

ERNESTO HENRIQUE RADIS STEINMETZ é doutorando em Ciência da Informação pela Universidade de Brasília (UnB), mestre em Gestão do Conhecimento e da Tecnologia da Informação pela Universidade Católica de Brasília (UCB). Atualmente é Coordenador Geral do Pólo EaD e docente da área de Informação e Comunicação do IFB - Campus Brasília.

ROBERTO DUARTE FONTES é mestre em Computação pela Universidade Federal do Rio e bacharel em Ciência da Computação e Grande do Sul (UFRGS). Atualmente se dedica à área acadêmica, na Coordenação de Pesquisa e Extensão e na Coordenação Adjunta PRONATEC do IFB - Campus Taguatinga.



Ministério da
Educação



CARTILHA LÓGICA DE
PROGRAMAÇÃO
IFB

CARTILHA LÓGICA DE PROGRAMAÇÃO

ERNESTO HENRIQUE RADIS STEINMETZ
ROBERTO DUARTE FONTES



Este trabalho apresenta a visão dos autores acerca de algoritmos e da lógica de programação. A proposta principal desta cartilha surgiu da troca de experiência de docentes da área de Computação no ensino destes assuntos, para alunos iniciantes de informática.

Buscou-se utilizar uma linguagem simples e direta para atender alunos dos cursos técnicos dos Institutos Federais que constituem a Rede Federal de Educação Profissional, Científica e Tecnológica, vinculada à Secretaria de Educação Profissional e Tecnológica (SETEC) do Ministério da Educação (MEC).

CARTILHA LÓGICA DE PROGRAMAÇÃO

Ernesto Henrique Radis Steinmetz
Roberto Duarte Fontes

EDITORA IFB
Brasília-DF
2013



REITOR

Wilson Conciani

PRÓ-REITORIA DE ADMINISTRAÇÃO

Valdelúcio Pereira Ribeiro

PRÓ-REITORIA DE DESENVOLVIMENTO INSTITUCIONAL

Rosane Cavalcante de Souza

PRÓ-REITORIA DE ENSINO

Nilton Nélio Cometti

PRÓ-REITORIA DE EXTENSÃO

Giano Luiz Copetti

PRÓ-REITORIA DE PESQUISA E INOVAÇÃO

Luciana Miyoko Massukado

CARTILHA LÓGICA DE PROGRAMAÇÃO

Ernesto Henrique Radis Steinmetz
Roberto Duarte Fontes

EDITORA IFB
Brasília-DF
2013

© 2013 EDITORA IFB

Todos os direitos desta edição reservados à Editora IFB.

Nenhuma parte desta publicação poderá ser reproduzida ou transmitida de qualquer modo ou por qualquer meio, eletrônico ou mecânico, incluindo fotocópia, gravação ou qualquer tipo de sistema de armazenamento e transmissão de informação, sem prévia autorização, por escrito, da Editora do IFB.



SGAN 610, Módulos D, E, F e G
CEP 70860-100 - Brasília -DF
Fone: +55 (61) 2103-2108
www.ifb.edu.br
E-mail: editora@ifb.edu.br

Conselho Editorial

Carlos Cristiano Oliveira de Faria Almeida
Cristiane Herres Terraza
Daniela Fantoni Alvares
Edilsa Rosa da Silva
Elisa Raquel Gomes de Sousa
Francisco Nunes dos Reis Júnior
Gabriel Andrade Lima de Almeida Castelo Branco
Gabriel Henrique Horta de Oliveira
Gustavo Abílio Galeno Arnt
José Gonçalo dos Santos
Josué de Sousa Mendes
Julie Kellen de Campos Borges
Juliana Rocha de Faria Silva (presidente)
Kátia Guimarães Sousa Palomo

Luciana Miyoko Massukado
Luciano Pereira da Silva
Luiz Diogo de Vasconcelos Junior
Marco Antônio Vezzani
Moema Carvalho Lima
Paulo Henrique de Azevedo Leão
Philippe Tshimanga Kabutakapua
Reinaldo de Jesus da Costa Farias
Renato Simões Moreira
Sheila Soares Daniel dos Santos
Tatiana de Macedo Soares Rotolo
Vanessa Assis Araujo
Veruska Ribeiro Machado
Vinicius Machado dos Santos

Coordenação de Publicações

Juliana Rocha de Faria Silva

Produção executiva

Sandra Maria Branchine

Gráfica

Divisão AGPRESS-AGBR Grupo AGBR

Tiragem

1.000

Ficha Catalográfica preparada por
Laura Cecília dos Santos Cruz CRB 2203

S823c Steinmetz, Ernesto Henrique Radis.
Cartilha: lógica de programação / Ernesto Henrique Radis;
Roberto Duarte Fontes. -- Brasília, DF : IFB, 2012.
94p.

Bibliografia
ISBN 978-85-64124-09-7

1. Programação (computadores). 2. Estrutura de dados.
3. Algoritmos de computador. I. Fontes, Roberto Duarte.
II. Título.

CDU – 004.421

Sumário

APRESENTAÇÃO	7
AGRADECIMENTO	9
INTRODUÇÃO	11
1. DEFINIÇÕES DE ALGORITMO	13
1.1. Conceitos	15
1.2. Fases de um algoritmo	16
1.3. Como Construir um algoritmo	17
1.4. Decomposição (<i>Top-Down</i>)	18
1.5. Desenvolvimento estruturado	20
1.6. Formas de representação de um algoritmo	20
1.7. Tipos de processamento	25
1.8. Tipos de dados	27
1.9. Variáveis (identificadores)	28
1.10. Declaração de variáveis	29
1.11. Constantes	30
1.12. Operações básicas:	32
1.13. Primeiras instruções	38
2. COMANDOS DE ENTRADA E SAÍDA	43
2.1. Comando de entrada de dados	43
2.2. Comando de saída de dados	43
3. TOMADA DE DECISÕES	47

4. ESTRUTURAS DE REPETIÇÃO	59
4.1. Repetição determinada	60
4.2. Repetição Indeterminada	63
4.2.1 Repetição Indeterminada com Validação Inicial	63
4.2.2 Repetição Indeterminada com Validação Final	65
5. ESTRUTURAS DE DADOS COMPOSTAS	73
5.1. Vetores	73
5.2. Matrizes	75
5.3. Algoritmos de pesquisa e ordenação de dados	79
6. MODULARIZANDO ALGORITMOS: PROCEDIMENTOS E FUNÇÕES	87
6.1. Procedimentos	88
6.2. Funções	90



Apresentação

Quando da resolução em um dado problema no computador, é necessário que seja primeiramente encontrada uma maneira de se descrever esse problema de uma forma clara e precisa. Sendo necessário encontrar uma sequência de passos que permitam que o problema possa ser resolvido de maneira automática e repetitiva. Esta sequência de passos é chamada de lógica de programação (algoritmo).

De anos de experiências adquiridas em sala aula, laboratórios de informática, surgiu este trabalho, que consiste em uma introdução ao estudo de lógica de programação.

O conteúdo desta cartilha é dirigido principalmente para ser utilizado como livro-texto em disciplinas sobre algoritmos, apresentando uma linguagem simples e exercícios práticos para serem aplicados em sala de aula.

Os exemplos desta cartilha foram desenvolvidos utilizando-se a ferramenta Visualg. Ferramenta esta ensinada em cursos em todo o Brasil, que permite a alunos iniciantes em programação o exercício dos seus conhecimentos num ambiente próximo da realidade.

Os tópicos desta cartilha constituem uma introdução aos conceitos de lógica de programação e uma preparação a conceitos mais avançados em programação de computadores. Estes estão agrupados em sete capítulos, cada um com o seguinte conteúdo:

1. Introdução que salienta a importância do estudo de algoritmos para os futuros profissionais de TI – Tecnologia da Informação;
2. Apresenta formalmente o conceito de Algoritmo e os conceitos básicos de lógica de programação;

3. Dá a você as formas de interatividade com o algoritmo, ou seja, a representação da troca de informações que ocorrerá entre o computador e o usuário;
4. Mostra as tomadas de decisão que permitem que determinados comandos sejam executados ou não, dependendo do resultado de um teste realizado (condição);
5. Aborda as estruturas de repetição necessárias de um ou vários trechos de um algoritmo um determinado número de vezes;
6. Apresenta estruturas de dados compostas que permitem a manipulação de dados multivalorados;
7. Introduz os conceitos de modularização de algoritmos, demonstrando técnicas de organização mais eficiente do código, bem como, a sua reutilização.

Esperamos que este trabalho possa contribuir para o seu aprendizado e consequentemente para seu futuro profissional.



Agradecimento

Agradecemos aos professores e amigos Anderson Luis Schvindt Bittencourt e Juliano de Oliveira Pires que contribuíram para a construção desta cartilha.

Agradecemos, ainda, ao professor Dr. Roberto Wagner da Silva Rodrigues que participou do início do processo de construção desta cartilha, mas por motivos profissionais acabou sendo forçado a abandonar este projeto de forma prematura.





Introdução

Existem muitos mitos e tabus sobre a dificuldade em aprender e estudar algoritmos, mas os algoritmos estão presentes em nosso cotidiano muito mais do que podemos imaginar. Quando vamos fazer uma viagem de férias e planejamos o roteiro de viagem definindo: pontos de parada, lugares a visitar, quanto tempo ficaremos em um determinado local, estradas que iremos percorrer etc., estamos montando um algoritmo para organizar de maneira lógica os passos da viagem em questão.

A sociedade moderna está amplamente dependente da automação de processos e o atual aperfeiçoamento tecnológico deve-se em parte à análise e à obtenção de descrições na execução de tarefas por meio de ações simples o suficiente, tal que pudessem ser automatizadas, sendo executadas pelo computador que é uma máquina especialmente desenvolvida para este fim.

Para resolver um problema no computador é necessário que seja primeiramente encontrada uma maneira de descrever este problema de uma forma clara e precisa. É preciso que encontremos uma sequência de passos que permitam que o problema possa ser resolvido de maneira automática e repetitiva. Esta sequência de passos é chamada de algoritmo.

Um algoritmo pode ser definido como um conjunto de regras (instruções), bem definidas, para solução de um determinado problema. Segundo o dicionário Michaelis, o conceito de algoritmo é “utilização de regras para definir ou executar uma tarefa específica ou para resolver um problema específico.”

A partir desses conceitos de algoritmos, pode-se perceber que a palavra algoritmo não é um termo computacional, ou seja, não se refere apenas à área de informática. É uma definição ampla que agora que você já sabe o que significa. Como já foi destacado anteriormente, é utilizada no cotidiano das pessoas.

Na informática, o algoritmo é o “projeto do programa”, ou seja, antes de se fazer um programa (*Software*) na Linguagem de Programação desejada (Pascal, C, Delphi etc.) deve-se fazer o algoritmo do programa. Já um programa, é um algoritmo escrito numa forma compreensível pelo computador (através de uma Linguagem de Programação), onde todas as ações a serem executadas devem ser especificadas nos mínimos detalhes e de acordo com as regras de sintaxe da linguagem escolhida.

Esta cartilha adota como linguagem padrão para codificação dos exemplos, o “Portugol”, aplicado na ferramenta VisialG, que é de livre distribuição e uso. Está disponível para download em <<http://www.apoioinformatica.inf.br/o-visualg>>.

1. Definições de algoritmo

A seguir são apresentados algumas definições de algoritmo por diferentes autores:

- “Um algoritmo é um conjunto de instruções, dispostas em uma sequência lógica, que levam a solução de um problema” (BENEDUZZI e METZ, 2010).
- “Um algoritmo deve especificar ações claras e precisas, que a partir de um estado inicial, após um período de tempo finito, produzem um estado final previsível e bem definido” (FORBELLONEE e EBERSPACHER, 2005).
- “É uma sequência de passos ordenados para a realização de uma tarefa” (PUGA e RISSETTI, 2011).
- “Algoritmo na Ciência da Computação (Informática), está associada a um conjunto de regras e operações bem definidas e ordenadas, destinadas à solução de um problema, de uma classe de problemas, em um número finito de passos” (MANZANO e OLIVEIRA, 2011).
- “Um algoritmo é uma sequência finita de instruções ou operações básicas (operações básicas sem ambigüidade e executadas em tempo finito dispendo-se apenas de lápis e papel), cuja a execução, em tempo finito, resolve um problema computacional, qualquer que seja a sua instância” (SALVETTI e BARBOSA, 2004).

A noção de algoritmo é central para toda a computação. A criação de algoritmos para resolver os problemas é uma das maiores dificuldades dos iniciantes em programação em computadores. Entretanto, isso não impede o surgimento de bons programadores e analista de sistemas.

Podemos comparar o aprendizado de algoritmos com o aprendizado de andar de bicicleta, veja na tabela 1.1.

Tabela 1.1 - Comparação do aprendizado de algoritmos com andar de bicicleta.

Perfil do estudante	Aprender algoritmos	Aprender a andar de bicicleta
iniciantes	<ul style="list-style-type: none"> - Precisam de auxílio de professores e material específico sobre o tema. - Cometem vários erros básicos, típicos dos iniciantes, mas devem continuar tentando. 	<ul style="list-style-type: none"> - Precisam de ajuda do pai ou de um irmão mais velho que já saiba andar de bicicleta. - Sofrem várias quedas, mas devem levantar e tentar novamente.
intermediários	<ul style="list-style-type: none"> - Conseguem desenvolver algoritmos de baixa e média complexidade sozinhos. - Ainda cometem alguns erros, principalmente em algoritmos mais complexos. 	<ul style="list-style-type: none"> - Conseguem andar sozinhos, mas eventualmente sofrem algumas quedas. - Quando se deparam com situações perigosas podem sofrer quedas.
avançados	<ul style="list-style-type: none"> - Conseguem desenvolver e entender algoritmos de complexidades variadas. - Tornam-se instrutores de algoritmos identificando erros dos iniciantes em questão de segundos. 	<ul style="list-style-type: none"> - Conseguem executar manobras radicais ou andar em diversos terrenos. - Tornam-se instrutores para iniciantes.

Uma das formas mais eficazes de aprender algoritmos é através de muitos exercícios. Vejam algumas dicas de como aprender e como não aprender algoritmos na tabela 1.2.

Tabela 1.2 - Como aprender algoritmos.

Algoritmos não se aprendem	Algoritmos se aprendem
Copiando algoritmos	Construindo algoritmos
Estudando algoritmos prontos	Testando algoritmos

1.1. Conceitos

Os conceitos apresentados neste tópico derivam do entendimento de professores da área (citados no agradecimento), que contribuíram para estas definições em conversas cotidianas informais e trocas de experiências sobre os assuntos abordados, bem como, foram elaborados com base na bibliografia utilizada na construção desta cartilha.

