

Aginaldo da Costa  
Cristiane Koehler  
Edison Andrade Martins Moraes

# LÓGICA DE PROGRAMAÇÃO

Aginaldo da Costa  
Cristiane Koehler  
Edison Andrade Martins Moraes

LÓGICA DE PROGRAMAÇÃO

Tecnologia

EDITORA  
FAEL





# Lógica de Programação

Agnaldo da Costa

Cristiane Koehler

Edison Andrade Martins Moraes

Curitiba  
2016

C837I Costa, Agnaldo da

Lógica de programação / Agnaldo da Costa, Cristiane Koehler, Edison

Andrade Martins Moraes - Curitiba: Fael, 2016.

180 p.: il.

ISBN 978-85-60531-53-0

1. Programação (Informática) 2. Lógica de programação I.

Koehler, Cristiane II. Moraes, Edison Andrade Martins III. Título

CDD 005.115

---

Direitos desta edição reservados à Fael.

É proibida a reprodução total ou parcial desta obra sem autorização expressa da Fael.

#### FAEL

|                              |   |
|------------------------------|---|
| <b>Direção de Produção</b>   | Fernando Santos de Moraes Sarmento                  |
| <b>Coordenação Editorial</b> | Raquel Andrade Lorenz                               |
| <b>Revisão</b>               | FabriCO   |
| <b>Projeto Gráfico</b>       | Sandro Niemicz                                      |
| <b>Capa</b>                  | Vitor Bernardo Backes Lopes                         |
| <b>Imagem Capa</b>           | Shutterstock.com/Thomas Hartwig Laschon/<br>SkillUp |
| <b>Diagramação</b>           | FabriCO   |
| <b>Arte-Final</b>            | Evelyn Caroline dos Santos Betim                    |

# Sumário

CARTA AO aluno | 5

1. CARACTERIZAÇÃO DA tarefa de programar computadores | 7

2. ALGORITMO E programação estruturada | 19

3. REPRESENTAÇÃO DE dados básicos  
(operadores e expressões) | 33

4. COMANDOS DE entrada e saída | 51

5. ESTRUTURAS DE seleção | 65

6. ESTRUTURAS DE repetição | 83

7. UTILIZAÇÃO DE vetores unidimensionais | 101

8. UTILIZAÇÃO DE Vetores Multidimensionais | 117

9. REGISTROS | 139

10. MODULARIZAÇÃO: FUNÇÕES e procedimento | 155

CONCLUSÃO | 173

REFERÊNCIAS | 177



# Carta ao aluno

ESTIMADO ALUNO,

Vamos pensar juntos: por que a disciplina de Lógica de Programação é primordial para nossa ascensão profissional na área de Computação ou Engenharia? É porque, com ela, torna-se possível o desenvolvimento de problemas computacionais, conforme o raciocínio lógico, de forma progressiva e prática. Por isso, nesta disciplina, você encontrará vários assuntos importantes para conduzi-lo na criação da lógica de programação. Tais como:

- × caracterização da tarefa de programar computadores;
- × visão de como o computador trabalha com a lógica de computadores;

- × noção dos conceitos de algoritmo e como desenvolvê-los;
- × compreensão das características da estrutura da programação estruturada;
- × utilização operadores e expressões como dados na linguagem computacional.

Ao final desta disciplina, você vai entender a importância de aprender Lógica de Programação para a realização de muitos exercícios e para o desenvolvimento da capacidade de resolução de problemas, principalmente, na hora de fazer uma programação. Preparado para começar os estudos? Então, vamos lá!

# 1

## Caracterização da tarefa de programar computadores

VOCÊ SABE QUAIS foram as motivações que levaram a humanidade a criar computadores? Simples: projetar uma máquina que fosse capaz de calcular, de forma rápida, operações complexas. E como são divididos os computadores? De modo geral, podemos dizer que eles são compostos por duas importantes partes:

- × *hardware*: componentes eletrônicos que é a parte física do computador (mouse, teclado, discos, placas, etc.);
- × *softwares*: a parte lógica, instituída por programas.

Mas qual é o principal programa (da parte lógica) de um computador? É o que chamamos de Sistema Operacional, responsável por gerenciar as instruções que são passadas para o *hardware*.



A seguir, vamos aprender como o computador reconhece os programas ou o conjunto de instruções que é repassado para seu Sistema Operacional, de acordo com os estudos das linguagens de baixo nível e de alto nível, bem como da sintaxe e da semântica. Siga em frente!

### Objetivos de aprendizagem:

- × Compreender os tipos de linguagens utilizados para Programação de Computadores.

## 1.1 Linguagem de Baixo Nível

Para programar um computador, o que temos que entender? Primeiramente, precisamos saber alguns termos utilizados na área da computação, certo? E um desses termos é a Linguagem de Computador Baixo Nível.

Mas o que isso significa? Para entendermos seu conceito, é preciso antes definir o que é linguagem. Segundo o Dicionário Aurélio Online, a palavra linguagem pode ser definida como “expressão do pensamento pela palavra, pela escrita ou por meio de sinais”, ou “o que as coisas significam”. Isto é, usamos a linguagem para expressar nossos sentimentos, por meio de sinais, gestos, sons etc.

Tendo em vista essa definição, podemos agora compreender melhor o termo Linguagem de Programação. Segundo Ascênsio (2009), Linguagem de Programação pode ser entendida como um conjunto de regras sintáticas e semânticas usadas para definir uma expressão matemática e instruções computacionais.

Aliás, existem no mercado diversos tipos de Linguagens de Programação. E suas principais diferenças estão relacionadas à sua sintaxe (regras), e sua semântica (significados). Outra forma de diferenciá-las (a mais importante), é como elas são classificadas: linguagem de baixo nível e linguagem de alto nível.

Para sabermos o que é linguagem de baixo nível, veremos qual é a representação, ou melhor, os sinais que os computadores trabalham. Primeiro, vale lembrar que as instruções ou os comandos que as máquinas recebem são

representados por códigos binários (0,1), os quais podem representar palavras de 8 a 64 bits.

Em outras palavras, podemos concluir que um computador trabalha somente com uma linguagem, representada por apenas dois códigos: 0,1, que pode ser ligado ou desligado. Isso quer dizer que tais códigos são pulsos elétricos enviados para os componentes eletrônicos através de instruções lógicas dos programas. Por exemplo, todo o vocabulário que utilizamos para expressar nossos sentimentos, ações ou emoções pode ser sintetizado em linguagem de máquina, por meio de combinações de números binários.

**Figura 1** – Códigos Binários Tabela ASCII- Código Padrão Americano para o Intercâmbio de Informação.

### ASCII Code - Character to Binary

|             |             |             |                 |
|-------------|-------------|-------------|-----------------|
| 0 0011 0000 | I 0100 1001 | b 0110 0010 | v 0111 0110     |
| 1 0011 0001 | J 0100 1010 | c 0110 0011 | w 0111 0111     |
| 2 0011 0010 | K 0100 1011 | d 0110 0100 | x 0111 1000     |
| 3 0011 0011 | L 0100 1100 | e 0110 0101 | y 0111 1001     |
| 4 0011 0100 | M 0100 1101 | f 0110 0110 | z 0111 1010     |
| 5 0011 0101 | N 0100 1110 | g 0110 0110 |                 |
| 6 0011 0110 | O 0100 1111 | h 0110 1000 | : 0011 1010     |
| 7 0011 0111 | P 0101 0000 | i 0110 1001 | ; 0011 1011     |
| 8 0011 1000 | Q 0101 0001 | j 0110 1010 | ? 0011 1111     |
| 9 0011 1001 | R 0101 0010 | k 0110 1011 | · 0010 1110     |
|             | S 0101 0011 | l 0110 1100 | ´ 0010 1111     |
|             | T 0101 0100 | m 0110 1101 | ! 0010 0001     |
| A 0100 0001 | U 0101 0101 | n 0110 1110 | ´ 0010 1100     |
| B 0100 0010 | V 0101 0110 | o 0110 1111 | “ 0010 0010     |
| C 0100 0011 | W 0101 0111 | p 0111 0000 | ( 0010 1000     |
| D 0100 0100 | X 0101 1000 | q 0111 0001 | ) 0010 1001     |
| E 0100 0101 | Y 0101 1001 | r 0111 0010 | space 0010 0000 |
| F 0100 0110 | Z 0101 1010 | s 0111 0011 |                 |
| G 0100 0111 |             | t 0111 0100 |                 |
| H 0100 1000 | a 0110 0001 | u 0111 0101 |                 |

Fonte: Shutterstock, 2015.

Na Figura 1, observe como essa combinação funciona: temos os números e as letras que representam os números binários, não é mesmo? É importante

salientar que essa tabela foi criada em 1960, e o significado do código ASCII (*American Standard Code for Information Interchange*) é: Código Padrão Americano para o Intercâmbio de Informação. Porém, lembre-se de que existem outras tabelas padronizadas por outros órgãos internacionais para representar os números binários.

Agora voltando ao assunto da Linguagem de Baixo Nível, você sabia que os primeiros softwares foram desenvolvidos no início da computação por meio dessa linguagem? Isso significa que as instruções de numeração e a representação dos dados eram repassadas aos circuitos eletrônicos (processadores, discos, placas etc.) em uma forma bastante primitiva, baseada no sistema binário.

Segundo Ascêncio (1999, p.10), quando fazemos um programa, este recebe os dados que devem ser armazenados no computador para que possam ser utilizados no processamento e no armazenamento da memória do computador, inclusive esses programas podem ser desenvolvidos em linguagens de Baixo Nível ou Alto Nível.

As principais Linguagens de Programação de Baixo Nível foram:

- × **Linguagem de Máquina:** linguagem nativa do *hardware* da máquina.
- × **Linguagem Hexadecimal:** evolução da linguagem de máquina, usando uma quantidade maior de representação de dados e instruções.
- × **Linguagem Assembly:** conhecida como linguagem simbólica, pois não trabalha com números binários ou hexadecimais, mas com símbolos que os representam.

Segundo Tanenbaum (2007), embora os computadores trabalhem com linguagem de máquina e binária como já definido, cada processador possui uma versão legível da sua linguagem, e tal versão pode mudar de processador para processador. Por isso, para realizar as operações, utiliza-se palavras reservadas denominadas de **mnemônicos**. As principais vantagens dessas linguagens é sua interatividade com o *hardware*, já que suas instruções são interpretadas mais rapidamente, por estar em um nível mais próximo dos componentes eletrônicos. Contudo, sua desvantagem é a complexidade de programar as instruções pois demanda tempo e tempo é custo.

---

**mnemônicos:** é um auxiliar de memória, que pode ter como suportes: os símbolos, os esquemas, os gráficos etc.

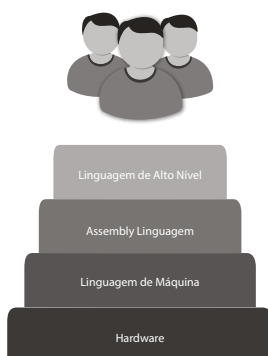
---

## 1.2 Linguagem de Alto Nível

Conforme Xavier (2007), com as dificuldades de trabalhar com as linguagens de Baixo Nível – que utilizam códigos binários –, houve a necessidade de trabalhar com programas que pudessem ser desenvolvidos mais próximos da linguagem humana. Assim, para que dar mais agilidade e tempo de programação, surge o conceito de linguagem de Alto Nível entre a década de 1950 e 1960, destacando-se os tipos: Fortran, Cobol, Algol e Basic.

De lá para cá, já se passaram mais de 50 anos. Ou seja, muitas das linguagens criadas nessa década não são mais utilizadas.

Figura 2 – Linguagem de Alto Nível - Mais perto da Linguagem Humana.



Fonte: elaborada pelo autor, com base em Tanenbaum, 2007.

Hoje as linguagens de Alto Nível possuem um nível de **abstração** maior, ou melhor, mais próximo da Linguagem Humana. Afinal, os recursos implementados nessas linguagens permitem ao programador desenvolver programas sem conhecer as características do processador.

As principais características da Linguagem de Alto Nível são:

- × **Segurança:** capacidade de interpretar os erros durante o desenvolvimento do programa, evitando assim o retrabalho e os erros de implementação.
- × **Clareza:** capacidade de utilizar códigos simples, que podem representar operações matemáticas complexas.
- × **Flexibilidade:** capacidade que essas linguagens possuem para programar variados tipos de aplicações e plataformas.
- × **Portabilidade:** capacidade de ser utilizada em diversos cenários tecnológicos, como os sistemas operacionais Linux, Windows, Os2 da IBM etc.

Figura 3 – Aplicações para Mobile.



Fonte: Shutterstock (2015).

- × **Eficiência:** capacidade de reduzir o tamanho do código e sua velocidade de desenvolvimento.

Atualmente, podemos escolher diversos tipos de linguagem para desenvolvermos nossos programas. E essa escolha está relacionada ao tipo de produto que o cliente quer desenvolver. Por exemplo, aplicações móvel, desktop, cliente-servidor etc.

---

**Abstração:** no conceito da computação, tem como objetivo concentrar-se nos aspectos essenciais e ignorar as partes menos importantes.

---

Quem hoje não possui um celular que pode baixar um APP (aplicativo), como jogos, player, para seu divertimento? Uma das linguagens de Alto Nível capaz de desenvolver aplicações para celulares é a Linguagem Java, a qual possui uma estrutura e um arcabouço de recursos para o desenvolvimento desse tipo de aplicação.



## Saiba mais

Segundo o site Tec Mundo, as principais linguagens de Alto Nível mais utilizadas são: 01- Java Script, 02- Java, 03-PHP, 04 – Python,

05- C, C++, C#. Acesse o site e faça uma leitura dessa importante pesquisa: <<http://www.tecmundo.com.br/programacao/82480-linguagens-programacao-usadas-atualmente-infografico.htm>>.



## 1.3 Sintaxe e Semântica de uma Instrução

Você concorda que, para nós brasileiros, a nossa língua nativa é o Português, certo? Inclusive, é esta a linguagem que estudamos na escola para podermos nos comunicar através da escrita e da fala. Para que haja melhora na comunicação, aprendemos regras gramaticais, e isso demanda tempo e dedicação.

Na linguagem de programação não é diferente. Ou seja, para aprendermos, é necessário que nos dediquemos à sua gramática, às suas regras, a fim de utilizá-las adequadamente em todas as situações.

Segundo ASCENCIO (2015), o primeiro conceito de uma instrução (na linguagem comum, entende-se por instruções) é um conjunto de regras ou normas definidas para a realização ou emprego de algo, para programar um computador é a **sintaxe**.

### 1.3.1 Sintaxe

Como vimos, a sintaxe é responsável por definir a forma como as instruções devem ser escritas ou desenvolvidas. Segundo Forbellone (2005), o conceito de sintaxe pode ser compreendido como um conjunto de regras formais que especificam a composição dos algoritmos a partir de letras, dígitos e outros símbolos.

Mas como é realizado o processo de análise sintática? Esse processo é realizado por um conjunto de regras, que define uma linguagem de programação. Afinal, toda linguagem, inclusive na de programação, há regras, portanto, se queremos aprender uma nova linguagem temos que conhecer suas sintaxes, certo?

As descrições que utilizamos para representar as formas das regras sintáticas são chamadas de **lexemas**. Os lexemas de uma linguagem de programação incluem palavras reservadas, palavras literais e operadores. Entenda melhor analisando o exemplo a seguir.

Quadro 1 – Exemplo de uma instrução: sintaxe e seus lexemas.

| Exemplo de sintaxe: Total= 4 * cont +24; |                           |
|--|---------------------------|
| LEXEMAS                                  | SÍMBOLOS                  |
| Total                                    | Identificador             |
| =  | Sinal de Igual            |
| 4  | Literal                   |
| *  | Operador de Multiplicação |
| Cont                                     | Identificador             |
| +  | Operador de Adição        |
| 24                                       | Literal                   |
| ;  | Ponto e Virgula           |

Fonte: Elaborado pelo autor, 2015.

Observe no quadro 1 que existem diversos símbolos os quais foram utilizados para que o computador pudesse executar uma instrução (**Total = 4 \* cont +24;**). Cada palavra, cada símbolo ou cada operador são interpretados e transformado em código binário pelo computador. Esse processo é chamado de **compilação** ou **interpretação** dos comandos que foram descritos pelo programador, conforme a sintaxe da linguagem de programação.

Caso haja figuras ou símbolos que não sejam possível fazer a interpretação, haverá **erro de sintaxe**. Tais erros sintáticos serão identificados pelo tradutor do programa, e este não funcionará, tendo em vista a impossibilidade de tradução para a linguagem de máquina.

---

---

**Compilação:** processo que transforma códigos de linguagens de programação em linguagem de máquina, isto é, linguagem de alto nível em linguagem de máquina.

---

---

### 1.3.2 Semântica

Se sintaxe representa os símbolos utilizados pelo programados para enviar instruções para a máquina, o que será então a semântica? A semântica

está relacionada à forma lógica como esses comandos ou instruções serão utilizados. Ou seja, é por meio da semântica que validamos um programa.


Para entender melhor isso, vamos exemplificar a semântica em três tipos:

- × **Semântica Estática:** durante a execução do programa, são estabelecidas as restrições que o programador, sintaticamente, realizou para estabelecer um significado.
- × **Semântica Dinâmica:** aqui se estabelece um comportamento de um programa, bem como os resultados deste, após a sua execução ou tradução.
- × **Semântica Operacional:** como resultado do significado de um programa, esse tipo semântico executa as instruções em uma máquina real ou virtual.



## Importante

Lembre-se de que os erros relacionados à semântica altera o resultado esperado, mas não impede que este seja executado. Diferente da sintaxe que, havendo erros, não executa a instrução.



Quando o código de um programa é compilado, por exemplo, ele gera um código-fonte. É a semântica que permite gerar esse código. Porém, caso haja erros de sintaxe do programa, este não conseguirá ser compilado. Conclusão, a semântica está ligada à lógica do programa, mas isso não quer dizer que objetivo foi alcançado.

## 1.4 Por que aprender a programar?

A programação é a técnica para desenvolver computadores e executar operações que desejamos, não é mesmo? Mas podemos afirmar que é, acima de tudo, uma arte. Isso porque permite a liberdade de fazer programas conforme nossa criatividade, além de requer do programador as seguintes habilidades:

- × eficiência;
- × clareza; e
- × manutenção.



Diversas iniciativas do Governo apontam para a necessidade de inserir a programação nas fases iniciais do aprendizado das escolas, pois trará vários benefícios para o desenvolvimento da capacidade cognitiva das crianças, e o melhor: de forma divertida e eficaz. Um exemplo é utilizando jogos de computadores. Afinal, programar desenvolve a capacidade de autonomia, boa memória, paciência, concentração e senso analítico.

### Importante

Para você que está iniciando sua carreira de programador, uma dica: não se atenha apenas a uma linguagem. Nós vimos que as linguagens evoluem rapidamente. Por isso, se você ficar viciado em um tipo de linguagem, ficará para trás no tempo. E lembre-se: a lógica adquirida para programar em uma linguagem será sempre usada em outras linguagens, bastando apenas aprender a sintaxe da nova linguagem.

Vale ressaltar que o aprendizado de Lógica de Programação demanda tempo e muita dedicação, pois lógica se aprende fazendo e não estudando. Assim, quanto mais você quebra a cabeça para a resolução de um algoritmo, maior será a abstração lógica que estará desenvolvendo. Com isso, você criará mais capacidade e resiliência na resolução de exercícios complexos.

A seguir, veja algumas dicas necessárias para criar hábitos salutareos no início da carreira de programador:

- × **Estude Inglês:** a maioria das linguagens de programação são desenvolvidas em inglês. Logo, para ter um aprendizado mais rápido, é ideal dominar o básico da linguagem.
- × **Leitura constante:** o hábito da leitura de livros ou revistas é fundamental para entender o mercado e suas tendências.
- × **Saber Googlear:** pesquisas em site importantes da área, é fundamental para a aprendizagem.

Figura 4 – Aprendendo a programar.



Fonte: Shutterstock (2015).

- × **Faça Exercícios:** os exercícios ajudam no desenvolvimento e na capacidade de resolver problemas em diversas situações.

---


**Salutar:** tem o sentido de fortalecer, despertar, acordar para um sentido específico.

---



## Saiba mais

Pratique suas habilidades no programa chamado **VisualG**, que lhe ajudará na arte da programação. Inclusive, esse programa poderá ser baixado gratuitamente pela Internet no seguinte endereço: <<http://www.baixaki.com.br/download/visualg.htm>>.



Antes de encerrar este capítulo, é importante saliente que a lógica de programação está ligada à área da Ciência da Computação, a qual é denominada uma ciência exata que trata da resolução de problemas computacionais. Leve sempre em conta que há diversas maneira de resolver um problema, e isso nem sempre será através da matemática, mas das interpretações dos problemas.

Segundo a revista Exame (2015), o futuro da área de Desenvolvimento de Software é promissora. O Brasil hoje é 5º maior mercado de TI, crescendo a cada ano. Aliás, há várias oportunidades nos setores públicos, por meio de concursos e no setor privado.

## Resumindo

Neste capítulo, compreendemos os tipos de linguagens utilizados para Programação de Computadores, inclusive como estes se relacionam com as máquinas. Também vimos como sua classificação é fundamental para quem está iniciando seus estudos em Lógica de Programação.

Inclusive, relembramos como os computadores são formados: *hardware* e *softwares*, ou seja, parte física (componentes eletrônicos) e parte lógica (programas utilizados para instruir os computadores nas tarefas), respectivamente.

Outro aspecto importante que aprendemos neste capítulo são as formas com que os computadores trabalham com números binários (0,1), resumindo toda linguagem humana em números representados e padronizados mundialmente em tabelas que contêm o alfabeto, os números e os símbolos. A partir desse conhecimento, entendemos que as linguagens de programação, por meio de suas instruções, utilizam esses códigos padronizados para realizar cálculos e processos nos computadores, e essas linguagens podem ser classificadas de acordo com seus níveis: de Linguagens de Baixo Nível (complexidade na implementação dos programas) e Linguagens de Alto Nível (facilidade na compreensão dos programas).

Além disso, compreendemos que, para programar um computador, é necessário entender o que é Sintaxe e Semântica das linguagens de programação. E, por último, podemos perceber a importância de programar e dominar uma linguagem de computação, já que o mercado atual está precisando muito de profissionais competentes nessa área.

# 2

## Algoritmo e programação estruturada

NESTA AULA, IREMOS aprender conceitos fundamentais sobre algoritmos e programação estruturada. Vamos compreender quais os procedimentos iniciais para o desenvolvimento de algoritmos, quais os objetivos na criação deles, mas para entender esses conceitos, precisamos entender os ensinamentos básicos de entrada e saída de dados, ou seja, como o computador trabalha com os dados que são enviados e por onde são enviados, como são executados, qual a sequência lógica que deve ser seguida para que um algoritmo resolva um problema computacional. Para solucionarmos um problema, devemos projetar a resposta, para isso, vamos aprender a desenhar algoritmos e programas.

Existem diversas formas de implementar um algoritmo e vamos estudá-las através das técnicas de implementação, aprendendo os principais comandos e como resolver problemas computacionais.

## Objetivo de aprendizagem:

- × Aplicar os conceitos de algoritmos e as características da programação estruturada.

## 2.1 O que é algoritmo?

Segundo Forbellone (2000, p.03), algoritmo pode ser definido como uma sequência de passos que visam atingir um objetivo bem definido. Quando elaboramos um algoritmo, devemos especificar ações claras e precisas que possam resultar na solução de um problema proposto. Pelo conceito dado, podemos entender que o algoritmo é uma sequência de procedimentos finita, que deverá previamente ser estabelecida para que possa atingir os objetivos delimitados.

Não pensando ainda de forma computacional, mas em lógica do nosso dia a dia, podemos criar algoritmos para qualquer situação. Uma receita de bolo é um exemplo em que podemos utilizar para explicitar como é um algoritmo. Primeiramente, separamos os ingredientes e, após, devemos, passo-a-passo, juntá-los. No caso do bolo, a sequência de passos deve ser realizada de forma lógica e sequencial, caso contrário o resultado será desastroso, pois o

Figura 1 – Sequência de um algoritmo para fritar um ovo.



Fonte: Shutterstock (2016).

sucesso do bolo está justamente na sequência de passos corretos. Surge então uma palavra chave: lógica.

Segundo Forbellone (2005), a lógica significa o uso correto das leis de pensamento da “ordem e da razão”, e de processos de raciocínio e simbolização formais. No exemplo do bolo, a lógica é inserir o fermento por último, senão o bolo não irá crescer.

Outro exemplo que pode ser analisado: fazer um ovo frito. Werlich (2007) mostra os passos a serem seguidos para fritar um ovo.

- × Aquecer a frigideira;


- × Adicionar óleo para fritar o ovo;
- × Quebrar o ovo;
- × Colocar o ovo na frigideira;
- × Esperar fritar;
- × Virar o ovo para fritar do outro lado;
- × Servir o ovo frito.

A sequência utilizada nesse exemplo foi criada para se chegar a um objetivo, mas ela poderia ser alterada, caso esse desafio fosse dado para outra pessoa. Como por exemplo: verificar se existe um ovo para ser frito, adicionar sal ou utilizar algum utensílio para virar o ovo.



### Saiba mais

Segundo Werlich (2007), o grau de detalhamento pode ou não ser indispensável no desenvolvimento de um algoritmo, tudo depende da situação que deverá ser muito bem observada pelo programador.



Um fator de impacto no desenvolvimento de algoritmo é a experiência do programador, que pode refletir diretamente na lógica, pois terá uma visão mais apurada, otimizando melhor o código proposto. Para conseguir mais experiência e uma lógica mais refinada só há um caminho: realizar exercícios.

Os algoritmos criados com o pseudocódigo, no nosso caso, devem ser independentes do tipo de linguagem de programação. Para isso, devemos realizar a conversão do algoritmo para uma linguagem de programação. O objetivo principal de criarmos algoritmos é a base de conhecimento que ele proporciona para aprendermos outras linguagens.

Para praticarmos o desenvolvimento de algoritmos voltados à linguagem de programação estruturada, utilizaremos o software visualG, que pode ser encontrado no seguinte endereço: <<http://www.apoioinformatica.inf.br/produtos/visualg>>, sem perder as principais características de sua finalidade, que é a prática da programação.

## Importante

“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” (FORBELLONE e EBERSPACHER, 2005).

## 2.2 Conceitos básicos de entrada e saída de dados e sequência lógica de programação

Antes de iniciarmos a construção de algoritmos, temos que compreender conceitos de entrada e saída de dados. Todo trabalho realizado por um computador é baseado na manipulação das informações de entrada de dados, , processamento de dados e saída de dados, entender esse conceito é fundamental para o programador, inicialmente vamos entender o que é *hardware* e *software*.

Figura 2 - Unidade Central de Processamento



Fonte: Shutterstock (2016).

O *hardware* do computador é dividido em três elementos principais:

### Hardware

**Unidade Central de Processamento (UCP ou CPU):** é nesse dispositivo, encontrado em todos os computadores, que as operações matemáticas são processadas, são compostas de vários registradores, que adquirem dados da memória e realiza cálculos complexos.

Na **Figura 2**, podemos perceber os pinos de encaixe de uma CPU na placa de um computador. Dentro da UCP ou CPU temos a ULA, que significa unidade lógica aritmética responsável pelas operações matemáticas, a outra parte chama-se UC, unidade de controle, responsável pelo controle de todos os elementos dos computadores. Quando iniciarmos a construção de nossos programas, você poderá entender que quem realizará os cálculos das operações será a unidade ULA (Unidade Lógica Aritmética) e o controle de saída de dados será executado pela UC (Unidade de Controle).

---

**ULA- Unidade Lógica Aritmética** – faz parte do processador e é um circuito onde os dados lógicos e matemáticos são processados.

**CPU – Unidade Central de Processamento** - realiza instruções de um programa de computador.

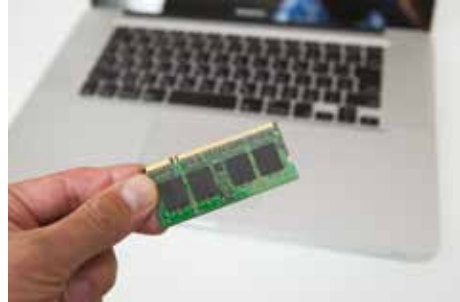
---

### Você sabia

Hoje os processadores podem executar bilhões de informações por segundo. Um processador que possui 1,2 GHz, executa um bilhão e duzentos milhões de ciclos por segundo.

**Memória principal:** conhecida como memória RAM, volátil de acesso aleatório, é uma memória que armazena as informações dos programas que serão executadas, nela são armazenadas os dados e informações que serão utilizadas durante a execução do programa. É composta de vários compartimentos, onde é possível guardar todas as informações passadas. Cada posição da memória possui um endereço, e um conteúdo poderá ser armazenado.

Figura 3 - Memória RAM - memória de acesso aleatório.



Fonte: Shutterstock (2015).

Ao ser desligado, todo o conteúdo armazenado nestes endereços é perdido. Para resolver esse problema, outros recursos chamados de memórias secundárias como os discos rígidos, *pendrives*, são utilizados para armazenar informações permanentes, pois mantêm os dados armazenados eletronicamente.

**Dispositivos de entrada e saída de dados:** os periféricos de entrada e saída de dados são responsáveis por enviarem as informações para as unidades de processamento (CPU), para serem processadas. Outros periféricos são responsáveis pela saída das informações.



**Entrada de dados:** teclado, mouse, scanner, caneta ótica, etc.

**Saída de dados:** impressora, monitor, entre outros.

Podemos perceber que uma simples construção de um algoritmo passa por diversos processos de *hardware* para ser mostrado o resultado na tela, os dados de entrada são repassados para o processador pelos dispositivos de entrada de dados, na memória são armazenados os endereços, no processador são executadas as operações de cálculos e devolvidos ao usuário pela unidade de controle, como saída de dados. Este processo deve estar bem claro em nossa mente para podemos entender o processo de processamento de um programa.

## Software


Quando falamos em *software* de computadores, o primeiro conceito que vem a nossa mente é a parte lógica do computador. Segundo TANENBAUM (2009), *software* é o conjunto de componentes lógicos de um computador ou sistema de processamento de dados; programa, rotina ou conjunto de instruções que controlam o funcionamento de um computador

O principal *software* utilizado pelo computador é o Sistema Operacional que tem a função de administrar todos os outros *softwares* e o *hardwares* da máquina. Outros *softwares* são chamados de aplicativos, são responsáveis por diferentes tipos de aplicações que o usuário deseja para realização das tarefas, podemos citar: Microsoft Word, Microsoft Excel, Autodesk Cad, um editor de áudio, fotografia, etc.



### Você sabia

**Software de Aplicação:** é constituído por uma variedade de programas que nos permitem realizar variadíssimas tarefas relacionadas com os nossos trabalhos e lazer.



Externamente ao Sistema Operacional, vamos encontrar um software extremamente importante para nossa aula, chamado de Compilador. Quer saber qual o papel desse *software* no processo de construção dos algoritmos? Leia a seguir.

**Compiladores:** são *softwares* utilizados para transformar um programa escrito em linguagem computacional em linguagem de máquina. Ela ocorre

com a passagem do algoritmo para um programa-fonte, posteriormente é compilado, no nosso caso, quem realiza essa compilação é o VisualG.

Outro aspecto importante dos compiladores é a verificação de erros de sintaxe e semântica. Caso o programa tenha erros, não irá compilar e não gerará o arquivo fonte pra ser usado.

**Sequência lógica de programação:** para desenvolver um algoritmo, devemos projetá-lo, essa projeção deve possuir uma sequência de passos que deveremos utilizar pra podermos chegar a resolução final do problema. Em programação, chamamos essa sequência lógica de algoritmo ou programa. Vamos entender um pouco mais por meio do exemplo abaixo, que realiza a multiplicação de dois números:

Tarefa: multiplicar dois números

Sequência lógica / algoritmo

**Passo 01** - Escrever o primeiro número no retângulo A (10)

**Passo 02** - Escrever o segundo número no retângulo B (8)

**Passo 03** - Multiplicar o número do retângulo A com o B

**Passo 04** - Escrever o resultado no retângulo C (80)

$$10 * 08 = 80$$

Figura 4 - Sequência para a multiplicação de dois números.



