



# ALGORITMOS E LÓGICA DE PROGRAMAÇÃO I



Professora Me. Gislaïne Camila Lapasini Leal

## UNICESUMAR

Av. Guedner, 1610 - Jardim Aclimação  
Cep 87050-900 - MARINGÁ - PARANÁ  
unicesumar.edu.br  
44 3027.6360

## UNICESUMAR EDUCAÇÃO A DISTÂNCIA

NEAD - Núcleo de Educação a Distância  
Bloco 4 - MARINGÁ - PARANÁ  
unicesumar.edu.br  
0800 600 6360

as imagens utilizadas neste  
livro foram obtidas a partir  
dos sites PHOTOS.COM e  
SHUTTERSTOCK.COM

## FICHA CATALOGRÁFICA

CENTRO UNIVERSITÁRIO DE MARINGÁ. Núcleo de Educação a Distância:

C397

Algoritmos e lógica de programação I / Gislaine Camila Lapasini Leal.

Reimpressão revista e atualizada, Maringá - PR, 2014.

195 p.

"Graduação - EaD".

1. Algoritmos 2. Programação . 4. EaD. I. Título.

ISBN 978-85-8084-295-1

CDD - 22 ed. 005.1  
CIP - NBR 12899 - AACR/2

Ficha catalográfica elaborada pelo bibliotecário  
João Vivaldo de Souza - CRB-8 - 6828



### Reitor

Wilson de Matos Silva

### Vice-Reitor

Wilson de Matos Silva Filho

### Pró-Reitor de Administração

Wilson de Matos Silva Filho

### Pró-Reitor de EAD

Willian Victor Kendrick de Matos Silva

### Presidente da Mantenedora

Cláudio Ferdinandi

### NEAD - Núcleo de Educação a Distância

#### Direção de Operações

Chrystiano Mincoff

#### Coordenação de Sistemas

Fabício Ricardo Lazilha

#### Coordenação de Polos

Reginaldo Carneiro

#### Coordenação de Pós-Graduação, Extensão e Produção de Materiais

Renato Dutra

#### Coordenação de Graduação

Kátia Coelho

#### Coordenação Administrativa/Serviços Compartilhados

Evandro Bolsoni

#### Coordenação de Curso

Danillo Xavier Saes

#### Gerência de Inteligência de Mercado/Digital

Bruno Jorge

#### Gerência de Marketing

Harrison Brait

#### Supervisão do Núcleo de Produção de Materiais

Nalva Aparecida da Rosa Moura

#### Supervisão de Materiais

Nádila de Almeida Toledo

#### Design Educacional

Camila Zaguini Silva

Fernando Henrique Mendes

Rossana Costa Giani

#### Projeto Gráfico

Jaime de Marchi Junior

José Jhonny Coelho

#### Editoração

Thayla Daiany Guimarães Cripaldi

#### Revisão Textual

Jaquelina Kutsunugi, Keren Pardini, Maria

Fernanda Canova Vasconcelos, Nayara

Valenciano, Rhaysa Ricci Correa e Susana Inácio

#### Ilustração

Robson Yuiti Saito

Nara Emi Tanaka Yamashita



Professor  
Wilson de Matos Silva  
Reitor

Viver e trabalhar em uma sociedade global é um grande desafio para todos os cidadãos. A busca por tecnologia, informação, conhecimento de qualidade, novas habilidades para liderança e solução de problemas com eficiência tornou-se uma questão de sobrevivência no mundo do trabalho.

Cada um de nós tem uma grande responsabilidade: as escolhas que fizermos por nós e pelos nossos fará grande diferença no futuro.

Com essa visão, o Centro Universitário Cesumar – assume o compromisso de democratizar o conhecimento por meio de alta tecnologia e contribuir para o futuro dos brasileiros.

No cumprimento de sua missão – “promover a educação de qualidade nas diferentes áreas do conhecimento, formando profissionais cidadãos que contribuam para o desenvolvimento de uma sociedade justa e solidária” –, o Centro Universitário Cesumar busca a integração do ensino-pesquisa-extensão com as demandas institucionais e sociais; a realização de uma prática acadêmica que contribua para o desenvolvimento da consciência social e política e, por fim, a democratização do conhecimento acadêmico com a articulação e a integração com a sociedade.

Diante disso, o Centro Universitário Cesumar almeja ser reconhecido como uma instituição universitária de referência regional e nacional pela qualidade e compromisso do corpo docente; aquisição de competências institucionais para o desenvolvimento de linhas de pesquisa; consolidação da extensão universitária; qualidade da oferta dos ensinamentos presencial e a distância; bem-estar e satisfação da comunidade interna; qualidade da gestão acadêmica e administrativa; compromisso social de inclusão; processos de cooperação e parceria com o mundo do trabalho, como também pelo compromisso e relacionamento permanente com os egressos, incentivando a educação continuada.





Professora  
**Kátia Solange Coelho**

Coordenadora de  
Graduação do NEAD  
UniCesumar

Seja bem-vindo(a), caro(a) acadêmico(a)! Você está iniciando um processo de transformação, pois quando investimos em nossa formação, seja ela pessoal ou profissional, nos transformamos e, consequentemente, transformamos também a sociedade na qual estamos inseridos. De que forma o fazemos? Criando oportunidades e/ou estabelecendo mudanças capazes de alcançar um nível de desenvolvimento compatível com os desafios que surgem no mundo contemporâneo.

O Centro Universitário Cesumar mediante o Núcleo de Educação a Distância, o(a) acompanhará durante todo este processo, pois conforme Freire (1996): “Os homens se educam juntos, na transformação do mundo”.

Os materiais produzidos oferecem linguagem dialógica e encontram-se integrados à proposta pedagógica, contribuindo no processo educacional, complementando sua formação profissional, desenvolvendo competências e habilidades, e aplicando conceitos teóricos em situação de realidade, de maneira a inseri-lo no mercado de trabalho. Ou seja, estes materiais têm como principal objetivo “provocar uma aproximação entre você e o conteúdo”, desta forma possibilita o desenvolvimento da autonomia em busca dos conhecimentos necessários para a sua formação pessoal e profissional.

Portanto, nossa distância nesse processo de crescimento e construção do conhecimento deve ser apenas geográfica. Utilize os diversos recursos pedagógicos que o Centro Universitário Cesumar lhe possibilita. Ou seja, acesse regularmente o AVA – Ambiente Virtual de Aprendizagem, interaja nos fóruns e enquetes, assista às aulas ao vivo e participe das discussões. Além disso, lembre-se que existe uma equipe de professores e tutores que se encontra disponível para sanar suas dúvidas e auxiliá-lo(a) em seu processo de aprendizagem, possibilitando-lhe trilhar com tranquilidade e segurança sua trajetória acadêmica.

**Professora Me. Gislaine Camila Lapasini Leal**

Graduada em Engenharia de Produção - Software pela Universidade Estadual de Maringá (2007) e em Processamento de Dados pelo Centro Universitário de Maringá (2004), com mestrado em Ciência da Computação pela Universidade Estadual de Maringá (2010). Atualmente é professora do Departamento de Engenharia de Produção da Universidade Estadual de Maringá atuando na área de Apoio à Tomada de Decisão (Pesquisa Operacional e Gestão de Tecnologia da Informação).

**SEJA BEM-VINDO(A)!**

Caro(a) aluno(a)! Seja bem-vindo(a) à disciplina de Algoritmos e Lógica de Programação I. Sou a professora Gislaine Camila e nesta disciplina estudaremos o conceito central da programação, os algoritmos e lógica de programação.

A atividade de programar envolve a construção de algoritmos, sendo o ponto de partida para a construção de programas, isto é, é o componente básico de qualquer software. O aprendizado de algoritmos é crucial para o desenvolvimento de software de qualidade.

Os algoritmos são utilizados para a solução de um problema, mas não constituem a única solução para um problema. Podemos ter vários algoritmos que resolvem o mesmo problema. De modo que não há uma receita a ser seguida na construção de algoritmos, o que devemos fazer é desenvolver o nosso raciocínio lógico a encadear pensamentos para atingir um objetivo.

Para desenvolver a lógica de programação precisamos colocar em prática os conceitos adquiridos. Não basta apenas observar ou copiar, é necessário praticar, construir algoritmos. Lembre-se, a prática é fundamental para o sucesso no processo de aprendizagem de algoritmos.

Apresento a você o livro que norteará seus estudos nesta disciplina, auxiliando no aprendizado de algoritmos e lógica de programação. Em cada unidade construiremos algoritmos passo a passo e você verá que à medida que avançamos aumentamos a gama de problemas que conseguimos resolver.

Na Unidade I estudaremos os conceitos e princípios básicos de lógica de programação e veremos que empiricamente já conhecemos algoritmos e utilizamos vários deles em nosso dia a dia. Estudaremos os tipos de algoritmos, conceito de variáveis, tipos de variáveis, constantes, expressões, funções intrínsecas, comando para atribuição, entrada e saída de dados. A partir destes conteúdos iniciaremos a construção de nossos primeiros algoritmos que resolvem problemas.

A Unidade II abordará o conceito de estrutura condicional, que nos possibilita impor condições para a execução de uma determinada instrução ou conjunto de instruções. Discutiremos sobre os tipos de estrutura condicional simples, composta, aninha e estrutura de decisão múltipla, destacando a sintaxe e como utilizar cada uma delas. Ao longo desta unidade construiremos algoritmos com cada um dos tipos de estrutura condicional. Com os conceitos desta unidade poderemos construir algoritmos com desvios de fluxo, aumentando o leque de problemas que conseguimos resolver.

Na Unidade III aprenderemos a construir algoritmos utilizando a estrutura de repetição para repetir um trecho de código quantas vezes forem necessárias sem ter que reescrever trechos idênticos. Discutiremos o funcionamento, como utilizar e como encadear as estruturas de repetição controladas e condicionais. Colocaremos todos os conceitos desta unidade em prática construindo algoritmos.

A Unidade IV trata sobre as estruturas de dados homogêneas e heterogêneas, as quais

# APRESENTAÇÃO

permitem agrupar diversas informações em uma única variável. Em relação às estruturas de dados homogêneas unidimensionais (vetores) e multidimensionais (matrizes) estudaremos como realizar a atribuição de valores, leitura, entrada e saída de dados neste tipo de estrutura. Conheceremos métodos que nos permitem classificar as informações de acordo com um critério e realizar pesquisa. Discutiremos, também, as estruturas de dados heterogêneas, abordando como realizar atribuição, entrada e saída de dados utilizando registros.

Por fim, na Unidade V, estudaremos a modularização de algoritmos utilizando sub-rotinas e a manipulação de arquivos. Conheceremos as sub-rotinas do tipo procedimento e função, sua sintaxe, peculiaridades e como utilizá-las. Veremos os conceitos relacionados ao escopo de variáveis e a passagem de parâmetros por valor e por referência, recursividade, construção de funções recursivas e aplicações de recursão. Trataremos, também, o conceito de arquivos, modos de concepção e como manipulá-los utilizando operações de que possibilitem consultar, inserir, modificar e eliminar dados.

Ao longo deste livro você encontrará indicações de leitura complementar as quais enriquecerão o seu conhecimento com mais exemplos de algoritmos. Há o momento de reflexão denominado REFLITA, que permite que você pense com calma sobre um determinado subtema. É importante que você desenvolva as atividades de autoestudo, pois é o momento que você tem para colocar em prática os conhecimentos adquiridos e identificar eventuais dificuldades.

Desejo a você um bom estudo!



# SUMÁRIO

## UNIDADE I

### CONCEITOS BÁSICOS

15	Introdução	
16	Conceituando Algoritmos	
19	Como Construir Algoritmos	
19	Tipos de Algoritmos	
22	Estudando Variáveis	
25	Tipos de Variáveis	
26	Constante	
27	Expressões	
30	Funções Intrínsecas	
31	Atribuição	
31	Entrada de Dados	
32	Saída de Dados	
32	Construindo Algoritmos	
43	Considerações Finais	

## UNIDADE II

### ESTRUTURA CONDICIONAL

57	Introdução	
58	Estrutura Condicional	
59	Estrutura Condicional Simples	



# SUMÁRIO

64	Estrutura Condicional Composta
67	Estrutura Condicional Aninhada
71	Estrutura de Decisão Múltipla
75	Considerações Finais

## ■ UNIDADE III

### ESTRUTURA DE REPETIÇÃO

89	Introdução
90	Estrutura de Repetição
91	Estrutura Para
96	Estrutura Enquanto
102	Estrutura Repita
104	Estruturas de Repetição Encadeadas
114	Considerações Finais

## ■ UNIDADE IV

### ESTRUTURAS DE DADOS HOMOGÊNEAS E HETEROGÊNEAS

129	Introdução
130	Estruturas de Dados Homogêneas
130	Vetores
134	Ordenação em Vetor
140	Busca em Vetor



# SUMÁRIO

140	Método Sequencial
142	Matrizes
145	Estruturas de Dados Heterogêneas
145	Registros
151	Problema
154	Considerações Finais

## UNIDADE V

### SUB-ROTINAS E PROGRAMAÇÃO COM ARQUIVOS

167	Introdução
168	Sub-Rotinas
169	Procedimentos
173	Escopo de Variáveis
174	Passagem de Parâmetros
180	Funções
181	Recursividade
184	Trabalhando com Arquivos
187	Considerações Finais
195	<b>Conclusão</b>
197	<b>Referências</b>





# CONCEITOS BÁSICOS

UNIDADE

I

## Objetivos de Aprendizagem

- Conhecer os princípios básicos de lógica de programação.
- Entender os passos para a construção de algoritmos.
- Aperfeiçoar o raciocínio lógico.
- Conhecer os tipos de dados.
- Conhecer os comandos de atribuição, entrada e saída de dados.

## Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Conceituando algoritmos
- Como construir algoritmos
- Tipos de algoritmos
- Estudando variáveis
- Tipos de variáveis
- Constantes
- Expressões
- Funções Intrínsecas
- Atribuição
- Entrada de dados
- Saída de dados
- Construindo algoritmos
- Problema 1
- Problema 2
- Problema 3



## INTRODUÇÃO

Nesta unidade você aprenderá os princípios básicos de algoritmos e lógica de programação. Algoritmos constituem o conceito central da programação e a atividade de programar envolve a construção de algoritmos. Em geral, são utilizados para a solução de um problema, contudo, não se constituem em única solução para um problema e podemos ter vários algoritmos que resolvem o mesmo problema.

Primeiramente, será abordado o conceito de algoritmos e você descobrirá que empiricamente já conhece e utiliza algoritmos no seu cotidiano. Estudaremos tipos de algoritmos e as formas que podemos utilizar para representar a solução do problema e, especificamente, trataremos sobre a representação por meio de linguagem natural, simbologia e linguagem restrita.

Para facilitar o processo de ensino-aprendizagem será apresentado um método para a construção de algoritmos que divide os problemas em três partes: Entrada – Processamento – Saída. Esse método nos auxiliará a sistematizar as informações para a construção de algoritmos.

Ao formular algoritmos precisamos guardar algumas informações do problema. Para isso veremos o conceito de variáveis e constantes. Conheceremos os tipos de variáveis numéricas, literais e lógicas e a característica da informação que cada uma delas pode armazenar. Além de guardar informações precisaremos obter dados, mostrar mensagens e resultados de processamento. Desta forma, estudaremos os comandos relacionados à entrada de dados que nos permitem interagir como usuário; atribuição que possibilita atribuir valor às variáveis; e, saída de dados que viabiliza o envio de mensagens e a exibição dos resultados do processamento.

Construiremos três algoritmos para visualizar a aplicação de cada um dos conceitos abordados. Após estudar esta unidade você poderá responder às questões relacionadas com o tema de Algoritmos e Lógica de Programação I, tais como: Que informações são de entrada e quais são de saída? Que tipo de variável devo utilizar? Como exibo um resultado ao usuário? Qual o resultado da simulação do algoritmo?

Além disto, você aprenderá os conceitos de variáveis, constantes, expressões, funções, atribuição, comando de entrada, comando de saída e técnicas de raciocínio lógico para a solução de problemas. Todo esse conhecimento você poderá utilizar para construir os primeiros algoritmos que resolvem problemas! Vamos lá?!

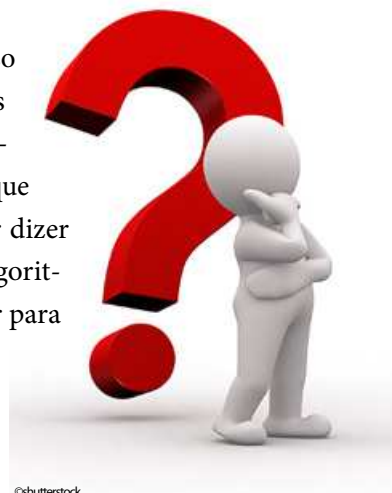
## CONCEITUANDO ALGORITMOS

Para dar início a nossa disciplina de Algoritmos e Lógica de Programação I precisamos entender o conceito de algoritmos. O que vem a ser um algoritmo? Você já ouviu esse termo?

Um algoritmo consiste em uma sequência finita de passos (instruções) para solucionar um problema. Podemos ter vários algoritmos que resolvem um mesmo problema, desta forma um algoritmo não é a única solução de um problema. Um algoritmo é um caminho para a solução de um problema, visto que podem existir diversos caminhos que conduzem à solução (LOPES; GARCIA, 2002).

Ao resolver algoritmos vamos construindo a nossa própria lógica de programação. Não há receita mágica e o aprendizado de algoritmos requer prática. Mas não se preocupe! No decorrer deste livro teremos contato com diversos exemplos e atividades de autoestudo.

Se pararmos um pouco para pensar, em nosso cotidiano encontramos uma série de problemas os quais demandam por uma solução. E um algoritmo nada mais é do que um conjunto de passos que resolvem um determinado problema. Isto quer dizer que empiricamente já conhecemos diversos algoritmos. Tomemos como exemplo o problema de ir para o trabalho (Quadro 1)



©shutterstock



1. Desligar o despertador
2. Ir para o banheiro
3. Lavar o rosto
4. Escovar os dentes
5. Tirar o pijama
6. Colocar uma roupa
7. Tomar café da manhã
8. Pegar o carro
9. Estacionar o carro
10. Entrar no escritório

Quadro 1: Algoritmo para ir ao trabalho

Observando o algoritmo para ir ao trabalho verificamos que em cada linha temos uma instrução. Podemos dizer que o algoritmo para ir ao trabalho tem dez instruções. Portanto, um algoritmo é um conjunto de instruções. Mas, o que é uma instrução? Uma instrução é uma operação básica (sem ambiguidade) que indica a um computador a ação que deve ser executada (SALVETTI; BARBOSA, 2004).

Agora que estamos mais familiarizados com o termo algoritmo podemos perceber que eles fazem parte do nosso dia a dia, por exemplo, quando seguimos instruções para uso de um medicamento, realizamos uma ligação telefônica, trocamos uma lâmpada, montamos um móvel ou aparelho ou até mesmo quando seguimos uma receita culinária (ZIVIANI, 2004).

O Quadro 2 apresenta o conjunto de instruções a serem seguidas para tomar sal de frutas (uso de medicamento).

1. Colocar 2/3 de água em um copo
2. Dissolver o sal de frutas
3. Esperar o efeito efervescente
4. Tomar a solução

Quadro 2: Algoritmo para tomar sal de frutas

A sequência de passos a ser seguida para realizar uma ligação telefônica é apresentada no Quadro 3.

1. Inserir o número
2. Apertar o botão para fazer ligação
3. Esperar atender
4. Falar no telefone
5. Apertar o botão para desligar ligação

Quadro 3: Algoritmo para fazer ligação telefônica  
Fonte: adaptado de (LOPES; GARCIA, 2002)

Uma solução para o problema de trocar uma lâmpada é representada por cinco instruções, como pode ser visto no Quadro 4.

1. Se (lâmpada estiver fora de alcance) Pegar uma escada
2. Pegar a lâmpada
3. Se (lâmpada estiver quente) Pegar um pano
4. Tirar a lâmpada queimada
5. Colocar a lâmpada boa

Quadro 4: Algoritmo para trocar lâmpada  
Fonte: LOPES; GARCIA (2002)

Como já sabemos o que é um algoritmo, outro conceito importante que precisamos compreender é o de programa de computador. Um programa nada mais é que uma sequência de instruções codificada em uma linguagem que pode ser seguida por um computador. É a representação de um algoritmo em uma linguagem de programação (C, Pascal, Java, Fortran etc.) (ZIVIANE, 2004; SALVETTI; BARBOSA, 2004).

## COMO CONSTRUIR ALGORITMOS

Sabemos que a construção de algoritmos requer prática. Para facilitar nosso processo de aprendizagem, Ascencio e Campos (2010) descrevem alguns passos que devemos seguir, sendo eles:

- Compreender o problema: definir qual o objetivo do algoritmo.
- Definir as informações de entrada: que informações precisamos obter do usuário.
- Definir o processamento: que cálculos devemos efetuar. É neste momento que os dados obtidos pela entrada serão transformados em informação útil para o usuário.
- Definir as informações de saída: que informações devemos fornecer ao usuário como resultado do processamento efetuado.

A separação do problema em **Entrada** (Que dados devemos obter do usuário?), **Processamento** (Que cálculos devemos efetuar?) e **Saída** (Que informações devemos fornecer ao usuário?) nos auxilia no processo de construção do raciocínio lógico, facilitando assim o nosso aprendizado de Algoritmos e Lógica de Programação I.

## TIPOS DE ALGORITMOS

Ascencio e Campos (2010) destacam que os tipos de algoritmos mais utilizados são: descrição narrativa, fluxograma e pseudocódigo.

A **descrição narrativa** consiste na representação do problema por meio da linguagem natural, descrevendo os passos que devem ser seguidos para a resolução de um problema, conforme foi visto no tópico “CONCEITUANDO ALGORITMOS”. Como vantagem dessa representação destaca-se a facilidade de aprendizado. No entanto, esse tipo de descrição pode ser ambígua, gerar diversas

interpretações e dificultar a conversão do algoritmo em um programa de computador (ASCENCIO; CAMPOS, 2010).

Vamos tomar como exemplo a soma de dois números. Seguindo os passos descritos na seção “Como Construir Algoritmos” temos que:






- Objetivo: somar dois números.
- Dados de entrada: obter do usuário quais são os dois números que devemos somar.
- Processamento: efetuar a operação de soma com os dois números obtidos.
- Saída: mostrar o resultado da soma.

Agora que estruturamos o nosso problema em Entrada – Processamento – Saída ficou mais fácil construir o nosso algoritmo. A representação do algoritmo como descrição narrativa pode ser visualizada no Quadro 5.

1. Obter dois números
2. Somar os dois números
3. Mostrar o resultado da soma

Quadro 5: Descrição Narrativa – Algoritmo Soma

O **fluxograma** consiste em uma notação gráfica que permite indicar as ações e decisões que devem ser seguidas para resolver o problema. Os símbolos utilizados para construir o fluxograma são apresentados no Quadro 6.

Símbolo	Descrição
	Indica o início e o fim do algoritmo.
	Indica o sentido do fluxo de dados.
	Indica a realização de cálculos e operações de atribuição.
	Representa a saída de dados.
	Indica que deve ser tomada uma decisão, há possibilidade de desvio do fluxo.

Quadro 6: Simbologia de Fluxograma

Fonte: adaptado de MANZANO; OLIVEIRA (1997)

Para melhor compreensão, apresenta-se na Figura 1, fluxograma para o problema de somar dois números.

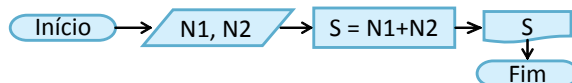


Figura 1: Fluxograma – Algoritmo Soma

Fonte: a autora

A vantagem do fluxograma está em facilitar o entendimento dos símbolos e como desvantagem destaca-se o fato de o algoritmo não apresentar muitos detalhes, o que irá dificultar a conversão do algoritmo em um programa (ASCENCIO; CAMPOS, 2010).

Outro tipo de algoritmo é o **pseudocódigo**, conhecido também como portugol ou português estruturado, que consiste em representar o problema por meio de regras pré-definidas. É uma linguagem restrita que descreve o significado para todos os termos utilizados nas instruções.

O Quadro 7 apresenta o pseudocódigo do algoritmo que efetua a soma de dois números. As palavras em negrito são termos que possuem significado específico. Notem que todo algoritmo possui um nome e é uma sequência finita, sendo delimitado por um início e por um fim.

```
Algoritmo soma
    Var n1, n2, s: inteiro
Início
    Leia n1, n2
    s ← n1 + n2
    Escreva (s)
Fim
```

Quadro 7: Pseudocódigo – Algoritmo Soma

O pseudocódigo é muito próximo à linguagem de programação, de tal modo que a passagem do algoritmo para o programa é quase imediata, bastando conhecer as palavras reservadas da linguagem, conforme destaca Ascencio e Campos (2010). A desvantagem está relacionada ao fato de ter que aprender as regras do pseudocódigo.

## ESTUDANDO VARIÁVEIS

Para elaborar algoritmos precisamos guardar algumas informações e para isto utilizamos as variáveis. Uma variável é um espaço na memória principal do computador que pode conter diferentes valores a cada instante de tempo (LOPES;GARCIA, 2002).

Uma variável pode ser vista como uma caixa que armazena pertences. Esta caixa tem um nome e somente guarda objetos do mesmo tipo. Uma variável possui um nome e seu conteúdo pode ser de vários tipos: inteiro, real, caractere, lógico entre outros.

Na Figura 2 temos uma variável com nome Idade. Essa variável pode guardar apenas valores inteiros. Com isto, temos que o valor “Casa” não pode ser armazenado nesta caixa, visto que se trata de um conjunto de caracteres. Em um algoritmo o conteúdo de uma variável pode ser modificado, consultado ou apagado quantas vezes forem necessárias. É importante ter ciência de



©shutterstock

que a variável armazena apenas um conteúdo por vez. Na Figura 2, nossa caixa denominada Idade recebe uma sequência de valores sendo eles 25, 60, 13 e 36. Sabemos que uma variável armazena apenas um conteúdo por vez, desta forma quando realizarmos uma consulta obteremos o valor 36 para a variável Idade.

As variáveis são definidas logo no início do algoritmo para que a área na memória seja alocada. A definição de variáveis é realizada utilizando o comando VAR, primeiro definimos o nome e, em seguida, o tipo, do seguinte modo:

**Var** <nome da variável> : <tipo da variável>

Ao declarar variáveis devemos tomar alguns cuidados: a palavra VAR é usada uma única vez na definição de variáveis; mais de uma variável do mesmo tipo pode ser definida em uma mesma linha, basta separar cada uma delas por vírgula; e, se há diferentes tipos de variáveis, cada tipo deve ser declarado em linhas diferentes.

Os identificadores são os nomes das variáveis, programas, rotinas, constantes etc. Para nomear esses

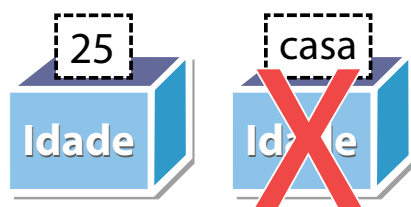


Figura 2: Variável  
Fonte: a autora

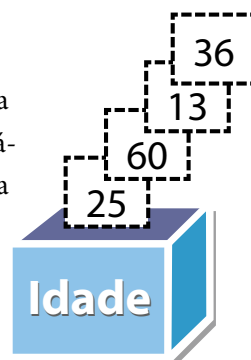


Figura 3: Variável  
Fonte: a autora

identificadores devemos seguir algumas regras (ASCENCIO; CAMPOS, 2010):

- O nome deve iniciar SEMPRE com letra. Isto indica que nossas variáveis podem ser chamadas de media, altura, idade, cidade. Mas, não pode ser 2cidade, 4x etc.
- O nome não pode conter espaços, ou seja, não podemos denominar uma variável de altura media.
- O nome não pode conter caracteres especiais (\$, #, @, ?, !, \*).
- Nenhuma palavra reservada poderá ser nome de variável. As palavras reservadas têm uso específico no pseudocódigo. Alguns exemplos são: Var, tipo, início, fim, se, então, senão, enquanto, repita, faça, caso, até, procedimento, função e outros. Não se assuste com todos esses termos, iremos vê-los no decorrer desta unidade e com a prática logo você estará familiarizado com todos eles.

O Quadro 8 apresenta alguns exemplos de identificadores válidos e inválidos, exemplificando cada uma das regras descritas anteriormente. É importante lembrar que seguiremos essas regras para nomear variáveis, constantes, algoritmos, rotinas e outros.

IDENTIFICADORES VÁLIDOS	IDENTIFICADORES INVÁLIDOS
A	2 <sup>a</sup>
a	b@
media	media idade
altura2	caso
media_idade	se
x36	x*y

Quadro 8: Exemplo de identificadores

Até o momento sabemos o que é um algoritmo, o que são variáveis e qual a sua importância. Na seção seguinte iremos entender quais os tipos de variáveis que dispomos para construir nossos algoritmos.



## TIPOS DE VARIÁVEIS

Como sabemos, as nossas variáveis só podem armazenar dados do mesmo tipo. Os tipos de variáveis são: inteiro, real, caractere e lógico.

Uma variável **inteira** armazena dados numéricos que não possuem componentes decimais ou fracionários. Podendo o número ser positivo ou negativo (ASCENCIO; CAMPOS, 2010). Alguns exemplos de números inteiros são: -3, -1, 0, 4, 7. Desta forma, a declaração de uma variável inteira é realizada da seguinte forma:

```
Var
Idade : Inteiro
```

Outro tipo de variável numérica é a **real**, que armazena componentes decimais ou fracionários, podendo estes ser positivos ou negativos. Alguns exemplos de números reais são: 15.02, 4.5, 6.07 e outros. A declaração de uma variável do tipo real é dada por:

```
Var
Altura : Real
```

Neste ponto você pode estar se perguntando: que diferença faz declarar uma variável como inteira ou real? Declaro todas as variáveis como real já que o conjunto dos inteiros está incluso nos reais?

A diferença está no tamanho do espaço de memória utilizado. Normalmente, uma variável inteira pode ocupar 1, 2 ou 4 bytes. Enquanto uma variável real poderá ocupar 4 ou 8 bytes (LOPES; GARCIA, 2002). Isto nos indica que se alocarmos todas as variáveis como real, estaremos alocando um espaço de memória desnecessário e sabemos que uma série de informações não assumem valores decimais ou fracionários. Por exemplo, quando perguntamos a idade para uma pessoa, obtemos como resposta 26 anos e não 26,3 anos. Desta forma, para armazenar a idade utilizamos sempre variáveis do tipo inteiro. Do mesmo modo, existem informações que são necessariamente do tipo real, tais como altura, peso, notas, médias e outras.

Até o momento conhecemos as variáveis numéricas, no entanto existem dados literais que devemos processar, para tanto utilizamos as variáveis do tipo

**caractere**, também conhecida como literais, alfanumérico, *string* ou cadeia (MANZANO; OLIVEIRA, 1997).

Um caractere é uma variável que armazena dado que contém letras, dígitos e/ou símbolos especiais (LOPES; GARCIA, 2002). Alguns exemplos são: “João”, “1234567”, “E”, “21/06/1984”.

As variáveis do tipo caractere possuem um comprimento associado, por exemplo, “Maria” tem comprimento 5, “21/06/1984” tem comprimento 10. O comprimento corresponde ao número de caracteres que a variável possui. Ao declarar uma variável do tipo caractere podemos definir o tamanho máximo de caracteres que ela poderá armazenar, para tanto basta inserir o tamanho entre colchetes. A declaração de uma variável do tipo caractere é realizada da seguinte forma:

Var

Nome : **Caractere** [tamanho]

Há o tipo de variável **lógica** ou **booleana**, que possui dois únicos valores lógicos: verdadeiro ou falso, ou, 1 e 0. Para declarar uma variável lógica realizamos o seguinte procedimento:

Var

Ocupado : **Lógico**

## CONSTANTE

Uma constante armazena informações que não variam com o tempo, ou seja, o seu conteúdo é um valor fixo. Da mesma forma que as variáveis, todas as constantes devem ser definidas no início do algoritmo. O comando utilizado para definir constantes é o **CONST** e sua definição é dada por:

## CONST

<nome da constante> = <valor>

## EXPRESSÕES

De acordo com Lopes e Garcia (2002), as expressões estão diretamente relacionadas ao conceito de fórmula matemática, em que um conjunto de variáveis e constantes relaciona-se por meio de operadores. As expressões dividem-se em: aritméticas, relacional, lógicas e literais.