Instrução: 1) são conjuntos de regras ou normas definidas para a realização ou emprego de algo. Em informática, é o que indica a um computador uma ação elementar a ser executada. Em um algoritmo, pode ser considerado um passo da sua execução. 2) São frases que indicam as ações a serem executadas. São compostas de um verbo no imperativo/infinitivo mais um complemento.

Ex. Bata (bater) duas claras em neve
Ligue (ligar) os faróis
Abra (abrir) a porta

Algoritmos: Um algoritmo é formalmente uma sequência lógica e finita de passos que levam a execução de uma tarefa. Podemos pensar em algoritmo como uma receita, uma sequência de instruções que dão cabo de uma meta específica. Estas tarefas não podem ser redundantes nem subjetivas na sua definição, devem ser claras e precisas.

Até mesmo as coisas mais simples, podem ser descritas por sequências lógicas. Por exemplo:

Exemplo 1:

“Chupar uma bala”

1. Pegar a bala.
2. Retirar o papel.
3. Chupar a bala.
4. Jogar o papel no lixo.

Exemplo 2:

“Somar dois números quaisquer”

1. Escreva o primeiro número no círculo A.
2. Escreva o segundo número no círculo B.
3. Some o número do círculo A com número do círculo B e coloque o resultado no retângulo.

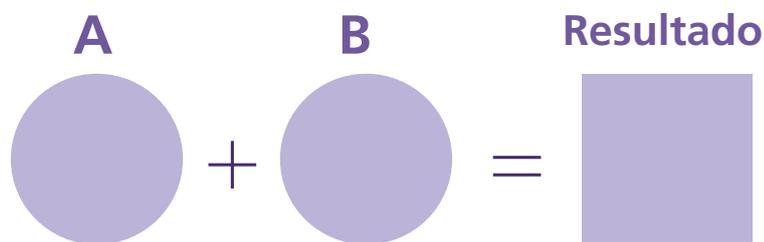


Figura 1.1 – Exemplo da sequência de passos para a soma de dois números.

Programa: Corresponde as ações de um algoritmo transcritas em uma linguagem de programação obedecendo à sua sintaxe. São exemplos de linguagens de programação: Java, Delphi, C, Pascal, etc.

1.2. Fases de um algoritmo

Para definirmos um algoritmo que representa uma solução de um problema que utiliza um computador, temos que observar as seguintes etapas:

1	Definir o problema. Não podemos misturar vários problemas em um e tentar resolver todos juntos.
2	Realizar um estudo da situação atual e verificar quais a(s) forma(s) de resolver o problema, registrando a solução encontrada.

3	Terminada a fase de estudo, utilizar uma linguagem de programação para escrever o programa que deverá, a princípio, resolver o problema.
4	Analisar junto aos usuários se o problema foi resolvido.
5	Se a solução não foi encontrada, o problema deverá ser retornado para a fase de estudo para se descobrir onde está a falha.

1.3. Como Construir um algoritmo

a) Análise preliminar

Entenda o problema, identifique os dados envolvidos e os resultados esperados.

Ao montar um algoritmo, precisamos primeiro dividir o problema apresentado em três fases fundamentais conforme a figura 1.2 que apresenta a descrição básica dos sistemas de informação.



Figura 1.2 – Etapas de um sistema de informação.

Onde temos:

ENTRADA: São os dados de entrada do algoritmo informados pelo usuário.

PROCESSAMENTO: São os procedimentos utilizados para chegar ao resultado.

SAÍDA: São os dados já processados, apresentados ao usuário.

b) Solução

Desenvolver a sequência lógica e finita de passos que levam a execução de uma tarefa para resolver o problema.

c) Teste de qualidade (Teste de mesa)

Ideal testar o algoritmo com o máximo de combinações possíveis.

d) Alteração

Caso o resultado do algoritmo não seja satisfatório, altere-o e submeta a um novo teste.

e) Produto final

Algoritmo concluído e testado.

1.4. Decomposição (*Top-Down*)

A decomposição (*Top-Down*) consiste em pegar um grande problema, de difícil solução, e dividi-lo em problemas menores que devem ser mais facilmente resolvidos.

Exemplo:

Divisão 1:

1. Trocar uma lâmpada queimada.

Divisão 2:

1. Pegar a lâmpada nova (Início)
2. Trocar a lâmpada queimada pela lâmpada nova (Meio)
3. Colocar a lâmpada queimada no lixo (Fim)

Divisão 3:

1. Pegue a escada
2. Coloque a escada abaixo da lâmpada queimada
3. Pegue a lâmpada nova
4. Suba na escada
5. Retire a luminária
6. Retire a lâmpada queimada
7. Coloque a lâmpada nova
8. Coloque a luminária
9. Desça da escada
10. Jogue a lâmpada queimada no lixo
11. Guarde a escada

Obs. Algumas instruções poderiam ser divididas mais de uma vez.



EXERCÍCIOS DE FIXAÇÃO

1. Crie uma sequência lógica para tomar banho.
2. Faça um algoritmo para somar dois números e multiplicar o resultado pelo primeiro número.
3. Descreva com detalhes a sequência lógica para Trocar um pneu de um carro.

1.5. Desenvolvimento estruturado

São técnicas que permitem sistematizar e ajudar no desenvolvimento de algoritmos para a resolução de grandes e complexos problemas computacionais.

Objetivos destas técnicas:

- Facilitar o desenvolvimento do algoritmo;
- Facilitar o seu entendimento pelos humanos;
- Antecipar a comprovação de sua correção;
- Facilitar a sua manutenção e modificação;
- Permitir que o seu desenvolvimento possa ser empreendido simultaneamente por uma equipe de pessoas.

1.6. Formas de representação de um algoritmo

a) Descrição narrativa

Ex. Cálculo da média de um aluno:

Obter as notas da primeira e da segunda prova;

Calcular a média aritmética entre as duas.

Se a média for maior ou igual a 7, o aluno foi aprovado, senão ele foi reprovado

b) Fluxograma

Os fluxogramas ou diagramas de fluxo são uma representação gráfica que utilizam formas geométricas padronizadas ligadas por setas de fluxo, para indicar as diversas ações (instruções) e decisões que devem ser seguidas para resolver o problema em questão. É uma forma de representação gráfica de algoritmos, ou seja, das instruções e/ou módulos do processamento.

O fluxograma permite visualizar os caminhos (fluxos) e as etapas de processamento de dados possíveis. Dentro destas etapas, os passos para a resolução do problema utiliza modelagem com uma linguagem visual para especificar o conjunto de instruções por meio de formas geométricas, facilitando a compreensão da lógica utilizada pelo profissional.

Existem atualmente vários padrões para definir as formas geométricas a serem utilizadas para as diversas instruções (passos) a serem seguidos pelo sistema. O padrão utilizado nesta disciplina, será definido pelo professor durante as aulas.

Seguem alguns símbolos que são utilizados no fluxograma:

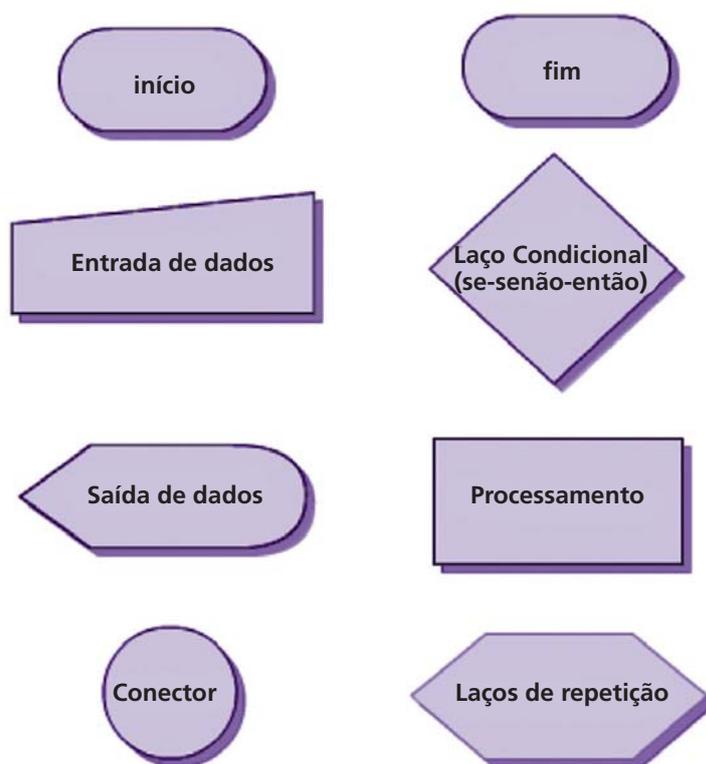


Figura 1.3 – Principais elementos para construção de fluxogramas.

A seguir temos um exemplo de um fluxograma que lê dois valores e mostra a soma destes valores:

- A princípio, marcamos o início do fluxograma;
- Em seguida, lemos o 1º valor que será armazenado na variável "valor1";
- Depois, lemos o 2º valor que será armazenado em uma variável diferente chamada "valor2".
- Efetuamos a soma de "valor1" com "valor2" e armazenamos o resultado desta operação aritmética na variável "resultado".
- Em seguida, mostramos o valor armazenado na variável "resultado".
- E finalizamos o fluxo.

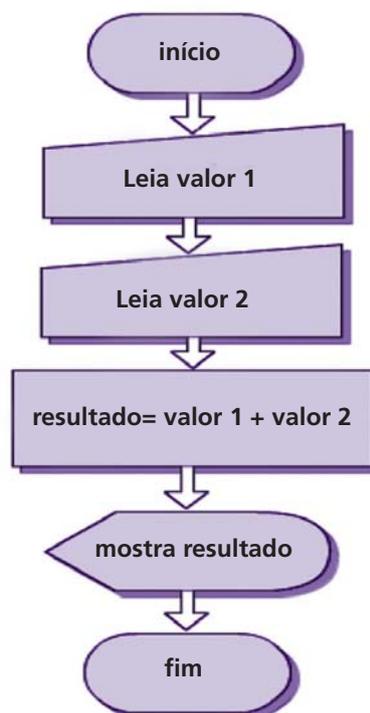


Figura 1.4 – Exemplo de um fluxograma.



EXERCÍCIOS DE FIXAÇÃO

Desenhe um fluxograma para as seguintes situações:

1. Receber o valor do salário de um trabalhador, calcular e mostrar o valor diário deste salário.
2. Calcular e mostrar a metade de um número qualquer informado pelo usuário.
3. Ler três números quaisquer e mostrar o resultado da soma destes números.
4. Ler quatro números quaisquer e mostrar o resultado da média destes números.
5. Calcular a área de um retângulo.
6. Calcular a área de um círculo.
7. Ler a idade de uma pessoa expressa em anos, meses e dias e mostrar expressa apenas em dias.
8. Ler um número qualquer e verificar se este número é maior ou igual a 10. Para a opção verdadeira mostrar a palavra "Maior", senão mostrar "Menor".
9. Ler 5 valores, calcular e mostrar a média aritmética destes valores. Se o resultado for maior que 100, mostrar a palavra "Maior", senão mostrar "Menor".
10. Mostrar se um número qualquer é positivo, ou negativo, ou zero.

c) Pseudocódigo, também conhecido como Linguagem Estruturada ou Portugol.

A representação de um algoritmo na forma de pseudocódigo é a seguinte:

```
Algoritmo Nome_Do_Algoritmo  
Variáveis  
    Declaração das variáveis  
Procedimentos  
    Declaração dos procedimentos  
Funções  
    Declaração das funções  
Início  
Corpo do Algoritmo:  
    Entrada  
    Processamento  
    Saida  
Fim
```

Exemplo:

```
1 Algoritmo "Média" // nome do algoritmo  
2 var // declaração de variáveis  
3 N1, N2, Media: real  
4 inicio // início do algoritmo  
5 escreva ("informe o primeiro valor")  
6 leia (N1)  
7 escreva ("informe o segundo valor")  
8 leia (N2)  
9 Media <- (N1+N2)/2  
10 se Media >= 7 entao  
11 escreva ("Aprovado!!")  
12 senao  
13 escreva ("Reprovado!!")  
14 fimse  
15 fimalgoritmo // fim do algoritmo
```

A **identação** (deslocamento para a direita) de uma instrução significa que tal instrução está subordinada à instrução anterior e facilita em muito a compreensão e a manutenção do algoritmo e dos códigos-fontes em uma linguagem de programação.

1.7. Tipos de processamento

a) Processamento sequencial

As instruções do algoritmo são executadas uma após a outra, sem que haja desvio na sequência das instruções, sendo cada instrução executada uma única vez.

Exemplo: Obter a média aritmética das quatro notas.

1. Início
2. Some as duas primeiras notas
3. Some a terceira nota com o resultado da instrução 2
4. Some a quarta nota com o resultado da instrução 3
5. Divida o resultado da instrução 4 por 4
6. Fim

Se as quatro notas são 10, 5, 8, 1 a execução das instruções ficará da seguinte forma:

1. Início
2. $10 + 5 = 15$
3. $15 + 8 = 23$
4. $23 + 1 = 24$
5. $24 / 4 = 6$
6. Fim

b) Processamento condicional

Um conjunto de instruções (pode ser apenas uma instrução) é executado ou não dependendo de uma condição. Se a condição que estiver sendo testada tiver resposta afirmativa, o conjunto de instruções é executado.

Exemplo: Obter a média aritmética das quatro notas. Se a média for maior ou igual a sete, o aluno está aprovado, caso contrário, está reprovado.

1. Início
2. Some as duas primeiras notas
3. Some a terceira nota com o resultado da instrução 2
4. Some a quarta nota com o resultado da instrução 3
5. Divida o resultado da instrução 4 por 4
6. Se o resultado da instrução 5 for maior ou igual a 7
 7. Aprove o aluno
8. Se o resultado da instrução 5 for menor que 7
 9. Reprove o aluno
10. Fim

Se as quatro notas são 10, 5, 8, 1, a execução das instruções ficará da seguinte forma:

1. Início
2. $10 + 5 = 15$
3. $15 + 8 = 23$
4. $23 + 1 = 24$
5. $24 / 4 = 6$
6. Resposta negativa
7. Resposta afirmativa - resultado < 7
 8. Então - aluno reprovado
9. Fim

Note que a instrução 6 foi executada. Como a resposta foi negativa (a média foi inferior a sete), o aluno foi reprovado. Não foi executada a instrução sete.

c) Processamento com repetição

Conjunto de instruções (pode ser apenas uma) que é executado um determinado número de vezes, sendo determinada uma condição de parada.

Exemplo: Obter a média aritmética das quatro notas de todos os alunos da sala.

