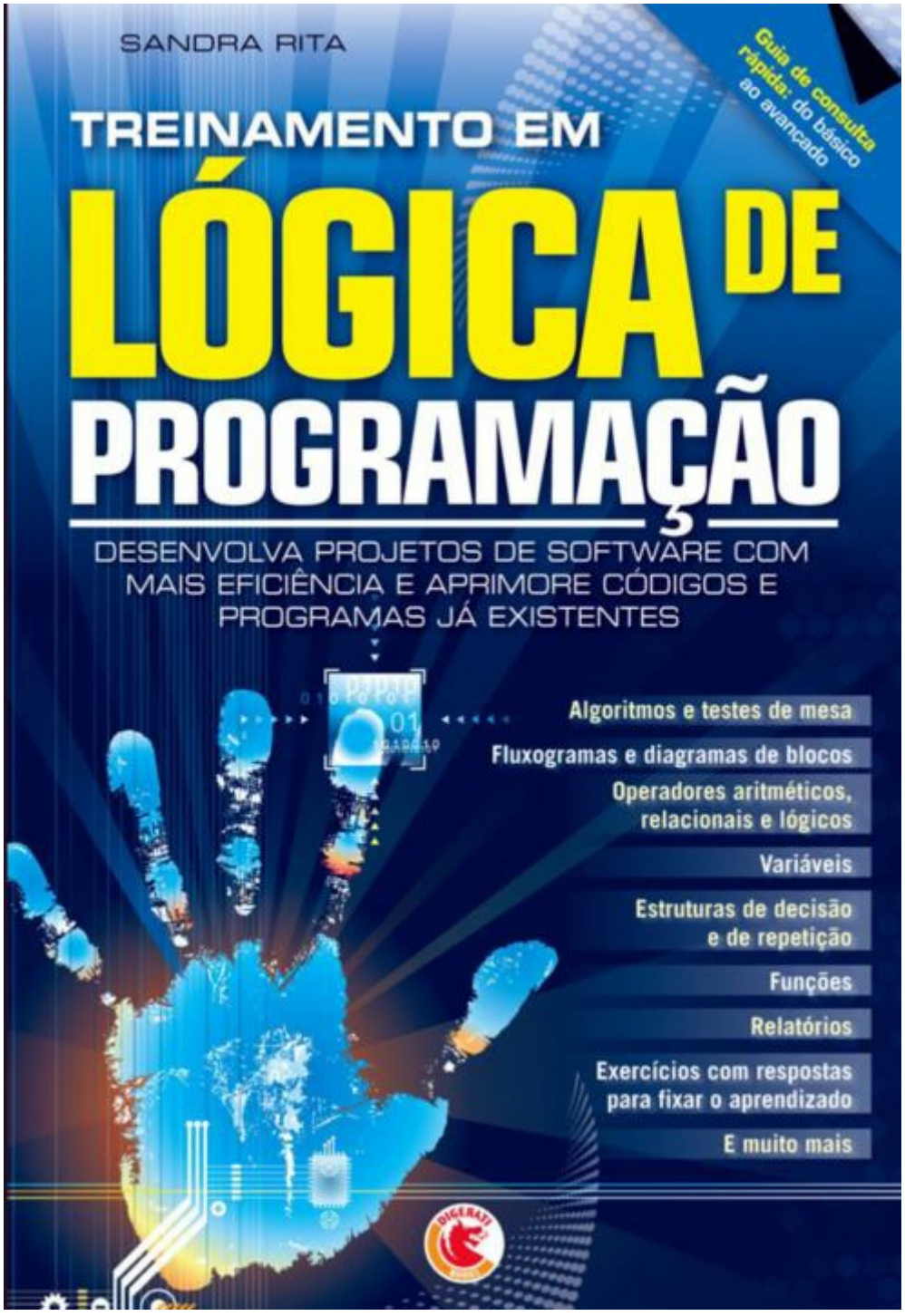
SANDRA RITA

Guia de consulta rápida: do básico ao avançado

TREINAMENTO EM LÓGICA DE PROGRAMAÇÃO

DESENVOLVA PROJETOS DE SOFTWARE COM MAIS EFICIÊNCIA E APRIMORE CÓDIGOS E PROGRAMAS JÁ EXISTENTES

- Algoritmos e testes de mesa
- Fluxogramas e diagramas de blocos
- Operadores aritméticos, relacionais e lógicos
- Variáveis
- Estruturas de decisão e de repetição
- Funções
- Relatórios
- Exercícios com respostas para fixar o aprendizado
- E muito mais


SANDRA RITA
Guia de consulta rápida: do básico ao avançado
TREINAMENTO EM
LÓGICA DE PROGRAMAÇÃO
DESENVOLVA PROJETOS DE SOFTWARE COM MAIS EFICIÊNCIA E APRIMORE CÓDIGOS E PROGRAMAS JÁ EXISTENTES
Algoritmos e testes de mesa
Fluxogramas e diagramas de blocos
Operadores aritméticos, relacionais e lógicos
Variáveis
Estruturas de decisão e de repetição
Funções
Relatórios
Exercícios com respostas para fixar o aprendizado
E muito mais

SANDRA RITA

TREINAMENTO EM

LÓGICA DE PROGRAMAÇÃO

DESENVOLVA PROJETOS DE SOFTWARE COM MAIS EFICIÊNCIA
E APRIMORE CÓDIGOS E PROGRAMAS JÁ EXISTENTES

São Paulo – 2009

© 2009 by Digerati Books
Todos os direitos reservados e protegidos pela Lei 9.610 de 19/02/1998.
Nenhuma parte deste livro, sem autorização prévia por escrito da editora, poderá ser reproduzida ou transmitida sejam quais forem os meios empregados: eletrônicos, mecânicos, fotográficos, gravação ou quaisquer outros.

Diretor Editorial
Luis Matos

Projeto Gráfico
Daniele Fátima

Assistência Editorial
Regiane Monteiro
Renata Miyagusku

Diagramação
Cláudio Alves
Stephanie Lin

Revisão
Shirley Ayres

Capa
Marcos Mazzei

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

P659t Pinto, Sandra Rita.

Treinamento em Lógica de Programação / Sandra Rita Bento Pinto. – São Paulo: Digerati Books, 2009.
144 p.

ISBN 978-85-7873-084-0

1. Lógica de Programação. 2. Informática. I. Título.

CDD 005.1

Universo dos Livros Editora Ltda.
Rua Haddock Lobo, 347 – 12º andar – Cerqueira César
CEP 01414-001 • São Paulo/SP
Telefone: (11) 3217-2600 • Fax: (11) 3217-2616
www.universodoslivros.com.br
e-mail: editor@universodoslivros.com.br

Sumário

Apresentação 5

Capitulo 1 - Algoritmos 7

Algoritmos
..... 8

Representação da Lógica de Programação
..... 10

Teste de Mesa
..... 16

Capitulo 2 - Fluxogramas ou diagramas de blocos 17

Capitulo 3 - Operadores aritméticos, relacionais e lõgicos
..... 23

Expressões e Operadores
..... 24

Capitulo 4 - Variãveis, constantes, tipos de dados e funções .....
33

Variável de memória
.....35

Tipos de Dados
..... 39

Constantes

.....43

Funções
.....44

Capitulo 5 - Estruturas de decisão 47

Tipos de Estruturas de decisão
..... 50

Capitulo 6 - Estruturas de repetição (loop) 69

Estrutura Faça Enquanto
..... 71

Estrutura Faça... Enquanto <condição>
..... 77

Estrutura Para... Passo... Faça
..... 82

Capitulo 7 - Vetores e matrizes 87

Vetores
.....88

Matriz
.....97

Capitulo 8 - Estrutura de dados e sub rotinas
..... 103

Estrutura de dados
..... 104

Manipulando Registros
..... 106

Manipulando Arquivos
..... 108

Listas Lineares
.....112

Inclusão de elementos em uma lista
..... 113

Alterar um elemento da lista
..... 114

Exclusão de um elemento na lista
..... 116

Subrotinas
.....117

Capitulo 9 - Validação de dados e relatõrios
..... 123

Funções de manipulações de strings
..... 125

Funções de conversão de tipos
..... 126

Relatórios
..... 132

Apêndice 135

Exercícios Propostos
..... 136

Solução dos algoritmos propostos
..... 138

Apresentação

Em nosso dia a dia, nos deparamos com vários problemas que parecem não ter solução e empregamos horas e horas tentando encadear nossos pensamentos em busca de tal resolução. Desde as tarefas mais simples, como tomar um banho, até no controle de nossas finanças, um conjunto de métodos determinam nosso raciocínio de forma ordenada, nos induzindo à solução eficaz do que precisamos realizar.

A organização de nossas ideias em um conjunto de procedimentos a fim de realizarmos uma tarefa ou solucionarmos determinado problema é conhecida, na linguagem de informática, como lógica; empregamos tal conceito diariamente e de forma alguma nos damos conta disso.

Portanto, podemos definir lógica de programação com uma sequência de procedimentos (instruções) que se alinham a fim de atender determinado objetivo.

Utilizar a lógica de programação é um fator relevante a todos aqueles que desejam exercer atividades na área de TI, tais como programadores, analistas, equipe técnica, gerentes e outros. 0 importante é aperfeiçoar o uso de tais procedimentos de forma crítica e inteligente, independente da linguagem a ser utilizada.

Hoje em dia é muito comum, nos cursos de Ciência e Engenharia da Computação, Engenharia Elétrica, Tecnologia da Informática e também nos cursos de Jogos Digitais, o ensino obrigatório de Lógica de Programação.

Capítulo 1

Algoritmos

Algoritmos

Para a solução de qualquer problema devemos encontrar uma sequência lógica, como uma receita de bolo, de forma que os dados sejam processados e armazenados no computador. A esta sequência finita, dá-se o nome de algoritmo.

Um algoritmo é um conjunto de regras ou instruções definidas de forma clara e precisa, utilizado para resolver um problema especifico. Vale ressaltar que um algoritmo não é a solução de um problema, mas sim, um caminho para tal.

Antes de fazer um programa na linguagem desejada, o primeiro passo é construir um algoritmo e, em seguida, utilizar uma linguagem de programação com a sintaxe correta de tais instruções.

Várias são as linguagens de programação utilizadas no desenvolvimento de aplicações, tais como C, Delphi, Java, Visual Basic, entre outras. Para o desenvolvimento de sites, as linguagens mais conhecidas são HTML, PHP e ASP. Uma linguagem de programação deverá possuir um "tradutor" do códigofonte (instruções do programa) para a linguagem de máquina, que poderia ser um interpretador ou um compilador.

Independente da linguagem a ser utilizada, o objetivo do códigofonte não é ser executado diretamente pelo processador, mas, sim, permitir que o programador defina uma sequência legível de todas as etapas a serem realizadas pela máquina (computador).

Para que o códigofonte seja convertido em um programa executável, isto é, que realize as tarefas descritas, será necessário que ele seja compreendido pelo computador em uma linguagem de máquina, o que significa ser interpretado ou compilado.

Podemos dizer que os compiladores que surgiram na década de 1950 são programas que traduzem as instruções descritas pelo programador para uma linguagem de máquina, transformando no que também é conhecido como programa objeto. Como exemplo de tais compiladores temos os da linguagem Cobol, Fortran, Pascal e C.

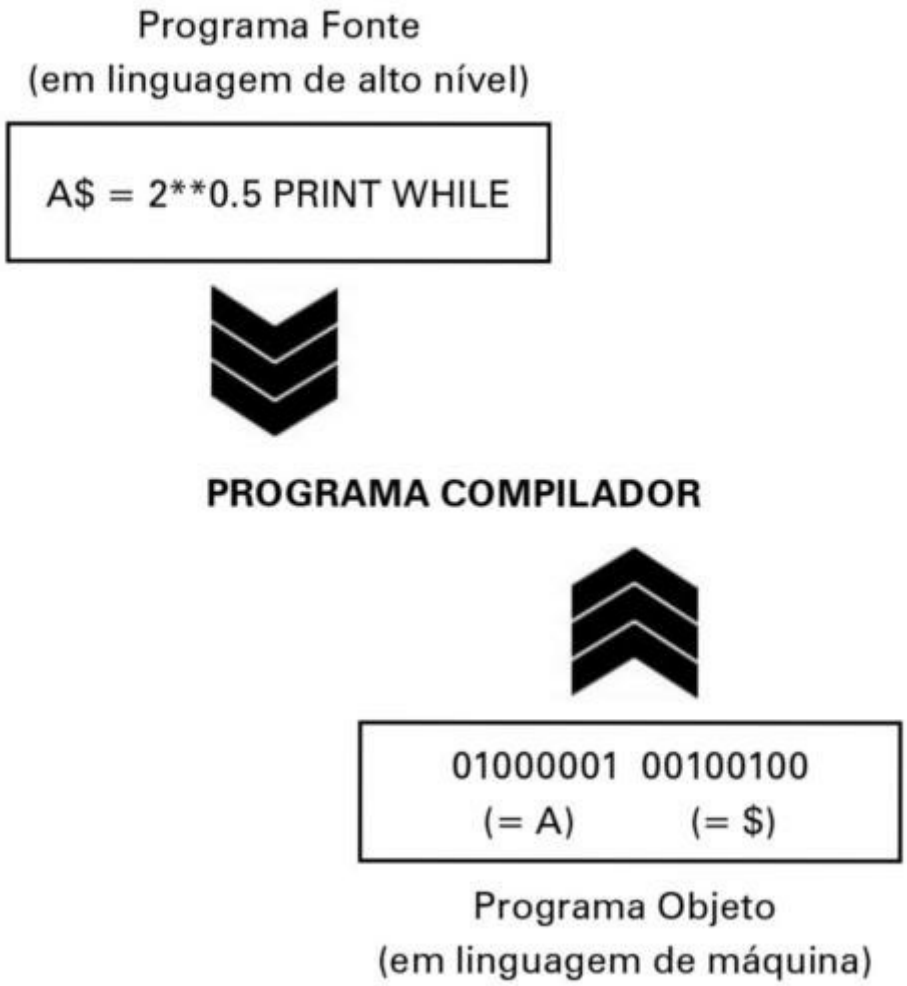


Figura 1.1.: 0 compilador é o intermediário entre o códigofonte e o objeto gerado.

Os interpretadores são programas que, além de traduzir para a linguagem de máquina (compiladores), também executam outros programas, escritos em linguagem de alto nível, linha a linha, e devem estar presentes na memória RAM do computador junto com o programa a ser utilizado. Alguns exemplos de linguagens interpretadas seriam o Basic e a linguagem SQL, entre outros.

No caso dos interpretadores, o códigofonte não é totalmente traduzido

antes de ser executado e não existe a etapa da geração de um código intermediário (código objeto), pois um interpretador lê e traduz o código de forma sequencial.

Como todos os programas, os compiladores e interpretadores, possuem vantagens e desvantagens que devem ser levadas em consideração antes de sua escolha e utilização. Veja uma comparação mais detalhada na tabela a seguir (Tabela 1.1):

Tipo	Vantagens	Desvantagens
Compiladores	• Execução mais veloz. • As estruturas de programação podem ser mais complexas. • Permite a otimização do código-fonte.	• Várias etapas de tradução. • Maior consumo de memória para a execução, uma vez que o programa final é maior. • O processo de depuração de cada linha é mais demorado.
Interpretadores	• Depuração mais simples. • Menor consumo de memória. • Resultado imediato da execução das instruções por ser passo a passo.	• A execução do programa é mais lenta. • A estrutura de dados é mais simples e limitada. • É necessário que a máquina possua o programa fonte.

Tabela 1.1.: Comparação entre compiladores e interpretadores.

Representação da Lógica de Programação

Diversas são as formas de representar a lógica de programação. A mais utilizada, com certeza, é o algoritmo, com uma sequência de instruções

finitas (procedimentos) que pode ser executada em determinado tempo com a finalidade de atingir um objetivo.

Um algoritmo pode ser representado de forma narrativa, como uma receita de bolo:

1. Separar os ingredientes solicitados na receita.

2. Pegar a tigela da batedeira.

3. Colocar a farinha.

4. Colocar os ovos.

5. Colocar a manteiga e o açúcar.

6. Colocar o leite.

7. Colocar a tigela na batedeira.

8. Ligar a batedeira e misturar muito bem.

9. Desligar a batedeira.

10. Despejar a massa em uma forma untada.

11. Levar ao forno pré-aquecido por 45 minutos.

É importante perceber que, na criação de um algoritmo, o problema sempre será dividido em três fases: Na Entrada, encontramos os dados necessários para a resolução de todo o problema, ou seja, na maioria das vezes é necessário solicitar informações (levantamento de dados) para tal solução. No caso da receita de bolo, a entrada seria os ingredientes, a tigela e a batedeira, isto é, do que precisamos para executar o processo.

No processamento encontramos os procedimentos (instruções) que foram utilizados para que o problema seja resolvido. Este conjunto de instruções sempre será finito, ou seja, terá um início e um fim. No exemplo da receita de bolo, seriam as ações de colocar os ingredientes na tigela, misturá-los na batedeira, colocar a mistura na forma untada e levá-la ao forno.

Na saída, teremos os dados processados, onde será apresentada a resolução do problema proposto. No exemplo que estamos utilizando, seria o bolo pronto.

Veja essas etapas, agora, por meio de outro exemplo: Problema: analisar as notas de português do primeiro (Notai) e segundo (Nota2) bimestre e calcular a média do semestre, identificando se o aluno foi aprovado (média maior ou igual a 6,0) ou reprovado (média inferior a 6,0).

Entrada: Notas Notal e Nota2.

Processamento: efetuar o cálculo da média aritmética, que será representada pela soma das duas notas (Notai + Nota2) dividido por 2. 0 resultado final deverá ser comparado com a nota 6,0.

Saída: aluno Aprovado ou Reprovado.

Veja outro exemplo: Problema: fazer um suco de laranja.

Entrada: laranjas Processamento: 1. Separar 4 laranjas; 2. Pegar uma faca; 3. Cortar as laranjas ao meio; 4. Passar cada metade da laranja pelo espremedor; 5. Retirar a jarra do espremedor.

Saída: suco de laranja pronto.

Até o momento, foi possível identificar que um algoritmo é apresentado em um determinado idioma, no caso, o português, para a descrição de suas instruções. Outra forma de apresentação é utilizar uma linguagem de programação. Veja a disposição das instruções na linguagem C:

```
#include <media.h>
  int main()

  {
    int a,b, med;
    a = 7; b = 8;
    med = (a + b) / 2;
    printf("media = %d\n", med);
  }
```

Outra forma de representação de um algoritmo é a utilização de gráficos, como o Diagrama de Nassi-Shneiderman ou Chapin, desenvolvido na década de 1950 e que apresenta uma visão estruturada e hierárquica de toda a lógica do programa, sendo composto por um único bloco (Figura 1.2):

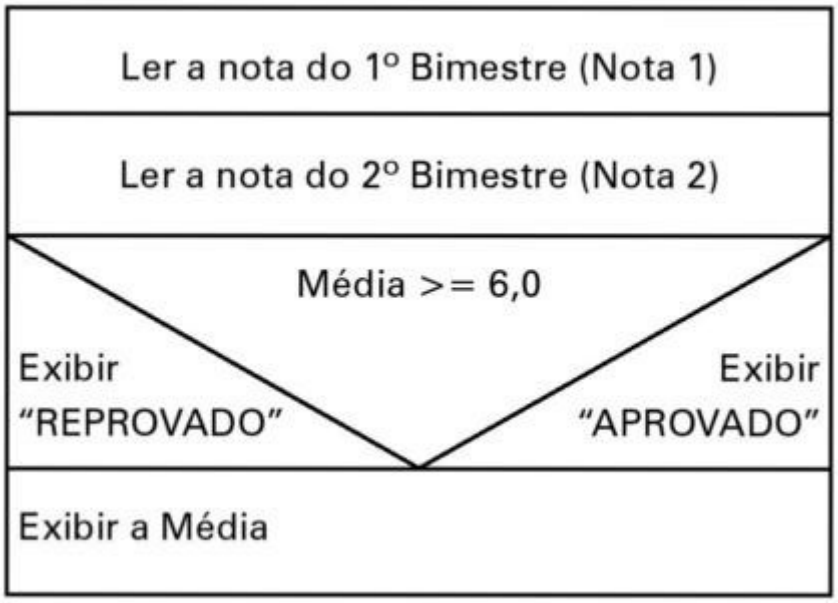


Figura 1.2.: Diagrama de Nassi-Shneiderman ou Chapin.

Ainda na forma de representação gráfica de algoritmos, temos os fluxogramas, ou diagramas de blocos (Figura 1.3), assunto abordado mais detalhadamente no próximo capítulo:

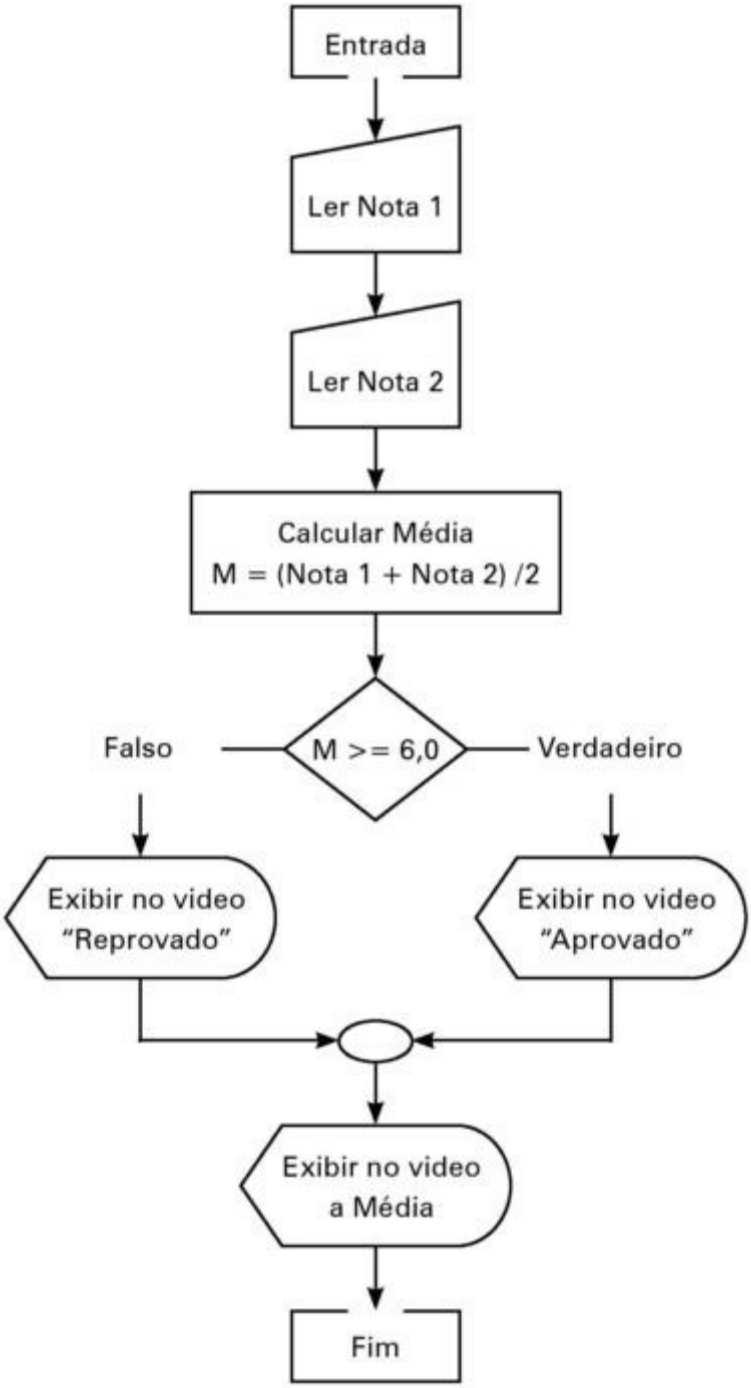


Figura 1.3.: Fluxograma ou Diagrama de Blocos.

Outra forma de representação de algoritmos é a que apresentaremos ao longo deste livro, conhecida como pseudo-código, sendo toda escrita em idioma português (também conhecido como português estruturado) e que, portanto, não apresentará ambiguidades e também não terá as normas (sintaxes) de uma linguagem de programação. 0 pseudo-código é escrito em forma de instruções construídas em frases que correspondem às

estruturas básicas de programação.

Para tal representação (pseudo-códigos), é importante seguir algumas regras: 1. 0 algoritmo sempre deve possuir um nome.

2. É importante descrever o objetivo do algoritmo.

3. Iniciar o algoritmo com a palavra INÍCIO.

4. Descrever a entrada de dados.

5. Para facilitar a leitura e compreensão, utilizar espaços de tabulações (identação) nos blocos de instruções.

6. Utilizar somente um verbo por instrução.

7. Utilizar frases (instruções) simples e curtas para facilitar o entendimento.

8. Enumerar cada uma das instruções.

9. Inserir comentários explicativos para as instruções, principalmente quando estiver programando.

10. Formatar em negrito as palavras que indicam ações ou condições a serem executadas ou seguidas, como por exemplo: Ler, Escrever, Se, Enquanto, Faça e outras que serão verificadas de forma mais detalhada ao longo deste livro.

11. Descrever a saída de dados.

12. Encerrar o algoritmo com a palavra FIM.

Além disso, a maioria dos programadores e autores especializados no assunto afirma que um algoritmo deve conter: • Finitude - todo algoritmo termina após um número finito de passos.

• Definição - cada um dos passos existentes no algoritmo deve ser definido com precisão.

• Entrada - todo algoritmo deve possuir zero ou mais entradas.

• Saída - todo algoritmo deve possuir uma ou mais saídas.

• Efetividade-as operações existentes em um algoritmo devem ser simples de forma a serem executadas em tempo limitado.

Agora confira tais regras aplicadas em um algoritmo:

Nome: Média
Objetivo: Calcular a média de um aluno e apresentar a mensagem "Aprovado" ou "Reprovado".
Entrada de Dados: Notas do 1° e 2° bimestre (*Nota1* e *Nota2*)
Saída de Dados: Exibir a média e resultado "Aprovado" ou "Reprovado".

```
Início
     1. Ler a nota do primeiro bimestre (Nota1)
     2. Ler a nota do segundo bimestre (Nota2)
     3. Calcular a média MÉDIA = (Nota1 + Nota2) / 2
     4. SE MEDIA > = 6,0
     ENTÃO
     4.1 Imprimir "Aprovado"
     SENÃO
     4.1 Imprimir "Reprovado"
     FIM SE
     5. Imprimir o valor da MEDIA
Fim
```

Teste de Mesa Após o desenvolvimento de um algoritmo, é necessário verificar cada um dos passos que foram determinados, ou seja, efetuar um teste. Para isso, leia cada uma das instruções e anote o resultado de cada tarefa/passo, verificando possíveis erros ou outras formas de solucionar o problema. Este teste é mais conhecido como Teste de Mesa.