As expressões **aritméticas** são aquelas em que o resultado consiste em um valor numérico. Desta forma, apenas operadores aritméticos e variáveis numéricas (inteiro e real) podem ser utilizados em expressão desse tipo.

O Quadro 9 apresenta os operadores aritméticos, destacando suas representações, forma de uso e prioridade. A prioridade entre operadores define a ordem em que os mesmos devem ser avaliados dentro de uma mesma expressão.

OPERAÇÃO	OPERADOR	SIGNIFICADO	PRIORIDADE
Soma	+	Utilizado para efetuar a soma de duas ou mais variáveis. Dada uma variável A e outra B, temos que a soma delas é representada por $A + B$ .	4
Subtração	-	Simboliza a subtração do valor de duas variáveis. Supondo uma variável A e B, a diferença entre elas é dada por: $A - B$ .	4
Multiplicação	*	O produto entre duas variáveis A e B é representado por $A * B$ .	3
Divisão	/	A divisão entre duas variáveis A e B é dada por: $A/B$ . Em relação à divisão é importante lembrar que não existe divisão por zero.	3

OPERAÇÃO	OPERADOR	SIGNIFICADO	PRIORIDADE
Exponenciação	**	É representada por ** mais o número que se quer elevar. Se quisermos elevar o valor da variável A ao quadrado, representamos por: A ** 2.	2
Resto	mod	Usado quando se deseja encontrar o resto da divisão entre duas variáveis A e B. A representação é dada por A mod B. Supondo A = 3 e B = 2, temos que A mod B = 1.	2
Divisão inteira	div	Representa o quociente da divisão entre dois números. Tomando A = 7 e B = 2, temos que A div B resulta em 3.	2

Quadro 9: Operadores aritméticos

As expressões **relacionais** referem-se à comparação entre dois valores de um tipo básico. Os operadores relacionais são destacados no Quadro 10, em que é possível visualizar o operador, símbolo associado e forma de uso.

OPERAÇÃO	OPERADOR	SIGNIFICADO
Igual	=	A = 1
Diferente	<>	A <> B
Maior	>	A > 5
Menor que	<	B < 12
Maior ou igual a	>=	A >= 6
Menor ou igual a	<=	B <= 7

Quadro 10: Operadores relacionais

Fonte: adaptado de (ASCENCIO; CAMPOS, 2010)

As expressões lógicas são aquelas cujo resultado consiste em um valor lógico verdadeiro ou falso. Neste tipo de expressão podem ser usados os operadores relacionais, os operadores lógicos ou expressões matemáticas.

No Quadro 11 são descritos cada um dos operadores lógicos: conjunção, disjunção e negação.

Operador	Símbolo	Explicação	Prioridade
Disjunção	OU	A disjunção entre duas variáveis resulta em um valor verdadeiro quando pelo menos uma das variáveis é verdadeira.	3
Conjunção	E	A conjunção entre duas variáveis resulta em um valor verdadeiro somente quando as duas variáveis são verdadeiras.	2
Negação	NÃO	A negação inverte o valor de uma variável. Se a variável A é verdadeira, a negação de A, torna o valor da variável falso.	1

Quadro 11: Operadores lógicos

Fonte: adaptado de (LOPES; GARCIA, 2002)

As expressões literais consistem em expressões cujo resultado é um caractere. Há um único operador literal, o +, que é utilizado na concatenação de duas ou mais variáveis do tipo caractere. Esse operador é utilizado para acrescentar o conteúdo de uma variável ao final de outra. Por exemplo, a concatenação de “ALGO” + “RITMO” resulta em “ALGORITMO”.

Em uma expressão podemos ter mais de um operador. Em situações que há um único operador a avaliação da expressão é realizada de forma direta. Quando há mais de um operador é necessária a avaliação da expressão passo a passo, ou seja, um operador por vez. A precedência das operações é realizada por meio de parênteses e de acordo com a prioridade dos operadores.

Por exemplo, temos:  $x + y * 25$ . Neste caso, a primeira operação a ser realizada é a multiplicação de y por 25, visto que o operador de multiplicação tem prioridade 3 e a soma prioridade 4. Se quisermos multiplicar o produto da soma de x e y por 25 temos que representar a expressão do seguinte modo:  $(x+y) * 25$ . Neste caso, primeiro é executada a operação que está entre os parênteses e depois a multiplicação, ou seja, os parênteses têm precedência.

Fique tranquilo! Colocaremos em prática as questões relacionadas à precedência de operadores na seção Construindo Algoritmos.

## FUNÇÕES INTRÍNSECAS

As funções são fórmulas matemáticas prontas que podemos utilizar em nossos algoritmos. O Quadro 12 apresenta as principais funções, destacando o comando associado e o que ela faz.

Função	Objetivo
ABS(variável)	Retorna o valor absoluto de um número qualquer.
ARCTAN(variável)	Retorna o arco tangente de um ângulo qualquer em radianos.
COS(variável)	Retorna o valor do cosseno de um ângulo qualquer em radianos.
EXP(variável)	Retorna o valor exponencial, ou seja, o número é elevado a um número qualquer.
FRAC(variável)	Retorna a parte fracionária.
LN(variável)	Retorna o logaritmo natural.
PI	Retorna o valor de PI.
SIN(variável)	Retorna o valor do seno de um ângulo qualquer em radianos.
SQR(variável)	Retorna o valor do parâmetro elevado ao quadrado.
SQRT(variável)	Retorna a raiz quadrada de um número positivo.

Quadro 12: Funções

Fonte: adaptado de ASCENCIO; CAMPOS (2010); LOPES; GARCIA (2002)

## ATRIBUIÇÃO

Já sabemos o que são variáveis, quais os tipos de variáveis e expressões que podemos associar a elas, agora precisamos entender como armazenar um dado em uma variável. A atribuição consiste no processo de fornecer um valor a uma variável, em que o tipo desse valor tem que ser compatível com a variável (LOPES; GARCIA, 2002).

O símbolo utilizado para a atribuição é  $\leftarrow$  e a representação é dada por:

identificador  $\leftarrow$  expressão

O identificador representa a variável a qual será atribuído um valor e o termo expressão é o que será atribuído, podendo ser uma expressão aritmética, lógica ou literal. Tomemos como exemplo atribuir a uma variável denominada idade o valor 26. Para isso, temos que:

idade  $\leftarrow$  26

A leitura dessa instrução é realizada do seguinte modo: a variável idade recebe o valor 26.

## ENTRADA DE DADOS

A entrada de dados permite receber os dados digitados pelo usuário e é realizada por meio do comando **leia**. Os dados recebidos são armazenados em variáveis (ASCENCIO; CAMPOS, 2010).

A sintaxe do comando é:

**leia** <variável>

Ao utilizarmos esse comando o computador fica “aguardando” uma ação do usuário, que é digitar o valor para a variável.

## SAÍDA DE DADOS

A saída de dados permite mostrar dados aos usuários. O comando utilizado é o **escreva**, que busca as informações na memória e posteriormente as disponibiliza por meio de um dispositivo de saída (monitor ou impressora).

A sintaxe do comando de saída é:

**escreva**                    <variável> ou <literal>

Com o comando de saída podemos enviar mensagens ao usuário, informando que ação estamos esperando ou enviar resultados dos dados processados. Um literal deve ser escrito sempre entre aspas, por exemplo:

**Escreva** "Estou estudando Algoritmos e Lógica de Programação I"

Podemos imprimir diversas variáveis ou combinar variáveis com literais em um único comando, basta separá-las por vírgula. Por exemplo:

**Escreva** "A idade é:", idade

**Escreva** n1, "x" n2, "é igual a", produto

## CONSTRUINDO ALGORITMOS

Já vimos o conceito de variáveis, constantes, expressões, conhecemos os operadores, funções, comandos de atribuição, entrada e saída de dados. Nesta seção reuniremos todos esses conhecimentos na construção de algoritmos.



©shutterstock



## PROBLEMA 1

Formular um algoritmo que leia e apresente as seguintes informações de uma pessoa: nome, idade, peso, altura e telefone.

Antes de partir para a elaboração do algoritmo dividiremos o problema tal como aprendemos na seção “COMO CONSTRUIR ALGORITMOS”, em objetivo do algoritmo, entrada, processamento e saída. Deste modo, temos que:

- **Objetivo do algoritmo:** ler e apresentar o nome, idade, peso, altura e telefone de uma pessoa.
- **Entrada:** precisamos obter os dados: nome da pessoa, a idade, o peso, a altura e o telefone.
- **Processamento:** não há processamento.
- **Saída:** informar o nome da pessoa, a idade, o peso, a altura e o telefone.

Estruturar o problema nestes quatro itens auxilia no processo de compreensão e, por conseguinte, na construção do algoritmo. Após entender o problema devemos analisar que variáveis são necessárias e qual o tipo de cada uma delas. Os dados que precisamos obter e armazenar são: nome, idade, peso, altura e telefone. Desta forma, temos uma variável associada a cada um. Sabemos que idade, peso e altura são variáveis numéricas, sendo que a idade é inteiro e os demais real. Para armazenar o nome e telefone, uma informação literal, utilizaremos a variável do tipo caractere.

Iniciando a formulação do algoritmo, temos que ele precisa de um nome o qual denominaremos de problema1. Portanto, a primeira linha será:

Quadro 64: Pseudocódigo – Algoritmo registro aluno

Em seguida, temos a declaração das variáveis, que é dada por:

**Var**

Idade : **inteiro**

Peso, altura : **real**

Nome : **caractere[50]**

Telefone : **caractere[15]**

Note que as variáveis do mesmo tipo podem ser declaradas na mesma linha, como é o caso de peso e altura. Se as variáveis do tipo caractere tivessem o mesmo tamanho poderíamos declará-las de modo análogo. Declarar como caractere[50] indica que o tamanho máximo que a variável irá armazenar é de 50 caracteres. Todo algoritmo é delimitado pela instrução de **Início**, logo:

**Início**

Na entrada de dados precisamos obter as informações definidas por meio do comando **leia**. No entanto, é importante enviar sempre uma mensagem informando ao usuário que dado está sendo solicitado, deste modo utilizamos o comando **escreva**. Com isto, temos:

```
Escreva ("Informe o nome:")  
Leia (nome)  
Escreva ("Informe a idade:")  
Leia (idade)  
Escreva ("Informe o peso:")  
Leia (peso)  
Escreva ("Informe a altura:")  
Leia (altura)  
Escreva ("Informe o telefone:")  
Leia (telefone)
```

Como não há processamento, realizaremos a saída de dados por meio do comando **escreva**:

```
Escreva ("O nome é:", nome)  
Escreva ("A idade é:", idade)  
Escreva ("O peso é:", peso)  
Escreva ("A altura é:", altura)  
Escreva ("O telefone é:", telefone)
```

Ao término do algoritmo inserimos a instrução fim, do seguinte modo:

**Fim.**

O algoritmo inteiro é apresentado no Quadro 13. Observe que as palavras destacadas (negrito) são palavras reservadas, ou seja, não podemos utilizá-las como nome de variável, constante, algoritmo e outros.

```

Algoritmo problema1

Var

    Idade : inteiro
    Peso, altura : real
    Nome : caractere[50]
    Telefone : caractere[15]

Início

    Escreva ("Informe o nome:")
    Leia (nome)
    Escreva ("Informe a idade:")
    Leia (idade)
    Escreva ("Informe o peso:")
    Leia (peso)
    Escreva ("Informe a altura:")
    Leia (altura)
    Escreva ("Informe o telefone:")
    Leia (telefone)
    Escreva ("O nome é:", nome)
    Escreva ("A idade é:", idade )
    Escreva ("O peso é:", peso)
    Escreva ("A altura é:", altura)
    Escreva ("O telefone é:", telefone)

Fim.

```

Quadro 13: Pseudocódigo - Problema1

O resultado da execução do algoritmo é apresentado no Quadro 14.

```
Informe o nome: Maria da Silva
Informe a idade: 30
Informe o peso: 60.3
Informe a altura: 1.58
Informe o telefone: 3011-1212
O nome é: Maria da Silva
A idade é: 30
O peso é: 60.3
A altura é: 1.58
O telefone é: 3011-1212
```

Quadro 14: Resultado da execução - Problema 1

Na resolução deste problema colocamos em prática o conceito de variáveis, tipos de variáveis e comandos de entrada e saída de dados.

## PROBLEMA 2

Um quadrado (Figura 4) é uma figura geométrica com quatro lados de mesmo comprimento ( $l$ ) e quatro ângulos retos. Elabore um algoritmo para calcular a área e o perímetro de um quadrado. Observando a Figura 4, a área é dada pela superfície (laranja) e o perímetro é a medida do contorno do objeto, a soma dos quatro lados (pontilhado).

Não iremos elaborar o pseudocódigo diretamente, primeiro vamos dividir o problema tal como aprendemos na seção “COMO CONSTRUIR ALGORITMOS”, em objetivo do algoritmo, entrada, processamento e saída. Com isso, temos que:

- **Objetivo do algoritmo:** calcular o perímetro e área de um quadrado.

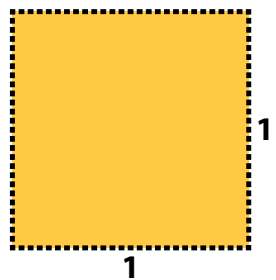


Figura 4: Quadrado  
Fonte: a autora

- **Entrada:** para calcular o perímetro e a área precisamos saber qual o comprimento do lado do quadrado, ou seja, temos que saber o valor de  $l$ .
- **Processamento:** cálculo do perímetro e da área. O perímetro ( $P$ ) é a soma dos quatro lados, pode ser representado por  $P = 4 \times l$ . A área ( $A$ ) é dada por  $A = l \times l$ .
- **Saída:** informar o valor do perímetro ( $P$ ) e o valor da área ( $A$ ).

Essa estruturação facilita o entendimento sobre o problema, tornando explícitas as informações que precisamos, o que devemos fazer e que resultado devemos fornecer. O próximo passo consiste em encontrar as variáveis do problema. Quantas variáveis precisamos? Qual o tipo de cada variável?

Na entrada de dados precisamos saber qual o valor do lado. Esta informação será fornecida pelo usuário e precisaremos armazená-la para efetuar os cálculos do processamento. Portanto, o valor do lado necessita ser armazenado em uma variável, a qual denominaremos de  $l$ . No processamento temos que armazenar o valor da área e o valor do perímetro, logo encontramos mais duas variáveis,  $A$  e  $P$ .

Definimos as variáveis, agora temos que especificar qual o tipo de cada uma delas. O comprimento do lado, a área e o perímetro são valores numéricos, podemos representar como inteiro ou real. Qual dos dois tipos devemos utilizar? Podemos ter parte decimal na medida do lado de um quadrado?

O tipo a ser utilizado é o real, já que a medida do lado, o perímetro e a área não necessariamente são números inteiros.

Agora que já entendemos o nosso problema, podemos iniciar a elaboração do algoritmo. Todo algoritmo precisa de um nome e este nome segue as mesmas regras de nomeação de identificadores, ou seja, não pode começar com números, ter espaços ou conter caracteres especiais. Além disso, o nome do algoritmo não pode ser o mesmo das variáveis. Que tal chamarmos de quadrado? Com isso nossa primeira linha é:

**Algoritmo** quadrado

A declaração de variáveis deve ser realizada no início do algoritmo, logo após a definição de seu nome, portanto temos:

**Var**

L, A, P: **real**

Sabemos que o algoritmo é uma sequência finita de passos, delimitada por Início e Fim, com as instruções de entrada, processamento e saída dentro desses limites. Portanto:

**Início**

Na entrada de dados precisamos obter o valor do lado (L), portanto utilizamos o comando **leia**. É importante fornecer ao usuário uma mensagem informando qual ação é esperada antes de utilizar o comando **leia**. Com isso, temos:

**Escreva** ("Informe o valor do lado do quadrado:")

**Leia** (L)

No processamento temos que calcular a área e o perímetro, os quais são representados pelas variáveis, A e P, respectivamente. O cálculo é representado por uma expressão matemática, que ao passar para algoritmo temos:

$A = L * L$

$P = 4 * L$

Estas instruções indicam que a variável A e a variável P armazenam o resultado da expressão que definimos.

Passamos pela entrada e processamento, agora temos que informar o resultado, ou seja, a saída de dados, que é realizada pelo comando **escreva**. O objetivo é informar o valor da área e do perímetro, portanto:

**Escreva** ("O perímetro é:", P)

**Escreva** ("A área é:", A)

Na instrução de saída utilizamos o comando **escreva** e combinamos uma saída literal com o valor da variável. Observe que a literal fica entre aspas, seguida de vírgula, variável. Como um algoritmo é uma sequência finita de passos, temos que finalizá-lo com a seguinte instrução:

**Fim.**

Note que a instrução **Fim** possui um ponto final. O algoritmo completo que formulamos encontra-se no Quadro 15. Note que as palavras-chave estão em negrito. Lembre-se que essas palavras são reservadas, desta forma não podemos utilizá-las para nomear os identificadores (programas, variáveis, constantes e outros).

```

Algoritmo quadrado
Var
    L,A,P: real
Início
    Escreva("Informe o valor do lado d quadrado:")
    Leia(L)
    A←L*L
    P←4*L
    Escreva("O perímetro é:", P)
    Escreva("A área é:", A)
Fim

```

Quadro 15: Pseudocódigo - Problema 2

Vamos simular o funcionamento do nosso algoritmo ou realizar o teste de mesa. Essa etapa é importante, pois auxilia na compreensão e verificação do algoritmo. Supondo que  $L = 3$ , logo temos que:

$$A = 3 * 3$$

$$P = 4 * 3$$

Desta forma, temos  $A = 9$  e  $P = 12$ , com isso a saída é dada por:

O perímetro é: 12

A área é: 9

No Quadro 16 é apresentado o resultado da execução do algoritmo que é mostrado em vídeo.

```

Informe o valor do lado do quadrado: 3
O perímetro é: 12
A área é: 9

```

Quadro 16: Resultado da execução - Problema 2

Neste problema, colocamos em prática o conceito de variáveis, tipos de variáveis, expressões aritméticas, entrada, processamento e saída de dados. E, ao final, testamos o algoritmo.



©shutterstock

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

### PROBLEMA 3

Elaborar um algoritmo que apresente o salário bruto, salário líquido, INSS e FGTS de um funcionário, sabendo que o salário bruto é dado por  $(\text{Horas Trabalhadas} + \text{Horas Extras} \cdot (1 + \text{Porcentagem Hora Extra})) \cdot \text{Valor por Hora}$ . A Porcentagem de Hora Extra é 100%. O valor do INSS é dado por  $\text{Salário Bruto} \cdot 9\%$  e o FGTS é  $\text{Salário Bruto} \cdot 8\%$ . Por fim, o salário líquido é dado pelo salário bruto – INSS.

Estruturando o problema em objetivo, entrada, processamento e saída, temos que:

- **Objetivo do algoritmo:** apresentar o valor do salário bruto, salário líquido, INSS e FGTS.
- **Entrada:** para calcular o salário bruto do funcionário precisamos obter o número de horas, número de horas extras e o valor da hora trabalhada.
- **Processamento:** calcular o salário bruto, INSS, FGTS e salário líquido.
- **Saída:** informar o salário bruto, salário líquido, INSS e FGTS.

A partir das informações da entrada de dados e do processamento é possível determinar as variáveis do problema, sendo elas: htrabalhadas, hextra, vhora, sbruto, inss, fgts e sliquido. Essas variáveis são numéricas e do tipo real.

O pseudocódigo para o problema descrito é apresentado no Quadro 17.



### **Algoritmo salario**

#### **Var**

Htrabalhadas, hextra, vhora, sbruto,  
inss, fgts, sliquido: real

#### **Const**

Phoraextra = 10

#### **Início**

**Escreva**("informe o número de horas tra-  
balhadas:")

**Leia**(htrabalhadas)

**Escreva**("Informe o número de horas  
extras:")

**Leia**(hextra)

**Escreva**("informe o valor da hora")

**Leia**(vhora)

sbruto← (htrabalhadas+hextra\*(1+(phoraex-  
tra/100)))\*vhora

inss← sbruto\*(9/100)

FGTS← sbruto\*(8/100)

sliquido← sbruto-inss

**Escreva**("o salário bruto é:", sbruto)

**Escreva**("O salário líquido:", sliquido)

**Escreva**("O INSS é:", inss)

**Escreva**("O FGTS é:", FGTS)

#### **Fim**

Quadro 17: Pseudocódigo – Problema 3

A constante phoraextra foi definida pelo fato do valor ser invariável, neste caso, sendo 100%. Após a entrada de dados foi efetuado o processamento, composto por quatro expressões, em que podemos observar o uso de parênteses. Esses parênteses servem para indicar a precedência das operações, ou seja, primeiramente é realizada a operação que está dentro deles e, em seguida, as demais, observando também a precedência.

Vamos tomar como exemplo os seguintes valores para a entrada de dados: Horas trabalhadas = 20, horas extras = 5 e valor da hora = 10. Substituindo esses valores na expressão para cálculo do salário bruto teríamos que:

$$\text{sbruto} \leftarrow (20 + 5 * (1 + (100/100))) * 10$$

Para facilitar o entendimento acompanhe a resolução deste cálculo fazendo-o no papel. A prioridade de execução é o conjunto de parênteses interno, ou seja, a divisão resultando em 1. Após a realização da divisão será executada a operação de soma cujo resultado é 2. Continuando a ordem de precedência temos que será realizada a multiplicação entre 5 e 2, resultando em 10, o qual será somado com o valor 20, tendo como resultado 30. Por fim, será realizada a multiplicação de 30 e 10, resultando em 300.

Ao final da execução do algoritmo obteremos os resultados, conforme apresentado no Quadro 18.

```
Informe o número de horas trabalhadas: 20
Informe o número de horas extras: 5
Informe o valor da hora: 10
O salário bruto é: 300
O salário líquido é: 273
O INSS é: 27
O FGTS é: 24
```

Quadro 18 - Resultado da execução - Problema 3

Neste problema, além dos conceitos de variáveis, tipos de variáveis, comando de atribuição, entrada e saída de dados, utilizamos constantes e a precedência dos operadores.

## CONSIDERAÇÕES FINAIS

Nesta unidade foi introduzido o conceito central da disciplina, algoritmo. O algoritmo consiste em um conjunto de instruções para solucionar um problema. Enfatizando, dado um problema, podemos ter vários algoritmos que o resolvem, ou seja, temos o algoritmo como um possível caminho para a solução. Além disso, você pôde perceber que o termo algoritmo está presente no nosso dia a dia nas mais diversificadas situações, sempre que temos uma sequência de passos para realizar uma tarefa.

Entendemos o processo de análise de problemas, que consiste na divisão do problema em três partes: Entrada (dados que precisamos solicitar ao usuário), Processamento (operações que devemos efetuar) e Saída (informações que devemos mostrar ao usuário). Percebemos que esses passos nos auxiliam na construção do algoritmo e raciocínio lógico e facilitam a nossa compreensão sobre o problema.

Dentre os tipos de algoritmos estudamos a descrição narrativa, o fluxograma e o pseudocódigo. Vimos que na descrição narrativa o problema é representado por meio da linguagem natural, o fluxograma apresenta uma notação gráfica específica para representação e o pseudocódigo utiliza regras pré-definidas. Ao longo de nossa disciplina adotaremos a representação em pseudocódigo devido a facilidade de conversão para uma linguagem de programação.

Aprendemos a armazenar os dados obtidos utilizando variáveis, que são espaços na memória principal e que contêm diversos valores a cada instante de tempo. Fizemos a analogia de variáveis com caixas, de modo que essas caixas só podem armazenar pertences do mesmo tipo. Com isto, vimos os tipos de variáveis numéricas (inteiro e real), lógicas (boolean) e literais (caractere).

Em relação às regras para nomear identificadores (nomes das variáveis, programas, rotinas, constantes etc.) aprendemos que: devem iniciar com letras, nunca números; não podem conter espaços; não podem conter caracteres especiais; e, não podem ser utilizadas palavras reservadas. As palavras reservadas são aquelas que possuem uso específico no pseudocódigo, tais como: var, tipo, início, fim, se, então, senão, enquanto, repita, faça, caso, até, procedimento, função, e outros.

Estudamos as expressões e os operadores **aritméticos** (soma, subtração,

divisão, multiplicação, exponenciação, divisão inteira e resto), **relacionais** (igual, diferente, maior, menor que, maior ou igual a, menor ou igual a) e **lógicos** (conjunção, disjunção e negação) que utilizamos em sua construção. No que se refere aos operadores lógicos, a conjunção retorna verdadeiro somente se as duas variáveis são verdadeiras. Na disjunção o valor é verdadeiro se pelo menos uma das variáveis for verdadeira. E a negação inverte o valor da variável.

Aprendemos os comandos que utilizamos para atribuição, entrada de dados e saída de dados. A atribuição é o processo de fornecer valor a uma variável, lembrando que esse valor tem que ser compatível com o tipo da variável. Ou seja, se a variável é do tipo inteira, só podemos armazenar um número inteiro nela.

O comando usado para atribuição é representado por  $\leftarrow$ . A entrada de dados é realizada por meio do comando **leia**. Para mostrar dados ao usuário utilizamos o comando de saída **escreva**.

Por fim, colocamos em prática a construção de alguns algoritmos utilizando pseudocódigo. Na construção desses algoritmos aplicamos os conceitos aprendidos no decorrer desta unidade. É importante que agora você pratique um pouco sozinho fazendo as atividades de autoestudo. Não se esqueça de dividir o problema em Entrada-Saída-Processamento para facilitar a construção do algoritmo.



#### SAIBA MAIS

**História da Programação:** Como tudo começou! SANTOS, R. PH. Disponível em:

<<http://www.techtudo.com.br/platb/desenvolvimento/2011/06/20/historia-da-programacao-como-tudo-comecou/>>.

Para entender melhor como elaborar o teste de mesa ou simulação de um algoritmo.

Veja: <<http://www.brasilacademico.com/ed/testemesa.htm>>.

O aprendizado de algoritmos exige prática. Não se pode aprender copiando ou simplesmente olhando.

## ATIVIDADES



1. Construa um algoritmo utilizando a descrição narrativa para pagar uma conta de luz em um caixa eletrônico.
2. Para os nomes de variáveis abaixo, marque (C) para os corretos e (I) para os incorretos. Para cada nome incorreto explique o que está errado.
 

<input type="checkbox"/> cidade	<input type="checkbox"/> media idade	<input type="checkbox"/> nome2
<input type="checkbox"/> endereço_nr	<input type="checkbox"/> a3	<input type="checkbox"/> 4cidade
<input type="checkbox"/> media_peso%	<input type="checkbox"/> endereço.cep	<input type="checkbox"/> cliente_nome
<input type="checkbox"/> aluno-nota	<input type="checkbox"/> B5	<input type="checkbox"/> 1234P

Elabore um algoritmo que leia, calcule e escreva a média aritmética entre quatro números.

3. Formule um algoritmo para ler um número positivo qualquer e apresentar o quadrado e a raiz quadrada deste número. Dica: utilize as funções apresentadas.
4. Analise os algoritmos abaixo e escreva o resultado de sua execução.

Algoritmo exe5a	Algoritmo exe5a
Var	Var
A,B,C: inteiro	A,B,C: inteiro
Início	Início
A ← 5	A ← 5
B ← 30	B ← 30
C ← A	C ← A
B ← C	A ← B
A ← B	B ← C
Escreva A	Escreva A
Escreva B	Escreva B
Escreva C	Escreva C
Fim	Fim



## EXERCÍCIOS DE FIXAÇÃO

1. Elabore um algoritmo que leia um número inteiro e apresente o antecessor, o número e o sucessor.

**Objetivo do algoritmo:** apresentar o antecessor e o sucessor de um número.

**Entrada:** ler um número inteiro.

**Processamento:** calcular o valor do número menos um (antecessor) e o valor do número mais um (sucessor).

**Saída:** apresentar o antecessor, o número e o sucessor.

```

Algoritmo numero
Var
    num: inteiro

Início
    Escreva ("Informe um número inteiro:")
    Leia (num)
    Escreva ("O antecessor é:", num -1)
    Escreva ("O número é:", num )
    Escreva ("O sucessor é:", num +1)

Fim.
    
```

Quadro 19: Pseudocódigo – Exercício 1

2. Escreva um algoritmo que calcule a área de um triângulo.

**Objetivo do algoritmo:** calcular a área de triângulo.

**Entrada:** obter o valor da base e da altura do triângulo.

**Processamento:** calcular o valor da área que é dado por:  $A = \frac{\text{base} \times \text{altura}}{2}$ .

**Saída:** imprimir o valor da área.

## ATIVIDADES



```
Algoritmo triangulo
Var
    base, altura, area: real
Início
    Escreva ("Informe a base do triângulo:")
    Leia (base)
    Escreva ("Informe a altura do triângulo:")
    Leia (altura)
    area ← (base * altura)/2
    Escreva ("A área do triângulo é:", área)
Fim.
```

Quadro 20: Pseudocódigo – Exercício 2

3. Construa um algoritmo que leia o preço de um produto, o percentual de desconto e calcule o valor a pagar e o valor do desconto.

**Objetivo do algoritmo:** calcular o valor a pagar de um produto.

**Entrada:** ler o preço de um produto e o percentual de desconto.

**Processamento:** calcular o valor do desconto e subtrair do preço do produto.

**Saída:** imprimir o valor a pagar do produto e o valor do desconto.

## ATIVIDADES



```
Algoritmo produto
Var
    Preco, pdesconto, vdesconto, vpagar; real
Inicio
    Escreva ("informe o preço do produto:")
    Leia (preco)
    Escreva ("Informe o percentual do desconto:")
    Leia (pdesconto)
    Vdesconto ← (preco*pdesconto)/100
    Vpagar ← preco - vdesconto
    Escreva ("O valor a pagar é:", vpagar)
    Escreva ("O valor do desconto é:", vdesconto)
Fim
```

Quadro 21: Pseudocódigo – Exercício 3

4. Elabore um algoritmo que leia a quantidade de livros que uma locadora de livros possui e o valor do aluguel por livro. Apresente as seguintes informações: a) faturamento mensal se todos os livros forem locados; b) faturamento anual se 20% dos livros não forem locados todo mês.

**Objetivo do algoritmo:** calcular o faturamento mensal e o anual de uma locadora de livros.

**Entrada:** ler a quantidade de livros e o valor do aluguel.

**Processamento:** calcular o faturamento mensal considerando que todos os livros foram locados e o faturamento anual considerando que 20% dos livros não serão locados.

**Saída:** imprimir o valor do faturamento mensal e o valor do faturamento anual.



## ATIVIDADES



```

Algoritmo faturamento
Var
    valor, fmensal, faual: real
    qidade: interio

Início
    Escreva("Informe a quantidade de livros:")
    Leia(qidade)
    Escreva("Informe o valor da locação:")
    Leia(valor)
    fmensal qidade*valor
    faual ((qidade*0,8)*valor)*12
    Escreva("O faturamento mensal é de:", fmen
    sal)
    Escreva("O faturamento anual considerando
    a locação de 80% dos livros é:", faual)

Fim

```

Quadro 22: Pseudocódigo – Exercício 4

- Escreva um algoritmo que leia o valor da hora aula, o número de aulas dadas no mês e o percentual de desconto do INSS. Calcule e apresente o salário líquido e o salário bruto.

**Objetivo do algoritmo:** calcular o salário líquido e o salário bruto de um professor.

**Entrada:** ler o valor da hora aula, o número de aulas dadas no mês e o percentual de desconto do INSS.

**Processamento:** calcular o salário bruto que é dado pelo produto do número de aulas pelo valor da aula e calcular o salário líquido que é o salário bruto menos o desconto do INSS.

**Saída:** imprimir o valor do salário bruto e o valor do salário líquido.



### Algoritmo salario

**Var**

vhora, naulas, pdesconto, sliquido, sbruto: real

**Início**

**Escreva** ("informe o valor da hora aula:")

**Leia** (vhora)

**Escreva** ("Informe o número de aulas dadas no mês:")

**Leia** (naulas)

**Escreva** ("Informe o percentual de desconto do INSS:")

**Leia** (pdesconto)

sbruto ← vhora\*naulas

sliquido ← sbruto - (sbruto\*pdesconto)/100

**Escreva** ("O valor do salário bruto é:", sbruto)

**Escreva** ("O valor do salário líquido é:", sliquido)

**Fim**

Quadro 23: Pseudocódigo – Exercício 5

6. Escreva um algoritmo que calcule a área e o perímetro de um círculo.

**Objetivo do algoritmo:** calcular a área e o perímetro de um círculo.

**Entrada:** ler o valor do raio.

**Processamento:** calcular a área que é dada por:  $A = \pi r^2$  e perímetro que é dado por  $P = 2\pi r$ .

**Saída:** imprimir o valor da área e do perímetro.

## ATIVIDADES



```
Algoritmo circulo
Var
    area, perimetro raio: real
Início
    Escreva ("Informe o valor do raio:")
    Leia (raio)
    area ← 2 * pi*raio
    Escreva ("A área é:", area)
    Escreva ("O perímetro é:", perimetro)
Fim
```

Quadro 24: Pseudocódigo – Exercício 6

7. Elabore um algoritmo que leia um número inteiro e apresente a raiz quadrada e o valor deste número elevado ao quadrado.