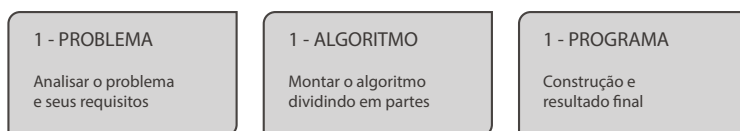
Fonte: elaborado pelo autor, 2015.

A instrução multiplicar foi o comando dado para realizar a operação matemática, os passos para alcançar a resolução do problema, o comando “escreva” são instruções dadas ao usuário do programa. A resolução do problema desse algoritmo consistiu nos seguintes passos: compreender o problema, identificar os dados de entrada, os dados de saída, construir o algoritmo, testar o algoritmo, executar o algoritmo.

## 2.3 Como desenhar algoritmos e programas

O desenho de algoritmos está relacionado à resolução de um problema, para resolver um problema devemos desenvolver passos ou sequências. Forbellone descreve esse método como descendente ou método de refinamento, pois consiste em dividir um problema em partes menores para que seja possível resolvermos o todo. É a famosa frase chamada de “dividir para conquistar” (História das Sociedades). Existem fases distintas para serem consideradas antes de desenhar um algoritmo: a análise do problema, quais requisitos serão considerados, a concepção do algoritmo, como iremos dividi-lo para resolver o problema e a tradução desse algoritmo em linguagem de programação, no nosso caso o *software* VisualG.

Figura 5 – Passos para o desenho de um programa.



Fonte: elaborado pelo autor, 2015.

Na figura 5, podemos observar a sequência que devemos seguir para desenhar um programa.

## 2.4 Distinção das diferentes representações de algoritmos

Existe uma certa divergência quando falamos em representação de algoritmos, mas podemos encontrar na literatura três formas distintas de representá-lo. Dentre as formas mais conhecidas, Ascencio (2012) enumera: descrição Narrativa, fluxograma convencional, pseudocódigo, ou linguagem estruturada.

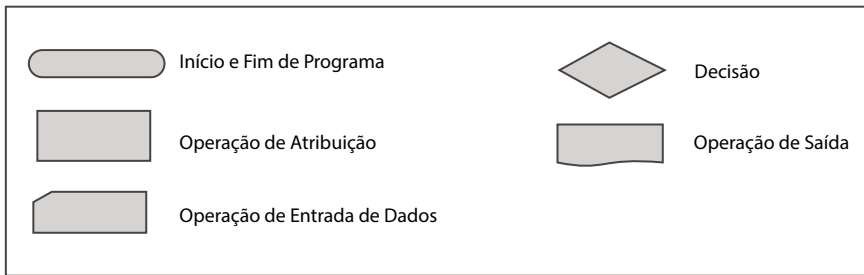
**Descrição Narrativa:** nesse tipo de representação é utilizada a linguagem natural para descrever um algoritmo. Pode-se exemplificar o modelo usado para se fazer um bolo, ou trocar um pneu de um carro, ou trocar uma lâmpada. É uma representação pouco utilizada, pois dá oportunidades a más interpretações, muitas ambiguidades e imprecisões.

Podemos prever essa imprecisão quando pedimos na receita de bolo para colocar meia xicara de óleo. Qual o tamanho da xicara, quanto tempo deve se misturar o óleo com a farinha?

**Fluxograma convencional:** se a descrição narrativa utiliza a linguagem natural para representação de um algoritmo de forma sequencial, o fluxograma convencional utiliza-se de formas gráficas para representar as sequencias de um algoritmo, em que ações, instruções, decisões são representados por meio de figuras geométricas.

No **Fluxograma convencional** para a construção de algoritmos temos várias simbologias, adotadas para representar operações de entrada e saída de dados realizados em dispositivos distintos, conforme Figura 7, onde temos a figura do início do programa, operação de atribuição, operação de entrada de dados, decisão, operação de saída de dados.

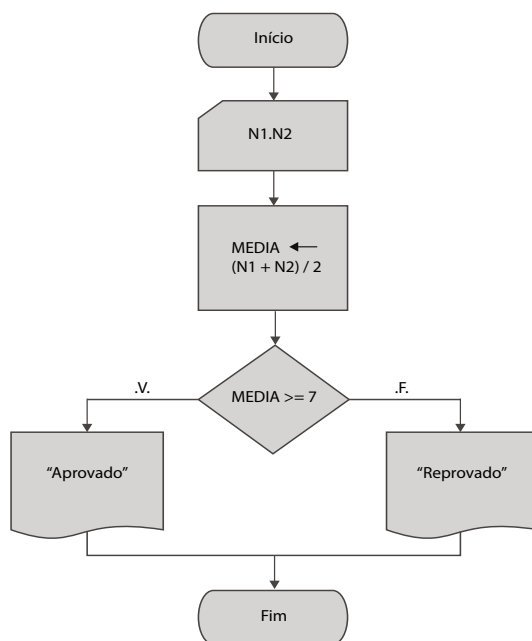
Figura 7- Representação e significados das figuras geométricas.



Fonte: elaborado pelo autor, 2015.

A utilização dessa técnica facilita o entendimento das ideias contidas nos algoritmos e é um método mais evoluído para o desenvolvimento de algoritmos que os fluxogramas convencionais. Um fluxograma se resume a um símbolo que é utilizado para mostrar o início do programa quando este é iniciado, e um ou mais símbolos que finalizam o programa quando seu fluxo chega ao fim. Na Figura 8, podemos ter uma visão de como um programa ou algoritmo pode ser representado.

Figura 8 - Fluxograma utilizado para representar um algoritmo.



Fonte: elaborado pelo autor, 2015.

Na Figura 8 temos o início do programa, e a declaração de duas variáveis (N1, N2). Perceba como as figuras mudam com o decorrer da sequência do algoritmo. Em seguida, temos um quadrado em que é executada uma operação matemática, atribuindo o valor  $(N1, N2) / 2$  a uma variável chamada média. Finalizando o algoritmo, ele apresenta uma estrutura de decisão: caso a média for maior ou igual a 7 “aprovado”, caso seja menor que 7 “reprovado”. E termina com a figura geométrica (igual do início), representando o fim do algoritmo.

**Pseudocódigo, ou linguagem estruturada:** esta forma de representação de algoritmos é a mais utilizada quando iniciamos o estudo de lógica de programação, pois é possível representar todos os dados de um problema e os manipularmos sem ambiguidade, outra grande vantagem é a semelhança na forma como os programas são escritos para os computadores, realizando apenas uma tradução dessa estrutura para estrutura computacional.

Forbellone (2005) apresenta um algoritmo na forma de pseudocódigo da seguinte forma:

```
Algoritmo <nome_do_algoritmo>  
    <declaração_de_variaveis>  
    <subalgoritmo>  
  
inicio  
  
    <corpo do algoritmo>  
  
fim
```

Essa é uma estrutura básica de um algoritmo, chamada de pseudocódigo, ou linguagem de programação, vamos entender o que significa cada uma dessas estruturas.

---

---

**Pseudocódigo:** é uma forma genérica de escrever um algoritmo, utilizando uma linguagem simples.

---

---

**Algoritmo:** palavra que define o algoritmo que será desenvolvido.

**<nome\_do\_algoritmo>:** nome dado ao algoritmo para definir sua finalidade

**<declaração\_de\_variaveis>:** local onde são definidas as variáveis que serão utilizadas em todo o pseudocódigo e nos subalgoritmos.

**<declaração\_de\_variaveis>:** uma opção que pode ser utilizada no pseudocódigo para a utilização de subalgoritmo que poderá eventualmente ser utilizado no pseudocódigo.

**Início e Fim:** são as palavras que definem o início e o fim do pseudocódigo.

Vamos analisar como ficaria o programa ou pseudocódigo mostrado como exemplo no fluxograma convencional da Figura 9.

Figura 9 - Pseudocódigo

```
Algoritmo Calculo_Media
Var N1, N2, MEDIA: real
Início
Leia N1, N2
MEDIA ← (N1 + N2) / 2
Se MEDIA >= 7 então
Escreva "Aprovado"
Senão Escreva "Reprovado"
Fim_se
Fim
```

Fonte: elaborado pelo autor, 2015.

Nessa estrutura, podemos perceber que os detalhes da implementação são mais ricos que outras formas de representar uma sequência de instruções para resolução de um problema. A transformação desse pseudocódigo em um programa de computador é transparente para qualquer linguagem de programação, por isso, esse modelo de estrutura de programação é mais utilizado para aprender lógica de programação. Vamos compreender mais um pouco sobre a linguagem estruturada. Na figura 10, temos um exemplo de como ficaria implementado o programa no *software* VisualG que pode ser adquirido no endereço: <<http://www.baixaki.com.br/download/visualg.htm>>.

Figura 10 - Exemplo prático de um algoritmo em VisualG.

```
Algoritmo "Média"
var                                     // nome do algoritmo
N1, N2, Media: real                   // declaração de variáveis
inicio                               // início do algoritmo
escreva ("informe o primeiro valor")
leia (N1)
escreva ("informe o segundo valor")
leia (N2)
Media <- (N1+N2)/2
se Media >= 7 então
Escreva ("Aprovado")
senao
Escreva ("Reprovado!!")
fimse
filmalgoritmo                        // fim algoritmo
```

Fonte: elaborado pelo autor, 2015.

## 2.5 O que é linguagem estruturada e suas características

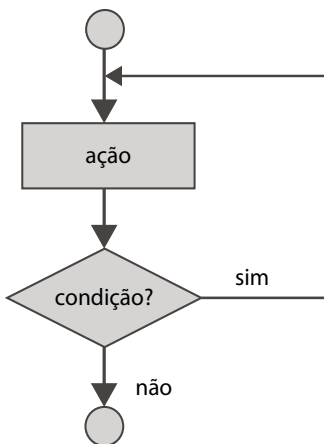
A programação estruturada é bastante utilizada por programadores, muitos programas, como a linguagem de programação C, utiliza a técnica de programação estruturada. Para iniciantes da área da computação é quase que obrigatório o uso dessa técnica para o desenvolvimento da lógica de programação.

### Importante

Programação estruturada é uma forma de programação de computadores que preconiza que todos os programas possíveis podem ser reduzidos a apenas três estruturas: sequência, decisão e iteração (esta última também é chamada de repetição), desenvolvida por Michael A. Jackson no livro "Principles of Program Design" de 1975.

A programação estruturada é realizada por meio de estruturas simples, reduzidas apenas três sequências básicas: sequência, seleção, iteração. Vamos conhecer detalhadamente as etapas da linguagem estruturada:

Figura 11 - Exemplo de comandos de condição (fluxograma).



Fonte: elaborado pelo autor, 2015.

**Sequência:** essa é uma característica bastante peculiar dos programas desenvolvidos em programação estruturada, pois define a sequência de passos necessários para a resolução do problema em forma de algoritmo.

**Seleção:** possibilidade de direcionar o caminho por onde o programa deve seguir baseado em ocorrências lógicas.


**Iteração:** permite executar certas estruturas do programa, repetindo várias vezes até que a condição seja satisfeita. Esse processo ocorre por meio de comandos específicos, como enquanto a condição for verdadeira “faça”.





## Importante

Quer aprofundar seus conhecimentos sobre as características principais das linguagens estruturadas? Acesse o link: <<http://www.dev-media.com.br/introducao-a-programacao-estruturada/24951>>.



## Resumindo

Nesta aula, aprendemos vários conceitos importantes para o desenvolvimento de seu aprendizado, vimos o que é um algoritmo e quais os passos básicos para seu desenvolvimento, aprendemos que existe uma estrutura computacional de entrada, processamento e saída dos dados, e os componentes que fazem parte dessa estrutura como: teclado, mouse, processadores, memórias, discos, etc, e que ajudam na execução dos algoritmos criados.

Aprendemos que é possível desenharmos e projetarmos algoritmos, e essas representações podem ser realizadas de diferentes formas, bem como as principais diferenças existentes para a representação de algoritmos e quais são mais importantes. Vimos também a programação estruturada como: descrição narrativa, fluxo convencional e pseudocódigo ou programa. Entendemos que a utilização do pseudocódigo traz um ganho maior no desenvolvimento de algoritmos, pois representa com mais clareza a realidade computacional. Compreendemos o conceito de programação estruturada e as características das linguagens estruturadas para o desenvolvimento de programas.

# 3

## Representação de dados básicos (operadores e expressões)

NESTE CAPÍTULO, VOCÊ IRÁ compreender os tipos de dados, os operadores lógicos e expressões matemática utilizadas para resolverem problemas computacionais por meio de algoritmos. Para isso, utilizaremos o *software* **VisuAlg**, que poderá ser baixado e testado nessa aula.

Vamos aprender quais as operações matemáticas podem ser utilizadas em nosso dia a dia para a resolução de algoritmos, como os operadores relacionais são aplicados a uma decisão computacional, comparando os números para saber quem é o maior. Os tipos de dados usados em variáveis, que podem ser inteiras, reais, lógicas e booleanas e suas aplicações nos programas.

## Objetivo de aprendizagem:

- × Compreender os principais tipos de dados operadores;
- × Aprender as lógicas e expressões;
- × Reconhecer as operações matemáticas aplicáveis no dia a dia;
- × Conhecer o funcionamento do *software* visuAlg.

## 3.1 Utilizando o VisuAlg para desenvolver um algoritmo

Segundo Pereira (2011), o VisuAlg é um programa simples, que pode ser utilizado por iniciantes de programação estruturada e tem por objetivo o desenvolvimento do raciocínio lógico. Possui uma linguagem denominada de ‘portugol’, que são comandos realizados na língua portuguesa, que facilitam o aprendizado do iniciante em lógica de programação.

Figura 1 – Tela do Site para baixar o programa VisuAlg.



Fonte: Apoio Informática, 2016.

Para baixar a versão do VisuAlg (versão 2.5), acesse o site do *software* localizado em: <<http://www.apoioinformatica.inf.br/produtos/visualg>>.

### 3.1.1 Instalação e requerimentos de hardware

Sua instalação não copia arquivos para nenhuma outra pasta, a não ser aquela em que for instalado, e exige cerca de 1 MB de espaço em disco. Pode ser executado em sistemas operacionais Windows.

### 3.1.2 A tela principal do VisuAlg

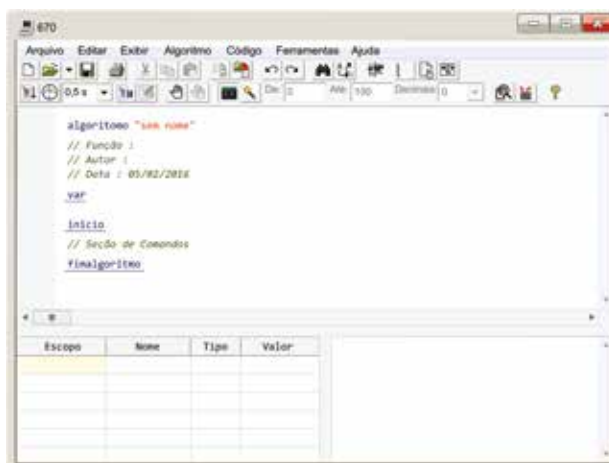
A tela do VisuAlg possui o padrão Windows e é composta de: barra de tarefas, do editor de textos (que toma toda a sua metade superior), do quadro

de variáveis (no lado esquerdo da metade inferior), do simulador de saída (no correspondente lado direito) e da barra de status, conforme Pereira (2011).

Quando o programa é carregado, já apresenta a estrutura para a utilização do pseudocódigo, com a intenção de poupar trabalho ao usuário e de mostrar o formato básico que deve ser seguido.

Explicaremos a seguir cada componente da interface do VisuAlg.

Figura 2 – Estrutura padrão do VisuAlg.



Fonte: elaborado pelo autor, 2016.

---

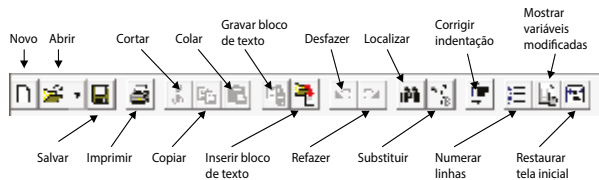
**Pseudocódigo:** é uma forma genérica de escrever um algoritmo, utilizando uma linguagem simples.

---

### 3.1.3 A barra de tarefas

A barra apresenta os comandos mais utilizados no VisuAlg. Como sua base é no padrão Windows, você já estará familiarizado, pois são os mesmos botões encontrados em programas como o Microsoft Word.

Figura 3 – Barra de tarefas do VisuAlg.



Fonte: Pereira, 2011.

A tabela 1 apresenta as descrições simplificadas de cada função que pode ser realizada pelo VisuAlg. A intenção é a facilitar a vida do programador.

Tabela 1 – Comandos do menu VisuAlg.

|                                |  |
|--------------------------------|--|
| <b>Abrir (Ctrl-A):</b>         | Abre um arquivo anteriormente gravado, substituindo o texto presente no editor.                  |
| <b>Novo (Ctrl-N):</b>          | Cria um novo “esqueleto” de pseudocódigo, substituindo o texto presente no editor.               |
| <b>Salvar (Ctrl-S):.</b>       | Grava imediatamente o texto presente no editor.  |
| <b>Imprimir:</b>               | Imprime imediatamente na impressora padrão o texto presente no editor.                           |
| <b>Cortar (Ctrl-X):</b>        | Apaga texto selecionado, armazenando-o em uma área de transferência.                             |
| <b>Copiar (Ctrl-C):</b>        | Copia o texto selecionado para a área de transferência.  |
| <b>Colar (Ctrl-V):</b>         | Copia texto da área de transferência para o local em que está o cursor.                          |
| <b>Desfazer (Ctrl-Z):</b>      | Desfaz último comando efetuado.  |
| <b>Refazer (Shift-Ctrl-Z):</b> | Refaz último comando desfeito.   |
| <b>Localizar (Ctrl-L):</b>     | Localiza no texto presente no editor determinada palavra especificada.                           |
| <b>Substituir (Ctrl-U):</b>    | Localiza no texto presente no editor determinada palavra especificada, substituindo-a por outra. |

|                                      |   |
|--------------------------------------|---|
| <b>Corrigir Indentação (Ctrl-G):</b> | Corrige automaticamente a indentação (ou tabulação) do pseudocódigo, tabulando cada comando interno com espaços à esquerda.   |
| <b>Numerar linhas:</b>               | Ativa ou desativa a exibição dos números das linhas na área à esquerda do editor.   |
| <b>Mostrar variáveis modificadas</b> | Ativa ou desativa a exibição da variável que está sendo modificada. Como o número de variáveis pode ser grande, muitas podem estar fora da janela de visualização; quando esta característica está ativada, o VisuAlg rola a grade de exibição, de modo que cada variável fique visível no momento em é modificada. Este recurso é importante para que o programador possa verificar os valores que cada variável recebe durante a aplicação do programa. |

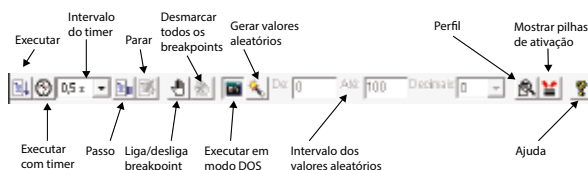
Fonte: elaborado pelo autor, com base em Pereira, 2011.

Abaixo, veremos a imagem do menu que possui os comandos ou manipulações específicas que podem ser realizadas no algoritmo desenvolvido pelo programador. São os comandos que mais serão utilizados no decorrer do nosso aprendizado.

## Importante

Os comandos que não estão descritos nas tabelas acima podem ser encontrados em outros manuais, inserimos o que consideramos essencial. No site <<http://www.apoioinformatica.inf.br/produtos/visualg>>, você pode encontrar todos os comandos no item “menu”.

Figura 4 – Menu VisuAlg – comandos principais.



Fonte: Pereira, 2011.

Tabela 2 – Comandos principais do VisuAlg.

|  |   |
|--|---|
| <b>Executar (F9):</b>                            | Inicia (ou continua) a execução automática do pseudocódigo. Esse comando será muito utilizado quando começarmos a executar os algoritmos.   |
| <b>Executar com timer (Shift-F9)::</b>           | Insere um atraso (que pode ser especificado no intervalo ao lado) antes da execução de cada linha. Também realça em fundo azul o comando que está sendo executado, da mesma forma que na execução passo a passo.                            |
| <b>Passo (F8):</b>                               | Inicia (ou continua) a execução linha por linha do pseudocódigo, dando ao usuário a oportunidade de acompanhar o fluxo de execução, os valores das variáveis e a pilha de ativação dos subprogramas.  |
| <b>Parar (Ctrl-F2):</b>                          | Termina imediatamente a execução do pseudocódigo.   |
| <b>Liga/desliga breakpoint (F5):</b>             | Insere/remove um ponto de parada na linha em que esteja o cursor. Permite a verificação dos valores das variáveis e da pilha de ativação de subprogramas. Esse comando é relevante pois possibilita a observação das variáveis no programa. |
| <b>Desmarcar todos os breakpoints (Ctrl-F5):</b> | Desativa todos os <i>breakpoints</i> que estejam ativados naquele momento.  |
| <b>Executar em modo DOS:</b>                     | Com esta opção ativada, tanto a entrada como a saída-padrão passa a ser uma janela que imita o DOS, simulando a execução de um programa neste ambiente.   |
| <b>Gerar valores aleatórios:</b>                 | Ativa a geração de valores aleatórios que substituem a digitação de dados. A faixa padrão de valores gerados é de 0 a 100.  |
| <b>Intervalo dos valores aleatórios:</b>         | Faixa de valores que serão gerados automaticamente, quando esta opção estiver ativada.  |
| <b>Mostrar pilha de ativação (Ctrl-F3):</b>      | Exibe a pilha de subprogramas ativados num dado momento. Convém utilizar este comando em conjunto com <i>breakpoints</i> ou com a execução passo a passo.   |

Fonte: elaborado pelo autor, com base em Pereira, 2011.

## Saiba mais

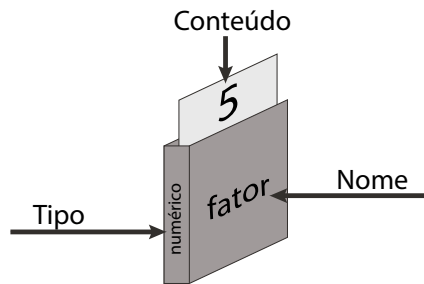
Para saber mais dicas sobre a utilização do VisualG e seus comandos acesse o site <<http://www.apoioinformatica.inf.br/produtos/visualg>>, no item produtos você encontrará vários exemplos importantes.

## 3.2 Variáveis simples

Segundo Ascencio (1999), quando construímos um programa, ele recebe os dados que devem ser armazenados na memória do computador que serão utilizados no processamento das informações. Para que ele seja classificado como variável, precisa apresentar a possibilidade de ser alterado em algum momento do programa.

Fonte: Furtado, 2015.

Figura 5 – Exemplo de variáveis.



Para entendermos o que é uma variável, podemos compará-la a uma caixa construída para armazenar objetos, conforme a figura 5. O conteúdo desta caixa não é fixo, não é permanente, ela pode receber diversos tipos de conteúdos, mas todos devem ser do mesmo tipo, no nosso caso, deverá ser numérico, recebendo apenas números, e recebe o nome de fator. O nome, tipo e conteúdo são nominados de acordo com o programador ou usuário.

Como é declarado uma variável? Conforme Forbellone (2000), a declaração ocorre antes de ela ser usada. É neste momento que informaremos um tipo para ela. Isso significa “informar” ao computador o que essa variável poderá receber e armazenar na memória do computador. Quando isso ocorre, um espaço da memória é reservado para receber esses dados.

Muitos são os tipos de variáveis. O VisuAlg prevê quatro tipos de dados: inteiro, real, cadeia de caracteres e lógico (ou booleano), assim como outras linguagens de programação que utilizam ferramentas para o desenvolvimento de seus programas. Um exemplo é eclipse, plataforma Java que trabalha com esses tipos, pois são os mais comuns. As palavras-chave que os definem são as seguintes (observe que elas não têm acentuação):



**Eclipse:** é um IDE (do inglês *Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado, é um programa de computador, que reúne características e ferramentas de apoio ao desenvolvimento de *software* com o objetivo de agilizar este processo.), para desenvolvimento Java, porém suporta várias outras.

Tabela 3 – Tipos de variáveis.

| TIPO DE VARIÁVEL | DESCRIÇÃO   |
|------------------|---|
| <b>inteiro:</b>  | Tipos de variáveis que recebem números inteiros sem casas decimais são utilizados para valores inteiros (números positivos ou negativos - sem vírgulas);  |
| <b>real:</b>     | Tipos de variáveis numéricas do tipo real, que possuem casas decimais exemplo: 10,20 – 9,8  |
| <b>caractere</b> | Tipos de variáveis do tipo <b>string</b> , ou seja, cadeia de caracteres, onde a junção desses caracteres forma as palavras e valores que receberão somente um caracter (letra ou número de 0 a 9 ou um sinal, o que chamamos de caracteres especiais); |
| <b>logico:</b>   | Tipos de são variáveis do tipo booleano, ou seja, com valor, verdadeiro e falso.  |

Fonte: Pereira, 2011.

### 3.2.1 Declarações de variáveis

Segunda Pereira (2011), para fazermos a declaração de variáveis, é necessário que os nomes comecem por uma letra. O restante pode ser composto por letras, números ou *underline*, até um limite de 30 caracteres. As variáveis podem ser simples ou estruturadas.



#### Saiba mais

Para a declaração de variáveis não pode haver duas variáveis com o mesmo nome, e nem a utilização das palavras reservadas de uso compilador.



No exemplo a seguir podemos observar como são declaradas as variáveis e sua sintaxe.

**Sintaxe:**

<Identificador 1>: <tipo das variáveis>

**Identificador 1:** o nome da variável que iremos dar ao construir o algoritmo.

**:** Dois pontos separa o nome da variável e seu tipo.

**Tipo das Variáveis:** onde é definido os tipos de variáveis, que podem ser **inteiro, real, character, logico** (todos sem acentos).

No exemplo a seguir, podemos verificar as declarações de variáveis realizadas. Veja que na variável do tipo real foram declarados três nomes para um tipo de variável: valor\_Produto1, valor\_Produto2, valor\_Produto3, isso poderá ser realizado caso as variáveis sejam do mesmo tipo.

A utilização do *underline* para escrever o nome das variáveis é outro recurso padrão utilizados pelos programadores para otimizar o código melhorando também sua documentação

**Exemplo de declaração de variáveis:**

numero: **inteiro**

Valor\_Produto1, valor\_Produto2, valor\_Produto3: **real**


nome\_do\_Produto: **caractere**

tem\_o\_Produto: **logico**



## Importante

Várias palavras e termos são usados para a construção do algoritmo e elas não devem ser utilizadas na declaração de variáveis.



Na tabela abaixo, veremos todos os termos que não devem ser colocados nas declarações, pois são reservadas para utilização do VisuAlg.

Tabela 4 – Palavras reservadas.

| PALAVRA RESERVADAS |                 |            |              |
|--------------------|-----------------|------------|--------------|
| aleatorio          | e               | grauprad   | passo        |
| abs                | eco             | inicio     | pausa        |
| algoritmo          | enquanto        | int        | pi           |
| arccos             | entao           | interrompa | pos          |
| arcsen             | escolha         | leia       | procedimento |
| arctan             | escreva         | literal    | quad         |
| arquivo            | exp             | log        | radpgrau     |
| asc                | faca            | logico     | raizq        |
| ate                | falso           | logn       | rand         |
| caracter           | fimalgoritmo    | maiusc     | randi        |
| caso               | fimenquanto     | mensagem   | repita       |
| compr              | fimescolha      | minusc     | se           |
| copia              | fimfuncao       | nao        | sen          |
| cos                | fimpara         | numerico   | senao        |
| cotan              | fimprocedimento | numpcarac  | timer        |
| cronometro         | fimrepita       | ou         | tan          |
| debug              | fimse           | outrocaso  | verdadeiro   |
| declare            | função          | para       | xou          |

Fonte: Apoio Informática, 2016.

### 3.3 Constantes

O tipo constante é utilizado em um algoritmo nas situações em que os valores não são alterados durante a execução do programa, diferentemente das variáveis, em que seus valores são alterados. Imagine uma situação na qual você precisaria informar a todo o instante um valor específico como: R\$ 30.563,892. Este número imenso, a todo o momento deve ser digitado. Uma vez definida a variável como sendo uma constante, o seu valor não poderá ser modificado. O VisuAlg tem três tipos de constantes:

Tabela 5 – Tipos de Constantes.

| TIPOS       | DESCRIÇÃO  |
|-------------|--|
| Númericos:  | Constantes numérica podem ser do tipo inteiros ou reais.   |
| Logicos:    | Constantes lógicas podem ser verdadeiro ou falso.          |
| Caracteres: | Uma cadeia de caracteres delimitada por aspas duplas (“”). |

Fonte: Ascencio, 2012.

A sintaxe de uma constante é definida:

<Identificador 1>: <valor>

**Identificador 1:** nome da constante;

**Valor:** o valor que ela receberá.

#### Exemplo de declaração de Constantes:

Nome = “Agnaldo da Costa”

Pi= 3,14

## 3.4 Expressões aritméticas

Agora vamos ver como utilizar as expressões aritméticas para os cálculos matemáticos. Para o processamento de dados, podemos usar os operadores de: divisão, subtração, multiplicação e divisão.

Tabela 6 – Operadores aritméticos.

| OPERADORES ARITMÉTICOS              | DESCRIÇÃO                    |
|-------------------------------------|------------------------------|
| +                                   | Adição:                      |
| -                                   | Subtração:                   |
| *                                   | Multiplicação:               |
| /                                   | Divisão:                     |
| \                                   | Divisão Inteira              |
| ^ ou <b>Exp</b> (<base>,<expoente>) | Exponenciação ou Potenciação |
| %                                   | Módulo (resto da divisão)    |

Fonte: elaborado pelo autor, 2016.

### 3.5 Expressões lógicas

As expressões lógicas são utilizadas em estruturas de decisão e retornam verdadeiro ou falso para a operação. Vamos trabalhar com quatro expressões lógicas: **nao**, **ou**, **e**, **xou**, para as operações de construção de algoritmos.

Tabela 7 – Expressões lógicas.

| EXPRESSÕES LÓGICAS | DESCRIÇÃO   |
|--------------------|---|
| <b>nao</b>         | Operador que resulta não VERDADEIRO = FALSO, e não FALSO = VERDADEIRO. Tem a maior precedência entre os operadores lógicos. |
| <b>ou</b>          | Operador que resulta VERDADEIRO quando um dos seus operandos lógicos for verdadeiro.  |
| <b>e</b>           | Operador que resulta VERDADEIRO somente se seus dois operandos lógicos forem verdadeiros.                                   |
| <b>xou</b>         | Operador que resulta VERDADEIRO se seus dois operandos lógicos forem diferentes, e FALSO se forem iguais.                   |

Fonte: elaborado pelo autor, com base em Ascencio, 2012.

Quer compreender melhor as expressões lógicas? Vamos analisar a tabela e testar os operadores **e ou xou**.

#### 3.5.1 Operadores relacionais

Segundo Forbellone (2005), os operadores relacionais são muito utilizados em programação, as decisões dos algoritmos geralmente são tomadas nas operações relacionais, ou seja, as decisões baseiam-se em testes do estado das variáveis. Logo, é muito importante entender o que é uma operação relacional e quais os operadores utilizados nesse tipo de expressão.

Tabela 8 – Tabela-verdade dos operadores.

| A | B | A e B | A ou B | não A |
|---|---|-------|--------|-------|
| V | V | V     | V      | F     |
| V | F | F     | V      | F     |
| F | V | F     | V      | V     |
| F | F | F     | F      | V     |

Fonte: elaborado pelo autor, com base em Ascencio, 2012.

Em várias situações que envolvam comparações de valores, obrigatoriamente usaremos os operadores relacionais. São eles: **igual, diferente, maior, menor, maior ou igual, menor ou igual**.