Capítulo 2

Fluxogramas ou diagramas de blocos

Anteriormente, verificamos que uma das formas de representação de um algoritmo é feita por meio de fluxogramas, também conhecidos como diagramas de blocos.

Um fluxograma preocupa-se com todos os detalhes de implantação do algoritmo, utilizando figuras geométricas que representam as ações a serem executadas, a entrada e saída de dados, as unidades de armazenamento, além do sentido do fluxo dos dados e tomadas de decisões. Ou seja, um fluxograma é utilizado para escrever cada um dos passos existentes no algoritmo para a solução do problema.

Na maioria das vezes, o entendimento de tais figuras geométricas facilita a compreensão de todo o texto existente em uma instrução. Alguns programadores deixam de utilizar os fluxogramas por acharem que o algoritmo não apresenta tantos detalhes, resumindo-o a um único símbolo até mesmo por desconhecer a sua simbologia, o que irá dificultar a transcrição das tarefas para um programa .

Vejamos na Tabela 2.1 quais são as figuras geométricas usadas na simbologia dos fluxogramas e suas respectivas utilizações.

Símbolo	Nome	Descrição
	Terminal	Símbolo utilizado para indicar o início ou fim de um algoritmo ou módulo.
	Setas	Símbolo utilizado para indicar o sentido do fluxo de dados ou conexão com outros símbolos ou blocos existentes no algoritmo.
	Processamento	Símbolo utilizado para indicar o processamento de dados, ou seja, uma ação a ser executada, tais como cálculos, atribuição de valores e outros.
	Entrada	Símbolo utilizado para representar a entrada de dados.

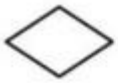
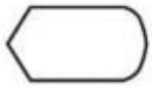
	Saída	Símbolo utilizado para representar a saída de dados como documento.
	Decisão	Símbolo utilizado na tomada de uma decisão, indicando a possibilidade de desvios no fluxo dos dados. Nesse caso, uma instrução é verificada e será comparada com uma condição e, a seguir, uma decisão poderá ser tomada caso os valores sejam verdadeiros ou falsos.
	Exibição	Símbolo utilizado para exibir os dados.
	Processamento predefinido	Símbolo utilizado para indicar a chamada a uma sub-rotina ou conjunto de instruções.
	Vários documentos	Símbolo utilizado para indicar o armazenamento ou leitura de dados em vários outros documentos.
	Conector de página	Símbolo utilizado para indicar a conexão ou continuação do fluxo em outra página (quebra de página em relatórios ou na exibição de dados na tela).
	Agrupar	Símbolo utilizado para indicar que os dados devem ser apresentados em forma de grupo, ou seja, agrupados por um nível.

Tabela 2.1.: Simbologia dos diagramas de fluxo de dados.

Há vários outros símbolos que acabaram caindo em desuso completo, pois o meio magnético deixou de ser utilizado, como por exemplo, as fitas e os cartões perfurados; portanto, os símbolos descritos anteriormente são os mais utilizados na criação de fluxogramas.

Vejamos um exemplo de fluxograma (Figura 2.1) que tem a finalidade

de exibir o resultado da multiplicação entre dois números:

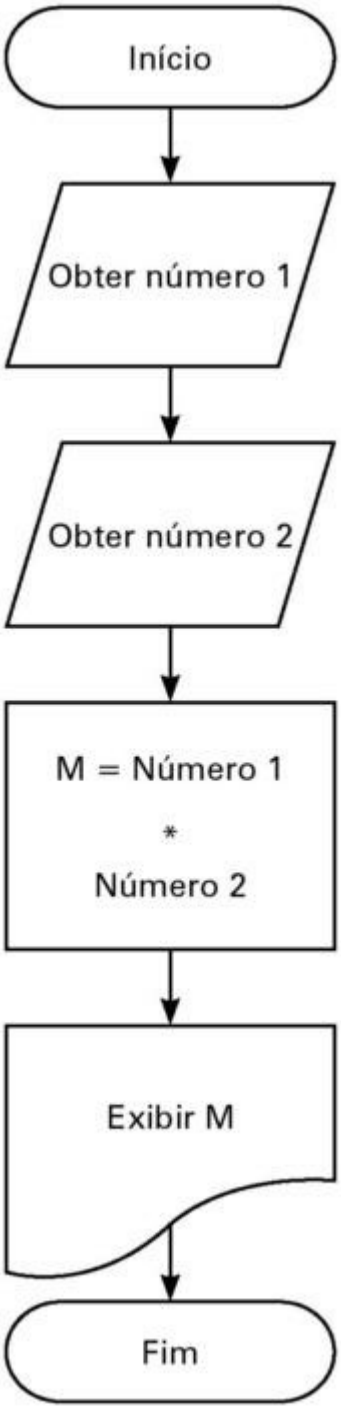


Figura 2.1.: Fluxograma exibindo uma rotina de multiplicação.

Há várias outras situações em que os fluxogramas são utilizados para a representação de situações ou processos, como, por exemplo, na descrição de responsabilidades de determinas tarefas de um setor ou de uma empresa (Figura 2.2).

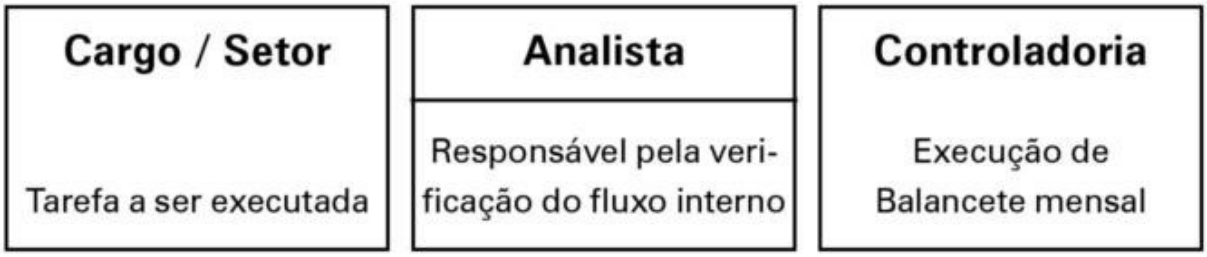


Figura 2.2.: Fluxograma exibindo descrições de tarefas.

Algumas empresas costumam adotar colunas para a disposição dos dados (Figura 2.3):

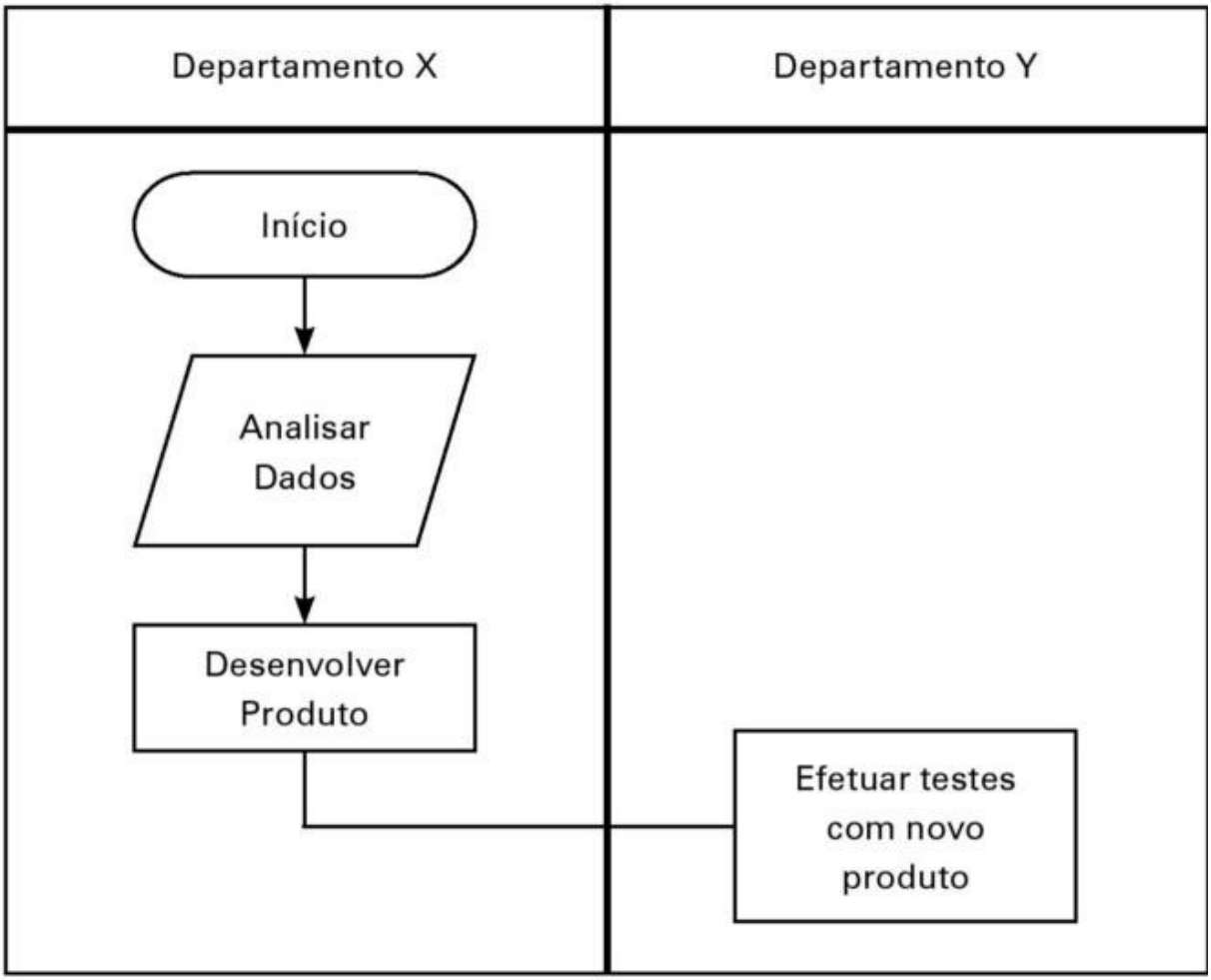


Figura 2.3.: Fluxograma exibindo dados que trafegam entre departamentos distintos.

Não importa a apresentação do fluxograma, pois alguns preferem, por questões de layout ou espaço, exibir os dados em caixas ou figuras na horizontal ou diagonal. 0 importante é perceber que a representação em

forma de imagens acaba facilitando a compreensão do fluxo das informações.

Mas não se preocupe em gravar o nome de cada imagem e o que ela representa, pois qualquer aplicativo capaz de realizar fluxogramas, por mais simples que seja, exibe o nome da imagem que está sendo utilizada.

Vários são os aplicativos existentes no mercado capazes de desenvolver desde o fluxograma mais simples até o mais elaborado. Muitas vezes, a ferramenta até existe em seu computador, mas você nunca se deu conta disso. Um bom exemplo são as formas existentes no pacote Office. Tanto o Word quanto o PowerPoint e o Excel possuem, nos grupos de opções Inserir, a opção Formas e os diversos símbolos existentes em fluxogramas. Se por acaso você trabalha com o BrOffice, pacote de aplicativos gratuito, pode utilizar a ferramenta Draw na criação dos fluxogramas.

Porém, se a ideia é trabalhar com representações mais elaboradas, há aplicativos disponíveis no mercado, alguns gratuitos e outros não, que valem a pena utilizá-los. Por exemplo: • Microsoft Visio • FlowChart

• BizAgi Process Modeler 1.4.0.0

• Best4c

• Diagram Designer • XMind

• Smartraw

• StarUML e outros.

Capítulo 3

Operadores aritméticos, relacionais e lógicos

Expressões e Operadores

É muito comum encontrarmos algoritmos que realizam cálculos matemáticos utilizando expressões aritméticas. Uma expressão nada mais é do que uma fórmula matemática que utiliza um conjunto de valores (variáveis ou constantes) e operadores, que após sua avaliação (execução), resultarão em um valor.

Existem vários tipos de expressões, sendo que as mais comuns são as aritméticas, seguidas das relacionais e lógicas.

Expressõesaritméticas São as expressões que, após sua execução, geram resultados cujo tipo sempre será numérico e permitem somente o uso de operadores aritméticos e variáveis numéricas em sua sintaxe. São elas: Soma

É representada pelo sinal de adição (+). Por exemplo:

Exemplo	Descrição
Nota1 + Nota2	Calcula a soma da expressão de duas variáveis.
10 + 20	Calcula a soma dos valores 10 e 20.

Tabela 3.1.: Operador de soma.

Subtração

É representada pelo sinal negativo (-). Por exemplo:

Exemplo	Descrição
Débito - Crédito	Calcula a subtração da expressão de duas variáveis.
20 - 5	Calcula a subtração do valor 20 por 5.

Tabela 3.2.: Operador de subtração.

Multiplicação Na Matemática, é representada pelo sinal x (xis) ou . (ponto). Em aplicativos no computador, sempre será representada pelo sinal (asterisco).

Exemplo	Descrição
Nota1 * 2	Calcula a multiplicação da variável Nota1 por 2.
10 * 2	Calcula a multiplicação dos valores 10 e 2.

Tabela 3.3.: Operador de multiplicação.

Divisão

Na Matemática, é representada pelo sinal - e em computadores pelo barra (/).

Exemplo	Descrição
Soma / 2	Calcula a divisão da variável Soma por 2.
10 / 2	Calcula a divisão do valore 10 pelo valor 2.

Tabela 3.4.: Operador de divisão.

Exponenciação Na Matemática, é representada pela base e um expoente, como por exemplo, 23. No computador, a exponenciação pode ser representada pelos sinais de dois asteriscos (**) ou acento circunflexo (^).

Exemplo	Descrição
M ** 2 2 ^ 3	Calcula o valor armazenado na variável M ao quadrado (elevado a 2). Calcula o resultado da exponenciação do valor 3 ao quadrado.

Tabela 3.5.: Operador de exponenciação.

Assim como na Matemática, existe uma ordem (precedência) de execução das operações, isto é, em uma expressão com vários cálculos a serem realizados, primeiramente serão executadas as exponenciações, seguidas das multiplicações e divisões, e por fim, as adições e subtrações. Também é possível utilizar parênteses para alterar essas precedências de

(2 + 3) * 5 / 2 – (5 – 3) ^ 2

5 * 5 / 2 – 2 ^ 2

25 / 2 – 4

12,5 – 4

cálculos. Na expressão: O resultado é 8,5.

Para o cálculo do resto da divisão entre dois valores inteiros, podemos utilizar o símbolo %, o que em algumas linguagens será representado pela expressão mod. Veja um exemplo: **5 % 2 ou 5 mod 2**

0 valor obtido como resultado será 1, pois indica a divisão do valor 5 por 2, e o que se deseja saber é qual o resto da divisão realizada.

Caso pretenda encontrar o quociente da divisão entre os dois valores, a expressão deverá ser: 5 div 2

0 valor obtido como resultado da divisão do valor 5 por 2 será 2.

Ao digitar uma expressão com div e %, o operador % terá prioridade sobre div, portanto, se for necessário que a operação div seja calculada

antes da %, coloque-o entre parênteses.

Acompanhe os seguintes exemplos:

```
Algoritmo divisão
Objetivo: Verificar prioridade em expressões com div e %

Início

1. Definir quatro variáveis que recebem números inteiros:
     Inteiro a, b, result, resto
2. Solicitar digitação do primeiro valor e armazenar em a:
     Ler a
     (Usuário digitou 20)
3. Solicitar digitação do segundo valor e armazenar em b:
     Ler b
     (Usuário digitou 10)
4.  Realizar a divisão de a por b e armazenar o resultado
em result:
     result = a div b
     (result = 20 / 10)
5. Exiba na tela o resultado da expressão armazenado em
result (2):
     Exiba result
6. Solicitar nova entrada para a:
     Ler a
     (usuário digitou 3)
7. Solicitar nova entrada para b:
     Ler b
     (usuário digitou 2)
8. Calcular o resto da divisão entre os dois valores e ar-
mazenar em resto:
     resto = a % b
9. Exibir na ela o resultado de resto (1).
10. Para que realize o cálculo de div primeiro, faça o se-
guinte, onde a = 135, b = 4:
       result = (a div b) % 3      <resultado = 0,75>

Fim
```

Outro ponto importante é que, assim como na Matemática, uma rotina não poderá ter nenhum valor dividido por zero (0), resultando também em erro.

Expressõesrelacionais Uma expressão relacional utiliza, basicamente, operadores que efetuam comparações entre dois ou mais valores

representados por constantes, variáveis ou expressões aritméticas. Tais expressões, após avaliarem a relação (comparação) entre os valores, geram um resultado os valores True (Verdadeiro) ou False (Falso), também conhecidos como valores booleanos.

Operador	Uso na Matemática	Uso em computadores	Exemplo	Resultado
Igual	=	==	2 = 4 ou 2 == 4	Falso
Diferente	≠	<>	2 ≠ 0 ou 2 <> 0	Verdadeiro
Maior	>	>	7 > 5	Verdadeiro
Menor	<	<	5 < 7	Verdadeiro
Maior ou Igual	>=	>=	7 >= 8	Falso
Menor ou igual	<=	<=	7 <= 8	Verdadeiro

Vejamos na tabela a seguir (Tabela 3.6) os operadores relacionais e seu respectivo uso na Matemática e em computadores: Tabela 3.6.: Operadores relacionais.

```
Algoritmo Relacional
Objetivo: Calcular novo salário
Início
1. Ler o salário atual
2. Se Salário Atual >= 1000
    Então
     2.1 Aumento = 10%
    Senão
            2.1 Aumento = 15%
    Fim Se
3. Calcular NovoSalário = Salário Atual + Aumento
4. Exiba NovoSalário
Fim
```

ExpressõesLõgicas

Operador	Uso	Uso na Matemática	Uso em computadores
Conjunção	Se dois operandos (valores) forem verdadeiros o resultado será verdadeiro.	E	&& ou And
Disjunção	Se um dos operandos (valores) for verdadeiro o resultado será verdadeiro.	Ou	|| ou Or
Negação	Se o operando (valor) é verdadeiro o resultado é falso.	Não	! ou Not

Uma expressão lógica é aquela cujos operadores são relações comparativas que produzem como resultado um valor booleano True (Verdadeiro) ou False (Falso). Vejamos os operadores lógicos na Tabela 3.7:

Tabela 3.7.: Operadores lógicos.

Ao incluir condições em algoritmos, estipulamos uma possibilidade de escolha do usuário e, a partir dessa premissa, novas decisões poderão ser tomadas. Para auxiliar nessa tomada de decisões, utilizamos as Tabelas de Decisões que são conhecidas em informática como Tabelas Verdade.

Tabela Verdade E (And)

Aluno	Você domina PHP?	Você domina SQL?	Resultado
1	Não	Não	Falso
2	Não	Sim	Falso
3	Sim	Não	Falso
4	Sim	Sim	Verdadeiro

Imagine que duas perguntas foram feitas a quatro alunos sobre os seus conhecimentos em linguagens de programação e somente será chamado

para uma entrevista, o candidato conhecedor de ambas as linguagens. Veja as respostas na Tabela 3.8: Tabela 3.8.: Tabela verdade do operador lógico E.

Nesse exemplo, foi questionado se o aluno domina a linguagem PHP e (and) SQL. Portanto, somente será convocado para uma entrevista o aluno 4, por dominar ambas as linguagens.

Tabela Verdade Ou (or)

Aluno	Você domina C?	Você domina Java?	Resultado
1	Não	Não	Falso
2	Não	Sim	Verdadeiro
3	Sim	Não	Verdadeiro
4	Sim	Sim	Verdadeiro

As mesmas perguntas foram feitas a outros quatro alunos, mas a entrevista será agendada com aqueles que dominam pelo menos uma das linguagens. Neste caso, o operador seria Ou (or). Vejamos quem deve ser convocado à entrevista (Tabela 3.9): Tabela 3.9.: Tabela verdade do operador lógico OU.

Nesse caso, somente o aluno 1 não será convocado para uma entrevista, pois ele não domina nenhuma das linguagens citadas.

Aluno	Você é menor de 18 anos?	Resultado
1	Sim	Falso
2	Sim	Falso
3	Não	Verdadeiro
4	Não	Verdadeiro

Tabela Verdade de Negação (NOT) Para a convocação das entrevistas, os candidatos não podem ser menores de 18 anos. Vejamos quais foram as

respostas (Tabela 3.10): Tabela 3.10.: Tabela verdade do operador lógico NOT.

Neste exemplo, foram convocados os alunos 3 e 4 para as entrevistas, por eles NÃO serem menores de 18 anos.

```
Algoritmo And
Objetivo: Calcular aumento de salário
Início
1. Ler o salário
2. Se salário >= 1000 E salário <2500
    Então
        2.1 Aumento = 5%
    Senão
        2.1 Aumento = 10%
    Fim Se
3. Calcular Novo = Salário + Aumento
4. Exiba Novo
```

Funcionário	Salário	Resultado	Aumento
Sérgio	R$ 1.800,00	Verdadeiro	5%
José Ricardo	R$ 990,00	Falso	10%
Guilherme	R$ 2.700,00	Falso	10%
Fernanda	R$ 2.450,00	Verdadeiro	5%

Fim

Vejamos os salários obtidos e seus respectivos aumentos na Tabela 3.11: Tabela 3.11.: Respostas obtidas a partir do algoritmo And.

Prioridade	Operador	Exemplo	Prioridade
1	Parênteses	= (2 * (5 + 2) / 3)	Em primeiro lugar, será feita a soma de 5 e 2 e, a seguir, o valor será multiplicado por 2 e dividido por 3.
2	Aritméticos	= 2 * 5 + 2 / 3	Em primeiro lugar a multiplicação, seguida da divisão e por fim a adição.
3	Lógicos	= Camisa Not Vermelha And Idade >=18 Or Linguagem = “Java”	Em primeiro lugar a prioridade será para o operador Not, seguido de And e finalmente Or.

No uso de expressões com operadores relacionais, lógicos e aritméticos também encontramos precedência nos cálculos (operações). Vejamos quais são: Tabela 3.12.: Exemplos de prioridade de cálculo de acordo com o operador e o uso de parênteses.

Capítulo 4

Variáveis, constantes, tipos de dados e funções

Vimos, até agora, que a principal função de um programa é manipular valores oriundos de cálculos realizados em expressões.

Existem determinadas situações em que o cálculo é realizado e o resultado de uma expressão será exibido somente na tela e mais nada. Porém, na maioria das vezes, o resultado de uma expressão ou de um conjunto de instruções (rotina) será utilizado em outras ocasiões. Para isso, o ideal é armazenar o valor na memória do computador.

Imagine que exista a necessidade de desenvolver um programa que exiba caixas de diálogo com o resultado de várias expressões a serem executadas. E que o programa deverá interromper a sua execução somente no momento em que uma determinada instrução for localizada, como, por exemplo, Sair. Nesse caso, várias informações serão solicitadas ao usuário e, após a entrada dessas, tais valores devem ser armazenados na memória do computador, pois eles servirão de base para a manipulação de outros dados.

Nos algoritmos realizados até aqui, nenhuma informação foi armazenada. Somente alguns valores foram lidos e operações aritméticas foram executadas. No caso de utilizar uma variável de memória, o que ocorrerá é que será reservado um espaço para os dados na memória principal (RAM) do computador. Esse espaço possui uma localização predefinida, onde o valor armazenado poderá ser recuperado a qualquer instante, sempre que for necessário, agilizando assim o processamento de todas as informações e, principalmente, a organização da aplicação.

Para o computador poder trabalhar com a informação contida em uma variável, primeiramente é necessário saber qual o endereço onde ela foi armazenada. Para exemplificar, imagine que a variável é uma pequena

caixa e ela será posicionada na primeira prateleira disponível. Fisicamente, cada caixa ocupa um local específico e outra caixa não poderá ser armazenada na mesma prateleira. Da mesma forma, caixas diferentes não podem receber o mesmo rótulo (nome).

0 endereço (prateleira) é representado por um número hexadecimal indicando onde a informação foi armazenada, qual o conteúdo e o tamanho em bytes da variável. Esse tamanho representa o espaço reservado para conter a informação e irá variar de linguagem para linguagem. Veja um exemplo na Tabela 4.1:

Endereço Físico	Conteúdo
2000:12EC	18
3000:B713	"José"

Tabela 4.1.: Demonstração de dados e seus respectivos endereços físicos (hexadecimais) de memória.

Para o computador isso é fácil de interpretar, agora, para usuários, é bastante complicado informar o endereço onde se deseja armazenar e o que se deseja armazenar. Pensando nisso, as linguagens de programação facilitam a manipulação de tais tipos de dados e, dessa forma, não é necessário definir a posição de memória no computador, ficando somente com a definição do endereço lógico, que na verdade é o nome da variável. Assim, ao definir o endereço lógico, estamos na verdade informando a existência da variável, bem como de seu conteúdo (Tabela 4.2):

Endereço Lógico	Conteúdo
Idade	18
Nome	"José"

Tabela 4.2.: Demonstração de dados e seus respectivos endereços lógicos (nomes) de memória.

Sem dúvida alguma, esse tipo de atribuição é muito mais fácil de ser compreendido.

variável de memória Podemos definir variável como um espaço que será alocado na memória do computador, com a finalidade de armazenar determinada informação. Como o próprio nome diz "variável" sugere que tais informações podem variar, ou seja, a cada vez que o sistema ou rotina for executada, o valor armazenado poderá ser diferente.

Dependendo da linguagem que se utiliza, algumas regras devem ser obedecidas ao nomear uma variável: • Sempre iniciar com uma letra.

- Nenhuma palavra reservada própria da linguagem poderá ser usada como nome de uma variável. Por exemplo, int ou string, que estão reservadas para definir um tipo de variável e nunca um nome.

- Espaços não serão permitidos, como, por exemplo, "media do aluno", sendo o correto MediaDoAluno ou Media_do_Aluno, por exemplo.