**Objetivo do algoritmo:** calcular a raiz quadrada e o valor do número elevado ao quadrado.

**Entrada:** ler um número inteiro.

**Processamento:** utilizar as funções SQR e SQRT para elevar ao quadrado e obter a raiz quadrada, respectivamente.

**Saída:** imprimir raiz quadrada e o valor do número elevado ao quadrado.

## ATIVIDADES



**Algoritmo funcoes**

**Var**

raiz: real

num, quadrado; inteiro

**Início**

**Escreva** ("Informe um número inteiro:")

Leia (num)

Raiz  $\leftarrow$  sqrt (num)

Quadrado  $\leftarrow$  sqrt (num)

**Escreva** ("A raiz quadrada é;, raiz)

**Escreva** ("O número elevado ao quadrado é:",  
quadrado)

**Fim**

Quadro 25: Pseudocódigo – Exercício 6



## LIVRO

### **Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados**

FORBELLONE, A.; EBERSPACHER, H

**Editora:** Makron Books

**Sinopse:** Este livro introduz o leitor no universo da lógica aplicada à programação de computadores. Ao final do estudo, o aluno estará capacitado a construir algoritmos, assim como a assimilar mais facilmente qualquer linguagem de programação existente ou futura. O texto não requer nenhum conhecimento prévio de informática e é independente de características de máquina. Cada capítulo conta com exercícios de fixação, que visam sedimentar os assuntos de cada subitem, e com exercícios propostos, que cobrem todo o conteúdo do capítulo. No anexo encontram-se resoluções dos exercícios de fixação. A pseudolinguagem utilizada é intencionalmente próxima das linguagens de programação comumente adotadas como primeira linguagem, para facilitar a posterior tradução e implementação prática.



## LIVRO

### **Lógica e Design de Programação**

Joyce Farrell

**Editora:** Cengage Learning

**Sinopse:** Lógica e Design de Programação é um guia de desenvolvimento de lógicas estruturadas para o programador iniciante. Sua leitura não pressupõe nenhuma experiência com linguagens de programação nem conhecimento matemático avançado. A obra contém muitos exemplos provenientes da área de negócios, que ajudam os estudantes a obter um conhecimento sólido em lógica, independentemente da linguagem de programação usada. Suas principais características incluem: - A lógica usada no livro pode ser aplicada em qualquer linguagem de programação. - Fluxogramas, pseudocódigos e muitas ilustrações tornam o aprendizado mais fácil - O ícone Não Faça Isso destaca os erros mais comuns, ajudando os estudantes a evitá-los. - A sessão Zona de Jogos incentiva os leitores a criarem jogos usando os conceitos aprendidos. - Cinco apêndices permitem que os estudantes tenham experiências adicionais de estruturar grandes programas desestruturados, criar formulários de impressão, usar o sistema binário de numeração, trabalhar com grandes tabelas de decisões e testar softwares.





# ESTRUTURA CONDICIONAL

UNIDADE



## Objetivos de Aprendizagem

- Conhecer a estrutura condicional simples.
- Conhecer a estrutura condicional composta.
- Conhecer a estrutura condicional aninhada.
- Conhecer a estrutura de decisão múltipla.
- Elaborar algoritmos utilizando estrutura condicional.

## Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Estrutura condicional
- Estrutura condicional simples
- Estrutura condicional composta
- Estrutura condicional aninha
- Estrutura de decisão múltipla





## INTRODUÇÃO

Nesta unidade você estudará a estrutura condicional conhecendo a estrutura condicional simples, composta, aninhada e de decisão múltipla. Com os conhecimentos adquiridos na Unidade I conseguimos construir algoritmos que a partir da entrada, os dados eram processados e apresentavam algumas informações na saída, isto é, algoritmos sequenciais.

Nos algoritmos sequenciais o fluxo é seguido de modo sequencial, ou seja, todas as instruções eram executadas uma a uma. Não conseguimos impor condições para a execução das instruções. Se pararmos para pensar em diversas situações temos que realizar uma verificação, analisar o resultado desta verificação para saber que caminho seguir, ou seja, temos desvios de fluxo. Isso acontece quando vamos comprar um carro, por exemplo, dependendo do valor da entrada a taxa de juros pode ser maior ou menor. Quando temos que apresentar a média escolar de um aluno e tomar a decisão se ele está aprovado, reprovado ou de exame. Não conseguimos tratar essas duas situações utilizando apenas os conceitos de variáveis, tipos de variáveis, constantes, comando de atribuição, entrada e saída de dados. Precisamos de uma estrutura que nos permita impor condições para a execução de uma determinada instrução, ou ainda, criar condições que possibilitem desviar o fluxo.

Estudaremos a estrutura condicional simples que nos permite tomar uma decisão. A estrutura condicional composta que a partir de uma expressão podemos seguir dois caminhos, um quando o resultado do teste é verdadeiro e outro quando o resultado é falso. A estrutura condicional aninhada ou encadeada que nos permite estabelecer verificação de condições sucessivas. E a estrutura de decisão múltipla que é uma generalização da estrutura condicional composta em que pode haver uma ou mais condições a serem testadas e cada uma delas pode ter uma instrução diferente associada. Além disso, veremos a tabela verdade para cada um dos operadores lógicos e como construir expressões relacionais e lógicas. Os operadores e expressões foram vistos na Unidade I, se você ficou com alguma dúvida não se preocupe, pois ao longo desta unidade revisaremos estes conceitos.

Ao estudar cada estrutura condicional veremos um exemplo e construiremos algoritmos para visualizar a aplicação dos conceitos estudados. Ao final desta unidade poderemos formular algoritmos mais elaborados, com desvios de fluxo o que nos possibilitará desenvolver soluções para uma gama maior de problemas.

## ESTRUTURA CONDICIONAL

Até o momento, os nossos algoritmos apresentavam um padrão em que a partir dos dados de entrada, esses eram processados e na saída mostrávamos algumas informações. O fluxo era seguido sequencialmente, sem nenhum desvio, ou seja, todas as instruções eram executadas. No entanto, em muitas situações necessitamos realizar algum teste antes de efetuar um processamento.

Vamos analisar a retirada de dinheiro em um caixa eletrônico. Após inserir o cartão é solicitado que a senha seja digitada. Se a senha digitada estiver correta poderemos efetuar o saque. Caso a senha esteja errada receberemos mensagem informando que a senha é inválida. Notem que nesta situação não conseguimos representar apenas com o conhecimento adquirido na Unidade I. Em situações como esta precisamos utilizar uma estrutura que nos permita fazer verificações para então saber que instruções devem ser executadas.

A estrutura que permite desviar o fluxo do programa é denominada de estrutura condicional, estrutura de seleção ou estrutura de controle (MANZANO; OLIVEIRA, 1997; ASCENCIO; CAMPOS, 2010).

A estrutura condicional consiste em uma estrutura de controle de fluxo que permite executar um ou mais comandos se a condição testada for verdadeira ou executar um ou mais comandos se for falsa. Essa estrutura divide-se em estrutura simples e estrutura composta, as quais veremos a seguir (LOPES; GARCIA 2002).



©shutterstock

## ESTRUTURA CONDICIONAL SIMPLES

Na estrutura condicional simples o comando só será executado se a condição for verdadeira. A sintaxe do comando é:

```
Se (<Condição>) então
    <instruções para condição verdadeira>
fim_se
```

A estrutura condicional simples tem por finalidade tomar uma decisão. De modo que se a condição que está sendo testada for verdadeira são executadas todas as instruções compreendidas entre o se e o fim\_se (MANZANO; OLIVEIRA, 1997). Ao término da execução o algoritmo segue o primeiro comando após o fim\_se. Se a condição que está sendo testada for falsa o algoritmo executa a primeira instrução após o fim\_se, não executando as instruções compreendidas entre o se e o fim\_se.

Vamos analisar o algoritmo apresentado no Quadro 26. Consideremos o valor de A como 15, desta forma ao testar a condição dada pela expressão  $A > 10$ , retorna um valor verdadeiro. Com isto, temos a execução do comando escreva que está compreendido entre o se e o fim\_se. Agora, tomemos A com valor 3. Ao testar a condição  $A > 10$  o valor retornado é falso. Deste modo, não é executada a instrução entre o se e o fim\_se.

```
Algoritmo exemplo
Var
    A : inteiro
Início
    Leia (A)
    Se (A > 10) então
        Escreva ("A é maior que 10")
    Fim_se
Fim.
```

Quadro 26: Pseudocódigo - Exemplo Estrutura Condicional Simples

De acordo com Lopes e Garcia (2002), a condição é uma expressão lógica, portanto ao ser testada devolve como resposta o valor verdadeiro ou falso. Uma condição pode ser representada por uma expressão relacional ou por uma expressão lógica formada por pelo menos duas expressões relacionais. Os operadores relacionais vistos na Unidade I são  $>$ ,  $<$ ,  $=$ ,  $>=$ ,  $<=$  e  $<>$ . Já os operadores lógicos são E, OU e NÃO.

Agora fica mais clara a aplicação dos operadores relacionais e como eles são utilizados em nossos algoritmos. Alguns exemplos de expressão relacional são:

`X > 16`

`A < B`

`Sexo = "F"`

`Resposta <> "Sim"`

Quando nossa condição é uma expressão lógica temos pelo menos duas expressões relacionais que estão ligadas por um operador lógico. Você se lembra do funcionamento dos operadores lógicos? O operador E resulta em verdadeiro somente quando as duas condições são verdadeiras, como pode ser visto na tabela verdade apresentada no Quadro 27.

OPERADOR E		
Condição 1	Condição 2	Resultado
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Falso
Falso	Verdadeiro	Falso
Falso	Falso	Falso

Quadro 27: Tabela Verdade - Operador E

Fonte: adaptado de (MANZANO; OLIVEIRA, 1997)

O operador OU resulta em verdadeiro quando pelo menos uma das condições é verdadeira. Sendo o resultado falso apenas quando as duas condições são falsas, como pode ser visualizado no Quadro 28.

OPERADOR OU		
Condição 1	Condição 2	Resultado
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Falso	Verdadeiro	Verdadeiro
Falso	Falso	Falso

Quadro 28: Tabela Verdade - Operador OU

Fonte: adaptado de (MANZANO; OLIVEIRA, 1997)

E o operador **NÃO** funciona como a negação do resultado, ou seja, inverte o resultado lógico. Segundo Manzano (2000), esse operador é utilizado em situações em que se necessita estabelecer que uma dada condição deve não ser verdadeira ou deve não ser falsa. A tabela verdade para este operador é apresentada no Quadro 29. Observe que o NÃO é um operador unário, ou seja, tem apenas uma condição.

OPERADOR OU	
Condição	Resultado
Verdadeiro	Falso
Falso	Verdadeiro

Quadro 29: Tabela Verdade - Operador NÃO

Fonte: adaptado de (MANZANO; OLIVEIRA, 1997)

A partir do entendimento da tabela verdade de cada um dos operadores lógicos vamos ver exemplos de expressão lógica:

$(X \geq 1) \text{ E } (X \leq 20)$

$(\text{Sexo} = \text{"F"}) \text{ OU } (\text{Sexo} = \text{"f"})$

$\text{NÃO } (X > 5)$

Note que as expressões lógicas são compostas utilizando operadores relacionais e lógicos.

Agora que conhecemos a sintaxe da estrutura condicional simples e sabemos como montar condições, vamos formular nosso primeiro algoritmo contendo desvio de fluxo.

O problema consiste em identificar se um número inteiro é um número par e então imprimir a metade do número.

Retomando nossos passos para a construção de algoritmos temos que:

- **Objetivo do algoritmo:** verificar se o número é par e imprimir a metade deste número.
- **Entrada:** obter um número.
- **Processamento:** verificar se o número é par.
- **Saída:** imprimir a metade do número, caso ele seja par.

O que é um número par? Um número par é um número inteiro múltiplo de 2, ou seja, um número cuja divisão por 2 resulte em resto igual a 0.

A entrada de dados consiste em obter um número inteiro, o qual denominaremos de N. O processamento consiste em encontrar o resto da divisão deste número por 2 e verificar se é igual a zero. Como faremos isso? Você se recorda de alguma função que faz isso? Na Unidade I vimos o operador MOD, que retorna o resto da divisão de dois números inteiros.

Se o resto for igual a zero calcularemos a metade deste número. E a saída consistirá em imprimir a metade do número. Se o resto for diferente de zero não será executada nenhuma instrução e também não haverá saída.

Uma solução para este problema é apresentada no Quadro 30, em que podemos visualizar o uso da estrutura condicional simples.

```
algoritmo par
Var
    n, resto, metade:inteiro
Início
    Escreva("Digite um número");
    Leia(n)
    resto←n mod2
    Se(resto=0) então
        Escreva("A metade do  número",metade)
    Fim_se
Fim.
```

Quadro 30: Pseudocódigo - Algoritmo Par

Neste algoritmo utilizamos os conceitos que já conhecíamos (variáveis, tipos de variáveis, atribuição, comando de entrada e saída de dados), agregando a estrutura condicional.

Em relação ao algoritmo apresentado, podemos colocar uma instrução escreva após o fim\_se dizendo que o número é ímpar? Não, pois para qualquer número obtido na entrada, indiferente de ser par ou ímpar, a mensagem seria impressa. Devemos lembrar que após o fim\_se o fluxo do algoritmo segue normalmente, sendo executada instrução a instrução.

Por que a variável metade foi declarada como inteira e não como real? A metade de qualquer número par é sempre um número inteiro. Como a operação está sendo executada apenas se o número é par, não há problema. Se a instrução que calcula a metade fosse executada fora da estrutura condicional a variável deveria ser real. Por exemplo, se o número 3 fosse obtido na entrada a metade seria 1.5, que não é um número inteiro.

Precisamos essa variável denominada metade? E a variável resto? Não, podemos realizar o teste lógico a partir da expressão relacional, não sendo necessária a variável metade. Quanto a variável resto, podemos enviar como saída a operação que calcula a metade. Com isto, teríamos um algoritmo que utiliza apenas uma variável e duas operações de atribuição menos, como pode ser visto no Quadro 31.

```

Algoritmo par
Var
    n: inteiro
Início
    Escreva ("Digite um número:")
    Leia (n)
    Se (n mod 2 = 0) então
        Escreva ("A metade do número é:", n/2)
    Fim_se
Fim.
    
```

Quadro 31: Pseudocódigo - Algoritmo Par

## ESTRUTURA CONDICIONAL COMPOSTA

Na estrutura condicional composta é realizada a avaliação de uma única expressão lógica. Se o resultado desta avaliação for verdadeiro é executado a instrução ou o conjunto de instruções compreendido entre o comando **se** e o **senão**. Se o resultado da avaliação for falso é executado a instrução ou o conjunto de instruções entre o **senão** e o **fim\_se** (MANZANO; OLIVEIRA, 1997).

A sintaxe da estrutura condicional composta é:

```
Se (<Condição>) então  
    <instruções para condição verdadeira>  
  
Senão  
    <instruções para condição falsa>  
  
fim_se
```

Agora que você conhece a estrutura condicional composta, podemos construir um algoritmo para verificar se um número inteiro é par ou ímpar. O Quadro 32 apresenta o pseudocódigo para verificar se um número é par ou ímpar.

```
Algoritmo parimpar  
Var  
    n: inteiro  
  
Início  
    Escreva ("Digite um número:")  
    Leia (n)  
  
    Se (n mod 2 = 0) então  
        Escreva ("O número é par")  
    Senão  
        Escreva ("O número é ímpar")  
  
    Fim_se  
Fim.
```

Quadro 32: Pseudocódigo - Algoritmo parimpar

Se o resultado do teste da expressão relacional  $n \bmod 2 = 0$  for verdadeiro é executada a instrução que se encontra entre o **se** e o **fim\_se**, ou seja, escreva “o



número é par”. Caso o resultado do teste seja falso é executada a instrução que se encontra em o senão e o fim\_se, escreve “O número é ímpar”. Por exemplo, se o número obtido na entrada for 5 temos que  $5 \bmod 2$  é igual a 1, ou seja, o teste da expressão resulta em falso, logo será impresso que “O número é ímpar”.

Na estrutura condicional composta podem ser utilizadas expressões relacionais e expressões lógicas, tal como na estrutura condicional simples.

Antes de conhecer outros tipos de estrutura condicional vamos praticar mais a construção de algoritmos utilizando expressões lógicas. O problema consiste em: dado um número inteiro verificar se ele está compreendido entre 20 e 90.

No processo de resolução seguiremos os seguintes passos:

- **Objetivo do algoritmo:** verificar se o número está compreendido entre 20 e 90.
- **Entrada:** obter um número inteiro.
- **Processamento:** verificar se o número está entre 20 e 90.
- **Saída:** imprimir se o número está dentro ou fora da faixa de 20 a 90.

Para identificar o número de variáveis devemos observar o que é requerido na entrada e no processamento, neste caso temos que a entrada é um número inteiro (denominadores variável  $n$  do tipo inteiro) e o processamento pode ser realizado usando apenas expressões lógicas, portanto não precisamos de mais variáveis.

A verificação se o número está na faixa entre 20 e 90, pode ser descrita como:  $n$  tem que ser maior que 20 e menor do que 90. Com isto, temos duas expressões relacionais:  $n > 20$  e  $n < 90$ .

A saída de dados consiste em imprimir se o número está ou não dentro dos limites dessa faixa. O Quadro 33 apresenta o algoritmo para o problema descrito.

```

Algoritmo faixa
Var
    n: inteiro
Início
    Escreva ("Digite um número inteiro:")
    Leia (n)
    Se (n > 20) e (n < 90) então
        Escreva ("O número está na faixa entre 20 e 90")
    Senão
        Escreva ("O número está fora da faixa")
    Fim_se
Fim.

```

Quadro 33: Pseudocódigo - Algoritmo faixa

Considerando n com valor 27, temos uma expressão lógica composta por duas expressões relacionais (condições) unidas pelo operador E. Você se lembra como esse operador funciona? Dadas duas condições o resultado é verdadeiro se e somente se as duas forem verdadeiras. O Quadro 34 representa a expressão lógica da estrutura condicional, em que  $27 > 20$  é verdadeiro e  $27 < 90$  também é verdadeiro, o que resulta em verdadeiro. Como o resultado do teste lógico é verdadeiro temos a execução do fluxo compreendido entre o se e o senão, ou seja, a mensagem "O número está na faixa entre 20 e 90".

OPERADOR E		
Condição 1	Condição 2	Resultado
$27 > 20$	$27 < 90$	Verdadeiro

Quadro 34: Representação da Expressão Lógica

E se na entrada recebêssemos o número 20? Vamos analisar cada uma das expressões relacionais (Quadro 35). Temos  $20 > 20$ , é uma expressão que resulta em verdadeiro ou falso? É falso, pois 20 é igual a 20 e não maior. Na segunda expressão temos  $20 < 90$ , que resulta em verdadeiro. Essas expressões estão unidas

pelo operador E sendo uma delas falsa e a outra verdadeira, o resultado do teste lógico é falso. Com isto, temos a execução da instrução compreendida entre o senão e o fim\_se, “O número está fora da faixa”.

OPERADOR E		
Condição 1	Condição 2	Resultado
$20 > 20$	$20 < 90$	Falso

Quadro 35: Representação da Expressão Lógica

## ESTRUTURA CONDICIONAL ANINHADA

Agora que você já conhece a estrutura condicional simples e a composta, vamos conhecer a estrutura condicional aninhada ou encadeada. Essa estrutura é utilizada quando precisamos estabelecer a verificação de condições sucessivas, em que uma determinada ação poderá ser executada se um conjunto anterior de instruções ou condições for satisfeito. A execução da ação pode, também, estabelecer novas condições. E o que isso quer dizer? Que podemos utilizar uma condição dentro de outra, ou seja, a estrutura pode possuir diversos níveis de condição (MANZANO; OLIVEIRA, 1997).

LOPES; GARCIA (2002) destacam que essa estrutura é utilizada quando sentimos a necessidade de tomar decisões dentro de uma das alternativas de uma condição.

Vamos visualizar a estrutura condicional aninhada em um problema que consiste em encontrar o maior número dentre três números. Seguindo o método de estruturação do problema visto na Unidade I, temos que:

**Objetivo do algoritmo:** encontrar o maior número.

**Entrada:** obter três números inteiros.

**Processamento:** comparar os números e armazenar o valor do maior.

**Saída:** imprimir o maior número.

A entrada de dados consiste em ler três números inteiros, os quais

armazenaremos em variáveis denominadas A, B e C. Para encontrar qual o maior número precisamos realizar comparações utilizando expressões relacionais do tipo:  $A > B$  e armazenar o valor do maior número em uma variável, a qual chamaremos de max. A saída consiste em enviar uma mensagem contendo o valor do maior número, que está armazenado na variável max. O algoritmo para o problema é apresentado no Quadro 36.

```
Algoritmo maior
Var
    a, b, c, max: inteiro
Início
    Escreva ("Digite o primeiro número inteiro:")
    Leia (a)
    Escreva ("Digite o segundo número inteiro:")
    Leia (b)
    Escreva ("Digite o terceiro número inteiro:")
    Leia (c)
    Se (a > b) então
        Se (a > c) então
            max ← b
        Senão
            max ← c
        Fim_se
    senão
        Se (b > c) então
            max ← b
        senão
            max ← c
        Fim_se
    Fim_se
    Escreva ("O maior número é:", max)
Fim.
```

Quadro 36: Pseudocódigo - Algoritmo maior

Se você não entendeu o funcionamento do algoritmo, fique tranquilo veremos passo a passo a estrutura condicional do problema em questão. Na Figura 5 temos a representação da estrutura condicional aninhada, os colchetes em azul representam a estrutura condicional composta, em que o trecho 1 é executado quando o resultado da expressão relacional  $A > B$  é verdadeiro. O trecho 2 apresenta o conjunto de instruções que é executado quando o resultado da expressão relacional é falso. Note que tanto no trecho 1 quanto no trecho 2, temos outra estrutura condicional, representada

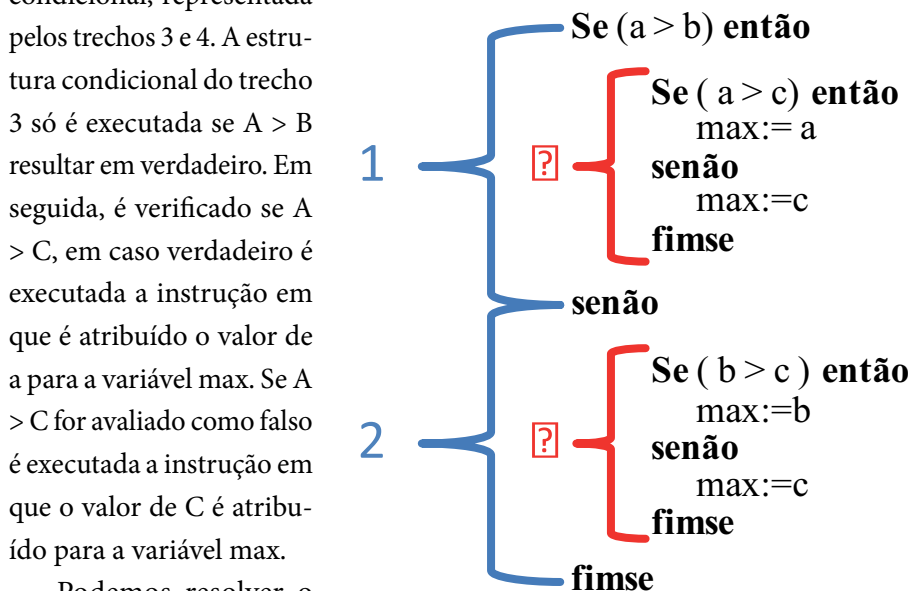


Figura 5: Estrutura condicional aninhada

Podemos resolver o problema de encontrar o maior número dentre três números sem utilizar a estrutura condicional aninhada? Sim, podemos, porém será executado um maior número de verificações, como pode ser visto no algoritmo do Quadro 37.

```
Algoritmo maior
Var
    a, b, c, max: inteiro
Início
    Escreva ("Digite o primeiro número inteiro:")
    Leia (a)
    Escreva ("Digite o segundo número inteiro:")
    Leia (b)
    Escreva ("Digite o terceiro número inteiro:")
    Leia (c)
    Se (a > c) então
        Se (a > b) então
            max ← a
        Senão
            max ← b
        Fim_se
    senão
        se (c > max) então
            max ← c
        Fim_se
    Fim_se
    Escreva ("O maior número é:", max)
Fim.
```

Quadro 37: Pseudocódigo - Algoritmo maior

Teste o funcionamento dos dois algoritmos supondo  $a=5$ ,  $b=3$  e  $c=12$ . Qual a vantagem de utilizar a estrutura condicional aninhada? A grande vantagem é que o uso destes encadeados melhora o desempenho do algoritmo, isto é, torna o algoritmo mais rápido por realizar menos testes e comparações. Ou ainda, executar um menor número de passos para chegar à solução do problema.

## ESTRUTURA DE DECISÃO MÚLTIPLA

A estrutura de decisão múltipla também denominada de estrutura de decisão do tipo escolha consiste em uma generalização do Se, em que somente uma condição era avaliada e dois caminhos poderiam ser seguidos, um para o resultado da avaliação ser verdadeiro e outro para falso. Na estrutura de decisão múltipla pode haver uma ou mais condições a serem avaliadas e um comando diferente associado a cada uma delas (LOPES; GARCIA, 2002).

A sintaxe dessa estrutura é (MANZANO; OLIVEIRA, 1997):

```

caso <variável>

    seja <valor 1> faça <instrução 1>
    seja <valor 2> faça < instrução 2>
    seja <valor N> faça < instrução N>
    senão < instrução >

Fim_caso
    
```

Nesta estrutura o termo <variável> indica a variável a ser controlada, o termo <valor> o conteúdo da variável que está sendo analisado e o termo <instrução> refere-se a instrução que será executada. O funcionamento desta estrutura consiste em ao entrar-se em uma construção do tipo **Caso**, o caso1 é testada: se for verdadeira, a instrução 1 é executada e após seu término, o fluxo de execução prossegue pela primeira instrução após o final da construção (**fim\_caso**); se o caso1 for falsa, o caso2 é testado: se esta for verdadeira, a instrução 2 é executada e ao seu término, a execução prossegue normalmente pela instrução seguinte ao final da construção (**fim\_caso**). De modo análogo ocorre para os demais casos da estrutura. Lopes e Garcia (2002) destacam que esse tipo de estrutura é bastante utilizado na construção de menus, tornando-os mais claros.

Vamos elaborar um algoritmo para dado um número inteiro escrever o mês correspondente. Se for digitado um número fora desse intervalo, deverá aparecer uma mensagem informando que não existe mês com este número. Retomando o nosso modelo de estruturação de problemas, temos que:

- **Objetivo do algoritmo:** retornar o mês equivalente ao número digitado.
- **Entrada:** obter um número inteiro.

- **Processamento:** verificar se o número digitado está entre 1 e 12.
- **Saída:** imprimir o mês equivalente por extenso.

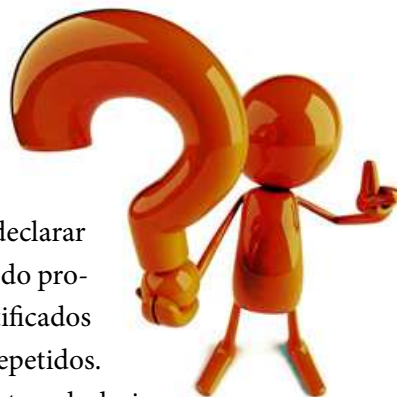
Na entrada de dados temos um número inteiro, que nomearemos com variável `num`. O processamento consiste em verificar se o número digitado está entre 1 e 12 e como saída escrever por extenso o mês correspondente. No Quadro 38 temos o pseudocódigo para o problema em questão.

```
Algoritmo Mês
Var
    num: inteiro
Início
    Escreva ("Digite um número de 1 a 12:")
    Leia (num)
Caso (num)
    Seja 1 faça Escreva ("Janeiro")
    Seja 2 faça Escreva ("Fevereiro")
    Seja 3 faça Escreva ("Março")
    Seja 4 faça Escreva ("Abril")
    Seja 5 faça Escreva ("Maio")
    Seja 6 faça Escreva ("Junho")
    Seja 7 faça Escreva ("Julho")
    Seja 8 faça Escreva ("Agosto")
    Seja 9 faça Escreva ("Setembro")
    Seja 10 faça Escreva ("Outubro")
    Seja 11 faça Escreva ("Novembro")
    Seja 12 faça Escreva ("Dezembro")
    Senão Escreva ("O número digitado não corres-
ponde a nenhum mês")
Fim_caso
Fim.
```

Quadro 38: Pseudocódigo - Algoritmo mês



Retomando um pouco o conhecimento sobre variáveis, no algoritmo acima, podemos alterar o nome da variável num para mes? Não, pois o nome do algoritmo se chama mes. Se quisermos declarar a variável como mes temos que modificar o nome do programa. Lembre-se não podemos ter nome de identificadores (variável, constante, programa, rotinas e outros) repetidos.



©photos

Ainda analisando o algoritmo que utiliza a estrutura de decisão múltipla, você acha que podemos escrevê-lo de outra forma?

Como ficaria o algoritmo se utilizássemos apenas a estrutura condicional simples?

Há pelo menos mais duas formas de resolvê-lo utilizando a estrutura condicional simples ou a estrutura encadeada. A construção do algoritmo utilizando apenas a estrutura condicional simples pode ser visualizada no Quadro 39.

```

Algoritmo mes2
Var
    num: inteiro
Início
    Escreva ("Digite um número de 1 a 12:")
    Leia (num)
    Se (num = 1) então
        Escreva ("Janeiro")
    Fim_se
    Se (num = 2) então
        Escreva ("Fevereiro")
    fim_se
    Se (num = 3) então
        Escreva ("Março")
    Fim_se
    Se (num = 4) então
        Escreva ("Abril")
    Fim_se

```

```

Se (num = 5) então
    Escreva ("Maio")
Fim_se
Se (num = 6) então
    Escreva ("Junho")
Fim_se
Se (num = 7) então
    Escreva ("Julho")
Fim_se
Se (num = 8) então
    Escreva ("Agosto")
Fim_se
Se (num = 9) então
    Escreva ("Setembro")
Fim_se
Se (num = 10) então
    Escreva ("Outubro")
Fim_se
Se (num = 11) então
    Escreva ("Novembro")
Fim_se
Se (num = 12) então
    Escreva ("Dezembro")
Fim_se

Se (num < > 1 ) e (num < > 2) e (num < >3) e (num <
>4) e (num < >5) e (num< >6) e (num< >7) e (num< >8)
e (num < > 9) e (num < > 10) e (num< >11) e (num< >12)
então

    Escreva ("O número digitado não corresponde a
nenhum mês")

fim_se
Fim.

```

Quadro 39: Pseudocódigo - Algoritmo mês

Os algoritmos apresentados no Quadro 38 e Quadro 39 são solução para o problema. Você sabe qual a diferença entre eles? Supondo que o número digitado tenha sido 3. No primeiro algoritmo acontecesse a verificação Caso 1, Caso 2 e Caso 3. Como Caso 3 é verdadeiro ocorre a execução da instrução Escreva “Março” e o fluxo de execução segue para o fim\_caso, isto é, os demais casos não são testados. No segundo algoritmo temos que todas as instruções serão verificadas. Desta forma, a diferença entre os dois algoritmos está no desempenho, no número de instruções que será executada e isso impacta no tempo de execução dos nossos programas. Portanto, é importante que você escreva algoritmos eficientes. Você já se perguntou como fazer um algoritmo eficiente? Um algoritmo eficiente utiliza apenas o número necessário de variáveis já que cada variável consiste um espaço de memória reservado. Além disso, podemos associar a eficiência com o número de instruções que o algoritmo executa para resolver um problema. Com isto, temos que não devemos declarar variáveis que não serão utilizadas e dar preferência à estrutura condicional aninha ou de escolha múltipla, ao invés de construir várias estruturas simples.