Tabela 9 – Operadores relacionais.

| OPERADORES<br>RELACIONAIS | DESCRIÇÃO      |
|---------------------------|----------------|
| >                         | Maior          |
| <                         | Menor          |
| >=                        | Maior ou igual |
| <=                        | Menor ou igual |
| =                         | Igual          |
| <>                        | Diferente      |

Fonte: elaborado pelo autor com base em Ascencio, 2012.

### Exemplo de comparações de valores

Vamos atribuir valores para A e B e verificar o retorno Verdadeiro ou Falso: A=5 e B=22.

A > B, resposta: falso

A >= B, resposta: falso

A < B, resposta: verdadeiro

A <=B, resposta: verdadeiro

A = B, resposta: falso

A<>B, resposta: verdadeiro

### Você sabia

Você sabe o que fazer quando houver uma expressão aritmética com parênteses? Sempre terão precedência os parênteses de dentro para fora. Observe a expressão:

$(9 + ((10 * 2) + (14 - 10)))$  o resultado será igual a 33.

## 3.6 Comandos de atribuição

Para inserirmos valores dentro de uma variável no desenvolvimento de um algoritmo utilizamos o operador de atribuição. Ele é responsável por indicar que a variável receberá um valor designado pelo programador ou operador do programa. O operador de atribuição é representado por uma seta (<-) apontando para a esquerda. Já a atribuição de valores a ela, segundo Xavier (2007), ocorre com o operador <-. Do seu lado esquerdo fica a variável, à qual está sendo atribuído o valor, e à sua direita pode-se colocar qualquer expressão (constantes, variáveis, expressões numéricas), desde que seu resultado tenha tipo igual ao da variável. A atribuição insere um valor no endereço de memória alocado para a variável quando de sua declaração. Então o conteúdo deve ser adequado ao tipo de dado declarado em função do espaço de memória alocado para conter o dado desta variável. Um valor real 1.0 não cabe em um espaço definido como inteiro 1.

### **Alguns exemplos de atribuições, usando as variáveis.**

```
b <- 90
```

```
Valor01 <- 33.5
```

```
Valor2 <- Valor1
```

```
Nome <- "Agnaldo da Costa"
```

```
Peso <- 100 // Este comando atribui à variável Peso o valor 100.
```

```
Idade_Maior <- FALSO // Este comando atribui à variável Idade_Maior o valor FALSO.
```

### **Importante**

Sempre à esquerda do comando de atribuição deve haver um (e somente um) identificador de variável.

## 3.7 Estrutura de um programa

Ao longo da disciplina, iremos porpor diversos exercícios, e todos serão desenvolvidos com o *software* VisuAlg. A linguagem que o programa interpreta é bem simples: é uma versão portuguesa dos pseudocódigos largamente

utilizados por estudantes da computação que estão iniciando programação, e essa linguagem é conhecida como “Portugol”.

Vamos abrir o VisuAlg e analisarmos sua estrutura, assim compreenderemos como é um programa realizado nesse *software*. Se você conseguiu baixá-lo e executá-lo, teremos a seguinte tela, conforme mostra a figura 06.

Figura 6 – Estrutura de um programa no VisuAlg

```

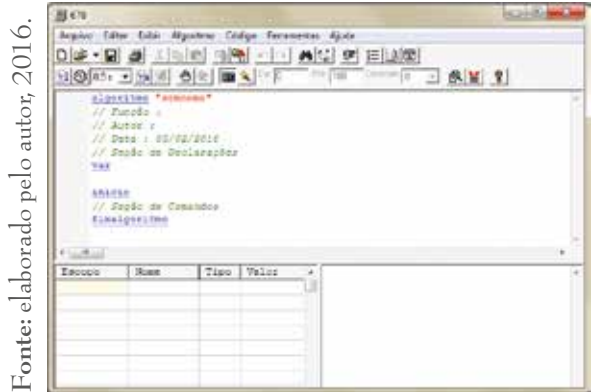
// Função:
// Autor:
// Data: 05/05/2016
// Estado de União Europeia

// Exibir o nome do usuário

```

O VisuAlg permite a inclusão de comentários: qualquer texto precedido de “//” é ignorado pelo compilador do VisuAlg, até se atingir o final da sua linha. Por este motivo, os comentários não se estendem por mais de uma linha. Quando se deseja escrever comentários mais longos, que ocupem várias linhas, cada uma delas deverá começar por “//”.

**Figura 6** – Estrutura de um programa no VisuAlg.



Fonte: elaborado pelo autor, 2016.

**Compilador:** programa que traduz um programa descrito em uma linguagem de alto nível para um programa equivalente em código de máquina para um processador.

## Saiba mais

O comentário usando `///` permite que o programador possa realizar a documentação do código. A cada linha criada pode ser comentada a sua função dentro do programa, isso permite que programadores possam realizar a alteração do código ou manutenção do mesmo sabendo o que trata cada linha do programa, agilizando assim a manutenção do programa.



O formato básico do nosso pseudocódigo é o seguinte:

Tabela 10 – Estrutura de um programa do VisuAlg.

| ESTRUTURA                  | DESCRIÇÃO   |
|----------------------------|---|
| 1. algoritmo<br>“semnome”  | A primeira linha é composta pela palavra-chave algoritmo, seguida do seu nome delimitado por aspas duplas. Este nome será usado como título dos programas desenvolvidos. É sempre bom colocar o nome do algoritmo, não confunda com o nome do arquivo.  |
| 2. // Autor :              | Neste espaço pode ser inserido o nome do autor do algoritmo.  |
| 3. // Função               | Qual o objetivo ou qual problema esse algoritmo estará resolvendo, é importante descrever em breves palavras.   |
| 4. // Data :               | O VisuAlg aloca a data do computador para esse campo.   |
| 5. // Seção de Declarações | A partir dessa seção, começam as declarações das variáveis que podem ser utilizadas pelo programa. É aqui que descrevemos os tipos de dados que serão usados na lista de comandos, que podem ser reais, inteiro caracteres, lógicos. A seção termina com a linha que contém a palavra-chave inicio. |
| 6. // Seção de Comandos    | Marca a indicação de que entre as palavras início e fim podemos escrever uma lista com uma ou mais instruções ou comandos para serem usados para resolver os algoritmos.  |
| 7. inicio                  | Inicia os comandos onde serão utilizadas as variáveis criadas, e comandos de entrada e saída e processamentos de dados.   |
| 8. fimalgoritmo            | Marca o fim da execução do algoritmo.   |

Fonte: elaborado pelo autor, 2016.

As palavras que integram a sintaxe da linguagem são as reservadas, ou seja, não podem ser usadas para outro propósito em um algoritmo que não seja aquele previsto nas regras de sintaxe. A palavra algoritmo, por exemplo, é uma palavra reservada. Neste texto, as palavras reservadas sempre aparecerão em destaque.

## Resumindo

Nessa aula, aprendemos sobre uma ferramenta importante no desenvolvimento de algoritmos para aprimorar o raciocínio lógico: o *software* VisuAlg, que é bem simples de ser utilizado, pois sua linguagem é natural (língua portuguesa). Vimos a estrutura desse programa e como ele funciona.

Compreendemos também como utilizar as expressões matemática para resolvermos problemas lógicos, utilizados juntamente com os operadores relacionais e operadores lógicos. Todos esses conjuntos de símbolos podem ser expressos nos algoritmos dentro do VisuAlg.

A partir de agora, temos que nos familiarizar com a ferramenta, aprendendo a declarar variáveis e seus tipos, suas constantes, para aperfeiçoar nosso conhecimento.



# 4

## Comandos de entrada e saída

NESTE CAPÍTULO, APRENDEREMOS a usar os comandos de entrada e saída de dados com o *software* VisuAlg. É importante entendermos os conceitos envolvidos nessas operações, pois serão desenvolvidos ao longo da aula.

Lembre-se que a prática é essencial para o aprendizado. Os comandos de entrada de dados serão fornecidos pelos usuários no transcorrer do programa, e podem ser armazenados em variáveis, já os comandos de saída são os resultados das operações realizadas com estas variáveis. Os comandos de saída de dados podem ser formatados por meio de formatações especiais.

## Objetivo de aprendizagem:

- × Aplicar comandos de entrada e saída em linguagem de programação estruturada.

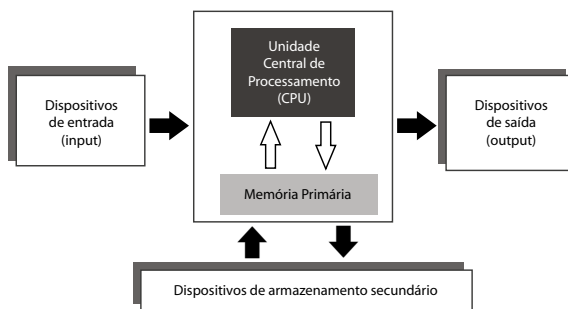
### 4.1 O conceito de entrada e saída

No algoritmo, precisamos representar as informações que iremos trocar entre a máquina e o usuário. Elas são realizadas por meio de comandos de entrada e saída de dados, tornando possível manipular dispositivos de hardware como: teclados mouses, discos, impressoras, etc.

Na linguagem de programação temos comandos específicos para lidar com cada dispositivo de entrada e saída de dados. Em nível de algoritmo, esses comandos representam apenas entrada e saída de informações, pois são dados direcionados para cada tipo de unidade de entrada/saída. Segundo Forbellone (2005), esses comandos são os que permitem interação com o usuário por meio de dispositivos.

Até agora falamos nos comandos de entrada e saída de dados. Mas para que servem? Os relativos à saída servem para representar os dados do computador para o usuário, exibidos na tela. Já os de entrada servem para enviar dados do usuário para o computador. Na figura 1, podemos exemplificar como é o processo realizado para a construção de um algoritmo que utiliza comando de entrada e saída de dados.

Figura 1 – Os dispositivos de entrada e saída de dados.



Fonte: elaborado pelo autor, com base em Tanenbaum (2007)

Nos dispositivos de entrada de dados serão inseridas as informações para serem processadas, essa missão é realizada por dispositivos ou usuário. O processamento dos dados inseridos pelos usuários é feito pelo processador, que posteriormente realiza a saída dos dados.

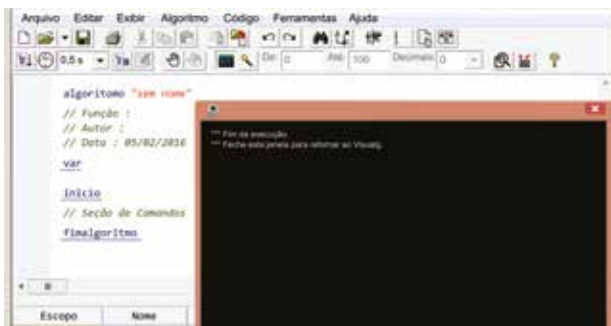
O *software* visuAlg, que utilizaremos para construir os algoritmos, realiza a operação de entrada de dados, processamento de dados e saída de dados via criação e declaração de variáveis no início do programa (seta amarela da Figura 2). A saída de dados ocorre por meio da execução do programa (processamento), como mostrado na tela a seguir (preta). Para testar se o VisuAlg está instalado de forma correta, pressione a tecla F9, no menu Algoritmo, Executar. O programa mostrará uma tela de saída, conforme podemos observar na tela preta da Figura 2.

---

**Software:** programa, rotina ou conjunto de instruções que controlam o funcionamento de um computador.

---

Figura 2 – Saída de dados no VisuAlg.



Fonte: elaborado pelo autor, 2016.

## Importante

Os programas são formados, essencialmente, por comandos (instruções sobre o que fazer). Segundo Tanenbaum (2007), os comandos são lidos sequencialmente da memória, um após o outro.

Existem dois comandos para trabalharmos a entrada e saída de dados. Para entrada, vamos usar o comando “leia”, e para saída de dados o comando “escreva”. Veremos como utilizar esses dois comandos nos próximos tópicos.

## Saiba mais

Os comandos determinam quando e quais ações “primitivas” devem ser executadas. São exemplos de ações “primitivas”: leitura de dados, saída de dados, atribuição de valor a uma variável. Além disso, os comandos podem ser “estruturados”. A “estruturação” dos comandos permite que eles sejam executados numa determinada ordem, que a sua execução seja repetida ou que se opte pela escolha de um ou outro comando subordinado.

### 4.1.1 Comandos de saída

Agora que já tivemos um panorama geral sobre os comandos de entrada e saída, passaremos a detalhar cada um deles. Iniciando pelo comando de saída para entendermos como são executadas pelo programa VisuAlg. Segundo Ascencio (2012), o comando de saída de dados é utilizado para mostrar dados na tela ou na impressora. Ele é representado pela palavra “Escreva” e os dados podem ser conteúdos de variáveis ou mensagens, e ser enunciados da seguinte forma:

**Escreva** (<expressão ou identificador ou constante>, <expressão ou identificador ou constante>

No VisuAlg, podemos encontrar dois comandos Escreva, os quais são utilizados em situações diferenciadas:

**Escreval** (<expressão ou identificador ou constante>) //Mostra o primeiro resultado na mesma linha depois em linhas diferentes.

**Escreva** (<expressão ou identificador ou constante>) //Mostra o resultado na mesma linha, mas em colunas diferentes.

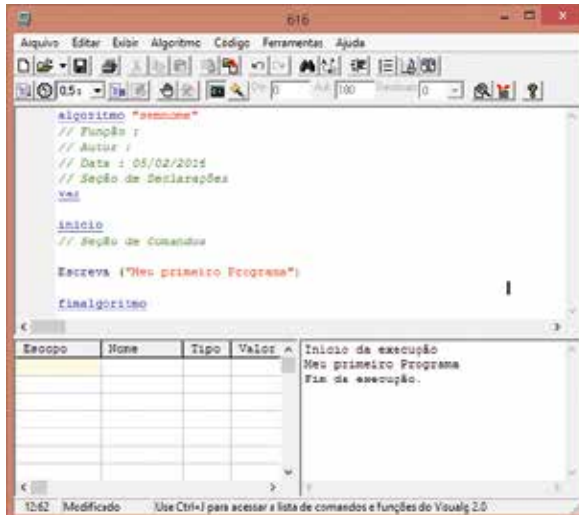
Vamos praticar esse comando realizando nosso primeiro exercício do comando “Leia”, na linguagem Portugol. Crie uma pasta com o nome “Exerc\_Comandos”. Abra o VisuAlg para realizarmos essa atividade.

Ao abrir o programa, primeiramente salvaremos nosso arquivo. Para isso, vamos em “Arquivo”, “Salvar”, para salvarmos na pasta “Exerc\_Comandos”, com

o nome “primeiro\_Programa\_Es”. Pronto! Para alterar esse programa, na linha algoritmo, escreva “pratica comando escreva”, conforme Figura 3. Na seção início, após a linha comentada // seção de comandos, escreva o seguinte código:

Escreva (“Meu primeiro Programa”)

Figura 3 – Primeiro programa no VisuAlg.



Fonte: elaborado pelo autor, 2016.

Vamos analisar o comando dado para que o computador possa executar essa operação. O comando “Escreva” é acompanhado por um parêntese ( ), em que mostra a mensagem que será escrita. Outro detalhe importante são as aspas duplas, que se referem a um texto escrito. Todas as vezes que formos dar uma mensagem para a tela de entrada de dados usaremos essa sintaxe.

## Importante

Nesse pequeno exemplo, não utilizamos nenhuma variável, apenas queremos que o VisuAlg escreva na tela uma mensagem.

Vamos agora executar nosso primeiro programa para vermos qual será o resultado da saída de dados. Pressione a tecla F9, caso tenha feito tudo certo, a seguinte tela aparecerá, conforme mostrado na figura 4.



Figura 4 – Saída do primeiro programa.

```
Meu primeiro Programa
*** Fim da execução.
*** Feche esta janela para retornar ao VisuAlg
```

Fonte: elaborado pelo autor, 2016.

Faça um teste e utilize o comando “Escreva”, com “l”, no final e note a diferença de formatação na saída do programa.

### 4.1.2 Comandos de entrada

Segundo Ascencio (2012), o comando de entrada de dado recebe valores digitados pelo usuário, atribuindo-os às variáveis cujos nomes estão declarados no início do algoritmo. A sintaxe do comando de entrada de dados é dada da seguinte forma:

**Leia** (<identificador>)

Atenção, esse comando sempre estará associado às variáveis criadas no início do programa. Como já aprendemos o comando “Escreva”, agora ficará mais fácil entender o comando “Leia”. Ao abrir o programa, primeiramente vamos salvar nosso arquivo. Para isso vamos em “Arquivo”, “Salvar”, salve na pasta “Exerc\_Comandos”, com o nome “primeiro\_Programa\_Leia”. Pronto!

Será que voce já está apto a colocar em prática tudo o que vimos até este momento? Vamos testar seus conhecimentos?

Figura 5 – Declaração de variáveis no VisuAlg.

```
algoritmo "pratica comando Leia"
// Função :
// Autor :
// Data : 30/12/2015
// Seção de Declarações
var
num1: inteiro
num2: inteiro
soma: inteiro
```

#### Problema:

Faça um algoritmo que peça dois números inteiros ao usuário, realize a soma destes números e apresente na tela o seu resultado.

Para resolvermos esse desafio, primeiramente devemos pensar na proposta sugerida. Segundo Forbellone (2005), para resolver

um problema no computador é necessário que seja encontrada uma maneira de descrevê-lo de forma clara e precisa. Precisamos encontrar uma sequência de passos que permitam a solução automática e repetitiva. Além disto, é preciso definir como os dados que serão processados serão armazenados no computador. Portanto, a solução é baseada em dois pontos: a sequência de passos e a forma como os dados serão armazenados no computador. Vamos a resolução:

O algoritmo pedia a soma de dois números inteiros. Então iremos declarar duas variáveis, conforme o primeiro passo. O algoritmo também pedia a soma desses valores, vamos declarar uma terceira variável, que irá receber esses valores digitados pelo usuário, conforme passo 1:

**1. Passo declarar as variáveis:**

num1 : inteiro


num2: inteiro

Soma: inteiro

A declaração das variáveis é realizada na seção Var, conforme a sequência:

### **Você sabia**

Caso o algoritmo não tivesse pedido o tipo de variável, iríamos utilizar variáveis reais, pois recebem números fracionados, para que não ocorresse erros no programa.



Depois de declaradas as variáveis que farão parte do programa, vamos à sequência de passos para a resolução do problema. O algoritmo dizia que o usuário digitaria dois números, para que se possa realizar essa operação, teremos que usar o comando “escreva” (como já visto no tópico anterior), esse passo começa após a declaração das variáveis, abaixo do comando início:

**início**

// Seção de Comandos

Escreval (“Digite o primeiro Número”)

Leia (num1)

Escreval (“Digite o segundo Número”)

Leia (num2)

## Atenção!!

Veja como o comando “Leia” é utilizando juntamente com as variáveis criadas (num1, num2). Quando o usuário digita o número na tela, ele é armazenado na memória da máquina por meio do comando “Leia”, por isso o comando:

**Leia (num1)**

**Leia (num2).**

Figura 6 – Comando Escreva no VisuAlg.

```
inicio  
// Seção de Comandos  
Escreval ("Digite o primeiro Numero")  
Leia (num1)  
Escreval ("Digite o segundo Numero")  
Leia (num2)  
soma<-num1+num2  
Escreval ("A soma total é", soma)  
fimalgoritmo
```

Fonte: elaborado pelo autor, 2016.

Figura 7 – Programa “Leia” e “Escreva” completo no VisuAlg.

```
algoritmo "pratica comando Leia"  
// Função :  
// Autor :  
// Data : 05/02/2016  
// Seção de Declarações  
var  
num1: inteiro  
num2: inteiro  
soma: inteiro  
inicio  
// Seção de Comandos  
Escreval ("Digite o primeiro Numero")  
Leia (num1)  
Escreval ("Digite o segundo Numero")  
Leia (num2)  
soma<-num1+num2  
Escreval ("A soma total é", soma)  
fimalgoritmo
```

Fonte: elaborado pelo autor, 2016.

O algoritmo pedia que esses dados fossem somados, para realizar essa operação, vamos usar o comando de atribuição <- que você já aprendeu. Chamamos esse passo de processamento das informações. Os dados ficarão da seguinte forma:

**Soma-< num1+num2**

Verifique que foram atribuídos operadores aritméticos de adição. Na memória da máquina criamos três variáveis, conforme declarado no programa, o resultado dos valores armazenados nas variáveis (num1, num2), inseridos pelos usuários, serão somados e guardados na variável (soma). Verifique como ficou o programa na figura 6.

Na saída “Escreva”, a soma total utiliza os valores armazenados na variável (soma), depois de fecharmos as palavras consideradas textos com as aspas duplas, em

seguida, usamos uma vírgula e chamamos a variável (soma) para ser impressa.

Vamos agora verificar como fica esse programa com os comandos “Leia” e “Escreva”, escritos no programa VisuAlg, confira linha a linha e execute o programa com o comando F9.

Digite o primeiro número pedido, 40, e o segundo número, 50. Dessa forma, teremos o resultado de saída, conforme mostra a figura 8:

Agora fica mais fácil entender todo o processo realizado pelo conjunto computacional. Analise a figura 9, e compare com a atividade desse algoritmo: realizamos a entrada de dado (teclado) pelo usuário com as variáveis (num1, num2, soma). Os dados foram gravados na memória da máquina para serem processadas.

Figura 8 – Tela de saída do VisuAlg.

```
Digite o primeiro Numero
40
Digite o segundo Numero
50
A soma total é 90

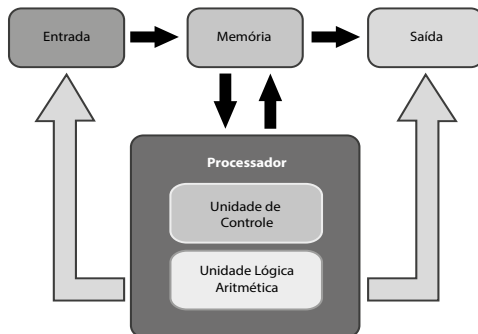
*** Fim da execução.
*** Feche esta janela para retornar ao Visualg.
```

Fonte: elaborado pelo autor, 2016.

## Você sabia

Para deixarmos o código mais limpo para ser utilizado, procure usar o comando “Escreva” com um “\n” no final, esse comando determina para o compilador pular uma linha na tela.

Figura 9 – Conjunto computacional de entrada, processamento e saída de dados.



Fonte: Oliveira, 2015.

O processamento das informações ocorreu quando pedimos que os valores das variáveis (num1, num2) fossem somados e armazenados na variável (soma). Para executar esse processamento, utilizamos operadores aritméticos e de atribuição: soma <- num1+num2. O processamento foi realizado e a saída foi dada na tela (monitor), programa VisuAlg.

---

---

**Unidade de Controle (UC):** responsável por gerar todos os sinais que controlam as operações no exterior do CPU, e dar as instruções para o correto funcionamento interno do CPU

**Unidade Lógica Aritmética (ULA):** é um circuito combinatório responsável pela execução de somas, subtrações e funções lógicas em um sistema digital.

---

---

## 4.2 Praticando comando de entrada e saída de dados

Que tal praticarmos mais os comandos de entrada e saída de dados com o programa VisuAlg? Confira o exercício abaixo:

### Prática 01

Faça um algoritmo que receba dois números e, ao final, mostre a subtração, multiplicação.

Resolução:

algoritmo “pratica01”

// Função : Algoritmo de subtração, multiplicação de números

Var

x, y, resultadoSub, resultadoMult: real

inicio

// Seção de Comandos

escreval(“Digite o primeiro número: “)

leia(x)

```

escreval("Digite o segundo número: ")
leia(y)
resultadoSub <- x-y
resultadoMult <- x*y

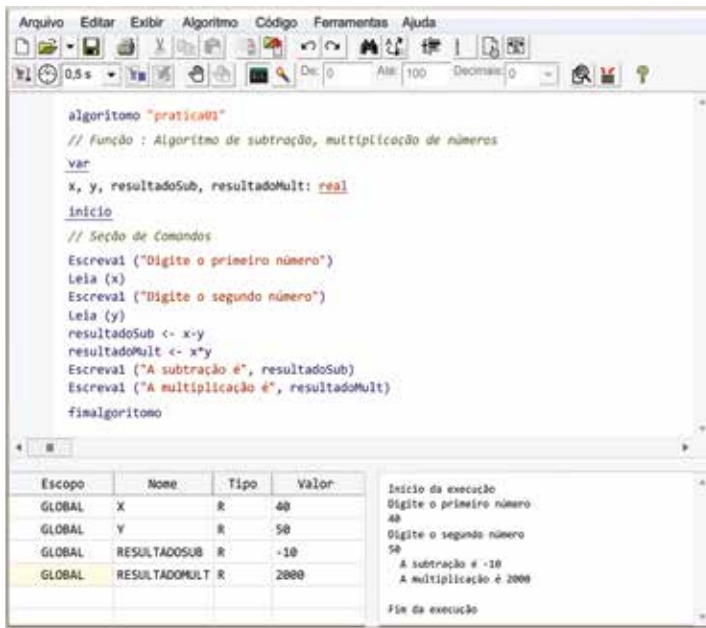
escreval("A subtração é: ", resultadoSub)
escreval("A multiplicação é: ", resultadoMult)

finalgoritmo

```

Nesse algoritmo, é importante notar que as variáveis (x,y), que estão na memória da máquina, podem ser manipuladas, conforme o programador desejar. Usando os mesmos valores, temos o seguinte resultado, como mostra a figura 10:

Figura 10 – Exercícios prática 01



Fonte: elaborado pelo autor, 2016.

## Saiba mais

Quando utilizamos todas as variáveis do mesmo tipo, podemos declará-las em uma mesma linha, separada por vírgulas, assim seu programa ficará mais otimizado.

### 4.3 Formatação de saída de dados

Ao utilizar o comando “Escreva” no VisuAlg, é possível realizar formatações para as saídas de dados. Lembre-se que já apresentamos o comando Escreva + “l”, onde as linhas são formatadas.

Figura 11 – Formatação do comando “Escreva”.

```
algoritmo "pratica comando Leia"  
// Função :  
// Autor :  
// Data : 05/02/2016  
// Seção de Declarações  
var  
num1: inteiro  
num2: inteiro  
soma: inteiro  
inicio  
// Seção de Comandos  
Escreval ("Digite o primeiro Numero")  
Leia (num1)  
Escreval ("Digite o segundo Numero")  
Leia (num2)  
soma<-num1+num2  
Escreval ("A soma total é", soma)  
fimalgoritmo
```

Fonte: elaborado pelo autor, 2016.

Outro exemplo que já realizamos foi a elaboração de um dado armazenado ou processado no comando “Escreva”. Na linha vermelha da figura 11 podemos ver como essa variável é chamada dentro do “Escreva” para mostrar os dados armazenados.

Podemos utilizar outros atributos formatados com o comando “Escreva”, para que os programas possam ser mais legíveis aos usuários.

No exercício da Prática 01 feito acima, realizamos as operações de multiplicação e subtração, para isso, utilizamos quatro variáveis (x, y, resultadoSub, resultadoMult), do tipo real. Com o comando “Escreva”, podemos simplificar a apenas duas variáveis(x,y). Observe como poderemos utilizar essa formatação:

Observe na linha vermelha que chamamos as variáveis “ x” e “y”, dentro do comando “Escreval”, realizando as operações de multiplicação e subtração. Dessa forma, otimizamos o programa com menos codificações.

Outro exemplo que podemos utilizar são as formatações de saída de dados, analise o algoritmo na figura 13.

Figura 13 – Formatação com casas decimais.

```
algoritmo "formatação"
var
x: real
inicio
x <- 55.653
escreval (x:2:2)
finalgoritmo
```

Fonte: elaborado pelo autor, 2016.

É possível formatar a saída do Visualg usando números reais, colocando duas casas decimais após a vírgula. Quando for usar o comando “Escreva”, coloque dois pontos e insira o número de casas pra esquerda da vírgula, depois coloca dois pontos e quantas casas após a vírgula você desejar.

Figura 12 – Formatações do comando “Escreva” na saída de dados.

algoritmo "pratica 01"

var

x, y: real

inicio

// Seção de Comandos

escreval ("Digite o primeiro número:")

leia (x)

escreval ("Digite o segundo número:")

leia (y)

escreval("A subtração é", x-y)

escreval("A multiplicação é", x\*y)

finalgoritmo

Fonte: elaborado pelo autor, 2016.



## Resumindo

Nesta aula, aprendemos dois comandos fundamentais para trabalharmos e resolvermos algoritmos: “Leia” e “Escreva”. Eles são responsáveis por contextualizar as operações de entrada de dados e saída de dados em um sistema computacional. Entender esse sistema é de fundamental importância para a resolução dos algoritmos que estaremos propondo. Vimos que existe uma sequência que teremos que seguir para a resolução dos algoritmos.

Primeiramente, temos que pensar no problema proposto, essa é uma das principais dificuldades encontradas, erros de interpretação. Depois, devemos criar as variáveis e seus tipos. Após, damos as entradas dos dados para que seja realizado o processamento das operações.

Outro passo importante é o processamento das operações, na qual usamos os comandos de atribuições, operações aritméticas, operadores lógicos, etc. Se você quiser aprofundar mais seus conhecimentos, realize todos os estudos de caso para entender como funcionam os comandos de entrada e saída de dados.

# 5

## Estruturas de seleção

NESTE CAPÍTULO APRENDEREMOS sobre as **estruturas de seleção** em programação de computadores. Veremos onde são aplicadas estas estruturas para resolução de problemas computacionais sua sintaxe. Tais estruturas têm o objetivo de testar o conteúdo da **variável de teste** em um **teste condicional** para que possa seguir com a execução do programa. Você sabe o que é um teste condicional e quais estruturas de seleção são possíveis de serem implementadas em um algoritmo? Esse tema também será tratado no capítulo.

## Objetivo de aprendizagem:

- × Aplicar estruturas de seleção e suas características em problemas computacionais.

### 5.1 Estruturas de seleção

Quando falamos em programação, a estrutura de seleção é um comando utilizado quando precisamos decidir sobre algo ou alguma coisa. O que isso quer dizer? Em outras palavras, é quando precisamos tomar uma **decisão** diante dos dados que temos e, para isso, fazemos um **teste condicional**, que possui apenas duas respostas esperadas: verdadeiro ou falso (valor lógico). Quando o teste condicional resulta em um valor **verdadeiro**, a execução do programa segue por uma sequência de comandos que estão de acordo com o teste condicional com esse resultado. Mas, e se o teste condicional resultar em um valor **falso**? Então, a execução do programa seguirá por outra sequência de comandos, considerando este outro resultado do teste condicional.

É importante lembrarmos que o teste condicional fará um **desvio** na execução do programa. Para que serve? Esse desvio condicional é usado para **decidir** se um conjunto de comandos deve ou não ser executado. Ou seja, para que exista um teste condicional é preciso, primeiramente, que exista uma **condição**. Ela é formulada com o uso dos **operadores relacionais** e **operadores lógicos** que têm como resultado os valores lógicos: verdadeiro (V) ou falso (F). Se tivermos uma situação onde queremos classificar um aluno pela sua média, poderíamos utilizar os operadores relacionais e operadores lógicos, como exemplo:

**Se** (Media <4) **Entao**

Escreva (“ O aluno esta reprovado”)

**Se** (Media < 7) **E** (Media >=4) **Entao**

Escreva (“ O aluno está em recuperação”),

**Se** ( Media >= 7,) **Entao**

Escreva (“O aluno está aprovado”)

Neste exemplo podemos perceber a aplicação dos operadores lógicos e relacionais como: <= , >= para classificar a média de um aluno.

Para lidar com os operadores relacionais e operadores lógicos, temos que saber suas aplicações. O Quadro 1 abaixo apresenta os operadores relacionais com os quais poderemos trabalhar dentro de um programa.

Quadro 1 – Operadores relacionais.

| Operadores relacionais | Descrição      |
|------------------------|----------------|
| >                      | Maior          |
| <                      | Menor          |
| >=                     | Maior ou igual |
| <=                     | Menor ou igual |
| =                      | Igual          |
| <>                     | Diferente      |

Fonte: Elaborado pelo autor, 2015.

As operações realizadas com os operadores relacionais estão associadas à comparação de valores. As expressões lógicas são utilizadas em estruturas de decisão e retornam verdadeiro ou falso para a operação. Vamos trabalhar com quatro expressões lógicas: **nao**, **ou**, **e**, **xou**, para as operações de construção de algoritmos.

