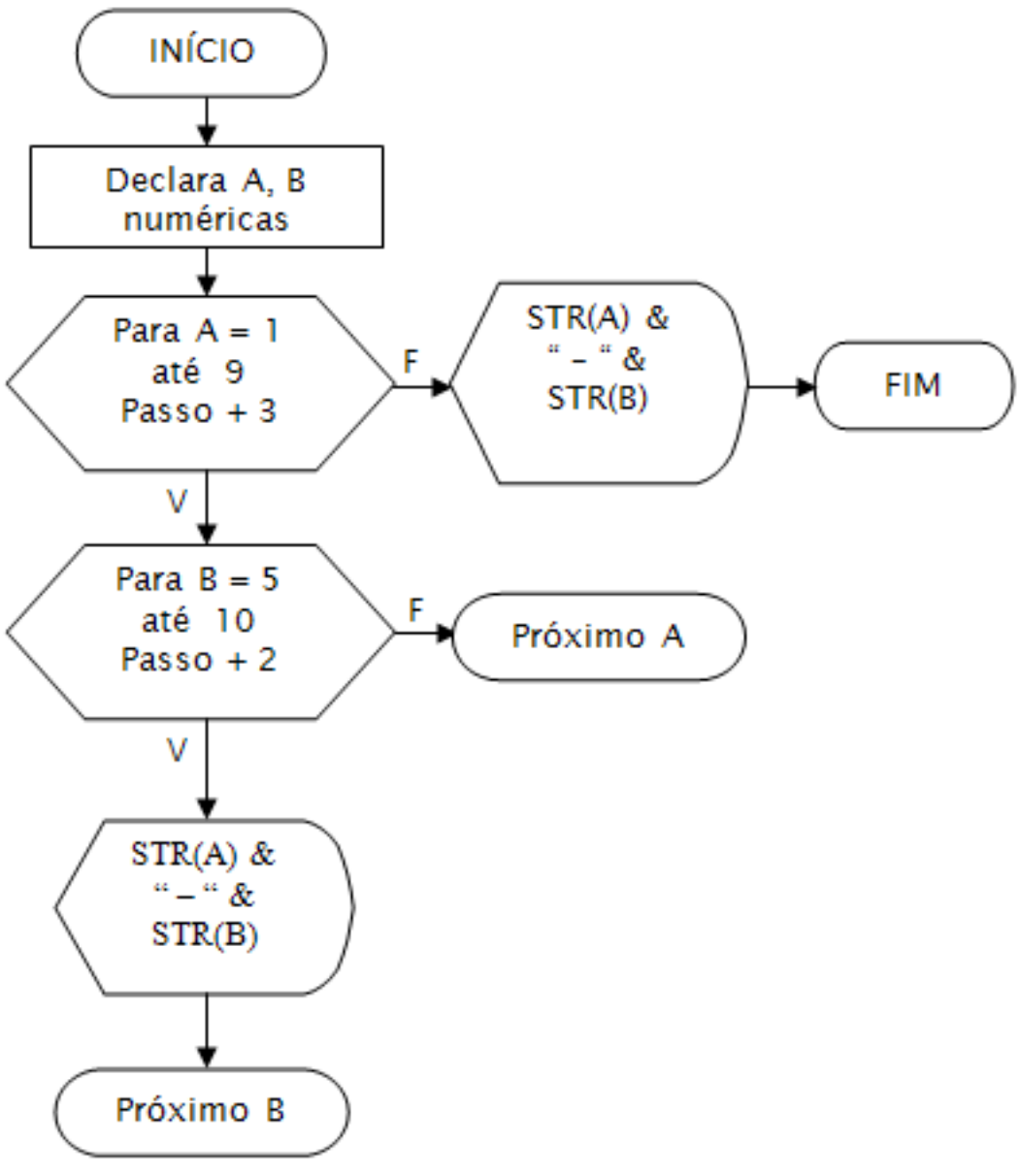
```
INÍCIO
  Declara VET [ 1:10 ], NUM, L numéricas, MSG alfanumérica
  MSG = “Não encontrou”
  NUM = 0
  Ler NUM
  Para L = 1 até 10 passo + 1
      Se VET [ L ] = NUM
            Então MSG =       “Número encontrado”
                  L = 10
      Fim Se
  Próximo L
  Exibir MSG
FIM
```


INÍCIO
Declara VET [1:10], NUM, L numéricas, MSG alfanumérica
MSG = "Não encontrou"
NUM = 0
Ler NUM
Para L = 1 até 10 Passo + 1
F
MSG
FIM
V
Se VET [ L ] = NUM
F
V
MSG = "Número encontrado"
L = 10
Próximo L

# 6.2.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
```




| 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 |

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

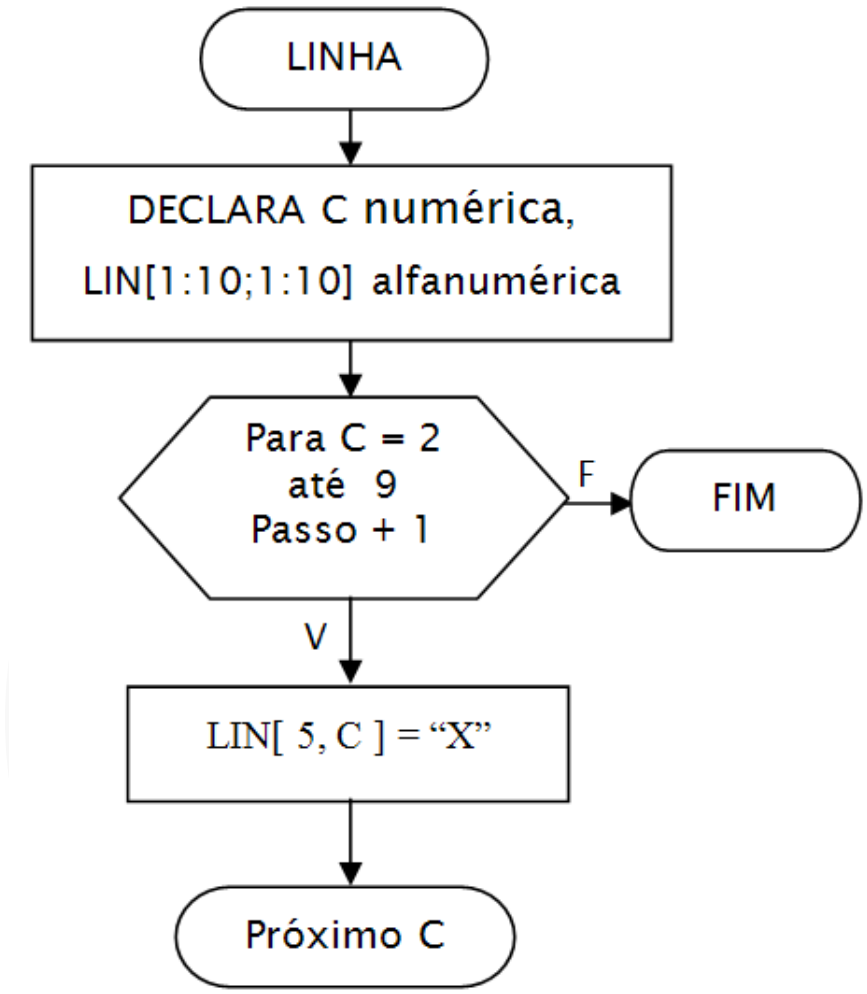
**Matriz**

**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.



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 | | |
| 3 | | X | X | X | X | X | X | X | | |
| 4 | | X | X | X | X | X | X | X | | |
| 5 | | X | X | X | X | X | X | X | | |
| 6 | | X | X | X | X | X | X | X | | |
| 7 | | X | X | X | X | X | X | X | | |
| 8 | | X | X | X | X | X | X | X | | |
| 9 | | | | | | | | | | |
| 0 | | | | | | | | | | |

```
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 | | L | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | | 3 | 2 | | 4 | 2 | | 5 | 2 | | 6 | 2 | | 7 | 2 | | 8 | 2 |
| 2 | 3 | | 3 | 3 | | 4 | 3 | | 5 | 3 | | 6 | 3 | | 7 | 3 | | 8 | 3 |
| 2 | 4 | | 3 | 4 | | 4 | 4 | | 5 | 4 | | 6 | 4 | | 7 | 4 | | 8 | 4 |
| 2 | 5 | | 3 | 5 | | 4 | 5 | | 5 | 5 | | 6 | 5 | | 7 | 5 | | 8 | 5 |
| 2 | 6 | | 3 | 6 | | 4 | 6 | | 5 | 6 | | 6 | 6 | | 7 | 6 | | 8 | 6 |
| 2 | 7 | | 3 | 7 | | 4 | 7 | | 5 | 7 | | 6 | 7 | | 7 | 7 | | 8 | 7 |
| 2 | 8 | | 3 | 8 | | 4 | 8 | | 5 | 8 | | 6 | 8 | | 7 | 8 | | 8 | 8 |