## CONSIDERAÇÕES FINAIS

Nesta unidade você aprendeu a construir algoritmos com desvios de fluxo, isto é, algoritmos em que podemos impor condições à execução de uma determinada instrução a um teste. Essa estrutura que nos possibilita desviar o fluxo do programa é conhecida na literatura como estrutura condicional, estrutura de seleção ou estrutura de controle.

Estudamos quatro formas de estrutura condicional: estrutura condicional simples, estrutura condicional composta, estrutura condicional encadeada e estrutura de decisão múltipla. Na estrutura condicional simples vimos que as instruções só serão executadas se a condição que está sendo avaliada for verdadeira. A sintaxe dessa estrutura sintaxe é dada por:

```
Se (<Condição>) então
    <instruções para condição verdadeira>
fim_se
```

Na estrutura condicional composta vimos que é realizada a avaliação de uma única condição. No entanto, temos dois caminhos para seguir, um quando o resultado da instrução é verdadeiro e outro quando é falso. A sintaxe da estrutura condicional composta é:

```
Se (<Condição>) então
    <instruções para condição verdadeira>

Senão
    <instruções para condição falsa>

fim_se
```

Aprendemos que a estrutura aninhada é usada quando temos que estabelecer verificações sucessivas, isto é, quando uma ação só poderá ser executada se um conjunto anterior de condições for satisfeito. Além disso, estudamos que o uso desta estrutura torna o algoritmo mais rápido devido ao fato de executar um menor número de passos para chegar à solução do problema.

Vimos também a estrutura de decisão múltipla, uma generalização da estrutura Se, em que pode haver uma ou mais condições a serem testadas e um comando associado a cada uma delas. A sintaxe dessa estrutura é:

```
Caso <variável>
    seja <valor 1> faça <instrução 1>
    seja <valor 2> faça <instrução 2>
    seja <valor N> faça <instrução N>
    senão <instrução>

Fim_caso
```

Além da estrutura condicional, trabalhamos a construção de expressões relacionais e lógicas, as quais são utilizadas na construção de condições. Entendemos, também, a tabela verdade dos operadores lógicos E, OU e NÃO.

Ao longo desta unidade construímos algoritmos utilizando todos os conceitos aprendidos e, também, discutimos que estrutura condicional é mais adequada para cada situação. Como o aprendizado de algoritmos requer prática, é importante que você faça as atividades de autoestudo para exercitar o raciocínio lógico.

## SAIBA MAIS



Para facilitar a leitura de um algoritmo, normalmente são adicionados espaços em branco no início de um bloco de comandos para mostrar quais instruções estão dentro dele, a este recurso denominamos indentação. Para saber um pouco mais, leia o conteúdo disponível em:

<<http://pt.wikipedia.org/wiki/Indent%C3%A7%C3%A3o>>.

## REFLITA



Na solução de um problema é fundamental estudar as várias opções de algoritmos a serem utilizados, pois os aspectos de tempo e espaço são considerações importantes que devem ser vistas com atenção (ZIVIANE, 2004).

## ATIVIDADES



1. Formule um algoritmo que leia a matrícula e nome de um vendedor, seu salário fixo e o total de vendas e calcule a comissão do vendedor. Se o total de vendas é inferior a R\$ 1500,00 o percentual de comissão é 2% e se for maior o percentual é de 4%. Apresente o nome do vendedor, matrícula, salário fixo e salário total.
2. Escreva um algoritmo que leia um número e informe se ele é divisível por 3 e por 7.
3. Formule um algoritmo que leia cinco números e conte quantos deles são negativos.
4. De acordo com uma tabela médica, o peso ideal está relacionado com a altura e o sexo. Elabore um algoritmo que receba altura e sexo de uma pessoa e calcule e imprima o seu peso ideal, sabendo que:

Para homens	$(72.7 \times \text{altura}) - 58$
Para mulheres	$(62.1 \times \text{altura}) - 44.7$

5. Elabore um algoritmo que leia o percurso em quilômetros, o tipo de moto e informe o consumo estimado de combustível, sabendo que uma moto do tipo A faz 26 km com um litro de gasolina, uma moto do tipo B faz 20 km e o tipo C faz 7 km.
6. Uma instituição financeira concederá crédito a uma taxa de juros de 3% aos seus clientes de acordo com o saldo médio do período. Elabore um algoritmo que calcule o valor que pode ser concedido ao cliente e imprima-o. Os clientes com saldo médio inferior a R\$ 500,00 não têm direito a crédito. Já os clientes com saldo entre R\$ 501,00 e R\$ 1000,00 podem obter créditos de 35% em relação ao saldo médio. Clientes com saldo entre R\$ 1001,00 a R\$ 3000,00 podem obter créditos de 50% em relação ao saldo médio. E para aqueles clientes com saldo superior a R\$ 3001,00 pode ser concedido crédito de 75% do valor do saldo.

## EXERCÍCIOS DE FIXAÇÃO

1. Construa um algoritmo que receba o nome e a idade de uma pessoa e informe se é menor de idade, maior de idade ou idoso.

**Objetivo do algoritmo:** informar se a pessoa é menor de idade, maior ou idoso.

**Entrada:** ler nome e idade.

**Processamento:** verificar se a idade é menor que 18 (menor de idade), entre 18 e 64 anos (maior de idade) ou maior ou igual a 65 anos (idoso).

**Saída:** escrever se a pessoa é menor de idade, maior ou idoso.

## ATIVIDADES



```

Algoritmo verificaidade
Var
    idade: inteiro
    nome: caractere[30]

Início
    Escreva ("Digite o nome:")
    Leia (nome)
    Escreva ("Digite a idade:")
    Leia (idade)

    Se (idade > = 65) então
        Escreva ("Idoso")

    Senão
        Se (idade < 18) então
            Escreva ("Menor de idade")
        Senão
            Escreva ("Maior de idade")

    Fim_se

Fim_se

Fim.

```

Quadro 40: Pseudocódigo – Exercício 1

2. Elabore um algoritmo que receba a idade de uma pessoa e identifique sua classe eleitoral: não eleitor (menor que 16 anos de idade), eleitor obrigatório (entre 18 e 65 anos) e eleitor facultativo (entre 16 e 18 anos e maior que 65 anos).

**Objetivo do algoritmo:** verificar a classe eleitoral de uma pessoa.

**Entrada:** ler idade.

**Processamento:** verificar se a idade é menor que 16 (não eleitor), entre 18 e 65 anos (eleitor obrigatório) ou entre 16 e 18 ou maior que 65 anos (eleitor facultativo).

**Saída:** escrever a classe eleitoral.



```

Algoritmo classeeleitoral
Var
    idade: inteiro
Início
    Escreva ("Digite a idade:")
    Leia (idade)
    Se (idade < 16) então
        Escreva ("Não eleitor")
    Senão
        Se (idade > 65) ou (idade < 18) então
            Escreva ("Eleitor facultativo")
        Senão
            Escreva ("Eleitor obrigatório")
        Fim_se
    Fim_se
Fim.
    
```

Quadro 41: Pseudocódigo – Exercício 2

- Escreva um algoritmo que calcule o IMC de uma pessoa e identifique se a pessoa está abaixo do peso (IMC menor que 20), normal (IMC entre 20 e 25), com excesso de peso (IMC entre 26 e 30), obesa (IMC entre 31 e 35) ou com obesidade mórbida (acima de 35). O cálculo do IMC é dado por:  $\frac{\text{peso}}{\text{altura}^2}$ .

**Objetivo do algoritmo:** verificar a faixa de risco de uma pessoa a partir do IMC.

**Entrada:** ler peso e altura.

**Processamento:** verificar se o IMC é menor que 20 (abaixo do peso), entre 20 e 25 (normal), entre 26 e 30 (excesso de peso), entre 31 e 35 (obesidade) ou acima de 35 (obesidade mórbida).

**Saída:** imprimir a faixa de risco da pessoa.



## ATIVIDADES



```

Algoritmo calculaimc
Var
    peso, altura, imc: real
Início
    Escreva ("Digite o peso:")
    Leia (peso)
    Escreva ("Digite a altura:")
    Leia (altura)
    imc peso/sqn(altura)
    Se (imc < 20) então
        Escreva ("Abaixo do peso")
    Senão
        Se (imc <= 25) então
            Escreva ("Normal")
        Senão
            Se (imc <= 30) então
                Escreva ("Excesso de peso")
            Senão
                se (imc <=35) então
                    Escreva ("Obesidade")
                Senão
                    Escreva ("obesidade mórbida")
            Fim_se
        Fim_se
    Fim_se
Fim.

```

Quadro 42: Pseudocódigo – Exercício 3

4. Elabore um algoritmo que receba o salário de um funcionário e o código do cargo e apresente o cargo, o valor do aumento e o novo salário. A Tabela abaixo apresenta os cargos.

## ATIVIDADES



CÓDIGO	CARGO	PERCENTUAL DO AUMENTO
1	Serviços gerais	50%
2	Vigia	30%
3	Recepcionista	25%
4	Vendedor	15%

**Objetivo do algoritmo:** calcular o aumento de salário de acordo com o cargo.

**Entrada:** ler salário e código do cargo.

**Processamento:** calcular o aumento salarial de acordo com o cargo da pessoa.

**Saída:** imprimir o código do cargo, nome do cargo, valor do aumento e novo salário.

**Algoritmo** reajuste

**Var**

cargo: inteiro

salario, reajuste, aumento: real

**Início**

**Escreva** ("Digite a cargo:")

**Leia** (cargo)

**Escreva** ("Digite a salario:")

**Leia** (salario)

**Se** (cargo = 1) **então**

**Escreva** ("Código do cargo:", cargo)

**Escreva** ("Serviços gerais")

aumento  $\leftarrow$  salario\*(50/100)

reajuste  $\leftarrow$  salario+aumento

**Escreva** ("O aumento é de:", aumento)

**Escreva** ("O novo salário é:", reajuste)

**Senão**

**Se** (cargo = 2) **então**

**Escreva** ("Digite a cargo:", cargo)

## ATIVIDADES



```

aumento ← salario*(30/100)
reajuste ← salario+aumento
Escreva ("O aumento é de:", aumento)
Escreva ("O novo salário é:", reajuste)

Senão
    Se (cargo = 3) então
        Escreva ("Digite a cargo:", cargo)
        Escreva ("Vigia")
    aumento ← salario*(25/100)
    reajuste ← salario+aumento
    Escreva ("O aumento é de:", aumento)
    Escreva ("O novo salário é:", reajuste)
Senão
    Se (cargo = 3) então
        Escreva ("Digite a cargo:", cargo)
        Escreva ("Vigia")
        aumento ← salario*(15/100)
        reajuste ← salario+aumento
        Escreva ("O aumento é de:",
aumento)
        Escreva ("O novo salário é:",
reajuste)

    Fim_se
Fim_se
Fim_se
Fim.

```

Quadro 43: Pseudocódigo – Exercício 4

## ATIVIDADES



5. Escreva um algoritmo para resolver equações do segundo grau ( $ax^2 + bx + c$ ). Sendo que: a variável a deve ser diferente de zero;  $\Delta = b^2 - 4 \times a \times c$ ; Se  $\Delta < 0$  não existe raiz real; se  $\Delta = 0$  existe uma raiz real que é dada por  $x = \frac{-b}{2a}$ ; se  $\Delta > 0$  existem duas raízes reais
6.  $x_1 = \frac{-b + \sqrt{\Delta}}{2a}$  e  $x_2 = \frac{-b - \sqrt{\Delta}}{2a}$

**Objetivo do algoritmo:** resolver a equação do segundo grau.

**Entrada:** ler os valores de a, b e c.

**Processamento:** calcular o delta e o valor das raízes.

**Saída:** imprimir o valor da raiz real.

```
Algoritmo raiz
Var
    a, b, c, delta, x1, x2: inteiro
Início
    Escreva ("Digite o valor de a:")
    Leia (a)
    Escreva ("Digite o valor de b:")
    Leia (b)
    Escreva ("Digite o valor de c:")
    Leia (c)
    Se (a = 0) então
        Escreva ("Não é uma equação do segundo grau")
    Senão
        delta ← delta + (b*b)*4*a*c
        Se (delta = 0) então
            Escreva ("Não existe raiz real")
        Senão
            se (delta > 0) então
                Escreva ("Existe uma raiz real")
            Senão
```

## ATIVIDADES

```

    se (delta = 0) então
        Escreva ("Existe uma raiz real")
    Senão
        se (delta = 0) então
            Escreva ("Existe uma raiz real")
             $x1 \leftarrow (-b) / (2 * a)$ 
            Escreva ("A raiz é:", x1)
        Senão
Se (delta = 0) então
    Escreva ("Existem duas raízes reais")
     $x1 \leftarrow (-b + )$ 
Senão
    delta  $\leftarrow$  delta  $\sqrt{b^2 - 4 * a * c}$ 
    Se (delta = 0) então
        Escreva ("Não existe raiz real")
    Senão
        se (delta = 0) então
            Escreva ("Existe uma raiz real")
        Senão
            se (delta = 0) então
                Escreva ("Existe uma raiz real")
             $x1 \leftarrow (-b + \sqrt{\text{delta}}) / (2 * a)$ 
            Escreva ("A raiz é x1:", x1)
             $x2 \leftarrow (-b + \sqrt{\text{delta}}) / (2 * a)$ 
            Escreva ("A raiz é x2:", x2)
        Fim-se
    Fim-se
Fim-se
Fim-se
Fim

```

Quadro 44: Pseudocódigo – Exercício 5



LIVRO

## **Algoritmos e Estruturas de Dados**

Lages e Guimarães

**Editora:** LTC

**Sinopse:** Os princípios da lógica de programação para computadores são apresentados utilizando uma linguagem de programação didática, denominada de Portugol, de modo a enfatizar a metodologia de criação de um algoritmo, independente da linguagem de programação utilizada pelo futuro programador.



# ESTRUTURA DE REPETIÇÃO



## Objetivos de Aprendizagem

- Estudar as estruturas de repetição controladas e condicionais.
- Conhecer as estruturas de repetição encadeadas.
- Construir algoritmos utilizando estruturas de repetição.

## Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Estrutura Para
- Estrutura Enquanto
- Estrutura Repita
- Estruturas de Repetição Encadeadas
- Problema 1
- Problema 2





## INTRODUÇÃO

Nesta unidade você aprenderá a construir algoritmos com repetição de um trecho de código. A repetição permite executar um conjunto de instruções quantas vezes forem necessárias sem ter que ficar reescrevendo trechos de códigos idênticos.

Será abordado o conceito de estrutura de repetição e sua aplicação. Estudaremos as estruturas de repetição com laços contados e laços condicionais. O uso de laços contados é restrito a situações em que sabemos previamente quantas vezes as instruções precisam ser executadas. Neste caso, abordaremos especificamente a Estrutura **Para**.

Nos laços condicionais não sabemos previamente o número de execuções e atrelamos a repetição a uma condição. Trataremos os casos com condição no início e no final do laço, estudando as estruturas: **Enquanto e Repita**.

Ao construir algoritmos utilizando estruturas de repetição, muitas vezes, precisaremos utilizar variáveis contadoras e acumuladoras. Aprenderemos os conceitos relacionados a essas variáveis, como utilizá-las e qual a diferença entre elas.

Estudaremos as estruturas de repetição (Para, Enquanto e Repita), destacando seu funcionamento, sintaxe e aplicação. Para facilitar o aprendizado construiremos algoritmos utilizando cada uma dessas estruturas.

Ao final desta unidade você saberá construir algoritmos com estruturas de repetição e poderá responder as questões relacionadas com o tema de Algoritmos e Lógica de Programação I, tais como: como repetir um trecho de código um número determinado de vezes? Como repetir um trecho de código com base em uma condição? Que estrutura de repetição é mais adequada para cada problema? Quando utilizar variável contadora e acumuladora? Quando utilizar estruturas de repetição encadeadas?

## ESTRUTURA DE REPETIÇÃO

Ao desenvolver algoritmos muitas vezes nos deparamos com situações em que precisamos repetir um determinado trecho de código ou todo o código um certo número de vezes. Por exemplo, se queremos efetuar a soma dos 100 primeiros números pares, somar  $n$  números enquanto o valor da soma não ultrapasse 500, calcular a média de 20 números, calcular a tabuada de um número, somar os números entre uma faixa de valores, efetuar um processamento enquanto o usuário informe “SIM”, validar um dado de entrada e outras (SALVETTI; BARBOSA, 1998; ASCENIO; CAMPOS, 2010).

Nos casos descritos acima e em muitos outros, podemos criar um *loop* para efetuar o processamento de um trecho de código quantas vezes forem necessárias. Na literatura essas estruturas de repetição (*loop*) são, também, denominadas de laços de repetição e malhas de repetição (MANZANO; OLIVEIRA, 1997).

Nas estruturas de repetição o número de repetições pode ser fixo ou estar relacionado a uma condição. Isto é, os laços de repetição podem ser classificados em laços contados e laços condicionais (ASCENCIO; CAMPOS, 2010).

Os laços contados são aqueles que utilizamos quando sabemos previamente quantas vezes o trecho do código precisa ser repetido. Por exemplo, realizar a leitura de 100 números, efetuar o somatório dos números entre 500 e 700 e outros. A estrutura utilizada para representar os laços contados é a **Estrutura Para**.

Os laços condicionais são utilizados quando não conhecemos o número de vezes que o trecho de código precisa ser repetido. A repetição está atrelada a uma condição que pode ser alterada dentro do laço. Por exemplo, solicitar que o usuário informe um número até que ele digite um número entre 1 e 12. Com isto, podemos efetuar a validação dos dados de entrada. Você se recorda do algoritmo que criamos na unidade II para escrever por extenso o nome do mês correspondente ao número digitado pelo usuário? Naquele caso, se o usuário informasse um número fora da faixa 1 e 12, emitíamos uma mensagem informando que não havia mês correspondente ao número digitado e a execução do algoritmo era encerrada. Utilizando laços podemos “forçar” a digitação de um dado de entrada válido, ou seja, enquanto o usuário não digitar um número dentro da faixa definida continuamos solicitando que ele informe um número.

Os laços condicionais podem ter o teste lógico no início ou no final do laço, configurando assim duas estruturas de repetição: **Estrutura Repita** e **Estrutura Enquanto**.

A vantagem da estrutura de repetição é que não precisamos reescrever trechos de código idênticos, reduzindo assim o tamanho do algoritmo. Além disso, podemos determinar repetições com número de vezes variável (LOPES; GARCIA, 2002).

No uso de estruturas de repetição observaremos que é necessário utilizar variáveis contadoras e acumuladoras. Uma variável contadora é uma variável que recebe um valor inicial antes de iniciar a estrutura de repetição e no interior dessa estrutura seu valor é incrementado em um valor constante. Já uma variável acumuladora é uma variável que recebe um valor inicial antes do início de uma estrutura de repetição e é incrementada no interior dessa estrutura em um valor variável. O que difere uma variável contadora de uma acumuladora é o valor que elas são incrementadas na estrutura de repetição. Em uma variável contadora o valor é fixo e em uma variável acumuladora o valor é constante.

Nas seções seguintes será apresentada cada uma das estruturas de repetição, destacando sua sintaxe e aplicação.

## ESTRUTURA PARA

A estrutura Para é uma estrutura do tipo laço contado, utilizada para um número definido de repetições. Isto é, devemos utilizar essa estrutura quando sabemos o número de vezes que o trecho de código precisa ser repetido. Outro termo utilizado para essa estrutura de repetição é o de estrutura de repetição com variável de controle, pois é utilizada uma variável contadora para controlar o número de repetições. A sintaxe da estrutura Para é:

```
para <variável> de <início> até <fim> passo <incremento>
faça
    <instruções>
fim_para
```

Em que:

- <variável>: é a variável contadora utilizada para controlar a estrutura de repetição. Esta variável tem que ser do tipo inteiro.
- <início> e <fim>: esses termos delimitam o intervalo para a execução do laço de repetição. Podem ser constantes inteiras, funções ou expressões que retornem números inteiros (SALVETTI; BARBOSA, 1998).
- <incremento> representa o valor que será incrementado ou decrementado (se for um valor negativo) a cada passagem do laço, isto é, como será a variação da variável de controle (contador).
- Esse termo pode ser representado por uma constante ou uma variável.

Lopes e Garcia (2002) destacam que o número de repetições do bloco de comandos é igual ao número de termos da série delimitada pelos termos <início> e <fim>. A variável contadora não deve aparecer em um comando de leitura dentro do bloco de repetição.

Agora que você conheceu a teoria sobre a estrutura de repetição para, vamos resolver um problema utilizando-a para tornar mais clara a sua aplicação prática. Você se lembra da tabuada? A Figura 6 apresenta a tabuada para o número 5, em que temos o produto entre o número 5 e os números compreendidos entre 0 e 10.

Que tal construir um algoritmo para efetuar a tabuada de um número qualquer?

O primeiro passo é estruturar o nosso problema seguindo os passos descritos na Unidade I:

**Objetivo do algoritmo:** calcular a tabuada de um número inteiro.

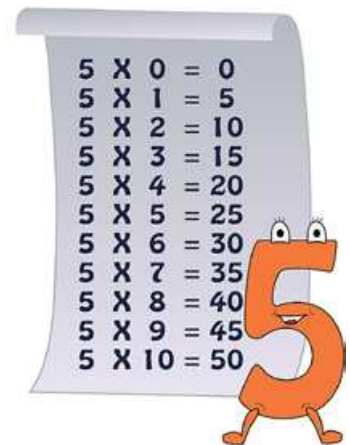


Figura 6: Tabuada do número 5

**Entrada:** obter um número inteiro.

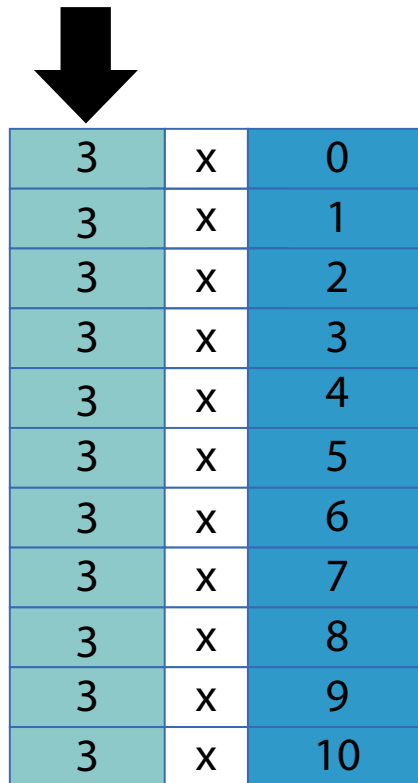
**Processamento:** efetuar a operação de multiplicação do número informado pelos valores compreendidos entre 1 e 10.

**Saída:** imprimir a tabuada de 1 a 10 do número informado na entrada.

Na entrada de dados temos que ler um número inteiro, isto implica que precisamos declarar uma variável do tipo inteira para armazenar o número digitado pelo usuário. Denominaremos essa variável de num.

O processamento consiste em multiplicar o número recebido na entrada (armazenado na variável num) pelos valores 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 e 10, como pode ser visualizado na Figura 7. Criaremos uma variável chamada mult para armazenar o resultado da multiplicação. Observe que temos a repetição de uma expressão aritmética de multiplicação ( $\text{num} \times i$ ), em que sabemos previamente o número de repetições. Portanto, podemos utilizar a estrutura Para. Lembre-se que ao utilizar esta estrutura precisamos declarar uma variável contadora que deve ser do tipo inteiro, nomearemos de i. A variável i deve ter início em 0 e fim em 10, pois queremos mostrar a tabuada de 0 a 10. O passo a ser utilizado é 1. Como saída do algoritmo temos que imprimir o resultado da operação de multiplicação.

Número obtido  
na entrada



3	x	0
3	x	1
3	x	2
3	x	3
3	x	4
3	x	5
3	x	6
3	x	7
3	x	8
3	x	9
3	x	10

Operação aritmética  
a ser executada

Figura 7: Processamento a ser executado  
Fonte: a autora

O Quadro 45 apresenta o algoritmo para o problema da tabuada. Observe que tanto o processamento (expressão aritmética dada por  $\text{mult} = \text{num} * i$ ) quanto a saída de dados (**Escreva** num, "x", i, "=", mult) se encontram dentro do laço de repetição. Por que isso acontece? Pois, temos que imprimir o resultado de 10 operações de multiplicação e não apenas uma. O que acontece se colocarmos a saída de dados fora do laço? No vídeo será apresentado apenas o resultado da última operação, em que num é igual a 3, i igual a 10. Portanto, será exibido  $3 \times 10 = 30$ .

Lembre-se que um laço de repetição pode ser utilizado tanto para entrada, processamento, quanto para a saída de dados.

```
Algoritmo tabuada
Var
    Num, i, mult: inteiro
Início
    Escreva ("Digite um número:")
    Leia (num)
    Para i de 1 até 10 passo 1 faça
        mult num*i
        Escreva (num, "x", i, "=", mult)
    fim_para
Fim.
```

Quadro 45: Pseudocódigo - Algoritmo tabuada

A Figura 8 ao lado ilustra a simulação do algoritmo apresentado no Quadro 45.

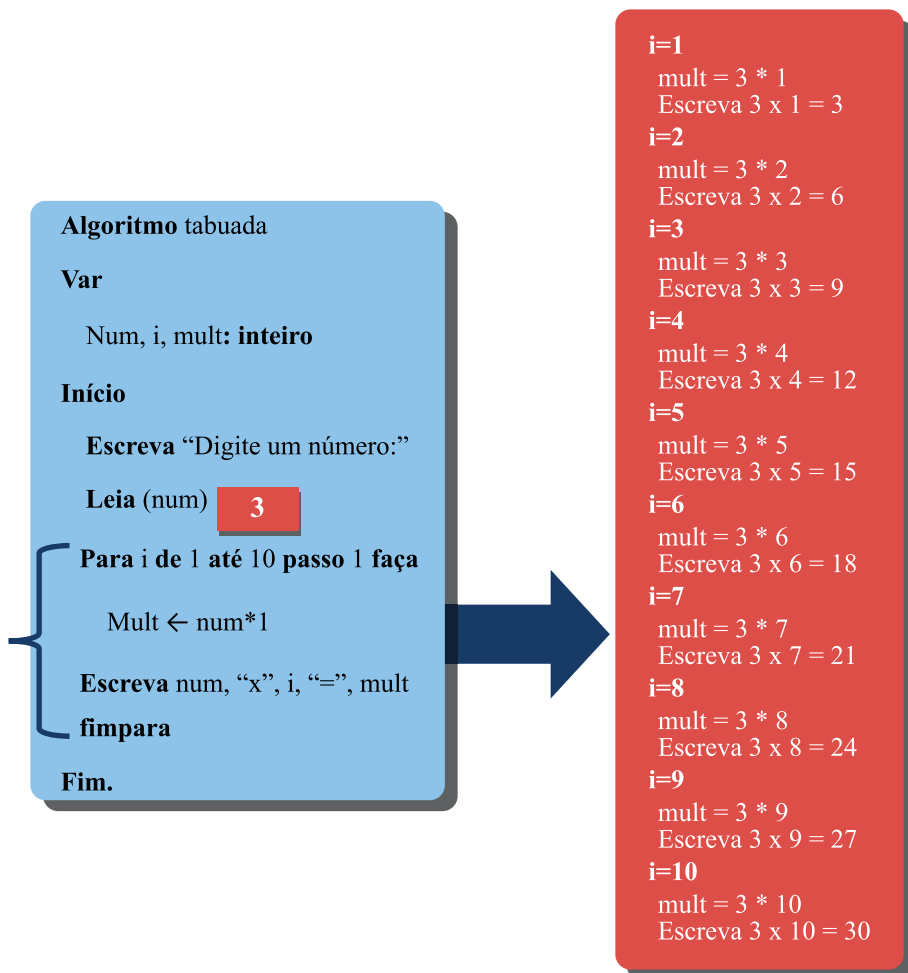


Figura 8: Simulação - Algoritmo Tabuada  
Fonte: a autora

Na Unidade II discutimos sobre algoritmos eficientes. Você se lembra o que são algoritmos eficientes? Podemos melhorar o algoritmo construído para a tabuada? Sim, podemos economizar uma variável, no caso a variável mult, e com isto, retirar a instrução de atribuição, realizando a operação aritmética diretamente no comando escreva. Isto é possível, pois no comando escreva podemos colocar uma expressão. No Quadro 46 podemos visualizar outro modo de escrever o algoritmo para a tabuada.

```
Algoritmo tabuada
Var
    Num, i: inteiro
Início
    Escreva ("Digite um número:")
    Leia (num)
    Para i de 1 até 10 passo 1 faça
        Escreva (num, "x", i, "=", mult*i)
    fim_para
Fim.
```

Quadro 46: Pseudocódigo - Algoritmo tabuada

Nesta seção estudamos a estrutura de repetição controlada, que utiliza uma variável contadora para controlar o laço. Essa estrutura deve ser utilizada nas situações em que sabemos previamente quantas vezes o comando deve ser repetido.

## ESTRUTURA ENQUANTO

A estrutura Enquanto é uma estrutura do tipo laço condicional, isto é, o *loop* baseia-se na análise de uma condição. Essa estrutura é utilizada quando temos um número indefinido de repetições e se caracteriza por realizar um teste condicional no início.

A sintaxe da estrutura Enquanto é:

```
Enquanto <condição> faça
    <instruções>
fim_enquanto
```

Na estrutura **para** tínhamos uma variável de controle (contador) para controlar o número de repetições do algoritmo. Na estrutura **Enquanto** não há variável de controle, sendo imposta uma condição para controlar a repetição do algoritmo.



Devemos tomar cuidado para garantir que em algum momento a condição será satisfeita, senão o algoritmo pode entrar em *loop* (não parar nunca). Para impedir o loop do algoritmo utilizamos uma expressão condicional, tal como vimos na Unidade II, para parar a repetição (LOPES; GARCIA, 2002).

Outra situação é que como o teste condicional é executado no início, podem ocorrer casos em que as instruções da estrutura de repetição nunca sejam executadas. Isso acontece quando o teste condicional da estrutura resulta em falso logo na primeira comparação (ASCENCIO; CAMPOS, 2010).

Agora que conhecemos os conceitos relacionados à estrutura enquanto, vamos construir um algoritmo para o seguinte problema: ler vários números e informar quantos se encontram no intervalo de 100 a 300. Se for digitado o valor 0, o algoritmo encerra sua execução.

Seguindo o método de estruturação de problemas visto na Unidade I, temos:

**Objetivo do algoritmo:** ler vários números e informar quantos estão no intervalo entre 100 e 300.

**Entrada:** ler números inteiros até que seja digitado o número zero.

**Processamento:** contar quantos números estão no intervalo entre 100 e 300.

**Saída:** imprimir a quantidade de números entre 100 e 300.

Na entrada de dados temos que realizar a leitura de números inteiros repetidas vezes, até que o valor zero seja digitado. O processamento consiste em contar a quantidade de número que estão na faixa entre 100 e 300, para isso utilizaremos uma variável do tipo contador, que nomearemos como *cont*. Para saber quantos valores estão dentro da faixa utilizaremos a estrutura condicional *Se*, conforme visto na Unidade II. Como saída temos que informar o valor da variável *cont*.

Na construção de algoritmos utilizando a estrutura enquanto temos que o teste lógico é realizado no início, deste modo precisamos ter um valor atribuído para a variável usada na condição antes de entrar na estrutura enquanto. Além disso, no conjunto de instruções dentro do laço de repetição deve haver uma instrução que modifique o valor dessa variável, senão entraremos em um *loop*. Isto nos indica que ao utilizar laços do tipo enquanto temos que ler a variável fora da estrutura de repetição e dentro. Observe o algoritmo do Quadro 47.

```
Algoritmo conta
Var
    Num, cont: inteiro
Início
    Escreva ("Digite um número:")
    Leia (num)
    cont 0
    Enquanto (num <> 0) faça
        Se (num >=100) e (num <=300) então
            cont cont +1
        fim_se
        Escreva ("Digite um número:")
    Leia (num)
    fim_enquanto
    Escreva ("A quantidade de números entre 100
    e 300 é:", cont)
Fim.
```

Quadro 47: Pseudocódigo - Algoritmo conta

Vamos analisar este algoritmo linha a linha a partir da instrução de Início. Temos um comando **escreva**, que envia uma mensagem ao usuário que digite um número. O número digitado pelo usuário é armazenado na variável **num** (comando **Leia**). Em seguida, temos uma atribuição a variável **cont**, que é um contador. Por quê? Sempre que utilizamos variáveis desse tipo devemos inicializá-la, pois uma variável é um espaço de memória e pode conter “lixos”. Portanto, sempre inicialize as variáveis que exercem função de contador e acumulador.