1. Início
2. Para cada aluno da sala
 3. Some as duas primeiras notas
 4. Some a terceira nota com o resultado da instrução 3
 5. Some a quarta nota com o resultado da instrução 4
 6. Divida o resultado da instrução 5 por 4
7. Fim

Se existem dois alunos na sala e suas notas são: 10, 5, 8, 1 e 4, 6, 7, 3 a execução das instruções ficará da seguinte forma

$$10+5=15$$

$$15+8=23$$

$$23+1=24$$

$$24/4=6$$

$$4+6=10$$

$$10+7=17$$

$$17+3=20$$

$$20/4=5$$

Observe que um laço de repetição necessita sempre de uma condição de parada que seja válido, caso contrário o programa entrará em um "laço" infinito.

1.8. Tipos de dados

Os dados são representados pelas informações a serem processadas por um computador. Estas informações estão caracterizadas por quatro tipos de dados: inteiros, reais, caracteres e lógicos.

a) Tipo inteiro	Dados numéricos positivos ou negativos, excluindo-se qualquer número fracionário. Exemplo: 35, 0, -56
b) Tipo real	Dados numéricos positivos, negativos e números fracionários. Exemplo: 1.2, -45.897
c) Tipo caractere	São do tipo caractere as sequências contendo letras, números e símbolos especiais. Uma sequência de caracteres deve ser indicada entre aspas. Sendo também conhecido como: alfanumérico, string, literal ou cadeia. Exemplo: "PROGRAMAÇÃO", "Rua Alfa, 52 apto 1", "7", "249-4455"
d) Tipo lógico	São do tipo lógico ou booleano os dados com valores verdadeiro/true (T) e falso/false (F), sendo que este tipo poderá representar um dos dois valores.

1.9. Variáveis (identificadores)

Podemos imaginar uma variável como sendo um local onde se pode colocar qualquer valor do conjunto de valores possíveis para o tipo definido para a variável.

Toda variável possui um nome, um tipo e um valor associado.

O nome de uma variável é utilizado para sua identificação e posterior uso dentro de um programa, sendo assim, é necessário estabelecer algumas regras de utilização destas.

- O nome de uma variável pode conter um ou mais caracteres;
- O primeiro caractere do nome de uma variável deve ser sempre uma letra;
- Não pode possuir espaços em branco;
- Não pode ser uma palavra reservada a uma instrução de programa;
- Não poderão ser utilizados outros caracteres que não sejam letras e números; e
- Os nomes escolhidos devem explicitar seu conteúdo.

Qualquer caminho seguido no diagrama abaixo levará a um nome de variável válido.

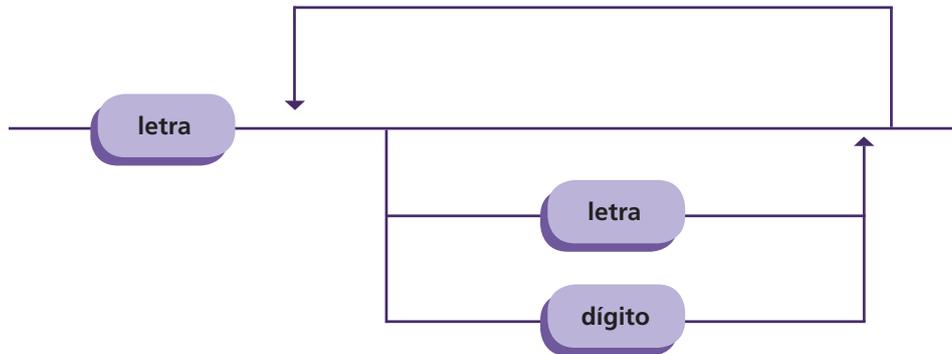


Figura 1.5 – Diagrama para validação de nomes de variável.

Onde:

Letra: Qualquer letra do alfabeto [a...z, A...Z]

Dígito: Dígitos de [0...9]

São identificadores válidos:

NOMEUSUARIO, FONE1, X, DELTA25, Z4, Idade, X1, MODELODEVeiculo.

São identificadores inválidos:

*nome, 2Fone, NOME/USUARIO, MODELO-DE-VEICULO, Nota do Aluno.

1.10. Declaração de variáveis

Para que os programas manipulem valores, estes devem ser armazenados em variáveis e para isso, devemos declará-las de acordo com a sintaxe:

```
<lista-de-variaveis>: <tipo-de-dado>  
<lista-de-variaveis>:vetor "["<lista-de-intervalos>"]" de <tipo-de-dado>
```

Exemplo:

```
3 var // declaração de variáveis
4   N1, N2, Media: real
5   idade: inteiro
```

```
7 var
8   vet: vetor [1..10] de real
9   matriz: vetor [0..4,8..10] de inteiro
10  nome_do_aluno: caractere
11  sinalizador: logico
```

Nesta definição, deveremos seguir as regras seguintes:

VARIÁVEIS é a palavra-chave que deverá ser utilizada uma única vez na definição das variáveis e antes do uso destas;

Variáveis de tipos diferentes deverão ser declaradas em linhas diferentes;

Em uma mesma linha, quando quisermos definir variáveis de mesmo tipo, deveremos usar o símbolo de vírgula (,) para separá-las.

1.11. Constantes

Tudo aquilo que é fixo ou estável.

Exemplo: o valor do $\pi = 3,14159$



EXERCÍCIOS DE FIXAÇÃO

1. Identifique os Atributos e declare variáveis para um objeto automóvel.
2. Identifique os Atributos e declare variáveis para um objeto Pessoa.
3. Identifique os atributos e declare variáveis para um objeto Eleitor.
4. Assinale com C os nomes corretos de variável e com I os incorretos. Explique o que está errado nos nomes incorretos.
 - a. certo
 - b. *oi
 - c. oi!
 - d. 'Lucio'
 - e. sorte#
 - f. i123453
 - g. Nota do Aluno
 - h. o
 - i. arquivox11
 - j. 1ate
5. Classifique os dados especificados abaixo de acordo com seu tipo, assinalando com **I** os dados do tipo inteiro, com **R** os reais, com **C** os caracteres (literais), com **B** os lógicos (booleanos) e com **N** aqueles para os quais não é possível definir o tipo de dado.
 - a. -988786,987
 - b. "34,50"
 - c. "Casa do Titio"
 - d. .F.
 - e. site
 - f. -33
 - g. ".V".
 - h. 0,5
 - i. .'V'.
 - j. ".F."
 - k. 12,89
 - l. 215333

1.12. Operações básicas:

a) Expressões

Na lógica de programação, uma expressão tem o mesmo objetivo/conceito do termo expressão da matemática comum, ou seja, um conjunto de variáveis e constantes que se relacionam por meio de operadores aritméticos. Este conjunto de expressão e operadores aritméticos (soma por exemplo) formam uma fórmula que, após solucionada fornecem um resultado em específico.

Observe o gráfico a seguir:

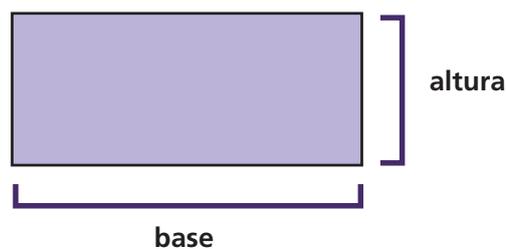


Figura 1.6 – Gráfico explicativo do cálculo da área de um triângulo.

Neste caso, a fórmula para se calcular a área de um retângulo é igual ao produto de sua altura por sua base. Poderemos, então, montar a fórmula como se segue:

```
AREA <- altura * base
```

Observe que, no exemplo acima, utilizamos três variáveis: altura, base e área. O resultado final é armazenado na variável AREA, ou seja, local de memória onde será armazenado o resultado da avaliação da expressão e o operador é o sinal de "*" que em linguagem de computador representa a multiplicação.

Já que falamos em operadores, vamos ser um pouco mais claros neste tópico.

Os operadores são elementos ativos que tem influência sobre os operandos e através de sua ação resultam em um determinado valor. No exemplo acima, os operandos seriam as variáveis "altura" e "base" e o operador de multiplicação, o "*".

Em lógica de programação, os operadores podem ser classificados em dois grupos:

- **Binários:** quando trabalham sobre os operandos em uma expressão. Os mais conhecidos são os símbolos +, -, * e /, que representam a soma, a subtração, a multiplicação e a divisão, respectivamente.
- **Unários:** quando tem influência sobre um único operando, indicando, por exemplo, que este se trata de um número negativo e não vem acompanhado de nenhum outro operando. Exemplo: ao representar o número quatro negativo podemos utilizar (-4). Isso não quer dizer que queremos subtrair o número quatro de outro valor.

Assim como classificamos os operadores, podemos também classificar as expressões quanto ao seu tipo, conforme mostra a lista a seguir:

- **Expressão aritmética:** quando o resultado de sua avaliação for um número, que pode ser positivo ou negativo, assim como inteiro ou real;
- **Expressão lógica:** quando o resultado de sua avaliação for um valor lógico, ou seja, verdadeiro (.T.) ou falso (.F.);
- **Expressão literal:** quando o resultado de sua avaliação for um valor literal.

Cada uma das expressões vistas acima possui seus operadores em específico. A seguir, vamos mostrar uma tabela contendo esses operadores:

b) Operadores

Na solução da grande maioria dos problemas, é necessário que as variáveis tenham seus valores consultados ou alterados e para isto, devemos definir um conjunto de OPERADORES, sendo eles:

- **Operador de Atribuição:**

NomeDaVariavel ← Valor ou Expressão Atribuída.

OU

NomeDaVariavel = Valor ou Expressão Atribuída

ATENÇÃO: usaremos o símbolo ← para atribuição.

- **Operadores Aritméticos:**

Tabela 1.3 – Operadores aritméticos.

+, -	Operadores unários, isto é, são aplicados a um único operando. São os operadores aritméticos de maior precedência. Exemplos: -3, +x. Enquanto o operador unário - inverte o sinal do seu operando, o operador + não altera em nada o seu valor.
\	Operador de divisão inteira. Por exemplo, $5 \setminus 2 = 2$. Tem a mesma precedência do operador de divisão tradicional.
+, -, *, /	Operadores aritméticos tradicionais de adição, subtração, multiplicação e divisão. Por convenção, * e / têm precedência sobre + e -. Para modificar a ordem de avaliação das operações, é necessário usar parênteses como em qualquer expressão aritmética.
MOD ou %	Operador de módulo (isto é, resto da divisão inteira). Por exemplo, $8 \text{ MOD } 3 = 2$. Tem a mesma precedência do operador de divisão tradicional.
^	Operador de potenciação. Por exemplo, $5 \wedge 2 = 25$. Tem a maior precedência entre os operadores aritméticos binários (aqueles que têm dois operandos).

- **Funções Primitivas:** SEN(x); COS(x); TG(x); ABS(x); INT(x); Raizq(x).

Podem variar, dependendo da linguagem de programação em que o algoritmo será descrito.

■ Operadores Relacionais:

São utilizados para relacionar variáveis ou expressões, resultando num valor lógico (Verdadeiro ou Falso), sendo eles:

Tabela 1.4 – Operadores relacionais.

$=, <, >, <=, >=, <>$	Respectivamente: igual, menor que, maior que, menor ou igual a, maior ou igual a, diferente de. São utilizados em expressões lógicas para se testar a relação entre dois valores do mesmo tipo. Exemplos: $3 = 3$ (3 é igual a 3?) resulta em VERDADEIRO; $"A" > "B"$ ("A" está depois de "B" na ordem alfabética?) resulta em FALSO.
-----------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

■ Operadores Lógicos:

São utilizados para avaliar expressões lógicas, sendo eles:

Tabela 1.5 – Operadores Lógicos.

nao	Operador unário de negação. nao VERDADEIRO = FALSO, e nao FALSO = VERDADEIRO. Tem a maior precedência entre os operadores lógicos. Equivale ao NOT do Pascal.
ou	Operador que resulta VERDADEIRO quando um dos seus operandos lógicos for verdadeiro. Equivale ao OR do Pascal.
e	Operador que resulta VERDADEIRO somente se seus dois operandos lógicos forem verdadeiros. Equivale ao AND do Pascal.

Veja a seguir a tabela verdade para as operações "e" e "ou" com várias formas de representação dos valores booleanos.

Tabela 1.6 – Tabela verdade dos operadores .E. e .OU.

A	B	A . E . B	A . OU . B
VERDADEIRO/ TRUE/.T./.V./1	VERDADEIRO/ TRUE/.T./.V./1	VERDADEIRO/ TRUE/.T./.V./1	VERDADEIRO/ TRUE/.T./.V./1
VERDADEIRO/ TRUE/.T./.V./1	FALSO/ FALSE/.F./0	FALSO/ FALSE/.F./0	VERDADEIRO/ TRUE/.T./.V./1
FALSO/ FALSE/.F./0	VERDADEIRO/ TRUE/.T./.V./1	FALSO/ FALSE/.F./0	VERDADEIRO/ TRUE/.T./.V./1
FALSO/ FALSE/.F./0	FALSO/ FALSE/.F./0	FALSO/ FALSE/.F./0	FALSO/ FALSE/.F./0

Veja o exemplo a seguir e, utilizando as tabelas verdade dos operadores, identifique o resultado booleano para as expressões propostas.

APROVADO?



Expressão 1 : (Média \geq 60) **e** (Frequência \geq 0.75)



REPROVADO?

Expressão 2 : (Média $<$ 60) **ou** (Frequência $<$ 0.75)

Tabela 1.7 – Tabela verdade do operador .E.

A	B	A e B
1	1	1
1	0	0
0	1	0
0	0	0

Tabela 1.8 – Tabela verdade do operador .OU.

A	B	A ou B
1	1	1
1	0	1
0	1	1
0	0	0

Tabela 1.9 – Tabela verdade do operador .NAO.

A	nao(A)
1	0
0	1

PRIORIDADE DE OPERADORES:

Durante a execução de uma expressão que envolve vários operadores, é necessário a existência de prioridades, caso contrário poderemos obter valores que não representam o resultado esperado.

A maioria das linguagens de programação utiliza as seguintes prioridades de operadores:

- 1º - Efetuar operações embutidas em parênteses "mais internos"
- 2º - Efetuar Funções
- 3º - Efetuar multiplicação e/ou divisão
- 4º - Efetuar adição e/ou subtração
- 5º - Operadores Relacionais
- 6º - Operadores Lógicos

OBS: O programador tem plena liberdade para incluir novas variáveis, operadores ou funções para adaptar o algoritmo as suas necessidades, lembrando sempre, que estes devem ser compatíveis com a linguagem de programação a ser utilizada.