Declarando variáveis Vários são os tipos de dados que podem ser armazenados em uma variável, por isso, é bom entender que, para cada conteúdo, um tipo de variável será declarado.

Ao declarar uma variável no início do algoritmo, estamos solicitando ao programa que seja reservada (alocada) uma área na memória do computador como um endereço fixo para ela. Além do mais, informa-se também que, naquele endereço, apenas um tipo de informação será armazenado.

Isso significa que, antes de indicar o conteúdo de uma variável, será necessário declarar a sua existência, determinado assim, o tipo de dado esperado. Normalmente, ao se trabalhar com algoritmos, verificamos a existência de quatro tipos de dados: inteiros, reais, caracteres e lógicos. Mas, ao programar, você verificará que outros tipos podem ser necessários, como data, hora, byte e outros, sendo que tudo dependerá da linguagem utilizada.

Para a declaração de uma variável, normalmente utilizamos a seguinte sintaxe nos algoritmos:

```
Tipo da variável nome da variável <comentário>
```

Por exemplo:

```
Algoritmo Exemplo _ Declarar
Objetivo: Calcular o reajuste de salário
Início
   1. Declarar a existência da variável x:
Inteiro x <variável que armazena a idade do usuário>

   2. Declarar a existência da variável salário:
Real salário <variável que armazena o salário atual>

   3. Declarar a existência da variável aumento:
Real aumento <variável que armazena o percentual de aumen-
to>

   4. Definir as etapas de processamento:
...
...
<rotina de processamento>

Fim
```

Atribuindovaloresãsvariáveis A ação de atribuir valores a uma variável indica que determinado conteúdo está sendo armazenado nela. Essa atribuição alterna de linguagem para linguagem, mas, normalmente, em algoritmos, utiliza-se a seguinte sintaxe:

```
Nome da Variável = valor atribuído
```

Vale ressaltar que na parte esquerda devemos indicar o nome da variável a qual se está atribuindo um valor, ou seja, quem irá receber a informação, seguida do valor que deverá permanecer à direita da atribuição. Alguns algoritmos permitem a atribuição de um valor utilizandose setas:

```
Nome da variável ←  valor atribuído
Algoritmo Exemplo _ Atribuir
Objetivo: Calcular o reajuste de salário
```

Início

1. **Declarar** todas as variáveis a serem utilizadas:
Inteiro x <variável que armazena a idade do usuário>
Real salário <variável que armazena o salário atual>
Real aumento <variável que armazena o percentual de aumento>
Real Novo <variável que armazena o novo salário>

2. **Ler** x (Armazenar o valor digitado pelo usuário na variável x):
x = 18

3. **Ler** salário (Armazenar o valor digitado na variável salário):
Salário = 1200,00

4. **Se** salário >=1000
Então
4.1 **Aumento = 0,10**
Senão
4.1 **Aumento = 0,15**
Fim Se

5. Calcular novo salário:
Novo = salário + aumento

6. **Exiba** novo

Fim

Esse algoritmo também poderá ser descrito da seguinte forma:

Algoritmo Exemplo _ Atribuir
Objetivo: Calcular o reajuste de salário

```
Início
     1. Declarar todas as variáveis a serem utilizadas:
    Inteiro x <variável que armazena a idade do usuário>
    Real salário <variável que armazena o salário atual>
    Real aumento <variável que armazena o percentual de aumento>
    Real Novo <variável que armazena o novo salário>

2. Ler x (Armazenar o valor digitado pelo usuário na variável x):
    x ← 18

3. Ler salário (Armazenar o valor digitado na variável salário):
    Salário ← 1200,00

4. Se salário >=1000
    Então
       4.1 Aumento ← 0,10
    Senão
       4.1 Aumento ← 0,15
    Fim Se

5. Calcular novo salário:
    Novo ← salário + aumento

6. Exiba novo

Fim
```

Tipos de Dados Vimos que, ao declarar uma variável, devemos informar o tipo de dado que ela deverá armazenar. Vejamos agora cada um dos tipos existentes de forma mais específica.

Numérico

Como o próprio nome indica, variáveis numéricas são aquelas que armazenam dados do tipo numérico, e costumam se dividir em duas classes: inteiro ou real.

As variáveis do tipo Inteiro servem para armazenar números inteiros positivos ou negativos, que não possuam dígitos decimais ou fracionários. Normalmente, esse tipo de variável costuma reservar um espaço de 1, 2 ou

4 bytes na memória do computador. Como exemplo, temos 24 (número inteiro positivo) ou -24 (número inteiro negativo).

As variáveis do tipo Real são aquelas que podem possuir dígitos decimais ou fracionários, positivos ou negativos. 0 espaço reserva do na memória poderá ser de 4 ou 8 bytes. Como exemplo de variáveis do tipo Real, podemos ter:

Valor	Descrição
12.05	Número real positivo com duas casas decimais.
-12.5	Número real negativo com uma casa decimal.
1200.	Número real positivo com zero casa decimal.

Tabela 4.3.: Exemplos de variáveis do tipo Real.

A1fanumérico Também conhecidas como varáveis do tipo caractere ou string (cadeia de caracteres), são aqueles que podem armazenar informações compostas por um simples caractere ou um conjunto de letras, dígitos e símbolos especiais.

Para sua atribuição, podemos utilizar as aspas simples (') ou duplas ("). Ao atribuir um valor para uma variável e solicitar à linguagem que ele seja interpretado como alfanumérico, utilizamos as aspas simples. Por exemplo:

```
Ano = '2009'
```

Ou

```
Ano ← '2009'
```

Nesse caso, o conteúdo da variável não servirá na realização de cálculos, pois será considerado como caracteres de texto.

Já as aspas duplas servem para a atribuição de uma cadeia de caracteres

Nome = "José Ricardo"

Ou

(string), como por exemplo: **Nome ← "José Ricardo"**

Conteúdo da variável:														
J	O	S	É	\0	R	I	C	A	R	D	O	\0		
Posições dos caracteres na variável:														
0	1	2	3	4	5	6	7	8	9	10	11	12	...	...

A quantidade de bytes reservados no armazenamento de uma variável do tipo string depende sempre da linguagem a ser utilizada, normalmente será armazenada como uma matriz linha (vetor):

As variáveis do tipo string não realizam cálculos.

Caractere

As variáveis do tipo caractere armazenam somente uma letra qualquer digitada pelo usuário e sua declaração deverá ser: **caractere X**

Será armazenado na variável, qualquer caractere digitado pelo usuário e **Caractere "X"**

Ou

armazenado o respectivo byte caso seja declarada como: **Caractere 'X'**

Lõgico

Variáveis do tipo lógico servem para armazenar somente um dos dois valores lógicos possíveis, Verdadeiro ou Falso, e também são conhecidas como variáveis do tipo booleano.

Continua = Verdadeiro
Continua = Falso

Ou

Continua ← Verdadeiro
Continua ← Falso

Veja um exemplo de atribuições de tipos de variáveis:

Algoritmo Definindo _ Tipos
Objetivo: Definir tipos de dados armazenados em variáveis.

Início
1. Declarar a existência da variável nome, que deverá armazenar o nome do usuário:
String nome

2. Declarar a existência da variável idade, que deverá armazenar a idade do usuário:
Inteiro idade

3. Declarar a existência da variável tecla, que deverá armazenar uma tecla pressionada:
Caractere tecla

4. Declarar a existência da variável salário, que deverá armazenar o salário do usuário:
Real Salário
5. Ler nome:
Nome = "Fernanda"

6. Ler idade:
Idade = 20

7. Ler tecla:
Tecla = A

8. Ler salário:
Salário = 900,50

9. Exibir "Olá!"
10. Exibir Nome, idade e salário.

Fim

Veja no exemplo a seguir se você consegue localizar qualquer erro na declaração das variáveis:

```
Algoritmo Definindo _ Erros
Objetivo: Definir tipos de dados armazenados em variáveis.
Início
1.   Declarar a existência da variável nome, que deverá ar-
mazenar o nome do usuário:
String nome

2.   Declarar a existência da variável idade, que deverá
armazenar a idade do usuário:
Inteiro idade
3.   Declarar a existência da variável tecla, que deverá
armazenar uma tecla pressionada:
Caracter tecla

4.   Declarar a existência da variável salário, que deverá
armazenar o salário do usuário:
Real Salário

5.   Ler nome:
Nome = "Fernanda"

6.   Ler idade:
Idade = "desconhecida"

7.   Ler tecla:
Tecla = "Barra de Espaços"

8.   Ler salário:
Salário = "1.200,00"

9.   Exibir "Olá !"
10.  Exibir Nome, idade e salário.

Fim
```

Fique atento, pois definir um tipo errado de variável ocasionará um erro no sistema, visto que muitos cálculos podem ser realizados a partir dessa entrada de valores e o aplicativo simplesmente indicará ao usuário que algo está errado na atribuição de valores às variáveis e ele pode nem saber o que fez de errado.

No exemplo anterior, a variável idade recebeu uma cadeia de caracteres e

não um valor inteiro. A variável tecla também recebeu uma cadeia de caracteres e não uma simples tecla e o salário também, quando na verdade, deveria receber um número real, sem aspas ou vírgulas.

Constantes

Uma constante, assim como as variáveis de memória, também se refere a um espaço reservado na memória do computador. A diferença entre ambas é que, como o próprio nome indica, uma variável terá seu conteúdo alterado cada vez que a rotina for executada, já a constante terá sempre o mesmo valor alocado em sua memória, do início ao fim da rotina, e não poderá ser alterado.

Sua declaração também deverá ser efetuada ao iniciar o algoritmo e da mesma forma:

```
Constante Editora = "Universo dos Livros"
Constante CNPJ = "11.111.111/0001-11"
Constante Número = 789
```

Funções

Podemos dizer que uma função nada mais é do que uma fórmula matemática, onde, realizaremos cálculos com um conjunto de valores e operadores e que sempre resultará em um valor. A maioria das funções possui argumentos que dependem da linguagem a ser utilizada.

Uma função é uma subrotina que deverá conter uma tarefa específica (cálculo) a ser realizado. Inicialmente, deve-se declarar todas as variáveis que farão parte da função, seguidas da sequência de instruções que serão executadas.

Algumas funções recebem valores e outras podem ser utilizadas na conversão de tipos de dados.

Funçõesnuméricas São as funções cujo resultado de sua execução irá gerar um dado do tipo numérico, podendo ser efetuadas somente com variáveis do tipo numérica. Podemos exemplificar, como funções numéricas, a função pi, seno, co-seno, tangente, absoluto, logaritmo e outras: a. Função pi

Esta função retornará o valor 3.14159265 e não possui argumentos. Sua sintaxe é: `pi`

b. Função seno Esta função retornará o valor do seno de um ângulo qualquer em radianos e possui a seguinte sintaxe:

```
Sen(x)
```

Antes da utilização da função seno, devemos transformar o ângulo em graus para radiano, ou seja: `Ang* 3.14159265 / 180`

Veja um exemplo da utilização da função:

```
EmGraus = Ang * pi / 180    <atribuído valor à variável Em-
Graus>
Exiba sen(EmGraus)    <exiba na tela o resultado da função
sen>
```

c. Função cosseno Função que retorna o valor do cosseno de um ângulo qualquer em radianos. Nessa função também devemos, primeiramente, transformar o ângulo em graus para radiano:

```
EmGraus = Ang * pi / 180    <atribuído valor à variável Em-
Graus>
Exiba cos(EmGraus)    <exiba na tela o resultado da função
cosseno>
```

d. Função tangente Função que retorna o valor da tangente de um ângulo qualquer em radianos. Também exige-se que o ângulo em graus seja transformado em radianos para sua utilização:

```
EmGraus = Ang * pi / 180    <atribuído valor à variável Em-
Graus>
Exiba tan(EmGraus)    <exiba na tela o resultado da função
tangente>
```

e. Função absoluto Função que retorna um valor absoluto de um número qualquer.

```
Abs(2)     <irá exibir como resultado 2>
Abs(-2)    <irá exibir como resultado 2>
```

f. Função exponencial Função que retorna o valor de um número e (base do algoritmo neperiano) elevado a outro número qualquer.

```
Exp(2)    <resulta no valor 7,389056099>
```

Seria o mesmo que digitar: `2.71828182846**2`

g. Função log Função que retorna o valor do logaritmo neperiano de um número qualquer.

```
log(2)      <resulta no valor 0,301029996>
```

h. Função Raiz Função que retorna o valor da raiz quadrada de um número positivo.

Sqr(4) <resulta em 2, é o mesmo que $\sqrt{4}\sqrt{4}$>

As funções de conversão de tipos de dados dependem totalmente da linguagem em que elas estão sendo utilizadas, por isso, não colocaremos suas sintaxes, mas basicamente todas elas possuem suas equivalentes de conversão de número real em inteiro, de comprimento de uma string (tamanho), retorno dos primeiros ou últimos caracteres de uma cadeia de dados, entre outras.

Capítulo 5

Estruturas de decisão

Até agora, praticamente todos os algoritmos desenvolvidos acompanharam uma linha reta de processamento de informações, isto é, seguiram um fluxo de cima para baixo executando pequenas tarefas. Mas para que um sistema seja considerado realmente interativo, algumas decisões devem ser tomadas. E qual o melhor caminho a seguir?

Assim como qualquer caminho, algumas bifurcações devem existir, fazendo com que a sequência de instruções tome um rumo previsto e/ou esperado. E para tal, é importante analisar qual será o trajeto a ser escolhido, ou seja, analisarmos todas as possibilidades.

Rotinas assim são conhecidas como estruturas de tomada de decisão, de controle do fluxo de informações ou também de estruturas condicionais. Nesse tipo de estrutura, determinadas instruções serão executadas ou não, dependendo do resultado de um teste ou verificação de uma condição.

Vejamos as seguintes situações: • Fazer um cadastro para participação de alunos em uma gincana.

• Fazer um cadastro para participação em uma gincana com alunos menores de 14 anos.

• Fazer um cadastro de três garotos e duas garotas menores de 14 anos para participação em uma gincana.

Podemos observar que, no primeiro caso, deverá ser feito um cadastro com dados de todos os alunos de determinada escola. Ou seja, é preciso criar um algoritmo que leia o nome, endereço, idade e telefone de todos aqueles entrevistados.

Algoritmo Cadastro Objetivo: Cadastrar nome, endereço, idade e telefone de alunos participantes de uma gincana

```
Início

     1. Declarar variáveis nome, endereço, idade e telefone:
       String nome, endereço, telefone
       Inteiro idade

     2. Ler nome
     3. Ler endereço
     4. Ler telefone
     5. Ler idade

Fim
```

Na segunda situação, o algoritmo deveria ser diferente, pois ao entrevistar o aluno, ele deve requisitar, em primeiro lugar, a sua idade. Se a criança for menor de 14 anos, teria o seu cadastro feito, caso contrário, nem preencheria a ficha.

Algoritmo CadastroParaMenores Objetivo: Cadastrar nome, endereço, idade e telefone de alunos menores de 14 anos para a participação em uma gincana.

```
Início

     1. Declarar variáveis nome, endereço, idade e telefone:
       String nome, endereço, telefone
       Inteiro idade

     2. Ler idade
       SE idade <= 14 anos
     2.1 Ler nome
     2.2 Ler endereço
     2.3 Ler telefone
       FIM SE
Fim
```

E na terceira situação, mais de uma condição deve ser analisada: primeiro o sexo, se feminino ou masculino, em seguida a idade, somente menores de 14 anos. Mas as duas decisões também deveriam ser repetidas somente três e duas vezes (preencher o cadastro) em cada caso, pois somente poderá contemplar três garotos e duas garotas.

```
Início

1. Declarar as variáveis sexo, nome, endereço, idade e te-
lefone:
String sexo, nome, endereço, telefone
Inteiro idade, cadastro

2. Ler conteúdo da variável idade
Ler idade

SE idade <=14 anos
2.1 Ler sexo
SE sexo = "feminino"
2.1.1 Cadastro = 1
2.1.2 Ler nome
2.1.3 Ler endereço
2.1.4 Ler telefone
2.1.5 Cadastro = cadastro + 1
SE cadastro <3
     2.2 repetir os passos de 2.1.2 a 2.1.5
FIM SE
FIM SE
SENÃO
2.1.1 Cadastro = 1
2.1.2 Ler nome
2.1.3 Ler endereço
2.1.4 Ler telefone
2.1.5 Cadastro = cadastro + 1
SE cadastro <=2
     2.2 repetir os passos de 2.1.2 a 2.1.5
FIM SE

FIM SE

Fim
```

Para a primeira situação, nenhuma surpresa, mas para a segunda e principalmente a terceira, devem ser testadas algumas condições para que o fluxo das informações siga por caminhos variados, ou seja, repita algumas instruções por um número determinado de vezes, como por exemplo, ler a idade do participante, se menor de 14 anos, realizar o cadastro, caso contrário finalizar o sistema.

Repare que, no terceiro algoritmo, o ideal seria testar a condição (se

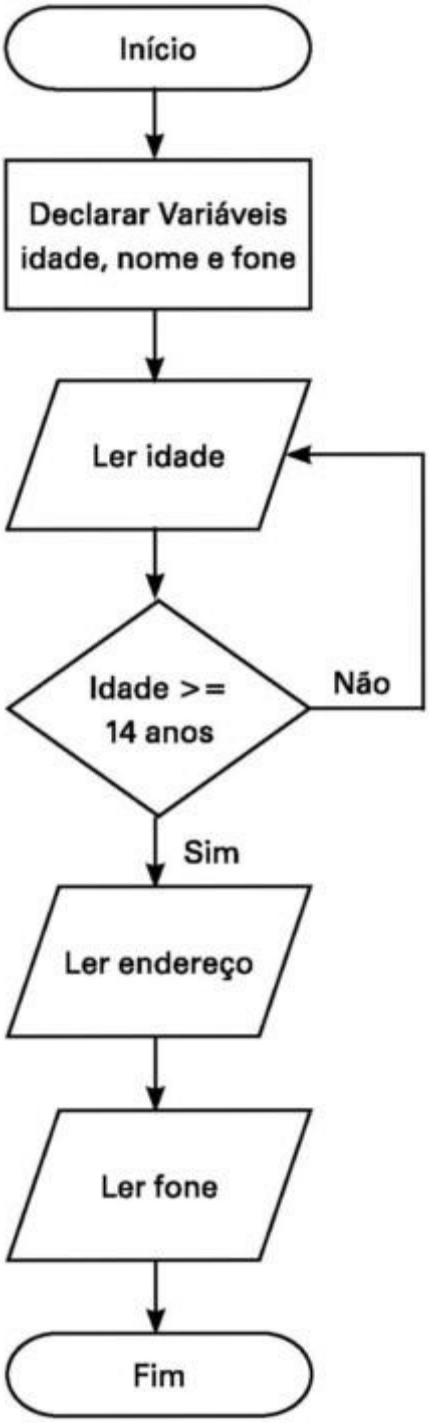
menor de 14 anos) e, em seguida, repetir o cadastro duas vezes, quando a condição for para o sexo feminino, já para o sexo masculino deverá repetir o cadastro por três vezes.

Os algoritmos acima servem somente para exemplificar as situações, pois como veremos a partir de agora, todo sistema possui estruturas condicionais e estruturas de repetição, também conhecidas como loops.

Tipos de Estruturas de decisão Estrutura de decisão simples Este tipo de estrutura permite avaliar se uma condição é verdadeira e direcionar o fluxo da rotina de acordo com o resultado do teste realizado. Também são conhecidas como estrutura SE. Veja a sua sintaxe:

```
SE (condição a ser verificada)

    Instruções a serem executadas caso a condição seja ver-
dadeira.

FIM SE
```

Para testar a funcionalidade de uma estrutura simples, acompanhe o



seguinte fluxograma:

Figura 5.1.: Fluxograma demonstrando leitura de dados.

Veja em forma de algoritmos:

```
Algoritmo Idade
Objetivo: Verificar se aluno possui idade menor ou igual a
14 anos
```

Início

1. Declarar as variáveis nome, endereço, idade e telefone:
String nome, endereço, telefone
Inteiro idade

2. Ler conteúdo da variável idade
Ler idade

SE idade <=14 anos
2.1 Ler sexo
2.2 Ler nome
2.3 Ler endereço
2.4 Ler fone
FIM SE

Fim

Vejamos outro exemplo de verificação de uma condição.

Ler um número inteiro
Condição: Verificar se o mesmo é maior que 50.
Se condição for verdadeira, exiba a metade do número digitado
Algoritmo Maior50
Objetivo: Verificar se número é maior que 50.
Início

1. Declarar a variável num:
Inteiro num

2. Exibir a mensagem na tela:
Exiba "Digite um número:"

3. Ler conteúdo da variável num:
Ler num

SE num > 50

```
     3.1 Exibir na tela a mensagem:
           Exiba "A metade deste número é:"
     3.2 Calcular a metade do número e exibir na tela o re-
sultado:
           Exiba Num / 2        (ou:)
     3.2 Exibir na tela a mensagem e o resultado:
           Exiba "A metade deste número é:" e num/2
FIM SE

Fim
```

Outra solução para o algoritmo consiste em declarar duas variáveis, uma para ler o número e outra para armazenar a metade deste valor:

```
Início

1. Declarar as variáveis num e metade:
Inteiro num
real metade

2. Exibir a mensagem na tela:
Exiba "Digite um número:"

3. Ler conteúdo da variável num:
Ler num

SE num > 50
     3.1 Calcular a metade do número e exibir na tela o re-
sultado:
           Metade = Num / 2
       Exiba "A metade do número é:" e metade
FIM SE

Fim
```

A estrutura permite outra sintaxe quando somente uma condição deve ser verificada e uma instrução deve ser executada:
`SE <condição> ENTÃO <uma instrução>`

A seguir, um algoritmo que utiliza essa estrutura:

```
Início

1. Declarar a variável num:
```

```
Inteiro num

2. Exibir a mensagem na tela:
Exiba "Digite um número:"

3. Ler conteúdo da variável num:
Ler num

SE num > 50 ENTÃO Exiba "A metade é:" e num/2

Fim
```

Até aqui, uma condição foi verificada e, se verdadeira, o fluxo das informações deverá seguir determinada sequência. Agora imagine que ela deverá executar uma tarefa se a condição for verdadeira e outro conjunto de instruções caso seja identificada como falsa. Esse tipo de estrutura é conhecida como estrutura de decisão composta e possui a seguinte sintaxe:

```
SE <condição> ENTÃO <instrução1>

SENÃO <instrução2>

FIM SE
```

Nesse caso, uma condição será analisada e, se verdadeira, executará a instrução existente após ENTÃO (instruçãol). Caso contrário, realizará a condição designada após SENÃO (instrução2).

Outra forma de escrever a estrutura de decisões composta é:

```
SE <condição> ENTÃO
     <conjunto de instruções1>
SENÃO
     <conjunto de instruções2>
FIM SE
```

Sendo que nesse caso: 1. A condição será avaliada.

2. Caso seja verdadeira, executará o conjunto de instruções apresentadas após ENTÃO (instruçõesl) e, neste caso, podem ser encontradas várias linhas de instruções.

3. Caso a condição não seja verdadeira, serão realizadas as ações designadas após SENÃO (instruções2).

4. Após realizar as instruções caso verdadeira ou falsa, serão executadas todas as instruções que aparecem após a instrução FIM SE até o final da execução do programa.

Para exemplificar, vamos criar um algoritmo que verifica se o número é par ou ímpar. Caso seja par, apresente a mensagem "0 número par digitado foi x". Caso ímpar, a mensagem deverá ser "0 número ímpar digitado foi x". Antes de finalizar o programa, apresente a mensagem "Até a próxima e obrigado".

```
Início

1. Declarar a variável par:
Inteiro numero

2. Exibir a mensagem na tela:
Exiba "Digite um número:"

3. Ler conteúdo da variável par:
Ler numero


SE ( numero % 2 = 0 ) ENTÃO        <verificar se o resto da divisão é 0>
     3.1 Exibir na tela a mensagem:
           Exiba "O número par digitado é " e numero
SENÃO
     3.1 Exibir na tela a mensagem:
           Exiba "O número ímpar digitado é " e numero
FIM SE

4. Exibir na tela a mensagem:
Exiba "Até a próxima e obrigado"

Fim
```

Em outro exemplo, uma empresa revende sapatos a R$ 70,00 cada. 0 usuário informa a quantidade de sapatos desejada e o sistema deve calcular o valor do frete deste pedido, informando o valor total da venda. Caso a entrega seja feita no estado de São Paulo, o frete deverá ser de 2% sobre o valor da venda, caso no estado do Paraná, o frete será de 5% sobre o mesmo valor.

```
Início

1. Declarar a variável qtde:
Inteiro qtde

2. Declarar a variável estado:
String estado

3. Declarar a variável frete:
Real frete

4. Exibir a mensagem na tela:
Exiba "Informe a sigla do estado (2 caracteres) da entre-
ga:"

5. Ler conteúdo da variável estado
Ler estado
SE estado = "SP" ENTÃO
     5.1 Exibir na tela a mensagem:
           Exiba "Informe a qtde de sapatos desejados:"
     5.2 Ler o conteúdo da variável qtde:
           Ler qtde
     5.3 Calcular o frete:
           Frete = (qtde * 70.) * 0.02
     5.4 Exibir a mensagem:
           Exiba "O valor total de seu pedido é:" e (qtde *
70.) + frete
SENÃO
     5.1 Exibir na tela a mensagem:
           Exiba "Informe a qtde de sapatos desejados:"
     5.2 Ler o conteúdo da variável qtde:
       Ler qtde
     5.3 Calcular o frete:
           Frete = (qtde * 70.) * 0.05
     5.4 Exibir a mensagem:
          Exiba "O valor total de seu pedido é:" e (qtde *
70.) + frete
FIM SE

Fim
```

Estrutura de decisão encadeada ou aninhada Outra forma de utilização das estruturas é encadear ou aninhar outras condições a uma estrutura de decisão. Nesse caso, as instruções que compõem um conjunto determinado como verdadeiro ou falso, também poderão receber uma ou mais condições

a serem testadas.