A próxima linha é a instrução **enquanto** em que temos o teste lógico que analisa se o número é diferente de 0. Se o resultado for verdadeiro, temos a execução das instruções que estão dentro do laço, senão vai para a instrução após o **fim\_enquanto**. No laço de repetição temos a verificação se o número está ou não na faixa estabelecida. Para isso é usada a estrutura condicional **Se**, em que temos duas expressões relacionais unidas por uma expressão lógica com o operador

E. Se o resultado do teste lógico for verdadeiro temos que cont recebe o valor que ele tem mais um, ou seja, é incrementado em uma unidade. Se o teste lógico resultar em falso a execução segue para a linha posterior ao fim\_se. Note que após o fim\_se temos a leitura da variável novamente. Por que isso acontece? Se a leitura da variável fosse realizada apenas fora do laço de repetição teríamos que o laço entraria em *loop*, uma vez que teríamos o mesmo valor para num. As instruções dentro do laço serão repetidas até que na entrada seja obtido o valor zero. Quando este valor for obtido tem-se a execução do comando após o fim\_enquanto, que exibe na tela o valor armazenado na variável cont. Na Figura 9 podemos visualizar a simulação do algoritmo descrito.

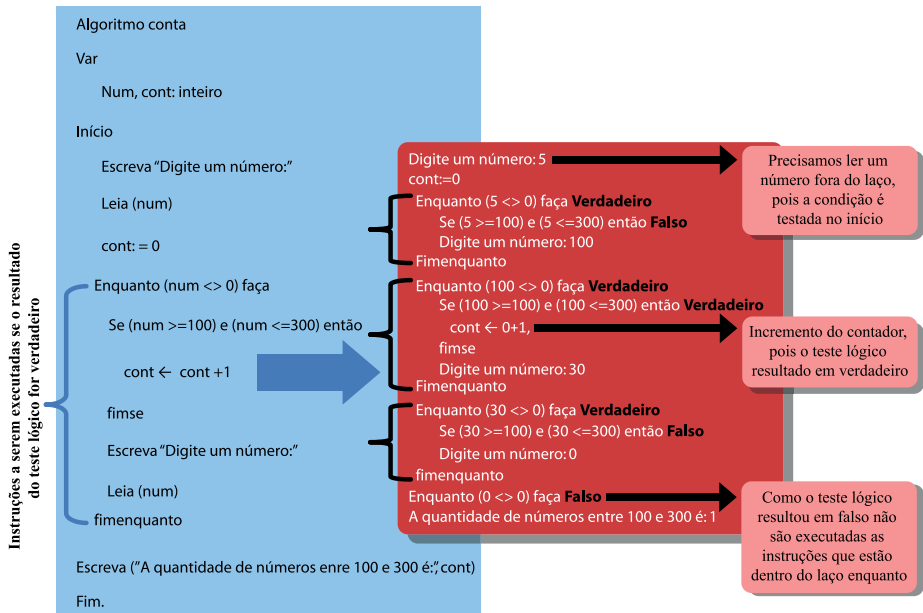


Figura 9: Simulação - Algoritmo Conta

Vamos analisar o comportamento do algoritmo (Quadro 48) sem a entrada de dados dentro da estrutura de repetição. Note que a leitura está sendo realizada apenas antes da estrutura de repetição.

```
Algoritmo conta
Var
    Num, cont: inteiro
Início
    Escreva ("Digite um número:")
    Leia (num)
    cont 0
    Enquanto (num <> 0) faça
        Se (num >=100) e (num <=300) então
            cont cont +1
        fim_se
    fim_enquanto
    Escreva ("A quantidade de números
    entre 100 e 300 é:", cont)
Fim.
```

Quadro 48: Pseudocódigo - Algoritmo conta

A Figura 10 apresenta um esquema que representa a simulação do algoritmo. A partir da entrada de dados, em que foi obtido o valor 130, tem-se a inicialização da variável cont e, em seguida, a estrutura enquanto. No início do enquanto há um teste lógico que analisa se num é diferente de 0. Como 130 é diferente de 0, a avaliação do teste resulta em verdadeiro, com isto, tem-se a execução das instruções que estão dentro da estrutura enquanto. Internamente, há um teste lógico que verifica se o número é maior ou igual a 100 e menor ou igual a 300, a avaliação deste teste é verdadeira. Portanto, tem-se o incremento da variável cont, que passa a armazenar o valor 1. Após o fim\_se não temos nenhuma instrução, deste modo a execução é retomada para a linha do enquanto, em que novamente o teste lógico é executado. A variável num continua com o valor 130, resultando em verdadeiro o teste lógico, pois 130 é diferente de 0. Na verificação da estrutura condicional o teste, também, resulta em verdadeiro, com isto é executada a instrução que incrementa a variável cont. Novamente a execução retorna para

a linha do enquanto e o funcionamento do algoritmo será o mesmo, de modo que apenas a variável `cont` está sendo incrementada a cada execução. Perceba que se não efetuarmos a leitura da variável `num` dentro da estrutura enquanto, também, seu valor não será alterado e o código entra em loop, pois sempre a condição `num <> 0` é satisfeita.

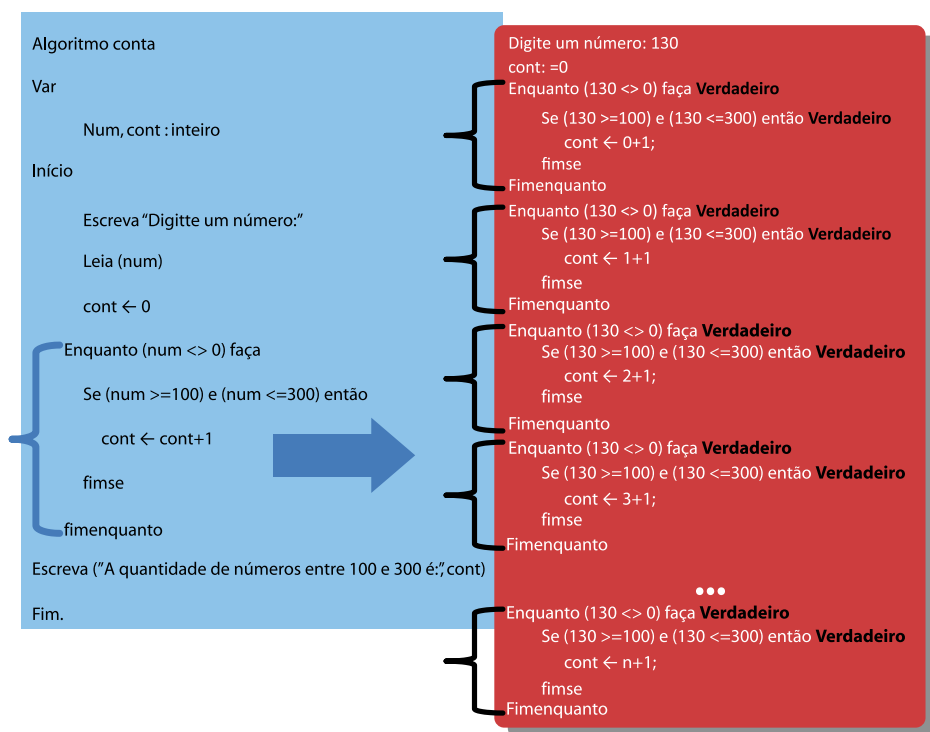


Figura 10- Simulação - Algoritmo conta

REFLITA



Lembre-se!!! SEMPRE que você utilizar uma estrutura de repetição condicional tem que ter uma instrução no interior desta estrutura que modifique o valor da variável que é utilizada no teste lógico. Variáveis contadoras e acumuladoras precisam ser inicializadas no início do código.

## ESTRUTURA REPITA

A estrutura Repita é uma estrutura do tipo laço condicional, isto é, o *loop* baseia-se na análise de uma condição. Essa estrutura é utilizada quando temos um número indefinido de repetições e precisamos que o teste condicional seja realizado após a execução do trecho de código. Isto é, devemos utilizar essa estrutura quando não sabemos o número de vezes que um trecho do código deve ser repetido (ASCENCIO; CAMPOS, 2010).

A sintaxe da estrutura **Repita** é:

```
Repita  
    <instruções>  
Até_que <condição>
```

Observe que na estrutura Repita as instruções dentro do laço serão executadas pelo menos uma vez, pois a análise condicional é executada ao final. Do mesmo modo que na estrutura condicional enquanto, lembre-se que nas instruções que estão dentro da estrutura de repetição tem que haver uma instrução que altere o valor da <condição>.

Com o conhecimento que temos sobre a estrutura Repita vamos reescrever o algoritmo que lê vários números e informa quantos estão no intervalo de 100 a 300. Se for digitado o valor 0, o algoritmo encerra sua execução. Descrevemos os passos da estruturação deste problema no tópico “ESTRUTURA ENQUANTO”.

A entrada de dados consiste na leitura de números inteiros repetidas vezes, até que o valor zero seja digitado. O processamento é contar a quantidade de número que estão na faixa entre 100 e 300. E a saída é informar quantos dos números lidos na entrada são maiores ou iguais a 100 e menores ou iguais a 300. O Quadro 49 apresenta um algoritmo para este problema utilizando a estrutura de repetição condicional Repita.

```

Algoritmo conta
Var
    Num, cont: inteiro
Início
    cont 0
    Repita
        Escreva ("Digite um número:")
        ... Leia (num)
        Se (num >=100) e (num <=300) então
            cont cont +1
        fim_se
    Até_ que (num = 0)
    Escreva ("A quantidade de números entre 100
    e 300 é:", cont)
Fim.

```

Quadro 49: Pseudocódigo - Algoritmo conta

Vamos estudar cada linha do algoritmo para entender melhor o funcionamento dessa estruturação de repetição. Na primeira linha temos a inicialização da variável `cont`, que conta o número de valores que estão na faixa entre 100 e 300. Afinal, por que inicializamos essa variável? Por exemplo, se efetuamos a leitura de vários números e nenhum deles estava na faixa entre 100 e 300, qual o valor de `cont`? Não há como garantir que o valor será zero. Como uma variável é um espaço em memória, devemos inicializá-lo para que não fique nenhum “lixo”.

Após a inicialização de `cont`, temos o início da estrutura `Repita`. Internamente a essa estrutura temos a leitura do número, o qual é armazenado na variável `num`. Em seguida, temos a estrutura condicional `se`, que analisa se o número é maior ou igual a 100 e menor ou igual a 300. O resultado do teste lógico é verdadeiro quando as duas expressões relacionais são verdadeiras e então `cont` é

incrementado em 1. Se o resultado do teste for falso vai diretamente para a linha que impõe a condição para o laço de repetição. Nesta linha, temos a verificação se o número é igual a zero, isto é, quando o número for igual a zero, a repetição do laço finaliza e é executada a instrução escreva.

Observe que na estrutura Repita a leitura da variável é realizada internamente. Isso acontece porque o teste lógico é executado no final. Deste modo, o conjunto de instruções que estão dentro do laço é executada uma ou mais vezes. Na estrutura enquanto o conjunto de instruções pode não ser executado, pois o teste lógico é realizado no início.

## ESTRUTURAS DE REPETIÇÃO ENCADEADAS

Do mesmo modo que na estrutura condicional, podemos ter encadeamento das estruturas de repetição, isto é, podemos ter uma estrutura de repetição dentro de outra. Neste caso é necessário que todas as instruções da construção interna estejam embutidas na construção externa (MANZANO; OLIVEIRA, 1997; LOPES; GARCIA, 2002).

Não existem regras para o encadeamento das estruturas de repetição. De modo que você precisa conhecer cada uma delas para saber quando é conveniente encadeá-las, quais devem ser utilizadas e como. Não se preocupe! Conforme vamos construindo algoritmos aperfeiçoamos o nosso raciocínio lógico de modo que automaticamente saberemos o momento de utilizar essas estruturas (MANZANO; OLIVEIRA, 1997).

A sintaxe para encadeamento da estrutura **Para** com **Para** é dada por:

```
para <var1> de <início> ate <fim> passo <incremento> faça  
    para <var2> de <início> ate <fim> passo <incre-  
mento> faça  
    <instruções>  
    fim_para  
fim_para
```



Podemos utilizar encadeamento utilizando a estrutura **Para** e **Enquanto**, em que a sintaxe é:

```

para <variável> de <início> até <fim> passo <incremento>
faça
    enquanto (<condição>) faça
        <instruções>
    fim_enquanto
fim_para

```

No encadeamento da estrutura **Para** com **Repita** a sintaxe é:

```

para <variável> de <início> até <fim> passo <incremento>
faça
    repita
        <instruções>
    Até_que (<condição>)
fim_para

```

Alguns exemplos de encadeamento de estruturas de repetição utilizando a estrutura **Enquanto** são:

```

enquanto (<condição1>) faça
    enquanto (<condição2>) faça
        <instruções>
    fim_enquanto
fim_enquanto

```

```

enquanto (<condição1>) faça
    repita
        <instruções>
    até_que (<condição2>)
fim_enquanto

```

```

enquanto (<condição>) faça
    para <var> de <início> até <fim> passo <incremento>
faça
    <instruções>

```

```

        fim_para
    fim_enquanto

```

Nos casos de encadeamento utilizando a estrutura **Repita** e as demais temos as seguintes sintaxes:

```

repita
    para <var> de <início> até <fim> passo <incremento>
    faça
        <instruções>
    fim_para
até_que (<condição>)

repita
    enquanto (<condição2>) faça
        <instruções>
    fim_enquanto
até_que (<condição1>)

repita
    repita
        <instruções>
    até_que (<condição>)
até_que (<condição>)

```

Fique tranquilo! Veremos a aplicação de estruturas encadeada no PROBLEMA 2.

## PROBLEMA 1

Você se recorda de fatorial? O fatorial de um número é o produto dos números naturais começando em  $n$  e decrescendo até 1, isto é, o produto de todos os inteiros positivos menores ou igual a  $n$ . O fatorial de um número é representado por  $n!$

Tomemos como exemplo o fatorial de 7, representado por  $7!$ . O cálculo de  $7!$  é:  $7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$ , que é igual a 5040.

Agora que recordamos o que é o fatorial, vamos escrever um algoritmo para calcular o fatorial de um número qualquer. Seguindo o método de estruturação de problemas, conforme visto na Unidade I, temos que:

- **Objetivo do algoritmo:** calcular o fatorial de um número.
- **Entrada:** ler o número que se deseja calcular o fatorial.
- **Processamento:** efetuar o produto de todos os números inteiros positivos menores ou igual ao número lido na entrada.
- **Saída:** imprimir o fatorial.

Na entrada do problema temos a leitura de um número inteiro, do qual queremos obter o fatorial. Precisamos armazenar este número em uma variável inteira, denominaremos num. O processamento consiste em efetuar a operação aritmética de multiplicação de todos os números inteiros positivos menores ou igual a num. Observe que temos operações de multiplicação sucessivas, partindo de 7, conforme ilustra a Figura 11. Aqui temos uma informação importante, a operação de multiplicação tem que começar em 7 e ir até 1, ou seja, sabemos o número de vezes que a multiplicação precisa ser executada. Quando sabemos o número de repetições que deve ocorrer temos um laço contado. Portanto, podemos utilizar a estrutura **Para** em nosso algoritmo. Nessa estrutura há uma variável de controle que delimita o intervalo para a execução do laço, no caso temos que partir de 7 e ir até 1, isto é, teremos um incremento de -1 (decremento). Não podemos esquecer de declarar mais uma variável, a variável de controle, que é do tipo int, chamaremos de cont. Como saída temos que informar o valor do cálculo do fatorial, utilizaremos o comando escreva.

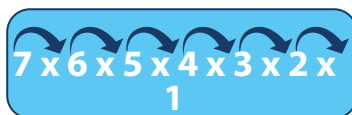


Figura 11: Representação do cálculo de 7!

No Quadro 50 temos o algoritmo para o problema do fatorial. Faça você um teste de mesa efetuando o cálculo do fatorial de 5.

**Algoritmo** fatorial

**Var**

num, fat, cont: **inteiro**

**Início**

**Escreva** ("Digite o número que deseja calcular o fatorial:")

**Leia** (num)

fat 1;

**Para** cont **de** num **até** 1 **passo** -1 **faça**

fat fat\*cont

**fim\_para**

**Escreva** ("O fatorial é: ", fat)

**Fim.**

Quadro 50: Pseudocódigo - Algoritmo fatorial

Ao analisar o algoritmo surge o questionamento: por que a variável fat foi inicializada com 1? Podemos inicializá-la com 0? Se a inicialização da variável for 0, e considerando num como 5, teríamos fat igual a 0, conforme ilustra a Figura 12.

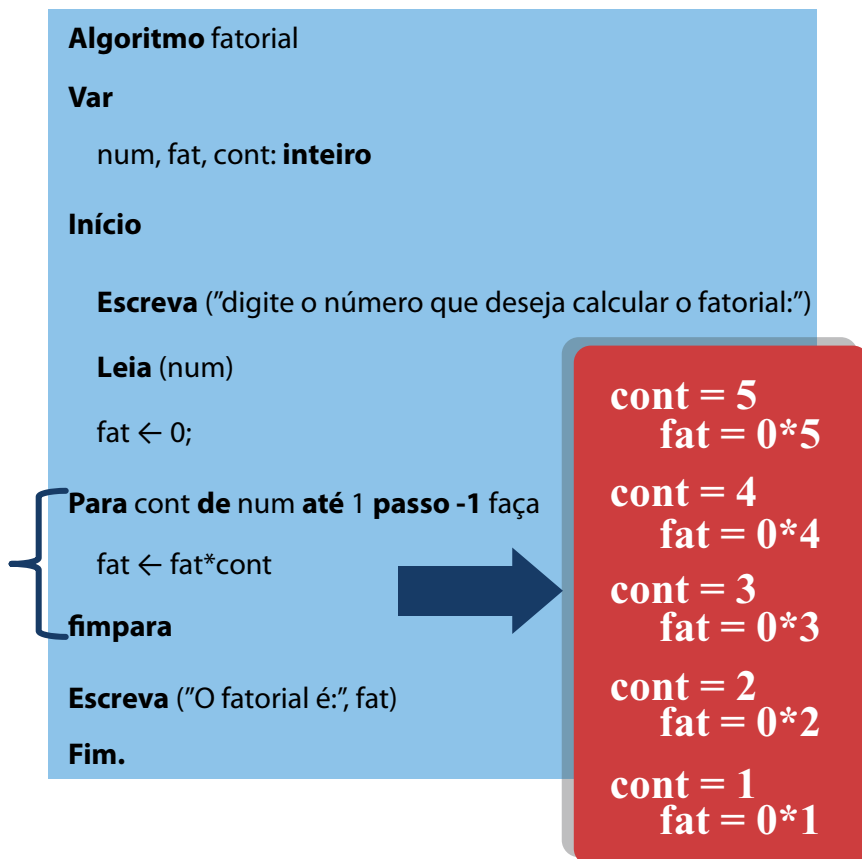


Figura 12: Simulação - Algoritmo Fatorial

Após a simulação, fica claro que a variável fat deve ser inicializada com 1. Deste modo, teríamos o resultado correto para o fatorial de 5, que é 120.

## PROBLEMA 2

A aprovação de um aluno em uma disciplina ocorre quando a média das notas é maior ou igual a 7. Para auxiliar o professor em suas atividades elabore um algoritmo, leia o número de notas da disciplina, código do aluno e as notas do aluno. Calcule a média final de cada aluno e informe o número de alunos aprovados e reprovados. O algoritmo deve ser executado até que seja informado um código 0. Sistematizando as informações do problema de acordo com o método, temos que:

- **Objetivo do algoritmo:** calcular o número de alunos aprovados e reprovados.
- **Entrada:** ler o número de notas da disciplina, código e notas do aluno.
- **Processamento:** calcular a média dos alunos e contar o número de aprovados e reprovados.
- **Saída:** imprimir o número de aprovados e reprovados.

Na entrada de dados temos que saber inicialmente o número de notas da disciplina. Este valor deve ser armazenado em uma variável (*nnotas*). Além disso, precisamos armazenar o código do aluno (*cod*) e nota (*nota*). O processamento consiste em a partir do número de notas informado para a disciplina, efetuar a repetição da leitura de notas e somá-las (variável *soma*) e calcular a média aritmética do aluno (variável *média*). A média deve ser armazenada em uma variável do tipo real. Se a média for maior ou igual a 7 o aluno está aprovado, senão reprovado. Para contabilizar o número de aprovados e reprovados, precisamos declarar duas variáveis do tipo inteira, aprovados e reprovados, respectivamente. Como saída temos o número de alunos aprovados e reprovados.

No Quadro 51 é apresentado o algoritmo para o problema descrito. Neste algoritmo temos a aplicação de estruturas de repetição encadeada. Por quê? Precisamos saber o número de notas que devem ser lidas e devem ser lidas as notas de vários alunos até que o código do aluno seja 0. Como não sabemos quantos alunos teremos que ler as notas, precisamos utilizar uma estrutura do tipo laço condicional. No primeiro momento optamos por utilizar a estrutura **Enquanto**. Nesta estrutura o teste condicional é realizado no início, deste modo temos que realizar a leitura da informação utilizada no teste condicional fora da estrutura. No caso, a informação obtida foi o código do aluno.

Se o teste condicional resulta em verdadeiro, ou seja, o código do aluno é diferente de 0, precisamos ler as notas do aluno. Em seguida, temos a inicialização das variáveis *soma* e *média*, pois a soma deve acumular as notas de um aluno e não de todos. Para a leitura das notas utilizamos a estrutura **Para**, pois sabemos previamente o número de notas que precisa ser lido. Dentro da estrutura **Para** é realizada a soma das notas do aluno. Após ser finalizada a leitura de todas as notas (acumuladas na variável *soma*) é efetuado o cálculo da média do

aluno. Em seguida, temos o teste condicional que analisa se o aluno está aprovado ou reprovado. Se a média é maior ou igual a 7 temos o incremento da variável aprovados, senão a variável reprovados é incrementada. As instruções seguintes efetuam a leitura do código do aluno. Depois do fim\_enquanto são apresentados os números de aprovados e reprovados.

```

Algoritmo fatorial
Var
    nnotas, cod, aprovados, reprovados, cont: inteiro
    media, nota, soma: real
Início
    Escreva ("Informe o número de notas da disciplina:")
    Leia (nnotas)
    aprovadas ← 0
    reprovada ← 0
    Enquanto (cod <> 0) faça
        soma ← 0
        media ← 0
        Para cont de 1 até nnotas passo 1 faça
            Escreva ("Informe a nota:")
            leia (nota)
            soma ← soma + nota
        Fimpara
        media ← soma/nnotas;
        se (media >=7) então
            aprovados ← aprovados + 1
        senão
            reprovados ← reprovados + 1
        Fim-se
        Escreva ("Informe o código do aluno:")
        Leia (cod)

```

**fimenquanto**

**Escreva** ("O número de aprovados é:",  
aprovados")

**Escreva** ("O número de reprovados é:",  
reprovados)

**Fim.**

Quadro 51: Pseudocódigo - Algoritmo notas

Em relação ao algoritmo apresentado: o que aconteceria se o cálculo da média fosse executado dentro da estrutura **Para**? Tomemos como exemplo, que foram obtidas quatro notas 7, 8, 7 e 9 respectivamente. Na primeira passagem da estrutura **Para** teríamos média igual a 1,75. Na segunda passada a média seria 2, na terceira 1,75 e na quarta 3. Isto é, a cada nota obtida seria efetuada a divisão desta nota pelo número de notas (nota/nnotas). Portanto, no interior da estrutura **Para** devemos apenas acumular o valor das notas e realizar o cálculo da média depois que todas as notas tiverem sido lidas.

Outro questionamento é: o que aconteceria se não tivéssemos a instrução para leitura do código do aluno no interior da estrutura **Enquanto**? O algoritmo entraria em *loop*, pois o valor da variável código nunca seria alterado. Portanto, lembre-se sempre que utilizamos a estrutura **Enquanto** devemos efetuar a leitura da variável utilizada no teste condicional antes do **Enquanto** e dentro da estrutura.

No Quadro 52 temos a solução do problema utilizando o encadeamento com estrutura **Repita** e **Para**.

**Algoritmo** fatorial

**Var**

nnotas, cod, aprovados, reprovados, cont: **inteiro**  
media, nota, soma: **real**

**Início**

**Escreva** ("Informe o número de notas da disciplina:")

**Leia** (nnotas)

aprovadas  $\leftarrow$  0



```

reprovada ← 0
Repita
    Escreva ("Informe o código do aluno:")
    leia (cod)
    soma ← 0
    media ← 0
    Para cont de 1 até nnotas passo 1 faça
        Escreva ("Informe a nota:")
        Leia (nota)
        soma ← soma + nota
    Fimpara
    media ← soma/nnotas
    se (media ≥ 7) então
        aprovados ← aprovados + 1
    senão
        reprovados ← reprovados + 1
    fim-se
    ate_que (cod = 0)
    Escreva ("O número de aprovados é:", aprova
dos)
    Escreva ("O número de reprovados é:", repro
vados)

Fim

```

Quadro 52: Pseudocódigo – Algoritmo notas

Observe que quando utilizamos o **Repita** não há a leitura da variável utilizada no teste condicional antes da estrutura condicional. Isso é possível, pois a estrutura **Repita** realiza o teste no final.

## CONSIDERAÇÕES FINAIS

Nesta unidade você aprendeu a construir algoritmos utilizando estruturas de repetição, que permitem a execução de um trecho de código repetidas vezes. As estruturas de repetição também são chamadas de laço de repetição.

Estudamos os laços de repetição contados e os laços condicionais. Nos laços de repetição contados conhecemos a estrutura **Para**, que é utilizada nos casos em que sabemos quantas vezes o trecho de código precisa ser repetido. A estrutura **Para** também é conhecida como estrutura de repetição com variável de controle, pois utilizamos uma variável contadora para controlar o número de repetições.

Nos laços de repetição condicionais vimos as estruturas **Enquanto** e **Repita**. Vimos que a estrutura **Enquanto** é utilizada quando não sabemos previamente o número de repetições que deve ser executado e impomos uma condição que é realizada no final. Aprendemos que no uso desta estrutura devemos utilizar um comando **leia** ou de atribuição antes do **enquanto** para entrar na repetição e um comando **leia** ou de atribuição (última instrução dentro da estrutura de repetição) para sair da repetição.

Do mesmo modo que a estrutura **Enquanto**, a estrutura **Repita** é utilizada quando temos um número indefinido de repetições, no entanto, o teste lógico é realizado no final. Além de estudar as estruturas de repetição conhecemos os conceitos e aplicações de variáveis do tipo contador e acumulador.

Observamos que as estruturas baseadas em laços condicionais são mais flexíveis e que podem ser substituídas uma pela outra, isto é, podemos resolver um problema com algoritmo utilizando a estrutura **Enquanto** ou com a estrutura **Repita**. Destaca-se que a estrutura **Para** pode ser substituída pelo uso de estruturas baseadas em laços condicionais. Mas, o contrário não é verdadeiro.

Vimos que o conceito de encadeamento pode ser aplicado às estruturas de repetição, de modo análogo à estrutura condicional, vista na Unidade II.

Ao longo desta unidade construímos algoritmos utilizando todos os conceitos aprendidos e, também, discutimos as particularidades de cada estrutura de repetição enfatizando a forma de uso de cada uma delas e o seu uso encadeado. Para aperfeiçoar o raciocínio lógico e fixar os conceitos vistos faça as atividades de autoestudo.

## REFLITA



As estruturas mais versáteis são **Enquanto** e **Repita**, pois podem ser substituídas uma pela outra, além de poderem substituir perfeitamente a estrutura **Para**. Porém, há de considerar-se que nem toda estrutura **Enquanto** ou **Repita** poderá ser substituída por uma estrutura **Para**.



## NA WEB

**Para entender um pouco mais sobre a estrutura de repetição, acesse o vídeo disponível em:**

<<http://www.youtube.com/watch?v=av5T0y6rwdk>>.

## ATIVIDADES



1. Escreva um algoritmo que leia 20 nomes e imprima o primeiro caractere de cada nome.
2. Formule um algoritmo que entre com o nome do aluno e as notas de quatro provas de 20 alunos. Imprima nome, nota1, nota2, nota3, nota4 e média de cada aluno e informe a média geral da turma.
3. Escreva um algoritmo que leia a quantidade de números que se deseja digitar. Em seguida, leia esses números e encontre o maior e o menor.
4. Construa um algoritmo que leia números inteiros até que seja digitado o 0. Calcule e escreva o número de valores lidos, a média aritmética, a quantidade de números pares e a quantidade de números ímpares.
5. Apresente todos os números divisíveis por 5 que sejam menores que 200.
6. Construa um algoritmo que leia números inteiros até que seja digitado um valor negativo. Ao final, informe a média dos números, o maior e o menor valor.
7. Escreva um algoritmo que leia vários nomes até que seja digitado o valor FIM. Imprima o primeiro caractere de cada nome. Dica: em uma variável do tipo caractere, para imprimir o primeiro caractere utilize o comando `Escreva nomevariavel[1]`.
8. Uma indústria produz e vende vários produtos e para cada um deles têm-se os seguintes dados: nome, quantidade produzida e quantidade vendida. Formule um algoritmo que:
  - Leia a quantidade de produtos que a empresa possui.
  - Imprima nome e quantidade em estoque para os produtos com estoque menor que 30.
  - Imprima nome do produto com maior quantidade em estoque.
9. Elabore um algoritmo que imprima todas as tabuadas do 1 ao 10.

## EXERCÍCIOS DE FIXAÇÃO

1. Escreva um algoritmo que leia o número de vezes que se deseja imprimir a palavra "ALGORITMOS" e imprimir.

**Objetivo do algoritmo:** informar se a pessoa é menor de idade, maior ou idoso.

**Entrada:** ler um número inteiro.

**Processamento:** não há.

## ATIVIDADES



**Saída:** imprimir a palavra “Algoritmos” o número de vezes informado.

```

Algoritmo palavra
Var
    num, i: inteiro

Início
    Escreva ("Informe o número de vezes que
    deseja imprimir:")
    Leia (num)
    Para i de 1 até num passo 1 faça
        Escreva ("ALGORITMOS")
    fim_para

Fim.

```

Quadro 53: Pseudocódigo – Exercício 1

2. Elabore um algoritmo que leia cem números inteiros e conte quantos são pares e quantos são ímpares.

**Objetivo do algoritmo:** ler cem números e contar os pares e ímpares.

**Entrada:** ler cem números inteiros.

**Processamento:** verificar se o número é par ou ímpar e contar a quantidade de pares e ímpares.

**Saída:** imprimir o número de pares e ímpares.

```

Algoritmo conta
Var
    num, npares, nimpares, i: inteiro

Início
    npares 0
    nimpares 0
    Para i de 1 até 100 passo 1 faça
        Escreva ("Digite um número:")
        Leia (num)

```

## ATIVIDADES



```
Se (num mod 2 = 0) então
    npar npar + 1
Senão
    nimpar nimpar + 1
Fim_se
fim_para
Escreva ("A quantidade de números pares é:",
npar)
Escreva ("A quantidade de números ímpares
é:", nimpar)
Fim.
```

Quadro 54: Pseudocódigo – Exercício 2

3. Construa um algoritmo que entre com números inteiros enquanto forem positivos e imprima quantos números foram digitados.

**Objetivo do algoritmo:** ler vários números enquanto forem positivos e contar quantos foram digitados.

**Entrada:** ler números enquanto forem positivos.

**Processamento:** contar a quantidade de números digitada.

**Saída:** imprimir a quantidade de números positivos digitadas.

```
Algoritmo conta
Var
    num, qtdade: inteiro
Início
    qtdade 0
    Escreva ("Informe um número:")
    Leia (num)
    Enquanto (num > 0) faça
        qtdade qtdade + 1
        Escreva ("Informe um número:")
        Leia (num)
```

## ATIVIDADES



**Fim\_enquanto**

**Escreva** ("O total de números positivos informado é:", qtdade)

**Fim.**

Quadro 55: Pseudocódigo – Exercício 3

4. Escreva um algoritmo que leia um conjunto de pedidos e calcule o total da compra. O pedido possui os seguintes campos: número, data (dia, mês e ano), preço unitário e quantidade. A entrada de pedidos é encerrada quando o usuário informa zero como número do pedido.