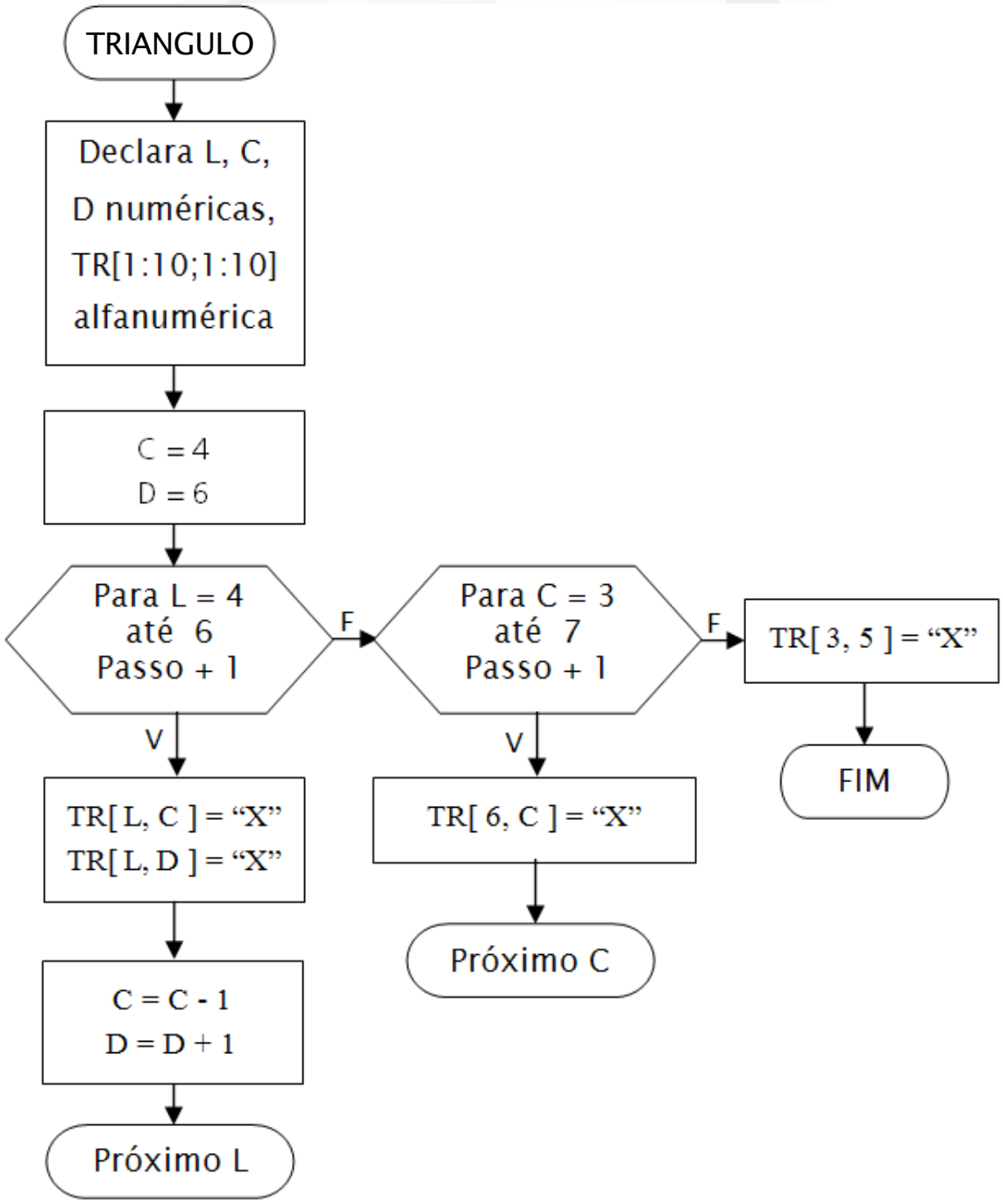
A seguir, temos um algoritmo que preenche a matriz conforme a figura utilizando dois laços:

**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** | | |
| 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
```



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 |

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
```

Teste seus conhecimentos

# Variáveis indexadas e laços encadeados

6

**1. Sobre as variáveis indexadas, julgue as afirmações a seguir:**

**1) Podem ser classificadas como vetores ou matrizes de acordo com sua dimensão.**
**2) Podemos armazenar dados de tipos diferentes em uma variável indexada.**
**3) Não podem ter mais que duas dimensões.**
**4) Podem ser referenciadas de maneira implícita ou explicita.**

- ☐ a) V, V, F, F
- ☐ b) F, V, F, V
- ☐ c) V, V, F, F
- ☐ d) V, F, F, V
- ☐ e) F, V, V, F

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

```
PROGRAMA1
INÍCIO
   DECLARA A, B, C NUMÉRICAS
A = 0
PARA B = 1, B < =  4, B = B + 1
        A = A + B
        PARA C = 1, C < = 3, C = C + 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.

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

```
PROGRAMA2
INÍCIO
  DECLARA A, B, C NUMÉRICAS
  A = 5
  PARA B = 2, B < = 4, B = B + 2
     A = A – B
     ENQUANTO A MAIOR QUE 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.

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

```
PROGRAMA3
INÍCIO
  DECLARA A, B, C NUMÉRICAS
  A = 5
  B = 1
  ENQUANTO A MAIOR QUE 2
     A = A – B
     PARA B = 2, B < = 3, B = B + 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.

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

```
PROGRAMA4
INÍCIO
  DECLARA A, B, C NUMÉRICAS
  A = 5
  B = 1
  ENQUANTO A MAIOR QUE 2
            A = A - B
            ENQUANTO A MAIOR QUE 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
- ☐ d) 3, 1, 2
- ☐ b) 2, 2, 4
- ☐ c) 2, 1, 4
- ☐ e) Nenhuma das alternativas anteriores está correta.

**6. Faça o teste de mesa para o algoritmo a seguir:**

```
PROGRAMA5
INÍCIO
  DECLARA A, B NUMÉRICAS
  DECLARA V[2] NUMÉRICA
  A = 2
  PARA B = 0, B < = 2, B = B + 1
            V[B] = A * B
            EXIBIR V[B]
  PRÓXIMO B
FIM
```

**Após o teste de mesa, os valores exibidos no vetor são:**

- ☐ a) 0, 2, 4
- ☐ b) 1, 4, 5
- ☐ c) 2, 4, 8
- ☐ d) 1, 2, 3
- ☐ e) Nenhuma das alternativas anteriores está correta.

6

Mãos à obra!

# Variáveis indexadas e laços encadeados

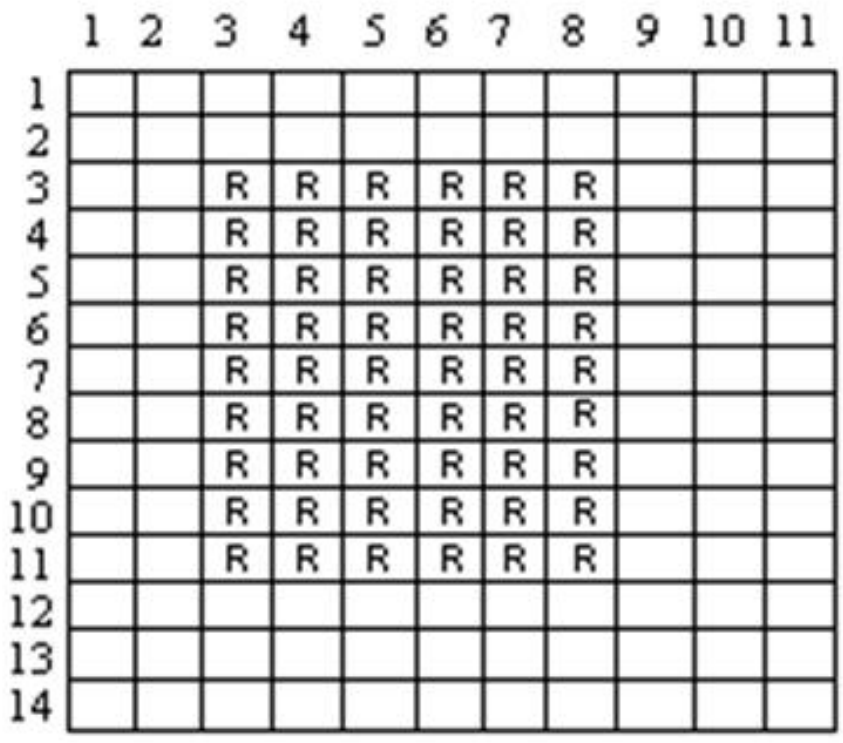
# Exercício 1

Dada a matriz XADREZ [8X8], escreva um algoritmo usando laço encadeado para preencher a matriz, conforme a figura a seguir:

| | 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 2

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



# Processamento predefinido

7

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

# 7.1.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 do processamento predefinido. 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 um processamento predefinido, é necessário declarar de que tipo 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.