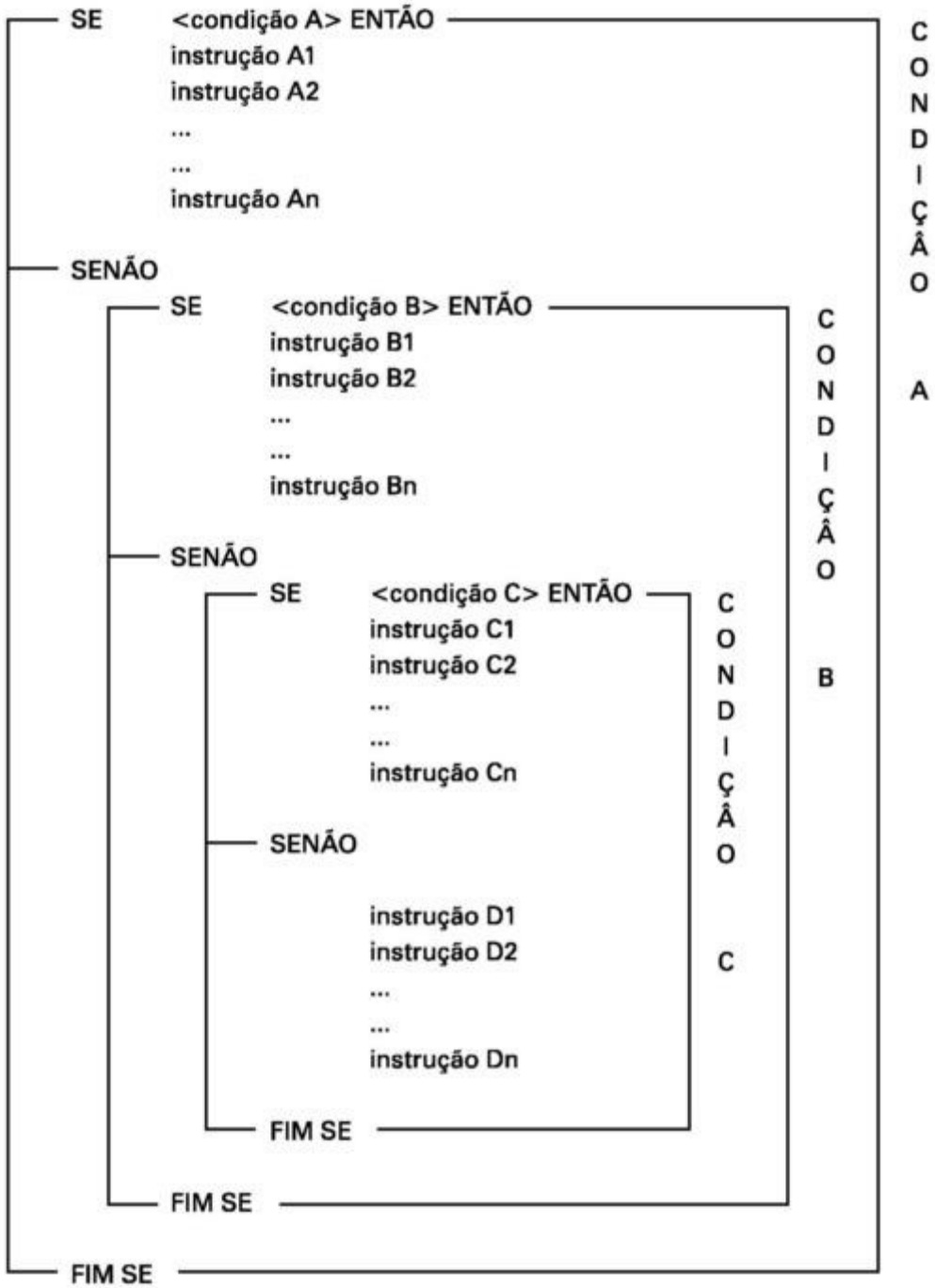


Figura 5.2.: Exemplo de estruturas aninhadas.

Vejamos um exemplo para cálculo da média aritmética semestral de alunos de determinada escola. Após solicitar as duas notas do aluno e calcular a média:

• exibirá a mensagem "Aprovado" quando a média for igual ou superior a 7.0;

• exibirá a mensagem "Exame" quando a média for igual a 5.0 e menor que 7.0;

• exibirá a mensagem "Reprovado" para aqueles cuja média for inferior a 5.0.

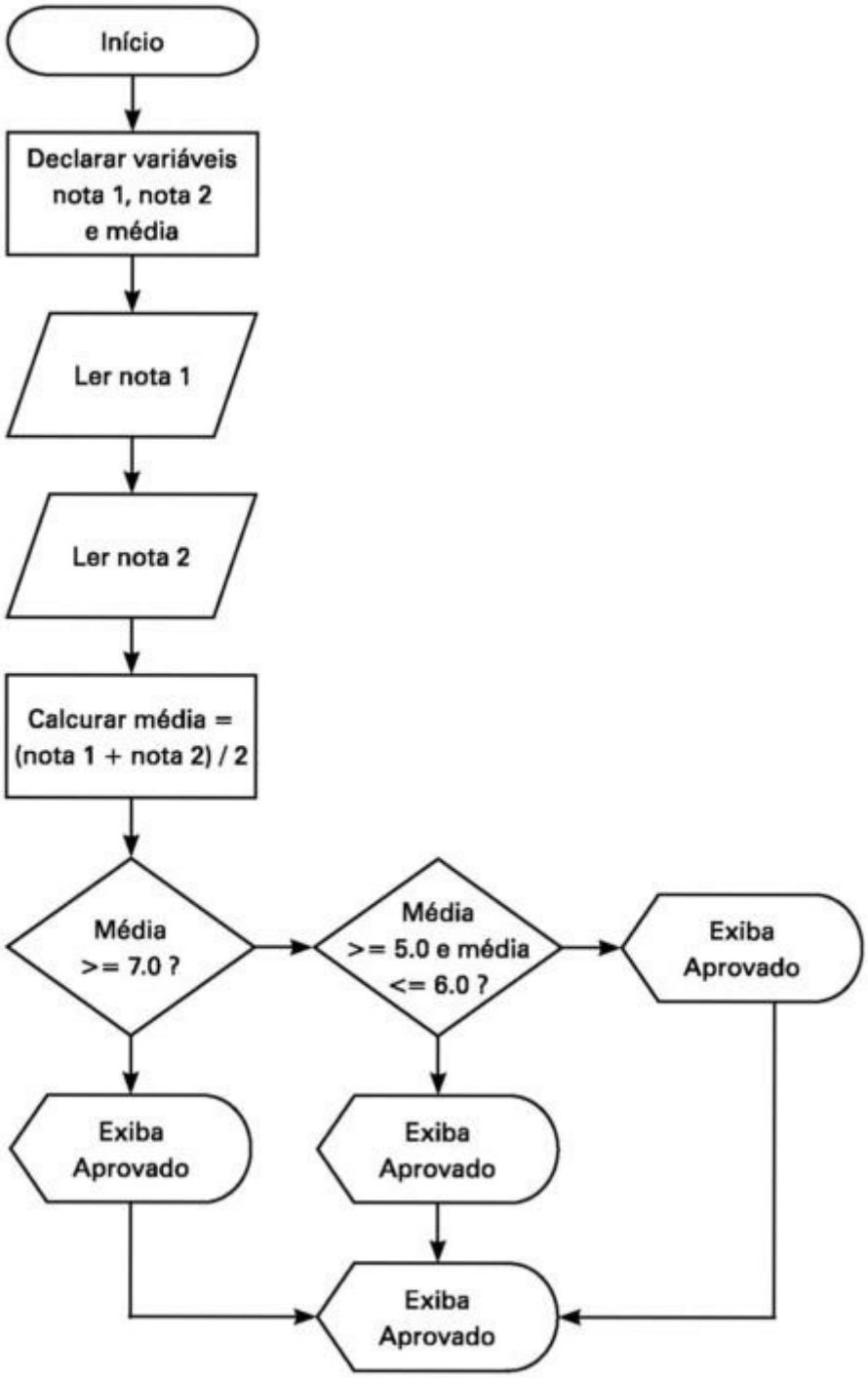


Figura 5.3.: Fluxograma de cálculo de média aritmética.

Vejamos como fica o algoritmo:

```
Algoritmo: Encadeados
Objetivos: Calcular média do aluno e determinar se aprovado, reprovado ou exame.

Início

1. Declarar as variáveis nota1, nota2 e média:
real nota1, nota2, média

2. Exibir a mensagem na tela:
Exiba "Entre com a primeira nota:"

3. Ler a resposta do usuário e armazenar em nota1
Ler nota1

4. Exibir a mensagem na tela:
Exiba "Entre com a segunda nota:"

5. Ler a resposta do usuário e armazenar em nota2:
Ler nota2

6. Calcular média:
Média = (nota1 + nota2) / 2

7. SE média >= 7.0 ENTÃO
      7.1 Exibir na tela a mensagem:
        Exiba "Parabéns, você foi Aprovado"
      SENÃO SE média >=5.0 E média<=6,0 ENTÃO
      7.1 Exibir a mensagem:
        Exiba "Falta pouco! Você deverá fazer o exame"
      SENÃO
      7.1 Exibir a mensagem
        Exiba "Sinto muito, mas você foi reprovado"
      FIM SE
FIM SE

Fim
```

Outro exemplo clássico de estruturas de decisão aninhas é a calculadora. Para verificar o seu funcionamento, devemos ler dois valores digitados pelo usuário (valorl e valor2), realizar o tipo de cálculo desejado (adição,

subtração, multiplicação ou divisão) e, além disso, exibir o resultado da operação.

```
Algoritmo: Calculadora
Objetivo: Efetuar uma das quatro operações básicas de uma calculadora.
Início

1. Declarar as variáveis valor1, valor2, resultado:
real valor1, valor2, resultado

2. Declarar a variável opera:
string opera

3. Exibir a mensagem na tela:
Exiba "Entre com o primeiro valor:"

4. Ler a resposta do usuário e armazenar em valor1:
Ler valor1

5. Exibir a mensagem na tela:
Exiba "Entre com o segundo valor:"

6. Exibir as mensagens na tela:
Exiba "C A L C U L A D O R A"
Exiba " + para adição"
Exiba " – para subtração"
Exiba "* para multiplicação"
Exiba "/ para divisão"
Exiba "Qual a operação desejada?"
7. Ler a resposta do usuário e armazenar em opera
Ler opera

8. SE opera = "+" ENTÃO
     8.1 Efetuar o cálculo e armazene em resultado
           Resultado = valor1 + valor2
     8.2 Exibir mensagem na tela:
           Exiba "O resultado da adição é:" e resultado
SENÃO SE opera = "-" ENTÃO
     8.1 Efetuar o cálculo e armazene em resultado
           Resultado = valor1 - valor2
     8.2 Exibir mensagem na tela:
           Exiba "O resultado da subtração é:" e resultado
```

```
      SENÃO SE opera = "*" ENTÃO
      8.1 Efetuar o cálculo e armazene em resultado
            Resultado = valor1 * valor2
      8.2 Exibir mensagem na tela:
            Exiba "O resultado da multiplicação é:" e resul-
tado
        SENÃO SE opera = "/" ENTÃO
      8.1 Efetuar o cálculo e armazene em resultado
            Resultado = valor1 / valor2
      8.2 Exibir mensagem na tela:
            Exiba "O resultado da divisão é:" e resultado
               SENÃO
      8.1 Exibir na tela a mensagem:
            Exiba "Operação não disponível"
               FIM SE
        FIM SE
      FIM SE
FIM SE

Fim
```

Com certeza você poderá encontrar outra solução ou se perder ao longo do caminho. Por isso, a necessidade de realizar o teste de mesa para conferir a eficiência do sistema.

Estruturademúltiplasdecisões Você pode notar que as estruturas SE são excelentes para a tomada de decisões, mas ela se torna um pouco complicada quando há inúmeras possibilidades de escolha do usuário. Para resolver o problema de múltiplas escolhas, o ideal seria utilizar a estrutura ESCOLHA ... CASO ... SENÃO, também conhecida como estruturas SELECT CASE. Caso o sistema seja desenvolvido na linguagem C, a estrutura será conhecida como SWITCH. Vejamos a sua sintaxe:

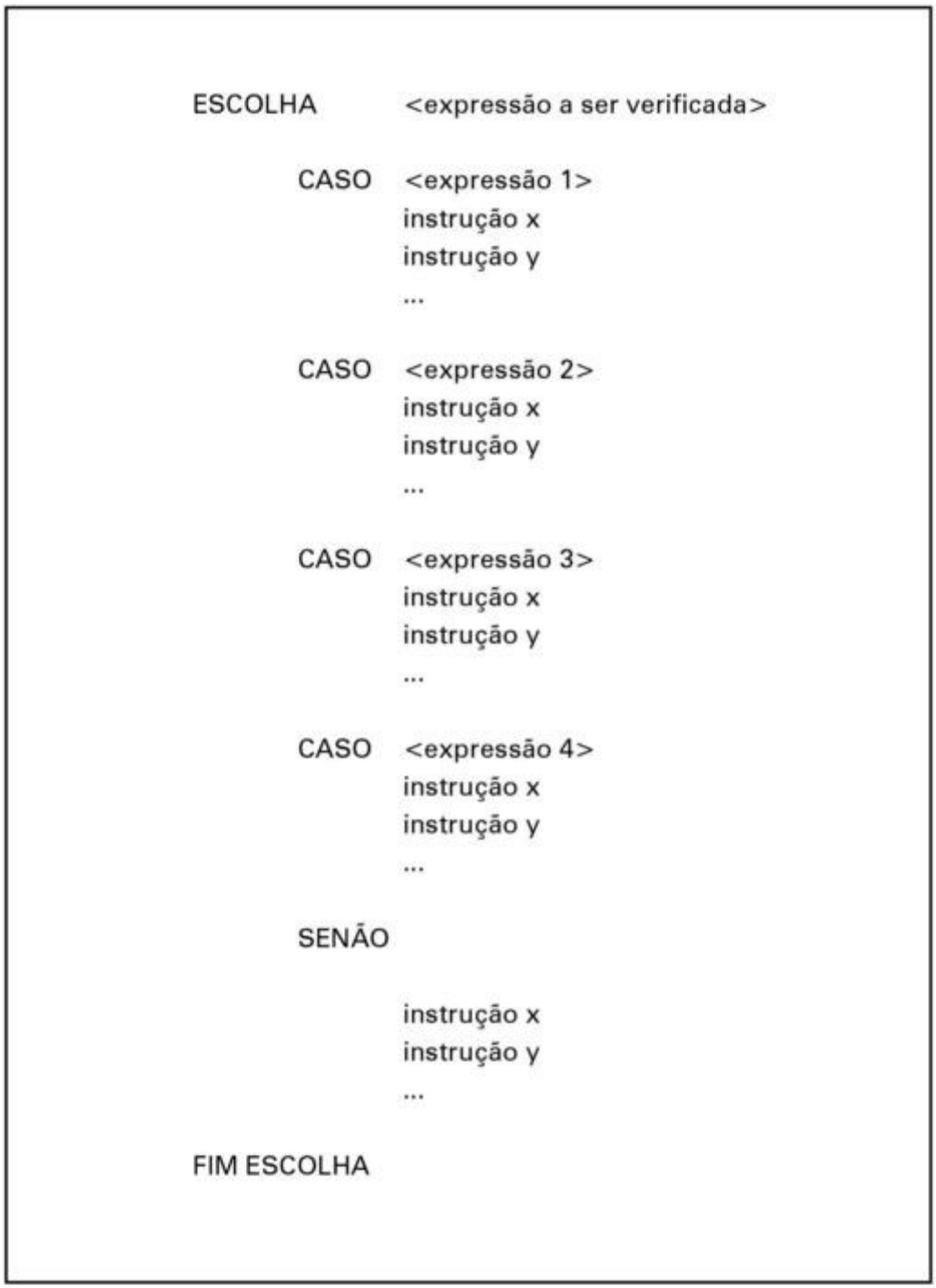


Figura 5.4.: Exemplo da estrutura do Escolha... Caso...

Ao iniciar a estrutura ESCOLHA, a expressão é avaliada e seu valor (resposta) será comparado com cada uma das opções existentes (CASO). Por exemplo, se tivermos um ESCOLHA estado (variável), assim que informarmos a resposta, o sistema procurará o caso exatamente descrito. Caso nenhuma das opções seja localizada, executará as instruções contidas em SENÃO. Vale ressaltar que essa opção (SENÃO) é opcional.

Imagine que sua empresa possui um valor de frete para cada cidade. Para isso, utilizaremos a estrutura ESCOLHA ... CASO:

```
Algoritmo: Escolha
Objetivos: Calcular o valor do frete de acordo com a cidade da entrega.

Início

1. Declarar a variável qtde:
inteiro qtde

2. Declarar as variáveis subtotal, frete e total:
real subtotal, frete, total

3. Declarar a variável cidade:
string Cidade

4. Exibir a mensagem na tela:
Exiba "Qual a quantidade de teclados desejada?"

5. Ler a resposta do usuário e armazenar em qtde
Ler qtde

6. Calcular subtotal do pedido:
Subtotal = qtde * 19.99        <preço unitário do teclado = 19,99>

7. Exibir a mensagem na tela:
Exiba "Qual a cidade da entrega?"

8. Ler a resposta do usuário e armazenar em cidade
Ler cidade
9. Escolha cidade

CASO "Curitiba"
     9.1 Calcular o frete 2% sobre o valor
          Total = (subtotal * 0.02) + subtotal
     9.2 Exibir a mensagem:
          Exiba "O total de seu pedido é:" e total
CASO "São Paulo"
     9.1 Calcular o frete 3% sobre o valor
```

```
            Total = (subtotal * 0.03) + subtotal
      9.2 Exibir a mensagem:
            Exiba "O total de seu pedido é:" e total
CASO "Rio de Janeiro"
      9.1 Calcular o frete 5% sobre o valor
            Total = (subtotal * 0.05) + subtotal
      9.2 Exibir a mensagem:
            Exiba "O total de seu pedido é:" e total
CASO "Florianópolis"
      9.1 Calcular o frete 10% sobre o valor
           Total = (subtotal * 0.1) + subtotal
      9.2 Exibir a mensagem:
            Exiba "O total de seu pedido é:" e total
CASO "Natal"
      9.1 Calcular o frete 8% sobre o valor
            Total = (subtotal * 0.08) + subtotal
      9.2 Exibir a mensagem:
            Exiba "O total de seu pedido é:" e total
FIM ESCOLHA

Fim
```

Acompanhe na tabela a seguir (Tabela 5.1), a porcentagem de desconto aplicada a um pedido de acordo com a quantidade de unidades solicitadas:

Quantidade de Peças	Desconto
Abaixo de 10 peças	0%
10	10%
20	15%
30 a 100	20%
Acima de 100 peças	25%

Tabela 5.1.: Tabela de descontos que serão uitlizados no algoritmo UsoDesSes.

Vejamos a diferença na utilização das estruturas SE aninhadas e Escolha,

cada um em seu respectivo algoritmo:

```
 Algoritmo: UsoDeSes
 Objetivo: Calcular valor de desconto de acordo com tabela.
Início

1. Declarar a variável qtde:
Inteiro qtde

2. Declarar as variáveis subtotal, desconto e total:
Real subtotal, desconto, total

3. Exibir a mensagem na tela:
Exiba "Qual a quantidade de DVDs desejada?"

4. Ler a resposta do usuário e armazenar em qtde
Ler qtde
```

5. **SE qtde < 10 ENTÃO**
    5.1 Calcular o total do pedido
        Total = (qtde * 49,99)
    5.2 Exibir a mensagem:
    Exiba "O total de seu pedido é:" e total
SENAO SE qtde >=10 e qtde < 20 ENTAO
    5.1 Calcular o subtotal do pedido:
        Subtotal = (qtde * 49,99)
    5.2 Calcular o valor do desconto de 10%
        Desconto = subtotal * 0.1
    5.3 Calcular o total do pedido
        Total = subtotal – desconto
    5.4 Exibir a mensagem:
        Exiba "O total de seu pedido é:" e total
SENÃO SE qtde = 20 ENTÃO
    5.1 Calcular o subtotal do pedido:
        Subtotal = (qtde * 49,99)
    5.2 Calcular o valor do desconto de 15%
        Desconto = subtotal * 0.15
    5.3 Calcular o total do pedido
        Total = subtotal – desconto
    5.4 Exibir a mensagem:
        Exiba "O total de seu pedido é:" e total
SENÃO SE qtde >= 30 e qtde <= 100 ENTAO
    5.1 Calcular o subtotal do pedido:
        Subtotal = (qtde * 49,99)
    5.2 Calcular o valor do desconto de 20%
        Desconto = subtotal * 0.20
    5.3 Calcular o total do pedido

```
            Total = subtotal - desconto
      5.4 Exibir a mensagem:
            Exiba "O total de seu pedido é:" e total
SENÃO
      5.1 Calcular o subtotal do pedido:
            Subtotal = (qtde * 49,99)
      5.2 Calcular o valor do desconto de 25%
            Desconto = subtotal * 0.25
      5.3 Calcular o total do pedido
            Total = subtotal - desconto
      5.4 Exibir a mensagem:
            Exiba "O total de seu pedido é:" e total
FIM SE
FIM SE
       FIM SE
      FIM SE

Fim
```

0 mesmo problema utilizando a estrutura ESCOLHA ... CASO:

```
Início

1.   Declarar a variável qtde:
Inteiro qtde

2.   Declarar as variáveis subtotal, desconto e total:
Real subtotal, desconto, total

3.   Exibir a mensagem na tela:
Exiba "Qual a quantidade de DVDs desejada?"

4.   Ler a resposta do usuário e armazenar em qtde
Ler qtde

5.   ESCOLHA qtde
Total = (qtde * 49,99)
```

```
CASO < 10
5.1 Exibir a mensagem:
      Exiba "O total de seu pedido é:" e total

CASO >= 10 e < 20
5.1 Calcular o valor do desconto de 10%
      Desconto = subtotal * 0.1
5.2 Calcular o total do pedido
      Total = subtotal - desconto
5.3 Exibir a mensagem:

      Exiba "O total de seu pedido é:" e total
CASO = 20 E < 30
5.1 Calcular o valor do desconto de 15%
      Desconto = subtotal * 0.15
5.2 Calcular o total do pedido
      Total = subtotal - desconto
5.3 Exibir a mensagem:
      Exiba "O total de seu pedido é:" e total
CASO >= 30 e < 100
5.1 Calcular o valor do desconto de 20%
      Desconto = subtotal * 0.2
5.2 Calcular o total do pedido
      Total = subtotal - desconto
5.3 Exibir a mensagem:
      Exiba "O total de seu pedido é:" e total
     SENÃO
5.1 Calcular o valor do desconto de 25%
      Desconto = subtotal * 0.25
5.2 Calcular o total do pedido
      Total = subtotal - desconto
5.3 Exibir a mensagem:
      Exiba "O total de seu pedido é:" e total
    FIM ESCOLHA

Fim
```

Comparando as duas estruturas com várias opções, fica claro que o ideal é utilizar a estrutura ESCOLHA ... SENÃO, visto que ela é mais simples do que aninhar os SE.

Capítulo 6

Estruturas de repetição (loop)

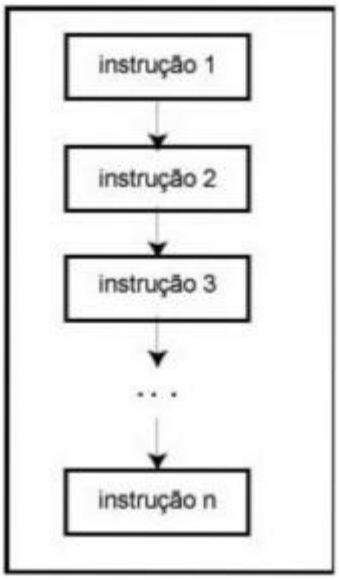
Pudemos observar que as estruturas vistas até o momento para a construção de algoritmos indicam instruções sequenciais e de seleção, ou de tomada de decisões.

Muitas vezes, tais estruturas devem ser combinadas com as estruturas de repetição a fim de criar algoritmos mais eficientes em busca de soluções para problemas mais complexos.

Uma estrutura de repetição, como o próprio nome indica, permite a execução de uma sequência de instruções repetidas vezes até que determinada condição seja avaliada como verdadeira, por um número específico de vezes ou para cada item (elemento) em uma coleção, ou seja, utilizando acumuladores e contadores. Tais estruturas também são conhecidas como de loop ou de laço.

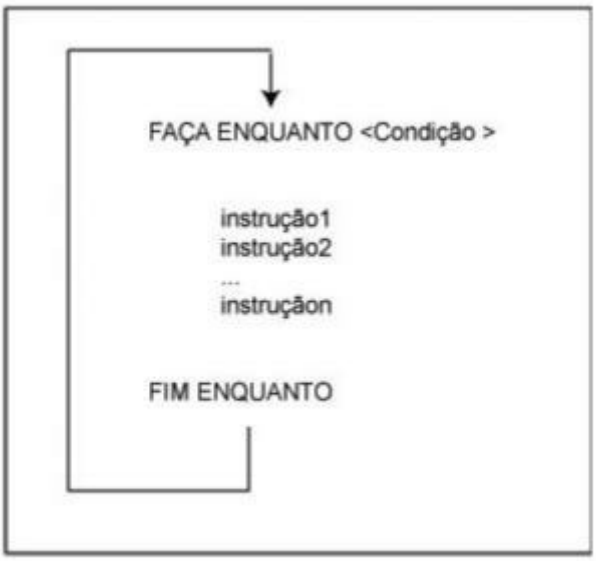
Um bom exemplo de utilização de acumuladores e contadores é a entrada de senhas, em que, após três tentativas, o acesso será bloqueado. Assim como as estruturas de tomada de decisão, todas as possibilidades devem ser avaliadas em testes de mesa.

A maior parte das linguagens de programação costuma definir três tipos de estruturas de repetição: • Faça enquanto • Enquanto Faça • Para... Faça Nas estruturas sequênciais, onde as instruções são executadas uma após a



outra, encontramos o seguinte fluxograma:

Estrutura Faça Enquanto Nesta estrutura, a rotina só será executada caso a condição estipulada seja avaliada como verdadeira. Após entrar na estrutura, o sistema executará todas as instruções contidas na sequência existente dentro do Faça Enquanto e Fim Enquanto. Sua sintaxe é:



Nesta estrutura, a condição deverá ser avaliada em seu início. Ao localizar a instrução Fim Enquanto a rotina retornará ao início da estrutura para avaliar novamente a condição. Se avaliada novamente como verdadeira, continuará a repetição.