**Objetivo do algoritmo:** ler vários pedidos e calcular o total da compra.

**Entrada:** ler pedidos de compra (número, data, preço quantidade) até que o número do pedido seja zero.

**Processamento:** calcular o preço total de cada pedido e o preço total da compra.

**Saída:** imprimir o valor total da compra.

**Algoritmo** compras

**Var**

pedido, qtdade, data: inteiro

preco, total : real

**Início**

total  $\leftarrow$  0

**Escreva** ("Informe o número do pedido:")

**Leia** (pedido)

**Enquanto** (num > 0) **faça**

**Escreva** ("Informe a data:")

**Leia** (data)

**Escreva** ("Informe a unitário:")

**Leia** (preco)

**Escreva** ("Informe a quantidade:")

**Leia** (qtdade)

## ATIVIDADES



```
total ← total + (preco*qtidade)
Escreva ("O valor da compra é:", pre
co*qtidade)
Escreva ("Informe o número do pedido
ou 0 para finalizar:")
Leia (pedido)
Fim-enquanto
Escreva ("O total da compra é:", total)
Fim.
```

Quadro 56: Pseudocódigo – Exercício 4

5. Construa um algoritmo que leia nome, sexo, idade, peso e altura dos atletas que participam de um campeonato até que seja informado o nome "FIM" e apresente: o nome do atleta do sexo masculino mais alto, o nome da atleta do sexo feminino mais pesada e a média de idade dos atletas.

**Objetivo do algoritmo:** ler informações sobre vários atletas e apresentar o atleta mais alto, mais pesado e média de idade.

**Entrada:** ler nome, sexo, idade, peso e altura até que seja digitado o nome "FIM".

**Processamento:** identificar o atleta do sexo masculino mais alto, a atleta do sexo feminino mais pesada e a média de idade dos atletas.

**Saída:** imprimir o nome do atleta mais alto e da atleta mais pesada e a média de idade.

```
Algoritmo atletas
Var
    idade, cont: inteiro
    peso, altura, media, alto, pesado, soma: real
    nome, nalto, npesado: caractere[30]
    sexo: caractere[1]
Início
    media ← 0
    cont ← 0
    alto ← 0
```



## ATIVIDADES



```

pesado ← 0
nalto ← " "
npesado ← " "

Escreva ("Informe o nome do atleta ou FIM
para encerrar:")
Leia (nome)
Enquanto (num <> "FIM") faça
    Escreva ("Informe a idade:")
    Leia (idade)
    Escreva ("Informe o peso:")
    Leia (peso)
    Escreva ("Informe a altura:")
    Leia (altura)
    Escreva ("Informe a sexo:")
    Leia (sexo)
    soma ← soma+idade
    cont ← cont+1

se (sexo = "M") ou (sexo = "m") então
    se (altura>alto) então
        alto ← altura
        nalto ← nome

    Fim-se

Fim-se

se (sexo = "F") ou (sexo = "f") então
    se (peso>pesado) então
        pesadop ← peso
        npesado ← nome

    Fim-se

Fim-se

```

## ATIVIDADES



**Escreva** ("Informe o nome do atleta ou FIM para encerrar:")

**Leia** (nome)

**Fim\_enquanto**

**Escreva** ("O nome do atleta mais alto é:", nalto)

**Escreva** ("O nome da atleta mais pesada é:", npesado)

**Fim**

Quadro 57: Pseudocódigo – Exercício 5

6. Faça um algoritmo que calcula a área de um triângulo e que não permita a entrada de dados inválidos, ou seja, as medidas devem ser maiores ou iguais a zero.

**Objetivo do algoritmo:** calcular a área de um triângulo.

**Entrada:** ler a base e a altura do triângulo.

**Processamento:** calcular a área.

**Saída:** imprimir o valor da área.

```
Algoritmo triangulo
Var
    base, altura, area: real
Inicial
Repita
    Escreva ("Informe a base:")
    Leia (base)
Até_que (base>0)
Repita
    Até_que (altura >=0)
    area ← (base*altura)/2
    Escreva ("A área é:", area)
Fim
```

Quadro 58: Pseudocódigo – Exercício 6

## ATIVIDADES



7. Construa um algoritmo que receba a idade e o estado civil de várias pessoas e imprima a quantidade de pessoas casadas, solteiras, separadas e viúvas. O algoritmo finaliza quando for informado o valor zero para idade.

**Objetivo do algoritmo:** ler idade e estado civil e contabilizar a quantidade de pessoas por estado civil.

**Entrada:** ler idade e estado civil de várias pessoas.

**Processamento:** contabilizar o número de pessoas casadas, solteiras, separadas e viúvas.

**Saída:** imprimir a quantidade de pessoas casadas, solteiras, separadas e viúvas.

```

Algoritmo pessoas
Var
    idade, ncasado, nseparado, nsolteiro, nviuva
: inteiro
    estado : caractere
Início
    ncasado 0
    nsolteiro 0
    nviuva 0
    nseparado 0
Repita
    Escreva ("Informe a idade:")
    Leia (idade)
    Até_que (idade >= 0)
    Enquanto (idade <> 0) faça
Repita
    Escreva ("Informe o estado civil:")
    Leia (estado)
    Até_que ((estado = "C") ou (estado = "D") ou
    (estado = "S") ou (estado = "V"))
    Se (estado = "C") então
        ncasado ncasado + 1
    fim_se

```



```

Se (estado = "S") então
    nsolteiro nsolteiro + 1
fim_se
    Se (estado = "D") então
        nseparado nseparado + 1
    fim_se
    Se (estado = "V") então
        nviuva nviuva + 1
    fim_se
    Repita
        Escreva ("Informe a idade:")
        Leia (idade)
        Até_que (idade >= 0)
    fim_enquanto
    Escreva ("O número de casados é:", ncasados)
    Escreva ("O número de solteiros é:", nsolteiro)
    Escreva ("O número de separados é:", nseparado)
    Escreva ("O número de viúvas é:", nviuva)
Fim.

```



### **Algoritmos: Programação para Iniciantes**

Gilvan Vilarim

**Editora:** Ciência Moderna

**Sinopse:** Este livro tem por objetivo apresentar os fundamentos da lógica para a programação de computadores, capacitando o leitor a construir algoritmos estruturados, e traz assuntos e exercícios baseados em experiências reais do autor vivenciadas em sala de aula. A leitura não exige maior conhecimento do assunto por parte do leitor, mas apenas familiaridade com a operação do microcomputador.

Voltado para estudantes dos cursos técnicos introdutórios em programação de computadores, aplicados nas áreas de Informática, Ciência da Computação, Engenharias, técnico profissionalizante de nível médio e outras, o livro apresenta como principais características uma nova linguagem estruturada para a construção de algoritmos, texto renovado e atual, fortemente baseado na didática utilizada em aulas de programação, intensa carga de exercícios resolvidos e problemas propostos, além de curiosidades sobre computadores, algoritmos e programação, estimulando o interesse do leitor pelo assunto.





# ESTRUTURAS DE DADOS HOMOGÊNEAS E HETEROGÊNEAS

UNIDADE

## IV

### Objetivos de Aprendizagem

- Estudar as estruturas de dados homogêneas (vetores e matrizes).
- Conhecer métodos de ordenação e pesquisa.
- Conhecer estruturas de dados heterogêneas (registros).
- Construir algoritmos utilizando estruturas de dados homogêneas e heterogêneas.

### Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Estrutura de Dados Homogênea
- Vetores
- Ordenação de Vetor
- Busca em Vetor
- Método Sequencial
- Matrizes
- Estrutura de Dados Heterogênea
- Registros
- Problema





## INTRODUÇÃO

Estamos chegando ao final do conteúdo da disciplina de Algoritmos e Lógica de Programação I. Nesta unidade você conhecerá as estruturas de dados homogêneas e heterogêneas. Essas estruturas permitem agrupar diversas informações em uma única variável.

As estruturas de dados homogêneas permitem a representação de diversas informações do mesmo tipo, sendo divididas em unidimensionais (vetores) e multidimensionais (matrizes). Estudaremos como atribuir valores, realizar a leitura e entrada de dados nesse tipo de estrutura. Além disso, conheceremos um método para a classificação (ordenação segundo algum critério) e busca em vetores, especificamente, ordenação com *Bubble Sort* e pesquisa sequencial.

Estudaremos as estruturas de dados heterogêneas, registros, que agregam informações de diversos tipos. Abordando, especificamente, como realizar atribuição, entrada e saída de dados utilizando registros.

Na construção de algoritmos utilizando estruturas homogêneas e heterogêneas, utilizaremos todos os conceitos vistos nas demais unidades, principalmente as estruturas de repetição, pois elas são utilizadas nas operações de atribuição, leitura e escrita.

Ao final desta unidade você saberá construir algoritmos utilizando as estruturas de dados homogêneas e heterogêneas, as quais permitem representar um agrupamento de informações em uma mesma variável. Entenderá, também, qual a importância dessas estruturas e como utilizá-las em aplicações práticas, aumentando assim a gama de problemas que consegue resolver utilizando algoritmos. Vamos lá?

## ESTUTURAS DE DADOS HOMOGÊNEAS

Estruturas de dados homogêneas são estruturas que agrupam diversas informações, do mesmo tipo, em uma única variável (FARRER, 1989). Essas estruturas homogêneas são divididas em estruturas unidimensionais (vetores) e multidimensionais (matrizes), as quais estudaremos a seguir.

### VETORES

As estruturas unidimensionais, conhecidas como vetores, consistem em um arranjo de elementos armazenados na memória principal, sequencialmente, todos com o mesmo nome (LOPES; GARCIA, 2002).

O vetor segue o mesmo princípio de uma matriz linha da matemática, como ilustra a Figura 13. Em cada coluna da linha temos uma variável com mesmo nome, no entanto em diferentes posições no arranjo. A posição de cada variável no arranjo é indicada por um índice. No caso apresentado, temos 5 posições de modo que na posição 1 temos o valor 4, na posição 2 o valor 7, na posição 3 o valor 10, na posição 4 o valor 1 e na posição 5 o valor 5.



Figura 13: Vetor

Fonte: adaptado de LOPES; GARCIA (2002)

O uso de vetores está associado à criação de tabelas, em que temos a definição de uma única variável que é dimensionada com um determinado tamanho. Sendo que dimensão deve ser uma constante inteira e positiva (MANZANO; OLIVEIRA, 1997).

A declaração de um vetor é realizada na seção de variáveis (Var) do seguinte modo:

**Variavel: vetor** [<dimensão>] **de** <tipo de dado>

Em que:

- <dimensão> representa os valores de início e fim do tamanho do vetor
- <tipo de dado> pode ser real, inteiro, lógico ou caractere.

Um exemplo de declaração de vetor é:

Media: **vetor** [1..5] **de** real

Em que o nome da variável é media, sendo que esta possui cinco posições e é do tipo real. O acesso a cada posição do vetor é realizado por meio da especificação do índice. Isto é media[1] indica o elemento que está na posição 1 do vetor, que no caso da Figura 13 é o valor 4.

As operações de atribuição, leitura e escrita são realizadas para cada elemento do vetor, de modo que não é possível operar sobre o vetor como um todo. Deste modo, ao utilizar os comandos de atribuição, leitura e escrita devemos utilizar o nome da variável mais o índice. Note que, como as operações devem ser realizadas elemento a elemento temos que utilizar uma estrutura de repetição para a entrada e saída de dados. Das estruturas de repetição vistas na Unidade III, qual a mais adequada? Como um vetor possui um tamanho fixo, informado no momento de sua declaração, sabemos previamente o número de repetições que precisa ser realizado. Portanto, a estrutura **Para** é adequada.

Se você não entendeu como funciona um vetor, fique tranquilo!! Construiremos um algoritmo para aplicar os conceitos estudados até aqui.

O problema consiste em ler o nome e quatro notas de 10 alunos e imprimir uma relação contendo o nome do aluno, notas e a média final.

Seguindo o método de estruturação de problemas, temos que:

- **Objetivo do algoritmo:** emitir uma relação contendo nome dos alunos, notas e média final.
- **Entrada:** ler nome e quatro notas de dez alunos.
- **Processamento:** calcular a média dos alunos.
- **Saída:** imprimir nome dos alunos, notas e média final.

Na entrada de dados temos que efetuar a leitura do nome do aluno e quatro notas. Observe que a leitura tem que ser realizada para dez alunos. Sem utilizar a estrutura de vetor precisaríamos de dez variáveis para armazenar nome, cinquenta variáveis para armazenar as quatro notas dos dez alunos e dez variáveis para armazenar a média. Com o uso de vetores precisamos declarar uma variável para armazenar nome, quatro variáveis para armazenar as notas e uma para armazenar a média. O tipo de dado utilizado para armazenar o nome é caractere e para as notas e médias utilizamos o real. Podemos utilizar inteiro? Não, pois tanto para nota quanto para média são admitidos valores decimais. A entrada de dados é realizada por meio do comando `leia`, no entanto, devemos lembrar que não é possível obter todos os valores do vetor de uma única vez, precisamos ler elemento a elemento. O processamento consiste em calcular a média das quatro notas para cada aluno e armazenar na variável `media`. Na saída de dados temos que informar o nome, notas e médias dos dez alunos. Para isso, utilizaremos o comando `escreva` em uma estrutura de repetição, pois temos que escrever elemento a elemento do vetor. No Quadro 60 é apresentado o algoritmo para o problema descrito.

**Algoritmo** media

**Var**

```
nome: vetor [1..10] de caractere
n1, n2, n3, n4, media : vetor[1..10] de real
cont: inteiro
```

**Início**

**Para** cont de 1 até 10 **passo** 1 **faça**

**Escreva** ("Informe nome do aluno:")

**Leia** (nome[cont])

**Escreva** ("Informe a nota 1:")

**Leia** (n1[cont])

**Escreva** ("Informe a nota 2:")

**Leia** (n2[cont])

```

Escreva ("Informe a nota 3:")
Leia (n3[cont])
Escreva ("Informe a nota 4:")
Leia (n4[cont])
media[cont] (n1[cont] + n2[cont]+n3
[cont] + n4[cont])/4

```

**fim\_para**

**Para** cont **de** 1 **até** 10 **passo** 1 **faça**

```

Escreva ("Aluno", nome[cont])
Escreva ("Nota1:", n1[cont])
Escreva ("Nota2:", n2[cont])
Escreva ("Nota3:", n3[cont])
Escreva ("Nota4:", n4[cont])
Escreva ("Media:", media[cont])

```

**Fim\_para**

**Fim**

Quadro 60: Pseudocódigo – Algoritmo média notas

Observe que nos comandos de entrada, saída e atribuição sempre utilizamos o nome da variável seguido do índice na posição do vetor. Se você ficou com dúvidas quanto ao funcionamento de vetores elabore o teste de mesa para o algoritmo apresentado.

Fique tranquilo!!! Na próxima seção construiremos mais algoritmos utilizando vetores.

## ORDENAÇÃO EM VETOR

No dia a dia nos deparamos com uma série de situações em que independentemente do modo que os dados foram informados, precisamos apresentá-los seguindo uma ordem. No caso de variáveis do tipo caractere temos que classificá-los em ordem alfabética (A-Z) para facilitar a localização de um nome, por exemplo. A ordenação crescente ou decrescente pode ser utilizada, também, para variáveis numéricas (MANZANO; OLIVEIRA, 1997).

A ordenação é o processo de rearranjar os elementos de acordo com um critério específico com o objetivo de facilitar a localização (WIRTH, 1999). Na literatura existem diversos métodos de ordenação, sendo o método da bolha (*Bubble Sort*) o mais conhecido.

O método da bolha não é o método mais eficiente, mas é bastante simples. Portanto, veremos seu funcionamento. O método consiste em percorrer o vetor repetidas vezes, comparando os elementos vizinhos. Se eles estão fora de ordem, é efetuada uma troca de posição. O número de repetições de “varredura” é dado pelo número de elementos do vetor menos 1. Vamos tomar como exemplo a ordenação (crescente) do vetor apresentado na Figura 14. Primeiramente, veremos o funcionamento do método passo a passo e, em seguida, a implementação do algoritmo. Como podemos observar o vetor não está ordenado.

4	7	10	1	5
---	---	----	---	---

Figura 14: Vetor inicial

Ao iniciar a ordenação estamos na posição 1 (Figura 15), temos que comparar o elemento da posição 1 com o elemento da posição 2 (Figura 16). Como 4 não é maior que 7 não efetuamos a troca.


				
4	7	10	1	5
1	<input type="checkbox"/>	<input type="checkbox"/>	4	5

Figura 15: Primeiro repetição

**4 é maior que 7?**



Figura 16: Comparação entre os elementos

Agora temos que comparar o elemento da posição 1 com o elemento da posição 3 (Figura 17). Como 4 não é maior que 10, não há troca. Agora temos que comparar o elemento da posição 1 com o elemento da posição 4 (Figura 18).

**4 é maior que 10?**



Figura 17: Comparação entre os elementos

Como 4 é maior do que 1, temos que trocar os elementos de posição, isto é o 1 vai para a posição 1 e o 4 vai para a posição 4. Ainda temos que comparar o elemento da posição 1 com o elemento da posição 5 (Figura 19).

**4 é maior que 1?**



Figura 18: Comparação entre os elementos

Como 1 não é maior que 5, a troca não é efetuada.

**4 é maior que 5?**



Figura 19: Comparação entre os elementos

Temos que repetir o processo de comparação, agora comparando o elemento da posição 2 com os demais elementos (Figura 20).



Figura 20: Segunda repetição

Comparando o elemento da posição 2 com o elemento da posição 3 (Figura 21), não há troca, pois 7 não é maior que 10. Vamos então comparar o elemento da posição 2 com o elemento da posição 4 (Figura 21).

**7 é maior que 10?**



Figura 21: Comparação entre os elementos

Como 7 é maior do que 4, há a troca de posição dos elementos (Figura 23).

**4 é maior que 4?**



Figura 22: Comparação entre os elementos

Prosseguindo com a comparação, agora temos que analisar o elemento da posição 2 e o da posição 5 (Figura 24).



Figura 23: Ordenação do vetor

**4 é maior que 5?**



Figura 24: Comparação entre os elementos

Como 4 não é maior que 5, não há troca. Agora devemos repetir a comparação passando para a posição 3 (Figura 25).



Figura 25: Terceira repetição

Comparando o elemento da posição 3 com o elemento da posição 4 (Figura 26), temos que 10 é maior que 7. Portanto, há troca (Figura 27).



**10 é maior que 7?**



Figura 26: Comparação entre os elementos



Figura 27: Ordenação do vetor

Ainda temos que comparar o elemento da posição 3 com o da posição 5 (Figura 28).

**7 é maior que 5?**



Figura 28: Comparação entre os elementos

Comparando 7 e 5, temos que 7 é maior que 5. Portanto, os elementos trocam de posição (Figura 29).



Figura 29: Ordenação do vetor

Repetindo o processo de comparações, iniciando na quarta posição, conforme ilustra a Figura 30.



Figura 30: Quarta repetição

Comparando o elemento da posição 4 com o da posição 5 (Figura 30), temos que 10 é maior que 7. Os elementos trocam de posição e como não há mais comparações, temos o vetor ordenado (Figura 32).

**10 é maior que 7?**



Figura 31: Comparação entre os elementos

1	4	5	7	10
---	---	---	---	----

Figura 32: Vetor ordenado

Ao final das comparações temos o vetor ordenado. O processo de percorrer o vetor realizando comparações foi repetido quatro vezes, ou seja, o número de elementos do vetor menos 1. De modo que na primeira repetição foram realizadas quatro comparações, na segunda três, na terceira duas e na última uma. O algoritmo que lê 5 valores e os apresenta em ordem crescente (utilizando o método da bolha) é apresentado no Quadro 61.

```
Algoritmo ordena
Var
    num: vetor[1..5] de inteiro
    i, j, aux: inteiro
Início
    Para i de 1 até 5 passo 1 faça
        Escreva ("Informe um número:")
        Leia (num[i])
    Fim_para
    Para i de 1 até 4 passo 1 faça
        Para j de i+1 até 5 faça
            Se (num[i] > num[j]) então
                uax num[i]
                num[i] num[j]
                num[j] aux
            Fim_se
        fim_para
    fim_para
    Para i de 1 até 5 passo 1 faça
        Escreva (num[i])
    Fim_para
Fim.
```

Quadro 61: Pseudocódigo – Algoritmo ordenação

No primeiro laço de repetição temos a leitura de cinco números, os quais são armazenados no vetor `num`. Após a entrada de dados temos o encadeamento de estruturas de repetição **Para**. O primeiro **Para** é responsável por percorrer o vetor da posição 1 até a posição 4 (número de elementos menos 1) e o segundo **Para** é que percorre os elementos da posição  $i+1$  (vizinho) até a posição 5 (número total de elementos). No interior da estrutura **Para** ocorre a comparação entre os números se o número é maior que o vizinho eles trocam de posição. Observe que neste ponto utilizamos uma variável denominada `aux`. **Para** que serve esta variável? Não é possível efetuar a troca entre dois elementos sem utilizar uma variável auxiliar para armazenar o valor. Se fizemos `num[i] ← num[j]` e `num[j] ← num[i]`, o que teríamos como resultado? Nos duas posições de memória teríamos o valor da variável `num[j]`. Sempre que for necessário efetuar a troca ou permutação entre elementos precisamos de uma estrutura auxiliar, para armazenar temporariamente o valor de uma das variáveis. Ao final da execução das duas estruturas de repetição temos o vetor ordenado. Em seguida, é realizada a saída de dados, também, utilizando uma estrutura de repetição, em que os valores já ordenados são impressos.

Se você ainda ficou com dúvidas quanto ao funcionamento da ordenação, faça o teste de mesa do algoritmo utilizando o vetor da Figura 33. Lembre-se não há aprendizado de algoritmos sem prática.

11	3	8	1	15
----	---	---	---	----

Figura 33: Vetor

## BUSCA EM VETOR

Há situações em que precisamos buscar por um determinado elemento em um vetor. Se considerarmos um vetor com cinco elementos é fácil realizar essa busca de forma manual. No entanto, nos casos em que temos muitos elementos este processo manual é inviável (MANZANO; OLIVEIRA, 1997).

Existem métodos que permitem verificar a existência de um valor dentro de um vetor, isto é, procurar dentre os elementos um determinado valor. Estudaremos o método de busca sequencial.

## MÉTODO SEQUENCIAL

A busca utilizando o método sequencial consiste em percorrer o vetor a partir do primeiro elemento, sequencialmente, até o último realizando testes lógicos verificando se o elemento do vetor, posição a posição, é igual ao elemento procurado. Neste método, também conhecido como busca linear, a busca termina quando uma das duas condições for satisfeita: o elemento foi encontrado ou todo o vetor foi analisado, mas o elemento não foi encontrado (WIRTH, 1999).

O algoritmo que realiza a busca sequencial de um número qualquer em um vetor é apresentado no Quadro 62.

```
Algoritmo busca
    Var
        vnum: vetor[1..20] de inteiro
        num: inteiro
        acha: lógico
    Início
        Para i de 1 até 20 passo 1 faça
            Escreva ("Informe um número:")
```

```

        Leia (vnum[i])

    Fim_para

    Escreva ("Informe o número que deseja buscar:")
    Leia (num)
    i ← 1
    acha falso

    Enquanto (i <= 20) e (acha = falso)
        faça
            se (num = vnum[i]) então
                acha verdadeiro
            senão
                i ← i + 1

        fim_se
    fim_enquanto

    Se (acha = verdadeiro) então
        Escreva ("O elemento foi encontrado na
        posição", i)
    Senão
        Escreva ("O elemento não foi encon
        trado")

    Fim_se

Fim.

```

Quadro 62: Pseudocódigo – Algoritmo Busca Sequencial

Inicialmente são obtidos os valores e armazenados no vetor **vnum**. Como utilizamos uma estrutura de repetição que realiza o teste lógico no início, o **Enquanto**, temos que inicializar as variáveis utilizadas na condição. A estrutura **Enquanto** percorre o vetor, comparando cada elemento como o número pesquisado. Por fim, temos um teste condicional que verifica se o número foi encontrado no vetor.

## MATRIZES

Uma matriz é uma variável homogênea multidimensional, formada por uma sequência de variáveis do mesmo tipo, com o mesmo nome e alocadas sequencialmente na memória. Para acessar cada elemento da matriz são utilizados índices, sendo que para cada dimensão devemos ter um índice.

A Figura 34 apresenta um exemplo de uma matriz bidimensional, com três linhas e quatro colunas.

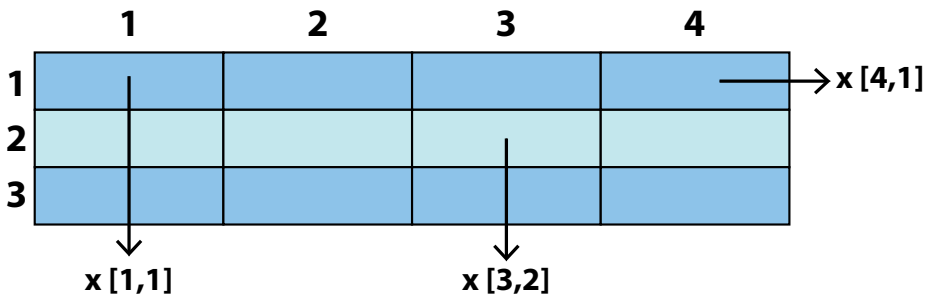


Figura 34: Matriz

Fonte: adaptado de (ASCENCIO; CAMPOS, 2010)

Da mesma forma que os vetores, a dimensão da matriz é formada por constantes inteiras e positivas. E a nomeação da matriz segue as mesmas regras das variáveis. A declaração de um vetor é realizada na seção de variáveis (Var) do seguinte modo:

**Variavel: vetor** [<dimensão1>, <dimensão2>] **de** <tipo de dado>

Em que:

- <dimensão1> e <dimensão2> representam o tamanho da tabela
- <tipo de dado> pode ser real, inteiro, lógico ou caractere.

Um exemplo de declaração de matriz é:

Notas: **vetor** [1..8, 1..4] **de** real

Neste exemplo, temos uma matriz com oito linhas e quatro colunas (Figura 35). Isto é, na primeira dimensão representamos o número de linhas e na segunda o número de colunas.

Coluna  
↓

1                  2                  3                  4

Linha →

1				
2				
3				
4				
5				
6				
7				
8				

Figura 35: Exemplo de matriz

Em uma matriz nas operações de atribuição, leitura e escrita devemos utilizar o número de repetições relativo ao tamanho das dimensões. Com isto, uma matriz de duas dimensões deve ser controlada por dois laços de repetição, de três dimensões três laços e assim por diante (MANZANO; OLIVEIRA, 1997).

Com base nos conceitos sobre matrizes vamos escrever um algoritmo para o seguinte problema: efetuar a leitura de quatro notas de vinte alunos, calcular a média de cada aluno e a média da turma.

Sistematizando o problema temos que:

- **Objetivo do algoritmo:** calcular a média de cada aluno e a média da turma.
- **Entrada:** ler quatro notas de vinte alunos.
- **Processamento:** calcular a média das quatro notas dos alunos e a média da turma.
- **Saída:** imprimir a média de cada aluno e a média da turma.

Na entrada de dados temos que efetuar a leitura de quatro notas de vinte alunos. Para isso, utilizaremos uma matriz com vinte linhas e quatro colunas, ou seja, a primeira dimensão (linhas) representa o número de alunos e a segunda

dimensão (colunas) as notas. O tipo de dado utilizado para armazenar essas informações é real. Como temos uma matriz bidimensional utilizaremos dois laços de repetição para efetuar a operação de leitura.

O processamento consiste no cálculo da média das quatro notas para cada aluno, que armazenaremos em uma variável vetor de dimensão vinte. Já para o cálculo da média da turma utilizaremos uma variável para acumular os valores das médias de cada aluno e ao sair da estrutura de repetição calculamos a média. Na saída temos que informar a média de cada aluno e da turma. No Quadro 63 é apresentado o algoritmo para o problema descrito.

```
Algoritmo media
Var
    media: vetor [1..20] de real
    notas: vetor[1..20, 1..4] de real
    somat, soma, mediat: real
    i, j : inteiro

Início
    somat 0
    soma 0
    Para i de 1 até 20 passo 1 faça
        Para j de 1 até 4 passo 1 faça
            Escreva ("Informe a nota")

        fim_para
        mediat somat/20
        Para i de 1 até 20 passo 1 faça
            Escreva ("A média do aluno", i , "é :",
                media[i])

        Fim_para
        Escreva ("A média da turma é:", mediat)

Fim.
```

Quadro 63: Pseudocódigo – Algoritmo média notas



Observe que nos comandos de entrada, saída e atribuição sempre utilizamos o nome da variável seguida dos índices que indicam a linha e a coluna da matriz, respectivamente. Como a matriz é bidimensional a leitura foi realizada dentro de um encadeamento da estrutura **Para**. A primeira estrutura **Para** percorre as linhas (alunos) e a segunda a coluna (notas). No interior do encadeamento de estrutura **Para** temos a leitura das notas de cada aluno e acumulamos a soma das quatro notas do aluno. Ao sair da primeira estrutura **Para** é calculada a média para cada aluno, que é armazenada no vetor **media** e armazenamos o valor individual da média de cada aluno na variável **somat**, que utilizamos no cálculo da média da turma. Ao sair da estrutura **Para** externa, após ter percorrido todos os alunos, calculamos a média geral da turma. E na saída de dados, temos a impressão do vetor **media** dentro de uma estrutura **Para**. Fora do laço de repetição escrevemos o valor da média geral da turma, armazenado na variável **mediat**.

Se você ficou com alguma dúvida, elabore o teste de mesa. Bom trabalho!!

## ESTRUTURAS DE DADOS HETEROGÊNEAS

As estruturas de dados heterogêneas permitem o agrupamento de informações de diferentes tipos de dados, sendo denominadas de registros.

## REGISTROS

Os registros são estruturas de dados que agregam diversas informações, que podem ser de diferentes tipos. Com essa estrutura é possível gerar novos tipos de dados, além dos definidos pelas linguagens de programação (ASCENCIO; CAMPOS, 2010).

Em um registro cada informação é denominada de campo, os quais podem ser de diferentes tipos, ou ainda, representar outros registros.

A declaração de uma variável registro deve ocorrer antes das variáveis, pois pode ocorrer a necessidade de declarar uma variável com tipo registro. A sintaxe para declaração é dada por (MANZANO; OLIVEIRA, 1997):

**Tipo**

```
<identificador> = registro  
    <lista dos campos e seus tipos>  
  
fim_registro  
  
var  
  
<variáveis> : <identificador>
```

Em que:

- <identificador>: nome do tipo registro.
- <lista dos campos e seus tipos>: relação de variáveis que serão usadas como campos e o tipo de cada uma delas.

Note que na seção **Var** devemos indicar a variável como tipo registro e a declaração do seu tipo de acordo com o identificador definido anteriormente. LEMBRE-SE: a instrução tipo deve vir antes da instrução **Var**.

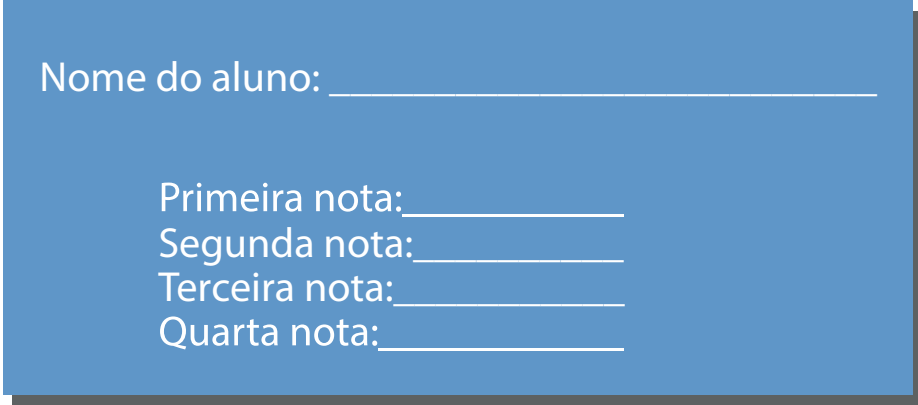
As operações de atribuição, leitura e escrita são realizadas utilizando o nome da variável registro e seu campo correspondente separado por um caractere “.” (ponto) (MANZANO; OLIVEIRA, 1997).

Já conhecemos os conceitos e sintaxe de registros, mas qual a estrutura de um registro e como utilizá-los?

Vamos retomar o problema visto na seção “Vetores”, que consistia em ler o nome e quatro notas de 10 alunos e imprimir uma relação contendo o nome do aluno, notas e a média final. Na resolução deste problema aplicamos o conceito de vetores e utilizamos cinco vetores, um para armazenar o nome, um para cada uma das notas e outro para armazenar a média das notas. Há uma forma mais fácil de resolver esse problema? Sim. Sabemos que um registro pode conter vários tipos de dados em uma mesma estrutura, então fica mais fácil agrupar os dois tipos de dados (caractere e real) em uma mesma estrutura.