EXERCÍCIOS DE FIXAÇÃO



1. Na expressão $A * B - C$, qual será a sequência de execução?
2. Na expressão $(A * B) - C$, qual será a sequência de execução?
3. Na expressão $A * (B - C)$, qual será a sequência de execução?
4. Na expressão $(A * (B - C * (D / E)))$, qual será a sequência de execução?

A lógica para se montar uma expressão é ponto determinante na questão do resultado ser ou não verdadeiro, ou seja, de ser o resultado esperado.

1.13. Primeiras instruções

Após dominarmos os assuntos anteriormente tratados, passaremos a estudar as instruções primitivas, ou seja, os comandos básicos que executam tarefas consideradas essenciais para a execução de qualquer programa de computador. Um exemplo deste tipo de instrução são aquelas responsáveis pela comunicação do operador com o computador por meio do teclado (entrada de dados) ou ainda a impressão de um relatório (saída de dados "sem contar com a parte estética do relatório, alinhamento de colunas etc.).

Toda linguagem de programação tem por obrigação possuir instruções primitivas, pois sem estas instruções não existiria comunicação com os periféricos.

Antes de qualquer coisa, você saberia diferenciar periféricos de entrada e de saída de dados? A resposta é simples, periféricos de entrada são aqueles responsáveis pela passagem de dados do mundo externo para a memória do computador, como por exemplo, o teclado, unidade de CD-ROM etc. Já os periféricos de saída recebem os dados do computador para outro equipamento externo, como por exemplo, o monitor de vídeo, impressora etc.

Toda instrução, primitiva ou não, possui uma sintaxe, ou seja, uma regra que deverá ser seguida para a construção de seu código. Caso esta regra não seja obedecida, o seu programa pode simplesmente não funcionar.

Devemos saber também que o conjunto de ações que serão realizadas pelo computador após a execução de um determinado comando é conhecida como semântica.



EXERCÍCIOS DE FIXAÇÃO

1. Quais os valores das expressões a seguir:

$$2 + 3 * 6$$

$$12 / 3 * 2 - 5$$

$$31 / 4$$

$$31 \text{ MOD } 4$$

$$31 \% 4$$

2. Considere o seguinte algoritmo:

```
1 algoritmo "Exercicio2"
2 var
3   Pig, Vari, Total, A, I : Inteiro
4   ValorA, X : Real
5 inicio
6   Vari <- 2
7   Total <- 10
8   ValorA <- 7.0
9   A <- -4
10  I <- 80
11  X <- 4.0
12  X <- Total / Vari
13  X <- X + 1
14  A <- A + I
15  Pig <- 10
16  A <- A + I % 6
17  ValorA <- Pig * ValorA + X
18 fimalgoritmo
```

EXERCÍCIOS DE FIXAÇÃO



Mostre quais os valores armazenados em cada uma das variáveis após a execução de cada um dos comandos do algoritmo acima.

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

```
1 algoritmo "Exercicio3"
2 var
3   Q, W, R :inteiro
4   Ex :Real
5 inicio
6   Q <- 10
7   Q <- 10 + 30
8   W <- -1
9   W <- W + Q
10  Q <- Q Resto W
11  Q <- W div (Q + 4)
12  Ex <- 2 * Q / W
13  R <- 0
14  R <- R + 1
15  R <- R + 1
16 fimalgoritmo
```

4. Considerando as variáveis abaixo, indique o resultado de cada uma das expressões a seguir:

X	Y	Z	W	Q
V	2	4	1,50	"PROVA"

- a. $(2 * (5 \text{ MOD } 3) + Y + W * 2) > Z$ ou
 $(Y < Z - 1)$ e $\text{nao}((Z * (Y - 1) + Y) < 100)$ ()
- b. $(X \text{ ou } .V.)$ ou $(Z * 2 \% Y < W * Y)$ ()
- c. F e $X = \text{nao}(.V.)$ ou $Y < Z$ ()
- d. $Y + 4 * (Z \text{ MOD } 3 + 1)$ ()
- e. $(3 + W - (Y + 1) / 2) + 1,5$ ()



EXERCÍCIOS DE FIXAÇÃO

5. Dada a declaração de variáveis:

I1, I2, I3 : inteiro

R1, R2, R3 : real

L1, L2 : literal

B1, B2 : lógico

Para as variáveis declaradas acima, às quais são dados os valores seguintes:

I1 \leftarrow 9

I2 \leftarrow 7

I3 \leftarrow 6

R1 \leftarrow 5.5

R2 \leftarrow 3.92

R3 \leftarrow 6.0

L1 \leftarrow 'Olá'

L2 \leftarrow 'A1'

B1 \leftarrow VERDADEIRO

B2 \leftarrow VERDADEIRO

Determine o resultado da avaliação das expressões abaixo. Se caso não for possível resolver a expressão responda "Expressão Inválida":

(a) $(I1+I2) > 10$ ou $(I1+I2) = (I3+R1)$

(b) $(I1^2) + (I3^2)$

(c) $L1 <> L2$

(d) $(I1+I2) > 10$ ou $(I1+I2) = (I3+R1)$ e $(I1 \geq I3)$ e $(R1 \geq I3)$

(e) $(R1 * I1) / (R3^B1) - B2$



6. Faça o teste de mesa dos algoritmos a seguir:

```
1 algoritmo "horas"
2 var
3   ht:inteiro
4   sm,a,al,b,c,d:real
5 inicio
6   escreval("informe o salario minimo:")
7   leia(sm)
8   escreval("informe as horas trabalhadas:")
9   leia(ht)
10  a <- sm /2
11  b <- ht*a
12  c <- b*3/100
13  d <- b*100
14  escreval("Uma hora trabalhada corresponde a :",a)
15  escreval("o salario bruto é :",b)
16  escreval(" imposto :",c)
17  escreval("salario bruto menos imposto:",d)
18 fimalgoritmo
```

2. Comandos de entrada e saída

Na maioria das vezes, um algoritmo necessita de interatividade com o usuário, ou seja, é preciso representar a troca de informações que ocorrerá entre o computador e o usuário. Essa interatividade é conseguida por meio dos comandos de entrada e saída.

2.1. Comando de entrada de dados

```
leia (<lista-de-variaveis>)
```

Este comando recebe os valores digitados pelos usuários, atribuindo-os às variáveis cujos nomes estão em <lista-de-variáveis> (é respeitada a ordem especificada nesta lista).

2.2. Comando de saída de dados

```
escreva (<lista-de-expressoes>)
```

Este comando escreve o conteúdo de cada uma das expressões que compõem a <lista-de-expressões> nos dispositivos de saída, para que o usuário possa visualizá-lo (as expressões dentro desta lista devem estar separadas por vírgulas).

```
escreval (<lista-de-expressoes>)
```

Idem ao anterior, com a única diferença de que este provoca o pulo de uma linha em seguida.

IMPORTANTE: Observe que, no pseudocódigo, quando queremos enviar uma expressão como saída, esta é colocada dentro de aspas, porém quando queremos enviar o conteúdo de uma variável, esta deverá estar fora de aspas.

Exemplo:

```
algoritmo "soma"  
var  
  n1, n2, total: real  
inicio  
  escreva("Digite n1: ")  
  leia(n1)  
  escreva("Digite n2: ")  
  leia(n2)  
  total <- n1 + n2  
  escreva ("A soma de",n1," +",n2," é", total)  
fimalgoritmo
```

Execução:



Figura 2.1 – Exemplo de execução de comandos de entrada e saída.



EXERCÍCIOS DE FIXAÇÃO

1. Escreva um algoritmo que leia dois números, calcule o produto entre eles, mostre o resultado e os números recebidos.
2. Crie um algoritmo que receba a idade de três pessoas, calcule e mostre a média das idades.
3. Faça um algoritmo que receba dois números, calcule a divisão, a multiplicação, a soma, a diferença entre eles e mostre os resultados.
4. Leia dois valores para as variáveis A e B, efetue a troca dos valores de forma que a variável A passe a ter o valor da variável B e que a variável B, o valor da variável A. Apresentar os valores trocados.
5. Faça um algoritmo que leia uma temperatura fornecida em graus Fahrenheit e a converta para o seu equivalente em graus centígrados.

$$\text{Obs.: } C = \frac{5}{9}(F - 32)$$

6. Uma empresa de energia elétrica calcula o valor da conta de luz de acordo com o consumo em Kw/h. Faça um algoritmo que receba o número da conta, a leitura anterior e a leitura atual, calcule o valor a ser pago, sabendo que a tarifa do Kw/h é de 0.20725. Mostre o número da conta, o valor da conta e o consumo de luz de um usuário.
7. Faça um algoritmo que receba a matrícula e as três notas do aluno, calcule a sua média, sabendo que a primeira nota tem peso dois, a segunda peso três e a terceira peso quatro. Mostre a matrícula e a média do aluno.

EXERCÍCIOS DE FIXAÇÃO



8. Leia as coordenadas de dois pontos no plano cartesiano e imprima a distância entre estes dois pontos. Obs.: fórmula da distância entre dois pontos (x_1, y_1) e $(x_2, y_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.
9. Dado o preço de um produto em Reais, converta este valor para o equivalente em Dólares. O programa deverá ler o preço e a taxa de conversão para o Dólar.
10. Faça um programa para calcular e imprimir o salário bruto a ser recebido por um funcionário em um mês. O algoritmo deverá utilizar os seguintes dados: número de horas que o funcionário trabalhou no mês, valor recebido por hora de trabalho e número de filhos com idade menor do que 14 anos (para adicionar o salário família). Considerar o valor do salário família por filho.
11. [Algoritmos - A. I. Orth] Escreva um algoritmo que lê o código da peça 1, o número de peças 1, o valor unitário da peça 1, o código da peça 2, o número de peças 2, o valor unitário da peça 2 e a percentagem de IPI a ser acrescentado e calcule o valor total a ser pago.
12. [Algoritmos - A. I. Orth] Escreva um algoritmo que lê o número de um vendedor, o seu salário fixo, o total de vendas por ele efetuadas e o percentual que ganha sobre o total de vendas. Calcule o salário total do vendedor. Escreva o número do vendedor e o salário total.
13. [Algoritmos - A. I. Orth] O custo ao consumidor, de um carro novo, é a soma do custo de fábrica com a percentagem do distribuidor e dos impostos (aplicados ao custo de fábrica). Supondo que a percentagem do distribuidor seja de 28% e os impostos de 45%, escreva um algoritmo para ler o custo de fábrica de um carro e escrever o custo ao consumidor.

3. Tomada de decisões

A representação da solução de um problema se faz através da descrição de um conjunto de instruções a serem seguidas. Nos exemplos e exercícios que vimos até agora, os recursos são limitados. Foi possível resolver apenas os problemas puramente sequenciais, ou seja, todas as instruções eram executadas seguindo a ordem do algoritmo (de cima para baixo). Agora começaremos a estudar os desvios condicionais. Desvios condicionais, como o próprio nome diz, permitem que determinados comandos sejam executados ou não, dependendo do resultado de um teste realizado (condição).

– Desvio Condicional Simples

```
se <expressao-logica> entao
    <sequencia-de-comandos>
fimse
```

Este comando analisa a <expressão-lógica>. Se o resultado do teste for VERDADEIRO, todos os comandos da <seqüência-de-comandos> serão executados. Se o resultado for FALSO, estes comandos são desprezados e a execução do algoritmo continua a partir da primeira linha depois do fimse.

A seguir é apresentada uma representação em fluxograma do desvio condicional simples:

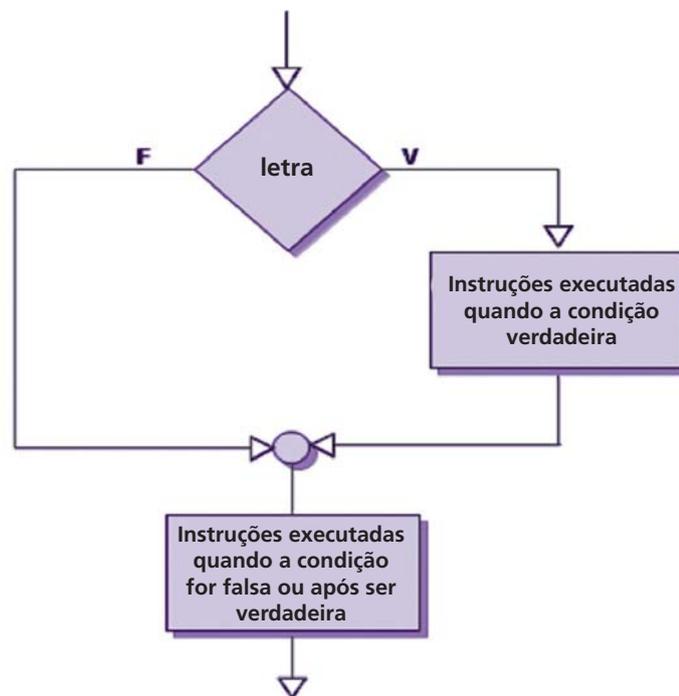


Figura 3.1 – Fluxograma do desvio condicional simples.

Vamos imaginar a seguinte situação: em uma empresa, será solicitado o salário de um determinado funcionário para se calcular seu novo salário, sendo que, se este tiver um salário inferior a R\$ 1000,00, o reajuste será de 8%.

Observe que teremos que testar o valor do salário, para saber se o reajuste será aplicado ou não.

Vamos criar então um pseudocódigo com este objetivo:

```

algoritmo "testa_salario"
var
    salario: REAL
inicio
    escreva("Informe o salário: ")
    leia (salario)
    se (salario < 1000) entao
        salario <- (salario*1.08)
    fimse
    Escreva("Salario = ",salario)
fimalgoritmo

```

Observe que, durante a execução do pseudocódigo, após obtermos, através de uma instrução de entrada de dados, o valor do salário do funcionário, efetuamos um teste "se". A instrução será executada somente se o teste for verdadeiro. A instrução "fimse" termina o bloco de testes. No caso de ser falso, o bloco de teste não será executado.

```
– Desvio Condicional Composto  
  
se <expressao-logica> entao  
    <sequencia-de-comandos-1>  
senao  
    <sequencia-de-comandos-2>  
fimse
```

No desvio condicional composto, se o resultado do teste da <expressao-logica> for VERDADEIRO, todos os comandos da <sequencia-de-comandos-1> serão executados, e a execução continua depois do fimse. Se o resultado for FALSO, todos os comandos da <sequencia-de-comandos-2> (SENAO) serão executados, e a execução continua depois do fimse.