# 7.2.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                          ← Programa principal
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


TROCAR(X,Y numéricas)              ← Processamento predefinido
Y=X
SAÍDA
```

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** ➔ 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

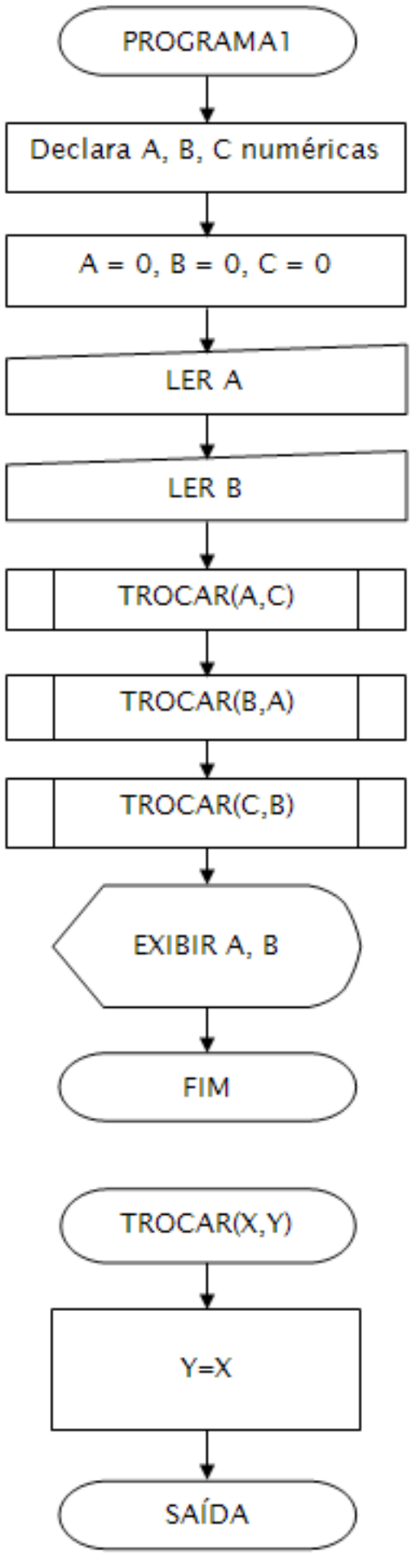
**B**: 0, 2 e 1

**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.

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



Teste seus conhecimentos

# Processamento predefinido

7

## 1. Sobre o processamento predefinido podemos afirmar:

- ☐ a) A função retorna um valor após ser executada, enquanto o procedimento, ou procedure, como também é chamado, não retornará nenhum valor.
- ☐ b) É um programa que pode ser usado em outro programa.
- ☐ c) No contexto de linguagem de programação, uma sub-rotina, uma função, um subprograma e um procedimento são todos processamentos predefinidos.
- ☐ d) Não é uma vantagem do processamento predefinido a diminuição de linhas de código do programa principal.
- ☐ e) As respostas B e C estão corretas.

**2. Dado o processamento predefinido:**

```
SOMAR (X, Y, Z NUMÉRICAS)
   Z = X + Y
SAÍDA
```

**Faça o teste de mesa para o programa a seguir:**

```
PROGRAMA1
INÍCIO
   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.

**3. 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
INÍCIO
  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, C e D 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.

4. Dados 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
         AUX = X
         X = Y
         Y = AUX
   FIM SE
SAÍDA
```

Faça o teste de mesa para o programa a seguir:

```
PROGRAMA3
DECLARA A, B, C, D NUMÉRICAS
INÍCIO
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, C e D 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.

**5. Dados os seguintes processamentos predefinidos:**

```
SOMAR (X, Y, Z NUMÉRICAS)
      Z = X + Y
SAÍDA

e

MEDIA (M, N, P NUMÉRICAS)
      M = (N + P) / 2
SAÍDA
```

**Faça o teste de mesa para o programa a seguir:**

```
 PROGRAMA4
INÍCIO
      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.

**6. Dados 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
INÍCIO
        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:**

- ☐ a) 7, 5, 12
- ☐ b) 5, 7, 6
- ☐ c) 5, 4, 6
- ☐ d) 12, 7, 9
- ☐ e) Nenhuma das alternativas anteriores está correta.

Mãos à obra!

7

# Processamento predefinido

# Exercício 1

Em um programa que verifica se um **CÓDIGO DE CARGO** consta de uma 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, VALIDAÇÃO NUMÉRICAS
        VALIDAÇÃO = 0
        EXIBIR "DIGITE O CÓDIGO DE UM CARGO"
        LER CODCARGO
        CHECACODIGO(CODCARGO, VALIDAÇÃO)
        SE VALIDAÇÃO < > 0
                BUSCAR A DESCRIÇÃO DO CARGO NA TABELA DE CARGOS
                EXIBIR A DESCRIÇÃO DO CARGO
          FIM SE
FIM
```

# Banco de dados

8

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

# 8.1.Introdução

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

# 8.2.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**

Coluna = Campo

| ID | NOME | ENDEREÇO | TELEFONE | UF |
|---|---|---|---|---|
| 01 | André | Rua Presidente, 07 | 99-99999-9199 | SP |
| 02 | Rute | Al. das Árvores, 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 |

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.

## 8.2.1. 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;

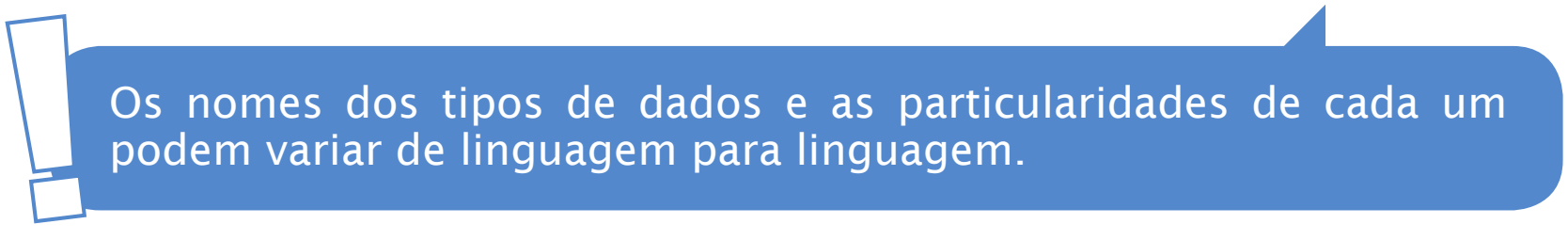
- Os tipos de operações que desejamos efetuar com os valores do campo. Por exemplo, não é possível somar valores em campo com tipo de dados **TEXTO**, mas em campos do tipo **NÚMERO** é possível;

- 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**.