### Saiba mais

Para assimilarmos melhor esse conteúdo assista o vídeo que mostra em detalhes os operadores lógicos e matemáticos que iremos utilizar durante essa aula: <<https://www.youtube.com/watch?v=yEvnBYjc0k>> e <<https://www.youtube.com/watch?v=LBFGJqPaow0>>.

## 5.2 Tipos de estruturas de seleção

Segundo FORBELLONE (2005), temos dois tipos de estruturas de seleção, as de seleção simples e as de seleção composta. Os tipos de estruturas de seleção são utilizados de acordo com o problema a ser resolvido. Os operadores relacionais e lógicos são muito importantes na elaboração do **teste condicional**.

### 5.2.1 Estruturas de seleção simples: SE... ENTÃO

Uma estrutura de seleção simples SE... ENTÃO é formada pelo comando SE e por um **teste condicional**, conforme a sintaxe apresentada na Figura 1.

Figura 1 – Sintaxe da estrutura de seleção simples com um comando.

|  |
|--|
| se<teste condicional>então<br>comando1 |
|--|

Fonte: Elaborada pelo autor, 2015.

#### Importante

Nos comandos de condição, não use o acento no comando “**então**”; Os acentos nos comandos provocará um erro na hora da interpretação da sintaxe do comando, e o compilador mostrará um erro fazendo com que o programa não seja executado

Se o teste condicional for verdadeiro, o programa executará o comando que está subordinado à estrutura de seleção SE... ENTÃO.

Exemplo: Se atribuirmos valores para as variáveis A=5, B=10, poderemos realizar essa comparação usando uma estrutura de seleção simples, como mostra a Figura 2, a seguir.

Figura 2 – Estrutura de seleção simples.

|  |
|--|
| SE (B>A) ENTÃO<br>Comando1<br>SE (A>B) ENTÃO<br>Comando2 |
|--|

Fonte: Elaborada pelo autor, 2015.

Se o teste condicional for falso para a primeira opção (B>A), o programa **não** executará o comando e seguirá com a execução, testando outras opções.

Para estruturas de condições simples que usam mais comandos, usamos a sintaxe apresentada na Figura 3, a seguir.

Figura 3 – Sintaxe da estrutura de seleção simples com mais de um comando.

```
se<teste condicional>entao
    comando1
    comando2
    comando3
    comando4
fimse l>entao
    comando1
```

Fonte: Elaborada pelo autor, 2015.

Note que a sintaxe da estrutura de seleção simples com mais de um comando necessita de uma delimitação, que é dada com o FIMSE. O FIMSE especifica que todos os comandos que estão dentro da estrutura de seleção devem ser executados, caso o teste condicional seja verdadeiro. O FIMSE assegura que todos os comandos serão executados. Vamos praticar utilizando o *software* VisuAlg. No exemplo, vamos implementar um algoritmo mostrando uma estrutura simples de comandos de condição.

**Proposta:** Abra o VisuAlg e implemente um algoritmo que receba um número e mostre duas mensagens: número é maior que 10 e algoritmo executado com sucesso. (Veja a Figura 4.)

---

---

**Sintaxe:** é um conjunto de regras que define a forma de uma linguagem de programação, estabelecendo como são compostas a sua linguagem e suas estruturas básicas (palavras).

---

---

Figura 4 – Sintaxe da estrutura de seleção simples com mais de um comando.

```
algoritmo "estrutura de condição simples com fimse"

var
numero: inteiro

inicio
// Seção de Comandos
escreva ("Digite o primeiro número:")
leia (numero)
se numero > 10 entao
    escreval ("Algoritmo Executado com sucesso")
    escreval ("O número é maior que 10")
fimse

finalgoritmo
```

Fonte: elaborada pelo autor, 2015.

### Importante

Na proposta implementada, percebemos que podemos utilizar os comandos de condição para verificar a condição de um número, se este é verdadeiro ou não. Observe que, juntamente com o comando de condição SE, foram utilizados os operadores relacionais. Eles sempre serão utilizados quando usarmos o comando de condição SE... ENTAO.

### Saiba mais

Os comandos de Seleção são usados com muita frequência na construção de algoritmos, por isso, é importante assimilar bem seus conceitos. Acesse o site <<https://www.youtube.com/watch?v=mmHfui8nenw>> e assista de forma detalhada esse conceito.

## 5.2.2 Estruturas de seleção composta:

### SE... ENTAO... SENAO

Segundo ASCENCIO (2012), a estrutura de seleção composta também é implementada pelo comando SE... ENTAO, por um **teste condicional**, mas inclui o comando ELSE, conforme a sintaxe mostrada na Figura 5, a seguir.

**Figura 5** – Sintaxe da estrutura de seleção composta SE... ENTÃO... SENÃO com um comando.

```

se<teste condicional>entao
    comando1
    senao
        comando2
    fimse
  
```

Fonte: Elaborada pelo autor, 2015.

A estrutura de seleção composta também é implementada a partir de um **teste condicional**. Nele, o **comando 1** será executado se o teste condicional for verdadeiro. Por outro lado, se o teste condicional for falso, o **comando 2** será executado e o comando 1, não.

Uma estrutura de seleção composta também pode necessitar que o programa execute mais de um comando, o que também exigirá a inserção do comando FIMSE, como mostrado na Figura 6 a seguir.

---



---

**Comando:** Na linguagem de programação podemos executar comandos ao computador. Esses comandos podem ser de processamento, entrada de dados ou saída de dados.

---



---

**Figura 6** – Sintaxe da estrutura de seleção composta SE... ENTÃO... SENÃO com mais de um comando.

```

se<teste condicional>entao
    comando1
    comando2
    comando3
    senao
        comando4
        comando5
        comando6
    FIMSE
  
```

Fonte: Elaborada pelo autor, 2015.



## Proposta

Abra o VisuAlg e implemente um algoritmo que leia dois números e mostre qual desses números é maior, utilizando os comandos SE... ENTÃO... SENÃO. Veja a Figura 7, a seguir.

Figura 7 – Sintaxe da estrutura de seleção composta SE... ENTÃO... SENÃO com um comando, utilizando o *Software* VisuAlg.

```
algoritmo "Estrutura de Seleção Composta SE .. ENTÃO .. SENÃO"

var
num1, num2: inteiro

inicio
// Seção de Comandos
escreva ("Digite o primeiro número:")
leia (num1)
escreva ("Digite o primeiro número:")
leia (num2)

se num1 > num2 entao
    escreva ("O primeiro número", num1, "é maior que o segundo", num2)
senao
    escreva ("O segundo número", num2, "é maior que o primeiro", num1)
fimse

finalgoritmo
```

Fonte: Elaborada pelo autor, 2015.

## Importante

Na proposta implementada no VisuAlg (Figura 7), percebemos como podemos utilizar os comandos de condição para verificar as saídas: caso seja verdadeiro, a saída será uma; caso seja falso, a saída será outra.

### 5.2.3 Estrutura de seleção composta: CASO

A estrutura de seleção composta ou múltipla *CASO* é utilizada quando existem situações mutuamente exclusivas, isto é, se uma situação for executada, as demais não serão. Veja a Figura 8.

Figura 8 – Sintaxe da estrutura de seleção composta.

```

        ESCOLHA<variável>
        CASO<valor1>: comando1
        CASO<valor2>: comando2
        CASO<valor3>: comando3
        CASO<valor4>: comando4
        CASO<valor5>: comando5
        OUTRAOPCAO
        comando6
        FIMESCOLHA
    
```

Fonte: Elaborada pelo autor, 2015.

A estrutura de seleção composta ESCOLHA... CASO testa o conteúdo de uma variável. Caso o conteúdo dessa variável seja verdadeiro em uma das opções, isto é, para algum dos casos dentro da estrutura ESCOLHA... CASO, o comando ou uma lista de comandos é executada e encerra a execução da estrutura ESCOLHA... CASO. Se nenhuma das opções de teste for verdadeira, o comando OUTRAOPCAO será executado e, por fim, a estrutura de seleção ESCOLHA... CASO será encerrada.

Em outras palavras, o comando ESCOLHA... CASO avalia o valor de uma variável para decidir qual caso será executado. Cada caso está associado a um possível valor da variável, que deve ser, obrigatoriamente, do tipo caractere ou inteiro. Outros tipos de dados não são aceitos na maioria das linguagens de programação. Após a execução de uma das opções do caso, a estrutura de seleção será imediatamente encerrada.

### Saiba mais

Uma forma prática de substituímos os comandos de seleção de forma otimizada é usarmos o comando CASO-ESCOLHA. Assista ao vídeo e visualize suas aplicações na construção de algoritmos: <<https://www.youtube.com/watch?v=5FNebG7sBP4>>.

Proposta: Abra o VisuAlg e implemente um algoritmo no qual o usuário adicionará dois valores. Utilizando o comando ESCOLHA... CASO... OUTROCASO, decida qual operação realizar. Veja Figura 9.

Figura 9 – Utilização do Comando CASO, ESCOLHA, OUTROCASO .

```
Algoritmo "Escolha , Caso, Cutraopcao"
var
num1, num2, total: real
operador: caracer
inicio
// Seção de Comandos
Escreva ("Entre com o primeiro valor:")
Leia(n1)
Escreva ("Entre com o segundo valor:")
Leia(n2)
Escreva ("Informe a operação que deseja realizar: ")
Leia(operador)
escolha operador
Caso "+"
total <- n1 - n2
Escreva ("O valor da soma é:" , total)
Caso "-"
total <- n1 + n2
Escreva ("O valor da subtração é:" , total)
Caso "*"
total <- n1 * n2
Escreva ("O valor da multiplicação é:" , total)
Caso "/"
total <- n1 / n2
Escreva ("O valor da divisão é:" , total)
outrocaso
Escreva ("Opção Errada")
Fimescolha
finalgoritmo
```

Fonte: Elaborada pelo autor, 2015.

Neste exercício implementado no visuAlg, podemos perceber a declaração de uma variável, chamada operador do tipo caractere. Quando usamos uma variável caractere e queremos comparar, colocamos seu conteúdo entre aspas duplas.

No início, pedimos para o usuário adicionar dois números e, após esses valores serem armazenados na memória, perguntamos qual opção ele deseja. Para isso, utilizamos o comando ESCOLHA... CASO, em que verificaremos qual opção foi digitada pelo usuário. Dependendo da opção desejada, será realizado o cálculo da operação escolhida.

### Você sabia

A estrutura ESCOLHA... CASO (em inglês SWITCH-CASE) é uma solução elegante quanto se têm várias estruturas de decisão (SE... ENTÃO... SENÃO) aninhadas. Isto é, outras verificações são feitas caso a anterior tenha falhado (ou seja, o fluxo do algoritmo entrou no bloco SENÃO). A proposta da estrutura CASO é permitir que se vá direto ao bloco de código desejado, dependendo do valor de uma variável de verificação.

Vamos entender de forma mais detalhada a utilização de dois comandos de condição realizando alguns exercícios na plataforma do VisuAlg.

## Saiba mais

Para compreendermos melhor as estruturas de seleção, realize os exercícios 13,14,15,16 do site <<http://partilho.com.br/visualg/exercicios-visualg/visualg-lista-de-exercicios/>>.

## 5.3 Exemplos de estruturas de seleção simples com um comando

O exemplo apresentado na Figura 10 mostra a leitura de notas de um estudante, o cálculo da média ponderada e a exibição de uma mensagem com o conceito final. Neste exemplo, a estrutura de seleção simples SE... ENTÃO é utilizada para apresentar a mensagem com o conceito final do estudante de acordo com o resultado da média ponderada das notas de um trabalho, de uma avaliação semestral e do exame final.

Figura 10 – Sintaxe da estrutura de seleção simples implementada no VisuAlg.

```

algoritmo " Estrutura Simples Se Entao Fimse"
Var
    nota_trab. aval_sem, exame, media : real
inicio
    escreva ("Digite a nota do trabalhohol:")
    leia (nota_trab)
    escreva ("Digite a nota da avaliação semestral:")
    leia (aval_sem)
    escreva ("Digite a nota do exame final:")
    leia (exame)
    media <- ((nota_trab * 2 + aval_sem * 3 + exame * 5) / 10)
    escreva ("Média ponderada:", media)
    se ((media >= 8) E (media<=10)) entao
        escreval ("Obteve conceito A")
    fimse
    se ((media >= 7) E (media<8)) entao
        escreval ("Obteve conceito B")
    fimse
    se ((media >= 6) E (media<7)) entao
        escreval ("Obteve conceito C")
    fimse
    se ((media >= 5) E (media<6)) entao
        escreval ("Obteve conceito D")
    fimse
    se ((media >= 0) E (media<5)) entao
        escreval ("Obteve conceito E")
    fimse
fimalgoritmo

```

Fonte: Elaborada pelo autor, 2015.

**Passo 1.** Iniciamos nosso algoritmo com a declaração de quatro variáveis do tipo real. São elas: nota\_trab, aval\_sem, exame, media. O tipo real é definido para variáveis que possuem valores após a vírgula, conforme Figura 11, a seguir.

Figura 11 – Declaração de variáveis implementadas no VisuAlg.

```
algoritmo "Estrutura Simples Se Entao Fimse"  
  
Var  
    nota_trab, aval_sem, exame, media : real
```

Fonte: Elaborada pelo autor, 2015.

**Passo 2.** Pedimos ao usuário que digite os valores para a nota do trabalho 1, nota de avaliação semestral, nota do exame final. Esses valores serão armazenados nas variáveis criadas na declaração do programa (nota\_trab, aval\_sem, exame). Nessa etapa é realizada a leitura e os valores são armazenados. (Veja a Figura 12.)

Figura 12 – Leitura e escrita implementadas no VisuAlg.

```
escreva ("Digite a nota do trabalho:")  
    leia (nota_trab)  
escreva ("Digite a nota da avaliação semestral:")  
    leia (aval_sem)  
escreva ("Digite a nota do exame final:")  
    leia (exame)
```

Fonte: Elaborada pelo autor, 2015.

**Passo 3.** Nesta etapa do programa é realizado o cálculo ou o processamento da operação, sendo que a variável média recebe os valores armazenados pelos dados digitados dos usuários, conforme mostrado abaixo, na Figura 13.

---

---

**Processamento da operação:** quando as variáveis que foram inseridas pelo usuário são processadas pela Unidade central de Processamento (CPU).

---

---

Figura 13 – Cálculo da média implementada no VisuAlg.

```
media <- ((nota_trab * 2 + aval_sem * 3 + exame * 5) / 10)
```

Fonte: Elaborada pelo autor, 2015.

**Passo 4.** Nesta etapa, a variável “media” armazenou o cálculo realizado pelo processador; então poderemos utilizar o comando simples de seleção para saber com que conceito o aluno ficou. Veja a Figura 14.

Figura 14 – Comando SE... ENTÃO implementado no VisuAlg.

```
escreva ("Média ponderada:", media)
se ((media >= 8) E (media<=10)) entao
    escreval ("Obteve conceito A")
fimse
se ((media >= 7) E (media<8)) entao
    escreval ("Obteve conceito B")
fimse
se ((media >= 6) E (media<7)) entao
    escreval ("Obteve conceito C")
fimse
se ((media >= 5) E (media<6)) entao
    escreval ("Obteve conceito D")
fimse
se ((media >= 0) E (media<5)) entao
    escreval ("Obteve conceito E")
fimse
fimalgoritmn
```

Fonte: Elaborada pelo autor, 2015.

## Importante

Neste exercício, os operadores relacionais e operadores lógicos foram utilizados para saber o conceito do aluno. Os operadores lógicos e relacionais sempre trabalham juntos. Observe os parênteses utilizados para trabalhar com números e variáveis. O operador lógico “E” será verdadeiro, ou seja, aparecerá na tela se atender às duas condições propostas em cada seleção realizada.

## Você sabia

Os comandos de seleção são utilizados em todas as linguagens de programação, como: JAVA, DELPHI, PHP, #NET etc. A sintaxe poderá mudar, mas o conceito é o mesmo.

## 5.4 Exemplos de estruturas de seleção simples com mais de um comando

O exemplo apresentado na Figura15 mostra a leitura de três notas de um estudante, o cálculo da média aritmética e a exibição de uma mensagem com o conceito final. Se o estudante tiver direito a fazer o exame final, surgirá também uma mensagem personalizada, com a nota que ele deverá tirar neste exame final para ser aprovado. Neste exemplo, a estrutura de seleção simples SE... ENTÃO é utilizada com mais de um comando para mostrar que nota o estudante precisará obter no exame final.

Figura 15 – Estruturas de seleção simples com mais de um comando implementadas no VisuAlg.

algoritmo " Exemplos de Estruturas de Seleção Simples com mais de um Comando"

```
var
NOTA1, NOTA2, NOTAS, MEDIA : REAL
inicio
//Seção de Comandos
ESCREVA("DIGITE A PRIMEIRA NOTA: ")
LEIA(NOTA1)
ESCREVA("DIGITE A SEGUNDA NOTA: ")
LEIA(NOTA2)
ESCREVA("DIGITE A TERCEIRA NOTA: ")
LEIA(NOTA3)

MEDIA <- (NOTA1 + NOTA2 + NOTAS) / 3;

SE MEDIA <= 4 ENTÃO
    ESCRVA("A MEDIA DO ALUNO FOI: ", MEDIA)
    ESCREVAL (" - ALUNO REPROVADO")
FIMSE
SE MEDIA <= 6.9 ENTÃO
    ESCRVA("A MEDIA DO ALUNO FOI: ", MEDIA)
    ESCREVAL (" - ALUNO DE RECUPERAÇÃO ")
FIMSE
SE MEDIA <= 7 ENTÃO
    ESCRVA("A MEDIA DO ALUNO FOI: ", MEDIA)
    ESCREVAL (" - ALUNO APROVADO ")
FIMSE
finalgoritmo
```

Fonte: Elaborada pelo autor, 2015.

## Explicação

**Passo 1.** Iniciamos nosso algoritmo com a declaração de cinco variáveis do tipo real. São elas: nota1, nota2, nota3, media. Os tipos reais são definidos para variáveis que possuem valores após a vírgula. Veja a Figura 16.

Figura 16 – Declaração de variáveis implementadas no VisuAlg.

```
var
NOTA1, NOTA2, NOTA3, MEDIA : REAL
```

Fonte: Elaborada pelo autor, 2015.

**Passo 2.** Pedimos ao usuário que digite os valores para nota1, nota2, nota3. Esses valores serão armazenados nas variáveis criadas na declaração do programa (nota1, nota2, nota3). Nessa etapa é realizada a leitura e os valores são armazenados. Os valores para as variáveis média serão processados pelo computador; portanto, não há necessidade de o usuário entrar com eles. Veja a Figura 17.

Figura 17 – Leitura e escrita no VisuAlg.

```
ESCREVA("DIGITE A PRIMEIRA NOTA: ")
LEIA(NOTA1)
ESCREVA("DIGITE A SEGUNDA NOTA: ")
LEIA(NOTA2)
ESCREVA("DIGITE A TERCEIRA NOTA: ")
LEIA(NOTA3)
```

Fonte: Elaborada pelo autor, 2015.

**Passo 3.** Nesta etapa do programa é realizado o cálculo ou o processamento da operação, sendo que a variável média receberá os valores armazenados pelos dados digitados dos usuários, conforme Figura 18, a seguir.

Figura 18 – Cálculo da média implementados no VisuAlg.

```
MEDIA <- (NOTA1 + NOTA2 + NOTA3) / 3;
```

Fonte: Elaborada pelo autor, 2015.

**Passo 4.** Nesta etapa, a variável “media” armazenou o cálculo realizado pelo processador; então podemos utilizar o comando simples de seleção para saber com que conceito o aluno ficou (conforme Figura 19).



Figura 19 – Utilização dos comandos SE... ENTAO com mais de um comando implementados no VisuAlg.

```
SE MEDIA <= 4 ENTAO
    ESCREVA("A MEDIA DO ALUNO FOI: ", MEDIA)
    ESCREVA1 (" - ALUNO REPROVADO ")
FIMSE
SE MEDIA <= 6.9 ENTAO
    ESCREVA("A MEDIA DO ALUNO FOI: ", MEDIA)
    ESCREVA1 (" - ALUNO DE RECUPERAÇÃO ")
FIMSE
SE MEDIA <= 7 ENTAO
    ESCREVA("A MEDIA DO ALUNO FOI: ", MEDIA)
    ESCREVA1 (" - ALUNO APROVADO ")
FIMSE
finalgoritmo
```

Fonte: Elaborada pelo autor, 2015.

Observe que podemos utilizar uma estrutura de seleção simples com mais de um comando, como destacado em vermelho.

## 5.5 Exemplos de estruturas de seleção

O exemplo apresentado no Figura 20 mostra a leitura de um número e o algoritmo verifica se esse número é par ou ímpar com o auxílio de uma estrutura de seleção composta SE... ENTAO... SENAO.

Figura 20 – Utilização dos comandos SE... ENTAO... SENAO com mais de um comando implementados no VisuAlg.

```
Var
    num: inteiro
inicio
    escreval ("Digite um número:")
    leia (num)

    se ((num) MOD 2 = 0) entao
        escreval ("Este número é par")
    senao
        escreval ("Este número é ímpar")
    fimse

finalgoritmo
```


Fonte: Elaborada pelo autor, 2015.

O Comando MOD, utilizado na implementação do algoritmo, retorna o resto da divisão. Caso seja 0 (exemplo 2/2), o resto da divisão será zero e o número será par; SENAO, o número será ímpar.



### Saiba mais

Os comandos de seleção compostos são utilizados para resolução de algoritmos que exigem uma verificação extensiva e para contextualizar melhor esse aprendizado. Confira mais no site: <<https://www.youtube.com/watch?v=a8Ew-5OEDL0>>.



---

---

**MOD:** Comando que mostra o resto da divisão inteira. Por exemplo,  $9 \% 2 = 1$ .

---

---

## Resumindo

Neste capítulo apresentamos a estrutura de seleção utilizada para tomada de decisões quando o assunto é programação. Aprendemos que a estrutura de seleção muda o raciocínio de programas simplesmente sequenciais para um raciocínio com base em tomadas de decisões, ou seja, com base em escolhas. Por fim, vimos o que é uma estrutura de seleção, suas sintaxes, funcionalidades, os tipos simples e composto, bem como exemplos de cada um dos tipos.



# 6

## Estruturas de repetição

NESTE CAPÍTULO APRESENTAREMOS as **estruturas de repetição** em lógica de programação. Estas estruturas são usadas de maneira especial quando temos que resolver problemas que precisam que determinadas partes do programa sejam repetidas um determinado número de vezes para que a solução do problema seja alcançada. São as chamadas estruturas de repetição.

A utilização dessa técnica computacional agiliza a estrutura do algoritmo, podendo o programador fazer uso desse recurso quantas vezes forem necessárias para otimizar seus programas.

Pronto para começar? Então, vamos juntos.

## Objetivo de aprendizagem:

- × Analisar e aplicar estruturas de repetição e suas características em problemas computacionais.

## 6.1 Estruturas de repetição

Segundo Forbellone (2005), uma estrutura de repetição em programação é um comando utilizado quando precisamos literalmente **repetir** algo; em outras palavras, é quando há necessidade de repetir um ou mais comandos. Essa repetição é realizada até que um **teste condicional** seja verdadeiro, isto é, a execução do(s) comando(s) será realizada enquanto uma condição for satisfeita.

Segundo Xavier (2007), uma estrutura de repetição é utilizada quando uma parte do programa – ou mesmo o programa inteiro – precisa ser repetida. O número de repetições pode ser fixo ou estar relacionado a uma condição. Dessa forma, existem estruturas de repetição para cada situação.

## 6.2 Tipos de estruturas de repetição

Forbellone (2005) aborda três tipos de estruturas de repetição:

- × PARA... FACA: estrutura de repetição para número definido de repetições;
- × ENQUANTO... FACA: estrutura de repetição para número indefinido de repetições e teste no início;
- × REPITA... ATE: estrutura de repetição para número indefinido de repetições e teste no final.

A seguir, detalhamos cada um dos tipos de estruturas de repetição em lógica de programação de computadores.

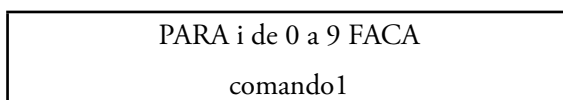
### Importante

Uma das estruturas que os alunos possuem maior dificuldades de assimilar são os laços de repetição, por isso, é importante a resolução de vários exercícios para assimilar o conteúdo.

### 6.2.1 Estrutura de repetição para número definido de repetições

Uma estrutura de repetição é utilizada quando se sabe o número de vezes que uma parte do programa deve ser repetida. O formato geral dessa estrutura é apresentado com o comando PARA, conforme mostra a Figura 1.

Figura 1 – Sintaxe da estrutura de repetição PARA... FACA com um comando.

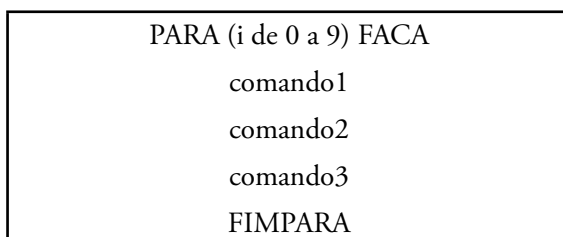


Fonte: Elaborada pelo autor, 2015.

O comando1 será executado utilizando-se a variável  $i$  como variável de controle. Seu conteúdo vai variar do valor inicial ( $i=0$ ) até o valor final ( $i=9$ ). A variável de controle  $i$  será incrementada até que seu conteúdo seja igual a 9. Em outras palavras, até que o **teste condicional não seja verdadeiro**, isto é, quando a variável  $i$  for maior ou igual a 10.

A estrutura de repetição também pode ser usada para a repetição de mais de um comando, isto é, quando precisamos que um bloco de comandos seja repetido por um determinado número de vezes ou até que uma condição seja satisfeita. Para isso, precisamos definir o início e fim do bloco de comandos que serão repetidos, conforme mostrado na Figura 2.

Figura 2 – Sintaxe da estrutura de repetição FACA com mais de um comando.



Fonte: Elaborada pelo autor, 2015.

Note que, assim como a estrutura de seleção, a sintaxe da estrutura de repetição com mais de um comando precisa de um delimitador de início e fim

do bloco de comandos. Os comandos entre o PARA e FIMPARA somente serão executados **enquanto** o teste condicional for verdadeiro.

### Saiba mais

Para aprender mais sobre esse importante comando de repetição, assista um vídeo ratico que descreve passo a passo como implementar. Acesse: <<https://www.youtube.com/watch?v=IQjGDLSRUd0>>.

### Proposta

Abra o VisuAlg e implemente um algoritmo que receba a idade de 75 pessoas e mostre mensagem informando “maior de idade” e “menor de idade” para cada pessoa. Considere a idade a partir de 18 anos como maior de idade. (Veja a Figura 3.)

Figura 3 – Sintaxe da estrutura de repetição PARA... FAÇA com mais de um comando realizado no *software* VisuAlg.

```
algoritmo "estrutura de repetição PARA FAÇA"
var
x, idade: inteiro

inicio
//Seção de Comandos
para x de 1 ate 75 faca
    escreva("Digite a idade: ")
    leia(idade)
    se idade >= 18 entao
        escreval("Fulano é maior de idade!")
    fimse
fimpara
fimalgoritmo
```

Fonte: Elaborada pelo autor, 2015.

### Importante

O programa VisuAlg não faz distinção entre maiúsculas e minúsculas; então podemos escrever LEIA (maiúsculo) e leia (minúsculo), ou ESCREVA (maiúscula) e escreva (minúsculo), que não fará diferença. Mas atenção: isso ocorre apenas no programa VisuAlg.

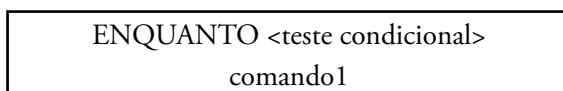
A implementação desse algoritmo no programa VisuAlg mostra algumas situações interessantes. O programa será executado 75 vezes quando o valor da variável “x” for 76; então ele deixará de repetir a operação. Outra situação é a quantidade de comandos que pode ser implementada dentro do laço de repetição. Podemos pedir ao usuário para entrar com os dados, realizar a leitura e a verificação dentro da estrutura de repetição.

### 6.2.2 Estruturas de repetição para número indefinido de repetições e teste no início

Segundo Ascencio e Campos (2012), essa estrutura de repetição é utilizada quando **não** se sabe o **número de vezes** que uma parte do programa deve ser repetida, embora também possa ser utilizada quando se tem tal informação. Essa estrutura baseia-se na análise de uma condição. A repetição será feita enquanto a condição mostrar-se verdadeira. Existem situações em que o teste condicional da estrutura de repetição, que fica no início, resulta em um valor falso logo na primeira comparação. Nesses casos, os comandos de dentro da estrutura de repetição não serão executados. Quando a condição proposta dentro de um laço de repetição não consegue chegar ao seu final e fica repetindo, infinitamente, porque não há condição de parada ou porque a condição existe, mas nunca é atingida, chamamos essa condição de **Loop Infinito**.

O formato geral dessa estrutura é apresentado com o comando ENQUANTO... FAÇA. Analise a Figura 4 a seguir.

Figura 4 – Sintaxe da estrutura de repetição ENQUANTO... FAÇA com o teste condicional no início.



Fonte: Elaborada pelo autor, 2015.

#### Proposta

Abra o VisuAlg e implemente um algoritmo usando o comando ENQUANTO... FAÇA... FIMENQUANTO. Enquanto o valor digitado for diferente de zero, ele realizará a soma do valor digitado até a condição seja aceita. Veja a Figura 5, a seguir.



Figura 5 – Sintaxe da estrutura de repetição ENQUANTO... FACA com mais de um comando realizado no *software* VisuAlg.

```
algoritmo "SomaEnquantoValorDiferenteDe0"
var
    valorDigitado : REAL
    soma : REAL
inicio
    soma := 0
    escreva ("Digite um valor para a soma: ")
    leia (valorDigitado)

    ENQUANTO valorDigitado <> 0 FACA
        soma := soma + valorDigitado
        escreval ("Total: ", soma)
        escreva ("Digite um valor para a soma: ")
        leia(valorDigitado)
    FIMENQUANTO

    escreval ("Resultado: ", soma)

finalgoritmo
```

Fonte: Elaborada pelo autor, 2015.

Essa estrutura de repetição é implementada a partir de um **teste condicional**, isto é, enquanto o teste condicional for verdadeiro, o **comando** será executado. No momento em que o teste condicional for falso, a execução da estrutura de repetição será encerrada e o programa não executará mais o comando.

### Você sabia

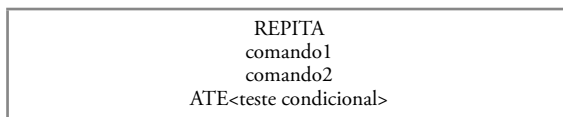
Estruturas de repetição, ou laço de repetição (looping), são usadas dentro da programação, quando precisamos repetir um determinado trecho de código n vezes. A maioria das linguagens de programação utiliza essa técnica para repetir um determinado código dentro de um programa. Podemos citar linguagens como: JAVA, DELPHI, PHP, C, entre outras, que usam essa técnica.

### 6.2.3 Estrutura de repetição para número indefinido de repetições e teste no final

Essa estrutura de repetição é utilizada quando não se sabe o **número de vezes** que uma parte do programa deve ser repetida, embora também possa ser utilizada quando se tem tal informação. Essa estrutura baseia-se na análise

de uma condição. A repetição será feita enquanto a condição mostrar-se verdadeira; nessa situação, o teste condicional da estrutura de repetição fica no final da estrutura, isto é, os comandos serão executados, ao menos uma vez, mesmo que o teste condicional seja falso. O formato geral dessa estrutura é apresentado com o comando REPITA... ATE, como mostra a Figura 6.

Figura 6 – Sintaxe da estrutura de repetição REPITA... ATE com teste no final.



Fonte: Elaborada pelo autor, 2015.

Essa estrutura de repetição é implementada a partir da execução da sequência de comandos, no mínimo, uma vez. No final, verifica-se o **teste condicional**, se ele for falso ou verdadeiro, isso dependerá da forma como o programador irá realizar a condição a estrutura de repetição continua a execução ou a estrutura de controle de repetição é finalizada.