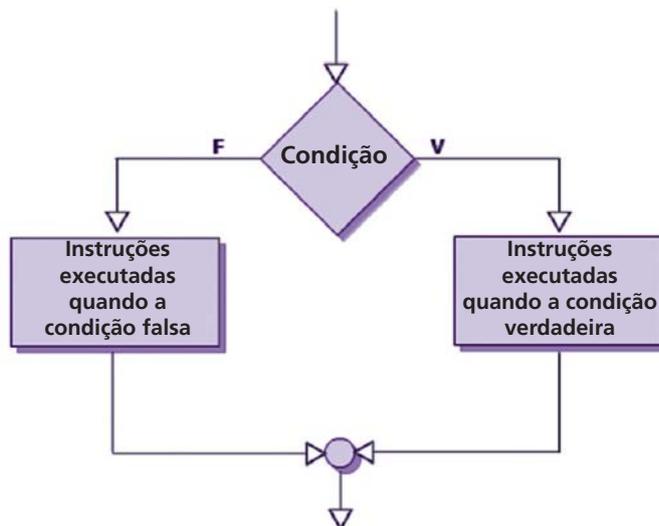


Figura 3.2 – Fluxograma do desvio condicional composto.

Vamos imaginar a seguinte situação: em uma empresa, será solicitado o salário de um determinado funcionário para se calcular seu novo salário, sendo que, se este tiver um salário inferior a R\$ 1000,00, o reajuste será de 8%, caso contrário o reajuste será de 5%.

Vamos criar, então, um pseudocódigo com este objetivo:

```
algoritmo "testa_salario2"  
var  
    salario: REAL  
inicio  
    escreva("Informe o salário: ")  
    leia (salario)  
    se (salario < 1000) entao  
        salario <- (salario*1.08)  
    senao  
        salario <- (salario*1.05)  
    fimse  
    Escreva("Salario = ",salario)  
fimalgoritmo
```

A instrução "se" possui duas condições, uma verdadeira e uma falsa. As instruções que serão executadas no caso de um teste verdadeiro devem estar abaixo da cláusula "entao", já as instruções que serão executadas no caso de um teste falso, devem estar abaixo da cláusula "senao". A instrução "fimse" termina o bloco de testes.

– Desvios condicionais encadeados

```
se <condicao1> entao  
    <instrucoes condicao1 verdadeira>  
senao  
    se <condicao2> entao  
        <instruções condicao2 verdadeira e condicao1 falsa>  
    senao  
        <instrucoes condicao2 e condicao1 falsa>  
    fimse  
fimse
```

Em várias situações, é necessário analisar sucessivamente diversas condições para se realizar uma tomada de decisão, estabelecendo condições dentro de condições. Chamados de aninhamentos ou encadeamentos, esse tipo de estrutura poderá ser constituído de diversos níveis de condições.

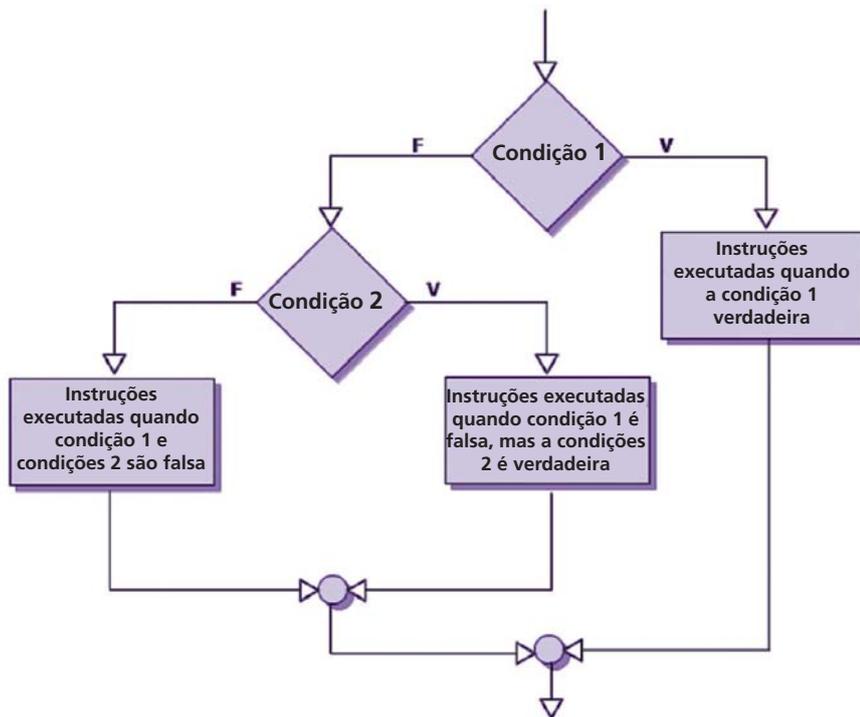


Figura 3.3 – Fluxograma do desvio condicional encadeado.

– Comando de seleção múltipla

```

escolha <expressao-de-selecao>
  caso <exp11>, <exp12>, ..., <exp1n>
    <sequencia-de-comandos-1>
  caso <exp21>, <exp22>, ..., <exp2n>
    <sequencia-de-comandos-2>
  ...
  outrocaso
    <sequencia-de-comandos-extra>
fimescolha
  
```

Para a tomada de uma decisão, existem casos em que não bastam apenas os desvios condicionais (verdadeiro ou falso), mas uma série de testes sobre um mesmo bloco. Este tipo de estrutura é chamado de estrutura de decisão do tipo "ESCOLHA".

No fluxograma, o símbolo que representa cada uma das condições acima é o mesmo que o símbolo que representa a estrutura de decisão. Veja a sintaxe da estrutura de decisão de escolha no fluxograma:

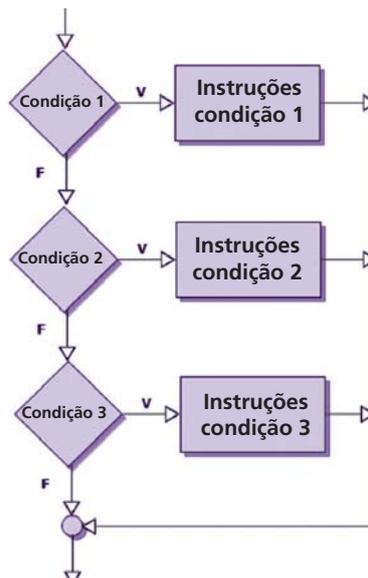


Figura 3.4 – Fluxograma de desvio de seleção múltipla.

Vejamos o exemplo a seguir, que ilustra bem o que faz este comando:

```

algoritmo "times"
var
time: caractere
inicio
  escreva ("Entre com o nome de um time de futebol: ")
  leia (time)
  escolha time
    caso "Flamengo", "Fluminense", "Vasco", "Botafogo"
      escreval ("É um time Carioca.")
    caso "Inter", "Grêmio", "Bagé", "Caxias"
      escreval ("É um time Gaúcho.")
    outrocaso
      escreval ("É de outro estado.")
  fimescolha
fimalgoritmo

```

Observe que temos aqui uma novidade, ou seja, em uma única condição, estamos na verdade realizando vários testes, isto é, verifica se a variável "time" é "Inter" ou "Grêmio" ou "Flamengo" ou "Vasco" ou ... e assim sucessivamente.



EXERCÍCIOS DE FIXAÇÃO

Seleção Simples

1. Escreva um algoritmo que receba um número e mostre a sua metade somente quando ela for maior que cinquenta.
2. Crie um algoritmo que receba um número e mostre o número e o seu dobro somente quando o número for maior que noventa e menor que cem.
3. Faça um algoritmo que receba um número e mostre a sua quinta parte somente quando ela for menor que cinquenta ou maior que mil.
4. Construa um algoritmo que receba um número e mostre o seu sêxtuplo somente quando o resultado não for menor que trezentos.
5. Elabore um algoritmo que receba um número e mostre o número e o sêxtuplo somente quando o número for maior que noventa.
6. Crie um algoritmo que receba dois números e mostre a diferença somente quando o primeiro for maior que o segundo.
7. Faça um algoritmo que receba um número e mostre o número somente quando o número for par.

EXERCÍCIOS DE FIXAÇÃO



8. Escreva um algoritmo que receba um número e mostre o número, se ele estiver entre quinze (inclusive) e quarenta.
9. Construa um algoritmo que receba um número e mostre o número somente se ele estiver entre trinta e duzentos e oitenta e um (inclusive).
10. Faça um algoritmo que receba nome, idade e altura, exiba somente o nome da pessoa com 1,70m e idade acima de 17 anos.
11. Elabore um algoritmo que receba o código, o valor unitário e a quantidade de mercadoria adquirida. Calcule 6% de descontos somente para o total da compra que ultrapassar a R\$ 100,00. Ao final mostre o código, o total a pagar com e sem desconto e o valor do desconto.
12. Escreva um algoritmo que receba o número da conta, o nome, o endereço e o consumo em kw/h, informe o número da conta, o nome e o endereço da conta de luz em que o consumo for inferior a 100 kw/h.
13. Faça um algoritmo que receba nome, turma e três notas do aluno. Calcule a média ponderada considerando: primeira nota peso um, segunda nota peso dois e terceira nota peso três, informar o nome, a turma e a média do aluno que a média for inferior a sete.
14. Construa um algoritmo que receba um número e mostre se o número recebido é ímpar.



EXERCÍCIOS DE FIXAÇÃO

15. Faça um algoritmo que receba o salário de um funcionário, calcule e mostre o novo salário, sabendo-se que este sofreu um aumento de 25%. Este aumento é válido para os funcionários com mais de cinco anos de serviço.
16. Crie um algoritmo que receba o salário de um funcionário, calcule e mostre o salário a receber, sabendo-se que esse funcionário tem gratificação de 5% sobre o salário-base. Pagará imposto de 7% sobre o salário, o funcionário cujo salário mais a gratificação ultrapassar R\$ 1.000,00.

Seleção Composta

17. Crie um algoritmo que leia um número inteiro. Se o número lido for positivo, escreva uma mensagem indicando se ele é par ou ímpar.
18. Construa um algoritmo que receba dois números e mostre quando o primeiro for maior e quando for menor que o segundo.
19. Elabore um algoritmo que receba dois números e mostre quando o primeiro for maior, quando for menor e quando for igual ao segundo.
20. Faça um algoritmo que receba três números e mostre quando o número estiver na situação de maior número digitado.
21. Escreva um algoritmo que receba dois números e informe a diferença do maior pelo menor.

EXERCÍCIOS DE FIXAÇÃO



22. Faça um algoritmo que receba a matrícula e duas notas do aluno. Calcular a média e mostrar a matrícula do aluno com as seguintes mensagens de acordo com os dados a seguir:

MÉDIA	MENSAGEM
Média > 7,0	"Aluno Aprovado"
Média = 7,0	"Aluno em Recuperação"
Média < 7,0	"Aluno Reprovado"

23. Escreva um algoritmo que receba o código, o preço e o tamanho da camiseta. Sabe-se que de acordo com o tamanho há um desconto conforme tabela abaixo. Informar o código, e o valor do desconto.

TAMANHO	DESCONTO
"G"	10%
"M"	20%
"P"	30%

24. Faça um algoritmo que receba o nome e o salário do funcionário, calcule os descontos conforme a tabela e informe o nome, o salário a receber e os descontos (separados) do funcionário.

SALÁRIO	INSS	IR
Até 500,00	3,00%	2,00%
De 500,01 até 950,00	5,00%	4,00%
Acima de 950,00	7,00%	6,00%



EXERCÍCIOS DE FIXAÇÃO

25. Construa um algoritmo que receba o código e o total de vendas do vendedor, calcule a comissão conforme a tabela e informe o código e a comissão do vendedor.

TOTAL DAS VENDAS	COMISSÃO
Até 100,00	0,00%
Acima 100,00 até 350,00	5,00%
Acima 350,00	10,00%

26. Faça um algoritmo que receba a matrícula e o salário do funcionário, calcule a gratificação conforme a tabela e informe a matrícula, salário a receber e o valor da gratificação do funcionário.

SALÁRIO	GRATIFICAÇÃO
Até 500,00	15,00%
De 500,01 até 1.000,00	10,00%
Acima de 1.000,00	5,00%

27. Faça um algoritmo que receba um número e informe quando ele for maior e menor que o número 100.
28. Crie um algoritmo que receba dois pesos e mostre quando o primeiro peso for maior que o segundo.
29. Construa um algoritmo que receba duas idades e mostre quando a primeira idade for maior, quando for igual e quando for menor que a segunda idade (mostrar separadamente)

EXERCÍCIOS DE FIXAÇÃO



30. Faça um algoritmo que receba três alturas e informe quando a altura que estiver na situação de maior altura.
31. Escreva um algoritmo que receba o nome e o sexo de uma pessoa. Informar o nome quando masculino e quando feminino.
32. Elabore um algoritmo que receba a matrícula e o salário do funcionário. Sabe-se que, de acordo com o salário, há uma gratificação conforme tabela abaixo. Informar a matrícula e a gratificação.

SALÁRIO	GRATIFICAÇÃO
Acima de R\$ 300,00	5%
Abaixo de R\$ 300,00	10%
R\$ 300,00	6%



4. Estruturas de repetição

Quando se torna necessário, na programação, a repetição de um ou vários trechos em vários momentos de um algoritmo em um determinado número de vezes, tem-se uma estrutura de repetição. Em linguagem de programação, as estruturas de repetição são conhecidas como laços de repetição (loopings) ou malhas de repetição. A estrutura de repetição, assim como a de decisão, envolve sempre a avaliação de uma condição.

Para que se possam abstrair estas estruturas, considere um algoritmo que deva executar “n” vezes um determinado trecho de instruções. Com o conhecimento que estudamos até o presente momento, com certeza você irá escrever o mesmo trecho, repetindo-o o número de vezes necessárias. Em vez de se fazer isto, existem estruturas de repetição apropriadas para executar um determinado trecho de instruções, o número de vezes necessário, fazendo com que o algoritmo passe a ter um tamanho reduzido, de fácil alteração na quantidade de repetições e com melhor compreensão.

Trabalhamos basicamente com dois tipos de estruturas de repetição:

Repetição determinada: quando se tem, de forma prévia, o número de vezes que uma determinada sequência de comandos será executado;

Repetição indeterminada: aqueles que não possuem um número pré-determinado de vezes que a estrutura de comandos será executada, porém este número estará vinculado a uma determinada condição.

4.1. Repetição determinada

```
para <variavel> de <valor-inicial> ate <valor-limite>  
[passo <incremento>] faca  
  <sequencia-de-comandos>  
fimpara
```

Esta estrutura repete uma sequência de comandos um determinado número de vezes, tendo o seu funcionamento controlado por uma variável denominada contador.

Onde:

Tabela 4.1 – Elementos do comando de repetição determinada.