## 8.2.2. 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).



# 8.3.Modelo de dados

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

## 8.3.1.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**.

### 8.3.1.1. 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.

### 8.3.1.2. 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**.

## 8.3.2.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.

### 8.3.3. Índice

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

### 8.3.4. 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.

### 8.3.5. Texto de validação

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

## 8.4. Criação de tabelas

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

**PLANEJAMENTO**

Banco de Dados: **DBAGENDA**

Tabela **TABAGENDA**
Campo chave primária = **CODAGENDA**

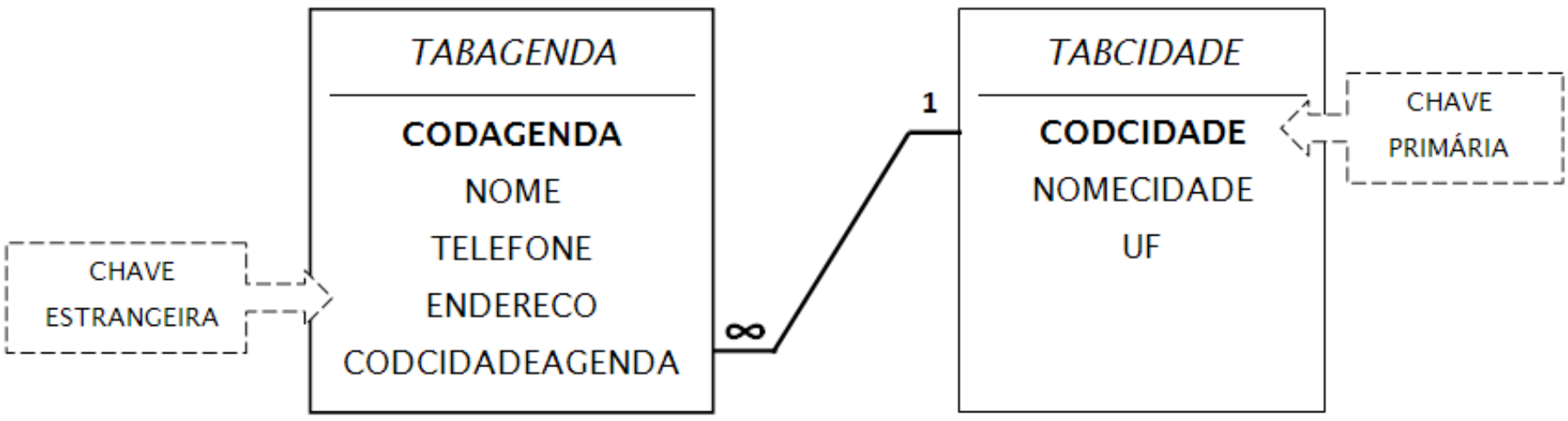
| DESCRIÇÃO | CAMPO | TIPO | TAMANHO |
|---|---|---|---|
| Código | CODAGENDA | NÚMERO | 5 |
| Nome do contato | NOME | TEXTO | 35 |
| Telefone | TELEFONE | TEXTO | 20 |
| Endereço | ENDERECO | TEXTO | 40 |
| Código da cidade | CODCIDADEAGENDA | NÚMERO | 4 |

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

| DESCRIÇÃO | CAMPO | TIPO | TAMANHO |
|---|---|---|---|
| Código da Cidade | CODCIDADE | NÚMERO | 4 |
| Nome da Cidade | NOMECIDADE | TEXTO | 40 |
| UF | UF | TEXTO | 2 |

# 8.5.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.

# 8.6.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:

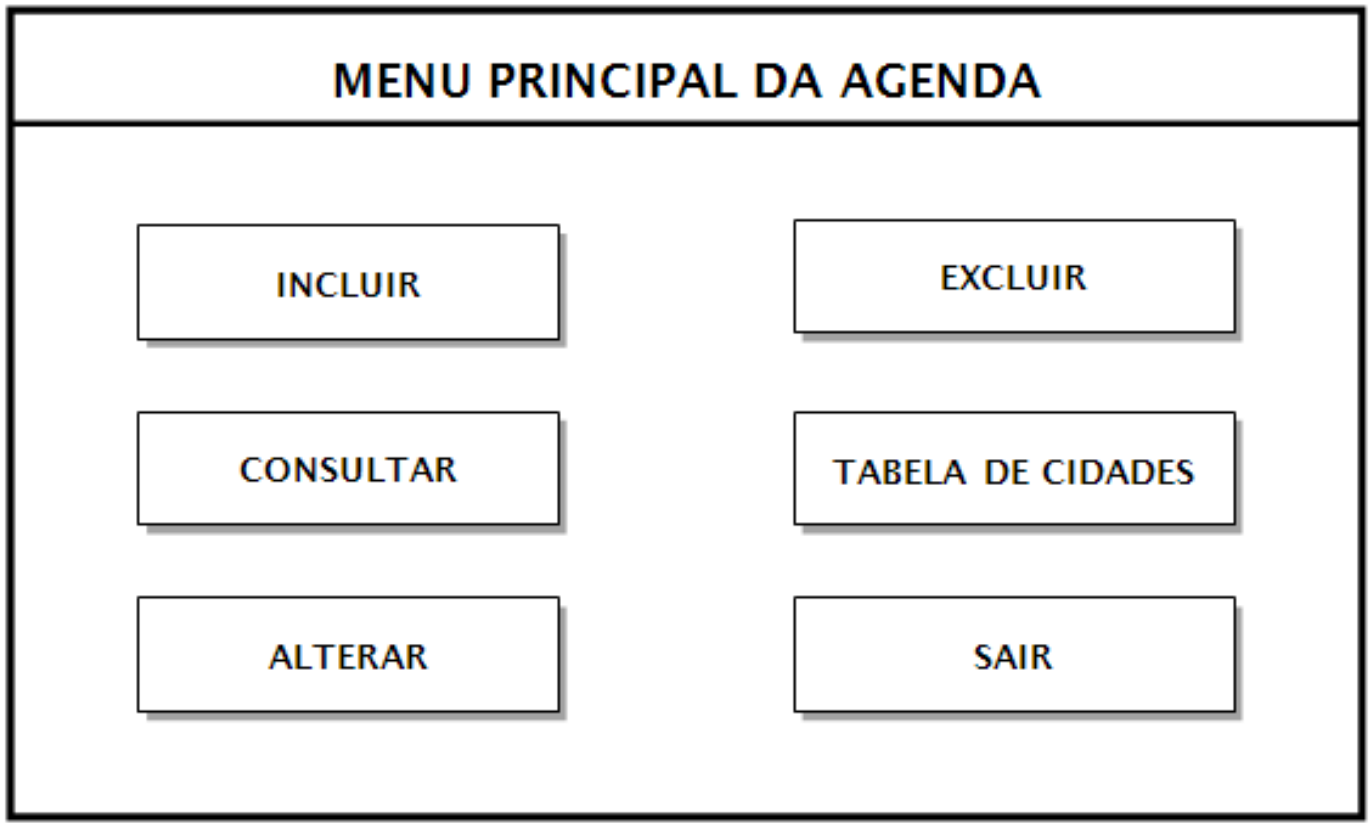
Tabela **TABCIDADE** (Tabela de Cidades)
Campo chave primária = **CODCIDADE**

| 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.

# 8.7.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 EXCLUSÂO 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.

## 8.7.1. 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**
- **VARENDERECO = Endereço do contato**
- **VARCODCID = Código da cidade**