Esse tipo de estrutura é recomendado quando o número de repetições das

instruções é desconhecido, pois, como já verificamos, a condição será avaliada sempre no início, permanecendo em um loop até que seja avaliada como falsa.

Para compreender melhor, vejamos o seguinte exemplo:

```
1. Valor = 1

2. Faça enquanto valor <=10

     2.1 Valor = 2
     2.2 Exiba valor

3. Fim enquanto

4. Exiba valor
```

Ao iniciar o algoritmo, a variável valor recebe o valor 1, por isso, irá à próxima instrução (2) Faça enquanto. Neste caso, como a variável recebeu o valor 1 (que é menor ou igual a 10), entrará em loop. Agora, já dentro do loop, a mesma variável recebe o valor 2 e é exibida na tela. A condição é testada novamente e verifica que a condição continua sendo verdadeira, por isso, o loop é repetido novamente.

Você pode reparar que esse loop será contínuo, isto é, o valor sempre será 2 e exibido na tela. Para que o mesmo seja interrompido, será necessário utilizar as teclas Ctrl + Break, a tecla Esc ou as teclas Ctrl + C, dependendo da linguagem a ser utilizada, ou uma instrução que fará com que a condição seja avaliada como falsa, portanto, o ideal é que as instruções possuam a estrutura:

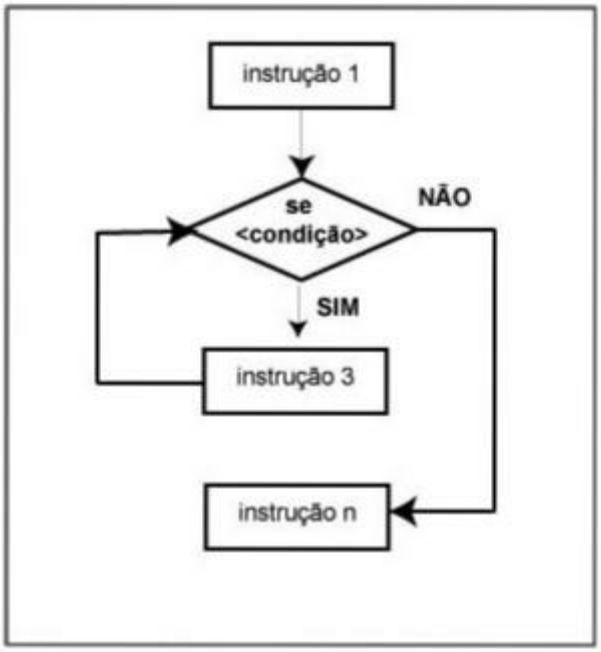
```
1. Valor = 1

2. Faça enquanto valor <=10

     2.1 Valor = Valor + 1
     2.2 Exiba valor

3. Fim enquanto

4. Exiba valor
```

Ao iniciar o algoritmo, a variável valor recebe o valor 1, portanto poderá entrar no loop (repetição). Em seguida, calculará o valor atual e adicionará 1 a seu valor, como em um acumulador. Nesse caso, o algoritmo exibirá o

conteúdo da variável valor como 2, portanto, estará ainda dentro do loop. Novamente, acrescenta-se 1 à variável valor, e seu conteúdo é exibido na tela. 0 algoritmo repetirá essas etapas sucessivas vezes, até que a variável valor contenha um valor acima de 10, o que provocará a saída do loop. Acompanhe na Tabela 6.1 os resultados apresentados enquanto em loop:

Tentativa	Valor armazenado na variável	Resultado
1	Valor = 1	1
2	Valor = valor + 1	2
3	Valor = valor + 1	3
4	Valor = valor + 1	4
5	Valor = valor + 1	5
6	Valor = valor + 1	6
7	Valor = valor + 1	7
8	Valor = valor + 1	8
9	Valor = valor + 1	9
10	Valor = valor + 1	10
11	Valor = valor + 1	11 – Sairá do loop

Tabela 6.1.: Resultados Em estruturas de repetição, encontramos o seguinte exemplo de fluxograma:



É importante lembrar que devemos ter a inicialização da variável que faz parte da expressão que controlará a repetição, visto que, para um comando de leitura poder entrar na estrutura de repetição, ela precisa possuir um valor válido e também precisa ser modificada para que a execução saia da mesma estrutura (caso contrário, entrará em um loop infinito). Veja como isso deve ser estruturado em um algoritmo:

```
Leia variável
Faça enquanto <declaração verdadeira>

Instruções a repetir
Leia a variável

    Fim enquanto
```

Vejamos outro exemplo:

```
Algoritmo Tabuada
Objetivo: Verificar uso da estrutura de repetição Faça en-
quanto ... Fim Enquanto
```

Início

1. Declarar as variáveis:
inteiro contador, resultado

2. Armazenar valor em contador:
contador = 1

3. Iniciar a estrutura de repetição (testar condição)
FAÇA ENQUANTO contador <=50

3.1 Armazenar valor em resultado:
resultado = contador * 5

3.2 Exibir a mensagem e valor de resultado:
Exiba contador e "* 5 = " e resultado

3.3 Acumular valor ao contador:
contador = contador + 1

4. Final da estrutura de repetição (condição foi avaliada como falsa)
FIM ENQUANTO
Fim

Tabela de resultados:

Tentativa	Valor da variável contador	Valor da variável resultado	
1	1	5	Entrada no loop
2	2	10	
3	3	15	
4	4	20	
5	5	25	
6	6	30	

7	7	35	
8	8	40	
9	9	45	
10	10	50	
11	11	55	Sair do loop

Tabela 6.2.

Repare que a característica principal dessa estrutura é que, se o resultado da <condição> for avaliada como FALSA pela primeira vez, a instrução ou o bloco verdade de instruções não será executado.

A maioria das linguagens de programação reconhece a estrutura Enquanto Faça ... Fim Enquanto como a estrutura Do While. Veja um exemplo de sua sintaxe utilizando a linguagem C:

```
Do While (condição a ser analisada)
{

     Instrução 1;
     Instrução 2;
     Instrução 3;
     ...
     Instrução n;

}
```

Nesse caso, a estrutura será repetida enquanto (While) a condição for verdadeira. No momento em que for avaliada como FALSA, o controle do fluxo de informações será desviado para fora da estrutura.

A estrutura de repetição Faça Enquanto ... Fim enquanto pode ser utilizada com as estruturas de decisão, o que aliás, é bastante comum. Por exemplo:

A professora de português da 5a série precisa calcular a média aritmética do primeiro semestre de 2009. Para isso, os 50 (cinquenta) alunos devem informar as notas referentes ao primeiro e segundo bimestre. Caso a média seja maior ou igual a 7.0, o aluno será aprovado; quando for maior que 5.0

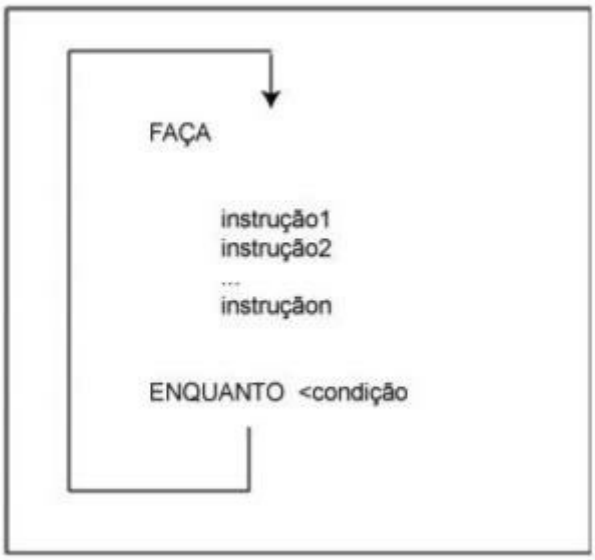
e menor que 7.0, ele ficará em recuperação; caso contrário (inferior a 5.0), será automaticamente reprovado.

```
Início

     1. Declarar variáveis:
       inteiro contador
       real nota1, nota2, média

     2. Iniciar contador (armazenar valor na variável)
       contador = 0

     3. Iniciar estrutura de repetição (verificação da variá-
vel)
       FAÇA ENQUANTO contador < 50

       3.1 Exibir mensagem:
             Exiba "Informe a primeira nota:"
       3.2 Ler valor armazenado na variável:
             Ler nota1
       3.3 Exibir mensagem:
             Exiba "Informe a segunda nota:"
       3.4 Ler valor armazenado na variável:
             Ler nota2
       3.5 Calcular média:
             média = (nota1 + nota2) / 2
       3.6 Exibir a média na tela:
             Exiba média
       3.7 SE média >= 7.0 ENTÃO
       3.7.1 Exibir a mensagem:
             Exiba "Parabéns, você foi aprovado"

             SENÃO

                   SE média >=5.0 e média <7.0 ENTÃO
                         3.7.1 Exibir mensagem:
                               Exiba "Infelizmente você
está em recuperação!"
                   SENÃO
                         3.7.1 Exibir mensagem:
                               Exiba "Mais esforço, você
foi reprovado!"
                   FIM SE

             FIM SE
```

```
        3.8 Incrementar a variável contador:
              contador = contador + 1

        3.9 Finalizar estrutura de repetição:
              FIM ENQUANTO

Fim
```

Estrutura Faça... Enquanto <condição> Outra forma de utilização da estrutura de repetição é analisar a variável ao final dela, procedimento muito comum quando não sabemos, antecipadamente, o número de vezes que as instruções devem ser repetidas. Por meio dessa estrutura, podemos saber, pelo menos, se essa execução ocorrerá ou não, independente do resultado da condição.

Sua estrutura é parecida com a anterior, sendo que a diferença ficará por conta da verificação da condição, que será executada em outro momento.




```
Algoritmo Tabuada2
Objetivo: Verificar uso da estrutura de repetição Faça ... enquanto
```

```
Início

    1. Declarar as variáveis:
      inteiro contador, resultado

    2. Armazenar valor em contador:
      contador = 1

    3. Iniciar a estrutura de repetição (testar condição)
      FAÇA

      3.1 Armazenar valor em resultado:
            Resultado = contador * 5

      3.2 Exibir a mensagem e valor de resultado:
            Exiba contador e "* 5 = " e resultado
      3.3 Acumular valor ao contador:
            contador = contador + 1

    4. Final da estrutura de repetição (condição foi avaliada como falsa)
      ENQUANTO contador <= 50

Fim
```

Um sapateiro resolveu verificar os consertos realizados durante uma semana e identificar quantos sapatos consertou, bem como se os serviços foram feitos em sapatos masculinos ou femininos. Para isso, são necessárias três variáveis de controle, um contador (para saber o número total de sapatos), um contador de sapatos masculinos e outro para os femininos. Além disso, ele deseja obter os resultados em porcentagem. Veja como resolver este algoritmo:

```
Algoritmo: Sapateiro
Objetivo: Executar estrutura de repetição juntamente com a estrutura Escolha.
```

Início

1. Declarar variáveis:
caracter tipo
inteiro cont; feminino; masculino
<contador de sapatos; masculinos e femininos>

real totSapatos; totFeminino; totMasculino
<armazena total em porcentagem dos tipos de sapatos consertados>

2. Armazenar valor em variáveis (zerar variáveis)
cont = 0
feminino = 0
masculino = 0

3. Iniciar estrutura de repetição:
FAÇA

3.1 Exibir a mensagem:
Exiba "**Informe o tipo de sapato M (masculino) ou F (feminino):**"

3.2 Armazenar resposta em tipo:
Leia tipo

3.3 Iniciar estrutura de escolha:

ESCOLHA tipo

CASO = "F"

3.3.1 Incrementar variável do tipo feminino:

feminino = feminino + 1

3.3.2 Incrementar variável com contador de sapatos:

cont = cont + 1

CASO = "M"

3.3.1 Incrementar variável do tipo masculino:

masculino = masculino + 1

3.3.2 Incrementar variável com contador de sapatos:

cont = cont + 1

3.4 Finalizar estrutura escolha:

FIM ESCOLHA

3.5 Finalizar estrutura de repetição (analisar se variável é verdadeira):

ENQUANTO TIPO <> " "

```
    4. Verificar se contador é maior que 0.
      SE cont >0 ENTÃO

      4.1 Calcular percentual sapatos femininos:
            totFeminino = (feminino*100) / cont

      4.2 Calcular percentual sapatos masculinos
            totMasculino = (masculino*100) / cont

      4.3 Exibir a mensagem:
            Exiba "Porcentagem de sapatos femininos:" e
totfeminino

      4.4 Exibir a mensagem:
            Exiba "Porcentagem de sapatos masculinos:" e
totMasculino

      SENÃO

      4.1 Exibir a mensagem:
            Exiba "Nenhum conserto foi realizado"

      FIM SE

Fim
```

Exemplo da utilização da estrutura Faça Enquanto utilizando a linguagem VBA ou Visual Basic:

```
Dim Cont As Integer
'Declarada a existência da variável cont do tipo inteiro

Dim QuebraLinha as String
'Declarada a existência da variável Quebra de linha como
tipo cadeia de caracteres.

Cont = 0
'Atribuir valor à variável de contador

QuebraLinha = Chr(13) & Chr(10)
'Atribuir cadeia de caracteres do teclado para quebra de
linha manual.
```

```
While Cont < 20
'Inicializada a estrutura de repetição FAÇA ENQUANTO
'Condição verificada no início da estrutura
'Fazer enquanto contador é menor que 20.

Cont += 1
'Incrementar uma unidade ao contador

TextBox1.Text += Cont & Quebra
'Na caixa de texto (TextBox1) exiba o número armazenado na
variável contador e faça uma quebra de linha.

End While
'Finalizada a estrutura de repetição (valor da variável foi
acrescido e atingiu o valor 21.
```

No Visual Basic e em várias outras linguagens de programação, encontramos a estrutura de repetição Do ... Loop, que permite a verificação da condição no início da estrutura e seu funcionamento é similar à estrutura While. A diferença reside no fato de que na estrutura While, a cada passagem pelo Loop (repetição), a condição será avaliada. Já na estrutura Do ... Loop, a condição poderá ser avaliada ao iniciar ou terminar o loop. Havendo a necessidade de interromper o Loop, basta inserir a instrução Exit Do.

Vejamos o mesmo código descrito na linguagem Visual Basic, mas utilizando a estrutura Do While com estrutura de tomada de decisão:

```
'Declarar as variáveis:
Dim Cont As Integer
Dim QuebraLinha As String

'Atribuir valores às variáveis:
Cont = 0
QuebraLinha = chr(13) & chr(10)
```

No Visual Basic no momento da declaração da variável, é possível atribuir um valor a ela, sem a necessidade de incluir 4 linhas de instruções. Para isso, basta utilizar:

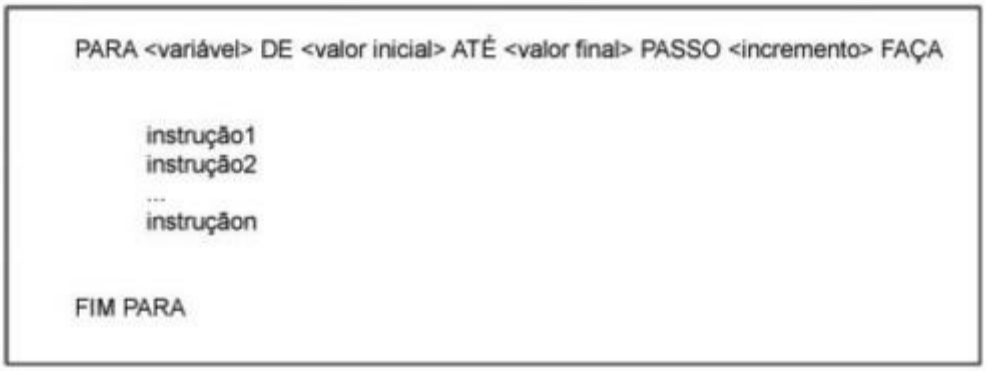
```
Dim Cont as integer = 0
Dim QuebraLinha As String = Chr(13) & Chr(10)
‘Inicializar estrutura de repetição com estrutura de tomada
de decisão:
 Do While Cont < 20
Cont += 1

If Cont = 10 Then
‘Contador atingiu valor 10 então deve sair do loop:
Exit Do
End If

TextBox1.Text += Cont & QuebraLinha

Loop
‘Repita as instruções existentes no laço.
```

Para verificar a condição ao final do Loop em linguagem Visual Basic:

```
Dim Cont As Integer = 0
Dim QuebraLinha As String = Chr(13) & Chr(10)
        Do
            Cont += 1
            If Cont = 10 Then
                Exit Do
            End If
            TextBox1.Text += Cont & QuebraLinha
        Loop Until Cont < 20
```

Estrutura Para... Passo... Faça Este tipo de estrutura permite controlar o número de vezes que as instruções devem ser repetidas. Portanto, sua principal característica é saber previamente o número de vezes que a estrutura será repetida, como, por exemplo, antes de bloquear um cartão magnético, o usuário terá a oportunidade de digitar sua senha três vezes.

Sua sintaxe é:



Nesse tipo de estrutura de repetição, temos: 1) Após a declaração da variável, um valor inicial é definido para ela.

2) Esse valor inicial será comparado ao valor final.

3) Caso a variável seja menor ou igual ao valor final, serão executadas as instruções existentes dentro da rotina de repetição.

a) incrementa-se o valor da variável em uma unidade; b) verifica-se novamente se variável está dentro dos limites em que se repete a tarefa de comparação da variável, e se ela é verdadeira.

4) Caso a variável contenha um valor maior que o declarado como valor final, será executada a instrução, ou conjunto de instruções, logo abaixo da instrução de finalização da repetição (FIM PARA).

Caso não seja verificada a opção PASSO, o contador será incrementado automaticamente em uma unidade. Portanto, contador terá seu passo determinado com avanço de uma em uma unidade.

Na maior parte das linguagens de programação, esse tipo de estrutura é conhecida como estrutura For i to n.

Para exemplificar, vejamos novamente um algoritmo com o cálculo da tabuada:

```
Algoritmo: PassoaPasso
Objetivo: Verificar a estrutura Para
```

```
Início

1. Declarar variáveis com contador e resultado:
     inteiro contador; resultado

2. Inicializar a variável contador com uma unidade:
     contador = 1

3. Iniciar estrutura de repetição:
     PARA contador DE 1 ATÉ 50 FAÇA

     3.1 Calcular a expressão:
              resultado = contador * 5

     3.2 Exibir o conteúdo das variáveis:
              contador e " x 5 = " e resultado

     3.3 Finalizar a estrutura de repetição:
              FIM PARA

Fim
```

Verifique, no algoritmo a seguir, como calcular a soma dos números pares de 100 a 200, determinando se o número é par ou não:

```
Início

1. Declarar variáveis:
     inteiro soma; i

2. Inicializar variável soma:
     soma = 0

3. Inicializar estrutura de repetição:
     PARA i DE 100 ATÉ 200 FAÇA

     3.1 Iniciar estrutura de repetição que verifica se valor
é par:
       SE (i % 2 = 0) ENTÃO

       3.1.1 Calcular a soma:
                    soma = soma + i
```

```
      3.2 Finalizar a estrutura que verifica se valor é par:
        FIM PARA

4. Exibir o resultado da soma dos pares e mensagem:
     Exiba "A soma dos números pares de 100 até 200 é:" e
soma

Fim
```

Inicialmente, a variável i recebe o valor 100 e deverá chegar até 200 (instrução PARA i DE 100 ATÉ 200 FAÇA). Esse valor inicialmente é comparado com o valor final (200), se menor, executa a instrução existente dentro do laço, que incrementa uma unidade ao contador (como o passo não foi determinado, o contador compreende que o próximo passo é uma unidade).

Essas etapas (incrementar uma unidade ao contador e verificar se o mesmo é par, se repetirá até que o valor atinja o valor de 201, desta forma a variável é analisada como falsa, pois o algoritmo determinou de 100 até 200. A cada incremento, a variável soma é utilizada para somar os valores de 100 a 200, mas somente quando eles forem pares.

Para que esse algoritmo funcione mais rápido, podemos incrementar o contador com passo de duas em duas unidades:

```
Início

      inteiro soma; i
      soma = 0
      PARA i DE 100 ATÉ 200 PASSO 2FAÇA
        SE (i % 2 = 0) ENTÃO
                     soma = soma + i
        FIM PARA
        Exiba "A soma dos números pares de 100 até 200 é:" e
soma

Fim
```

Imagine o que deveria ter sido feito caso as estruturas de repetição não fossem encontradas:

```
Início
inteiro número

Exiba "Informe um número:"
Ler número
Exiba "Informe um número:"
Ler número
Exiba "Informe um número:"
Ler número
Exiba "Informe um número:"
Ler número
Exiba "Informe um número:"
Ler número
Exiba "Informe um número:"
Ler número
Exiba "Informe um número:"
Ler número
Exiba "Informe um número:"
Ler número
Exiba "Informe um número:"
Ler número
Exiba "Informe um número:"
Ler número
Exiba "Informe um número:"
Ler número
Fim
```

Veja que houve a necessidade de apresentar as mesmas instruções por dez vezes. Agora acompanhe o uso de contadores e estrutura de repetição:

```
Início
inteiro número

PARA número = 1 ATÉ 10 FAÇA

     Exiba "Informe um número:"
     Ler número

FIM PARA
Fim
```

Assim como as demais estruturas de tomada de decisão, as de repetição também podem ter outras estruturas aninhadas (encadeadas) a ela.

Capítulo 7

Vetores e matrizes

Anteriormente vimos que, utilizando as estruturas de repetição, é possível armazenar a entrada de vários dados no que chamamos de variáveis de memória. Mas, podemos perceber que ao inserir rodar o programa novamente, a variável é descarregada, ou seja, valor armazenado é "jogado" fora e nova informação será armazenada no mesmo local, como em uma pequena gaveta. Um objeto somente poderá ser guardado, quando o anterior for retirado.

Imagine poder armazenar em uma mesma variável, diversos valores e os mesmos identificados pelo nome desta e por números. Isso é possível ao utilizar estruturas homogêneas conhecidas como vetores e matrizes ou, como na maioria das linguagens de programação, arrays, variáveis indexadas, variáveis compostas, arranjos e tabelas em memória.

vetores

Vetor é a organização de elementos a serem armazenados na memória do computador, recebendo o mesmo nome.

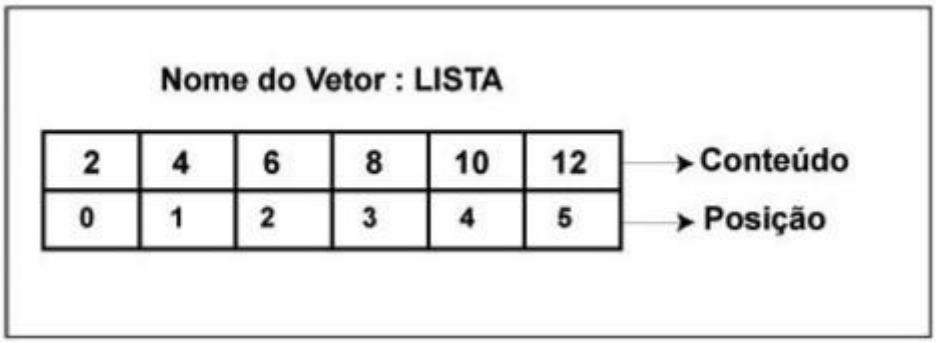
Para facilitar a compreensão, façamos uma analogia entre uma gaveta, onde somente um objeto pode ser guardado, e uma variável de memória. Agora imagine que esta gaveta possui algumas divisões, onde cada divisão receberá um número que irá determinar a posição dentro da gaveta. Assim, para identificarmos um determinado objeto, é importante informar sua posição na gaveta através deste número.

Portanto, podemos dizer que um vetor é um conjunto de variáveis, onde cada uma delas poderá armazenar valores (informações) diferentes, mas todas possuem o mesmo nome.

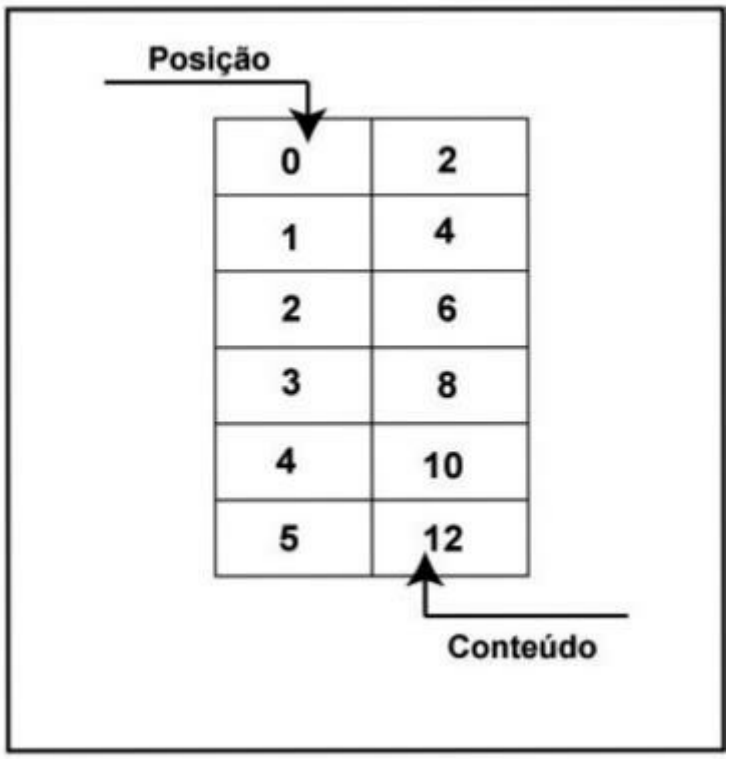
Para identificar a posição da informação dentro de um vetor, devemos

associar alguns índices ao seu nome, como se fossem o número da linha onde o dado está armazenado, como uma representação virtual. Desta forma, estaremos individualizando todos os elementos que representam o conjunto destas informações.

Veja um exemplo de vetor com o nome Lista:



Nada melhor do que representar um vetor como uma coluna com várias linhas:



0 vetor nomeado como nome Lista possui o seguinte conteúdo:
$Lista_0 = 2$ $Lista_1 = 4$ $Lista_2 = 6$ $Lista_3 = 8$ $Lista_4 = 10$ $Lista_5 = 12$

Em algoritmos tais conteúdos são representados por:

Lista[0] = 2 Lista[1] = 4 Lista[2] = 6 Lista[3] = 8 Lista[4] = 10 Lista[5] = 12

Ou:

Lista[0]:= 2 Lista[1]:= 4 Lista[2]:= 6 Lista[3]:= 8 Lista[4]:= 10 Lista[5]:= 12

Por padrão, dizemos que um vetor é unidimensional, pois possui somente um índice.