<variavel>	É a variável contadora que controla o número de repetições do laço. Deve ser necessariamente uma variável do tipo inteiro, como todas as expressões deste comando.
<valor-inicial>	É uma expressão que especifica o valor de inicialização da variável contadora antes da primeira repetição do laço.
<valor-limite>	É uma expressão que especifica o valor máximo que a variável contadora pode alcançar.
<incremento>	É opcional. Quando presente, precedida pela palavra "passo", é uma expressão que especifica o incremento que será acrescentado à variável contadora em cada repetição do laço. Quando esta opção não é utilizada, o valor padrão de <incremento> é 1. Vale a pena ter em conta que também é possível especificar valores negativos para <incremento>. Por outro lado, se a avaliação da expressão <incremento > resultar em valor nulo, a execução do algoritmo será interrompida, com a impressão de uma mensagem de erro.
fimpara	Indica o fim da sequência de comandos a serem repetidos. Cada vez que o programa chega neste ponto, é acrescentado à variável contadora o valor de <incremento >, e comparado a <valor-limite >. Se for menor ou igual (ou maior ou igual, quando <incremento > for negativo), a sequência de comandos será executada mais uma vez; caso contrário, a execução prosseguirá a partir do primeiro comando que esteja após o fimpara.

A seguir é apresentada uma representação em fluxograma da repetição determinada:

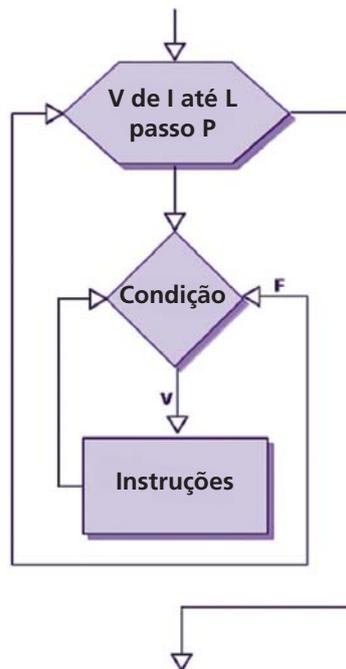


Figura 4.1 – Fluxograma do comando de repetição determinada.

A lógica de trabalho da repetição determinada é apresentada a seguir:

1. No início da leitura do código/fluxo, o <valor-inicial> é atribuído à <variavel>;
2. O valor da <variavel> é comparado ao <valor-limite>;
3. Se o valor da <variavel> for maior que o <valor-limite>, a sequência de comandos que faz parte do laço não é executado e o controle do processo passa para a próxima linha após o final do laço (fimpara) ou para o símbolo ao lado (no caso de fluxograma);
4. Se o valor da <variavel> for menor ou igual ao do <valor-limite>, a sequência de comandos é executada e, ao final do último comando, o valor do <incremento> é adicionado à <variavel> e retorna-se à comparação entre <variavel> e <valor-limite>, repetindo-se todo o processo anterior.

Vale lembrar que o incremento não é de declaração obrigatória, e, caso não seja declarado, assume automaticamente o valor 1.

Exemplo:

```
algoritmo "Números de 1 a 10"  
var  
  v: inteiro  
inicio  
  para v de 1 ate 10 faca  
    escreval (v)  
  fimpara  
fimalgoritmo
```

Importante:

Se, logo no início da primeira repetição, o <valor-inicial> for maior que o <valor-limite> (ou menor, quando o <incremento> for negativo), o laço não será executado nenhuma vez. O exemplo a seguir não imprime nada.

```
algoritmo "Numeros de 10 a 1 (não funciona)"  
var  
  v: inteiro  
inicio  
  para v de 10 ate 1 faca  
    escreval (v)  
  fimpara  
fimalgoritmo
```

Este outro exemplo, no entanto, funcionará por causa do passo -1:

```
algoritmo "Numeros de 10 a 1 (este funciona)"  
var  
  v: inteiro  
inicio  
  para v de 10 ate 1 passo -1 faca  
    escreval (v)  
  fimpara  
fimalgoritmo
```

4.2. Repetição Indeterminada

Este tipo de estrutura, também chamado de laços condicionais, são aqueles cujo conjunto de comandos em seu interior é executado até que uma determinada condição seja satisfeita. É utilizado para repetir “N” vezes uma ou mais instruções. Tendo como vantagem o fato de não ser necessário o conhecimento prévio do número de repetições.

Existem dois tipos de estruturas de repetição indeterminada:

- Repetição Indeterminada com Validação Inicial.
- Repetição Indeterminada com Validação Final.

4.2.1 Repetição Indeterminada com Validação Inicial

```
enquanto <expressao-logica> faca  
    <sequencia-de-comandos>  
fimenquanto
```

Esta estrutura repete uma sequência de comandos enquanto uma determinada condição (especificada através de uma expressão lógica) for satisfeita.

Tabela 4.2 – Fluxograma do desvio condicional encadeados.

<code><expressao-logica></code>	Esta expressão é avaliada antes de cada repetição do laço. Quando seu resultado for VERDADEIRO, <code><sequencia-de-comandos></code> é executada.
<code>fimenquanto</code>	Indica o fim da <code><sequencia-de-comandos></code> que será repetida. Cada vez que a execução atinge este ponto, volta-se ao início do laço para que <code><expressao-logica></code> seja avaliada novamente. Se o resultado desta avaliação for VERDADEIRO, a <code><sequencia-de-comandos></code> será executada mais uma vez; caso contrário, a execução prosseguirá a partir do primeiro comando após <code>fimenquanto</code> .

A seguir é apresentada uma representação em fluxograma da repetição indeterminada com validação inicial:

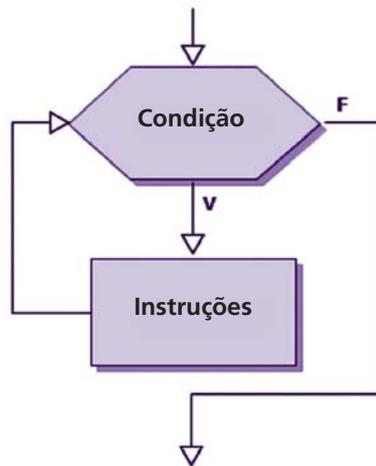


Figura 4.2 – Fluxograma do comando de repetição indeterminada.

A sintaxe de execução deste é:

- No início da execução do enquanto, a condição é testada;
- Se o resultado do teste for verdadeiro, a sequência de comandos é executada e ao término desta, retorna-se ao teste da condição;
- Sendo a condição for falsa, o controle da aplicação passa para a próxima linha após o `fimenquanto`.

Exemplo:

```
algoritmo "Números de 1 a 10 (com enquanto...faca)"  
var  
  v: inteiro  
inicio  
  v <- 1  
  enquanto v <= 10 faca  
    escreva (v)  
    v <- v + 1  
  fimenquanto  
finalgoritmo
```

4.2.2 Repetição Indeterminada com Validação Final

```
repita  
  <sequencia-de-comandos>  
ate <expressao-logica>
```

Esta estrutura repete uma sequência de comandos até que uma determinada condição (especificada por meio de uma expressão lógica) seja satisfeita.

Tabela 4.3 – Elementos do comando de repetição indeterminada.

repita	Indica o início do laço.
até <expressao-logica>	Indica o fim da <sequencia-de-comandos> a serem repetidos. Cada vez que o programa chega neste ponto, a <expressao-logica> é avaliada: se seu resultado for FALSO, os comandos presentes entre esta linha e a linha repita são executados; caso contrário, a execução prosseguirá a partir do primeiro comando após esta linha.

A seguir é apresentada uma representação em fluxograma da repetição indeterminada com validação final.

A sintaxe de execução deste é:

- Executa a primeira iteração do laço;
- Se o resultado do teste for verdadeiro, a sequência de comandos é executada e ao término desta, retorna-se ao teste da condição;
- Se a condição for falsa, executa-se nova iteração do laço, se a condição de parada for verdadeira, encerra-se a execução do laço.

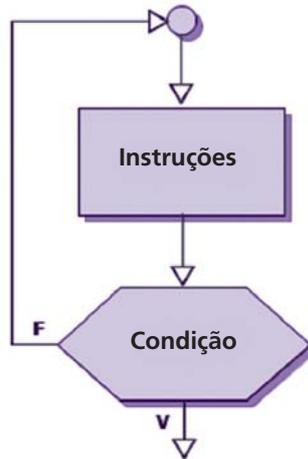


Figura 4.3 – Fluxograma do comando de repetição indeterminada com validação final.

Exemplo:

```

algoritmo "Números de 1 a 10 (com repita)"
var
  v: inteiro
inicio
  v <- 1
  repita
    escreval (v)
    v <- v + 1
  ate v > 10
fimalgoritmo
  
```



EXERCÍCIOS DE FIXAÇÃO

Repetição determinada

1. Construa um algoritmo que receba trinta números e mostre a soma total dos números recebidos.
2. Construa um algoritmo que receba cinquenta números e mostre a média dos números que foram digitados.
3. Construa um algoritmo que receba a idade de cem pessoas e mostre a média das idades destas pessoas.
4. Construa um algoritmo que leia cem números e mostre qual o maior número que foi lido.
5. Construa um algoritmo que leia cinquenta números e mostre qual o menor número lido.
6. Construa um algoritmo que leia cento e cinquenta números e mostre qual o maior e o menor número lido.
7. Construa um algoritmo que receba cem números e informe a média e a soma entre os números positivos.
8. Construa um algoritmo que receba quinze números quaisquer e informe qual o maior e o menor entre os números que foram lidos.

EXERCÍCIOS DE FIXAÇÃO



9. Escreva um algoritmo que receba 100 números, e conte quantos deles estão no intervalo $[10, 20]$ e quantos deles estão fora do intervalo, escrevendo estas informações.
10. Faça um algoritmo que receba o peso, a idade e a altura de cem pessoas, calcule e informe os valores de: maior peso, menor peso, maior altura, menor altura, maior idade e menor idade deste grupo.
11. Escrever um algoritmo que leia 50 números e informe quantos destes valores são negativos.
12. Uma loja tem 150 clientes cadastrados e deseja mandar uma correspondência a cada um deles anunciando um bônus especial. Escreva um algoritmo que leia o nome, o endereço do cliente e o valor de suas compras e calcule um bônus de 10% se o valor das compras for menor ou igual a R\$ 500.000,00 e de 15 %, se superior a este valor.
13. Faça um algoritmo que receba o salário-base dos 1.200 funcionários de uma fábrica e calcule os descontos com vale transporte (vt) 2% e vale refeição (vr) 3%. Mostrar o total dos descontos efetuados separadamente.
14. Faça um algoritmo que receba o número do apartamento e o consumo em kw/h dos setenta e dois apartamentos deste edifício. Informe os apartamentos com o consumo inferior ou igual a 100 kw/h (inclusive) e os que ultrapassaram este consumo.
15. Faça um algoritmo que receba o tamanho de 500 camisetas existente no almoxarifado e ao final informe quantas camisetas de cada tamanho P, M, G, GG.



EXERCÍCIOS DE FIXAÇÃO

Repetição Indeterminada

16. Elabore um algoritmo para somar os números recebidos. O algoritmo encerra quando digitado o número zero.
17. Construa um algoritmo que leia vários números e mostre quantos números foram lidos. O algoritmo encerra quando digitado o número zero.
18. Faça um algoritmo que receba vários números e mostre a média dos números recebidos. O final é conhecido pelo número zero.
19. Escreva um algoritmo que receba vários números, calcule e informe a média, a soma e a quantidade de números recebidos. O algoritmo encerra quando digitado o número zero.
20. Faça um algoritmo que receba a sigla da cidade de origem de um grupo de pessoas, ao final informe quantas foram digitadas das cidades do Rio de Janeiro, Belo Horizonte e Santa Catarina (separadamente). O algoritmo encerra quando digitado "fim".
21. Elabore um algoritmo que receba o sexo dos alunos de um determinado colégio e informe o número de alunas e alunos separados e o número de digitações inválidas. O algoritmo encerra quando digitado "fim".
22. Construa um algoritmo que receba o estado civil (c | s | v) de um grupo de pessoas, calcule e informe a quantidade de solteiros, casados, viúvos, o número de digitações válidas e inválidas. O algoritmo encerra quando digitado "fim"

EXERCÍCIOS DE FIXAÇÃO



23. Foi feita uma pesquisa entre os habitantes de uma região. Foi coletado o salário de cada habitante. Calcule e informe:

a) a média de salário;

b) o maior e o menor salário.

Encerre a entrada de dados quando for digitado o salário zero.

24. Foi realizada uma pesquisa de algumas características físicas da população de certa região, a qual coletou os seguintes dados referentes a cada habitante para serem analisados:

- sexo (masculino e feminino)

- olhos (claros e pretos)

- cabelos (louros, castanhos ou pretos)

Faça um algoritmo que calcule e informe a quantidade de pessoas do sexo feminino, com olhos claros e cabelos louros. O final do conjunto de habitantes é reconhecido pelo sexo em branco " " .

25. Faça um algoritmo que receba a idade e a altura de um conjunto de pessoas. Calcule e informe a média de altura e de idade das pessoas. Para encerrar a entrada de dados, digite a idade igual a 0.



EXERCÍCIOS DE FIXAÇÃO

26. Construa um algoritmo que receba o peso dos bois de uma determinada fazenda, calcule e informe a quantidade de bois, o maior peso e o menor peso. Encerrar quando for digitado o peso 0.
27. Crie um algoritmo que receba a altura de um grupo de pessoas, calcule e informe a maior altura e a menor altura.
28. Foi feita uma pesquisa de audiência de canal de televisão em várias casas de uma determinada cidade. Será fornecido o número da casa e o do canal de televisão que estiver ligado no momento, caso a televisão esteja desligada é fornecido o número zero, bem como para a residência fechada. Calcular e informar: a quantidade de residências fechadas, televisões desligadas e a quantidade de residências com a televisão sintonizada no canal dois. Encerrar a pesquisa quando for fornecido para o número da residência um valor negativo.



5. Estruturas de dados compostas

As estruturas de dados compostas são assim chamadas, pois permitem armazenar vários dados de um determinado tipo, se variáveis indexadas forem utilizadas. Estas variáveis são de uso frequente na programação convencional. Variável indexada é um conjunto de variáveis do mesmo tipo, referenciadas pelo mesmo nome e que armazenam valores distintos.

A sua distinção se dá pela divisão da variável, que possui uma posição de armazenamento dentro de seu conjunto, sendo que a alteração ou a leitura do seu valor é realizada por meio de índices, daí o nome indexado. Existem dois tipos de variáveis indexadas:

- Vetor: quando a variável indexada possui um único índice (uma dimensão);
- Matriz: quando a variável indexada possui dois índices (duas dimensões – X, Y/ Coluna, Linha).