- **Algoritmo**

1. INCLUSÃO
2. Declara VARCODAGENDA, VARCODCID numéricas e VARNOME, VARTELEFONE, VARENDERECO alfanuméricas
3. VARCODAGENDA = 0
4. VARNOME = “ ”
5. VARTELEFONE = “ “
6. VARENDERECO = “ “
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 VARNOME, VARTELEFONE, VARENDERECO, 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 = VARNOME
19. TELEFONE = VARTELEFONE
20. ENDERECO = VARENDERECO
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
```

## 8.7.2. 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
- **VARCODCID** = Código da cidade
- **VARNOMECIDADE** = Nome da cidade

- **Algoritmo**

1. CONSULTA
2. Declara VARCODAGENDA numérica e VARNOME, VARTELEFONE, VARENDERECO, VARNOMECIDADE alfanuméricas
3. VARCODAGENDA = 0
4. VARNOME = “ ”
5. VARTELEFONE = “ “
6. VARENDERECO = “ “
7. VARNOMECIDADE = “ “
8. Abrir tabelas TABAGENDA e TABCIDADE
9. Abrir tela de consulta de registros na tabela da TABAGENDA
10. Ler VARNOME
11. Ordenar a tabela TABAGENDA pelo campo NOME
12. Buscar na tabela TABAGENDA no campo NOME = VARNOME
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. VARNOMECIDADE = NOMECIDADE
19. Exibir na tela as variáveis VARCODAGENDA, VARNOME, VARTELEFONE, VARENDERECO e VARNOMECIDADE
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

## 8.7.3. 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
- **VARCODCID** = Código da cidade
- **VARNOMECIDADE** = Nome da cidade

- **Algoritmo**

1. ALTERAÇÃO
2. Declara VARCODAGENDA, VARCODCID numéricas e VARNOME, VARTELEFONE, VARENDERECO, VARNOMECIDADE alfanuméricas
3. VARCODAGENDA = 0
4. VARNOME = “ ”
5. VARTELEFONE = “ “
6. VARENDERECO = “ “
7. VARCODCID = 0
8. VARNOMECIDADE = “ “
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. VARNOME = NOME
17. VARTELEFONE = TELEFONE
18. VARENDERECO = ENDERECO
19. VARCODCID = CODCIDADEAGENDA
20. VARNOMECIDADE = NOMECIDADE
21. Exibir na tela as variáveis VARCODAGENDA, VARNOME, VARTELEFONE, VARENDERECO, VARCODCID e VARNOMECIDADE
22. Desproteger permitindo alteração as variáveis VARNOME, VARTELEFONE, VARENDERECO e VARCODCID
23. *** Comentário: Fazer os campos receberem o conteúdo das variáveis e depois gravar o registro
24. CODAGENDA = VARCODAGENDA
25. NOME = VARNOME
26. TELEFONE = VARTELEFONE
27. ENDERECO = VARENDERECO
28. CODCIDADEAGENDA = VARCODCID
29. Gravar o registro na tabela TABAGENDA
30. Fechar tela de alteração de registros da TABAGENDA
31. Fechar tabelas TABAGENDA e TABCIDADE
32. SAÍDA

## 8.7.4. 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
- **VARCODCID** = Código da cidade
- **VARNOMECIDADE** = Nome da cidade

- **Algoritmo**

1. EXCLUSÃO
2. Declara VARCODAGENDA, VARCODCID numéricas e VARNOME, VARTELEFONE, VARENDERECO, VARNOMECIDADE alfanuméricas
3. VARCODAGENDA = 0
4. VARNOME = “ ”
5. VARTELEFONE = “ “
6. VARENDERECO = “ “
7. VARCODCID = 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. VARNOME = NOME

17. VARTELEFONE = TELEFONE
18. VARENDERECO = ENDERECO
19. VARCODCID = CODCIDADEAGENDA
20. VARNOMECIDADE = NOMECIDADE
21. Exibir na tela as variáveis VARCODAGENDA, VARNOME, VARTELEFONE, VARENDERECO, VARCODCID 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

## 8.7.5. 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 como 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.

Teste seus conhecimentos

# Banco de dados

8

**1. Qual das opções a seguir não é considerada um banco de dados?**

- ☐ a) Agenda de contatos
- ☐ b) Porta-arquivo
- ☐ c) Livro
- ☐ d) Lista telefônica
- ☐ e) Nenhuma das alternativas anteriores está correta.

**2. Utilizando os conceitos básicos de banco de dados relacional, classifique os itens da lista a seguir como Registro (R) ou Campo (C):**

**1) Nome**
**2) Eletricista**
**3) 1322-8577**
**4) Telefone**
**5) Tomás**
**6) Profissão**
**7) Endereço**
**8) Salário**

- ☐ a) C, C, R, R, R, C, C, R
- ☐ b) C, R, R, C, R, C, C, C
- ☐ c) C, C, R, C, R, R, C, C
- ☐ d) C, R, R, C, R, C, C, C
- ☐ e) Nenhuma das alternativas anteriores está correta.

**3. Dadas as afirmações a seguir, marque Verdadeiro (V) ou Falso (F):**

**1) Para gerenciar um banco de dados, os programadores utilizam uma ferramenta chamada de sistema gerenciador de banco de dados, que é um grupo de programas responsáveis por controlar tudo o que se refere a um banco de dados.**
**2) Quando possuímos uma grande quantidade de dados, como é o caso de muitas empresas, as planilhas eletrônicas são o bastante para gerenciá-los.**
**3) É possível manter os dados organizados em diversas tabelas, cada uma representando entidades diferentes.**
**4) O cliente é o computador que fornece um determinado serviço.**

- ☐ a) V, F, V, F
- ☐ b) V, F, V, V
- ☐ c) F, F, V, V
- ☐ d) F, F, V, F
- ☐ e) Nenhuma das alternativas

**4. O modelo Entidade-Relacionamento é representado por quais itens?**

- ☐ a) Coluna, Linha e Tabela
- ☐ b) Entidades, Relacionamentos e Atributos
- ☐ c) Registro, Campo e Atributos
- ☐ d) Funcionário, Departamento e Dependente
- ☐ e) Nenhuma das alternativas anteriores está correta.

**5. Qual o nome do atributo da entidade que nunca será repetido e é utilizado como um identificador nas tabelas?**

- ☐ a) Atributo
- ☐ b) Campo primário
- ☐ c) Chave primária
- ☐ d) Registro
- ☐ e) Nenhuma das alternativas anteriores está correta.

Apêndice 1

# Linguagens de programação e exemplos de programas

Criação de exemplos:

Bruno Bove Barbosa, João Henrique Cunha Rangel e Ricardo Azzi Silva

# 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

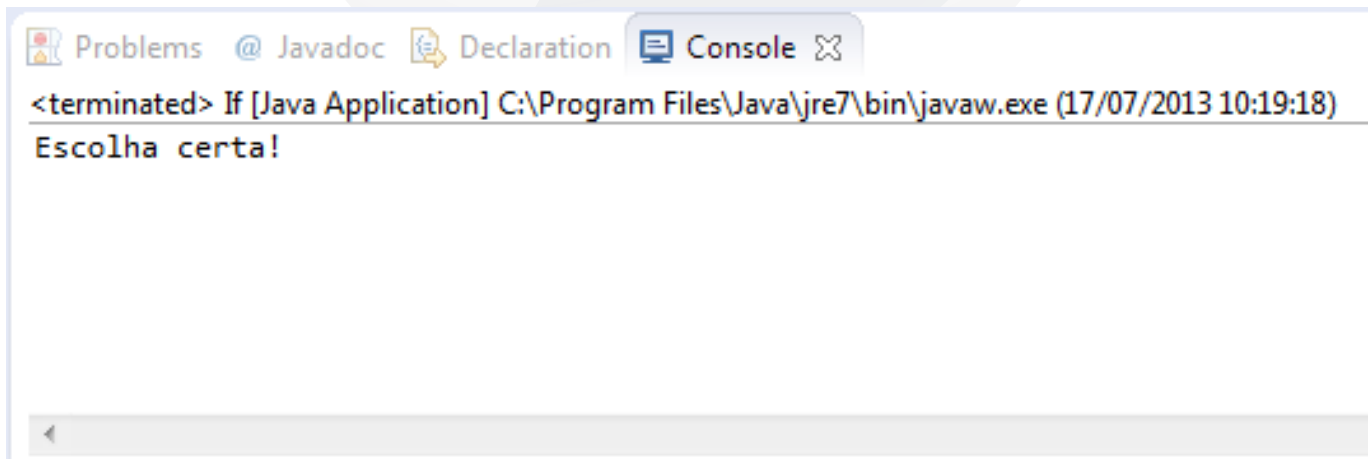
A seguir, temos um exemplo de declaração de variáveis em um programa em Java. Note que há linhas que se iniciam com barras duplas (//), e linhas entre /** e */. Utilizados para comentar as linhas de código, esses caracteres são inseridos pelo programador para facilitar o entendimento do código. Os comentários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

```
/** 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 = 922337203685477580 7L;
      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);

      }

}
```

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 válidar 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!");
          }

          }

  }
```