## Saiba mais

Para aprender mais sobre esse importante comando de repetição assista um vídeo pratico que descreve passo a passo como implementar algoritmos com esse comando. (<https://www.youtube.com/watch?v=cgfe08eg85o>)

## Proposta

Abra o VisuAlg e implemente um algoritmo que mostre na tela um número de 1 até 10 usando o comando REPITA... ATE. (Veja a Figura 7.)

Figura 7 – Sintaxe da estrutura de repetição REPITA... ATE com mais de um comando realizado no *software* VisuAlg.

```

var j: inteiro
inicio
j <- 1
repita
    escreva (j : 3)
    j <- j + 1
até j > 10
finalgoritmo
  
```

Fonte: Elaborada pelo autor, 2015.

No exercício, podemos verificar que a variável “j” recebe o valor 1 antes do início do comando de repetição (REPITA... ATE). Após essa atribuição, o comando é implementado e a estrutura será repetida até que a variável “j” seja maior que 10. Enquanto essa condição não for satisfeita, o programa continuará.

Vejam os de forma mais detalhada a utilização de dois comandos mais utilizados: PARA... FAÇA e ENQUANTO... FAÇA.

## 6.3 Exemplo da estrutura de repetição PARA

O comando de repetição PARA... FAÇA estrutura a repetição para um número definido de repetições. Vejam os com mais detalhes a proposta do algoritmo apresentado na Figura 8. Este algoritmo realiza a leitura e informa o nome e o sexo de 56 pessoas. No final, informa o total de homens e de mulheres.

Figura 8 – Programa elaborado com o *software* VisuAlg para a prática do comando PARA... FAÇA.

```
algoritmo "Estrutura de Repetição PARA"

var
nome, sexo: caractere
x, h, n: inteiro

início
// Seção de Comandos
para x de 1 até 5 faça
    limpatela
    escreva("Digite o nome: ")
    leia(nome)
    escreva("H - Homem ou M - Mulher: ")
leia(sexo)
escolha sexo
    caso "H"
        h <- h + 1
    caso "M"
        m <- m + 1
    outrocaso
        escreval("Sexo só pode ser H ou M!")
fimescolha
fimpara
limpatela
escreval("Foram inseridos",h," Homens")
escreval("Foram inseridos",m," Mulheres")

fimalgoritmo
```

Fonte: Elaborada pelo autor, 2015.

## Importante

Para realizar as atividades propostas, utilize o software VisuAlg, implementando passo a passo os exercícios propostos. Vamos à explicação detalhada desses comandos.

**Passo 1.** Teremos que declarar as variáveis que utilizaremos durante a execução do programa. Para isso, declararemos as variáveis sexo, nome e as variáveis que contarão a quantidade de homens e mulheres, e uma variável que atenderá à condição do comando PARA... FACA. A declaração ficará conforme a Figura 9.

Figura 9 – Programa elaborado com o *software* VisuAlg para a prática do comando PARA... FACA – declaração de variáveis.

```
algoritmo "Estrutura de Repetição PARA"
var
nome, sexo: caractere
x, h, n: inteiro
```

Fonte: Elaborada pelo autor, 2015.

**Passo 2.** Após a declaração das variáveis foi utilizada a estrutura do comando PARA... FACA, acompanhada da quantidade de vezes que essa operação será repetida. Observe que após a estrutura do comando de repetição temos um comando novo chamado “limpa tela”. Esse comando é utilizado para limpeza da tela do usuário, limpando as opções de nome e sexo cada vez que for digitado pelo usuário. Após esse comando são realizadas a escrita e a leitura das variáveis “nome” e “sexo”, e armazenadas na memória da máquina. Veja a Figura 10.

Figura 10 – Programa elaborado com o *software* VisuAlg para a prática do comando PARA... FACA – Declaração e leitura dentro do comando PARA... FACA.

```
inicio
// Seção de Comandos
para x de 1 até 5 faca
    limpatela
    escreva("Digite o nome: ")
    leia(nome)
    escreva("H - Homem ou M - Mulher: ")
    leia(sexo)
```

Fonte: Elaborada pelo autor, 2015.

**Passo 3.** Após a entrada dos dados será necessário realizar as verificações usando o comando “ESCOLHA... CASO”, e inserir os acumuladores para saber quantas mulheres e quantos homens foram cadastrados. Caso nenhuma das opções seja por homem ou mulher, ele cairá na opção “OUTROCASO”, e uma mensagem será enviada para a tela. Lembre-se, serão digitados cinco registros, conforme a estrutura do comando PARA... FACA. Veja a Figura 11, a seguir.

Figura 11 – Programa elaborado com o *software* VisuAlg para a prática do comando PARA... FACA – Usando o comando CASO... ESCOLHA.

```
escolha sexo
    caso "H"
        h <- h + 1
    caso "M"
        m <- m + 1
    outrocaso
        escreval("Sexo só pode ser H ou M!")
fimescolha

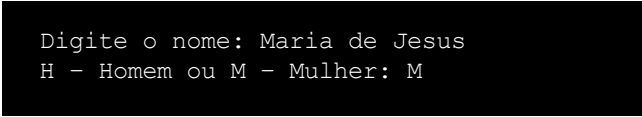
fimpara
limpatela
escreval("Foram inseridos",h," Homens")
escreval("Foram inseridos",m," Mulheres")

finalgoritmo
```

Fonte: Elaborada pelo autor, 2015.

**Passo 4.** A tela de cadastro, com as saídas do programa, executará 5 cadastros, como foi configurado no comando “Para Faca”, ficando a tela de cadastro conforme a **Figura 12**. Ao final dos cadastros a tela mostrará os contadores com as estatísticas, conforme **Figura 13**

Figura 12 - Programa elaborado com o *software* VisuAlg para a prática do comando PARA... FACA – Cadastro dos usuários



```
Digite o nome: Maria de Jesus
H - Homem ou M - Mulher: M
```

Fonte: Elaborada pelo autor, 2015.

Figura 13 - Programa elaborado com o *software* VisuAlg para a prática do comando PARA... FAÇA – dados de saída

```
Foram inseridos 2 Homens
Foram inseridos 3 Mulheres

*** Fim de execução.
*** Feche esta janela para retornar ao VisuAlg.
```

Fonte: Elaborada pelo autor, 2015.

Temos a finalização da estrutura do comando “ESCOLHA – CASO” e do comando de repetição “PARA... FAÇA”. A tela será limpa novamente e os valores que foram acumulados dentro dos contadores serão mostrados com o comando ESCREVA.

### Você sabia

O Comando PARA-FAÇA é um dos comandos mais utilizados dos laços de repetição. A facilidade de sua sintaxe e sua estrutura simples permite a sua usabilidade em programas que exigem a repetição de um código.

## 6.4 Exemplo da estrutura de repetição ENQUANTO

O comando de repetição ENQUANTO... FAÇA será utilizado por uma condição na qual deverá ser testada a cada laço que for realizado. A Figura 14 mostra um algoritmo que recebe os dados de N pessoas (nome, sexo, idade e saúde) e informa se estão aptas ou não a cumprir o serviço militar obrigatório. Informe o total de pessoas. Novamente vamos utilizar um comando de repetição com contadores de registros.

Figura 14 – Exemplo de estrutura de repetição ENQUANTO com teste condicional no início.

```
algoritmo "Exercicio comandos Enquanto- Faca"
var
programa, idade, apto: inteiro
nome, sexo, saude, opc: caractere
totApto, total: inteiro
```

```
inicio
// Seção de Comandos
programa <- 1
enquanto programa = 1 faca
    limpatela
    apto <- 1
    saude <- "B"
    total <- total + 1
    escreva("Digite o nome: ")
    leia(nome)
    escreva("Digite o sexo (M/F): ")
    leia(sexo)
    escreva("Digite a idade: ")
    leia(idade)
    se idade < 18 entao
        apto <- 0
    fimse
    escreval("Digite o estado de saúde: ")
    escreva("(B) Bom - (R) - Ruim - ")
    leia(saude)

    se saude = "R" entao
        apto <- 0
    senao
        se saude <> "B" entao
            apto <- 0
        fimse
    fimse
    se apto = 1 entao
        totApto <- totApto + 1
    fimse
    escreval("Deseja continuar filtrando (S/N)? ")
    leia(opc)
    se opc = "N" entao
        programa <- 0
    fimse
fimenquanto
limpatela
escreval("Resumo geral: ")
escreval("Foram filtrados: ",total," pessoas")
escreval("Aptos: ",totApto)
escreval("")

fimalgoritmo
```

Fonte: Elaborada pelo autor, 2015.

### Saiba mais

Para aprender mais sobre esse importante comando de repetição assista um vídeo pratico que descreve passo a passo como implementar algoritmos com esse comando. (<https://www.youtube.com/watch?v=8-JWuzb-gIE>)

**Passo 1.** Teremos que declarar as variáveis que utilizaremos durante a execução do programa. Para isso, declararemos as variáveis programa, idade, apto (verificar a condição), nome, sexo e saúde, totapto (contadores) e total (contadores). Observe que declaramos várias variáveis com tipos diferentes, pois o algoritmo necessita deles para realizar o cálculo computacional. A declaração fica conforme a Figura 15.

Figura 15 – Programa elaborado com o *software* VisuAlg para a prática do comando ENQUANTO... FACA – Declaração de variáveis.

algoritmo "Exercicio comandos Enquanto- Faca"

```
var
programa, idade, apto: inteiro
nome, sexo, saude, opc: caractere
totApto, total: inteiro
```

Fonte: Elaborada pelo autor, 2015.

**Passo 2.** Após a declaração das variáveis, foi utilizada a estrutura do comando ENQUANTO... FACA, acompanhada da condição em que ela será executada. Para que ela seja executada pelo menos uma vez, a condição deve ser atendida. Para isso, a variável “programa” receberá o valor 1. Teremos que atribuir o valor 1 para a variável “apto” e realizar sua verificação dentro do laço de repetição.

O contador “total <= total +1” recebe o valor 1 e a variável “saúde” recebe o valor de “B”. São realizadas a escrita e leitura das variáveis “nome”, “sexo” e “idade” e armazenadas na memória da máquina. Veja a Figura 16.

Figura 16 – Programa elaborado com o *software* VisuAlg para a prática do comando ENQUANTO... FACA – Leitura e escrita.

```
programa <- 1
enquanto programa = 1 faca
    limpatela
    apto <- 1
    saude <- "B"
    total <- total + 1

    escreva("Digite o nome: ")
    leia(nome)
    escreva("Digite o sexo (M/F): ")
```



```
leia (sexo)
escreva("Digite a idade: ")
leia(idade)
```

Fonte: Elaborada pelo autor, 2015.

Vejamos agora por que atribuímos esses valores a essas variáveis.

**Passo 3.** A primeira verificação após as entradas dos dados é feita pelo comando de condição SE, verificando se o usuário possui mais de 18 anos. Se a idade for menor, a variável “apto” receberá 0 e deixará de ter o valor 1, colocando esse usuário fora da lista dos contadores de Aptos. Analise a Figura 17, a seguir.

---

---

**Entrada de Dados:** quando as variáveis são lidas e armazenadas na memória da máquina pelo usuário ou programadas por meio de comandos solicitados na tela.

---

---

Figura 17 – Programa elaborado com o *software* VisuAlg para a prática do comando ENQUANTO... FACA – Comandos de verificação.

```
escreva ("Digite a idade:")
leia (idade)
se idade < 18 entao
    apto <- 0
fimse
```

Fonte: Elaborada pelo autor, 2015.

**Passo 4.** Após essa verificação, o estado de saúde da pessoa será verificado. As opções apresentadas são BOM e RUIM. Outro comando de condição SE é utilizado para realizar essa verificação. Caso o usuário digite RUIM ou algo diferente de BOM, a variável “apto” receberá 0, não classificando o indivíduo. Mas caso Apto ainda seja igual a 1, então o contador “total <\_ total +1” receberá o valor 1 e a variável “SAÚDE” receberá o valor de “B”. A Figura 18, a seguir, ilustra essa situação.

**Figura 18** – Programa elaborado com o software VisuAlg para a prática do comando ENQUANTO... FACA – Comandos de verificação e a contador.

```

escreval("Digite o estado de saúde: ")
escreva("(B) Bom - (R) - Ruim - ")
leia(saude)

se saude = "R" entao
    apto <- 0
senao
    se saude <> "B" entao
        apto <- 0
    fimse
fimse
se apto = 1 entao
    totApto <- totApto + 1
fimse

```

Fonte: Elaborada pelo autor, 2015.

**Passo 5.** Após essa verificação o contador ficará acumulando a quantidade de pessoas aptas. A cada passagem, também perguntará ao usuário se deseja continuar realizando os cadastros. Quem controla essas repetições é o comando ENQUANTO... FACA, verificando a situação da variável “programa”. Caso o usuário deseje sair, a variável “programa” recebe 0 e mostrará o total de Aptos por meio do comando ESCREVA. Veja a Figura 19, a seguir.

**Figura 19** – Programa elaborado com o *software* VisuAlg para a prática do comando ENQUANTO... FACA – Resumo geral dos resultados.

```

escreval("Deseja continuar filtrando (S/N)? ")
leia(opc)
se opc = "N" entao
    programa <- 0
fimse
fimenquanto
limpatela
escreval("Resumo geral: ")
escreval("Foram filtrados: ",total," pessoas")
escreval("Aptos: ",totApto)
escreval("")

finalgoritmo

```

Fonte: Elaborada pelo autor, 2015.

**Passo 6.** A saída do programa ficará conforme a **Figura 20**, onde a cada cadastro realizado o programa pedirá: nome, sexo, idade e estado de saúde. O programa pedirá também se o usuário deseja continuar. Neste caso entra o comando de repetição e, caso a resposta seja “S”, pedirá novo cadastro com os dados. Mas, caso a resposta seja “N”, o programa se encerra mostrando a estatística dos cadastros, conforme a **Figura 21**, e apenas um cadastro será realizado.

**Figura 20** - Programa elaborado com o *software* VisuAlg para a prática do comando ENQUANTO... FACA – Cadastro de um Usuário

```
Digite o nome: Agnaldo da Costa
Digite o sexo (M/P): M
Digite a idade: 40
Digite o estado de saúde:
(B) Bom - (R) - Ruim - B
Deseja continuar filtrando (S/N):
```

Fonte: Elaborada pelo autor, 2016.

**Figura 21** - Programa elaborado com o *software* VisuAlg para a prática do comando ENQUANTO... FACA – Resultado Estatístico do Cadastro.

```
Resumo geral:
Foram filtrados: 1 pessoas
Aptos: 1
```

Fonte: Elaborada pelo autor, 2016.

### Importante

O programa VisuAlg não utiliza acentos e cedilha, como em FACA, ENTAO. Se forem utilizados, haverá um erro do compilador na hora de compilar o programa.

### Saiba mais

Para revisar todos os comandos aprendidos assista ao vídeo completo que demonstra exemplos e aplicações de estruturas de repetições em: <<https://www.youtube.com/watch?v=yUPCGrj3J08>>.

## Resumindo

Este capítulo apresentou a estrutura de controle de repetição, uma das mais importantes estruturas de controle em programação de computadores. Neste capítulo apresentamos os três tipos de estruturas de repetição: PARA, ENQUANTO e REPITA - ATE. Vimos que a estrutura de repetição PARA é utilizada quando temos um número definido de repetições, isto é, quando sabemos exatamente quantas vezes é necessário que haja a repetição de determinados comandos. Já a estrutura de controle de repetição do tipo ENQUANTO é utilizada quando não sabemos o número definido de repetições, sendo que estas repetições dependem de um teste condicional. E a estrutura de controle de repetição do tipo REPITA... ATE também é utilizada quando não sabemos o número exato de repetições; ela também depende de um teste condicional. No entanto, este teste condicional é executado no final da primeira execução da repetição. Esta é a diferença entre o ENQUANTO e o REPITA... ATE: a primeira estrutura de controle de repetição tem o teste condicional no início e a segunda, no final.



# 7

## Utilização de vetores unidimensionais

VOCÊ SABE COMO armazenar dados em uma variável do tipo **vetor**? Esse é um dos assuntos que trataremos neste capítulo. O vetor é uma variável como qualquer outra, no entanto permite o armazenamento de vários valores do mesmo tipo de dado em uma única variável.

Esse recurso é extremamente importante, pois o programador poderá armazenar uma quantidade maior de dados em uma variável do tipo vetor e poderá manipular esses valores na memória da máquina.

## Objetivo de Aprendizagem:

- × Aplicar o uso de vetores para armazenar, classificar e pesquisar listas e tabelas de valores.

### 7.1 Vetores

Segundo FORBELLONE (2005), um **vetor** é uma variável dividida em várias “caixas”. Cada “caixa” é identificada por um número que se refere à sua posição no vetor. Esse número é chamado de índice do vetor. Em um vetor, cada uma das “caixas” pode armazenar um dado diferente, mas, obrigatoriamente, todos esses dados precisam ser do mesmo tipo. Isso quer dizer que, se um vetor for do tipo de dado **inteiro**, somente **números inteiros** poderão ser armazenados nas “caixas”. Da mesma forma, se o vetor for do tipo de dado **texto**, somente **palavras ou caracteres alfanuméricos** poderão ser inseridos nessas “caixas”.

---

---

**Caracteres alfanumérico:** conjunto de **caracteres** alfabéticos e numéricos utilizado para descrever a coleção de letras latinas e algarismos árabicos ou um texto construído a partir desta coleção.

---

---

#### 7.1.1 Criando um vetor

Para criar um vetor, é necessário definir um **nome**. E como qualquer outra variável simples, também devem ser definidos o **tipo de dado** e o **tamanho** do vetor. Já o tamanho do vetor deve ser entendido como a quantidade de variáveis que vão compor esse vetor, ou seja, a quantidade de “caixas” que ele terá para armazenar os dados, como mostrado na Figura 1.

Por exemplo:

Nome do Vetor: *vetor1*

Tamanho do Vetor: 4

Tipo de Dado do Vetor: *inteiro*

Figura 1 - Representação de um vetor na memória.

|        |   |   |   |   |
|--------|---|---|---|---|
| vetor1 | 0 | 1 | 2 | 3 |
|        |   |   |   |   |

**Exemplo:** Quando se declara a variável **vetor1** são criados os índices e os campos que irão receber os valores para cada índice criado.

Fonte: Elaborada pelo autor, 2015.

Observe que o vetor começa com o número “0”. Quando trabalhamos com um vetor, precisamos pensar em uma variável com várias “caixas”, e cada “caixa” receberá um determinado valor. No caso de nosso exemplo, estamos utilizando um **dado do tipo inteiro**. Isso quer dizer que o **vetor1** pode receber até quatro números inteiros, quem determina essas sequencias são os índices, como mostrado na Figura 2.

Figura 2 - Dados inseridos em um vetor.

|         |     |    |    |      |
|---------|-----|----|----|------|
| Índice: | 0   | 1  | 2  | 3    |
| vetor1  | 676 | 98 | 12 | 1256 |

**vetor1 [3] = 1256** – No índice 3 temos os valores que foram adicionados ao **vetor1**

Fonte: Elaborada pelo autor, 2015.

## Importante

Para criar um vetor, precisamos saber o **nome**, o **tipo de dado** e o **tamanho do vetor**, isto é, quantas “caixinhas” a variável vetor terá para armazenar os dados.

## 7.2 Vetores unidimensionais

Segundo ASCENCIO (2012), na linguagem Portugol podemos utilizar a seguinte sintaxe para declararmos uma variável do tipo vetor.

<variável> : vetor [intervalo] de <tipo de dados>



Em que

<variável> é o nome do vetor;

<intervalo> são dois valores inteiros com “...” entre eles;

<tipo de dados> pode ser inteiro, real, lógico ou caractere.

Os vetores têm 0 como índice do primeiro elemento. Portanto, quando criamos um vetor de inteiros (10) elementos, isso quer dizer que o **índice** do vetor varia de **0 a 9**. Segundo Forbellone (2005), quando o compilador encontrar uma declaração de vetor, automaticamente é reservado um espaço suficiente na memória do computador para armazenar o número de “caixas” especificadas como índice.

Por exemplo:

```
float vetor1[0..9]
```

Essa declaração informa ao compilador que deverá ser reservado  $4 \times 10 = 40$  **bytes** na memória do computador para armazenar o conteúdo desta variável. É importante lembrar que os bytes são reservados sequencialmente, um após o outro. Cada valor armazenado será de 4 bytes perfazendo um total de 40 bytes.

---

---

**Portugol:** Linguagem que foi criada para uso educacional que ajuda na compreensão da lógica e na construção de algoritmos por meio de um software denominado de VisuAlg.

---

---

### 7.2.1 Atribuindo valores a um vetor

Quando desejamos preencher um vetor com dados, é necessário que o endereço da “caixa” seja informado, ou seja, o *índice* do vetor. Como mostrado abaixo, cada “caixa” do vetor possui um *índice* e, para preenchê-lo, precisamos informar a posição do vetor a que queremos atribuir um determinado valor. A seguir, mostramos como fazer a atribuição de valores às posições do *vetor1*.

```

vetor1 [0] = 98
vetor1 [1] = 98
vetor1 [2] = 12
vetor1 [3] = 12,56
vetor1 [4] = 84
vetor1 [5] = 32
vetor1 [6] = 56
vetor1 [7] = 25
vetor1 [8] = 89
vetor1 [9] = 7,7

```

Segundo Ascencio e Campos(2012), as atribuições em vetor são realizadas a partir do comando chamado de *atribuição de valor*, que tem como objetivo inserir um determinado conteúdo na variável vetor, na sua respectiva posição de memória (*índice*), como mostrado acima. Todos os vetores têm o primeiro elemento no *índice* zero (0). Assim, se tomarmos “k” como sendo o tamanho do vetor, a última posição do vetor será a de índice “k-1”. A sintaxe abaixo mostra que foi colocado o inteiro 98 na **primeira** posição do vetor.

```
vetor1[0] = 98
```

E, a sintaxe a seguir, mostra que foi colocado o inteiro 1256 na **última** posição do vetor.

```
vetor1[9] = 7,7
```

## 7.2.2 Preenchendo um vetor

Preencher um vetor significa atribuir valores a todas as suas posições ao mesmo tempo. Dessa forma, devemos implementar um mecanismo que controle o valor do **índice** do vetor. Para controlar esse valor, será criada uma variável do tipo inteiro, chamada “i”.Essa variável será usada em um comando de repetição chamado **PARA- FACA**, conforme exemplo a seguir.

PARA (i de 0 até 3) FAÇA

vetor1[i] = 30

---

**Índice** (do latim *índex*, que significa “o que indica”): é um indício ou um sinal de algo.

---

O preenchimento de um vetor pode ser feito de diversas formas, desde a mais simples, em que é atribuído um tipo inteiro a todas as posições do vetor, até outras formas de preenchimento, as quais exigem uma atenção especial quanto à manipulação da posição que desejamos preencher no vetor.

Nesse momento, precisamos retomar os estudos sobre as estruturas de repetição. Perceba que a sintaxe utilizada no último exemplo foi a da estrutura de repetição **PARA... FAÇA**. Veja também que o **vetor1** será preenchido com o inteiro trinta (30), da posição inicial zero (0) até a posição final tres (3).

Lembre-se que a variável *i* indica a posição no vetor. Essa variável é inicializada com zero (0) para que o compilador da linguagem de programação comece a atribuição de valores na posição **vetor1[0]**. Dessa forma, o **vetor1** será preenchido com o inteiro trinta (30) em todas as suas posições, desde a posição **vetor1[0]** até a posição **vetor1[3]**, que totaliza as quatro (4) posições do vetor.

## Saiba mais

Para aprender mais sobre declaração, formas, tipos e estruturas usando vetores no VisuAlg, acesse o vídeo e faça o exercício proposto.  
<<https://www.youtube.com/watch?v=AGse74xOz3Y>>.

## Proposta

Abra o VisuAlg e implemente um algoritmo utilizando vetores unidimensionais para preencher um vetor com 10 números inteiros e mostrar os números preenchidos usando o comando **PARA... FAÇA**. (Veja a Figura 3, a seguir.)

**Figura 3** - Declaração de vetor utilizando o VisuAlg.

```

algoritmo "Percorrendo Valores de um Vetor"
var
Mat1 : vetor(1..10) de inteiro
x: inteiro
inicio
Escreval ("Digite valores aleatorios para preencher a matriz de 10 posições")
    Para x de 1 até 10 faca
        Escreval ( "Escreva o valor para a posição", x)
        Leia (Mat[x])
    fimpara

para x de 1 ate 10 faca
    Escreval ( "Escreva o valor das posições ", (Mat1[x]))
fimpara
finalgoritmo

```

Fonte: Elaborada pelo autor, 2015.

Neste exercício implementado no VisuAlg, podemos perceber como realizamos a declaração de um vetor, analisando a Figura 4.

**Figura 4** - Como declarar vetores no VisuAlg.

```
Mat1 : vetor (1..10) de inteiro
```

Fonte: Elaborada pelo autor, 2015.

A declaração poderá ser realizada para diversos tipos de dados como real, caracteres, inteiros. O comando de repetição PARA... FACA, foi utilizado para armazenar os dados dentro da variável Mat1. Para utilizar as dez (10) posições da memória da variável Vetor Mat1, tivemos que percorrer esses endereços. No vetor a seguir (Figura 5) podemos visualizar como foi utilizada a estrutura para percorrer cada endereço e guardar as informações.

**Figura 5** - Índices atribuídos ao vetor Mat1

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |   |   |   |

Fonte: Elaborada pelo autor, 2015.

Os números indicam o índice de cada posição da memória da variável Mat1, e a variável x criada começa no zero e vai até o 9, perfazendo um total de dez posições na memória, que serão preenchidas com o comando na Figura 6 abaixo.

Figura 6 - Leitura e preenchimento de dados em um vetor.

```
Para x de 1 até 10 faça  
Escreval ("Escreva o valor para a posição", x)  
Leia (Mat[x])  
fimpara
```

Fonte: Elaborada pelo autor, 2015.

Após as informações serem armazenadas em cada endereço de memória da variável Mat1, usamos outro PARA... FAÇA para buscar os dados e mostrar novamente com o comando ESCREVA. Veja Figura 7, a seguir.

Figura 7 - Armazenamento de dados em um vetor no VisuAlg.

```
Para x de 1 até 10 faça  
Escreval ("Escreva o valor das posições", (Mat1[x]))  
fimpara
```

Fonte: Elaborada pelo autor, 2015.

### Você sabia

Para trabalharmos com vetores unidimensionais, é importante dominarmos os comandos de repetição PARA... FAÇA, que sempre terá uma variável de controle de quantas vezes a estrutura irá se repetir em uma ordem programada pelo programador.

#### × Preenchendo um vetor com intervalo de números determinados

Preencher um vetor com uma sequência de números de 1 a 4, por exemplo, significa que estamos atribuindo valores incrementados em um (1)  $i+1$ , a cada uma das suas posições. Assim, a sintaxe do comando para preenchimento do vetor1 será da seguinte forma:

```
PARA (i de 0 até 4)  
vetor1[i] = i + 1
```

Ou, para preencher um vetor com uma sequência de números de 1 a 4, mas de forma decrescente, isto é, de 4 a 1. Dessa forma, a sintaxe do comando para preenchimento do vetor1 será a seguinte:

PARA (i de 0 até 5)

vetor1[i] = 4 – i

Isso quer dizer que podemos manipular o incremento do índice do vetor para preenchermos as posições de acordo com as necessidades. Podemos utilizar o comando PARA... FAÇA de forma inversa. Em algumas situações esse procedimento ajuda a resolver alguns problemas computacionais.

## Saiba mais

Para aprender mais detalhes sobre a utilização desse comando com vetores assista ao vídeo que mostra de forma pratica como utilizar esse recurso: <<https://www.youtube.com/watch?v=xuVDA5jjRQI>>.

## Proposta

Abra o VisuAlg e implemente um algoritmo que leia 10 números pelo teclado e exiba os números na ordem inversa da que os números foram digitados. Veja a Figura 8.

Figura 8 - Vetores que ordenam em ordem inversa no VisuAlg.

```
algoritmo "vetor"
var
    numeros: vetor (1 . . 10) de inteiro
    i: inteiro
inicio
    para i de 1 ate 10 faca
        escreva ("Digite um numero")
        leia(numeros[i])
    fimpara
    escreva ("Numeros na ordem inversa:")
    para i de 10 ate 1 passo -1 faca
        escreva (numeros[i])
    fimpara
finalgoritmo
```

Fonte: Elaborada pelo autor, 2015.

Observe que nesse exercício é criado um vetor para ser preenchido com dez (10) posições.

Para isso, utilizou-se o comando PARA... FACA na primeira estrutura. A Figura 9, a seguir, ilustra isso.

Figura 9 - Vetores que serão preenchidos utilizando o VisuAlg.

```
para i de 1 ate 10 faca
    escreva ("Digite um numero")
    leia (numeros[i])
fimpara
```

Fonte: Elaborada pelo autor, 2015.

Para invertermos os números que foram digitados, devemos inverter o comando PARA... FACA, ficando a sintaxe mostrada na Figura 10.

Figura 10 - Vetores que serão preenchidos utilizando o VisuAlg.

```
escreva ("Numeros na ordem inversa: ")
para i de 10 ate 1 passo -1 faca
    escreva (numeros[i])

fimpara
```

Fonte: Elaborada pelo autor, 2015.

---

---

**Sintaxe:** é um conjunto de regras que incorpora símbolos usados na definição de uma linguagem de programação.

---

---

### Saiba mais

Para aprender mais sobre esse importante comando de repetição assista um vídeo pratico que descreve passo a passo como implementar algoritmos com esse comando ([https://www.youtube.com/watch?v=OugAXLsal\\_w](https://www.youtube.com/watch?v=OugAXLsal_w))

### × Procurando valores dentro de um vetor de dados

Em algumas situações há necessidade de buscarmos valores dentro de um vetor. Como podemos realizar essa busca, se não soubermos em qual índice da memória foi guardada a informação que desejamos? Vamos fazer esse exercício.

### Proposta

Abra o VisuAlg e implemente um algoritmo que leia um vetor com dez (10) posições de números inteiros. Em seguida, receba um novo valor do usuário e verifique se este valor encontra-se no vetor. (Veja a Figura 11, a seguir.)

Figura 11 - Buscando valores dentro dos vetores utilizando o VisuAlg.

```

algoritmo "busca valor"
var
    numeros: vetor(1..10) de inteiro
    i, valor : inteiro
    encontrou: logico

inicio
    para i de 1 ate 10 faca
        escreva("Digite um número ")
        leia(numeros[i])
    fimpara

    escreva("Digite um número para ser buscado no vetor: ")
    leia(valor)

    encontrou <- falso
    para i de 1 ate 10 faca
        se (numero[i] = valor) entao
            encontrou <- verdadeiro
        fimse
    fimpara
    se encontrou entao
        escreva("O valor se encontra no vetor")
    senao
        escreva("O valor não se encontra no vetor")
    fimse
finalgoritmo
  
```

Fonte: Elaborada pelo autor, 2015.

**Passo 1.** Nesta atividade é criado um vetor de dez (10) posições. O usuário preenche essas posições utilizando o comando PARA... FAÇA. Para isso são criadas as variáveis “numeros”, um vetor de inteiros, a variável “i”, que



estará percorrendo o vetor, o valor que deverá ser encontrado, e uma variável do tipo lógica, chamada de “encontrou”, que utilizaremos caso se encontre o valor solicitado. Veja a Figura 12, a seguir.

Figura 12 - Declaração de variáveis para buscar valores em vetores.

```
var  
    numeros: vetor(1..10) de inteiro  
    i, valor : inteiro  
    encontrou: logico
```

Fonte: Elaborada pelo autor, 2015.

**Passo 2.** Após a criação das variáveis, o comando PARA... FACA é utilizado para percorrer um vetor e preenchê-lo com os valores digitados pelo usuário. Acompanhe pela Figura 13.

Figura 13 - Preenchendo os vetores para buscar valores em vetores.

```
para i de 1 ate 10 faca  
    escreva ("Digite um número")  
    leia (numeros[i])  
fimpara
```

Fonte: Elaborada pelo autor, 2015.