Manipulaçãodevetores Dimensionar e declarar vetores Para dimensionar um vetor, devemos declarar a variável da seguinte forma: Tipo do vetor <nome do vetor> [dimensão do vetor]

Onde:

• Tipo do vetor pode abranger inteiro, real ou string (matriz) • Nome do vetor é com as mesmas regras para nomeação de variáveis.

• Dimensão é o número de elementos existentes no vetor. 0 vetor Lista possui seis elementos.

Algumas linguagens declararam vetores com a seguinte sintaxe: ***Var <nome>:MATRIZ[<COLUNA_INICIAL> .. <COLUNA_FINAL>] DE tipo***

Por exemplo:

Lista:Matriz[6..1] de Inteiros

ou

Lista:Matriz[6] de inteiros

Algumas linguagens costumam atribuir instruções à um vetor da seguinte forma:

<nome_do_vetor>:=<expressão>

Além do nome da variável é necessário indicar qual o índice onde a informação está armazenada (posição) no vetor: Lista[2J: =100

Veja como referenciar objetos do tipo vetor em Javascript:

```
var Lista = new Array();
Lista[0] = 2;
Lista[1] = 4;
Lista[2] = 6;
Lista[3] = 8;
Lista[4] = 10;
Lista[5] = 12;
```

ou

```
var Lista = new Array(2, 4, 6, 8, 10, 12)
```

Leituradevetores Para escrever na tela o conteúdo existente na posição 4 do vetor com o nome Lista, devemos entrar com a seguinte instrução: ***LER <nome_do_vetor> [índice]***

Ler Lista[2]

Não é possível manipular todos os elementos de um vetor ao mesmo tempo, mas sim, cada um deles isoladamente.

Armazenar informação em um vetor Para armazenar dados (informações) em um vetor é necessária a utilização de uma estrutura de repetição, pelo simples fato de um vetor possuir várias posições onde cada uma delas deverá ser preenchida.

Exemplo de definição de um vetor sem utilização de uma estrutura de repetição:

```
Início
1. Declarar o vetor de cinco elementos com nome Valor:
     Valor:matriz[5] de inteiros;

2. Ler o conteúdo de cada um dos elementos do vetor:
     Valor[0]:=leia();
     Valor[1]:=leia();
     Valor[2]:=leia();
     Valor[3]:=leia();
     Valor[4]:=leia();

3. Exibir na tela os valores digitados:
Exiba "valores digitados:", Valor[0], "," , Valor[1], "," ,
Valor[2], "," , Valor[3], "," , Valor[4];

<Ou se preferir digite:>

Imprima("valores digitados:", Valor[0], "," , Valor[1], "," ,
Valor[2], "," , Valor[3], "," , Valor[4]);

Fim
```

Utilizando uma estrutura de repetição, teremos:

```
Início
1. Declarar o vetor de cinco elementos com nome Valor:
     Valor:matriz[5] de inteiros;
     Inteiro i    <ou i:inteiro>

2. Iniciar rotina de repetição:
     PARA i de 0 até 4 FAÇA
       Valor[i]:=leia();
     FIM PARA

3. Exibir valores na tela:
     Imprima("valores digitados:", Valor[0], "," , Valor[1],
"," , Valor[2], "," , Valor[3], "," , Valor[4]);

Fim
```

A estrutura deverá ser definida com a seguinte sintaxe:

```
Para nome_contador de índice_inicial até índice_final Faça

     Ler Lista[x]
     <ou>
     Lista[x]:=leia()

Fim Para
```

Veja no algoritmo abaixo como obter a nota de 20 alunos e calcular a média aritmética geral:

```
Início

1.   Declarar as variáveis com a soma (total) e média das notas (média) e um contador (acumulador):
Real total; média
Inteiro contador

2.   Armazenar conteúdo nas variáveis (zerar contadores):
Total = 0
Média = 0
Contador = 0

3.   Iniciar estrutura de repetição para ler a nota de 20 alunos:
Para x de 1 até 20 Faça

4.   Ler a nota de cada aluno:
Ler nota[x]

5.   Calcular a soma das notas digitadas:
Total = total + nota[x]

6.   Interromper estrutura de repetição:
Fim Para

7.   Calcular média geral dos alunos:
Média = total / 20

8.   Criar estrutura de repetição para informar número de alunos acima da média:
```

Para n de 1 até 20 Faça

SE nota[n] > média Então

Contador = contador + 1

FIM SE

Fim Para
9. Exibir conteúdo da variável contador:
Exiba contador

Fim

Para exemplificar a utilização de vetores, veja o algoritmo que permite a entrada de 15 nomes e exibe os mesmos na tela:
Início

1. Declarar variável de vetor:
String nomes[15]

2. Declarar variável inicial:
Inteiro i

3. Criar a estrutura de repetição que solicita nome e armazena no vetor:
Para i de 0 até 14 Faça

Exiba "O nome" , i + 1 , " é: "
Ler nomes[i]

Fim Para

4. Criar a estrutura de repetição para exibir nomes na tela:
Para i de 0 até 14 Faça

Exiba nomes[i]

Fim Para

Fim

Ordenandovetores É muito comum a utilização de vetores para a

classificação de todos os dados existentes. Por isso, é preciso verificar passo a passo todas as instruções a serem realizadas no algoritmo. Para isso iremos criar uma lista de alunos: 1. Declarar a variável de vetor e uma variável auxiliar.

2. Ler os nomes existentes na lista de alunos.

3. Se o nome existente na posição 0 (11 posição) aparecer depois do nome existente na posição 1 (21 posição), então os mesmos devem ter suas posições invertidas. Caso contrário, ambos devem permanecer na mesma posição.

Posição	Conteúdo
0	Miguel
1	Rafael
2	Carolina
3	Fernanda
4	Alexandre

Para exemplificar, temos uma lista com os seguintes nomes de alunos:

Tabela 7.1.: Lista com nome de alunos.

No exemplo anterior, o nome da posição 0 é Miguel e o da posição 1 é Rafael, portanto, devem permanecer na mesma posição.

Posição	Conteúdo	Variável auxiliar	Nova ordem
0	Miguel	Miguel	Carolina
1	Rafael		Rafael
2	Carolina		Miguel
3	Fernanda		Fernanda
4	Alexandre		Alexandre

4. Se o nome da posição 0 (Miguel) deve aparecer após o nome existente na posição 2 (Carolina), então é necessário invertê-los. Para isso, será necessário armazená-lo na variável auxiliar e logo a seguir inverter a sua ordem:

Tabela 7.2.

5. Agora o algoritmo deverá ler o novo nome da posição 0 (Carolina) e compará-lo com o existente na posição 3, ou seja, a partir deste ponto, cada elemento deverá ser comparado com o existente na posição 0 e invertido quando necessário ou mantido quando não for necessário o processo. Desta forma, após todas as comparações em trocas, teremos na posição 0 o nome Alexandre.

6. Após inserir Alexandre na primeira posição (0), deve-se comparar o nome da posição 1 (logo abaixo) um a um com os demais elementos, repetir o mesmo processo utilizado para armazenar na variável auxiliar e realizar a inversão ou não do seu conteúdo.

Veja que nesse caso, também é necessária uma rotina de repetição que compara cada elemento existente no vetor com o seu sucessor (elemento na próxima posição).

Portanto, o trecho de repetição da ordenação deverá ter a seguinte

```
PARA i = 0 Até 4 FAÇA

      PARA e < i + 1 até 3 FAÇA

        SE nomes[i] > nomes[e] ENTÃO
               Auxiliar < nomes[i]
               Nomes[i] < nomes[e]
               Nomes[e] < auxiliar
        FIM SE

      FIM PARA
```

estrutura: `FIM PARA`

Para melhor exemplificar, veja como ordenar de forma crescente (ascendente) um vetor de nome Alunos com cinco elementos através do seguinte algoritmo:

`Início`

1. Declarar as variáveis: i (contador inicial), c (compara contador), aux (variável auxiliar), Alunos (vetor com 5 posições):

```
Inteiro i, c
String Alunos[5], aux
```

2. Iniciar estrutura de repetição que solicita o nome e armazena no vetor:

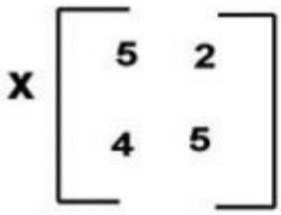
```
PARA i=0 ATÉ  4 FAÇA
              Exiba "Digite o nome: "
              Leia Alunos[i]
FIM PARA
```

3. Iniciar estrutura de repetição que lê o nome da posição inicial, compara com o próximo e se necessário altera sua posição e em seguida repete a instrução com os demais nomes existentes no vetor:

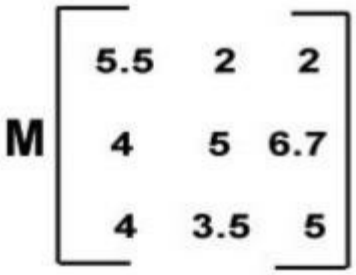
```
        PARA i = 0 ATÉ i = 4 FAÇA
        <irá comparar somente a partir do segundo elemento do
vetor>

                PARA c = i+1 ATÉ c<=4 FAÇA
                        SE Alunos[i] > Alunos[c] ENTÃO

                        aux = Alunos[i]    <armazena 1ª posição
na variável auxiliar>
                        Alunos[i] = Alunos[c]
                        Alunos[c] = aux <troca as posições das
variáveis>

                        FIM SE

                FIM PARA

        FIM PARA

        PARA i = 1 ATÉ 5 FAÇA
                Exiba "\n", i+1, " - ", Alunos[i]
                <avança o prompt do cursor, exibe a posição e
nome do aluno>

                Exiba "\n"
                <avança o prompt do cursor em uma linha>

        FIM PARA

Fim
```

Matriz

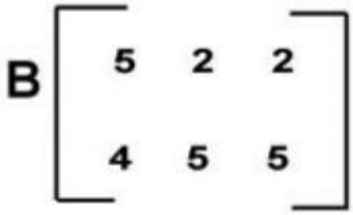
Como podemos observar, um vetor é como uma variável que armazena vários elementos na memória do computador, onde todos compartilham o mesmo nome. Agora veremos o conceito de matriz, a qual também é uma faixa com vários elementos. Na verdade, é considerada como uma tabela que servem de base para vários cálculos. Assim, como os vetores, os dados existentes em uma matriz, devem ser do mesmo tipo e todos eles referenciados por um número (índice) que determina sua posição dentro da matriz.



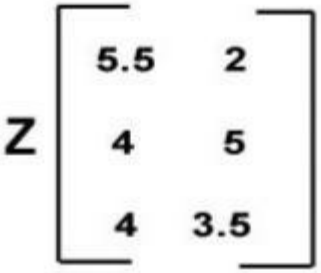
Matriz X com duas linhas e duas colunas



Matriz M com três linhas e três colunas



Matriz B com duas linhas e três colunas

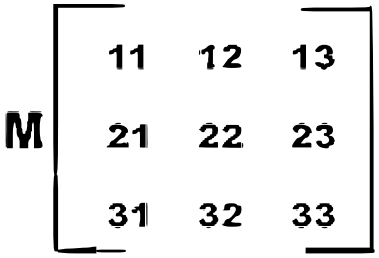


Matriz Z com três linhas e duas colunas

Até aqui, nada de diferente de um vetor. A diferença fundamental é que um vetor pode possuir um único nível ou linha (unidimensional) e uma matriz possui mais de um nível, por isso é considerada como multidimensional.

Há matrizes retangulares e quadradas. Quando o número de linhas é diferente do número de colunas, uma matriz é considerada como retangular. Caso o número de linhas e colunas seja idêntico, a matriz é quadrada.

Assim como na Matemática, uma matriz é considerada como ordem m x n, onde m é o número de linhas e n o número de colunas.



Matriz M de ordem 3

Por exemplo, uma matriz de ordem 2x 3 é aquela onde seus elementos estão dispostos em duas linhas e três colunas.

Uma matriz de ordem 3 é considerada como uma matriz quadrada, pois possui o mesmo número n x m. Por exemplo: Dimensionarumamatriz Para dimensionar uma matriz devemos utilizar a seguinte sintaxe: ***Tipo Nome_da_matriz[intervalo de linhas] [intervalo de colunas]***

Uma matriz poderá ser do tipo inteiro, real ou string e seu nome também obedece as mesmas regras definidas para uma variável.

Por exemplo:

Inteiro notas [10] [10]
Real salários [3] [2]

Em uma matriz de ordem 10 (10 linhas por 10 colunas), será gerado na memória principal do computador um espaço com 100 elementos (10x 10).

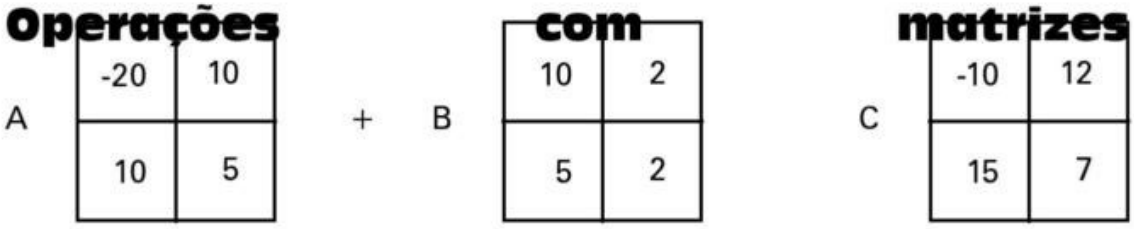
Vejamos um exemplo de criação de uma matriz com 100 elementos (10x 10): Início

```
1. Declarar variáveis:
     Inteiro Centena[10][10
     Inteiro i,j
     <i indica o índice de linha e j o índice de coluna>

2. Iniciar estrutura de repetição para criar e armazenar
valores em centena:
     PARA (início:i  fim i<10 alteraçãoi+1)    <alteraçãoi+1
= passo>

          PARA (início:j  fim j<10 alteraçãoj+1)
```

```
        Exiba "Digite um valor: "
        Ler centena [i][j]

          FIM PARA

            FIM PARA
```

```
3. Exibir valores digitados:
PARA (início:i   fim i<10 alteraçãoi+1)      <alteraçãoi+1 =
passo>

          PARA (início:j  fim j<10 alteraçãoj+1)

        Exiba "Os valores digitados foram: "
        Exiba centena [i][j]

          FIM PARA

      FIM PARA
```

Fim



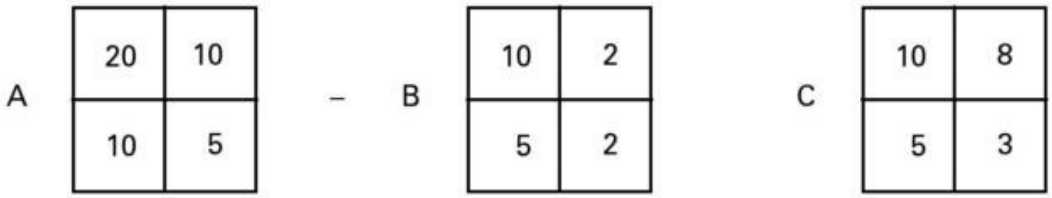
1. Soma entre matrizes Ao somar duas matrizes da mesma ordem (M x N) iremos gerar uma terceira matriz. Por exemplo:

Portanto, podemos dizer que a„ + b11 = c1,, a21 + b21 = c21 e assim sucessivamente.

Em algoritmos teremos: Início

```
1. Declarar as três matrizes:
inteiro A [2][2]
inteiro B [2][2]
inteiro soma [2][2]

2. Iniciar estrutura de repetição:
para (início: i=0 fim i<2 alteraçãoi+1)
```

```
para (início: j=0 fim j<2 alteraçãoj+1)
soma[i][j]= A [i][j]+ B [i][j]
fim para
fim para
```



Fim

2. Subtração entre matrizes A subtração entre matrizes é similar à adição, alterando somente o sinal da expressão:

Em algoritmos:

```
Início

1. Declarar as três matrizes:
inteiro A [2][2]
inteiro B [2][2]
inteiro subtrai [2][2]

2. Iniciar estrutura de repetição:
para (início: i=0 fim i<2 alteraçãoi+1)
para (início: j=0 fim j<2 alteraçãoj+1)
soma[i][j]= A [i][j] – B [i][j]
fim para
fim para
```

Fim

3. Multiplicação de matriz por um escalar Ao multiplicar a matriz denominada A de ordem 2 pelo escalar 2 teremos uma segunda matriz (C):

Teremos 20*4, 10 *4*, *10*4e5*4

```
Início

1. Declarar as três variáveis:
inteiro A [2][2]
inteiro escalar
inteiro resultado[2][2]

2. Iniciar estrutura de repetição:
para (início: i=0 fim i<2 alteraçãoi+1)
para (início: j=0 fim j<2 alteraçãoj+1)
resultado[i][j]= escalar * A [i][j]
fim para
fim para
```

Em algoritmos: **Fim**

4. Multiplicação entre matrizes Nem é preciso dizer que para multiplicar duas matrizes, as mesmas devem possuir o mesmo número de elementos (número de linhas (m) x número de colunas (n)).

```
Início

1. Declarar as três matrizes:
inteiro A [2][2]
inteiro B [2][2]
inteiro resultado[2][2]

2. Iniciar estrutura de repetição:
para (início: i=0 fim i<2 alteraçãoi+1)
para (início: j=0 fim j<2 alteraçãoj+1)
      para (início: k=0 fim k<2 alteraçãoj+1)

resultado[i][j]= resultado[i][j] + A[i][k] * B[k][j]

    fim para
fim para
fim para
```

Capítulo 8

Estrutura de dados e sub rotinas

Estrutura de dados

A estrutura de dados nada mais é do que uma área onde ficam armazenadas as informações em seu computador facilitando o gerenciamento destas informações. Estas informações podem ser recuperadas, atualizadas e até mesmo excluídas de forma rápida e segura.

Até o momento, pudemos observar que os algoritmos manipulam dados ou informações. Sempre que tais dados estão dispostos de forma organizada, caracterizam-se como uma estrutura de dados.

Em linguagens de programação podemos encontrar os seguintes tipos de dados:

• Primitivos: Como visto anteriormente, são aqueles predefinidos pelas linguagens que armazenam informações do tipo numérica (inteiro), monetária (real), cadeia de caracteres (strings), lógico (booleano) e outras. Além de permitirem que novos tipos sejam criados.

• Estáticos ou Heterogêneo: São aqueles gerados a partir de dados preexistentes e não sofrem alterações em suas características ou escopo durante a execução do sistema. São mais conhecidos como Vetor, Matriz e Registro.

• Dinâmicos: São aqueles que sofrem alterações durante a execução do sistema, como por exemplo os ponteiros (pilha, fila, lista e árvore).

Campo

É um conjunto de caracteres ou um dado isolado como, por exemplo o campo Nome completo. Simplesmente possui características (propriedades predefinidas) para armazenar as informações com o nome completo do funcionário.

As características de um campo englobam o tipo de dados que o mesmo deverá armazenar, tamanho, regras e outras informações que são peculiares a cada gerenciador de banco de dados ou linguagem de programação.

Registro

É um tipo de dado estático. Na verdade um registro é um conjunto de campos com informações sobre uma determinada pessoa, local, receita e outros. Por exemplo, um registro de um funcionário, nada mais é do que todas as informações referentes a um determinado empregado, como nome completo, endereço, data de nascimento e admissão, salário, dependentes e outros.

Podemos dizer que o registro é o único tipo de dado que pode agregar em sua estrutura diferentes tipos de dados. Ou seja, um registro com nome completo, endereço, idade e salário, pode conter em cada um dos campos caracteres, números, unidade monetária e outros.

Arquivo

Nada mais é do que um conjunto de registros (linhas) que identificam a informação através de uma chave ou índice agilizando a manipulação da informação.

Banco de dados É um conjunto de arquivos ou tabelas que podem ser compartilhadas entre vários usuários e com informações relacionais.

Figura 8.1.: Exemplos de bancos de dados.

Até aqui vimos somente conceitos simples. Vejamos cada uma dessas estruturas sob formas mais complexas e inseridas em algoritmos.

Manipulando Registros Para declarar um registro em um algoritmo, devemos fazer de forma similar à declaração de variáveis, ou seja, no início do algoritmo. Veja um exemplo:

```
   File: REGISTRO                    <início da estrutura com nome File>
      NOME_COMPLETO : String [55]
      IDADE: Inteiro
      ENDEREÇO: String[60]
      SEXO:Char     <caractere>
      SALÁRIO: Real
   FIM_REGISTRO                      <fim da estrutura>
```

Neste exemplo, será criada a estrutura que receberá o nome File e contém os campos Nome_Completo, que receberá uma cadeia de

caracteres, Idade que receberá um número inteiro, Endereço que receberá uma cadeia de caracteres, Sexo que receberá somente um caractere e Salário que receberá valores no formato monetário.

Como criar e ler a estrutura de dados:

```
Início

1. Declarar as variáveis para criação do registro Cadastro:
     Cadastro = registro
         NomeAluno  : string [30]
         Nota1 : real
         Nota2 : real
         Nota3 : real
         Nota4 : real
     Fim _ registro

2. Criar uma variável de nome Aluno que terá o formato de Cadastro (registro):

     Aluno :  Cadastro

3. Solicitar a digitação do nome e notas e armazenar nas variáveis:

     Ler Aluno.NomeAluno
     Ler Aluno.Nota1
     Ler Aluno.Nota2
     Ler Aluno.Nota3
     Ler Aluno.Nota4
     Escrever Aluno.Nota1

Fim
```

Se preferir, você poderá armazenar as notas em um vetor (conjunto de registros), utilizando a seguinte estrutura de repetição:

```
Início
Semestre : vetor[1..4] tipo Real   <uma linha e quatro co-
lunas/elementos>

Cadastro = registro
     Nome : string[40]
     Nota : vetor [1..4] tipo Real
Fim _ registro

Aluno:Cadastro
i:inteiro

Exiba "Digite o nome do aluno: "
ler aluno.Nome
Exiba "Digite as notas dos alunos: "

PARA i de 1 até 4 FAÇA
     Ler aluno.Nota[i]
FIM PARA
Fim
```

Agora, se o intuito é armazenar informações de vários alunos, a rotina deverá ser incrementada:

```
Início
     Inteiro: i, j, n
     Dados = vetor [1..6] real

     Cadastro = registro
       Nome : string [40]
       Idade : inteiro
       Turma : string[5]
```

```
      Notas : dados
    Fim _ registro

    Aluno: alunos
    Exiba "Digite a quantidade de alunos:"
    Leia n

    PARA i de 1 até n FAÇA
      Exiba "Digite o nome:"
      Leia alunos[i].Nome
      Exiba "Digite a idade:"
      Leia alunos[i].idade
      Exiba "Digite a turma:"
      Leia alunos[i].turma
Exiba "Digite as notas do aluno:"

      PARA j de 1 até 6 FAÇA
          Leia alunos[i].dados[j]
      FIM PARA

    FIM PARA

Fim
```

Manipulando Arquivos No exemplo anterior, o registro (conjunto de campos ou informações) foi armazenado na memória do computador enquanto o algoritmo se encontrava em execução. Uma boa opção para armazenar tais informações é guardar fisicamente em outro dispositivo, tais como CDs, DVDs ou até no próprio computador, mas em forma de arquivo.

Um arquivo nada mais é do que uma estrutura de dados que possui um ou mais registros de forma física e não virtual.

As informações existentes em um arquivo podem estar organizadas de forma sequencial ou direta.

• Forma sequencial: Onde todos os registros são armazenados (inseridos) na ordem digitada pelo usuário, ou seja, em sequência.

• Forma direta: Neste caso o registro é acessado (manipulado) através de um identificador.

Ao armazenar os dados do registro em um arquivo, teremos a facilidade de efetuar várias operações, como edição (alteração dos dados), inclusão e exclusão de informações.

Antes de especificar os dados do registro, é necessário declarar a existência do arquivo (assim como em variáveis) e abri-lo, ao final do algoritmo, é importante que o arquivo seja fechado.

Para declarar um arquivo, devemos utilizar a seguinte sintaxe: Arquivo <tipo de organização> de nome do registro: nome do arquivo

Por exemplo: Arquivo sequencial de Aluno: Cadastro Para abrir o arquivo, use a sintaxe: Abra nome do arquivo <tipo de utilização> Exemplo de utilização do arquivo: Abra CADASTRO leitura Abra CADASTRO escrita Abra CADASTRO

Para fechar o arquivo: Feche <nome do arquivo> Por exemplo: Feche CADASTRO

É preciso ressaltar que o formato de leitura e escrita do arquivo depende exclusivamente do tipo de organização (sequencial ou direta).

Quando organizado de forma sequencial, os registros são armazenados de forma contínua, um após o outro. Desta forma, a leitura de um registro somente será feita após a leitura de todos os registros anteriores. Com relação à escrita, esta operação somente será realizada após ler o último registro existente no arquivo.

Veja a sintaxe para leitura e escrita de um arquivo do tipo sequencial:

Leia NOME DO ARQUIVO. NOME DO REGISTRO

Escreva NOME DO ARQUIVO. NOME DO REGISTRO

Leia CADASTRO.NomeAluno Escreva CADASTRO.NomeAluno Sendo necessário incluir um registro em próxima posição do arquivo, é preciso acrescentar a instrução próximo ao comando de escrita:

Escreva próximo CADASTRO.NomeAluno Para indicar se determinado arquivo atingiu o seu final, todo banco de dados possui uma variável lógica

chamada Fim de Arquivo (FDA) ou End of File (EOF). Para indicar o início do arquivo, será encontrada a variável Início De Arquivo (IDA) ou Begin of File (BOF).

Na organização direta será necessário percorrer todos os registros anteriores para que se tenha acesso à informação. Isso é possível pois no momento de criação do arquivo é definida uma chave que determinada sua exclusividade. Por exemplo, devemos localizar um aluno, cuja matrícula é 999. Foi determinado um campo com o nome matrícula que armazena o número da matrícula de cada aluno, assim como em casos de RGs, CPFs e outros dados que não podem pertencer a dois indivíduos diferentes.