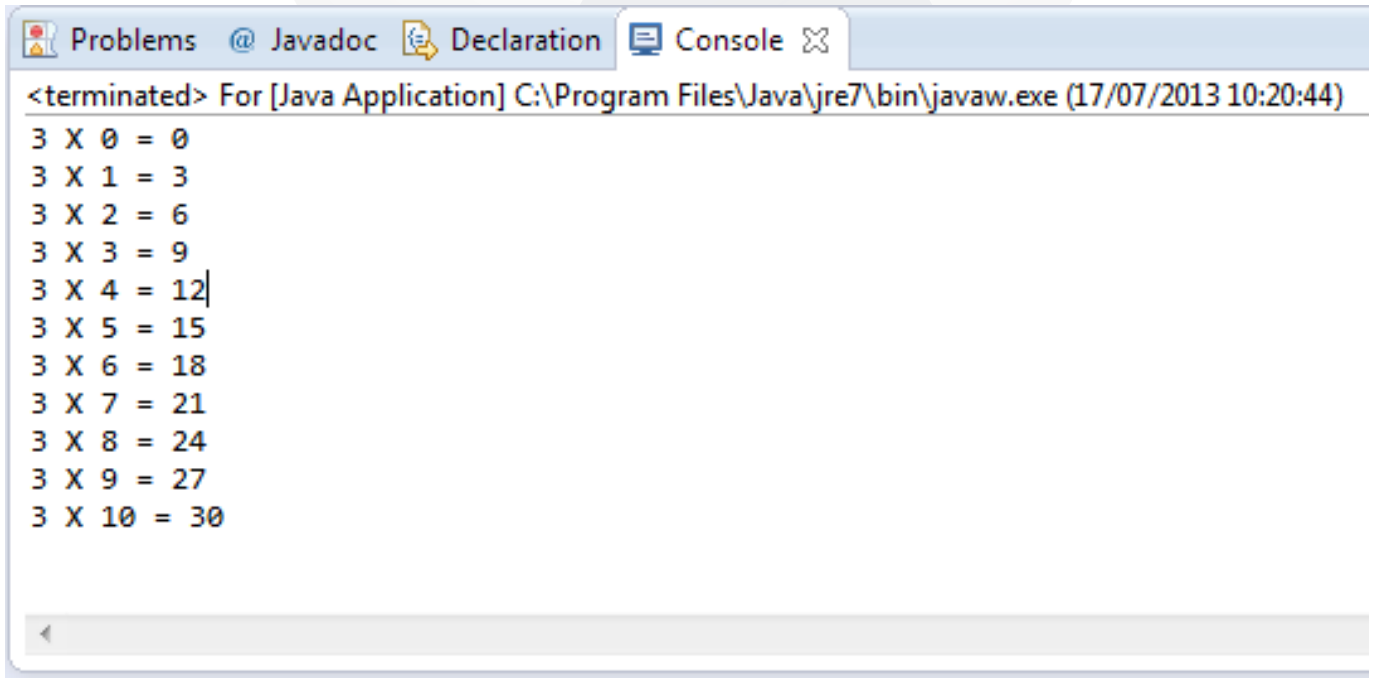
O resultado do código anterior é o seguinte:



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:



# 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
```

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 logicos. (=, <=. <, >, >=, !=)
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 difer-
ente 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
```

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.

## 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 variavel
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 variavel por padrão é do tipo e valor undefined
var minhaVariaval, 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 ausencia 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#.

```
//    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 ExemploDeLogica1
{
    class Program
    {
        //  TODO programa em C# é iniciado pela função Main()
        static void Main(string[] args)
        {
            //  A declaração de uma variavel no C# é feita definindo o tipo de
variavel e um nome:
            bool verdadeiro_ou_falso; // bool -> 1 bit

            //  Você pode definir um valor para a variavel 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 e uma frase
            char mander = 'A';
            string mensagem = "Olá mundo!";
        }
    }
}
```

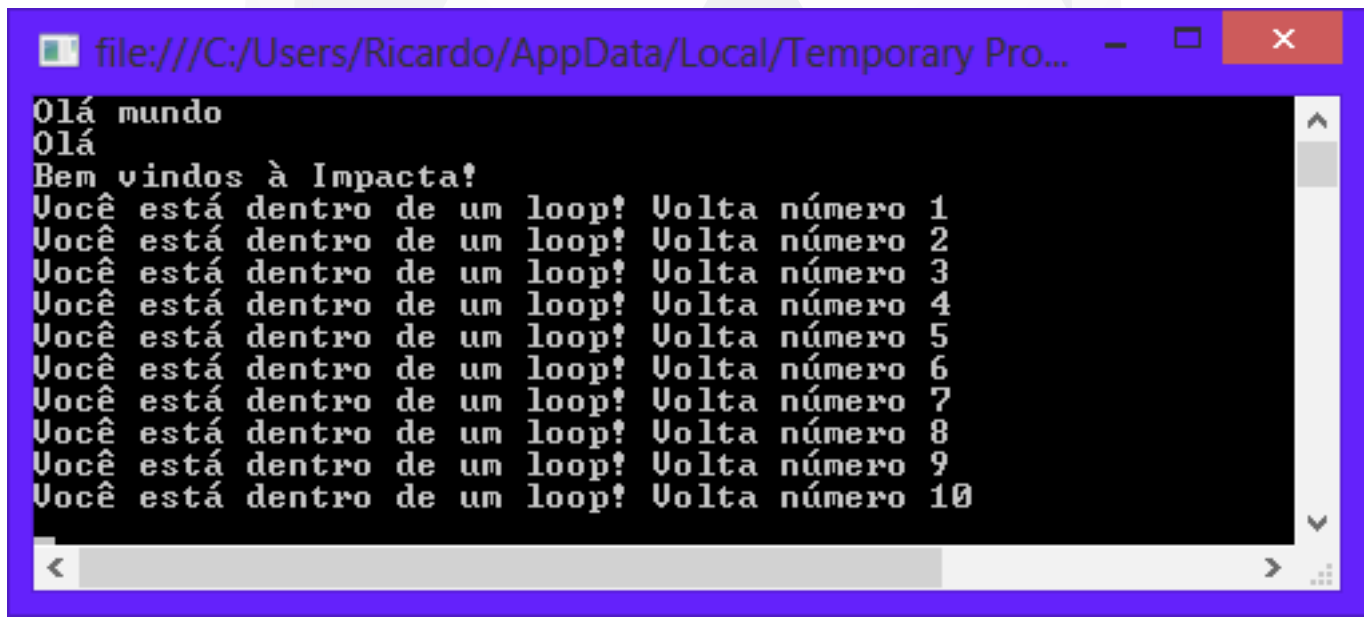
Este outro exemplo mostra o uso de **IF** e loop em C#:

```
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 variaveis 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 concluido:
            for (int i = 1; i <= 10; i++)
            {
                //  A concatenação em C# é feita com o sinal + e só é possível
com strings, se você deseja
                //  concatenar um valor númerico com uma string, deve con-
verte-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 concluidas com um
Console.ReadKey(),
            //  dessa forma o programa só irá encerrar se o usuário precionar
alguma tecla,
            //  isso dará tempo à ele de ler o que foi escrito na tela
            Console.ReadKey();
        }
    }
}
```

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.

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
      $meubooleano = 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 maíusculas 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
<?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).

```
Option Explicit
Sub CalculaMedia01()

'Versão 1, com cada nota sendo lida e verificada individualmente
'A 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
```

```
If Msg <> “” Then
    ‘O conteúdo das variáveis MSG e TIT são apresentadas em tela
    ‘juntamente com o endereço da célula onde o erro foi encontrado
    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

  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 "

'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 & vbLf & "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 processamaneto é encerrado
        Exit Sub
      End If
  Next
```

```
  'Não foram encontrados erros, a natureza das variáveis é convertida, 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 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

'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 verificar os vári-
os
'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

  Do While True
    'A variável NOTA1 é preenchida por digitação do operador, na caixa de en-
trada
    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 anteriores
  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

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
```