**Passo 3.** Após o vetor ser preenchido, vamos localizar o valor que se deseja localizar. Pedimos esse valor ao usuário e, em seguida, atribuímos o valor falso para a variável “encontrou”. Após essas configurações, utilizaremos o comando PARA... FACA para percorrer novamente os valores digitados para encontrarmos o valor solicitado. Veja a Figura 14, a seguir.

Figura 14 - Buscando valores em vetores.

```
escreva("Digite um número para ser buscado no vetor: ")  
leia(valor)  
  
encontrou <- falso  
para i de 1 ate 10 faca  
    se (numero[i] = valor) entao  
        encontrou <- verdadeiro  
    fimse  
fimpara
```

Fonte: Elaborada pelo autor, 2015.

Se os valores da variável “valor” e variável “numero”, forem iguais “encontrou”, recebe o *status* de verdadeiro e em seguida é mostrada a mensagem de encontrou, conforme código na Figura 15, a seguir.

Figura 15 - Mostrando os valores do vetor.

```
se encontrou entao
    escreva("O valor se encontra no vetor")
senao
    escreva("O valor não se encontra no vetor")
fimse
```

Fonte: Elaborada pelo autor, 2015.

### × Realizando a soma de um vetor com dados de outro vetor

O desafio de desenvolver um algoritmo que realiza a soma de dois vetores nos ajuda a entender melhor as estruturas de uma variável vetor e seus índices, pois deveremos dominar os laços de repetição e os índices desses vetores para percorrer suas estruturas. Vamos implementar esse algoritmo no *software* VisuAlg.

## Saiba mais

Para aprender mais sobre soma utilizando vetores e suas particularidades acesse o vídeo <<https://www.youtube.com/watch?v=aQNc0iY9Ags>> e realize o exercício para fixação.

**Proposta:** Abra o VisuAlg e implemente um algoritmo em que o usuário lerá dois vetores de números inteiros, cada um com 200 posições. Crie um terceiro vetor em que cada valor seja a soma dos valores contidos nas posições respectivas dos vetores originais. Imprima depois os três vetores. Veja a Figura 16, a seguir.

Figura 16 - Realizando a soma de um vetor com dados de outro vetor.

```
algoritmo "2 vetores + 3 vetor de soma"
var

vet1      : vetor [1..20] de real
vet2      : vetor [1..20] de real
vet3      : vetor [1..20] de real
i         : inteiro

inicio
```

```
-
    escreva("Entre com o ", i, "valor: ")
    leia( vet1[i])
fimpara

para i de 1 ate 20 faca
    escreva("Entre com o ", i, "valor: ")
    leia( vet2[i])
fimpara

para i de 1 ate 20 faca
    vet3[i] <- vet1[i] + vet2[i]
fimpara

para i de 1 ate 20 faca
    escreva(vet1[i], " + ", vet2[i], " = ", vet3[i], " ")
fimpara

escreva(" ")
finalgoritmo
```

Fonte: Elaborada pelo autor, 2015.

### Saiba mais

Para aprender mais sobre esse importante comando de repetição assista um vídeo pratico que descreve passo a passo como implementar algoritmos com esse comando. (<https://www.youtube.com/watch?v=IQKnckQrxmc>)

**Passo 1.** Nesta atividade são criados três vetores de 20 posições cada um. O usuário preencherá essas posições utilizando o comando PARA... FACA. Para isso são criadas as variáveis “inteiro”, que percorrerão os vetores, conforme a Figura 17.

Figura 17 - Declaração das variáveis vetores.

```
var

vet1      : vetor      [1..20]      de real
vet2      : vetor      [1..20]      de real
vet3      : vetor      [1..20]      de real
i          : inteiro
```

Fonte: Elaborada pelo autor, 2015.

**Passo 2.** Após a criação das variáveis, o comando PARA... FAÇA é utilizado para percorrer os dois vetores e preenchê-los com os valores digitados pelo usuário. Você poderá visualizar esse código na Figura 18.

Figura 18 - Preenchendo os valores dos dois vetores.

```
para i de ate 20 faca
    escreva("Entre com o ", i, "valor: ")
    leia( vet1[i])
fimpara

para i de ate 20 faca
    escreva("Entre com o ", i, "valor: ")
    leia( vet2[i])
fimpara
```

Fonte: Elaborada pelo autor, 2015.

**Passo 3.** Após o vetor ser preenchido, vamos realizar a operação das somas dos vetores. Novamente devemos percorrer os índices dos vetores para realizar a soma dos vetores  $\text{vet}[i] + \text{vet2}[i]$ , no vetor  $\text{vet3}[i]$ . Veja a Figura 19.

Figura 19 - Realizando a soma de dois vetores em terceiro vetor.

```
para i de 1 ate 20 faca
    vet3[i] <- vet1[i] + vet2[
fimpara
```

Fonte: Elaborada pelo autor, 2015.

**Passo 4.** Após o vetor  $\text{vet}[i]$  ser preenchido, vamos mostrar seu resultado por meio do comando ESCREVA. Observe que para mostrar os dados posso separar as escritas das variáveis vetores por meio de aspas duplas (" "). Acompanhe Figura 20, a seguir.

Figura 20 - Realizando a soma de dois vetores em terceiro vetor.

```
para i de 1 ate 20 faca
    escreva( vet1[1], " + ", vet2[i], " = ", vet3[1], " ")
fimpara
```

Fonte: Elaborada pelo autor, 2015.

## Resumindo

Neste capítulo aprendemos como criar, armazenar, ordenar e pesquisar dados em uma variável do tipo **vetor**. Vimos também como utilizar uma variável do tipo constante para gerenciar o tamanho de um vetor. Descobrimos como ordenar um vetor nas ordens crescente e decrescente e a fazer a leitura de dados digitados pelo usuário para serem armazenados nos vetores e como mostrar esses valores armazenados em um vetor.

# 8

## Utilização de Vetores Multidimensionais

CARO ALUNO, NESTE capítulo você irá aprender sobre a utilização dos vetores multidimensionais, também conhecidos como matrizes. Matriz é um tipo de dado muito utilizado na área de programação cujas principais características são armazenar dados de mesmo tipo e possuir mais de uma dimensão, esta geralmente representada por linhas (dimensão um) e colunas (dimensão dois). Durante o capítulo vamos analisar e compreender como as matrizes podem ser utilizadas para armazenar, classificar e pesquisar listas e tabelas de valores. Para isso, aprenderemos o que são estes vetores multidimensionais, conhecendo exemplos de vetores bidimensionais e tridimensionais e entendendo como é a sua utilização na área de programação, desde o surgimento do conceito na área da matemática; finalmente, conheceremos os principais algoritmos utilizados em algumas das aplicações que envolvem matrizes, tais como soma e multiplicação de matrizes, cálculo de determinantes, dentre outros. Tenha um ótimo estudo!

Objetivo de Aprendizagem:

- × Aplicar o uso de vetores para armazenar, classificar e pesquisar listas e tabelas de valores.

8.1 O que são Vetores Multidimensionais?

Vetores Multidimensionais, também conhecidos como Variáveis Compostas Multidimensionais, ou simplesmente Matrizes, são conjuntos de dados referenciados por um mesmo nome e que necessitam de dois ou mais índices para que seus elementos sejam individualizados. Cada índice de uma matriz representa uma dimensão. A figura 1, a seguir, mostra o exemplo da matriz M, com duas dimensões, representadas pelas linhas e colunas.

Figura 1 - Matriz M Bidimensional.

| Matriz: M (Bidimensional) |   |                   |        |        |        |        |
|---------------------------|---|-------------------|--------|--------|--------|--------|
|                           |   | Índice de Colunas |        |        |        |        |
|                           |   | 1                 | 2      | 3      | 4      | 5      |
| Índices de Linhas         | 1 | M[1,1]            | M[1,2] | M[1,3] | M[1,4] | M[1,5] |
|                           | 2 | M[2,1]            | M[2,2] | M[2,3] | M[2,4] | M[2,5] |
|                           | 3 | M[3,1]            | M[3,2] | M[3,3] | M[3,4] | M[3,5] |
|                           | 4 | M[4,1]            | M[4,2] | M[4,3] | M[4,4] | M[4,5] |
|                           | 5 | M[5,1]            | M[5,2] | M[5,3] | M[5,4] | M[5,5] |
|                           | 6 | M[6,1]            | M[6,2] | M[6,3] | M[6,4] | M[6,5] |
|                           | 7 | M[7,1]            | M[7,2] | M[7,3] | M[7,4] | M[7,5] |
|                           | 8 | M[8,1]            | M[8,2] | M[8,3] | M[8,4] | M[8,5] |

Fonte: Elaborada pelo autor, 2016.

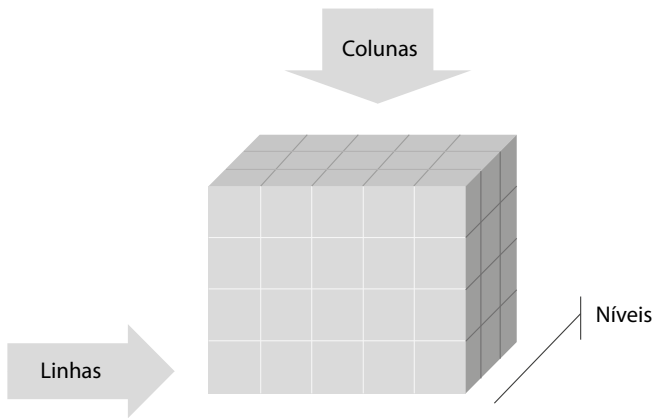
Veja que na Matriz M há 8 linhas e 5 colunas, ambas representadas por índices (1 a 8 para as linhas; 1 a 5 para as colunas). Cada dado armazenado na matriz é referenciado pelo nome da matriz, seguido da linha e coluna (entre

colchetes) onde ele está posicionado. O elemento da posição  $M[5,2]$ , por exemplo, representa o conteúdo da matriz  $M$  na linha 5, coluna 2.

Uma característica importante das matrizes está no fato de que todos os seus elementos (dados) devem ser, necessariamente, de mesmo tipo. Logo, ou uma matriz armazena dados numéricos, ou apenas caracteres, ou somente datas, etc.

Além de matrizes com apenas duas dimensões, conhecidas como bidimensionais, também existem matrizes com três dimensões, ou tridimensionais. Forbellone (FORBELLONE, 2005) explica que para este tipo de matriz repete-se a estrutura bidimensional o mesmo número de vezes que o número dos elementos da terceira dimensão, numerando-as de acordo com os limites especificados na declaração de tipo. A figura 2 mostra um exemplo de uma matriz tridimensional, a Matriz  $T$ :

Figura 2 - Dados inseridos em um vetor.



Fonte: Elaborada pelo autor, 2016.

A principal diferença da matriz tridimensional  $T$ , da figura 2, em relação à matriz bidimensional  $M$ , representada na figura 1, é que os elementos da matriz  $T$  são representados por três índices que indicam, respectivamente, a linha, coluna e nível (ou profundidade) da matriz a qual pertencem. O elemento da posição  $T[3,4,2]$ , por exemplo, representa o conteúdo da matriz  $T$  na linha 3, coluna 4, nível 2.




Através deste exemplo é possível observar então que uma matriz multidimensional é, na verdade, um conjunto de vetores unidimensionais. Não há limites para o número de níveis de uma matriz, entretanto as mais comumente utilizadas são as bidimensionais. Suas principais aplicações serão apresentadas no próximo tópico.



### Saiba mais

Os vetores unidimensionais têm como principal característica a necessidade de apenas um índice para endereçamento de seus dados (FORBELLONE, 2005). Assim como as variáveis compostas multidimensionais, também armazenam conjuntos de valores de mesmo tipo, referenciados por um mesmo nome para todo o conjunto, e por um índice distinto para cada posição.



## 8.2 A utilização de Vetores Multidimensionais na Programação

O estudo das matrizes tem sua origem na matemática, onde a teoria necessária para manipulação de matrizes é estudada na área conhecida como Álgebra Linear. Nela, as matrizes são arranjos retangulares de elementos de um conjunto, organizados em  $m$  linhas e  $n$  colunas, sendo que  $m$  e  $n$  devem ser necessariamente maiores ou iguais a um. Além da matemática, sua teoria pode ser utilizada para várias finalidades em diversas outras áreas relacionadas.

Há usos mais sofisticados como na engenharia civil, por exemplo, onde as matrizes podem ser utilizadas na resolução de cálculos complexos necessários para construção de pontes, prédios, etc. Ou na área de computação gráfica, onde determinados elementos gráficos são representados por *pixels* gerados por uma matriz na tela de um monitor de computador, por exemplo.



### Importante

Pixel, junção dos termos *picture* (fotografia, retrato, quadro, figura, desenho, etc.) e *element* (elemento, componente, etc.), é um ponto luminoso em um monitor que, em conjunto com outros pixels, forma

as imagens que aparecem nesta tela. Quando visualizamos uma imagem bem de perto, é possível identificar pequenos quadrados coloridos que, quando somados, formam uma imagem completa. O pixel é então cada um destes pontos, que representam a menor parte da imagem. O termo resolução de tela, que é utilizado para atribuir quantos pixels em altura e largura uma imagem tem, é empregado atualmente como a principal medida da qualidade para monitores de computador, televisões, etc. Quanto maior a quantidade de pixels por região, melhor a qualidade da imagem.



As matrizes também são muito utilizadas em nosso dia-a-dia para diversas aplicações relativamente simples. Organizações simples de dados utilizando softwares do tipo planilha eletrônica, tabelas de campeonatos, cartões de loteria, calendários, dentre outros, são exemplos de utilização de matrizes.

Uma das loterias mais conhecidas no Brasil, por exemplo, é a Loteca (antiga Loteria Esportiva). Um cartão desta loteria é uma típica aplicação das matrizes. Veja na figura 3 que cada coluna (jogo, coluna 1, x, coluna 2 e data) de uma cartela representa uma coluna da matriz, e cada jogo representa uma linha.

Veja que a matriz da Loteca ilustrada na Figura 3 pode ser representada por uma matriz de 5 colunas por 14 linhas, conforme pode ser observado na Figura 4, a seguir.

Figura 3 - Loteca.

| Jogo | Coluna 1                  | x | Coluna 2         | Data      |
|------|---------------------------|---|------------------|-----------|
| 1    | 2 INTERNACIONAL/RS        |   | CRUZEIRO/MG      | 0 Domingo |
| 2    | 0 GOIÁS/GO                |   | SÃO PAULO/SP     | 1 Domingo |
| 3    | 0 CORITIBA/PR             |   | VASCO DA GAMA/RJ | 0 Domingo |
| 4    | 1 FLAMENGO/RJ             |   | PALMEIRAS/SP     | 2 Domingo |
| 5    | 1 FIGUEIRENSE/SC          |   | FLUMINENSE/RJ    | 0 Domingo |
| 6    | 0 JOINVILLE/SC            |   | GRÊMIO/RS        | 2 Domingo |
| 7    | 5 SANTOS/SP               |   | ATLÉTICO/PR      | 1 Domingo |
| 8    | 3 ATLÉTICO/MG             |   | CHAPECOENSE/SC   | 0 Domingo |
| 9    | 0 PONTE PRETA/SP          |   | SPORT/PE         | 1 Domingo |
| 10   | 1 CORINTHIANS/SP          |   | AVAI/SC          | 1 Domingo |
| 11   | 2 F. EUCLIDES DA CUNHA/BA |   | F. MUNDO NOVO/BA | 1 Domingo |
| 12   | 16 F. LUSACA/BA           |   | F. CATU/BA       | 1 Domingo |
| 13   | 4 F. JUVENTUDE/BA         |   | F. ITABUNA/BA    | 0 Domingo |
| 14   | 8 F. VITÓRIA/BA           |   | F. REDENÇÃO/BA   | 0 Domingo |

Fonte: Caixa Econômica Federal, 2016.

Figura 4 - Matriz Loteca Bidimensional.

| Matriz: Loteca (Bidimensional) |    |                   |                 |                       |                 |                     |
|--------------------------------|----|-------------------|-----------------|-----------------------|-----------------|---------------------|
|                                |    | Índice de Colunas |                 |                       |                 |                     |
|                                |    | 1<br>(Jogo)       | 2<br>(Coluna 1) | 3<br>(Coluna do Meio) | 4<br>(Coluna 2) | 5<br>(Data do Jogo) |
| Índices de Linhas              | 1  | LOTECA[1,1]       | LOTECA[1,2]     | LOTECA[1,3]           | LOTECA[1,4]     | LOTECA[1,5]         |
|                                | 2  | LOTECA[2,1]       | LOTECA[2,2]     | LOTECA[2,3]           | LOTECA[2,4]     | LOTECA[2,5]         |
|                                | 3  | LOTECA[3,1]       | LOTECA[3,2]     | LOTECA[3,3]           | LOTECA[3,4]     | LOTECA[3,5]         |
|                                | 4  | LOTECA[4,1]       | LOTECA[4,2]     | LOTECA[4,3]           | LOTECA[4,4]     | LOTECA[4,5]         |
|                                | 5  | LOTECA[5,1]       | LOTECA[5,2]     | LOTECA[5,3]           | LOTECA[5,4]     | LOTECA[5,5]         |
|                                | 6  | LOTECA[6,1]       | LOTECA[6,2]     | LOTECA[6,3]           | LOTECA[6,4]     | LOTECA[6,5]         |
|                                | 7  | LOTECA[7,1]       | LOTECA[7,2]     | LOTECA[7,3]           | LOTECA[7,4]     | LOTECA[7,5]         |
|                                | 8  | LOTECA[8,1]       | LOTECA[8,2]     | LOTECA[8,3]           | LOTECA[8,4]     | LOTECA[8,5]         |
|                                | 9  | LOTECA[9,1]       | LOTECA[9,2]     | LOTECA[9,3]           | LOTECA[9,4]     | LOTECA[9,5]         |
|                                | 10 | LOTECA[10,1]      | LOTECA[10,2]    | LOTECA[10,3]          | LOTECA[10,4]    | LOTECA[10,5]        |
|                                | 11 | LOTECA[11,1]      | LOTECA[11,2]    | LOTECA[11,3]          | LOTECA[11,4]    | LOTECA[11,5]        |
|                                | 12 | LOTECA[12,1]      | LOTECA[12,2]    | LOTECA[12,3]          | LOTECA[12,4]    | LOTECA[12,5]        |
|                                | 13 | LOTECA[13,1]      | LOTECA[13,2]    | LOTECA[13,3]          | LOTECA[13,4]    | LOTECA[13,5]        |
|                                | 14 | LOTECA[14,1]      | LOTECA[14,2]    | LOTECA[14,3]          | LOTECA[14,4]    | LOTECA[14,5]        |

Fonte: Elaborada pelo autor, 2016.

Para que a cartela da Loteca representada na figura 3 seja armazenada na matriz representada na figura 4, podemos, por exemplo, atribuir o valor 1 às colunas em laranja (coluna 1, meio e coluna 2) e zero às outras posições. Desta forma, a matriz loteca da figura 4 ficaria com os seguintes valores armazenados.

Figura 5 - Matriz Loteca Preenchida.

| Matriz Loteca (Bidimensional). |                 |                       |                 |                     |
|--------------------------------|-----------------|-----------------------|-----------------|---------------------|
| Índice de Colunas              |                 |                       |                 |                     |
| 1<br>(Jogo)                    | 2<br>(Coluna 1) | 3<br>(Coluna do Meio) | 4<br>(Coluna 2) | 5<br>(Data do Jogo) |

|                   |    |    |   |   |   |   |
|-------------------|----|----|---|---|---|---|
| Índices de Linhas | 1  | 1  | 1 | 0 | 0 | 1 |
|                   | 2  | 2  | 0 | 0 | 1 | 1 |
|                   | 3  | 3  | 0 | 1 | 0 | 1 |
|                   | 4  | 4  | 0 | 0 | 1 | 1 |
|                   | 5  | 5  | 1 | 0 | 0 | 1 |
|                   | 6  | 6  | 0 | 0 | 1 | 1 |
|                   | 7  | 7  | 1 | 0 | 0 | 1 |
|                   | 8  | 8  | 1 | 0 | 0 | 1 |
|                   | 9  | 9  | 0 | 0 | 1 | 1 |
|                   | 10 | 10 | 0 | 1 | 0 | 1 |
|                   | 11 | 11 | 1 | 0 | 0 | 1 |
|                   | 12 | 12 | 1 | 0 | 0 | 1 |
|                   | 13 | 13 | 1 | 0 | 0 | 1 |
|                   | 14 | 14 | 1 | 0 | 0 | 1 |

Fonte: Elaborada pelo autor, 2016.

Observe ainda na figura 5 que a coluna Dia do Jogo teve armazenado o valor correspondente ao número do dia da semana em que a partida foi realizada (1 - Domingo, 2 - Segunda, 3 - Terça, 4 - Quarta, 5 - Quinta, 6 - Sexta, 7 - Sábado). Já para identificarmos o vencedor do jogo basta verificar em que coluna posição a matriz está preenchida com valor 1. O jogo 3, por exemplo, foi empate (coluna do meio), já no jogo 9, a vitória foi do visitante (coluna 2).

Além do uso no dia-a-dia, as matrizes também possuem grande aplicação na área de programação (computação), sendo utilizadas na construção de algoritmos para solução de diversos problemas, que são soluções necessárias em diversas outras áreas de conhecimento.

Este tipo de estrutura de dados, justamente pelo seu formato como tabela, é muito utilizado para armazenar, classificar e pesquisar listas e tabelas de valores. O armazenamento dos dados ocorre em cada célula da tabela, e o dado é referenciado pelo número da linha e coluna. A classificação dos dados ocorre através da ordenação dos valores armazenados, o que facilita sua pesquisa.

Existem vários algoritmos de classificação (ordenação) e pesquisa (busca). Para classificação, os mais conhecidos são *Insertionsort*, *Heapsort*, *Bubblesort*,

*Mergesort*, *Quicksort*, *Shellsort* e *Countingsort*. Já os de pesquisa são a pesquisa sequencial ou direta e suas classificações (CORMEN et al, 2002).

Além de tornar as buscas mais eficientes, os algoritmos de ordenação são importantes principalmente por necessidades específicas das aplicações que, por sua vez, cumprem as regras de negócios dos sistemas que elas atendem, por exemplo, ordenação dos alunos por ordem alfabética para emissão do diário de classe; relação de funcionários ordenados por departamento; lista de dependentes por data de nascimento etc.

Cada algoritmo de ordenação possui uma estratégia e desempenho diferentes. A eficiência do algoritmo, sob o ponto de vista de tempo de execução, varia de acordo com o tipo de entrada. Em geral, algoritmos baseados em contagem, como o *Countingsort*, tendem a ser mais rápidos na maioria dos casos, entretanto são de implementação bem mais complexa, se comparados aos algoritmos *Heapsort* e *Quicksort*, por exemplo. Já algoritmos baseados em inserção, como o *Insertionsort*, possuem implementação simples, mas não são eficientes em diversas situações.


Os algoritmos de busca mais utilizados são aqueles de busca direta, como a busca binária (*Binary Search*), por exemplo. Buscas sequenciais, em geral, são lentas, pois percorrem todos os elementos do vetor (unidimensional ou bidimensional) quando uma busca é realizada. Imagine, por exemplo, percorrer todas as posições um vetor com milhões de elementos. É possível imaginar o quanto este algoritmo pode ser lento quando se faz uma busca por um elemento que não está no vetor, por exemplo, que seria o pior caso. A busca percorre as milhões de posições do vetor para descobrir que o valor pesquisado não está em nenhuma de suas posições. Este tipo de busca só faz sentido quando o vetor não está ordenado.

Já nos algoritmos de busca direta é necessário que o vetor esteja ordenado, pois as implementações de busca direta mais comuns utilizam esta premissa como estratégia. Ao pesquisar pelo nome José, por exemplo, em um vetor de nomes ordenados, a ideia é ir diretamente às pessoas cujo nome inicia com a letra J, ao invés de percorrer todas as posições do vetor. Veja que neste caso o tempo de pesquisa tende a ser bem menor na maioria dos casos.



## Saiba mais

Você pode encontrar mais detalhes sobre algoritmos de busca de ordenação no clássico livro Algoritmos: Teoria e Prática (CORMEN et al 2002 Parte II - Págs. 99 a 152). Já exemplos de sua implementação podem ser encontrados em <https://support.microsoft.com/pt-br/kb/169617>.



## 8.3 Vetores Multidimensionais

Os comandos a seguir apresentam a sintaxe básica de declaração dos vetores multidimensionais. As palavras destacadas em **negrito** são chamadas de palavras reservadas ou palavras chave, que são identificadores predefinidos de comandos da linguagem, que possuem significados especiais para o interpretador do algoritmo. Este tipo de termo não pode ser utilizado para outra finalidade, como nomes de variáveis ou constantes, por exemplo.

Sintaxe:

```
tipo <Identificador> = matriz [li1..:lf1, li2..:lf2, ..., lin..:lfn] de <tipo>
```

Onde,

**tipo** – palavra chave para declaração de identificadores.

<Identificador> – nome da variável que identifica a matriz.

**matriz** – palavra chave que indica uma matriz.

li1..:lf1, li2..:lf2, ..., lin..:lfn – São os limites inicial e final do índice da matriz.

<tipo> – tipo da variável (caracteres ou números).

A seguir são apresentados exemplos de declaração de vetores multidimensionais ou matrizes:

1. Declare uma matriz bidimensional Nota, com valores numéricos de 3 linhas por 4 colunas.

```
tipo Nota = matriz [1..3, 1..4] de números;
```

2. Declare uma matriz tridimensional NotaFreq, com valores numéricos de 3 linhas, 4 colunas e 2 níveis.

```
tipo NotaFreq = matriz [1..3, 1..4, 1..2] de números;
```

A figura 6 a seguir apresenta as duas declarações de matrizes utilizando a ferramenta VisuAlg:

Figura 6 -Declaração de matrizes na ferramenta VisuAlg.

```
Nota: Vetor[1..3, 1..4] de inteiro
Nota: Vetor[1..2, 1..4, 1..2] de inteiro
```

Fonte: Elaborada pelo autor, 2016.

Uma vez declarada a matriz, o próximo passo é proceder com a leitura de seu conteúdo. Neste caso, quando utilizamos um vetor unidimensional apenas um laço de repetição e um índice são necessários para sua manipulação. No caso

Figura 7 - Algoritmo “leituramatriz” na ferramenta VisuAlg.

```
Algoritmo "leituramatriz"
Var
  l, c: inteiro
Nota: Vetor[1..3, 1..4] de inteiro

Inicio
  para l de 1 ate 3 passo 1 faca
    para c de 1 ate 4 passo 1 faca
      escreval ("Informe o valor: ")
      leia(Nota[l,c])
    fimpara
  fimpara
FimAlgoritmo
```

Fonte: Elaborada pelo autor, 2016.

das matrizes, é necessário um laço e um índice para cada dimensão. Veja o exemplo a seguir (figura 7) que lê o conteúdo de uma matriz bidimensional chamada “Nota”:

O teste de mesa a seguir mostra como a matriz será preenchida a medida em que os dados (notas - número reais aleatórios que variam de 0 a 10) forem informados, através do comando “leia” na linha 4. Neste teste os valores

entre parênteses indicam que o conteúdo da variável foi lida pelo programa.

| Linha do Algoritmo | l | c | Nota[l,c] |
|--------------------|---|---|-----------|
| 2                  | 1 | ? | ?         |
| 3                  |   | 1 | ?         |
| 4                  |   |   | (5,0)     |
| 3                  |   | 2 | ?         |

| Linha do Algoritmo | l | c | Nota[l,c] |
|--------------------|---|---|-----------|
| 4                  |   |   | (9,9)     |
| 3                  |   | 3 | ?         |
| 4                  |   |   | (3,6)     |
| 3                  |   | 4 | ?         |
| 4                  |   |   | (1,8)     |
| 2                  | 2 | ? | ?         |
| 3                  |   | 1 | ?         |
| 4                  |   |   | (1,5)     |
| 3                  |   | 2 | ?         |
| 4                  |   |   | (3,3)     |
| 3                  |   | 3 | ?         |
| 4                  |   |   | (1,9)     |
| 3                  |   | 4 | ?         |
| 4                  |   |   | (6,7)     |
| 2                  | 3 | ? | ?         |
| 3                  |   | 1 | ?         |
| 4                  |   |   | (1,7)     |
| 3                  |   | 2 | ?         |
| 4                  |   |   | (6,5)     |
| 3                  |   | 3 | ?         |
| 4                  |   |   | (3,9)     |
| 3                  |   | 4 | ?         |
| 4                  |   |   | (4,4)     |

## 8.4 Operações Típicas em Matrizes

Cormen et al. (2002, p. 571) citam diversas operações que podem ser realizadas sobre matrizes, tais como adição (soma), subtração e multiplicação,



cálculo de matriz inversa, ordenação e cálculo de determinantes. Além destas operações, também é possível verificar igualdade entre matrizes, cálculo de matriz transposta e oposta. A seguir, são apresentados os seguintes algoritmos: adição entre matrizes; cálculo de determinante de matriz quadrada de ordem dois; cálculo da soma das linhas e de todos os elementos de uma matriz; multiplicação entre matrizes.

- 1. Adição entre matrizes (A + B): pode ser realizada quando ambas as matrizes possuem as mesmas dimensões, ou seja, a mesma quantidade de linhas e colunas. Basicamente a matriz C, resultante da soma das matrizes A e B, é dada pela soma de cada termo na mesma posição em ambas as matrizes. A fórmula a seguir mostra como este cálculo é feito:  $C[m,n] = A[m,n] + B[m,n]$ , para cada “l” de 1 a m e cada “c” de 1 a n. O algoritmo apresentado na figura 8 a seguir implementa esta soma:

Figura 8 - Algoritmo “somadematrizes” na ferramenta VisuAlg.

```
Algoritmo "somadematrizes"
Var
    l, cl: inteiro
    A: Vetor[1..3, 1..4] de inteiro
    B: Vetor[1..3, 1..4] de inteiro
    C: Vetor[1..3, 1..4] de inteiro
Inicio
    para l de 1 ate 3 passo 1 faca
        para cl de 1 ate 4 passo 1 faca
            escreval("Informe o valor: ")
            leia(A[l,cl])
        fimpara
    fimpara

    para l de 1 ate 3 passo 1 faca
        para cl de 1 ate 4 passo 1 faca
            escreval("Informe o valor: ")
            leia(B[l,cl])
        fimpara
    fimpara

    para l de 1 ate 3 passo 1 faca
        para cl de 1 ate 4 passo 1 faca
            escreval("Informe o valor: ")
            C[l,cl] <- A[l, cl] + B[l, cl]
        fimpara
    fimpara
FimAlgoritmo
```

Fonte: Elaborada pelo autor, 2016.

O teste de mesa a seguir mostra como funciona o laço para soma dos termos (linhas 23 a 27) do algoritmo de soma de matrizes.

| Linha | l | cl | A[l,cl] | B[l,cl] | C[l,cl] |
|-------|---|----|---------|---------|---------|
| 23    | 1 | ?  | ?       | ?       | ?       |
| 24    |   | 1  | ?       | ?       | ?       |
| 25    |   |    | 3       | 4       | 7       |

| Linha | l | cl | A[l,cl] | B[l,cl] | C[l,cl] |
|-------|---|----|---------|---------|---------|
| 24    |   | 2  | ?       | ?       | ?       |
| 25    |   |    | 10      | 2       | 12      |
| 24    |   | 3  | ?       | ?       | ?       |
| 25    |   |    | 0       | 1       | 1       |
| 24    |   | 4  | ?       | ?       | ?       |
| 25    |   |    | 2       | 2       | 4       |
| 23    | 2 | ?  | ?       | ?       | ?       |
| 24    |   | 1  | ?       | ?       | ?       |
| 25    |   |    | 7       | 8       | 15      |
| 24    |   | 2  | ?       | ?       | ?       |
| 25    |   |    | 11      | 6       | 18      |
| 24    |   | 3  | ?       | ?       | ?       |
| 25    |   |    | 10      | 15      | 25      |
| 24    |   | 4  | ?       | ?       | ?       |
| 25    |   |    | 2       | 8       | 10      |
| 23    | 3 | ?  | ?       | ?       | ?       |
| 24    |   | 1  | ?       | ?       | ?       |
| 25    |   |    | 13      | 6       | 19      |
| 24    |   | 2  | ?       | ?       | ?       |
| 25    |   |    | 5       | 3       | 8       |
| 24    |   | 3  | ?       | ?       | ?       |
| 25    |   |    | 9       | 6       | 15      |
| 24    |   | 4  | ?       | ?       | ?       |
| 25    |   |    | 5       | 6       | 11      |

2. Cálculo do determinante de uma matriz quadrada de ordem dois: na matemática, o determinante de uma matriz é uma função que associa a determinada matriz um número que a representa, ou seja, transforma uma matriz em um número real. Para que seja possível calcular o deter-

minante de uma matriz, é necessário que ela seja quadrada, ou seja, possua o mesmo número de linhas e colunas.