A Figura 36 ilustra o *layout* de um registro, contendo os campos: nome,

primeira nota, segunda nota, terceira nota e quarta nota.

A blue rectangular form with a dark blue border. It contains four lines of text, each followed by a horizontal line for input. The text is white and left-aligned.

Nome do aluno: \_\_\_\_\_

Primeira nota: \_\_\_\_\_

Segunda nota: \_\_\_\_\_

Terceira nota: \_\_\_\_\_

Quarta nota: \_\_\_\_\_

Figura 36: Registro aluno

Fonte: adaptado de (MANZANO; OLIVEIRA, 1997)

Um algoritmo que realiza a leitura de nome e quatro notas de um aluno utilizando o conceito de registros é apresentado no Quadro 64. Com este exemplo fica mais claro como deve ser realizada a declaração, atribuição, leitura e escrita utilizando registros.

### Quadro 64: Pseudocódigo – Algoritmo registro aluno

```
Algoritmo registroaluno

Tipo

    Cad_aluno = registro
    Nome: caractere
    Nota1: real
    Nota2: real
    Nota3: real
    Nota4: real
    fim_registro

Var

    Aluno: cad_aluno

Início

    Escreva ("Informe nome do aluno:")
    Leia (aluno.nome)
    Escreva ("Informe a nota 1:")
    Leia (aluno.nota1)
    Escreva ("Informe a nota 2:")
    Leia (aluno.nota2)
    Escreva ("Informe a nota 3:")
    Leia (aluno.nota3)
    Escreva ("Informe a nota 4:")
    Leia (aluno.nota4)

Fim.
```

Quadro 64: Pseudocódigo – Algoritmo registro aluno

Note que utilizando registros precisamos declarar apenas uma variável que é do tipo `cad_aluno` (identificador do registro). O registro possui cinco campos, sendo eles: `nome`, `nota1`, `nota2`, `nota3` e `nota4`. Para acessar cada um desses campos utilizamos o nome da variável que declaramos como registro seguida de ponto e o nome do campo. Por exemplo, `aluno.nome`.

Você imagina outras situações que podemos aplicar o conceito de registros? No dia a dia nos deparamos com diversos casos. Se pensarmos em uma empresa temos cadastros com informações sobre clientes, produtos e serviços. Podemos agrupar as informações relacionadas a cada um destes elementos em registros, facilitando o acesso às informações relacionadas a um cliente, produto ou serviço.

Ao criar registros podemos definir um vetor ou até mesmo uma matriz dentro de um registro. O algoritmo do Quadro 65 apresenta um registro utilizando vetores para o problema de ler o nome e as quatro notas de um aluno.

```

Algoritmo registroaluno
Tipo
    notas = vetor [1..4] de real
    Cad_aluno = registro
    Nome: caractere
    Nota: notas
fim_registro
Var
    Aluno: cad_aluno
    i: inteiro
Início
    Escreva ("Informe nome do aluno:")
    Leia (aluno.nome)
    Para i de 1 até 4 passo 1 faça
        Escreva ("Informe a nota", i, ":")
        Leia (aluno.nota[i])
    fim_para
Fim.

```

Quadro 65: Pseudocódigo – Algoritmo registro aluno

No exemplo apresentado na seção “Vetores” tínhamos a leitura de nome e notas para dez alunos. Podemos resolver o mesmo problema utilizando registros? Sim, pois podemos declarar vetores de registros, conforme apresentado no Quadro 66.

```
Algoritmo registroaluno
Tipo
    notas = vetor [1..4] de real
    Cad_aluno = registro
    Nome: caractere
    Nota: notas
fim_registro
Var
    Aluno: vetor [1..10] de cad_aluno
    i, j: inteiro
Início
    Para i de 1 até 10 faça
        Escreva ("Informe nome do aluno:")
        Leia (aluno[i].nome)
        Para j de 1 até 4 passo 1 faça
            Escreva ("Informe a nota", j,
                ":")
            Leia (aluno[i].nota[j])
        Fim_para
    Fim_para
    Para i de 1 até 10 faça
        Escreva ("Aluno:", aluno[i].nota[j])
        Para j de 1 até 4 passo 1 faça
            Escreva ("Nota", j, ":", alu
                no[i].nota[j])
        Fim_para
    Fim_para
Fim.
```

Quadro 66: Pseudocódigo – Algoritmo registro aluno

Se você ficou com dúvidas, não se preocupe!! Vamos escrever mais um algoritmo aplicando os conceitos aprendidos nesta Unidade na próxima seção.

## PROBLEMA

O problema consiste em ler informações (código, descrição, preço e quantidade em estoque) de 20 produtos. Além disso, deve ser permitido executar quantas consultas o operador desejar, em que ele digita o código do produto e é apresentado o preço e o saldo em estoque do produto. Se o código digitado não existir, informar o usuário.



Sistematizando o problema temos que:

- **Objetivo do algoritmo:** ler informações sobre 20 produtos e realizar consulta pelo código do produto.
- **Entrada:** ler código, descrição, preço e quantidade em estoque de 20 produtos e o código para pesquisa.
- **Processamento:** encontrar o produto com código informado pelo usuário.
- **Saída:** imprimir o preço e o saldo em estoque do produto, se existir o código informado. Senão informar ao usuário que não há produto cadastrado com o código digitado.

O Quadro 67 apresenta o algoritmo para o problema descrito. Observe que este algoritmo envolve o conceito de registros, laço de repetição e busca em vetor.

```

Algoritmo registroproduto
Tipo
    Cad_produto = registro
    codigo: inteiro
    nome: caractere[30]
    preço: real
    saldo: inteiro
fim_registro
Var
    produto: vetor [1..10] de cad_produto
  
```

```
i, codigo: inteiro
acha: lógico
resp: caractere[3]

Início

Para i de 1 até 20 faça
    Escreva ("Informe código do produto:")
    Leia (produto[i].codigo)
    Escreva ("Informe a descrição do produto:")
    Leia (produto[i].nome)
    Escreva ("Informe o preço do produto:")
    Leia (produto[i].preco)
    Escreva ("Informe o saldo em estoque do produto:")
    Leia (produto[i].saldo)

Fim_para
Resp "sim"

    Enquanto (resp = "sim") faça
        Escreva ("Informe o código a ser pesquisado:")
        Leia (codigo)
        i ← 1
    acha falso

Enquanto (i <= 20) e (acha= falso) faça
    Se (produto[i].codigo = código) então
        Acha ← verdadeiro

    Senão
        i ← i + 1

    fim_se

fim_enquanto

Se (acha=verdadeiro) então
    acha ← falso
```



```

Enquanto (i < = 20) e (acha= falso) faça
    Se (produto[i].codigo = código) então
        Acha verdadeiro
    Senão
        i ← i + 1
    fim_se
fim_enquanto
Se (acha=verdadeiro) então
    Escreva ("O preço é:", produto[i].preco)
    Escreva ("O saldo em estoque é:", produto[i].saldo)
Senão
    Escreva ("Não há produto com o código informado")
Fim_se
    Escreva ("Deseja continuar a pesquisa?")
    Leia (resp)
Fim_enquanto

Fim.

```

Quadro 67: Pseudocódigo – Algoritmo registro aluno

## CONSIDERAÇÕES FINAIS

Nesta unidade você conheceu as estruturas de dados homogêneas e heterogêneas. Nas estruturas de dados homogêneas estudamos os vetores e as matrizes.

Vimos que vetores e matrizes são estruturas homogêneas que agrupam diversas informações, do mesmo tipo, em uma única variável. Sendo os vetores unidimensionais e as matrizes multidimensionais. Em relação aos vetores conhecemos o método de classificação *Bubble Sort*, que varre o vetor repetidas vezes, comparando os elementos vizinhos. Se eles estão fora de ordem, é efetuada uma troca de posição. Esse método é utilizado quando queremos reorganizar o vetor segundo algum critério como, por exemplo, ordem crescente, decrescente ou alfabética. Ainda em relação a vetores conhecemos o método de pesquisa sequencial, que permite verificar se um dado elemento encontra-se no vetor ou não.

Estudamos as estruturas heterogêneas, aquelas que agregam diversas informações, que podem ser de diferentes tipos, sendo denominadas de registros. Este tipo de estrutura é bastante utilizado para representar cadastros, tais como: produtos, clientes, alunos, serviços e outros.

Na construção de algoritmos utilizando estruturas homogêneas e heterogêneas utilizamos todos os conceitos vistos nas demais unidades, revisando, principalmente as estruturas de repetição, pois elas são utilizadas nas operações de atribuição, leitura e escrita.

Por fim, elaboramos um algoritmo que envolveu todos os conceitos vistos na Unidade. Agora, para fixar todos os conceitos é importante que você desenvolva as atividades de autoestudo propostas. Assim, você assimila com mais facilidade tudo o que vimos, desenvolve seu raciocínio lógico e aumenta o poder de resolução de problemas, afinal, elaboramos algoritmos com o intuito de resolver problemas.

## REFLITA



A variável caractere é armazenada na memória principal como sendo um vetor, mesmo sem ser declarada como tal.



## NA WEB

**Para saber um pouco mais sobre o método de ordenação da bolha, acesse os vídeos disponíveis em:**

<<http://www.youtube.com/watch?v=AW5TlqCHV8U>>.

<<http://www.youtube.com/watch?v=otqltM-ou0o>>.

## ATIVIDADES



1. Elabore um algoritmo que leia um vetor de 50 números inteiros e imprima o maior e o menor número.
2. Escreva um algoritmo que leia dois vetores A e B, com 20 números inteiros. Efetue a soma dos dois vetores em um vetor C e imprima o vetor C em ordem crescente.
3. Construa um algoritmo que leia o preço de compra e o preço de venda de 30 produtos e imprima o número de mercadorias que apresenta lucro  $< 15\%$  e quantas apresentam lucro  $> 30\%$ .
4. Formule um algoritmo que leia uma matriz  $5 \times 5$  de números inteiros e imprima os elementos da diagonal principal.
5. Desenvolva um algoritmo que efetue a leitura dos nomes de 5 alunos e também de suas quatro notas bimestrais. Calcule a média de cada aluno e apresente os nomes classificados em ordem crescente de média.
6. Escreva um algoritmo que receba duas matrizes inteiras de ordem 5 e imprima a soma e a diferença entre elas.
7. Formule um algoritmo que receba os valores de contas a pagar de uma ao longo do ano, de modo que cada linha representa um mês do ano e cada coluna uma semana do mês. Com isso, temos uma matriz  $12 \times 4$ . Calcule e imprima: total de contas a pagar por mês e o total anual.
8. Escreva a estrutura de um registro para um cadastro de livros contendo as seguintes informações: título, autor, editora, edição e ano.
9. Elabore um cadastro para 15 professores, contendo as seguintes informações: matrícula, nome, formação e salário. Desenvolva um menu que:
  1. cadastre os professores;
  2. imprima o nome dos professores por ordem crescente de salário;
  3. pesquise um professor pela matrícula e informe nome e formação, caso exista;
  4. saia do programa.

## ATIVIDADES



### EXERCÍCIOS DE FIXAÇÃO

1. Escreva um algoritmo que leia um vetor com 30 elementos inteiros e escreva-os em ordem contrária a da leitura.

**Objetivo do algoritmo:** ler e imprimir um vetor com 30 elementos inteiros.

**Entrada:** ler um vetor com 30 números inteiros.

**Processamento:** não há.

**Saída:** imprimir os 30 elementos em ordem contrária a da leitura.

#### Quadro 68: Pseudocódigo – Exercício 1

```

Algoritmo levetor
Var
    num: vetor [1..30] de inteiro
    i, j                : inteiro
Início
    Para i de 1 até 30 passo 1 faça
        Escreva ("Informe um número:")
        Leia (num[i])
    fim_para
    Para i de 20 até 1 passo -1 faça
        Escreva (num[i])
    Fim_para
Fim.

```

#### Quadro 68: Pseudocódigo – Exercício 1

2. Elabore um algoritmo que leia duas matrizes de 5x5. Calcule e imprima a diferença entre elas.

**Objetivo do algoritmo:** ler duas matrizes e imprimir a diferença entre elas.

**Entrada:** ler duas matrizes 5 x 5.

## ATIVIDADES



**Processamento:** calcular a diferença entre as duas matrizes.

**Saída:** imprimir a matriz diferença.

```
Algoritmo diferenca
Var
    A, B, C : vetor[1..5, 1...5] de inteiro
    i, j : inteiro
Início
    Para i de 1 até 5 passo 1 faça
        Para j de 1 até 5 passo 1 faça
            Escreva ("Informe o elemento",
                i, j, "da matriz A")
            Leia (A[i,j])
        Fim_para
    fim_para
    Para i de 1 até 5 passo 1 faça
        Para j de 1 até 5 passo 1 faça
            Escreva ("Informe o elemento",
                i, j, "da matriz B ")
            Leia (B[i,j])
            C A[i,j] - B[ i,j]
        Fim_para
    fim_para
    Para i de 1 até 5 passo 1 faça
        Para j de 1 até 5 passo 1 faça
            Escreva (C[i,j])
        Fim_para
    fim_para
Fim.
```

## ATIVIDADES



3. Construa um algoritmo para uma agenda telefônica (nome, fone, e-mail) com 20 contatos. Além disso, deve ser permitido executar quantas consultas o operador desejar, em que ele digita o nome do contato e é apresentado o telefone. Se o nome digitado não existir, informar o usuário.

**Objetivo do algoritmo:** ler informações sobre 20 contatos e realizar consulta-zelo nome.

**Entrada:** ler nome, fone e e-mail de 20 contatos e o nome para pesquisa.

**Processamento:** encontrar o nome informado pelo usuário.

**Saída:** imprimir o telefone se existir o nome informado. Senão, informar ao usuário que não há contato cadastrado com o nome digitado.

```

Algoritmo registrocontato
Tipo
    Cad_agenda = registro
    nome: caractere[30]
    fone: caractere[15]
    e-mail: caractere[20]
Fim_registro
Var
    contato: vetor[1..20] de cad_agenda
    i: inteiro
    acha: lógico
    nomebusca: caractere[30]
    resp: caractere[3]
Início
    Para i de 1 até 20 faça
        Escreva ("Informe nome do contato:")
        Leia (contato[i].nome)
        Escreva ("Informe o telefone:")
        Leia (contato[i].fone)
        Escreva ("Informe o e-mail:")
        Leia (contato[i].email)

```

## ATIVIDADES



```
Fim_Para
Resp "sim"
Enquanto (resp= "sim") faça
    Escreva ("Informe o nome ser
    pesquisado:")
    Leia (nomebusca)
     $i \leftarrow 1$ 
     $acha \leftarrow \text{falso}$ 
    Enquanto ( $i \leq 20$ ) e ( $acha = \text{falso}$ ) faça
        se (contato[i].nome = nomebusca) então
             $acha \leftarrow \text{verdadeiro}$ 
        senão
             $i \leftarrow i + 1$ 
        fim_se
    Fim_Enquanto
    Se ( $acha = \text{verdadeiro}$ ) então
        Escreva ("O fone é:", contato[i].fone)
        senão
            escreva ("Não há contato com
            o nome informado")
    Fim_se
    Escreva ("Deseja contunuar a pesquisa?")
    Leia (nestp)
Fim_enquanto
Fim
```

Quadro 70: Pseudocódigo – Exercício 3

4. Elabore um algoritmo que leia informações (matrícula, nome, lotação e salário) de 30 professores. Deve ser permitido executar quantas consultas o operador desejar, em que ele digita a matrícula e é apresentado a lotação e o salário do professor. Se a matrícula digitada não existir, informar o usuário.



## ATIVIDADES



**Objetivo do algoritmo:** ler informações sobre 30 professores e realizar consulta pela matrícula.

**Entrada:** ler matrícula, nome, lotação e salário e a matrícula para pesquisa.

**Processamento:** encontrar o professor com a matrícula informada pelo usuário.

**Saída:** imprimir a lotação, se existir a matrícula informada. Se não, informar ao usuário que não há professor com a matrícula digitada.

```

Algoritmo registroprofessor
Tipo
    Cad_professor = registro
    matricula: inteiro
    nome: caractere[30]
    lotacao: caractere[20]
    salario: real

fim_registro
Var
    professor: vetor [1..30] de professor
    i, registro: inteiro
    acha: lógico
    resp: caractere[3]

Início
    Para i de 1 até 30 faça
        Escreva ("Informe a matrícula:")
        Leia (professor[i].matricula)
        Escreva ("Informe o nome:")
        Leia (professor[i].nome)
        Escreva ("Informe a lotação:")
        Leia (professor[i].lotacao)
        Escreva ("Informe o salário:")
        Leia (professor[i].salario)

    Fim_para
  
```



```

Resp "sim"
Enquanto (resp = "sim") faça
  Escreva ("Informe a matricula a ser pesquisado:")
  Leia (registro)
   $i \leftarrow 1$ 
  acha  $\leftarrow$  falso
  Enquanto ( $i \leq 20$ ) e (acha= falso) faça
    Se (professor[i].matricula = registro)
      então
        Acha  $\leftarrow$  verdadeiro
      Senão
         $i \leftarrow i + 1$ 
    fim_se
  fim_enquanto
  Se (acha=verdadeiro) então
    Escreva ("A lotação é:", professor[i].lotacao)
    Escreva ("O salário é:", professor[i].salario)
  Senão
    Escreva ("Não há professor com a matrícula informada")
  Fim_se
  Escreva ("Deseja continuar a pesquisa?")
  Leia (resp)
Fim_enquanto

Fim.

```

# SUB-ROTINAS E PROGRAMAÇÃO COM ARQUIVOS

UNIDADE

V

## Objetivos de Aprendizagem

- Conhecer e desenvolver sub-rotinas (procedimentos e funções).
- Entender os mecanismos de passagem de parâmetros e escopo de variáveis.
- Compreender algoritmos recursivos.
- Estudar organização, declaração, forma de acesso e operações em arquivos.

## Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Sub-rotinas
- Procedimentos
- Escopo de variáveis
- Passagem de Parâmetros
- Funções
- Recursividade
- Trabalhando com arquivos



## INTRODUÇÃO

Caro aluno, chegamos à última unidade da disciplina de Algoritmos e Lógica de Programação I. Nesta unidade você estudará as sub-rotinas e aprenderá a trabalhar com a manipulação de arquivos.

No dia a dia um programador, geralmente, encontra problemas complexos e abrangentes. Para resolver esse tipo de problema o primeiro passo consiste em decompô-lo em subproblemas para assim facilitar o processo de entendimento, análise e resolução. Na solução dos subproblemas são utilizadas sub-rotinas, bloco de instruções que realizam tarefas específicas. Na literatura encontramos, também, os termos subalgoritmo, subprograma e módulo, que são equivalentes a sub-rotinas. Existem dois tipos de sub-rotinas: os procedimentos e as funções. Estudaremos cada uma delas destacando sua sintaxe e modo de aplicação.

Ao trabalhar com sub-rotinas surge a necessidade de entender o conceito de escopo de variáveis, compreender o que são variáveis locais e variáveis globais e como elas impactam no algoritmo. O escopo está relacionado à visibilidade de uma variável ao longo do algoritmo, sendo que uma variável local é aquela que podemos utilizar apenas dentro da sub-rotina e uma variável global é aquela que está acessível de qualquer parte do algoritmo.

Ainda no contexto de sub-rotinas estudaremos a passagem de parâmetros por valor e por referência e o conceito de recursividade. Em relação à passagem de parâmetros veremos quando utilizar cada uma delas e qual o seu impacto no algoritmo. Estudaremos funções recursivas, que são funções que fazem chamadas a si mesmo, como construí-las, quais suas vantagens e desvantagens.

Conheceremos o conceito de arquivos, modos de concepção e como manipulá-los utilizando operações que possibilitem consultar, inserir, modificar e eliminar dados.

Ao final desta unidade você saberá construir algoritmos modularizados, utilizar arquivos para armazenamento dos dados e responder a questões do tipo: quando utilizar uma função ou um procedimento? O que são parâmetros? Como deve ser realizada a passagem de parâmetros? Qual a diferença entre passar por valor ou por referência? Como acessar um arquivo? Como percorrer um arquivo?

## SUB-ROTINAS

Em muitas situações nos deparamos com problemas complexos e abrangentes. Para resolver esse tipo de problema temos que dividi-lo em subproblemas mais simples e específicos, dividindo assim a sua complexidade e facilitando a resolução (FORBELLONE; EBERSPACHER, 2005).

A decomposição do problema em subproblemas é determinante para a redução da complexidade e, por conseguinte, na resolução do problema. Esse processo de decomposição, também, é conhecido como refinamento sucessivo ou abordagem *top down*, em que a análise parte do todo para o específico (FORBELLONE; EBERSPACHER, 2005; MANZANO; OLIVEIRA, 1997).

Na divisão dos problemas complexos utilizamos sub-rotinas para resolver cada subproblema, permitindo assim possuir a modularização. Na literatura encontramos, também, as terminologias de subprograma, módulos e subalgoritmos para designar as sub-rotinas.

E o que vem a ser então uma sub-rotina? Uma sub-rotina consiste em blocos de instruções que realizam tarefas específicas. É um trecho menor de código, um algoritmo mais simples, que resolve um subproblema por meio de operações de entrada, processamento e saída (ASCENCIO; CAMPOS, 2010; MANZANO; OLIVEIRA, 1997).

Uma sub-rotina é carregada apenas uma vez e pode ser executada quantas vezes for necessário, podendo ser utilizada para economizar espaço e tempo de programação. Em cada sub-rotina, além de ter acesso às variáveis do programa que o chamou (variáveis globais), pode ter suas próprias variáveis (variáveis locais), que existem apenas durante sua chamada. Na chamada de uma sub-rotina é possível passar informações, as quais são denominadas parâmetros (ASCENCIO; MOURA, 2010; GUIMARÃES; LAGES, 1994).

Guimarães e Lages (1994) destacam que modularização dos algoritmos apresenta as seguintes vantagens:

- Divide o problema em partes coerentes.
- Facilita o teste.

- Evita repetição do código.
- Maior legibilidade do algoritmo.

Há dois tipos de sub-rotinas: os procedimentos e funções. A seguir, estudaremos cada uma delas.

## PROCEDIMENTOS

Um procedimento consiste em um trecho de código (conjunto de instruções) com início e fim e identificado por um nome, o qual é usado para chamar a rotina de qualquer parte do programa principal ou em uma sub-rotina qualquer. Quando uma sub-rotina é chamada, ela é executada e ao seu término o processamento retorna para a linha seguinte a da instrução que a chamou (MANZANO; OLIVEIRA, 1997).

Uma sub-rotina de procedimento se caracteriza por não retornar valor para quem as chamou (ASCENCIO; CAMPOS, 2010). E sua sintaxe é:

```
procedimento <nome do procedimento>
var
    <variáveis>
início
    <instruções>
fim_procedimento
```

Outro ponto importante é que as sub-rotinas devem ser declaradas antes do programa ou sub-rotina que as chama.

Para exemplificar o funcionamento de sub-rotinas do tipo procedimento, vamos construir um programa calculadora que apresenta um menu de opções no programa principal e possibilita ao usuário escolher as seguintes operações: adição, subtração, multiplicação, divisão e sair (MANZANO; OLIVEIRA, 1997).

Sistematizando as informações do problema temos que:

**Objetivo do algoritmo:** criar uma calculadora com as operações de adição, subtração, multiplicação e divisão.

**Entrada:** ler a opção do usuário. E se for uma operação aritmética efetuar a operação.

**Processamento:** efetuar a operação aritmética selecionada.

**Saída:** imprimir o resultado da operação aritmética.

Na construção deste algoritmo vamos aplicar o conceito de procedimentos, de modo que cada operação aritmética deve ser realizada por um procedimento específico. O programa principal irá ler a opção selecionada pelo usuário e efetuar chamada aos procedimentos. Precisamos criar variáveis para armazenar a opção selecionada pelo usuário, os dois números que o usuário deseja efetuar a operação e outra variável para armazenar o resultado da operação aritmética. O processamento consiste em efetuar a operação solicitada e como saída temos que informar ao usuário o resultado da operação. Lembre-se: as rotinas devem ser declaradas antes do programa principal.

O Quadro 72 apresenta o algoritmo que efetua as quatro operações aritméticas. Note que para cada operação há um procedimento associado.

```
Algoritmo calculadora
Procedimento soma
    Var:
        A, B, Resultado: real
    Início
        Escreva ("Procedimento SOMA")
        Escreva ("Informe o valor de A:")
        Leia (A)
        Escreva ("Informe o valor de B:")
        Leia (B)
        Resultado ← A + B
        Escreva ("O resultado da soma é:",
            resultado)
```



**Fim\_procedimento**

**Procedimento** subtracao

**Var:**

A, B, Resultado: **real**

**Início**

**Escreva** ("Procedimento SUBTRAÇÃO")

**Escreva** ("Informe o valor de A:")

**Leia** (A)

**Escreva** ("Informe o valor de B:")

**Leia** (B)

Resultado  $\leftarrow$  A - B

**Escreva** ("O resultado da subtração é:", resultado)

**Fim\_procedimento**

**Procedimento** multiplicacao

**Var:**

A, B, Resultado: **real**

**Início**

**Escreva** ("Procedimento MULTIPLICAÇÃO")

**Escreva** ("Informe o valor de A:")

**Leia** (A)

**Escreva** ("Informe o valor de B:")

**Leia** (B)

Resultado  $\leftarrow$  A \* B

**Escreva** ("O resultado da multiplicação é:", resultado)

**Fim\_procedimento**

**Procedimento** multiplicação (A, B: **DIVISÃO**)

**Var:**

Resultado: **real**

**Início**

**Escreva** ("Procedimento DIVISÃO")

```
        Escreva ("Informe o valor de A:")
        Leia (A)
        Escreva ("Informe o valor de B:")
        Leia (B)
        Resultado ← A / B
        Escreva ("O resultado da divisão é:",
        resultado)

Fim_procedimento

Var

        opcao: inteiro

Início

        opcao 0

        Enquanto (opcao <> 5) faça
            Escreva ("1 - Soma")
            Escreva ("2 - Subtração")
            Escreva ("3 - Multiplicação")
            Escreva ("4 - Divisão")
            Escreva ("5 - Sair")
            Escreva ("Digite a opção:")
            Leia (opcao)
            Se (opcao <> 5) faça
                Caso (opcao)
                    Seja 1 faça soma
                    Seja 2 faça subtracao
                    Seja 3 faça divisao
                    Seja 4 faça multiplicacao
                    Senão escreva ("Opção inválida")

                fim_caso
            fim_se
        fim_enquanto

Fim
```

Quadro 72: Pseudocódigo – Algoritmo calculadora

Na solução do problema foram utilizados quatro procedimentos, os quais foram declarados antes do programa principal. Cada procedimento possui um início e fim, e declaração de variáveis. As variáveis declaradas em uma sub-rotina são ditas variáveis locais. Na seção seguinte estudaremos o escopo das variáveis.

Observe que no programa principal ocorre a chamada para cada um dos procedimentos declarados. O problema “calculadora” foi dividido em subproblemas (cada operação aritmética) e para esses subproblemas foram construídas sub-rotinas do tipo procedimento.

## ESCOPO DE VARIÁVEIS

As variáveis declaradas no interior de uma sub-rotina são chamadas de **variáveis locais**, pois podem ser utilizadas apenas dentro da sub-rotina. Ao final da execução da sub-rotina essas variáveis são destruídas e seus conteúdos são perdidos (ASCENCIO; CAMPOS, 2010). No exemplo apresentado na seção “Procedimentos” as variáveis A, B e resultado são variáveis locais.

As **variáveis globais** são aquelas declaradas fora das sub-rotinas. Elas estão acessíveis em qualquer parte do algoritmo, inclusive dentro das sub-rotinas e são destruídas apenas ao final da execução do algoritmo (ASCENCIO; CAMPOS, 2010). É o caso da variável opção utilizada no exemplo da calculadora.

O escopo de uma variável está vinculado a sua visibilidade em relação às sub-rotinas de um programa. Devemos atentar para o fato de que uma variável pode ser considerada global para todas as sub-rotinas inferiores a uma rotina principal, e dentro de uma dessas sub-rotinas a mesma variável pode ser utilizada como local.

A Figura 37 ilustra o conceito de escopo de variáveis. Temos uma rotina principal que contém as rotinas 1 e 2. Internamente a rotina 1, temos duas rotinas (1.1 e 1.2) e internamente a rotina 2, temos a rotina 2.1 Observe que a rotina principal possui duas variáveis A, B, que são variáveis globais em relação as sub-rotinas 1 e 2. No entanto, na rotina 1 temos a declaração de uma variável A, a

qual assume um contexto local para esta sub-rotina mas, é global para as sub-rotinas 1.1 e 1.2 do mesmo modo a variável X. Já as variáveis W e Y são variáveis locais que pertencem às rotinas 1.1 e 1.2, respectivamente.

Em relação à sub-rotina 2 temos que as variáveis A e B são globais para a rotina 2.1 A variável M declarada na rotina 2 é uma variável global para a rotina 2.1, que possui uma variável X que é local e não apresenta nenhuma relação com a variável X definida na rotina 1.

Em relação à sub-rotina 2 temos que as variáveis A e B são globais para a rotina 2.1 A variável M declarada na rotina 2 é uma variável global para a rotina 2.1, que possui uma variável X que é local e não apresenta nenhuma relação com a variável X definida na rotina 1.

Manzano e Oliveira (1997) destacam que uma variável declarada antes de uma sub-rotina é uma variável global para a sub-rotina.

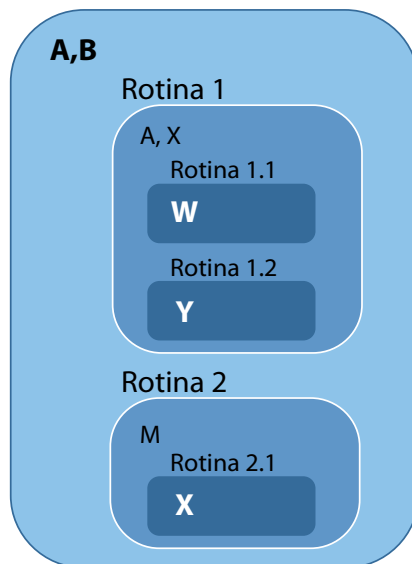


Figura 37: Escopo de variáveis  
Fonte: MANZANO; OLIVEIRA (1997)

## PASSAGEM DE PARÂMETROS

Os parâmetros servem como um ponto de comunicação bidirecional entre uma sub-rotina e o programa principal ou outra sub-rotina. Possibilitam a passagem de valores de uma sub-rotina ou rotina chamadora à outra sub-rotina e vice-versa. Devemos fazer distinção entre dois tipos de parâmetros, os formais e os reais (MANZANO; OLIVEIRA, 1997).

Os parâmetros formais são aqueles declarados por meio de variáveis juntamente com a identificação do nome da sub-rotina. Já os parâmetros reais são

aqueles que substituem os parâmetros formais quando do uso da sub-rotina por um programa principal ou rotina chamadora.

O Quadro 73 apresenta o algoritmo para o problema da calculadora utilizando o conceito de parâmetros.

```

Algoritmo calculadora
Procedimento soma (A, B: real)
    Var:
        Resultado: real
    Início
        Resultado ← A + B
        Escreva ("O resultado da soma é:",
            resultado)
Fim_procedimento
Procedimento subtracao (A, B: real)
    Var:
        Resultado: real
    Início
        Resultado ← A - B
        Escreva ("O resultado da subtração
            é :", resultado)
Fim_procedimento

Procedimento multiplicação (A, B: real)
    Var:
        Resultado: real
    Início
        Resultado ← A * B
        Escreva ("O resultado da
            multiplicação é :", resultado)
Fim_procedimento
    
```

```
Procedimento divisão (A, B: real)
    Var:
        Resultado: real
    Início
        Resultado ← A / B
        Escreva ("O resultado da divisão é :",
            resultado)
Fim_procedimento

Var
    opcao: inteiro
    X, Y: real
Início
    opcao ← 0
Enquanto (opcao <> 5) faça
    Escreva ("Informe o valor de X:")
    Leia (X)
    Escreva ("Informe o valor de Y:")
    Leia (Y)
    Escreva ("1 - Soma")
    Escreva ("2 - Subtração")
    Escreva ("3 - Multiplicação")
    Escreva ("4 - Divisão")
    Escreva ("5 - Sair")
    Escreva ("Digite a opção:")
    Leia (opcao)
Se (opcao <> 5) faça
    Caso (opcao)
        Seja 1 faça soma(X,Y)
        Seja 2 faça subtracao(X,Y)
        Seja 3 faça divisao(X,Y)
        Seja 4 faça multiplicacao(X,Y)
        Senão escreva ("Opção inválida")
    fim_caso
    fim_se
fim_enquanto
Fim.
```

Quadro 73: Pseudocódigo – Algoritmo calculadora

Observe que agora temos a leitura das variáveis X e Y no programa principal e na identificação dos procedimentos temos os parâmetros que são exigidos pelos mesmos. Os parâmetros reais são os valores de X e Y obtidos na entrada de dados e os parâmetros formais são A e B.