Imagine a seguinte estrutura:

MATRÍCULA	NOME	IDADE
111	Guilherme	13
222	Alexandre	10
333	Rafael	4
444	Fernanda	2

Tabela 8.1.

Vejamos um exemplo de algoritmo que deverá localizar o aluno 333 e exibir os seus dados:

```
Início

Cadastro = registro
     String Matrícula
     String Nome
     Inteiro Idade
Fim registro

Cadastro:Dados
Arquivo sequencial de Dados:Cadastro
Abra Cadastro leitura

FAÇA
     Leia Cadastro.Dados

     SE Dados.Matrícula = "333" ENTÃO
          Exiba Dados.Nome
          Exiba Dados.Idade
          Abandone           <sair da estrutura de loop>
     FIM SE

ENQUANTO Cadastro.FDA        <final de arquivo localizado>
Feche Cadastro
Fim
```

Neste caso, para ter acesso ao registro foi necessário efetuar a leitura do arquivo utilizando a chave de procura (campo matrícula). Caso a chave não seja localizada, será encontrada uma variável INV (condição de chave inválida).

Para a leitura e escrita da chave utilize as seguintes sintaxes: Leia item [NOME CHAVE] NOME ARQUIVO.NOME REGISTRO

Leia item MATRÍCULA Cadastro.NomeA/uno Escreva item [NOME CHAVE] NOME ARQUIVO. NOME REGISTRO

Leia item MATRÍCULA Cadastro.NomeAluno

Caso a chave não exista, você poderá definir uma mensagem informando o usuário. Para isso, entre com as instruções:

SE Cadastro.INV ENTÃO <chave não encontrada>
Exiba "Aluno inexistente"
SENÃO
Exiba Dados.NomeAluno
FIM SE

Listas Lineares Uma estrutura de dados dinâmicos (lista linear) representa um conjunto de informações que preserva uma relação, como uma sequência ordenada de elementos. Por exemplo, uma fila de clientes em uma agência bancária possui o primeiro elemento (primeiro cliente na fila), o último elemento (último cliente na fila) e uma ordem de atendimento para os mesmos (sequencial): $E_1, E_2, E_3 \ldots E_n$

E, é o primeiro elemento da lista En é o último elemento da lista Utilizando uma lista linear será possível: • Criar: Será gerada uma estrutura dinâmica durante a execução do sistema.

• Destruir: Após a utilização da lista linear, a estrutura será destruída.

• Percorrer / Atravessamento: Para a manipulação dos elementos da lista, todos os seus elementos devem ser percorridos. Ou seja, desde o início da lista até encontrar o seu final.

• Buscar: É possível buscar um elemento da lista através de sua posição ou conteúdo.

• Inserir: Novo elemento é incluído à lista incrementado n em uma unidade.

• Remover: Um elemento é retirado da lista ocorrendo um decréscimo em uma unidade de n.

Vejamos como realizar a operação de atravessamento (percorrer todos os elementos) de uma lista linear até encontrar o último registro (n):

```
Início
1. Declarar a existência da lista:
   LISTA: MATRIZ[N] DE CHAR ;

2. Declarar uma variável que armazena o número de elementos
da lista:
   N:INTEIRO

3. Declarar uma variável que servirá de contador:
Cont:Inteiro

4. Criar a rotina de atravessamento:
PROCEDA ATRAVESSAMENTO _ CONTI
SE  N < 1 ENTÃO
                Exiba "Lista e branco"
SENÃO
    Cont : = 0

    FAÇA ENQUANTO cont < n
       Cont: = Cont + 1
       Exiba Lista[cont]

  FIM ENQUANTO
        FIM _ SE
FIM
```

Inclusão de elementos em uma lista Para que novo dado seja criado em uma lista, primeiramente devemos criar um espaço em branco (como novo registro), armazenar o elemento no espaço gerado e verificar a existência do novo elemento em toda a lista. Desta forma, se o dado não existir, para incluí-lo, devemos remanejar todos os demais elementos abaixo da posição atual do cursor.

Vejamos um algoritmo que descreve cada uma das ações a serem executadas:

```
Início
1. Declarar a lista que receberá novos dados:
LISTA:MATRIZ [N+1] DE CHAR

2. Declarar uma variável que armazena o número de elementos
na lista:
N:INTEIRO

3. Declarar uma variável que indica a posição de inclusão
do novo elemento:
POS:INTEIRO

4. Iniciar a inclusão do novo elemento:
NOVA _ LETRA

5. Declarar uma variável que controla o loop:
CONT :INTEIRO

6. Iniciar a rotina de inclusão:
PROCEDA INCLUSAO _ CONTI
Exiba "Informe a letra para ser incluída na lista"
Leia NOVA _ LETRA
Pos: = 1
ENQUANTO POS < = N E LISTA[POS]< NOVA _ LETRA
POS: = POS+1
FIM _ ENQUANTO

SE LISTA[POS] = NOVA _ LETRA ENTÃO
EXIBA "Infelizmente esta letra já existe"
SENÃO
CONT: = N
ENQUANTO CONT > = POS
LISTA [ CONT+1 ] : = LISTA [ CONT ]
CONT : = CONT - 1
FIM _ ENQUANTO

LISTA [ POS ] : = NOVA _ LETRA
N : = N + 1
FIMSE
Fim
```

Alterar um elemento da lista Para alterar um elemento dentro de uma lista, devemos ler o dado informado, procurar a sua posição dentro da lista (verificar se o mesmo existe ou não) e também verificar se a alteração que

está sendo efetuada não irá reordenar os demais elementos existentes, para depois, efetuar a alteração desejada.

```
Início

1. Declarar a lista
LISTA : MATRIZ [ N ] DE CHAR

2. Declarar uma variável com o número de elementos da lis-
ta:
N:INTEIRO

3. Declarar as variáveis com a posição (Pos), elemento a ser
alterado (Elemento), novo elemento que será acessado (novo)
e variável lógica que armazena alterações (altera):

Inteiro POS
Caractere ELEMENTO
Caractere NOVO
Logico ALTERA
4. Iniciar rotina de alteração:

PROCEDA ALTERACAO _ CONTI
Exiba "Digite a letra que deseja alterar: "
Leia ELEMENTO
Exiba "Digite a nova letra: "
Leia NOVO
POS: = 1
```

```
ENQUANTO POS < = N E LISTA [POS]<> ELEMENTO FAA
    POS: = POS+1
FIM _ ENQUANTO

SE POS > N ENTÃO
    Exiba "Alteração inválida"

SENÃO
    ALTERA: = FALSO

   CASO
N = 1                                  <primeira situação>
ALTERA: = VERDADEIRO

POS = 1                              <segunda situação>
SE NOVO < LISTA[POS + 1] ENTÃO
ALTERA: = VERDADEIRO
FIM SE
POS = N                              <terceira situação>
SE NOVO > LISTA[POS - 1] ENTÃO
     ALTERA: = VERDADEIRO
FIM _ SE

    SENÃO      <quarta situação>
SE NOVO > LISTA[POS -1] E NOVO< LISTA[POS + 1] ENTÃO
    ALTERA:=VERDADEIRO
FIM _ SE

    FIM CASO

    SE ALTERA = VERDADEIRO ENTÃO
LISTA  [POS] : = NOVO
    SENÃO
Exiba "A alteração não foi efetuada"
   FIM _ SE
  FIM _ SE

Fim
```

Na primeira situação (N = 1), a quantidade de elementos da lista é igual a 1, portanto qualquer valor será permitido para a alteração. Quando POS = 1 (segunda situação) considera-se que o elemento a alterar é o primeiro da lista, portanto, o novo elemento (novo) deverá ser menor que o da próxima

posição para que a alteração seja concluída com êxito.

Caso o elemento a alterar seja o último da lista (POS = N), o novo elemento deverá ser maior que o elemento da posição anterior. Caso nenhuma dessas situações seja verdadeira, a lista possui mais do que dois elementos, portanto, o novo deverá ser maior que o da posição anterior e menor que da posterior.

Exclusão de um elemento na lista Para a exclusão de um elemento, é preciso efetuar a entrada do mesmo, pesquisar a posição na lista e remanejar os demais elementos existentes.

```
Início
1. Declarar a matriz e número de elementos
LISTA : MATRIZ [ N ] DE CHAR
 N:INTEIRO
2. Declarar variáveis com posição, controle loop e elemento
a excluir
Inteiro POS
Inteiro CONT
Caractere ELEMENTO

3. Iniciar  a rotina de exclusão
    PROCEDA EXCLUSÃO _ CONTI
         Exiba " Qual a letra a ser escluída?"
         Leia ELEMENTO
         POS: = 1
         ENQUANTO POS < = N E LISTA [POS]<> ELEMENTO FAÇA
POS: = POS+1
         FIM _ ENQUANTO
```

```
          SE POS > N ENTÃO
Exiba "A exclusão é inválida"
         SENÃO
CONT: = POS
ENQUANTO CONT <  N  FAÇA
LISTA [CONT] : = LISTA [CONT+1]
CONT : = CONT + 1
FIM _ ENQUANTO

LISTA [N] : = NULO
N: = N - 1

         FIM SE
Fim
```

Subrotinas

À medida que o algoritmo aumenta e torna-se mais complexo podemos encontrar a necessidade de dividi-lo em pequenas porções para facilitar a sua compreensão. Desta forma, módulos menores, conhecidos como subrotinas, podem ser aproveitados em outros programas, facilitando a correção de todo o sistema.

0 algoritmo será constituído de um programa principal que controla todas as tarefas existentes nas subrotinas e teremos variáveis que podem ou não ser reconhecidas pelo programa principal ou somente para algumas subrotinas.

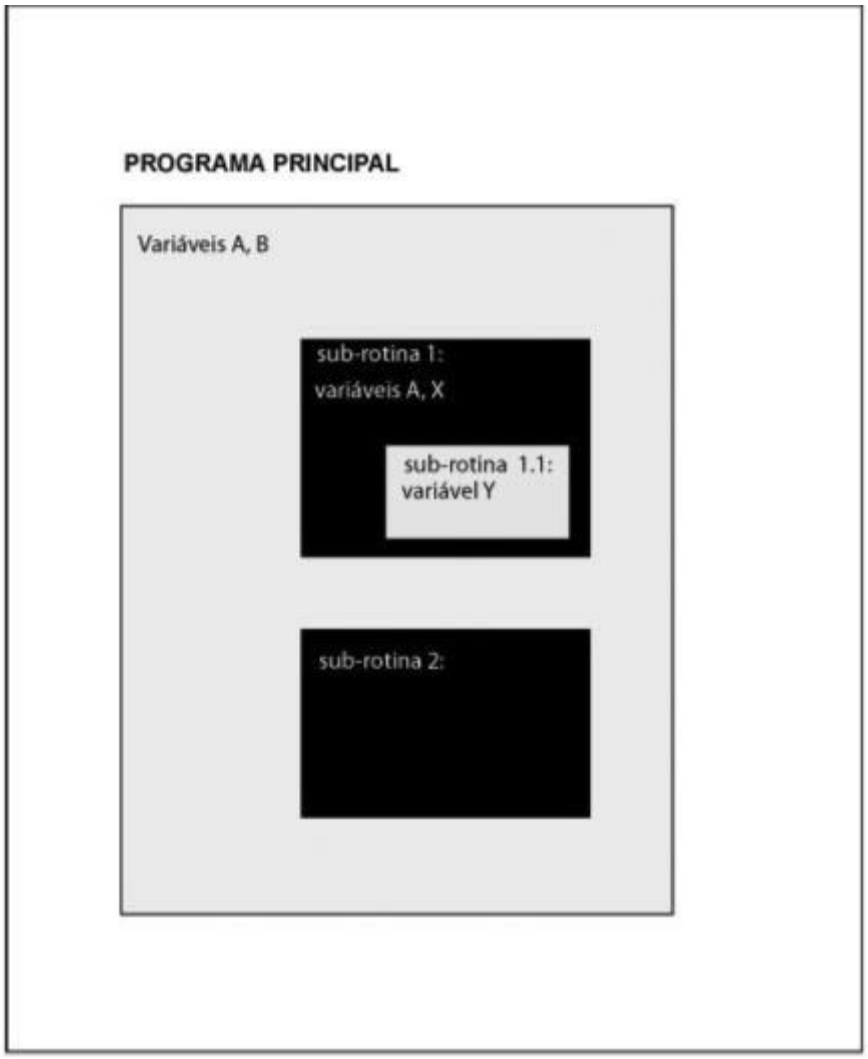


Figura 8.2.

Ao definir subrotinas, devemos ter um cuidado especial com a declaração das variáveis pois, no exemplo anterior, todas as subrotinas enxergam as variáveis A e B, pois foram declaradas no início do programa principal.

Caso a variável seja definida dentro de uma subrotina, somente será visualizada pela subrotina que a declarou, não sendo visualizada nem pelo programa principal.

Criação da subrotina Para criar uma subrotina, utilize a seguinte sintaxe:

Subrotina NOME (parâmetros)
Declaração das variáveis locais
Instruções da sub-rotina
Fim subrotina

Para chamar uma subrotina dentro de um programa principal, use: Nom e_subrotina (parâmetros) Uma subrotina e funções são exemplos de rotinas secundárias que obedecem hierarquicamente a um programa principal, além de poderem conter subrotinas e funções dentro delas.

A diferença entre uma subrotina e uma função é que a função sempre retorna um valor ao algoritmo que a chamou. Para criar uma função, use:

Função tipo NOME (parâmetros)
Declaração das variáveis locais
Instruções
Fim função

Para chamar uma função: Nome da função (parâmetros) Um exemplo de subrotina:

1. Declarar variáveis globais – todas as sub-rotinas a enxergam

Inteiro A.B

2. Criação da sub-rotina X:

Sub-rotina y(inteiro c,d)
<somente a sub-rotina reconhece os parâmetros c e d.>
Instruções da sub-rotina
...
fim_sub-rotina

Sub-rotina z(inteiro e)
Inteiro f
comandos de
fim_sub-rotina

y(1,2)
z(10)

Exemplo de uma rotina que deverá ler três valores e ordená-los:

```
Início

Inteiro A, B, C, contador, auxiliar
Cont = 0

FAÇA

     Leia A, B, C
     Escreva A, B, C

     SE C > A ENTÃO
          Auxiliar = C
          C = A
                    A = auxiliar
     FIM SE

SE C> B ENTÃO
          Auxiliar = C
          C = B
                    B = auxiliar
     FIM SE

SE B > A ENTÃO
          Auxiliar = C
          C = A
                    A = auxiliar
     FIM SE

     Escreva A, B, C
     Contador = contador + 1

Enquanto CONTADOR = 4

Fim
```

Para utilizar subrotinas ou sistema de modularização:

```
Início

Inteiro A, B, C, D, contador

<rotina para comparação dos valores e troca de posições
– COMPARA>

PROCEDIMENTO COMPARA _ TROCAR(val1; val2)
    Inteiro val1, val2

    SE Val1 > Val2 ENTÃO
     Auxiliar = Val1
     Val1 = Val2
     Val2 = auxiliar
    FIM SE

FIM PROCEDIMENTO

<PROGRAMA PRINCIPAL:>
Contador = 0
FAÇA
     Leia A, B, C, D
      Escreva A, B, C, D
      COMPARA _ TROCAR(D,A)
      COMPARA _ TROCAR(D,B)
      COMPARA _ TROCAR(D,C)
      COMPARA _ TROCAR(C,A)
      COMPARA _ TROCAR(C,B)
      COMPARA _ TROCAR(A,B)
      Escreva a, B, C, D
      Contador = contador + 1

ENQUANTO contador = 5

Fim
```

Vejamos um exemplo de criação da função dobro, que recebe um número do usuário e calcula o dobro do mesmo:

```
Início
Inteiro num
Função dobro(inteiro n):inteiro
     Inteiro numero

     Numero = n*n
     Retorne numero
Fim Função

Escreva dobro(num)
Fim
```

Uma função poderá passar um valor à rotina principal utilizandose do valor ou da referência. Caso utilize um valor, a função recebe o parâmetro de entrada, realiza o processamento das instruções e alterações na variável declarada como parâmetro, mas não reflete este valor quando utilizada fora da subrotina ou da função, seria o mesmo que dizer que possui os valores locais (somente funcionam dentro da instrução ou procedimento onde foi declarada).

Já na passagem de parâmetros por referência todas as alterações realizadas nos parâmetros (variáveis) refletem globalmente, ou seja, são enxergados por todos os procedimentos da rotina principal (demais rotinas e funções).

A maior parte das linguagens de programação considera que as funções utilizam passagens por valor, ao passo que as subrotinas usam referências.

Capítulo 9

Validação de dados e relatórios

Imagine que, após a verificação de todo o sistema, nem tudo está perfeito. E agora? Por melhor que seja, todo sistema é passível de falhas, mas isso pode colocar tudo a perder. Por isso, o ideal é checar as informações (dados) antes de começar tudo novamente.

Essa verificação de dados é conhecida como depuração, e consiste de testes que verificam a validade das informações; desta forma, você irá garantir a integridade dos dados e de todo o sistema.

Tais testes devem ser previstos durante a criação do módulo e deverão anteceder a integração do procedimento com outros módulos e subrotinas existentes.

Alguns testes durante o período de implantação do sistema podem apontar falhas relacionadas a todos os aspectos existentes no sistema, até mesmo aquelas em equipamentos ou mesmo na comunicação.

Você pode não ter percebido, mas criou algumas rotinas (algoritmos) de verificação a consistência de dados ao utilizar as estruturas de repetição. Veja alguns exemplos:

```
SE CÓDIGO <> “ ” ENTÃO

     instruções a realizar

FIM SE


CASO CÓDIGO

     CÓDIGO > 1
         instruções

     CÓDIGO E>10 E <=20
         instruções

FIM CASO
```

Na realização de teste de verificação de dados, poderá ser útil a utilização de funções para manipular strings e conversão de tipos de dados. Por isso, foram selecionadas as funções que a maior parte dos algoritmos reconhece para a criação de rotinas mais complexas.

Funções de manipulações de strings • Função strtam utilizada para determinar o tamanho de uma cadeia de caracteres:

Exiba "digite seu nome:"

Leia nome

Exiba "Seu nome tem " e strtam(nome) e "caracteres"

Por exemplo, você digitou MIGUEL e a função strtam informou que a largura (tamanho) da cadeia de caracteres nome é 6.

• Função strconcat utilizada para juntar (concatenar) duas ou mais strings (cadeia de caracteres): Exiba "Digite a agência: "

Leia agencia

Exiba "Digite a conta corrente: Leia conta

Exiba "Faça um depósito na Agência e Conta: " e

strconcat(agencia,conta) • Função strcopia utilizada para copiar o conteúdo de uma string: Exiba strcopia("cadeia")

Resultado: copia a palavra cadeia Exiba "Digite o nome do arquivo: "

Leia arquivo Exiba "Você deverá abrir o arquivo " strcopia(arquivo) • Função strprim utilizada para retornar o primeiro elemento de uma cadeia de caracteres:

Exiba "Digite seu nome: "

Leia nome

Exiba "A primeira letra de seu nome é " e strprim(nome) • Função strresto utilizada para retornar todos os elementos de uma cadeia de caracteres, exceto o primeiro:

Exiba strresto("Maria") 0 resultado é arfa • Função strelem utilizada para retornar o enésimo elemento de uma string:

Exiba "Digite a agência no formato xxxxx-x"

Leia agencia Exiba "0 dígito de controle de sua agência é: " e strelem(agencia,7)

Funções de conversão de tipos Em algumas validações de dados, é preciso converter tipos de dados que foram armazenados de forma diferentes, para isso, veremos as funções que fazem a conversão de tipos de dados: • Função intreal utilizada para converter um número inteiro em real:

Exiba intreal(20) Resultado é 20.0

• Função realint utilizada para converter um número real em inteiro: Exiba realint(20.8) Resultado é 21.

Vejamos um exemplo mais concreto por meio do seguinte algoritmo: Muitas empresas adotam sistemas nos quais deverá ser feita a entrada de dados do cliente do tipo pessoa física ou jurídica. Normalmente é necessária a validação de dados do CNPJ e CPF.

No algoritmo seguinte, faremos a validação de um número de CNPJ digitado pelo usuário. Mas, primeiramente, é importante saber como as informações devem ser processadas: 1. 0 CNPJ é composto por três segmentos de algarismos, sendo o primeiro o número da inscrição, logo após a barra o número de filiais que a empresa possui e os dois últimos dígitos as considerados como dígitos verificadores:

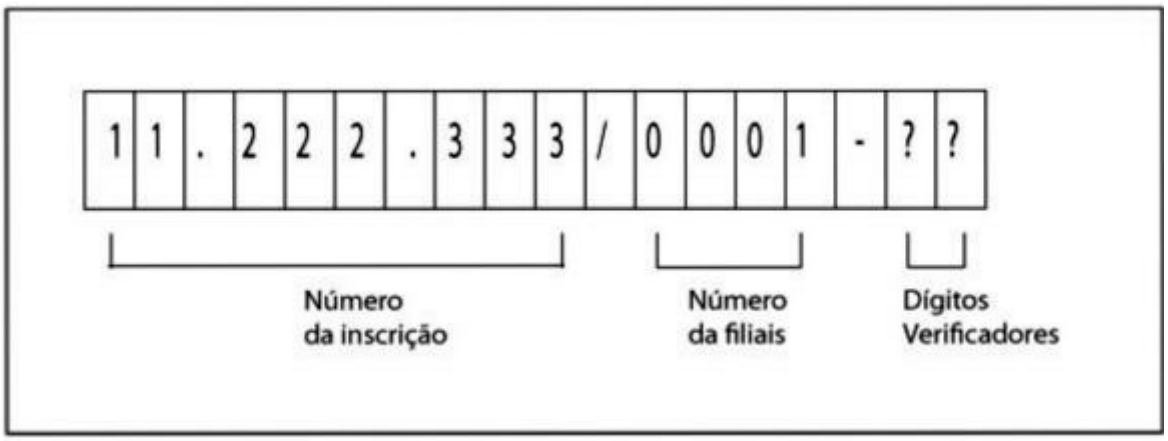


Figura 9.1.: Composição do CNPJ.

2. Devemos alinhar os número do CNPJ com os algarismos 5,4,3,2,9,8,7,6,5,4,3 e 2:

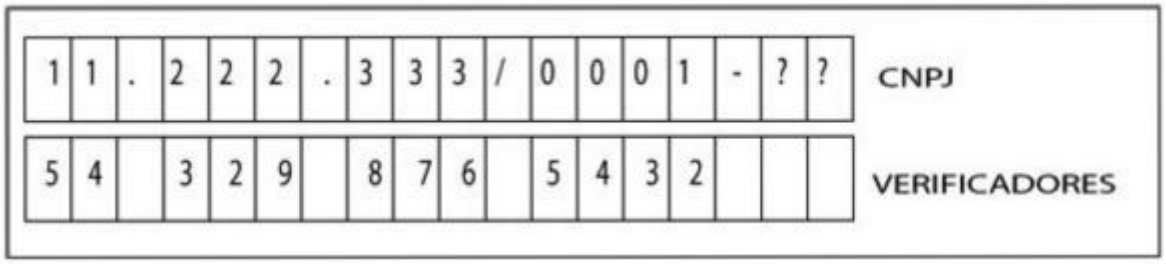


Figura 9.2.: Alinhamento do CNPJ com dígitos verificadores.

3. Agora cada valor do CNPJ deverá ser multiplicado pelos dígitos verificadores:

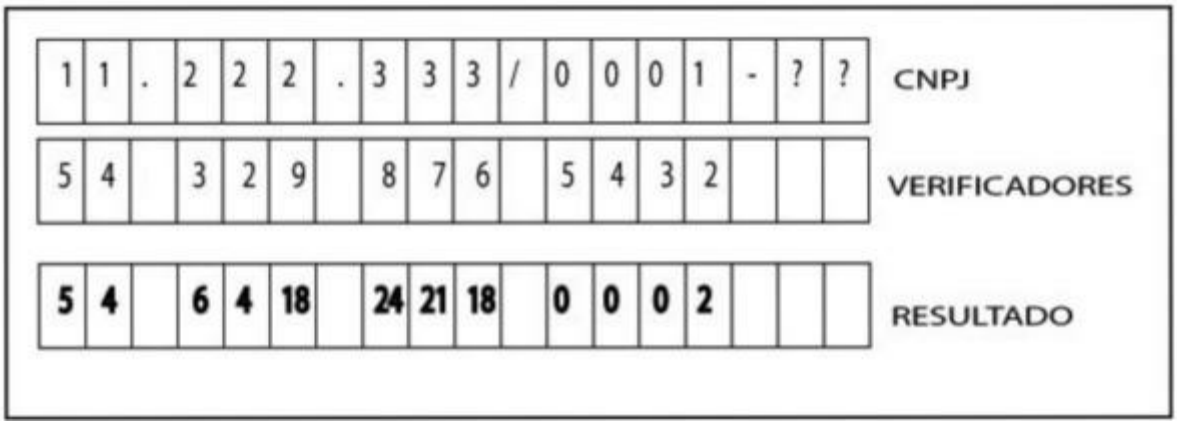


Figura 9.3.: Resultado do alinhamento.

4. Agora devemos somar todos os valores obtidos no resultado:

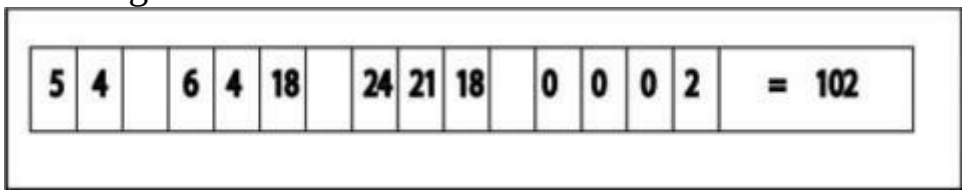


Figura 9.4.: Soma dos valores obtidos.

5. Devemos verificar o resultado da soma (102) e dividir este valor por 11, e teremos:

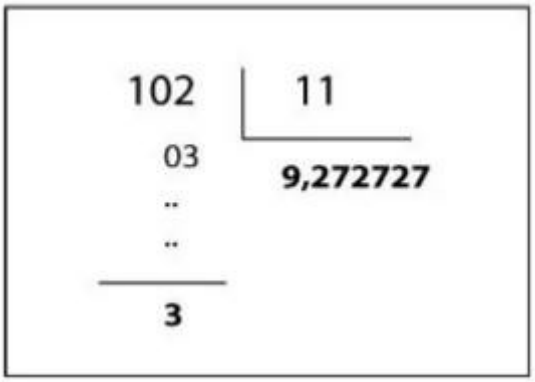


Figura 9.5.: Divisão do resultado da soma por 11.