Quando a matriz possuir apenas um elemento, o que representa apenas uma linha e uma coluna (matriz de ordem um), o determinante desta matriz é seu próprio elemento. A figura 9, a seguir, mostra o exemplo da Matriz A de ordem um.

Figura 9 - Matriz de Ordem 1.

| Matriz: A (Ordem 1) |    |
|---------------------|----|
|                     | 1  |
| 1                   | 15 |

Fonte: Elaborada pelo autor, 2016.

O cálculo do determinante da matriz A é, então, seu próprio elemento, representado por:  $\det A = |15| = 15$ .

Já nas matrizes com duas linhas e duas colunas (ordem dois), o cálculo do determinante é feito pela diferença do produto dos elementos da diagonal principal com o produto dos elementos da diagonal secundária. A Figura 10, na sequência, mostra o exemplo da Matriz B de ordem 2.

Figura 10 - Armazenamento de dados em um vetor no VisuAlg.

| Matriz: B (Ordem 2) |    |   |
|---------------------|----|---|
|                     | 1  | 2 |
| 1                   | 10 | 3 |
| 2                   | 2  | 8 |

Fonte: Elaborada pelo autor, 2016.

O cálculo do determinante da matriz B é feito de seguinte forma:

$$\det B = (10 \times 8) - (3 \times 2)$$

$$\det B = 80 - 6$$

$$\det B = 74$$

O algoritmo da figura 11 a seguir implementa o cálculo do determinante de matrizes de ordem dois.

Figura 11 - Algoritmo “determinante” na ferramenta VisuAlg.

```

Algoritmo "determinante"
Var
  l, c: inteiro
  A: Vetor[1..2, 1..2] de inteiro
  detB: real
Inicio
  para l de 1 ate 2 passo 1 faca
    para c de 1 ate 2 passo 1 faca
      escreva("Informe o valor: ")
      leia(B[l,c])
    fimpara
  fimpara

  detB <- (B[1,1]*B[2,2]) - (B[1,2]* B[2,1])

  Escreva ("O determinante de B é: ", detB)
FimAlgoritmo

```

Fonte: Elaborada pelo autor, 2016.

O teste de mesa a seguir mostra como é feito o calculo do determinante passo a passo.

| Linha do Algoritmo | l | c | B[l,c] | detB |
|--------------------|---|---|--------|------|
| 8                  | 1 | ? | ?      | ?    |
| 9                  |   | 1 | ?      | ?    |
| 10                 |   |   | (5)    | ?    |
| 9                  |   | 2 | ?      | ?    |
| 10                 |   |   | (6)    | ?    |
| 8                  | 2 | ? | ?      | ?    |
| 9                  |   | 1 | ?      | ?    |
| 10                 |   |   | (3)    | ?    |
| 9                  |   | 2 | ?      | ?    |

| Linha do Algoritmo | l | c | B[l,c] | detB        |
|--------------------|---|---|--------|-------------|
| 10                 |   |   | (4)    | ?           |
| 14                 |   |   |        | 2 (5*4-6*3) |



### Você sabia

Existem várias outras técnicas utilizadas para cálculo do determinante de matrizes quadradas. Regras de Sarrus ou Chió, Teoremas de Laplace, Jacobi ou Binet são algumas delas. Mas, independentemente da técnica, existem algumas propriedades das matrizes que podem facilitar o cálculo dos determinantes e, em alguns casos, facilitar muito as contas em determinadas situações, são elas: o determinante será zero se a matriz possuir uma linha ou uma coluna nula; uma matriz possui sempre o mesmo determinante de sua matriz transposta; o sinal do determinante é trocado se invertermos as duas linhas ou as duas colunas da matriz; ao multiplicarmos um valor n qualquer pelos elementos de uma linha ou de uma coluna da matriz, seu determinante também será multiplicado pelo mesmo valor n; o determinante é nulo se uma matriz possuir duas linhas ou colunas iguais ou múltiplas; o determinante não será alterado se somarmos uma linha ou coluna à outra que foi multiplicada por um número; o determinante do produto de duas matrizes é igual ao produto de seus determinantes.



3. Cálculo da soma das linhas e de todos os elementos de uma matriz: Considerando a matriz D, de 3 x 4 elementos, ilustrada na figura 12.

Figura 12 - Matriz “D”

| Matriz: D |    |   |    |    |
|-----------|----|---|----|----|
|           | 1  | 2 | 3  | 4  |
| 1         | 10 | 9 | 11 | 13 |
| 2         | 5  | 2 | 7  | 5  |
| 3         | 8  | 4 | 9  | 8  |

Fonte: Elaborada pelo autor, 2016.

O algoritmo da figura 13 a seguir implementa a soma de cada linha e de todos os seus elementos.

Figura 13 - Algoritmo “somalinhasetotal” na ferramenta VisuAlg.

Algoritmo “somalinhasetorial”

Var

D: Vetor[1..3, 1..4] de inteiro

l, c: inteiro

somalinha, somatotal: real

Inicio

somalinha <- 0

somatotal <- 0

para l de 1 ate 3 passo 1 faca

para c de 1 ate 4 passo 1 faca

escreval(“Informe o valor: ”)

leia(D[l,c])

fimpara

fimpara

para l de 1 ate 3 passo 1 faca

para c de 1 ate 4 passo 1 faca

somalinha <- somalinha + D[l,c]

somatotal <- somatotal + D[l,c]

fimpara

Escreva (“A soma da linha ”, l, “ é:”, somalinha)

somalinha <- 0

fimpara

Escreva (“A soma total é:”, somatotal)

FimAlgoritmo

Fonte: Elaborada pelo autor, 2016.

O teste de mesa a seguir mostra como funciona o laço para cálculo da soma das linhas e dos elementos (linhas 18 a 26) . Os valores entre chaves representam a saída do algoritmo.

| Linha | l | c | D[l,c] | somalinha | somatotal |
|-------|---|---|--------|-----------|-----------|
| 8     | ? | ? | ?      | ?         | 0         |
| 9     | ? | ? | ?      | 0         | 0         |
| 18    | 1 | ? | ?      | 0         | 0         |
| 19    |   | 1 | 10     | 0         | 0         |
| 20    |   |   | 10     | 10        | 0         |

| Linha | l | c | D[l,c] | somalinha | somatotal |
|-------|---|---|--------|-----------|-----------|
| 21    |   |   | 10     | 10        | 10        |
| 19    |   | 2 | 9      | 10        | 10        |
| 20    |   |   | 9      | 19        | 10        |
| 21    |   |   | 9      | 19        | 19        |
| 19    |   | 3 | 11     | 19        | 19        |
| 20    |   |   | 11     | 30        | 19        |
| 21    |   |   | 11     | 30        | 30        |
| 19    |   | 4 | 13     | 30        | 30        |
| 20    |   |   | 13     | 43        | 30        |
| 21    |   |   | 13     | 43        | 43        |
| 23    |   |   |        | {43}      |           |
| 24    |   |   |        | 0         |           |
| 18    | 2 | ? | ?      | 0         | 43        |
| 19    |   | 1 | 5      | 0         | 43        |
| 20    |   |   | 5      | 5         | 43        |
| 21    |   |   | 5      | 5         | 48        |
| 19    |   | 2 | 2      | 5         | 48        |
| 20    |   |   | 2      | 7         | 48        |
| 21    |   |   | 2      | 7         | 50        |
| 19    |   | 3 | 7      | 7         | 50        |
| 20    |   |   | 7      | 14        | 50        |
| 21    |   |   | 7      | 14        | 57        |
| 19    |   | 4 | 5      | 14        | 57        |
| 20    |   |   | 5      | 19        | 57        |
| 21    |   |   | 5      | 19        | 62        |
| 23    |   |   |        | {19}      |           |
| 24    |   |   |        | 0         |           |
| 18    | 3 | ? | ?      | 0         | 62        |
| 19    |   | 1 | 8      | 0         | 62        |
| 20    |   |   | 8      | 8         | 62        |
| 21    |   |   | 8      | 8         | 70        |
| 19    |   | 2 | 4      | 8         | 70        |

| Linha | l | c | D[l,c] | somalinha | somatotal |
|-------|---|---|--------|-----------|-----------|
| 20    |   |   | 4      | 12        | 70        |
| 21    |   |   | 4      | 12        | 74        |
| 19    |   | 3 | 9      | 12        | 74        |
| 20    |   |   | 9      | 21        | 74        |
| 21    |   |   | 9      | 21        | 83        |
| 19    |   | 4 | 8      | 21        | 83        |
| 20    |   |   | 8      | 29        | 83        |
| 21    |   |   | 8      | 29        | 83        |
| 23    |   |   |        | {29}      |           |
| 24    |   |   |        | 0         |           |
| 26    |   |   |        |           | {83}      |

4. Multiplicação entre matrizes: a multiplicação entre duas matrizes P e Q só é possível quando o número de colunas da matriz P for igual ao número de linhas da matriz Q. Considerando, por exemplo, a matriz P, de 3 linhas e 4 colunas, e a matriz Q, de 4 linhas e 5 colunas, o resultado de sua multiplicação será, necessariamente, a matriz R de 3 linhas e 5 colunas. A figura 14 ilustra as matrizes P e Q.

Figura 14 - Matrizes P e Q.

| Matriz: P |   |   |   |   |
|-----------|---|---|---|---|
|           | 1 | 2 | 3 | 4 |
| 1         | 2 | 3 | 1 | 5 |
| 2         | 3 | 5 | 5 | 3 |
| 3         | 6 | 2 | 8 | 2 |

| Matriz: Q |   |   |   |   |   |
|-----------|---|---|---|---|---|
|           | 1 | 2 | 3 | 4 | 5 |
| 1         | 2 | 5 | 4 | 5 | 2 |



|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 2 | 3 | 5 | 2 | 4 | 6 |
| 3 | 2 | 4 | 1 | 8 | 3 |
| 4 | 1 | 2 | 3 | 1 | 5 |

Fonte: Elaborada pelo autor, 2016.

Os elementos da matriz R resultante da multiplicação entre P e Q são calculados somando a multiplicação de todos os elementos da primeira linha da matriz P pelos elementos da primeira coluna da matriz Q. O exemplo a seguir mostra como isso ocorre:

$$R[1,1] = (P[1,1]*Q[1,1]) + (P[1,2]*Q[2,1]) + (P[1,3]*Q[3,1]) + (P[1,4]*Q[4,1])$$

$$R[1,1] = (2*2) + (3*3) + (1*2) + (5*1)$$

$$R[1,1] = 4+9+2+5$$

$$R[1,1] = 20$$

A figura 15 mostra a matriz R resultante da multiplicação entre as matrizes P e Q.

Figura 15 - Matriz R = P x Q.

| Matriz: R |    |   |   |   |   |
|-----------|----|---|---|---|---|
|           | 1  | 2 | 3 | 4 | 5 |
| 1         | 20 |   |   |   |   |
| 2         |    |   |   |   |   |
| 3         |    |   |   |   |   |

Fonte: Elaborada pelo autor, 2016.

O algoritmo da figura 16 a seguir implementa esta multiplicação.

Figura 16 - Realizando a soma de dois vetores em terceiro vetor.

**Algoritmo** "multiplicacaodematrizes"**Var**P: **Vetor**[1..3, 1..4] de **inteiro**Q: **Vetor**[1..3, 1..5] de **inteiro**R: **Vetor**[1..3, 1..5] de **inteiro**l, c, i: **inteiro**acumula: **real****Inicio**

para l de 1 ate 3 passo 1 faca

para c de 1 ate 4 passo 1 faca

escreval("Informe o valor: ")

leia(P[l,c])

fimpara

fimpara

para l de 1 ate 4 passo 1 faca

para c de 1 ate 5 passo 1 faca

escreval("Informe o valor: ")

leia(Q[l,c])

fimpara

fimpara

para l de 1 ate 3 passo 1 faca

para c de 1 ate 5 passo 1 faca

acumula &lt;- 0

para i de 1 ate 4 passo 1 faca

acumula &lt;- acumula + (P[l,i

fimpara

R[l,c] = acumula;

fimpara

fimpara

**FimAlgoritmo**

Fonte: Elaborada pelo autor, 2015.

## Resumindo

Vetor multidimensional, ou simplesmente matriz, é um assunto muito importante na área de programação. Trata-se de um tipo de dado muito utilizado, que oferece aos programadores diversos recursos que permitem resolver uma série de problemas computacionais. Este capítulo abordou seu conceito,

utilização na área de programação, alguns tipos de matrizes e apresentou algoritmos utilizados nas suas principais operações. Entretanto, é importante que você saiba que existem diversas outras aplicações e operações possíveis para as matrizes, o que possibilita uma gama de possibilidades e mostra a necessidade de estudo aprofundado neste conteúdo para que programador possa ter uma formação mais completa.

Neste capítulo você aprendeu sobre a utilização de vetores multidimensionais, entendendo como as matrizes são utilizadas para armazenar, classificar e pesquisar listas e tabelas de valores. Aprendemos ainda sobre vetores bidimensionais e tridimensionais e sua utilização na área de programação. A partir de agora você é capaz de implementar algoritmos em matrizes.

# 9

## Registros

NESTE CAPÍTULO, VAMOS aprender sobre os registros, que são um conjunto de dados logicamente relacionados, de tipos diferentes. Esta estrutura é muito utilizada na área de programação, pois proporciona maior flexibilidade no código-fonte produzido. Sua principal característica é permitir a gravação de dados de forma permanente em memória secundária, ou seja, os dados não são perdidos quando o computador é desligado.

Durante este módulo, compreenderemos e aplicaremos o uso de registros para armazenar, classificar e pesquisar listas e tabelas de valores. Para isso, veremos, por meio de exemplos, o que são os registros e como eles se dividem em campos. Vamos entender, também, como é a sua aplicação na área de programação de computadores, especialmente vinculando seu uso junto com vetores que manipulam um conjunto de registros.

Objetivo de Aprendizagem:

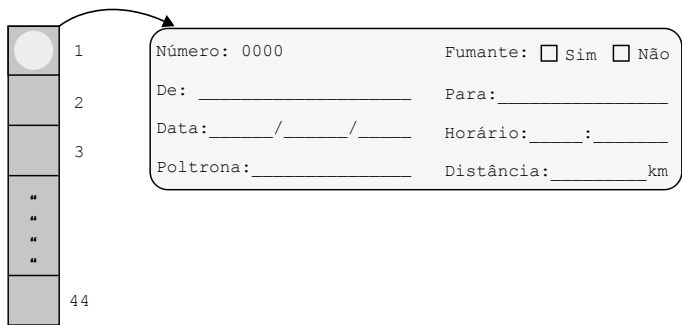
- × Compreender o que são os registros;
- × Entender como aplicá-los na programação de computadores;
- × Aprender a utilizar registros em conjunto com vetores para armazenar e pesquisar listas e tabelas de valores.

9.1 O que são registros?

Registros, ou Variáveis Compostas Heterogêneas, são conjuntos de dados logicamente relacionados, mas de tipos diferentes (heterogêneos). Eles visam facilitar o agrupamento de variáveis que não são do mesmo tipo, mas que guardam uma estreita relação lógica. (XAVIER, 2007)

Para entendermos melhor o que isso significa, vamos conferir o exemplo de Forbellone (2005). Para ficar mais claro o conceito de registros, o autor relaciona um vetor de informações com passagens de ônibus. Cada poltrona (ou passagem) é uma posição do vetor e representa um registro que, por sua vez, agrupa um conjunto de informações (Número, Data, Horário, Poltrona, Distância, etc.) de diferentes tipos, mas relacionadas. A figura 1, a seguir, ilustra este exemplo.

Figura 1 - Representação de um vetor na memória.



Fonte: Forbellone, 2005.

Cada posição do vetor, representado na figura 1, contém um registro, que corresponde a conjuntos de posições de memória, e devem ser chamados por um mesmo nome (identificador). No caso do vetor apresentado na imagem acima,

ele possui 44 posições, ou seja, podemos entender que os dados de 44 passagens serão armazenados para que seja possível sua manipulação. É importante ressaltar que esses registros não possuem relação direta, pois cada um representa uma única passagem do mundo real, vendida para determinada pessoa/passageiro

O registro torna-se então um caso mais geral de variável composta, na qual os elementos do conjunto não precisam ser, necessariamente, homogêneos (de mesmo tipo). (FARRER et al, 1996).

Cada informação do registro é chamada de campo, e cada campo possui, ou não, um tipo diferente. O campo “Poltrona”, por exemplo, pode ser do tipo caractere, enquanto o campo “Distância” tende a ser numérico.

Para compreendermos melhor este conceito, vamos analisar outro exemplo, considerando agora o contexto de um sistema de uso escolar. Imagine que precisásemos armazenar no sistema os dados de diversos alunos matriculados em uma escola, para que seja possível sua gestão posterior.

Se utilizássemos apenas vetores, e considerando que um vetor precisa armazenar dados de somente um tipo específico (numérico ou caractere, por exemplo), seriam necessários diversos vetores para resolver o problema. Em outras palavras, teríamos que criar, por exemplo, um vetor para armazenar os nomes (do tipo caractere), outro vetor para armazenar a idade (do tipo numérico) e outro vetor para armazenar informações sobre deficiências (do tipo booleano). Isso torna a gestão do sistema inviável. A solução, neste caso, é utilizar um registro contendo todos os campos que descrevem determinado aluno, ou seja, a estrutura do registro conterá um campo para armazenar o nome, a idade, a existência de deficiências e qualquer outra informação que seja importante.

A figura 2, a seguir, apresenta um modelo de estrutura para o registro de alunos, que poderia ser utilizada neste sistema.

Figura 2 - Dados inseridos em um vetor.

|                             |                         |
|-----------------------------|-------------------------|
| Matrícula: 00000            | Deficiente: __sim __não |
| Nome: _____                 | Idade: 00               |
| Endereço: _____             | Curso: _____            |
| Telefone: (__) _____ - ____ | Turma                   |
| Sexo: _____                 |                         |

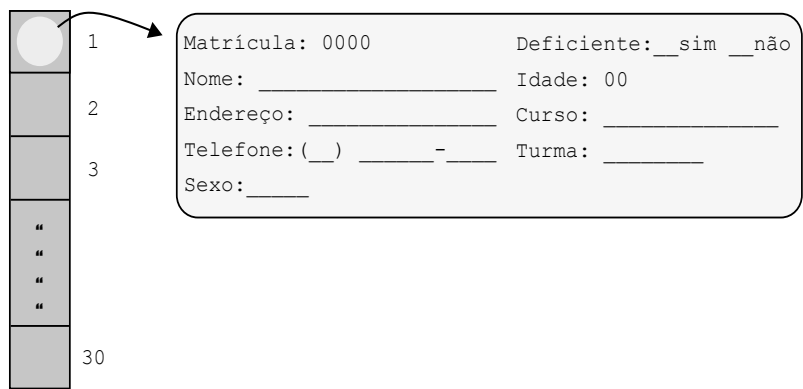
Fonte: Elaborada pelo autor, 2016.

Veja que neste registro existem dados de diversos tipos: os campos “Matrícula” e “Idade” armazenam valores numéricos, os campos “Nome”, “Endereço”, “Telefone”, “Curso” e “Turma” armazenam caracteres e o campo “Deficiente” armazena um valor do tipo booleano. Veja como a estratégia de recursos facilita e flexibiliza muito o trabalho de programação, permitindo criar um conjunto de dados relacionados em uma mesma estrutura.

Entretanto, apenas o registro não consegue resolver o problema de armazenarmos os dados de diversos alunos de uma escola, pois ele se refere aos dados de apenas um aluno. Qual a solução neste caso? Uma das possibilidades é utilizar vetores do registro, isto é, declarar um vetor do tipo do registro que especificamos. No caso do nosso exemplo, criaríamos um vetor do tipo “Aluno”, indicando quantos estudantes seriam armazenados neste vetor (em outras palavras, quantas posições o vetor teria).

A figura 3 abaixo apresenta de forma gráfica o relacionamento do vetor com o registro “Aluno” criado anteriormente.

Figura 3 - Apresentação gráfica do vetor do tipo “Aluno”.



Fonte: Elaborada pelo autor, 2016.

Note que o vetor possui 30 posições, todas elas do tipo “Aluno”. Se analisarmos cada uma dessas posições individualmente, veremos que dentro dela existe uma estrutura com diversos campos que caracterizam os alunos da escola. Também podemos entender esta imagem como sendo um conjunto de registros de estudantes armazenado em um vetor unidimensional.

Ainda não ficou claro? Vejamos outro exemplo. Vamos criar agora uma estrutura para armazenar os dados de diversos carros em um sistema de estacionamento. Considerando que todos os carros possuem basicamente as mesmas características ou informações, tais como cor, marca, modelo e ano de fabricação, iremos criar um registro responsável por agrupar esses dados em uma mesma estrutura. Assim, o sistema do estacionamento poderia, posteriormente, criar um vetor desses registros e armazenar todas as informações dos carros que lá estiverem estacionados em determinado momento.

Analisar a figura 4 a seguir, que apresenta a estrutura do registro “Carro”.

Figura 4 - Registro de carro.

|                                |                              |
|--------------------------------|------------------------------|
| Renavam: 00000                 | Possui avarias: __sim __não  |
| Marca: _____                   | Placa: _____ - _____         |
| Modelo: _____                  |                              |
| Ano de fabricação: 0000        |                              |
| Horário de entrada: ____: ____ | Horário de saída: ____: ____ |

Fonte: Elaborada pelo autor, 2016.

Note que, com apenas esta estrutura do registro, é possível gerenciar todos os carros que entram e saem do estacionamento, inserindo informações particulares e específicas de cada um deles.



## Saiba mais

A manipulação de registros varia nas diversas linguagens de programação disponíveis no mercado atualmente. Para conhecer um pouco mais sobre essas características particulares, acesse os links a seguir.

Java:

[http://orivaldo.net/web/disciplinas/AEDI/apresentacoes/strings\\_registros\\_vetores.pdf](http://orivaldo.net/web/disciplinas/AEDI/apresentacoes/strings_registros_vetores.pdf)

C++:

<https://docente.ifrn.edu.br/brunogurgel/disciplinas/2012/fprog/aulas/cpp/aula10registros.pdf>



Python:

<http://www.dc.uba.ar/materias/int-com/2011/cuat1/Descargas/RegistrosP.pdf>

C#:

<http://pt.slideshare.net/denisfg/explicando-estruturasregistros-no-c>



## 9.2 Aplicações de registros em programação de computadores

Existem diversas estruturas que podem ser utilizadas na programação de computadores para realizar a manipulação de registros, tais como vetores (unidimensionais e multidimensionais) e arquivos. Neste tópico, iremos focar na implementação de registros em conjunto com vetores, pois o uso de arquivos envolve diversas questões que estão fora do nosso escopo de estudo neste momento. Em outras palavras, primeiramente, precisamos aprender a trabalhar com vetores de registros para, posteriormente, manipularmos arquivos de registros.



### Você sabia

O armazenamento de registros também pode acontecer com o apoio de arquivos, cuja estrutura de acesso pode ser sequencial ou direta. Acesse os links abaixo e obtenha maiores informações de como escrever algoritmos que utilizam arquivos para manipular registros.

Capítulos 4 e 5:

[http://gerson.luqueta.com.br/index\\_arquivos/Algoritmos.pdf](http://gerson.luqueta.com.br/index_arquivos/Algoritmos.pdf)

Capítulo 4:

<http://www.cesarbt.xpg.com.br/arq/algoritmos.pdf>



Entretanto, antes de implementarmos um código que envolva armazenamento de registros em vetores, precisamos saber como deve ser feita a declaração da estrutura do registro no código.

A sintaxe básica para declaração da estrutura de registros é a seguinte:

```
tipo <ident_registro> = registro

    <tipo>: <ident_campos>;

fim registro;
```

Em que,

**tipo** – palavra chave (reservada) para declaração de identificadores.

<ident\_registro> – nome da variável que identifica o registro.

**registro** – palavra chave (reservada) que indica um registro.

<tipo> – tipo da variável (caractere, números, etc).

<ident\_campos> – nome da(s) variável(is) de determinado(s) tipo(s) que identificam os campos do registro.

**fim registro** – palavras chaves (reservadas) que indicam o final da declaração do registro.

Para facilitar o entendimento, vamos fazer a declaração da estrutura de alguns registros na prática. Para isso, iremos utilizar os três registros mostrados anteriormente nas figuras 1 (Passagem), 2 (Aluno) e 4 (Carro). Sendo assim, a declaração do registro de “passagem” seria da seguinte maneira:

```
tipo passagem = registro

    inteiro: Número, Poltrona, Distância;

    caractere: De, Para, Data, Horário;

    booleano: Fumante;

fim registro;
```

Veja que estamos criando um tipo de dados chamado “passagem”, assim como temos os tipos inteiro, caractere e booleano. Por isso, dissemos anteriormente que o registro é uma estrutura que ajuda muito a flexibilizar a

estrutura dos programas de computador, pois permite a criação de um conjunto de dados com diversos tipos (ao contrário do que ocorre com os vetores, que possibilitam armazenar em suas posições apenas um tipo de dado). Em seguida, fizemos a declaração dos campos deste registro, de acordo com seu tipo de dado. Pense nas possibilidades que a estrutura de registros nos oferece: poderíamos até mesmo criar uma contendo um campo que fosse do tipo de outro registro!

Dando sequência aos nossos exemplos, iremos declarar o registro “Aluno”:

```
tipo aluno = registro  
  
    inteiro: Matrícula, Idade;  
  
    caractere: Nome, Endereço, Telefone, Sexo,  
Curso, Turma;  
  
    booleano: Deficiente;  
  
fim registro;
```

Note que seguimos a mesma estrutura do exemplo anterior, primeiro indicamos o nome do novo tipo (“aluno”) e, em seguida, especificamos quais campos o compõe, separados de acordo com seu tipo de dados.

Para finalizar esta exemplificação de como utilizar a sintaxe básica de declaração de registros, faremos a declaração do registro “carro”, considerando os campos apresentados anteriormente na figura 4.

```
tipo carro = registro  
  
    inteiro: RENAVAL, AnoDeFabricação;  
  
    caractere: Marca, Modelo, Placa, HorárioEn-  
trada, HorárioSaída;  
  
    booleano: PossuiAvarias;  
  
fim registro;
```

Você percebeu que este exemplo não apresentou nenhuma novidade em relação aos outros, pois utiliza a palavra reservada “tipo” para definir o novo

tipo de dados (chamado, neste caso, de “carro”) e especifica os campos do registro de acordo com seus tipos de dados.

Antes de prosseguir neste capítulo, tenha certeza que entendeu completamente os motivos de se utilizar registros em programas de computador, além de ter conseguido escrever a estrutura deles utilizando a sintaxe padrão apresentada.

### 9.2.1 Integrando registros e vetores

Conforme dissemos no início do tópico 9.2, os vetores são estruturas que podem ser utilizadas para manipular registros dentro de um programa de computador. Por terem uma declaração simplificada e serem relativamente fáceis de manipular, é importante que você estude e compreenda como ocorre na prática essa integração entre os vetores e registros.

Para isso, vamos trabalhar inicialmente com o registro “passagem”, visto no início do capítulo, na qual iremos definir um vetor de 44 posições. Em cada uma dessas posições, será armazenado um registro do tipo “passagem”. Sendo assim, precisamos declarar este vetor, considerando que o registro já tenha sido especificado. Veja a seguir a sintaxe a ser utilizada:

```
tipo    nome_do_tipo:    vetor[tamanho_do_vetor]    de
registro;
```

Neste caso, o programa ficaria da seguinte maneira:

```
tipo vetorPassagens: vetor[1..44] de Passagem;
```

Veja que estamos declarando um novo tipo de dados chamado “vetor-Passagens”, o qual é composto por um vetor de 44 posições do tipo “passagem” (que é um registro contendo sua própria estrutura). Sendo assim, para utilizar este novo tipo, será necessário declarar uma variável, cujo exemplo é mostrado a seguir.

```
vetorPassagens: viagem;
```

---

---

**Tipo de dados:** pode ser entendido como um conjunto de valores e operações que determinada variável pode executar dentro do contexto de um programa de computador. É importante salientar que podem ocorrer variações nos tipos de dados em função do sistema operacional e da linguagem de programação utilizados.

---

---

Para facilitar o entendimento, vamos agrupar esses códigos em um mesmo algoritmo, criando uma visão mais conjunta das declarações.

```
1  Algoritmo Controle_de_Passagens;  
2  Variáveis  
3      tipo passagem = registro  
4      inteiro: Número, Poltrona, Distância;  
5      caractere: De, Para, Data, Horário;  
6      booleano: Fumante;  
7      fim registro;  
8  
9      tipo vetorPassagens: vetor[1..44] de Passagem;  
10     vetorPassagens: viagem;  
11  Início  
12     //Instruções do algoritmo  
13  FimAlgoritmo
```

Analisando o algoritmo apresentado, note que a definição da estrutura do registro “passagem” foi feita entre as linhas 3 e 7, onde os campos que fazem parte deste registro foram especificados. Na linha 9, declaramos um novo tipo chamado “vetorPassagens”, o qual é composto de um vetor com 44 posições do tipo “passagem” (que é o registro que declaramos entre as linhas 3 e 7). Para usar este novo tipo, declaramos uma variável do tipo “vetorPassagens” chamada “viagem”. Dentro do início e fim do algoritmo, esta variável seria utilizada para manipular os dados de todas as passagens de determinada viagem que o programa de computador deveria gerenciar.

### 9.2.2 Manipulando registros dentro de vetores

Assim como o acesso aos dados de um vetor é feito elemento a elemento, seja para atribuímos um valor ou para consultarmos um valor previamente armazenado, o mesmo acontece nos casos envolvendo registros, em que o acesso deve ser campo a campo. Em outras palavras, como a estrutura de um registro pode possuir diversos campos, sempre que quisermos acessar um campo específico precisamos informar ao algoritmo qual é este campo.

Outra regra a ser seguida para realizar este acesso aos campos de registros é com relação ao uso do operador “.” (ponto final). Ele serve para separarmos a variável que contém o registro do campo que pretendemos acessar.

Por exemplo, considere que temos uma variável chamada “alunos” do tipo registro “Aluno”, declarado anteriormente neste capítulo. Se fosse necessário ler o nome de um aluno, utilizaríamos a seguinte instrução:

```
leia (alunos.nome) ;
```

A mesma regra se aplica para o caso de necessitarmos obter o valor atualmente armazenado em um campo do registro. Para exemplificar, a instrução a seguir faz a impressão do valor atual do campo “nome” do registro “alunos”:

```
escreva (alunos.nome) ;
```

Entretanto, como os vetores possuem posições, precisamos considerar esta característica no momento da sua utilização em conjunto com os registros. Na prática, é necessário apenas inserir na instrução o número da posição que desejamos acessar para alterar o campo do registro. Considerando que a variável “alunos” é um vetor com 10 posições do registro “Aluno”, a escrita e a leitura do campo “nome” armazenado na terceira posição do vetor ficariam da seguinte forma:

```
leia (alunos[3].nome) ;
```

```
escreva (alunos[3].nome) ;
```

Neste caso, estamos especificando qual a posição que desejamos trabalhar, porém como se trata de vetor, é altamente recomendável a utilização de um laço para fazer a leitura dos dados, pois pode ser que o número de posições a serem preenchidas seja grande.