A passagem de parâmetro ocorre quando é realizada a substituição dos parâmetros formais pelos reais no momento da execução da sub-rotina. A passagem de parâmetros ocorre de duas formas: por valor e por referência (MANZANO; OLIVEIRA, 1997).

A passagem de parâmetro **por valor** é caracterizada pela não alteração do valor do parâmetro real quando o parâmetro formal é manipulado na sub-rotina. Isto é, qualquer alteração na variável local da sub-rotina não afetará o valor do parâmetro real correspondente. Na passagem de parâmetros por valor a sub-rotina trabalha com cópias dos valores passados no momento de sua chamada (MANZANO; OLIVEIRA, 1997; ASCENCIO; CAMPOS, 2010).

Na passagem de parâmetro **por referência** os parâmetros passados para a sub-rotina consistem em endereços de memória ocupados por variáveis. O acesso a determinado valor é realizado por apontamento do endereço. Na passagem por referência o valor do parâmetro real é alterado quando o parâmetro formal é manipulado dentro da sub-rotina (ASCENCIO; CAMPOS, 2010).

Vamos exemplificar a passagem de parâmetro por valor e por referência utilizando o problema de calcular o fatorial de um número inteiro. No Quadro 74 é apresentado o algoritmo utilizando a passagem de parâmetros por valor.

```

Algoritmo calculafatorial
Procedimento fatorial (X: inteiro)
    Var:
        i, fat: inteiro
    Início
        fat ← 1
        Para i de 1 até x passo 1 faça
            fat ← fat * i
        fim_para

```

```
                Escreva ("O fatorial é:", fat)

Fim_procedimento

Var

    n: inteiro

Início

    Escreva ("Informe o número que deseja calcular o fatorial:")

    Leia (n)

    Fatorial(n)

Fim.
```

74: Pseudocódigo – Algoritmo fatorial  
Fonte: MANZANO; OLIVEIRA (1997)

Neste caso, temos que a variável X é o parâmetro formal que receberá o valor da variável n por meio do procedimento fatorial. Ao término do laço de repetição, o valor da variável fat é impresso. Lembre-se: a variável fat é válida apenas dentro do procedimento fatorial.

No Quadro 75 temos a passagem de parâmetros por referência, no caso a variável Fat. Sempre que a passagem for por referência devemos inserir a instrução Var na declaração da sub-rotina. A variável X continua sendo passada por valor.

```
Algoritmo calculafatorial

Procedimento fatorial(X:inteiro Var Fat: inteiro)

    Var:

        i:inteiro

    Início

        fat←1

        Para i de 1 até x passo 1 faça

            fat←fat*i

        fim_para

    Fim_procedimento
```



```

Var

    n, resultado: inteiro

Inicio

    Escreva ("Informe o número que deseja calcular o fatorial")

    Leia (n)

    Fatorial(n, resultado)

    Escreva ("O fatorial é:", resultado)

Fim

```

Quadro 75: Pseudocódigo – Algoritmo fatorial  
Fonte: MANZANO; OLIVEIRA (1997)

Ao final do laço de repetição o resultado do fatorial está armazenado na variável Fat, que foi passada para a sub-rotina como referência, e ao retornar ao programa principal este valor é transferido para a variável resultado do programa principal, a qual apresenta o valor recebido dentro da sub-rotina por meio da variável fat. Este tipo de passagem de parâmetro é utilizado para se obter um determinado valor de dentro de uma sub-rotina (MANZANO; OLIVEIRA, 1997).

A Figura 38 ilustra a passagem de parâmetros por referência no cálculo do fatorial. No algoritmo principal temos a leitura do número que devemos calcular o fatorial. Este valor é armazenado na variável n. Em seguida, há o chamado do procedimento fatorial com os parâmetros n sendo passado por valor e resultado por referência. O valor de n é copiado para a variável x. Para calcular fatorial é utilizado o valor de x para controlar o laço de repetição. Ao término da execução do procedimento o valor da variável fat é passado por referência e modifica o valor da variável resultado.

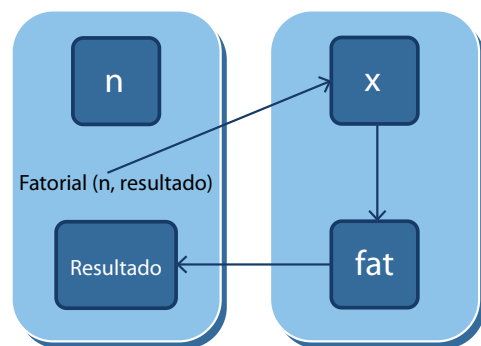


Figura 38: Representação da passagem por referência

## FUNÇÕES

Uma função, também é uma sub-rotina, que tem como objetivo desviar a execução do programa principal para realizar uma tarefa específica, com uma única diferença: sempre retorna um valor (ASCENCIO; CAMPOS, 2010).

A sintaxe é dada por:

```
funcao <nome da função> (parâmetros): <tipo da função>
var
    <variáveis>
inicio
    <instruções>
    retorne <valor>
fim_funcao
```

Um exemplo de uso de função é apresentado no Quadro 76. Note que o nome da função recebe o valor da variável fat. Observe, também, que a função tem um tipo associado, no caso inteiro.

```
Algoritmo calculafatorial
Função fatorial (X: inteiro): inteiro
    Var:
        i: inteiro
    Início
        fat←1
        Para i de 1 até x passo 1 faça
            fat fat * i
        fim_para
        retorne fat
Fim_funcao
Var
    n, resultado: inteiro
```

**Início**

**Escreva** ("Informe o número que deseja calcular o fatorial:")

**Leia** (n)

Resultado Fatorial(n)

**Escreva** ("O fatorial é:", resultado)

**Fim.**

Quadro 76: Pseudocódigo – Algoritmo fatorial  
Fonte: MANZANO; OLIVEIRA (1997)

No algoritmo principal a chamada da função é atribuída a variável resultado, pois toda função retorna um valor.

## RECURSIVIDADE

Um objeto é dito recursivo se ele for definido em termos de si próprio. Wirth (1999) destaca que o conceito de recursão não é encontrado apenas na matemática, podemos encontrá-lo no nosso dia a dia. Como, por exemplo, quando vemos uma imagem que contém a si própria (Figura 39).

A recursividade é um mecanismo que permite uma função chamar a si mesma direta ou indiretamente. Uma função é recursiva quando possui uma chamada a si própria (ZIVIANE, 2004; GUIMARÃES; LAGES, 1994).

O princípio da recursividade consiste em diminuir sucessivamente o problema em um problema menor até que a simplicidade do problema permita resolvê-lo de modo direto, isto é, sem recorrer a si mesmo. Deste modo, uma função recursiva possui um passo básico e um passo recursivo. O passo básico é imediatamente conhecido e o passo recursivo tenta resolver um subproblema do problema inicial (GUIMARÃES; LAGES, 1994).

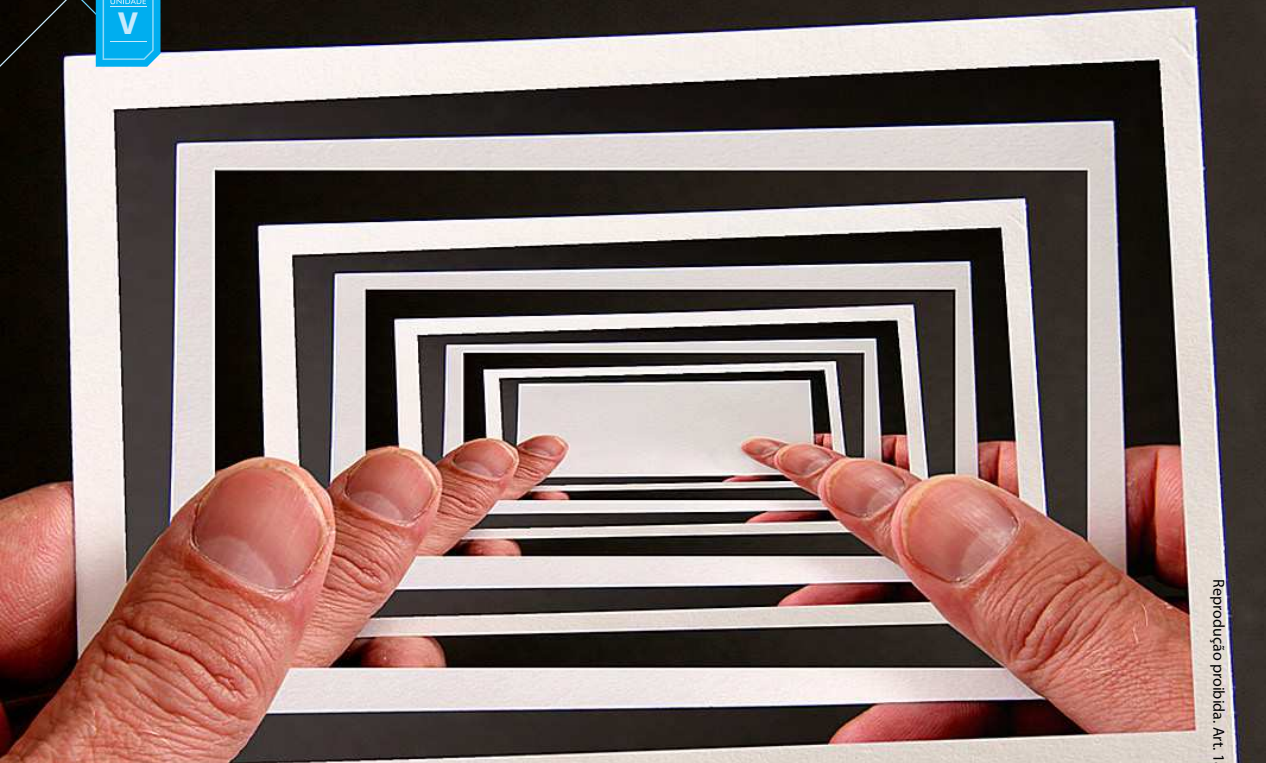


Figura 39: Imagem recursiva

Você conhece algum problema recursivo? Vamos retomar o problema do fatorial, que é um exemplo clássico de recurso. A definição recursiva é dada por:

$$n = \begin{cases} 1 & \text{se } n = 0 \\ n * (n - 1)! & \text{se } n > 0 \end{cases}$$

O Quadro 77 apresenta a função recursiva para o cálculo do fatorial.

```
Função fatorial (x: inteiro):  
    inteiro  
    Início  
        Se (x= 0) então  
            retorne 1  
        senão  
            retorne x * fatorial (n-1)  
        fim_se  
    Fim_funcao
```

Quadro 77: Pseudocódigo – Função Fatorial recursiva

A Figura 40 ilustra a simulação da função fatorial para o cálculo do fatorial de 5. Observe que são realizadas chamadas sucessivas à função fatorial até chegar na condição de parada.

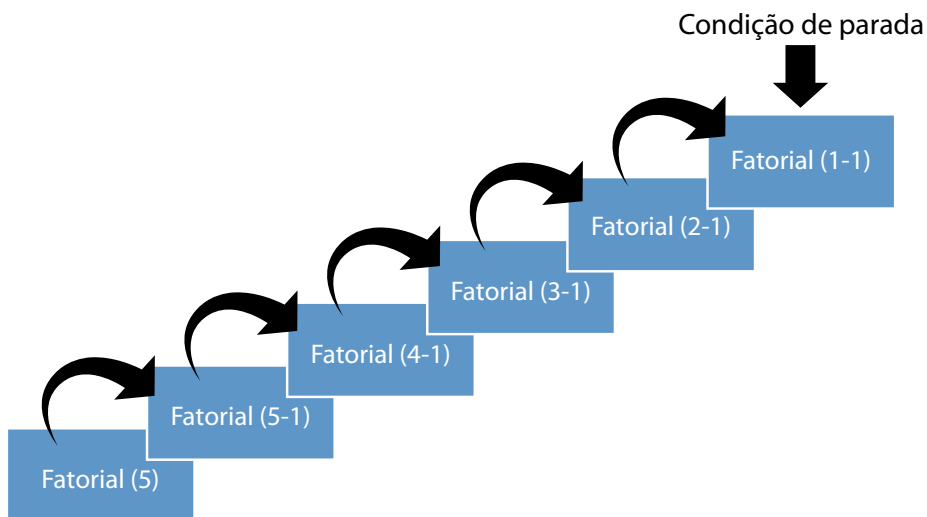


Figura 40: Simulação da Função Fatorial

Outro exemplo clássico de problema recursivo é a série de Fibonacci [0, 1, 1, 2, 3, 5, 8, 13, 21,...]. A definição recursiva é dada por:

$$Fib(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ Fib(n-2) + Fib(n-1) & \text{se } n > 1 \end{cases}$$

No Quadro 78 é apresentada a função que calcula a série de Fibonacci.

```
Função Fibonacci (x: inteiro): inteiro  
    Início  
        Se (x= 0) ou (x=1) então  
            retorne x  
        senão  
            retorne Fibonacci(n-2) + Fibonacci (n-1)  
        fim_se  
    Fim_funcao
```

Quadro 78: Pseudocódigo – Função Fibonacci recursiva

Ziviane (2004) destaca que a recursividade permite uma descrição mais concisa dos algoritmos, especialmente, quando o problema tem natureza recursiva ou utiliza estruturas recursivas. Algumas desvantagens da recursão são: algoritmos recursivos consomem mais recursos, especialmente memória e são mais difíceis de serem depurados.

## TRABALHANDO COM ARQUIVOS

Até aqui trabalhamos com a manipulação de dados que estavam na memória, ou seja, após a execução do algoritmo esses dados são perdidos. A partir de agora trabalharemos com arquivos para armazenar e manipular os dados. Você sabe o que são arquivos? Conhece algum arquivo?

Quando pensamos em arquivo nos vem em mente aquele armário com um monte de pastas para guardar informações (Figura 41). É exatamente essa a ideia de arquivos, um local para armazenar informações.



Figura 41: Arquivo

Formalmente, temos que os arquivos são estruturas de dados manipulados fora do ambiente do programa (memória principal). Um arquivo é um conjunto de registros no qual cada registro não ocupa uma posição fixa dentro da estrutura (FORBELLONE; EBERSPACHER, 2005).

Os arquivos são utilizados no armazenamento de uma grande quantidade de informações por um grande período de tempo. Um arquivo pode ser lido ou escrito por um programa, sendo constituído por uma coleção de caracteres (arquivo texto) ou bytes (arquivo binário) (ASCENCIO; CAMPOS, 2010).

Para exemplificar o conceito de arquivos vamos tomar como exemplo um cadastro de produtos com as seguintes informações: código, descrição, unidade, preço e saldo em estoque.

Como um arquivo é um conjunto de registros, o primeiro passo é definir o registro que compõe o arquivo (Quadro 79).

```
Tipo
  Cad_produto = registro
    Codigo: inteiro
    Descricao: caractere
    Unidade: caractere
    Preco: real
    Saldo: inteiro
fim_registro
Tipo
  Arq_produto = arquivo composto de Cad_produto
  Cadastro: Cad_Produto
  Produto: Arq_Produto
```

Quadro 79: Pseudocódigo – Estrutura do Registro

Sempre que formos trabalhar com arquivo, primeiro temos que abri-lo, para deixá-lo disponível para leitura/gravação. A abertura é realizada pelo comando **Abra( )**. Considerando o exemplo de cadastro de produtos teríamos: **Abra** (Produto).

Do mesmo modo que precisamos abrir o arquivo antes do processamento, precisamos fechá-lo após o uso para que as suas informações não sejam violadas. O fechamento de um arquivo é realizado por meio da instrução **Fech**( ). Para o exemplo do cadastro de produtos o fechamento é realizado com a instrução **Fech**(Produto).

Para copiar uma informação que está em um arquivo utilizamos o comando **Copie**( ), o qual possui dois parâmetros: variável do arquivo e variável registro com o mesmo formato do registro que compõe o arquivo. Ou seja, no primeiro parâmetro indicamos de onde será copiado e no segundo o local que armazenaremos o valor copiado. Lembre-se que o formato da variável deve ser igual ao do arquivo. No exemplo de cadastro de produtos a instrução para cópia seria **Copie**(Produto, Cadastro).

Para armazenar uma informação no arquivo utilizamos o comando **Guarde**( ), que possui dois parâmetros, em que o primeiro indica onde será guardado e o segundo o que será guardado. Não se esqueça de que o formato do registro que será guardado tem que ser idêntico ao do arquivo. Com isto, para o exemplo do cadastro de produto a instrução é: **Guarde**(Produto, Cadastro).

Muitas vezes, queremos apagar algumas informações (registros) do arquivo. Para isso, utilizamos o comando **Elimine**( ) que um parâmetro, o arquivo pois, elimina sempre o registro da posição corrente. Para o exemplo, teríamos **Elimine**(Produto).

Ao criar um arquivo definimos a forma como os registros são armazenados. Esta concepção pode ser sequencial ou direta.

Nos casos de concepção sequencial temos que a gravação dos registros é realizada de modo contínuo, um após o outro. Com isto, a localização de um registro obedece sua ordem de gravação. Para percorrer o arquivo utilizamos o comando **Avance**( ), que coloca o arquivo na posição do próximo registro. Para verificar se o arquivo chegou ao fim utilizamos o comando **Fda**( ). Utilizando uma estrutura de repetição o comando **Avance** e **Fda**, conseguimos percorrer o arquivo registro a registro (FORBELLONE; EBERSPACHER, 2005).

Na concepção direta, a localização de um registro no arquivo está relacionada com algum campo do arquivo, de modo que por meio deste campo conseguimos localizar o lugar que o registro está armazenado. O campo que determina



a posição do registro no arquivo é chamado de chave. Devemos atentar para o fato de que esse campo deve ser único (FORBELLONE; EBERSPACHER, 2005).

## CONSIDERAÇÕES FINAIS

Nesta unidade você aprendeu a modularizar os algoritmos utilizando sub-rotinas. A modularização é uma técnica bastante utilizada em programação devido à complexidade e tamanho dos problemas que temos que resolver no dia a dia. O princípio desta técnica consiste em decompor um problema em subproblemas de menor complexidade com o objetivo de facilitar o entendimento, análise e resolução.

Vimos que na resolução dos subproblemas são utilizadas sub-rotinas, que consistem em blocos de instruções que realizam tarefas específicas. Aprendemos que as sub-rotinas podem ser de dois tipos (procedimentos e funções) e que quando uma sub-rotina é chamada, ela é executada e ao seu término o processamento retorna para a linha seguinte a da instrução que a chamou. Estudamos que uma sub-rotina do tipo procedimento não retorna valor para quem a chamou e já uma função sempre retorna um valor.

Conhecemos o conceito de escopo de variáveis e vimos que as variáveis podem ser locais ou globais. Sendo que uma variável local é aquela que está acessível apenas dentro da sub-rotina, enquanto que uma variável global é acessível de qualquer parte do algoritmo. Além disso, estudamos o funcionamento da passagem de parâmetros por valor e por referência. Na passagem de parâmetros por valor não há alteração do valor do parâmetro real, pois a sub-rotina trabalha com cópias dos valores passados no momento de sua chamada. Já na passagem de parâmetros por referência esses valores são alterados, pois os parâmetros passados são endereços de memória.

Estudamos o conceito de recursividade, que é um mecanismo que permite uma função chamar a si mesma, e aprendemos que toda função recursiva é formada por um passo básico e um recursivo. Além disso, construímos funções

recursivas para calcular o fatorial e a série de Fibonacci, que são problemas clássicos de recursão.

Por fim, conhecemos os comandos para manipular arquivos e os modos de concepção de arquivos sequencial e direto. Em relação à concepção de arquivos vimos como realizar a inserção, consulta, alteração e exclusão de dados.



#### REFLITA

Na passagem de parâmetros por referência o parâmetro é o endereço de memória e não o valor.



#### SAIBA MAIS

Para saber um pouco mais sobre recursividade, leia o artigo “Ensino de Programação recursiva em Ciência da Computação”, disponível em: <[ftp://ftp.usjt.br/pub/revint/115\\_45.pdf](ftp://ftp.usjt.br/pub/revint/115_45.pdf)>.

O artigo destaca a importância da recursividade e apresenta cenários de aplicação.

## ATIVIDADES



1. Desenvolva um cadastro de produtos que contenha código, descrição, unidade e preço para 20 produtos. Defina um menu com as seguintes opções:
  1. Cadastrar os 20 registros.
  2. Pesquisar um produto pelo código.
  3. Classificar por ordem de descrição os registros cadastrados.
  4. Classificar por ordem de código.
  5. Apresentar todos os registros.
  6. Sair do programa de cadastro.
2. Elabore um algoritmo para cadastro de 10 pessoas contendo as seguintes informações: nome, idade, altura e peso. O menu deve apresentar as seguintes opções:
  1. Cadastrar os 10 registros.
  2. Ordenar os registros por idade.
  3. Apresentar a média de peso das pessoas.
  4. Apresentar o nome das pessoas com peso acima de 50kg.
  5. Sair do programa.
3. Escreva uma função que receba um caractere e retorne 1 se for uma consoante e 0 se for vogal.
4. Elabore um programa com uma sub-rotina que apresente o somatório dos N primeiros números pares, definidos por um operador. O valor de N será informado pelo usuário.
5. Escreva um algoritmo que efetue a leitura de um número inteiro e apresente uma mensagem informando se o número é par ou ímpar.
6. Desenvolva um programa que por meio de uma sub-rotina apresente como resultado o valor absoluto de um número.
7. Elabore um algoritmo que por meio de função efetue e apresente o valor da conversão de um valor em real de um valor lido em euros. Deverá ser solicitado por meio do programa principal o valor da cotação do euro e a quantidade de euro disponível.
8. Escreva um algoritmo utilizando função que converta uma dada temperatura lida em Celsius para Fahrenheit.



## EXERCÍCIOS DE FIXAÇÃO

1. Escreva uma sub-rotina que apresente o somatório dos N primeiros números inteiros.

**Objetivo do algoritmo:** ler um número inteiro e apresentar o somatório dos números no intervalo de 1 até N.

**Entrada:** ler um número inteiro.

**Processamento:** efetuar o somatório de 1 até N.

**Saída:** imprimir o valor do somatório.

```

Algoritmo somatorio
Função soma (X: inteiro): inteiro
    Var:
        i, total: inteiro
    Início
        total ← 0
        Para i de 1 até x passo 1 faça
            total ← total + i
        fim_para
        retorne total
Fim_funcao
Var
    n, resultado: inteiro
Início
    Escreva ("Informe o número:")
    Leia (n)
    Resultado ← soma(n)
    Escreva ("O somatório é:", resultado)
Fim.
    
```

## ATIVIDADES



2. Elabore uma sub-rotina que receba um número e apresente sua raiz quadrada.

**Objetivo do algoritmo:** ler um número e apresentar sua raiz quadrada.

**Entrada:** ler um número inteiro.

**Processamento:** calcular a raiz quadrada.

**Saída:** imprimir o valor da raiz quadrada.

```

Algoritmo raizquadrada
Função raiz (X: inteiro): real
    Início
        retorne sqr(x)
    Fim_funcao
Var
    num: inteiro
    resultado: real
Início
    Escreva ("Informe o número:")
    Leia (num)
    resultado ← raiz(n)
    Escreva ("A raiz quadrada é :", resultado)
Fim.
  
```

Quadro 81: Pseudocódigo – Exercício 2

3. Elabore uma sub-rotina que receba 3 valores de entrada e retorne o maior valor.

**Objetivo do algoritmo:** ler três números inteiros e retornar o maior valor.

**Entrada:** ler três números inteiros.

**Processamento:** comparar os números e selecionar o maior.

**Saída:** imprimir o valor do maior número.



```

Algoritmo raizquadrada
Função maior (X, Y, Z: inteiro): inteiro
    Var
        num: inteiro
    Início
        Se (x > y) então
            num ← x
        senão
            num ← y
        fim_se
        Se (z > num) então
            num ← z
        fim_se
        retorne num
Fim_funcao
Var
    resultado, a, b, c: inteiro
Início
    Escreva ("Informe o valor de A:")
    Leia (a)
    Escreva ("Informe o valor de B:")
    Leia (b)
    Escreva ("Informe o valor de C:")
    Leia (c)
    resultado ← maior(a, b, c)
    Escreva ("O maior valor é:", resultado)
Fim.
    
```

Quadro 82: Pseudocódigo – Exercício 3

4. Elabore um procedimento que receba um valor em segundos e converta para horas, minutos e segundos.

## ATIVIDADES



**Objetivo do algoritmo:** ler um valor em segundos e converter para horas, minutos e segundos.

**Entrada:** ler um número inteiro.

**Processamento:** converter os segundos para horas, dividindo por 3600. O resto da divisão deve ser dividido por 60 (minutos) e o resto da divisão resultante deve ser dividido por 60 para obter os segundos.

**Saída:** imprimir as horas, minutos e segundos resultantes da conversão.

```

Algoritmo calculatempo
Procedimento tempo (X: inteiro Var h, m, s: inteiro)
    Var:
        resto: inteiro
    Início
        h ← x/3600
        resto ← x mod 3600
        m ← resto/60
        s ← resto mod 60
    Fim_procedimento
Var
    n, hora, minuto, segundo: inteiro
Início
    Escreva ("Informe os segundos que deseja converter:")
    Leia (n)
    tempo(n, hora, minuto, segundo)
    Escreva ("Horas:", hora)
    Escreva ("Minutos:", minuto)
    Escreva ("Segundos:", segundo)
Fim.
  
```



5. Elabore uma sub-rotina que receba o valor antigo de um produto e o percentual de reajuste e retorne o valor reajustado.

**Objetivo do algoritmo:** reajustar o valor de um produto.

**Entrada:** ler o preço do produto e o percentual de reajuste.

**Processamento:** calcular o reajuste do produto.

**Saída:** imprimir o valor do produto reajustado.

```

Algoritmo produto
Função reajuste (precop, percentualp: real): real
    Var
        valor: real
    Início
        valor ← precop + (precop * (percentualp/100))
    retorne valor
Fim_funcao
Var
    valor, percentual, resultado: real
Início
    Escreva ("Informe o preço do produto:")
    Leia (valor)
    Escreva ("Informe o percentual de reajuste:")
    Leia (percentual)
    resultado ← reajuste(valor, percentual)
    Escreva ("O valor reajustado é:", resultado)
Fim.
    
```

Quadro 84: Pseudocódigo – Exercício 5



## CONCLUSÃO

Caro(a) aluno(a), chegamos ao fim de parte do nosso trabalho!

Diante dos conceitos que foram apresentados, você deve estar mais preparado para construir algoritmos e entender a importância que eles têm no desenvolvimento de software. Iniciamos o nosso aprendizado construindo algoritmos simples, que tinham apenas entrada e saída de dados, e conforme avançamos passamos a construir algoritmos mais complexos, que envolviam entrada, processamento, saída de dados e modularização.

Na Unidade I, discutimos o conceito de algoritmos, um conjunto de passos para solucionar um problema, e estudamos que dado um problema não há um único algoritmo que é solução, podemos ter vários algoritmos. Isto quer dizer que um algoritmo é um possível caminho para a solução de um problema. Entendemos o processo de análise de problemas a partir do qual estruturamos um problema em Entrada, Processamento e Saída. Estudamos os principais tipos de algoritmos, sendo eles: a descrição narrativa, o fluxograma e o pseudocódigo. O tipo adotado foi o pseudocódigo em função da facilidade de conversão para uma linguagem de programação. Vimos o conceito de variáveis, os tipos de variáveis (inteiro, real, caractere e lógica), as regras para nomeação de identificadores, palavras reservadas, expressões e operadores (aritméticos, relacionais e lógicos), comandos de atribuição, entrada e saída de dados.

A Unidade II tratou das estruturas condicionais, também denominadas estruturas de seleção ou estrutura de controle, que possibilitam a construção de algoritmos com desvios de fluxos. Isto é, algoritmos cuja execução de uma instrução ou conjunto de instruções está condicionada a um teste condicional. Estudamos quatro formas de estrutura condicional: estrutura condicional simples, estrutura condicional composta, estrutura condicional encadeada e estrutura de decisão múltipla. Revimos a construção de expressões lógicas e a tabela verdade de cada um dos operadores.

Na Unidade III aprendemos a construir algoritmos utilizando as estruturas de repetição, utilizando laços contados e laços condicionais. Nos laços de repetição vimos a estrutura Para, que é utilizada nos casos em que sabemos quantas vezes o trecho de código precisa ser repetido. Nos laços de repetição condicionais estudamos as estruturas Enquanto e Repita. A estrutura Enquanto é utilizada quando não sabemos previamente o número de repetições que deve ser executado e impomos uma condição que é realizada no final. A estrutura **Repita** é utilizada quando temos um número indefinido de repetições, no entanto, o teste lógico é realizado no final. Foi abordado, também, como o conceito de encadeamento pode ser aplicado às estruturas de repetição.

A Unidade IV discutiu as estruturas de dados homogêneas e heterogêneas. Nas estruturas de dados homogêneas conhecemos os vetores (unidimensionais) e as matrizes (multidimensionais), que agrupam diversas informações, do mesmo tipo, em uma única variável. Tratamos, também, como realizar ordenação e busca. Em relação às estruturas heterogêneas estudamos os registros, que agregam diversas



# CONCLUSÃO

informações, que podem ser de diferentes tipos.

Por fim, na Unidade V aprendemos a construir algoritmos modularizados utilizando sub-rotinas. Foram abordadas as sub-rotinas de procedimento e função, sendo que a diferença entre elas é que na sub-rotina do tipo procedimento não há retorno de valor para quem a chamou e já na função sempre retorna um valor. Estudamos o conceito de escopo de variáveis em que vimos as variáveis globais e locais. Foi tratada, também, a passagem de parâmetros por valor e por referência. Vimos que na passagem de parâmetros por valor não há alteração do valor do parâmetro real, pois a sub-rotina trabalha com cópias dos valores passados no momento de sua chamada. Já na passagem de parâmetros por referência esses valores são alterados, pois os parâmetros passados são endereços de memória. Discutimos o conceito de recursividade, que é um mecanismo que permite uma função chamar a si mesma, e em que situações pode ser aplicado. Aprendemos os comandos e operações para manipular arquivos e como estes podem ser concebidos.

Em cada uma das unidades apresentamos e construímos algoritmos que possibilitaram a aplicação prática dos conceitos e estimularam o desenvolvimento do raciocínio lógico. Nestas cinco unidades consolidamos a visão de Algoritmos e Lógica de Programação que é imprescindível para o desenvolvimento de software.

Muito sucesso a você!

Professora Camila



## REFERÊNCIAS

ASCENCIO, A. F. G.; CAMPOS, E. A. V. **Fundamentos da programação de computadores**. 5. ed. São Paulo: Prentice Hall, 2010.

CORMEN, T. H.; RIVEST, R.; LEISERSON, C. E. **Algoritmos: teoria e prática**. Rio de Janeiro: Campus, 2002.

FARRER, H. **Programação Estruturada de Computadores**. Rio de Janeiro: Ed. LTC, 1989.

FORBELLONE, A. L. V.; EBERSPACHER, H. F. **Lógica de Programação**. 3. ed. São Paulo: Makron Books, 2005.

GUIMARÃES, A. M.; LAGES, N.A.C. **Algoritmos e Estruturas de Dados**. Rio de Janeiro: LTC, 1994.

LOPES, A.; GARCIA, G. **Introdução à Programação**. Rio de Janeiro: Elsevier, 2002.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. **Estudo dirigido de algoritmos**. 3. ed. São Paulo: Érica, 1997.

SALVETTI, D. D.; BARBOSA, L. M. **Algoritmos**. São Paulo: Pearson Makron Books, 1998.

WIRTH, N. **Algoritmos e Estruturas de Dados**. Rio de Janeiro: Editora LTC, 1999.

ZIVIANE, N. **Projeto de Algoritmos com implementações em Pascal e C**. 2. ed. São Paulo: Pioneira Thomson Learning, 2004.