6. 0 importante é considerar somente o valor inteiro como quociente (9). Agora devemos verificar o resto da divisão (3) que será responsável pelo cálculo do primeiro dígito verificador.

7. Caso o resto da divisão seja menor que 2 (dois), o valor do dígito verificador é 0 (zero). Caso contrário, devemos subtrair este valor de 11,

obtendo assim, o primeiro dígito verificador. No exemplo anterior, teremos 11 - 3 = 8. Portanto 8 deverá ser o primeiro dígito verificador:

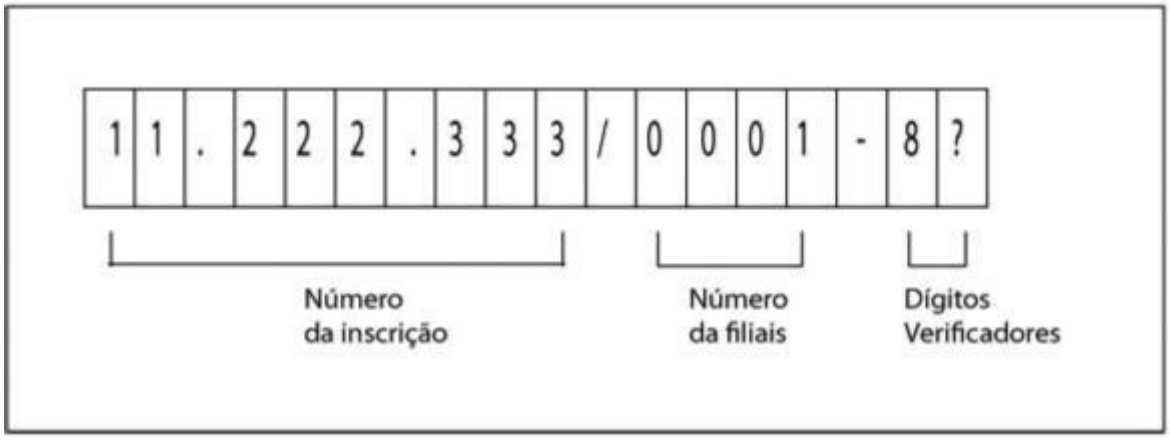


Figura 9.6.: Definição do primeiro dígito verificador.

8. Para fazer o cálculo do segundo dígito verificador novamente devemos alinhar os valores do CNPF aos valores 6,5,4,3,2,9,8,7,6,5,4,3 e 2:

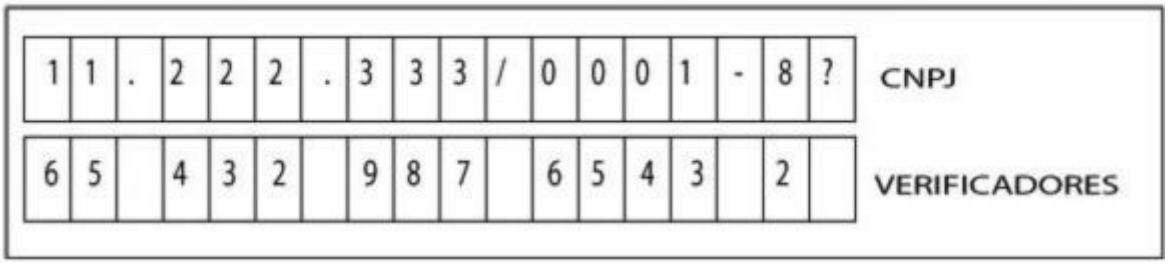


Figura 9.7.: Alinhamento para o cálculo do segundo dígito verificador.

9. Novamente multiplicar os valores do CNPJ pelos valores alinhados e realizar a soma dos mesmos. Cujo resultado será 120.

10. Com o resultado obtido (120), dividir por 11.

11. Assim como no cálculo anterior, o cálculo do último dígito verificador será feito com o resto da divisão (10). Subtraia 11 a este valor e teremos como resultado 1 (um).

12. A mesma regra deverá ser obedecida, caso o resultado da subtração for menor que 2 (dois), o valor do dígito verificador será 0 (zero), caso contrário este será o dígito verificador. Portanto, o dígito verificador será 1 e obteremos o seguinte número (fictício) do CNPJ:

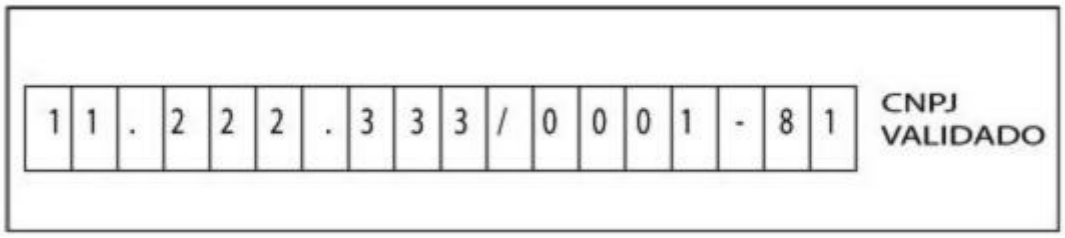


Figura 9.8.: Definição do segundo dígito verificador.

Com você pode observar, não foi nada simples, vários cálculos foram realizados; agora imagine como colocar isso em um sistema que responderá em fração de segundos se o CNPJ é válido ou não?

Para facilitar, acompanhe o algoritmo a seguir, que deverá ser adaptado à linguagem de programação utilizada e deverá retornar Verdadeiro (True) ou Falso (False) assim que entrar com o CNPJ. Portanto, iremos criar uma função, pois sabemos que ela é responsável pelo retorno de um valor ao sistema.

Início

```
1. Inicialização da função:
     Função VerificaCNPJ(sCNPJ As String) As String

2. Declarar variáveis que recebem os valores do CNPJ:
     Inteiro d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12,
d13, d14

3. Declarar variáveis que recebem dígitos verificadores e
último dígito:
     Inteiro DV1, DV2, Ultimo
4. Verificar se a largura do CNPJ é menor que 14 posições, se
sim, irá preencher com zero à esquerda

     SE strtam(sCNPJ) < 14 ENTÃO
         sCNPJ = String(14 - strtam(sCNPJ), "0") & sCNPJ
     FIM SE

5. Verificar a posição do último dígito para alinhar os va-
lores:
     Ultimo = strtam(sCNPJ)
```

6. Se o usuário deixar o CNPJ em branco, deverá sair da função:

```
     SE sCNPJ = "00000000000000" ENTÃO
    VerificaCNPJ = ""
    Abandone
     FIM SE
```

7. Armazenar cada dígito do CNPJ atual em uma variável:

```
d1 = strelem(sCNPJ,14)
d2 = strelem(sCNPJ,13)
d3 = strelem(sCNPJ,12)
d4 = strelem(sCNPJ,11)
d5 = strelem(sCNPJ,10)
d6 = strelem(sCNPJ,9)
d7 = strelem(sCNPJ,8)
d8 = strelem(sCNPJ,7)
d9 = strelem(sCNPJ,6)
d10 = strelem(sCNPJ,5)
d11 = strelem(sCNPJ,4)
d12 = strelem(sCNPJ,3)
d13 = strelem(sCNPJ,2)
d14 = strelem(sCNPJ,1)
```

8. Calcular os dígitos verificadores corretos:

```
DV1 = (d1 * 6) + (d2 * 7) + (d3 * 8) + (d4 * 9) + (d5 * 2) +
(d6 * 3) + (d7 * 4) + (d8 * 5) + (d9 * 6) + (d10 * 7) + (d11
* 8) + (d12 * 9)
```

9. Calcular o resto da divisão entre a soma dos valores por 11:

```
     DV1 = DV1 Mod 11
```

10. Verificar se resultado é menor que 2:

```
SE DV1 > 2 ENTÃO
    DV1 = 0
SENÃO
    DV1 = 11 - DV1
FIM SE
```

```
11. Calcular o dígito verificador 2:
DV2 = (d1 * 5) + (d2 * 6) + (d3 * 7) + (d4 * 8) + (d5 * 9) +
(d6 * 2) + (d7 * 3) + (d8 * 4) + (d9 * 5) + (d10 * 6) + (d11
* 7) + (d12 * 8) + (DV1 * 9)

12. Calcular o resto da divisão entre a soma dos valores por
11:
DV2 = DV2 Mod 11

13. Verificar se resultado é menor que 2:
SE DV2 > 2 ENTÃO
    DV2 = 0
SENÃO
    DV2 = 11 - DV2
FIM SE
14. Comparar os dígitos existentes com os obtidos em DV1 e
DV2:
SE d13 = DV1 E d14 = DV2 ENTÃO
    VerificaCNPJ = "ESTE CNPJ É VÁLIDO !"
SENÃO
    VerificaCNPJ = "ESTE CNPJ É INVÁLIDO"
FIM SE

FIM FUNÇÃO
Fim
```

Relatórios

Normalmente, um relatório é um registro impresso de informações processadas pelo sistema. Para isso as informações foram armazenadas em um dispositivo de armazenamento, como um arquivo em CDs, no HD ou em disquetes, tais informações foram processadas e ao invés de exibi-Ias na tela, deverá enviar ao papel.

Ao trabalhar com relatórios é preciso verificar se os dados serão impressos em formulários predefinidos, daí a necessidade de controle de locais em que os mesmos devem aparecer. Outra preocupação é verificar o número de linhas de informações em uma página, além do salto das páginas; pois é fundamental que o relatório não envie informações para fora das margens do papel.

Muitos relatórios possuem cabeçalhos, rodapés e numeração de páginas que também deve ser controladas.

Figura 9.9.: Estrutura típica de um relatório.

A utilização de cabeçalhos auxilia na identificação do assunto, já o rodapé, poderá conter linhas totalizadoras, data, numeração de página e outras informações sobre a empresa ou departamento.

Para a impressão de relatórios é necessário: 1. Abrir o arquivo de dados e ler as suas informações.

2. Criar variável com contador de linhas.

3. Criar variável com totalizador, para no caso de inserção da informação ao final da página (rodapé).

4. Criar um cabeçalho com a identificação do relatório e numeração de página.

5. Controlar a quebra de páginas.

6. Criar layout apropriado para os detalhes do relatório, muitas vezes com definições para as colunas onde os mesmos devem ser impressos, por exemplo, imprimir o código na coluna 4, o nome na coluna 13, o valor na

coluna 40 e determinar em quais todos os dados devem aparecem em uma linha.

Acompanhe no fluxograma as etapas que o sistema deverá executar:

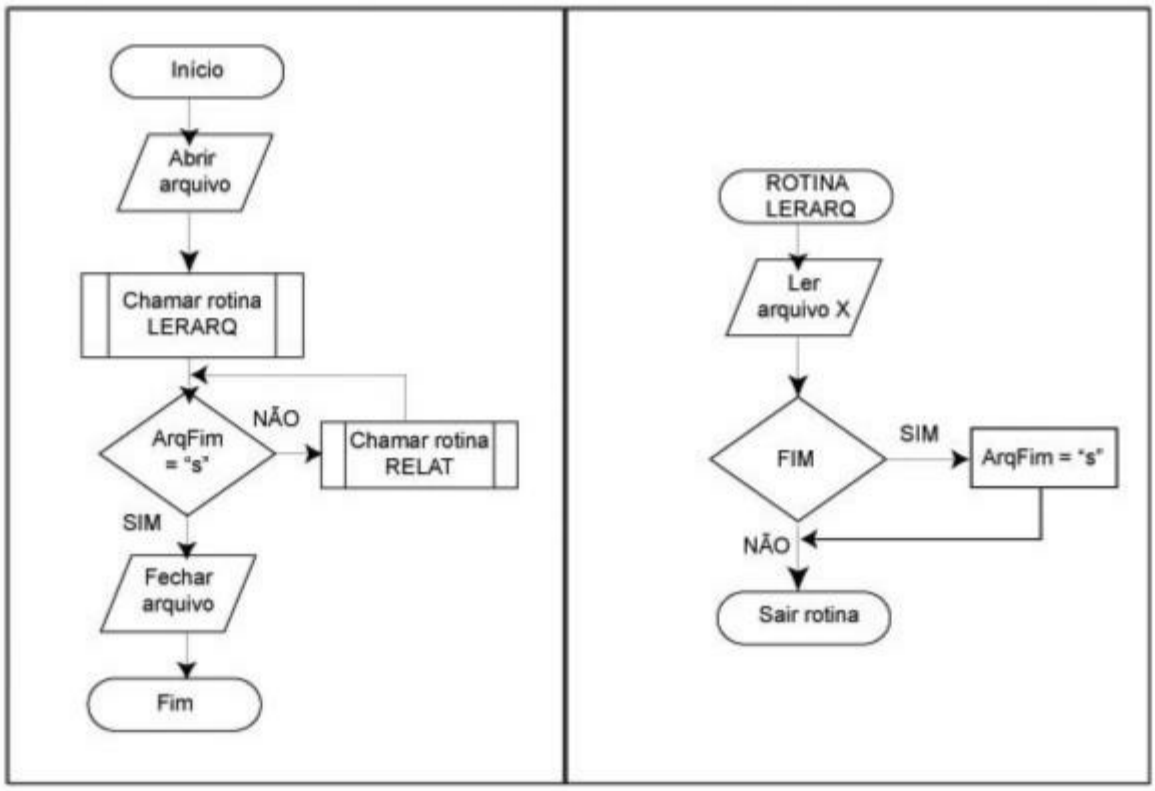


Figura 9.10.: Fluxograma de criação de relatório.

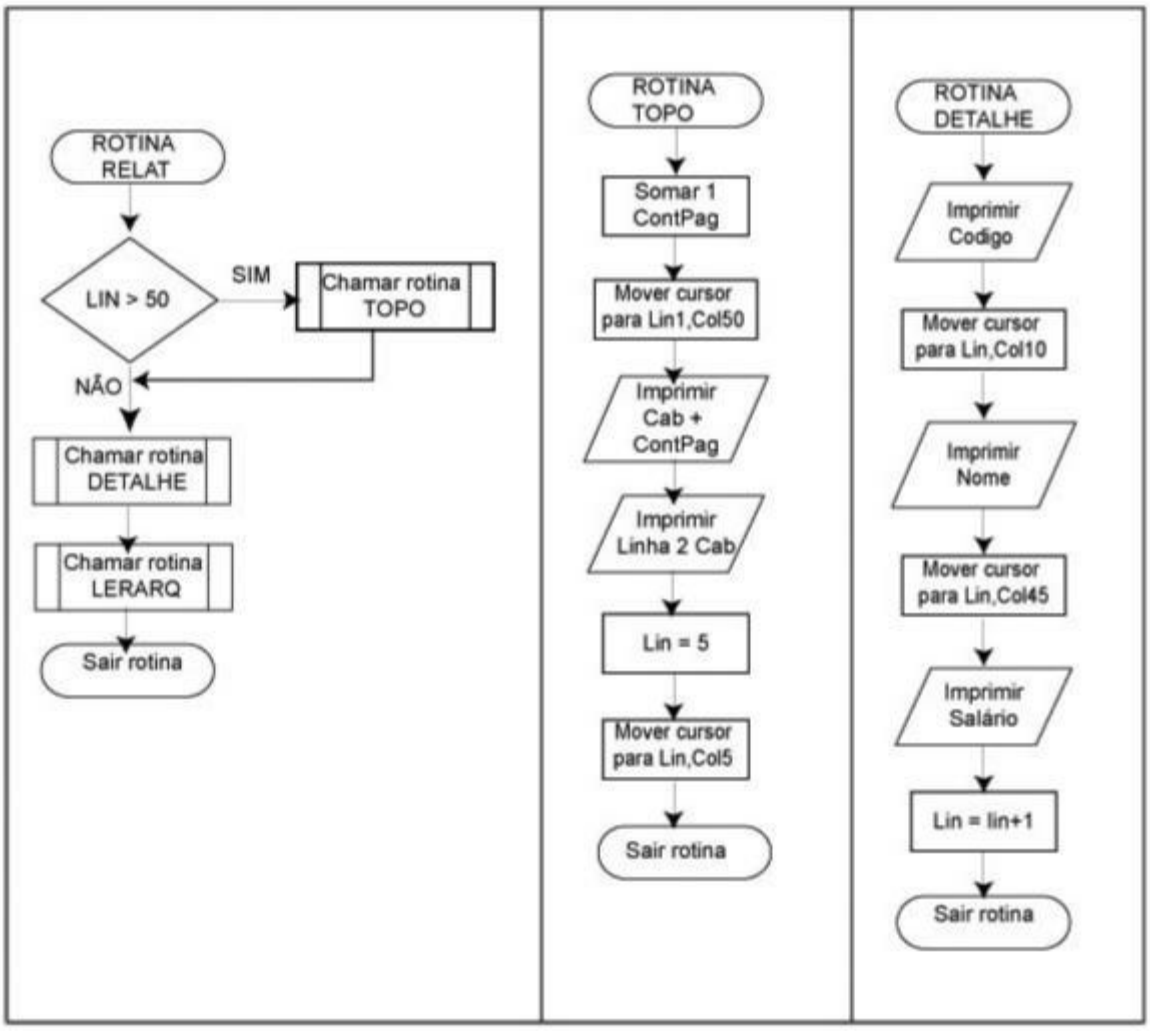


Figura 9.11.: Fluxograma de rotina de criação de relatórios.

Apêndice

Exercícios propostos e Solução dos algoritmos propostos

Exercícios Propostos

Capítulo4-OperadoresAritméticos,RelacionaiseLõgicos Algoritmo 04_01: Ler um número inteiro e exibir seu sucessor e antecessor.

Algoritmo 0402: Solicite a digitação de quatro números e exibir a média ponderada dos mesmos. Sendo que os respectivos pesos são 1, 2, 3 e 4.

Algoritmo 04_03: Solicite a digitação do saldo atual da caderneta de poupança, exiba o resultado do reajuste de 1% sobre o valor digitado e informe ao usuário.

Algoritmo 04_04: Miguel resolveu fazer uma aplicação de R$ 500,00 (quinhentos reais) por seis meses a uma taxa de juros de 1%

$$\text{valor acumulado} = P * \frac{(1+i)^n - 1}{i}$$

i = taxa
P = aplicação mensal
n = número de meses

ao mês. Qual será o valor acumulado da aplicação? Para isso, utilize a seguinte expressão para o cálculo do valor acumulado: Capítulo6-EstruturasdeDecisão Algoritmo 0601: Para crédito imobiliário, uma instituição bancária estipulou que a prestação não poderá ultrapassar 30% do salário bruto do funcionário. Crie um algoritmo que solicita o salário e indica se o crédito foi aprovado ou não.

Algoritmo 06_02: Um clube de uma cidade do interior resolveu promover eliminatórias para o mundial de natação. Para isso, é ne cessário que o nadador informe sua idade e o sistema informe a categoria a que ele pertence, de acordo com a seguinte tabela:

Infantil 1 - de 5 a 7 anos Infantil 2 - de 8 a 10 anos Juvenil - de 11 a 17 anos Adulto - maiores de 18 anos Algoritmo 06_03: Um restaurante possui 3 opções de pratos para a refeição principal: peixe com 230 calorias, frango, com 250 e carne com 350 calorias; além disso, tem as seguintes opções de sobremesa: abacaxi com 50 calorias, mousse com 350 calorias e sorvete com 170 calorias. Crie um algoritmo que solicita qual o prato principal e a sobremesa e informe ao cliente, quantas calorias serão ingeridas com a refeição.

Capitulo7-EstruturasdeRepetição(Loop) Algoritmo: 07 01: Entrevistar 20 pessoas e especificar seu estado civil, sendo C para casados, S para solteiros ou V para viúvos. Ao final do algoritmo calcular a distribuição entre eles.

Algoritmo: 07_02: Entrar com 100 números e informar ao final quantos são pares e quantos são ímpares.

Algoritmo: 07_03: Entrar com 20 números e exibir a soma dos positivos e dos negativos.

Capitulo8-VetoreseMatrizes Algoritmo: 08_01: Solicitar a digitação de nome e salário de 10 pessoas. Calcular um reajuste de 5% sobre esse salário e exibir os nomes e novos salários de cada um.

Solução dos algoritmos propostos Capitulo4-OperadoresAritméticos,RelacionaiseLõgicos Algoritmo 04_01:

```
var numero : inteiro
    sucessor : inteiro
    antecessor : inteiro

início

escreva ("Entre com um número ")
leia (numero)
antecessor:= numero -1
sucessor:= numero + 1

escreva ("O sucessor de ", numero, " é: ", sucessor)
escreva ("O antecessor de ", numero, " é: ", antecessor)

fim algoritmo
```

Algoritmo 04_02:

```
var valor1, valor2, valor3, valor4, media: real

início

escreva ("Entre com o primeiro número ")
leia (valor1)
escreva ("Entre com o segundo número ")
leia (valor2)
escreva ("Entre com o terceiro número ")
leia (valor3)
escreva ("Entre com o quarto número ")
leia (valor4)
media <- (valor1*1 + valor2*2 + valor3*3 + valor4*4)/10

escreva ("A média ponderada dos valores é: ", media)

fim algoritmo
```

Algoritmo 04_03

```
var poupaAtual, poupaAjustado: real

início

escreva ("Entre com o saldo atual de sua poupança ")
leia (poupaAtual)
poupaAjustado <- (poupaAtual * 1.01)

escreva ("O saldo atual de sua poupança com ajuste de 1% é:
", poupaAjustado)

fim algoritmo
```

Algoritmo 04_04

```
var aplicado, acumulado, taxa: real
    meses: inteiro

início

escreva ("Entre com o valor da aplicação ")
leia (aplicado)
escreva ("Entre com a taxa (de 0 a 1)")
leia (taxa)
escreva ("Qual o período da aplicação? ")
leia (meses)
acumulado <- aplicado * (((1 + taxa)** meses)-1) / taxa

escreva ("O valor acumulado é: ", acumulado)

fim algoritmo
```

Capitulo6-EstruturasdeDecisão Algoritmo 06_01

```
var
   salario, prestacao: real

início
   escreva ("Digite o salário ")
   leia (salario)

   escreva ("Digite o valor da prestação: ")
   leia (prestacao)
```

```
   se ( prestacao <= salario * 0.3) então
      escreva (" O financiamento pode ser realizado ")
   senão
     escreva ("Infelizmente com o salário atual, o financia-
mento foi negado")
   fim se

fim algoritmo
```

Algoritmo 06_02

```
var
  idade: inteiro

início
escreva ("Qual a sua idade ?")
leia (idade)

escolha idade
    caso <5
     escreva ("Somente serão aceitos nadadores a partir dos
5 anos")
   caso >=5 e <=7
     escreva ("A categoria é Infantil 1")
   caso >=8 e <=10
     escreva ("A categoria é Infantil 2")
   caso >=11 e <=17
     escreva "A categoria é Juvenil")
   outro caso
     escreva "A categoria é Adulto")
 fim escolha
fim algoritmo
```

Algoritmo 06_03

```
var prato, sobremesa, calorias: inteiro
inicio
calorias:=0
escreva("Qual o prato principal: 1-Peixe, 2-Frango ou 3-Car-
ne")
leia (prato)
se prato = 1 então
   calorias := calorias +  230
senão
    se prato = 2 então
      calorias := calorias + 250
    senão
      calorias := calorias + 350
    fim se
fim se
escreva("Qual a sobremesa: 1- Abacaxi, 2-Mousse ou 3-Sorve-
te")
leia (sobremesa)
se sobremesa = 1 então
   calorias := calorias +  50
senão
    se sobremesa = 2 então
      calorias := calorias + 350
    senão
      calorias := calorias + 170
    fim se
fim se

escreva("O total de calorias ingeridas será de: ", calo-
rias)
fim algoritmo
```

Capitulo 7 - Estruturas de Repetição (Loop) Algoritmo 07_01:

```
var
   civil : caracter
   x, solteiro, casado, viuvo: inteiro
início

solteiro <- 0
casado <- 0
viuvo <- 0
para x de 1 ate 20 faca
  repita
      escreva("Item ",x," Escolha Estado Civil S(solteiro),
C(casado) ou V(viúvo): ")
     leia(civil)
  ate (civil="s") ou (civil="c") ou (cıvil="v")
  escolha civil
  caso "c"
     casado <- casado + 1
  caso "s"
     solteiro <- solteiro + 1
  outro caso
     viúvo <- viuvo + 1
  fim escolha
fim para
escreva(" Os Solteiros são: ", solteiro)
escreva(" Os Casados são: ", casado)

escreva(" Os Viúvos são: ", viuvo)
fim algoritmo
```

Algoritmo 07_02

```
var
 numero, pares, impares, c: inteiro
início
pares:= 0
para c de 1 ate 100 faça
     escreva (“Digite um numero: “)
     leia (numero)
     se numero/2=0 então
        pares:= pares + 1
     senão
      impares:= impares + 1
     fim se

fim para
escreva (“Total de pares: “, pares)
escreva (“Total de ímpares: “, impares)

fim algoritmo
```

Algoritmo 07_03

```
var contador, negativo: inteiro
      numero, soma:real

início
negativo:= 0
soma:= 0

para contador de 1 ate 20 faça
     escreva (“Entre com número positivo ou negativo”)
     leia (numero)
     se numero>0 então
        soma:= soma + numero
     senão
        se numero<0 então
           negativo:= negativo + negativo
        fim se
     fim se
fim para

escreva (“Total dos positivos é: “, soma)
escreva (“Total dos negativos é: “, negativo)
fim algoritmo
```

Capitulo8-VetoreseMatrizes Algoritmo: 0801

```
var
 L, c, t : inteiro
 nomes: vetor [5] de caracter
 salario: vetor [5] de real
início

para L de 1 ate 5 faça
     escreva (“Digite um nome: “)
     leia (nomes[L])
     para c de 1 ate t faça
        escreva (“Digite o salario: “)
        leia (salario[L])
        salario:= salario * 1.05
     fim para
fim para

para l de 0 ate 5 faca
   escreva (nomes[L],salario[L])
fim para

fim algoritmo
```