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 dimen-
são
'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 condição for ver-
dadeira
    '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 automaticamente
  'passará para a linha de comando após a clausula 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
```

Apêndice 2

# Sistemas de numeração

# 2.1.Bit e byte

**BIT** é a contração de **BI**nary digi**T**, 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

*2.2.*

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 ($2^9 = 512$).

# 2.3.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.

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 |

## 2.3.1.Sistema Decimal

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

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

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.

### 2.3.2. 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.

### 2.3.3. 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.

## 2.4. 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.

### 2.4.1. Conversão de Binário para Decimal

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

Vejamos um exemplo:

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

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:

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

## 2.4.2. Conversão de Hexadecimal para Decimal

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

Vejamos um exemplo:

$400_{(16)} = 4*(16)^2 + 0*(16)^1 + 0*(16)^0$
$= 4*256 + 0*16 + 0*1$
$= 1024 + 0 + 0$
$= 1024_{(10)}$

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:

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

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.

## 2.4.3. 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:

$0011\ \ 1100\ \ 1111_{(2)}$
$\ \ 3\ \ \ \ C\ \ \ \ F_{(16)}$ ← utilizando a tabela de conversão de sistemas de numeração

## 2.4.4. 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 $0010_{(2)}$

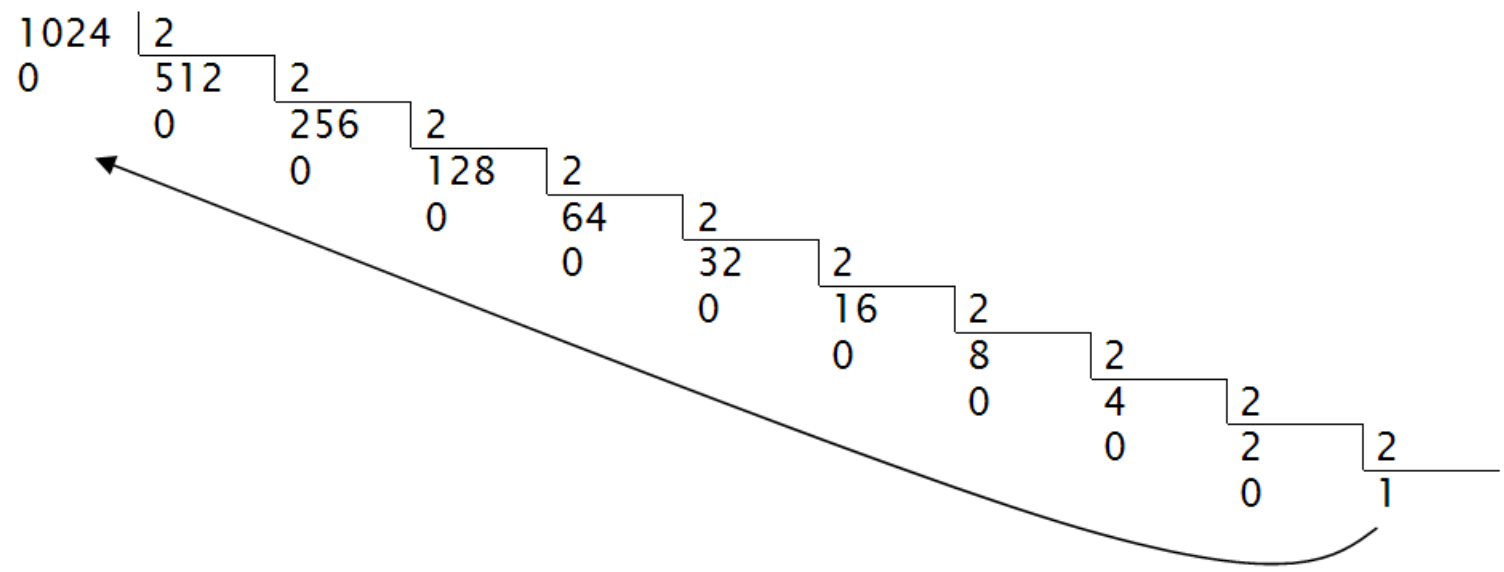
| Decimal | Binário | Hexadecimal |
|---|---|---|