5.1. Vetores

O número de índices dentro de uma variável é denominado dimensão. Vetor é um conjunto de variáveis, no qual cada uma pode armazenar uma informação diferente, mas todas compartilham o mesmo nome. São associados índices a esse nome, que representam as posições do vetor, permitindo, assim, individualizar os elementos do conjunto.

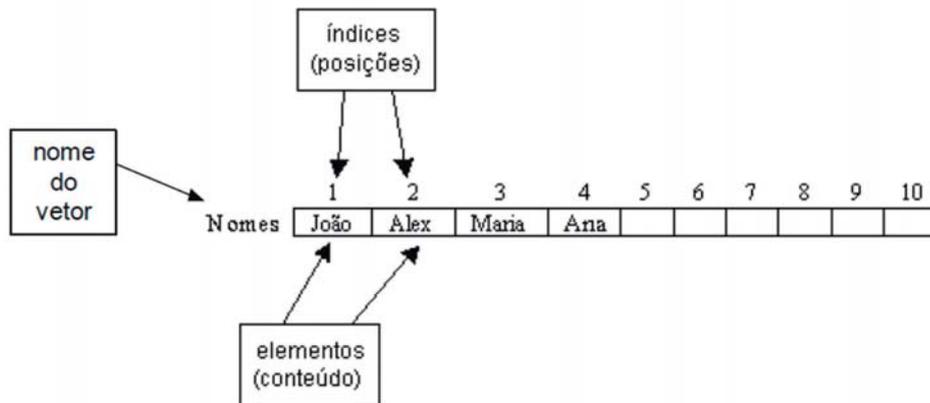


Figura 5.1 – Exemplo de um vetor.

Nesta figura, podemos perceber que um vetor é composto por várias “células” que podem armazenar um determinado tipo de dado. Estas “células” podem ser acessadas por meio de um endereço que as identifica e ordena, chamado índice.

Para se definir uma variável indexada, utilizamos a seguinte sintaxe:

<variaveis>: vetor["<intervalos>"] de <tipo-de-dado>

Veja a seguir alguns exemplos de definição de variáveis indexadas:

vet: vetor [1..10] de real

nomes: vetor [1..5] de caracter

No exemplo acima, criamos, na verdade, cinco variáveis “nomes”, porém elas estão divididas da seguinte forma:

nomes[1], nomes[2], nomes[3], nomes[4], nomes[5].

Cada uma das dimensões poderá armazenar um valor diferenciado para um ou mais caracteres. Sendo assim, para acessarmos um dos elementos do vetor, temos que indicar qual elemento queremos obter, utilizando o nome da variável e o índice da posição conforme o código a seguir.

```

algoritmo "vetores"
var
  nomes: vetor [1..10] de caracter
  j:inteiro
inicio
  // lendo os valores do vetor nome
  para j de 1 ate 10 faca
    leia (nomes[j])
  fimpara

  // mostrando o valor da posição 3
  escreval(nomes[3])

  // mostrando o valor da posição 9
  escreval(nomes[9])
finalgoritmo

```

5.2. Matrizes

Uma matriz corresponde a vários vetores colocados um abaixo do outro, formando, assim, uma tabela composta por várias linhas e várias colunas.

Veja a seguir alguns exemplos de definição de variáveis indexadas:

```

matriz: vetor [0..4,0..10] de inteiro
codigoProduto: vetor [1..3,1..10] de inteiro

```

No exemplo anterior, teremos a matriz `codigoProduto`, dividida nas seguintes dimensões:

Tabela 5.1 – Exemplo de uma matriz.

CodigoProduto[1,1]	CodigoProduto[2,1]	CodigoProduto[3,1]
CodigoProduto[1,2]	CodigoProduto[2,2]	CodigoProduto[3,2]
CodigoProduto[1,3]	CodigoProduto[2,3]	CodigoProduto[3,3]
CodigoProduto[1,4]	CodigoProduto[2,4]	CodigoProduto[3,4]
CodigoProduto[1,5]	CodigoProduto[2,5]	CodigoProduto[3,5]

CodigoProduto[1,6]	CodigoProduto[2,6]	CodigoProduto[3,6]
CodigoProduto[1,7]	CodigoProduto[2,7]	CodigoProduto[3,7]
CodigoProduto[1,8]	CodigoProduto[2,8]	CodigoProduto[3,8]
CodigoProduto[1,9]	CodigoProduto[2,9]	CodigoProduto[3,9]
CodigoProduto[1,10]	CodigoProduto[2,10]	CodigoProduto[3,10]

Quando estamos trabalhando com variáveis indexadas, temos que obrigatoriamente especificar o índice da variável na qual queremos trabalhar. No exemplo acima, temos que indicar a coluna e a linha para acessarmos o valor de uma “célula” da tabela. Veja o exemplo a seguir:

```

algoritmo "matrizes"
var
  codigoproduto: vetor[1..3,1..10] de inteiro
  j,x:inteiro
inicio
  // lendo os valores da matriz
  para j de 1 ate 3 passo 1 faca
    para x de 1 ate 10 passo 1 faca
      leia (codigoproduto[j,x])
    fimpara
  fimpara

  // mostrando o valor da posição 2,5
  escreval(codigoproduto[2,5])

  // mostrando o valor da posição 1,8
  escreval(codigoproduto[1,8])
fimalgoritmo

```

Para validarmos o entendimento sobre variáveis indexadas, vamos fazer um pequeno exercício. Dada a seguinte variável indexada bidimensional A, ou seja, a matriz A:

Tabela 5.2 – Exemplo de uma matriz com valores.

1	175	225	10	9000	3,7	4,75
2	9,8	100	363	432	156	18
3	40	301	30,2	6381	1	0
4	402	4211	7213	992	442	7321

5	21	3	2	1	9000	2000
	1	2	3	4	5	6

1. Quantos elementos fazem parte do conjunto?
2. Qual o conteúdo do elemento identificado por A [4, 5] ?
3. Qual o conteúdo de X após a execução do comando $X := A [3, 2] + A [5, 1]$?
4. O que aconteceria caso fosse referenciado o elemento A [6, 2] no programa?
5. Somar os elementos da quarta coluna: $A [1, 4] + A [2, 4] + A [3, 4] + A [4, 4] + A [5, 4]$.
6. Somar os elementos da terceira linha: $(A [3, 1] + A [3, 2] + A [3, 3] + A [3, 4])$.

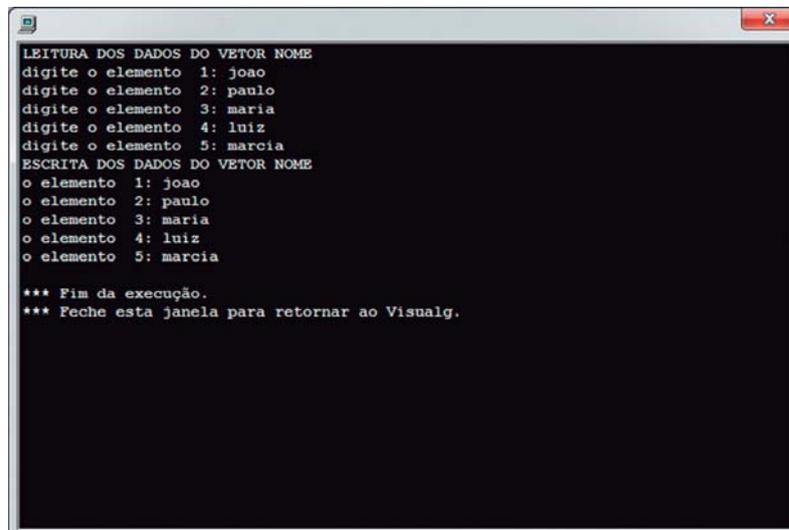
No próximo exemplo, temos um algoritmo no qual iremos preencher os valores para a variável "nome" em suas diversas dimensões e em seguida listá-los:

```

1 algoritmo "Nomes"
2
3 var
4   nomes: vetor [1..5] de caracter
5   posicao: inteiro
6 inicio
7   // utilizamos um laço de repetição para
8   // ler os dados do vetor nomes
9   escreval("LEITURA DOS DADOS DO VETOR NOME")
10  para posicao de 1 ate 5 faca
11    escreva("digite o elemento ",posicao," : ")
12    leia (nomes[posicao])
13  fimpara
14
15  // utilizamos um laço de repetição para
16  // mostrar os dados do vetor nomes
17  escreval("ESCRITA DOS DADOS DO VETOR NOME")
18  para posicao de 1 ate 5 faca
19    escreva("o elemento ",posicao," : ")
20    escreval (nomes[posicao])
21  fimpara
22
23 finalgoritmo

```

Veja abaixo o resultado da execução do código anterior:



```
LEITURA DOS DADOS DO VETOR NOME
digite o elemento 1: joao
digite o elemento 2: paulo
digite o elemento 3: maria
digite o elemento 4: luiz
digite o elemento 5: marcia
ESCRITA DOS DADOS DO VETOR NOME
o elemento 1: joao
o elemento 2: paulo
o elemento 3: maria
o elemento 4: luiz
o elemento 5: marcia

*** Fim da execução.
*** Feche esta janela para retornar ao Visualg.
```

Figura 5.2 – Resultado da execução do algoritmo “nomes”.

EXERCÍCIOS DE FIXAÇÃO



1. Escreva um pseudocódigo que irá ler e armazenar em variável indexada 10 números e após o final da leitura, irá exibir a soma dos 10 números armazenados nas dez posições da variável indexada.
2. Escreva um pseudocódigo que irá ler e armazenar em variável indexada 50 números e após o final de leitura, irá apresentar o número da posição e o valor armazenado na posição de variável indexada que conterà o maior valor lido.

5.3. Algoritmos de pesquisa e ordenação de dados

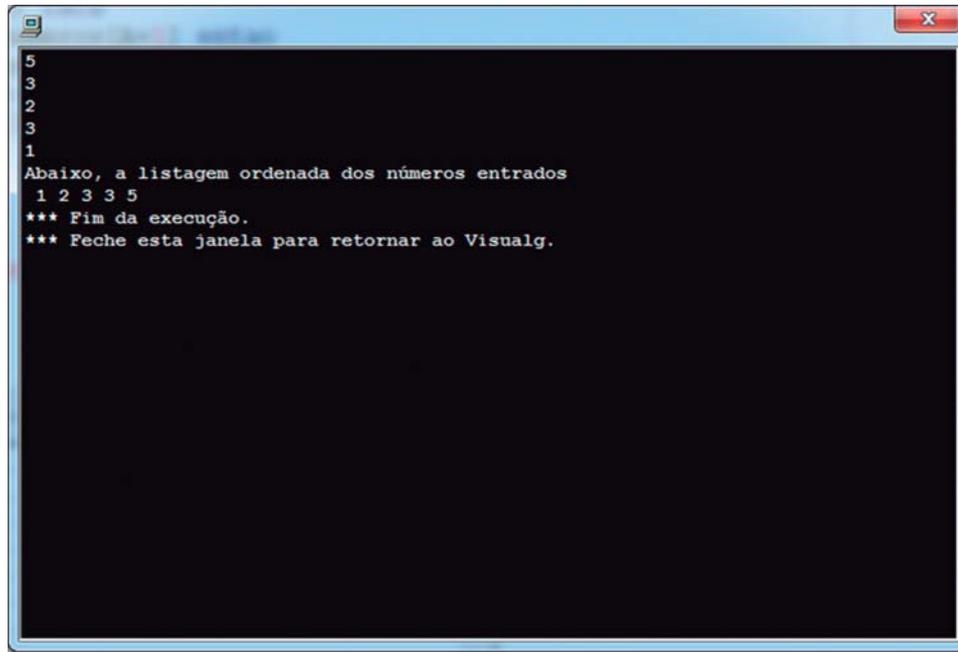
Conforme já foi dito anteriormente, quando trabalharmos com variáveis indexadas, poderemos entre outros criar “índices”, ou seja, ordenar os vetores para que eles sejam apresentados em uma determinada ordem. Este procedimento também é conhecido, em sua forma mais simples, como “Bolha de Classificação”. Sua lógica consiste na leitura de todos os elementos de um vetor, comparando os valores de seus elementos vizinhos, e neste momento é empregada a seguinte sequência:

1. Serão realizadas menos varreduras que o total de elementos do vetor;
2. Na primeira varredura, verificamos que o elemento do vetor já se encontra em seu devido lugar;
3. Na segunda varredura, o procedimento é análogo à primeira varredura e vai até o último elemento;
4. Ao final do procedimento, o vetor estará classificado segundo o critério escolhido.

Vamos à prática. No exemplo a seguir, iremos realizar a leitura de 5 números e em seguida classificá-los em ordem crescente:

```
1 algoritmo "classificacao"
2 var
3   numeros: vetor[1..5] de inteiro
4   Aux,A,B: INTEIRO
5
6 inicio
7   para A de 1 ate 5 faca
8     leia(numeros[A])
9   fimpara
10  B<-5
11  enquanto B > 1 faca
12    para A de 1 ate (B-1) faca
13      se numeros[A] > numeros[A+1] entao
14        Aux<-numeros[A]
15        numeros[A]<-numeros[A+1]
16        numeros[A+1]<-Aux
17      fimse
18    fimpara
19    B<-B-1
20  fimenquanto
21  escreval("Abaixo, a listagem ordenada dos números entrados")
22  para A de 1 ate 5 faca
23    escreva(numeros[A])
24  fimpara
25 fimalgoritmo
```

Veja abaixo o resultado da execução do código anterior:



```
5
3
2
3
1
Abaixo, a listagem ordenada dos números entrados
1 2 3 3 5
*** Fim da execução.
*** Feche esta janela para retornar ao Visualg.
```

Figura 5.3 – Resultado da execução do algoritmo “classificação”.

Outro processo que poderá ser utilizado em programação é a pesquisa sequencial, ou seja, serão verificados todos os componentes de um vetor, para identificar se nestes estão armazenados um determinado valor.

Exemplo:

Iremos efetuar a leitura de 100 nomes, em seguida, será solicitado um nome qualquer e, iremos verificar se, entre os 100 nomes entrados, existe o nome que foi solicitado na pesquisa. Veja este algoritmo a seguir:

```

1 algoritmo "pesquisa"
2 var
3   nome: vetor[1..10] de caracter
4   pesquisado : caracter
5   contador : inteiro
6   encontrado : logico
7
8 Inicio
9   escreval("digite dez caracteres")
10  para contador de 1 ate 10 faca
11    leia (nome[contador])
12  fimpara
13  escreval("digite caracter a ser pesquisado")
14  leia (pesquisado)
15  contador<-1
16  encontrado<-FALSO
17  enquanto ((contador<11) e (Encontrado = FALSO)) faca
18    se (nome[contador]=pesquisado) entao
19      Encontrado <- VERDADEIRO
20    senao
21      Contador<-contador+1
22    fimse
23  fimenquanto
24  se (encontrado = VERDADEIRO) entao
25    escreval("O valor está contido no banco de dados")
26  senao
27    escreval("O valor não existe no banco de dados")
28  fimse
29
30 fimalgoritmo

```

Veja abaixo o resultado da execução do código anterior:

```

digite dez caracteres
3
e
r
5
g
d
c
t
u
n
5
digite caracter a ser pesquisado
g
O valor está contido no banco de dados

*** Fim da execução.
*** Feche esta janela para retornar ao Visualg.

```

Figura 5.4 – Resultado da execução do algoritmo “pesquisa”.

EXERCÍCIOS DE FIXAÇÃO



1. Faça um algoritmo que carregue um vetor com 150 números reais, determine e mostre o seu elemento máximo (maior valor) e a posição na qual ele se encontra.
2. Faça um algoritmo que carregue um vetor com 150 números inteiros, determine e mostre o seu elemento mínimo (menor valor) e a posição na qual ele se encontra.
3. Escreva um algoritmo que armazene em um vetor os 100 primeiros números ímpares. Após isso, o algoritmo deve imprimir todos os valores armazenados.
4. Escreva um algoritmo que armazene em um vetor o quadrado dos números ímpares no intervalo fechado de 1 a 20. Após isso, o algoritmo deve imprimir todos os valores armazenados.
5. Escreva um algoritmo que receba dez números inteiros e armazene em um vetor a metade de cada número. Após isso, o algoritmo deve mostrar todos os valores armazenados.
6. Escreva um algoritmo que receba dez números do usuário e armazene em um vetor o quadrado de cada número. Após isso, o algoritmo deve imprimir todos os valores armazenados.
7. Escreva um algoritmo que receba e armazene, em um vetor, quinze números. Ao final, mostre todos os números pares do vetor e sua respectiva posição.
8. Fazer um algoritmo em que sejam carregados dois vetores de 25 números inteiros.



EXERCÍCIOS DE FIXAÇÃO

- a. Intercale os elementos destes dois vetores formando um novo vetor de 50 elementos;
 - b. exiba os números do novo vetor.
9. Fazer um algoritmo que:
- a. leia um vetor A com 30 valores numéricos;
 - b. leia um outro vetor B com 30 valores numéricos;
 - c. leia o valor de uma variável X;
 - d. verifique qual o elemento de A é igual a X;
 - e. exiba o elemento de B de posição correspondente à do elemento de A igual a X, se existir.

Dadas as variáveis indexadas A e B abaixo:

A			
7	8	4	9
2	1	7	3

B			
6	9	11	15
32	19	3	4

10. Calcular o conjunto $C = A + B$.
11. Calcular o produto dos elementos $A[1, 2]$, $A[3, 1]$, $B[2, 2]$, $B[4, 2]$.

EXERCÍCIOS DE FIXAÇÃO



12. Construa um algoritmo que carregue 100 números inteiros positivos (deve ser feito o teste para garantir a não inclusão de números inválidos – utilize o laço “repita até” neste teste). O programa deve calcular a média dos números do vetor e, ao final, mostrar todos os números que são maiores que a média.
13. Faça um algoritmo que carregue um vetor com 50 nomes, ordene e mostre o vetor ordenado.
14. Faça um algoritmo que carregue um vetor com 50 nomes, depois solicite ao usuário um nome e verifique se este existe no vetor ou não (mostrar o nome e a posição no vetor, caso exista, e uma mensagem indicativa, caso não exista). A solicitação deve ser repetida até que o usuário digite a palavra “fim”.
15. Faça um algoritmo que carregue um vetor com 100 números inteiros, verifique cada número, se este for par, multiplique por 3 e atualize o seu valor no vetor. Ao final, mostre todos os elementos do vetor.
16. Faça um algoritmo que carregue um vetor com 150 números inteiros, gere um segundo vetor com todos os números do primeiro vetor multiplicados por 2. Mostre ambos os vetores.
17. Faça um algoritmo que carregue uma matriz de dimensão 5x5, calcule e mostre a soma de todos os elementos e uma listagem com cada um dos elementos da matriz.
18. Faça um algoritmo que carregue duas matrizes de dimensão 6x6. Gere uma terceira matriz com a soma das duas anteriores e mostre seus valores.



EXERCÍCIOS DE FIXAÇÃO

19. Faça um algoritmo que carregue um vetor com 60 posições, calcule e mostre a soma de todos os elementos das posições pares e ímpares do vetor.
20. Faça um algoritmo que carregue uma matriz de dimensão 10x10, calcule e mostre a soma de cada uma das linhas e de cada uma das colunas da matriz.
21. Faça um algoritmo que: o usuário preencha com quaisquer valores inteiros um vetor de trinta posições e que calcule a média deles. Os valores menores que a média devem ser copiados em outro vetor, os quais devem ficar em ordem crescente. Liste os dados do primeiro vetor e tantos quantos houver no segundo vetor. Use mensagens claras para o usuário tomar conhecimento dos passos sendo executados.
22. Faça um algoritmo que: o usuário preencha com quaisquer valores inteiros um vetor de trinta posições e que calcule a média deles. Os valores maiores que a média devem ser copiados em outro vetor, os quais devem ficar em ordem crescente. Liste os dados do primeiro vetor e tantos quantos houver no segundo vetor. Use mensagens claras para o usuário tomar conhecimento dos passos sendo executados. O fluxograma e o português devem ser entregues passados a limpo. Consulte apenas o seu material.
23. Faça um algoritmo que: o usuário insere em um vetor "N" 40 números negativos, seguido da digitação de outros 40 números maiores que zero em um "P" (faça o teste para permitir somente a entrada de valores válidos). Faça um vetor "V", de tal forma que cada elemento seu seja a soma dos elementos de "P" e "N", de mesma i -ésima posição, e caso a soma seja negativa transforme-a em positiva. Liste "V", mostre a média, o maior e o menor valor

EXERCÍCIOS DE FIXAÇÃO



deste. Apresente ao usuário a opção de pesquisar um valor qualquer em "V", informando sua localização ou declarando que não existe tal valor no vetor. Repetir o procedimento de consulta, até que o usuário decida encerrar a execução do algoritmo.

24. Faça um algoritmo que leia um vetor de 20 valores numéricos e ordene esse vetor em ordem crescente. O programa também deve ler um número K e imprimir, antes e depois da ordenação, o K-ésimo termo da variável composta.
25. Faça um algoritmo que leia e armazene 30 valores reais positivos, determine a sua média e gere um segundo vetor, no qual cada elemento é calculado a partir do elemento correspondente ao primeiro vetor, mais 30%, caso seja menor que a média, senão mais 20%. Mostre os dois vetores. O cálculo de 30% corresponde a multiplicar por 1,3 e 20% por 1,2.
26. Faça um algoritmo que leia e armazene setenta valores reais positivos, determine a sua média e mostre-a. Gere um segundo vetor, no qual cada elemento é calculado a partir do elemento correspondente do primeiro vetor subtraído da média determinada para o primeiro vetor. Calcule o somatório dos valores negativos do segundo vetor e mostre-o. Mostre os dois vetores.

6. Modularizando algoritmos: Procedimentos e Funções

Em sistemas de grande porte, devido à complexidade dos seus algoritmos, é necessária a divisão destes em diversas partes, para que, assim, o sistema tenha uma operação precisa. Essa divisão de tarefas é denominada de “subalgoritmos”. Os subalgoritmos nada mais são do que rotinas que possuem uma função específica.

A modularização consiste em um método para facilitar a construção de grandes programas, através de sua divisão em pequenas etapas, que são: módulos, rotinas, sub-rotinas ou sub-programas. Isto permite o reaproveitamento de código, já que podemos utilizar um módulo quantas vezes forem necessárias, eliminando assim a necessidade de escrever o mesmo código em situações repetitivas.

Um exemplo clássico da utilização de sub-programas é a validação de CPF. Para um número de CPF ser considerado válido, devemos aplicar uma rotina que calcula o dígito verificador e compara com o dígito informado. Em um sistema bancário, este procedimento de validação é utilizado em vários momentos diferentes dentro dos programas/algoritmos do sistema.

Neste caso, a utilização de uma rotina de validação permite que o código seja descrito somente uma única vez, podendo ser utilizado em várias partes do sistema. Imaginando que tal situação se repete por 20 vezes no sistema. Teríamos que escrever 20 vezes o mesmo trecho de código? Não, para isso teríamos uma rotina “subalgoritmo” que teria esta função. Este subalgoritmo teria um nome e, sempre que fosse necessária a verificação do CPF, bastaria invocar, “chamar”, este subalgoritmo.

De um modo geral, os subalgoritmos são importantes devido aos seguintes aspectos:

- Estruturação de algoritmos, facilitando assim a detecção de erros;

- Modularização de sistemas, que justamente é utilizada para a reutilização de rotinas “subalgoritmos” em vários pontos do algoritmo principal;
- Subdivisão de algoritmos extensos, facilitando assim a sua compreensão.

O esquema de um algoritmo e seus subalgoritmos pode ser observado a seguir:

```
Algoritmo (nome do algoritmo)
Variáveis
    Definição das variáveis globais
    <definição dos subalgoritmos>
Início
    <estrutura do algoritmo principal>
Fim.
```

A maioria das linguagens de programação possibilitam criar subalgoritmos, é preciso descrevê-los após a declaração das variáveis e antes do corpo do programa principal. Isso é a definição do subalgoritmos ou a sua declaração. A chamada do subalgoritmo pode ocorrer no programa principal, ou até mesmo em outro subalgoritmo, no qual é necessário que seja executado o código correspondente ao subalgoritmo em questão. Os subalgoritmos podem ser de dois tipos:

- funções;
- procedimentos.

6.1. Procedimentos

Um procedimento, também conhecido como sub-rotina, é composto por um conjunto de instruções que realiza uma determinada tarefa específica. Procedimento não retorna nenhum valor. Sua declaração, que deve estar entre o final da declaração de variáveis e a linha inicial do programa principal, segue a sintaxe abaixo:

```

Procedimento <nome-de-procedimento> [(<sequencia-de-declaracoes-de-
parametros>)]// Seção de Declarações Internas
inicio
    // Seção de Comandos
fimprocedimento

```

Os parâmetros são variáveis para se estabelecer uma troca de informações entre o programa principal e o procedimento. Nem sempre é necessário que exista o uso de parâmetros.

Exemplo - Procedimento para realizar a soma de dois valores, sem a passagem de parâmetros.

```

procedimento soma //procedimento sem parâmetros
var
    resp,n,m: inteiro
inicio
    escreval("Digite um número:")
    leia (n)
    escreval("Digite outro número:")
    leia (m)
    resp <- n + m
    escreval("Soma dos números é: ",resp)
fimprocedimento

algoritmo "Exemplo procedimento sem parametros"
inicio
    soma // chamada do procedimento soma
finalgoritmo

```

Exemplo - Procedimento para realizar a soma de dois valores, com passagem de parâmetros.

```

procedimento soma (n,m: inteiro) //procedimento com parâmetros
var
    resp: inteiro
inicio
    resp <- n + m
    escreval("Soma dos números é: ",resp)
fimprocedimento

algoritmo "Exemplo procedimento sem parametros"
inicio
    soma(2,3) // chamada do procedimento soma enviando dois valores
finalgoritmo

```

6.2. Funções

Uma função é um subprograma que retorna um valor, assim, as funções devem sempre utilizar um comando que permita este retorno de valores. No nosso caso, usaremos a cláusula "retorne". De modo análogo aos procedimentos, sua declaração deve estar entre o final da declaração de variáveis e a linha "início" do programa principal. Segue a sintaxe abaixo:

```
Função <nome-de-funcao> [(<sequencia-de-declaracoes-de-parametros>)]:  
<tipo-de-dado>  
    // Seção de Declarações Internas  
início  
    // Seção de Comandos  
    retorne  
fimfuncao
```

Exemplo - Função para realizar a soma de dois valores, com passagem de parâmetros.

```
funcao soma(n,m: inteiro): inteiro //função com parâmetros  
var  
    resp: inteiro  
início  
    retorne n + m  
fimfuncao  
  
algoritmo "Exemplo procedimento sem parametros"  
var  
    resp: inteiro  
início  
    resp <- soma(2,3) // chamada da função enviando 2 valores e retornando a soma  
    escreval("Soma dos números é: ",resp)  
finalgoritmo
```



EXERCÍCIOS DE FIXAÇÃO

1. Escreva um algoritmo que leia um número qualquer e diga se ele é positivo ou negativo juntamente com a resposta se ele par ou ímpar, utilize um procedimento para cada uma das verificações.
2. Escreva um algoritmo que leia o percentual de faltas e as duas notas de aluno, calcule a média e mostre a mensagem aprovado, se o aluno obtiver média maior ou igual a 6 com menos de 20% de faltas. Para calcular a média, utilize uma função.
3. Construa um programa que leia um número inteiro e verifique se este número é positivo, ou negativo, ou igual a zero. Utilize procedimentos para mostrar o resultado.
4. Escreva um algoritmo que leia dois números e a operação aritmética para ser efetuada com estes números, sendo: a – adição; s – subtração; m – multiplicação; d – divisão; e p – potenciação. Utilizando funções para efetuar a operação indicada, mostre o resultado da operação indicada.

REFERÊNCIAS BIBLIOGRÁFICAS



BENEDUZZI, H. M.; METZ, J. A. Lógica e linguagem de programação: introdução ao desenvolvimento de software. Editora do Livro Técnico Curitiba, PR, 2010.

FORBELLONEE A.; EBERSPACHER H. Lógica de programação, Ed. Pearson, São Paulo, SP, 2005.

MANZANO, J. A. N. G. & OLIVEIRA, J. F. Algoritmos. Lógica para Desenvolvimento de Programação de Computadores. Ed. Érica. São Paulo, SP, 2011.

PUGA, S.; RISSETTI, G. Lógica de programação e estrutura de dados, com aplicações em Java. Pearson. São Paulo, 2011.

SALVETTI, D. D. & BARBOSA, L. M. Algoritmos. Makron Books. São Paulo, SP, 2004.

Esta obra foi composta por Ernesto Henrique Radis Steinmetz e Roberto Duarte Fontes.
Fonte Família Frutiger LT std, corpo 11, Família Caecilia LT std e impressa pela gráfica
AGBR em papel couche fosco 115g.