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110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

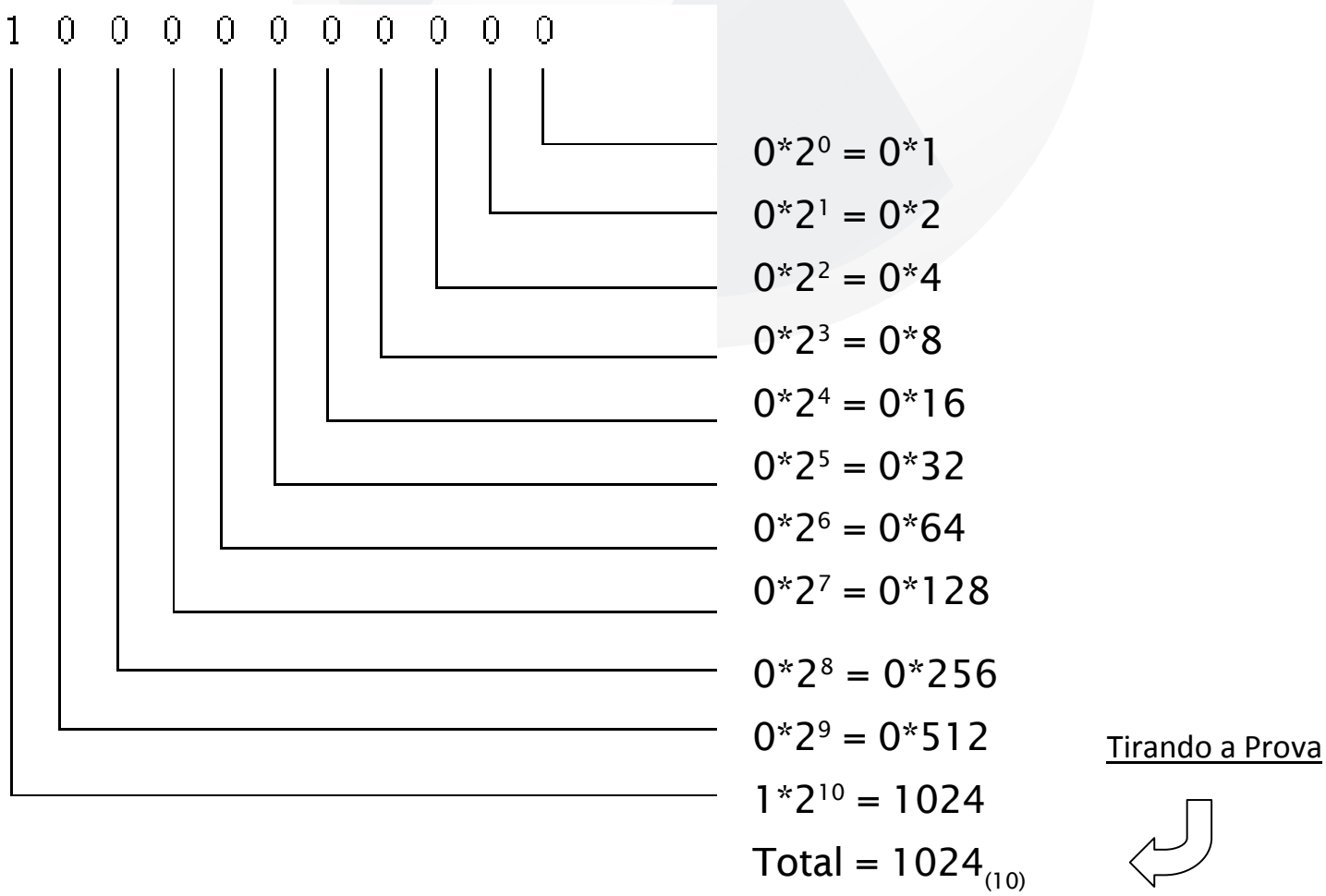
## 2.4.5. Conversão de Decimal para Binário

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**

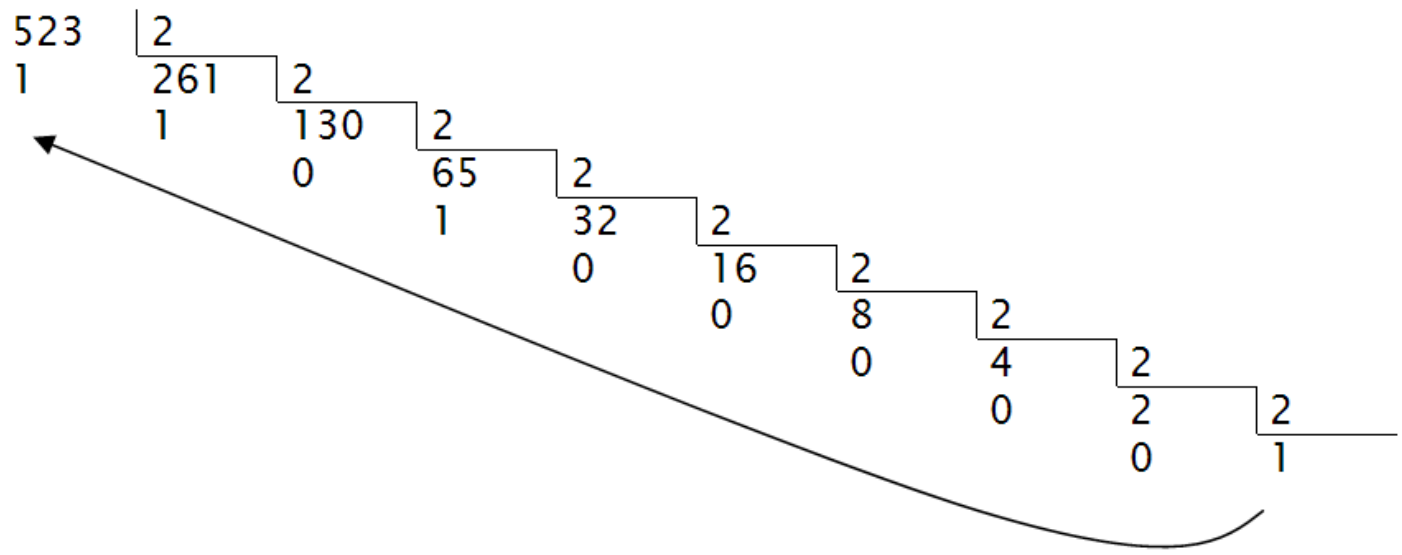
$1.024_{(10)} = 100\ 0000\ 0000_{(2)}$



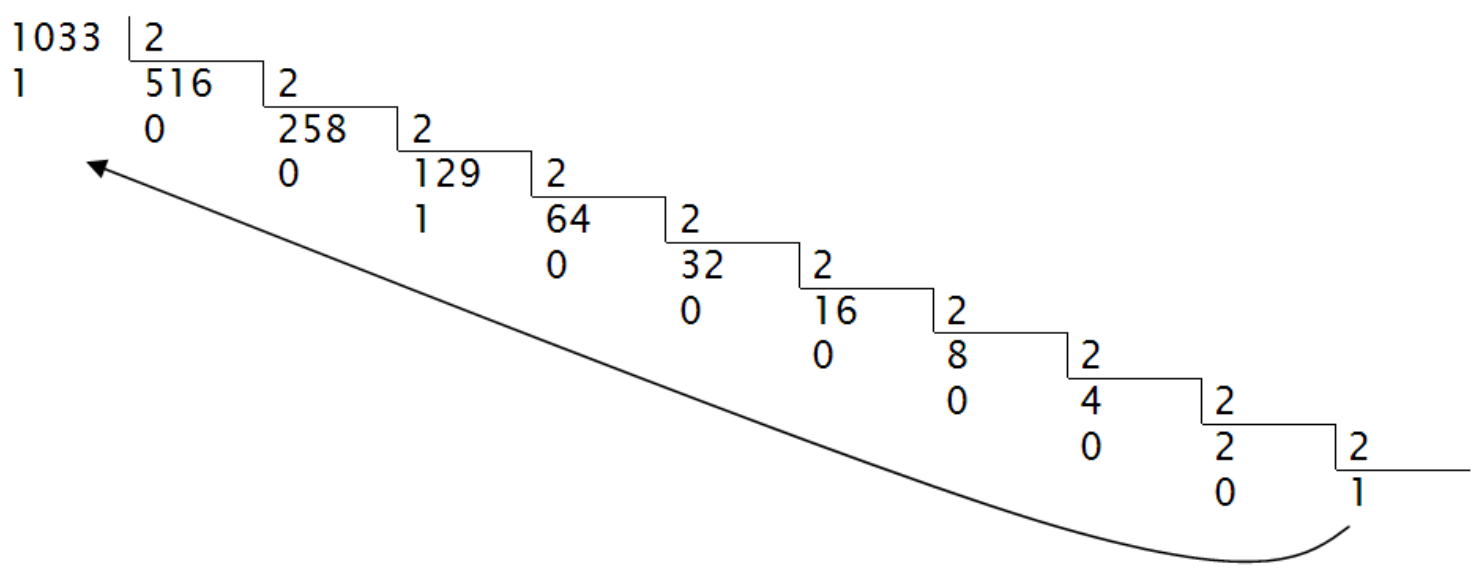
- **Exemplo 2**

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




- **Exemplo 3**

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

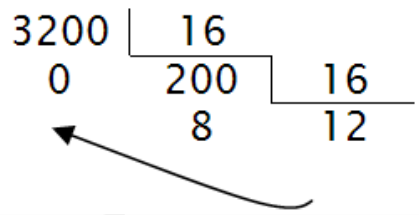


## 2.4.6. Conversão de Decimal para Hexadecimal

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.

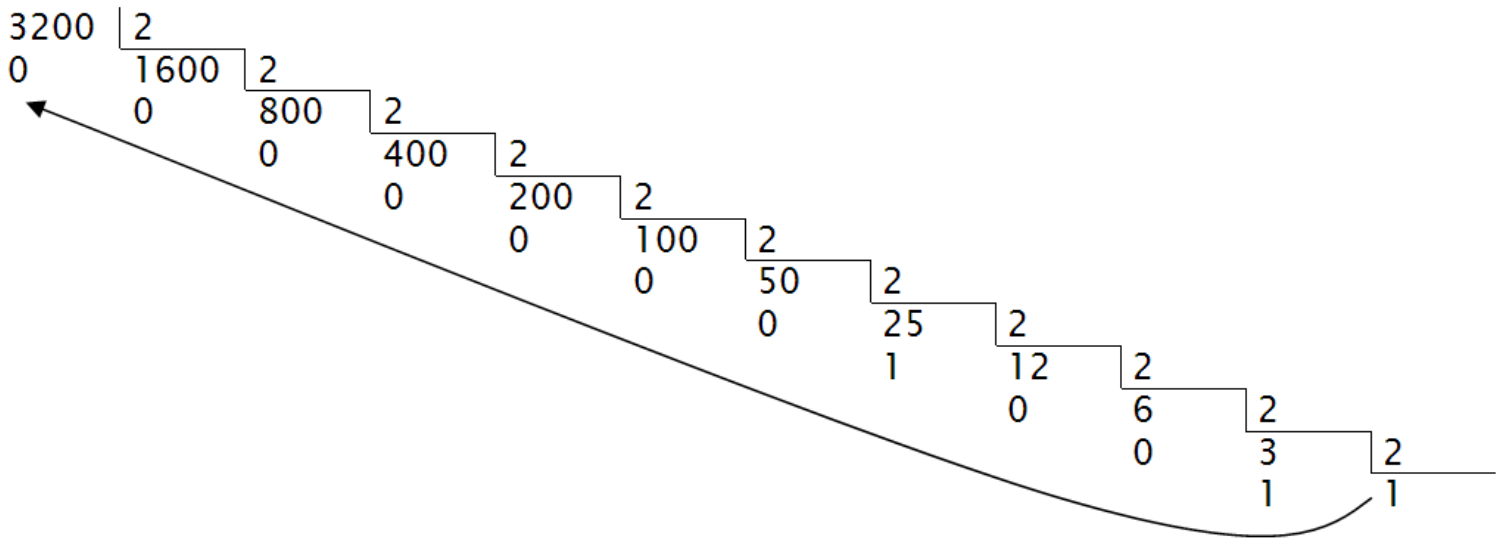
- **Exemplo**

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




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.




$1100\ \ 1000\ \ 0000_{(2)}$

$C \quad 8 \quad 0_{(16)}$

# 2.5.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.

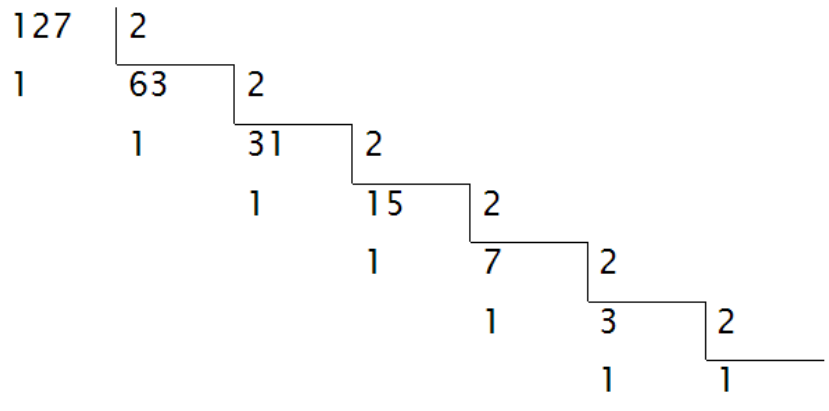
A forma rápida de conversão consiste em escrever numa linha os números da direita para a esquerda a partir do número 1 e, depois, sempre o dobro, ou seja, são as potências de 2.

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

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

- **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 + 4

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

- **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 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

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

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 |
|---|---|
| $2^0 = 1$ | $16^0 = 1$ |
| $2^1 = 2$ | |
| $2^2 = 4$ | |
| $2^3 = 8$ | |
| $2^4 = 16$ | $16^1 = 16$ |
| $2^5 = 32$ | |
| $2^6 = 64$ | |
| $2^7 = 128$ | |
| $2^8 = 256$ | $16^2 = 256$ |
| $2^9 = 512$ | |
| $2^{10} = 1024$ | |
| $2^{11} = 2048$ | |
| $2^{12} = 4096$ | $16^3 = 4096$ |
| $2^{13} = 8192$ | |
| $2^{14} = 16\ 384$ | |
| $2^{15} = 32.768$ | |
| $2^{16} = 65.536$ | $16^4 = 65.536$ |
| $2^{17} = 131.072$ | |
| $2^{18} = 262.144$ | |
| $2^{19} = 524.288$ | |
| $2^{20} = 1.048.576$ | $16^5 = 1.048.576$ |