Vamos ver como isso funciona na prática? Iremos implementar as instruções para ler todos os dados das 44 passagens definidas pelo algoritmo “*Controle\_de\_Passagens*”. Considere que as instruções relacionadas a seguir fossem escritas entre os blocos de início e fim do algoritmo, a partir da linha 12. Além disso, considere que foi declarada uma variável do tipo “inteiro” chamada “i” (para fazer as interações no laço).

```
1   para i de 1 até 44 faça
2       escreva ("Informe o número da passagem");
3       leia (viagem[i].Número);
4       escreva ("Informe o número da poltrona");
5       leia (viagem[i].Poltrona);
6       escreva ("Informe o ponto de origem");
7       leia (viagem[i].De);
8       escreva ("Informe o ponto de destino");
9       leia (viagem[i].Para);
10      escreva ("Informe a data");
11      leia (viagem[i].Data);
12      escreva ("Informe o horário");
13      leia (viagem[i].Horário);
14      escreva ("Informe a distância");
15      leia (viagem[i].Distância);
16      escreva ("Informe se é fumante");
17      leia (viagem[i].Fumante);
18  fim para
```

Veja que no início das instruções definimos um laço que irá rodar as instruções 44 vezes, permitindo assim o preenchimento de todas as posições do vetor “viagem”. Utilizando o comando “leia”, fazemos a leitura de cada um dos campos do registro “passagem”, armazenando os dados de acordo com cada posição do vetor (controlado pela variável “i”). Na primeira vez que o laço estiver sendo executado, estaremos preenchendo os campos da posição 1 do vetor; em seguida, o preenchimento será para a posição 2, e assim por diante, até terminar na posição 44.

Caso fosse necessário imprimir os valores armazenados nos registros do vetor, poderíamos utilizar a mesma estratégia de especificar a posição do vetor (com o uso da variável “i” incrementada automaticamente pelo laço) em conjunto com o operador “.”, indicando qual campo do registro deve ser impresso. As instruções para impressão dos dados seriam similares ao conjunto que é apresentado a seguir.

```

1   para i de 1 até 44 faça
2       escreva ("Dados da passagem");
3       escreva (viagem[i].Número);
4       escreva (viagem[i].Poltrona);
5       escreva (viagem[i].De);
6       escreva (viagem[i].Para);
7       escreva (viagem[i].Data);
8       escreva (viagem[i].Horário);
9       escreva (viagem[i].Distância);
10      escreva (viagem[i].Fumante);
11  fim para

```

Veja que nas instruções apresentadas também utilizamos um laço “para” com o objetivo de percorrer as 44 posições do vetor “viagem”. Para cada interação do laço, escrevemos os dados atuais da passagem, de acordo com os campos do registro.

### 9.2.3 Realizando pesquisas e exclusões nos registros

Uma das mais importantes operações que um programa de computador pode executar é a pesquisa de dados. De nada adianta nosso algoritmo ser muito eficaz no armazenamento dos dados se ele não permite que o usuário faça pesquisas nos dados já guardados. No caso de vetores de registros, o processo de pesquisa é relativamente simples, como veremos a seguir.

Iremos utilizar como base tanto o registro “passagem” quanto o algoritmo “*Controle\_de\_Passagens*”, especificados anteriormente. Considere que todas as quarenta e quatro posições do vetor “viagem” já estão preenchidas



com dados de determinada viagem. Nosso objetivo é fazer uma pesquisa pelo número da passagem para alterar a poltrona do cliente, ou seja, o usuário irá informar qual o número da passagem deseja pesquisar e qual o novo número da poltrona. O programa fará essa busca e, caso ele obtenha sucesso, a alteração da poltrona será realizada.

O algoritmo apresentado a seguir realiza tanto a pesquisa no vetor quanto a alteração da poltrona, conforme o novo valor informado pelo usuário.

```
1  Algoritmo Pesquisa_Passagem;
2  Variáveis
3  tipo passagem = registro
4  inteiro: Número, Poltrona, Distância;
5  caractere: De, Para, Data, Horário;
6  booleano: Fumante;
7  fim registro;
8
9  tipo vetorPassagens:
    vetor[1..44] de Passagem;
10 vetorPassagens: viagem;
11 inteiro: i, NúmeroPassagem, NovaPoltrona;
12 Início
13 leia (NúmeroPassagem);
14 leia (NovaPoltrona);
15 para i de 1 até 44 faça
16 se (NúmeroPassagem = viagem[i].Número)
17 então viagem[i].Poltrona ← NovaPoltrona;
18 escreva ("A poltrona foi alterada.");
19 interrompa;
20 fim se
21 fim para
22 FimAlgoritmo
```

Analisando o algoritmo apresentado, notamos que foram lidos dois valores: na linha 13 é lido o número da passagem a ser pesquisado e na linha

14 o novo número da poltrona do passageiro. A ideia é percorrer todo o vetor e pesquisar alguma passagem com o número igual ao informado pelo usuário. Dentro do laço que realiza as 44 iterações, temos na linha 16 uma condição que verifica se o número informado pelo usuário é igual ao número armazenado no registro da posição “i” do vetor; caso os valores sejam iguais, a nova poltrona informada pelo usuário é armazenada no campo “Poltrona” do registro (linha 17) e o laço é interrompido (linha 19), pois partimos do princípio que somente pode existir uma passagem com determinado número.

Outra situação comum, ao se trabalhar com essas estruturas de armazenamento, é a necessidade de exclusão de determinado registro dentro de um vetor. Para isso, podemos utilizar a instrução “elimine”, juntamente com a indicação de qual posição do vetor deverá ser excluída. Desta forma, se na posição indicada para a exclusão existir um registro, estaremos automaticamente eliminando todos os campos do registro, não sendo necessária a exclusão individual deles.

Utilizando o código de pesquisa pelo número da passagem apresentado neste mesmo subtópico, podemos executar a instrução “elimine” caso uma passagem com este mesmo número seja encontrada dentro do vetor. Para isso, bastaria alterar a linha 17 para:

```
então elimine(viagem[i]);
```

Desta forma, o registro armazenado na posição “i” do vetor, ou seja, aquela posição onde se encontra o número da passagem informado pelo usuário, seria totalmente excluído. Você poderia também alterar a mensagem apresentada na linha 18 para indicar que o registro foi excluído.

## Importante

Antes de alterar ou excluir determinado registro, é muito importante certificar-se de que a posição corrente no vetor está correta. Isso pode ser feito apresentando os dados que nela constam para que o usuário faça uma confirmação da operação, pois não há como desfazer a ação.

## Resumindo

Neste capítulo, estudamos os principais conceitos relacionados aos registros, além de aplicar esses conhecimentos em exemplos práticos envolvendo a utilização de vetores para armazenamento de dados deles. Vale ressaltar que o registro pode ser caracterizado como um conjunto de dados relacionados logicamente e que podem ser de tipos diferentes.

Aprendemos, também, por meio de exemplos práticos, como pode ser realizada uma busca nos registros armazenados em um vetor, a qual pode ser feita por qualquer um dos campos que compõem essa estrutura. Trabalhamos ainda com a exclusão de registros de um vetor, o que pode ser feito com simplicidade em termos de instruções no programa de computador.

Em termos de mercado de trabalho, é fundamental conhecer as principais características e as possibilidades de utilização dos registros em programas de computador, pois são estruturas realmente fáceis de implementar e que tornam o programa muito mais flexível.

# 10

## Modularização: funções e procedimento

NESTE CAPÍTULO VOCÊ IRÁ aprender sobre o conceito de modularização e sobre como utilizar funções e procedimentos para aplicação de recursividade em programação estruturada. Modularizar significa decompor um problema maior em diversos subproblemas, com o objetivo principal de simplificar o entendimento do mesmo, tornar o desenvolvimento de programas mais fácil, a manutenção mais simples e facilitar os testes, dentre outros benefícios.

Neste capítulo veremos que as principais ferramentas de modularização são as funções e os procedimentos, e iremos compreender como elas são utilizadas na estrutura dos programas. Em seguida, serão apresentados exemplos de algoritmos utilizando estes elementos para que você também aprenda sobre recursividade.

Boa aula!

## Objetivo de Aprendizagem:

- × Entender o conceito de modularização, suas funções e aplicação;
- × Conhecer as principais ferramentas de modularização;
- × Aprender sobre recursividade.

### 10.1 Modularização

Algoritmos são utilizados para resolver problemas, cuja complexidade varia em maior ou menor grau. Um problema complexo pode ser simplificado quando dividimos em vários subproblemas (FORBELLONE, 2005, p. 123). Esta simplificação é conhecida como decomposição ou modularização.


Modularização é um fator determinante para a redução da complexidade dos problemas, uma vez que quando decompomos um problema maior em diversos subproblemas, estamos dividindo também a sua complexidade e, conseqüentemente, simplificando sua resolução.

O processo de modularização também é conhecido como Técnica de Refinamentos Sucessivos, justamente por partirmos de um problema maior e o dividirmos em problemas mais simples e específicos. (FORBELLONE, 2005, p. 124).



#### Saiba mais

Refinamentos sucessivos podem ser feitos por meio de duas técnicas: *Top-Down* ou *Bottom-Up*. *Top-Down*, ou de cima para baixo, significa que o problema será decomposto a partir de conceitos mais abrangentes, ou abstratos, até o nível de detalhamento desejado. Já o processo inverso, que consiste em partir de conceitos mais detalhados e agrupá-los em níveis mais abrangentes até o nível de abstração desejado, é chamando de *Bottom-Up*, ou de baixo para cima. O processo mais utilizado, justamente por ser uma forma mais natural de análise dos problemas, é a técnica *Top-Down*.



Cada pedaço decomposto do problema pode ser entendido como um módulo que contém um grupo de comandos e que constitui um trecho do algo-

ritmo. Esse módulo possui ainda uma função bem definida e a proposta é que ele seja o mais independente possível em relação ao restante do programa.

A maneira mais simples e intuitiva de realizar a modularização de problemas é a partir da definição de um módulo principal de controle e de módulos específicos para as funções do algoritmo.

A seguir, veremos as principais vantagens e desvantagens da modularização.

### 10.1.1 Vantagens e desvantagens da modularização dos algoritmos

Existem várias vantagens e, naturalmente, algumas desvantagens na modularização de programas. Algumas destas vantagens são específicas de cada linguagem de programação que se escolhe para a implementação do algoritmo. Farrer et al. (1996, p. 97 a 102) citam as principais vantagens:

- × Partes comuns a vários programas ou que se repetem dentro de um mesmo programa, quando modularizadas, podem ser programadas e testadas de uma só vez, ainda que tenham que ser executadas com variáveis diferentes. Isso permite que correções dos módulos possam ser feitas em separado do programa principal;
- × É possível construir bibliotecas de módulos, ou seja, uma coleção de algoritmos que podem ser usados em diferentes programas sem a necessidade de alteração por outros programadores. Por exemplo, algumas funções pré-definidas das linguagens de programação constituem uma biblioteca de módulos. Desta forma, um módulo independente pode ser utilizado em outros algoritmos que requeiram o mesmo processamento por ele executado;



#### Você sabia

Cada linguagem de programação já traz implementada uma série de funções e procedimentos pré-definidos, que facilitam muito a vida do programador. A linguagem Java, uma das mais utilizadas na atualidade, por exemplo, possui um tutorial online, disponível em <http://docs.oracle.com/javase/tutorial/>, que contém vasto material sobre a linguagem, incluindo as funções nativas. Já a linguagem PHP dispo-

nibiliza conteúdo oficial em <[http://php.net/manual/pt\\_BR/tutorial.php](http://php.net/manual/pt_BR/tutorial.php)>. Lembre-se que, ao estudar uma linguagem de programação, é muito importante que você busque material de referência de qualidade, preferencialmente nos chamados sites oficiais das linguagens.



- × A modularização dos programas permite preservar na sua implementação os refinamentos obtidos durante o desenvolvimento dos algoritmos;
- × Auxilia na economia de memória do computador, uma vez que o módulo é armazenado uma única vez, mesmo que utilizado em diferentes partes do programa.
- × Permite que, em determinado instante da execução do programa, estejam na memória principal apenas os módulos necessários à execução de determinado trecho do programa.
- × A independência do módulo permite uma manutenção mais simples e evita efeitos colaterais em outros pontos do algoritmo;
- × A elaboração do módulo pode ser feita independentemente e em época diferente do restante do algoritmo.

A principal desvantagem da modularização está no fato de que existe um acréscimo de tempo na execução de programas constituídos de módulos, devido ao tratamento adicional de ativação de cada módulo.

Na próxima seção iremos conhecer as duas principais ferramentas de modularização, as funções e os procedimentos.

### 10.1.2 Ferramentas para modularização

As duas ferramentas básicas de modularização são as funções e os procedimentos. A principal diferença entre elas é que as funções retornam um valor, enquanto os procedimentos não. O valor retornado em um procedimento se dá por meio de seus parâmetros. Esta distinção será mais detalhada nos próximos tópicos.

Tanto as funções quanto os procedimentos possuem três objetivos básicos (FARRER et al, 1996, p. 97 a 102):

- × Evitar que certa sequência de comandos necessária em vários locais de um algoritmo tenha que ser escrita repetidamente nestes locais;
- × Dividir e estruturar um algoritmo em partes fechadas e logicamente coerentes;
- × Aumentar a legibilidade do algoritmo.

E o que são exatamente as funções e os procedimentos? São módulos hierarquicamente subordinados a um algoritmo, o qual é comumente chamado de módulo principal. Da mesma forma, ambos podem conter outras funções e procedimentos aninhados. As funções e os procedimentos são criados por meio de declarações em um algoritmo e, para serem executados, necessitam de ativação provocada por um comando de chamada.

A declaração é constituída de um cabeçalho e de um corpo. O cabeçalho, que identifica a função ou procedimento, contém o seu nome e a lista de parâmetros formais. O corpo contém declarações locais e os comandos da função ou procedimento. A ativação de ambas é feita por meio de referência a seu nome e a indicação dos parâmetros atuais (FARRER et al, 1996, p. 97 a 102).

Nos tópicos a seguir veremos como as funções e procedimentos são declarados, executados e finalizados. Para isso, vamos compreender como são escritos nas estruturas dos programas.

## 10.2 Compreendendo as funções na estrutura do programa

As funções são módulos que possuem como característica o retorno de um valor, ou seja, retornam ao algoritmo que as chamou um valor associado ao nome da função.

A sintaxe básica de declaração de funções é a seguinte:

```
funçãoNomeDaFunção (lista-de-parâmetros) //cabeçalho
da função
    //declaração de objetos locais à função
    //comandos da função
retorne (valor) ;
```



**fimfunção**//fim da função

Em que,

**função**: palavra chave que identifica uma função.

NomedaFunção: é um nome dado à uma função.

lista-de-parâmetros: é a lista dos objetos que serão substituídos por outras variáveis, fornecidos quando da chamada da função.

retorne: obrigatório nas funções, este comando determina qual valor será retornado pela função.

valor: valor retornado pela função.

Vamos conferir um exemplo para esclarecer melhor a aplicação.

Exemplo: crie um algoritmo que contenha uma função que implemente a soma de dois números. Estes números serão lidos no programa principal e passados como parâmetros para uma função chamada “soma”. Esta função será responsável por realizar o cálculo e retornar o valor final encontrado, o qual será exibido para o usuário.

A Figura 1 a seguir apresenta este algoritmo na ferramenta “VisuAlg”:

Figura 1 - Apresentação do algoritmo “funcao” na ferramenta VisuAlg.

```
Algoritmo "funcao"
// Disciplina: Lógica de Programação
// Descrição : Exemplo de função
funcao soma (var n1, n2: inteiro): inteiro
inicio
    Retorne n1 + n2
fimfuncao
Var
    numeral, numero2, somanumeros: Inteiro
Inicio
    Limpatela
    Escreval("Informe o número 1: ")
    Leia(numeral)
    Escreval("Informe o número 2: ")
    Leia(numero2)
    somanumeros <- soma(numero1, numero2)
    Escreval("A soma dos números é: ", somanumeros)
fimAlgoritmo
```


Fonte: Elaborada pelo autor, 2016.

No exemplo anterior podemos notar que a função é constituída por uma sequência de comandos que operam sobre um conjunto de objetos que podem ser globais ou locais. As variáveis “numero1”, “numero2” e “soma-numeros” são globais, uma vez que foram declaradas no programa principal (onde é feita a chamada à função “soma”), enquanto as variáveis “n1” e “n2” são locais, pois foram declaradas dentro da função.



## Saiba mais

Objetos globais são entidades que podem ser usadas em módulos internos a outro módulo do algoritmo onde foram declaradas. Neste caso, podemos utilizá-las dentro da função. Já objetos locais são entidades que só podem ser usadas no módulo (função ou procedimento) do algoritmo em que foram declaradas. Estes objetos não possuem qualquer significado fora deste módulo. Isso quer dizer que as variáveis “n1” e “n2” só existem dentro da função “soma”. O programa principal ou alguma outra função não conseguem acessá-las.



Além dos objetos globais e locais, é possível verificar que a comunicação entre o programa principal e a função “soma” acontece por meio de passagem (ou transferência) de parâmetros.

Neste exemplo, o comando “soma(numero1,numero2)” indica que o valor atual das variáveis “numero1” e “numero2” foi passado como parâmetro para a função “soma”. Este tipo de passagem de parâmetro é conhecido como “Passagem por Valor”.

Na próxima seção, vamos utilizar um exemplo parecido com este para explicar como funcionam os procedimentos. Você irá aprender, ainda, outro tipo de passagem de parâmetro, conhecido como “Passagem por Referência”.

## 10.3 Procedimentos na estrutura do programa

Os procedimentos, diferentemente das funções, são módulos que possuem como característica o fato de não retornarem um valor. Quando um procedimento precisa retornar algum valor para quem o executou, ele utiliza

os parâmetros para realizar essa tarefa. Entretanto, na maior parte das vezes, os procedimentos simplesmente não retornam nada.

A sintaxe básica de declaração de procedimentos é a seguinte:

```
procedimentoNomeDoProcedimento (lista-de-parâ-  
metros) //cabeçalho  
    //declaração de objetos locais ao procedimento  
    //comandos do procedimento  
fimprocedimento //fim do procedimento
```

Em que,

**procedimento:** palavra chave que identifica um procedimento.

NomeDoProcedimento: é um nome dado ao procedimento.

lista-de-parâmetros: é a lista dos objetos que serão substituídos por outras variáveis, fornecidos quando da chamada do procedimento.

Veja que há uma diferença importante entre a sintaxe do procedimento e da função declarada na seção anterior. O procedimento não utiliza a função **retorne** em seu corpo. Não há necessidade justamente pelo fato do procedimento não retornar diretamente nenhum valor ao programa principal. Quando isso for necessário, deverá ser feito por meio de um parâmetro, como veremos no exemplo a seguir.

Exemplo: crie um algoritmo que contenha um procedimento que calcule a multiplicação de dois números. Estes números serão lidos no programa principal, passados como parâmetros para o procedimento "multiplica", que irá realizar a multiplicação e retornar o valor armazenado em outra variável, também passada como parâmetro.

A Figura 2 a seguir apresenta este algoritmo na ferramenta "VisuAlg":

Figura 2 - Apresentação do algoritmo "procedimento" na ferramenta VisuAlg.

```
Algoritmo "procedimento"  
// Disciplina: Lógica de Programação  
// Descrição : Exemplo de procedimento  
  
procedimento multiplica(var n1, n2, valorfinal: inteiro)
```

```

inicio
    valorfinal <- n1 * n2
fimprocedimento

Var
    numero1, numero2, resultado: Inteiro

Inicio
    Limpatela
    Escreval("Informe o número 1: ")
    Leia(numero1)
    Escreval("Informe o número 2: ")
    Leia(numero2)

    multiplica(numero1, numero2, resultado)

    Escreval("A multiplicação dos números é: ", resultado)
fimAlgoritmo

```

Fonte: Elaborada pelo autor, 2016.

Neste exemplo, três parâmetros foram passados para o procedimento “multiplica”: “numero1”, “numero2” e “resultado”. Na prática, os parâmetros “numero1”, “numero2” foram passados por valor, uma vez que tiveram apenas o seu valor utilizado dentro do procedimento. Em outras palavras, o valor desses parâmetros não sofreu alteração dentro do procedimento. Já o parâmetro “resultado” foi declarado no escopo do programa principal e teve seu conteúdo alterado dentro do procedimento, indicando que ocorreu uma passagem de parâmetro por referência.

Na próxima seção veremos outros exemplos de algoritmos que utilizam funções e procedimentos em sua estrutura.

## 10.4 Funções e procedimentos

Vamos conhecer agora alguns algoritmos que utilizam funções e procedimentos em suas soluções.

1. O algoritmo abaixo recebe um número como entrada e retorna se ele é par ou ímpar. A primeira implementação utiliza uma função e a segunda um procedimento.

A Figura 3 a seguir apresenta este algoritmo na ferramenta “VisuAlg”:

Figura 3 - Apresentação do algoritmo “par\_impar\_com\_funcao” na ferramenta VisuAlg.

```
Algoritmo "par_impar_com_funcao"  
// Disciplina: Lógica de Programação  
// Descrição : Número par e ímpar usando  
função  
funcao par_impar(numeros: inteiro) : carater  
    var  
        resto, quociente: inteiro  
    inicio  
        quociente <- numero div 2  
        resto <- numero-(quociente*2)  
  
        se (resto=0) entao  
            retorne "PAR"  
        senao  
            retorne "ÍMPAR"  
        fimse  
    fimfuncao  
  
    var  
        numero: inteiro  
        tiponumero: caracter  
  
    inicio  
        Limpatela  
        Escreval("Informe o número: ")  
        Leia(numero1)  
  
        tiponumero <- par_impar(numero)  
  
        Escreval("O número é: ", tiponumero)  
  
    fimAlgoritmo
```

Fonte: Elaborada pelo autor, 2015.

A Figura 4 a seguir apresenta este algoritmo na ferramenta “VisuAlg”:

Figura 4 - Apresentação do algoritmo “par\_impar\_com\_procedimento” na ferramenta VisuAlg.

#### Algoritmo

// Disciplina: Lógica de Programação

// Descrição : Número par e ímpar usando procedimento

procedimento par\_impar(var numero: inteiro; var tiponumero: caracter)

var

resto, quociente: inteiro

inicio

quociente <- numero div 2

resto <- numero-(quociente\*2)

se (resto=0) entao

tiponumero "PAR"

senao

tiponumero "ÍMPAR"

fimse

fimprocedimento

var

numero: inteiro

tiponumero: caracter

inicio

Limpatela

Escreval("Informe o número: ")

Leia(numero)

par\_impar(numero, tiponumero)

Escreval("O número é: ", tiponumero)

fimAlgoritmo

Fonte: Elaborada pelo autor, 2016

2. O algoritmo a seguir recebe dois números inteiros como entrada e retorna a soma dos N números inteiros existentes entre eles. Assim como no exemplo anterior, a primeira implementação utiliza uma função e a segunda um procedimento.

A Figura 5 a seguir apresenta este algoritmo na ferramenta “VisuAlg”:

Figura 5 - Apresentação do algoritmo “soma\_termos\_com\_funcao” na ferramenta VisuAlg.

```
Algoritmo "soma termos _ com_funcao"  
// Disciplina: Lógica de Programação  
// Descrição : Soma dos termos usando função  
funcao soma_termos(n1, n2: inteiro): inteiro  
var  
    soma, i: inteiro  
inicio  
    se (n2>=n1) entao  
        retorne 0  
    fimse  
  
    para i de n1 ate n2 faca  
        soman <- soman +i  
    fimpara  
  
    retorne soman  
fimfuncao  
  
var  
    n1, n2, soma: inteiro  
  
inicio  
    Limpatela  
    Escreval("Informe o número 1: ")  
    Leia(n1)  
    Escreval("Informe o número 2: ")  
    Leia(n2)  
    soma <- soma termos(n1, n2)  
    Escreval("A soma dos termos é: ", soma)  
fimAlgoritmo
```

Fonte: Elaborada pelo autor, 2016.

A Figura 6 a seguir apresenta este algoritmo na ferramenta “VisuAlg”:

Figura 6 - Apresentação do algoritmo “soma\_termos\_com\_procedimento” na ferramenta VisuAlg.

```
Algoritmo "soma termos _ com_procedimento"  
// Disciplina: Lógica de Programação  
// Descrição : Soma dos termos usando procedimento  
  
funcao soma termos(n1, n2: inteiro; var soma: inteiro)  
var  
    i: inteiro  
    . . .
```

```

Algoritmo "soma termos _ com procedimento"
// Disciplina: Lógica de Programação
// Descrição : Soma dos termos usando procedimento

funcao soma termos(n1, n2: inteiro; var soma: inteiro)
var
  i: inteiro
inicio
  se (n2>=n1) entao
    soma <- 0
  senao
    para i de n1 ate n2 faca
      soma <- soma + i
    fimpara
  fimse
fimprocedimento
var
  n1, n2, soma: inteiro

inicio
  Limpatela
  Escreval(nInforme o número 1: ")
  Leia(n1)
  Escreval(nInforme o número 2: ")
  Leia(n2)
  soma termos(n1, n2, soma)
  Escreval(nA soma dos termos é: n, soma)
fimAlgoritmo

```

Fonte: Elaborada pelo autor, 2016.

Agora que você já implementou alguns algoritmos utilizando funções e procedimentos, iremos aprender outro conceito bastante importante, que é a recursividade.

### 10.4.1 Recursividade

Existem alguns tipos de algoritmos que possuem uma propriedade especial, em que uma instância de determinado problema possui uma instância menor do mesmo problema. A esta característica chamamos de recursividade, uma vez que este tipo de problema possui a sua estrutura recursiva.

Em outras palavras, isso quer dizer que um módulo recursivo é uma função ou procedimento capaz de chamar a si mesmo na solução do problema.



A principal vantagem está na redução do tamanho do algoritmo, permitindo que ele seja descrito de forma mais clara.

## Importante

Recursividade pode ser usada tanto em funções quanto em procedimentos, sendo que todo algoritmo recursivo existe outro correspondente, não recursivo, que executa a mesma tarefa. Algoritmos não recursivos são chamados de iterativos.

Vamos entender na prática como essa técnica pode ser útil para a elaboração de nossos algoritmos. Para isso, apresentaremos um problema hipotético que pode ser resolvido com algoritmo recursivo. Primeiro, veremos a solução sem recursão, em seguida com recursividade.

1. O algoritmo a seguir implementa o cálculo do fatorial de um número inteiro e positivo  $N$ . O fatorial é calculado multiplicando o número  $N$  por seus antecessores até o valor 1.  $5!$  (lê-se cinco fatorial), por exemplo, é calculado da seguinte forma:  $5 \times 4 \times 3 \times 2 \times 1 = 120$ .

---

**Fatorial:** o fatorial de um número natural  $n$ , representado por  $n!$ , é o produto de todos os inteiros positivos menores ou iguais a  $n$ . Na matemática, uma das principais aplicações do fatorial está na resolução de problemas de contagem, em que precisamos calcular, por exemplo, o número de possibilidade de ocorrência de determinado evento. Quando um evento, por exemplo, é composto por  $n$  etapas sucessivas e independentes, de tal forma que o número de possibilidades da primeira etapa é  $m$  e as possibilidades da segunda etapa são  $n$ , consideramos então que o número total de possibilidades do evento ocorrer é dado pelo produto  $m \times n$ , e assim sucessivamente.

---

A Figura 7 a seguir apresenta este algoritmo na ferramenta “VisuAlg”:

Figura 7 - Apresentação do algoritmo “fatorial\_nao\_recursivo” na ferramenta VisuAlg.

```

Algoritmo "fatorial _ nao_recursivo"
// Disciplina: Lógica de Programação
// Descrição : fatorial não recursivo de um número

funcao fatorial(n: inteiro): inteiro
var
    fn, inteiro
inicio
    fn <- 1

    para i de n ate 1 passo -1 faca
        fn <- fn*i
    fimpara

    retorne fn
fimfuncao
var
    n, fatN: inteiro

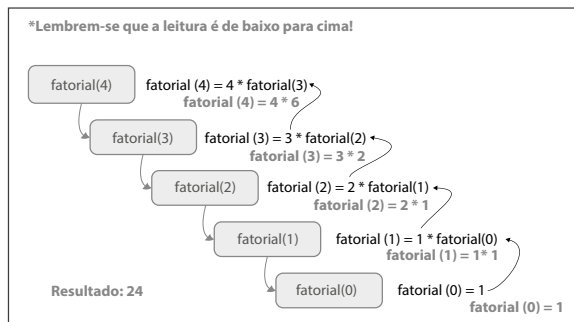
inicio
    Limpatela
    Escreval("Informe o número: ")
    Leia(n)
    fatN <- fatorial(n)
    Escreval("O fatorial de N é: ", fatN)
fimAlgoritmo

```

Fonte: Elaborada pelo autor, 2016.

Para entender melhor o conceito de recursividade, observe a Figura 8, onde é calculado, de forma recursiva, o fatorial do número 4.

Figura 8 - Fatorial com recursividade.



Fonte: Alves, 2016.

Note que em cada linha na qual a função fatorial aparece, duas chamadas à função “fatorial” são realizadas. A primeira referente ao número atual e a segunda ao número atual menos 1. O fatorial do número 4, neste exemplo, é o resultado do seguinte cálculo:

$$\text{Fatorial}(4) = 4 * \text{Fatorial}(3)$$

$$\text{Fatorial}(3) = 3 * \text{Fatorial}(2)$$

$$\text{Fatorial}(2) = 2 * \text{Fatorial}(1)$$

$$\text{Fatorial}(1) = 1 * \text{Fatorial}(0)$$

$$\text{Fatorial}(0) = 1$$

Na prática temos que  $\text{Fatorial}(4) = 4*3*2*1*1$ .

A seguir, veremos a implementação recursiva do cálculo do fatorial usando função.

A Figura 9 a seguir apresenta este algoritmo na ferramenta “VisuAlg”:

Figura 9 - Apresentação do algoritmo “fatorial\_recursivo” na ferramenta VisuAlg.

```
Algoritmo "fatorial_recursivo"  
// Disciplina: Lógica de Programação  
// Descrição : fatorial recursivo de um número  
  
funcao fatorial(n: inteiro): inteiro  
inicio  
    se (n<=2) entao  
        retorne n  
    senao  
        retorne n * (fatorial(n-1))  
    fimse  
fimfuncao  
  
var  
    n, fatN: inteiro  
inicio  
    Limpatela  
    Escreval("Informe o número: ")  
    Leia(n)  
    fatN <- fatorial(n)  
    Escreval("O fatorial de N é: ", fatN)  
fimAlgoritmo
```

Fonte: Elaborada pelo autor, 2016.

Ao implementarmos módulos recursivos, temos que ter cuidado especial com o critério de parada, que determina quando o módulo deverá parar de chamar asi mesmo. Isso impede que o chamamento ocorra infinitas vezes.


Mas quando exatamente podemos usar a recursividade? Ela pode ser utilizada principalmente quando o desempenho da versão recursiva for igual ou superior à versão iterativa do código, ou quando a versão iterativa é complicada demais. A recursividade deve ser evitada quando produz código complexo, de difícil entendimento e manutenção. Em outras palavras, usar a recursividade para resolver problemas em que exista um algoritmo iterativo simples, conciso, eficiente, e que resolva o mesmo problema, geralmente não é a melhor solução.



## Importante

Um limitador importante para a utilização de recursividade está no espaço de memória que a pilha de recursão, gerada pelos algoritmos, consome. Todo programa executado em um computador gera, em memória, uma pilha de execução (do inglês *Callstack*), cuja operação pode ser descrita conceitualmente da seguinte maneira:

Todo algoritmo constituído de uma ou mais funções (sendo uma delas a principal e a primeira a ser executada), ao fazer a chamada de uma função (recursiva ou não), é criada na memória do computador um novo espaço de trabalho, que contém todos os parâmetros e todas as variáveis locais da função, conhecida como pilha de execução (por cima do espaço de trabalho que invocou a função). É neste espaço que a execução da função começa (confinada ao seu espaço de trabalho). Quando ela termina, o seu espaço de trabalho é removido da pilha e descartado. O espaço de trabalho que estiver agora no topo da pilha é reativado e a execução é retomada do ponto em que havia sido interrompida. Logo, as funções recursivas possuem seu número de chamadas limitado ao espaço disponível na pilha.



## Resumindo

Neste capítulo, você aprendeu como utilizar funções e procedimentos para aplicação de recursividade em programação estruturada. Compreendeu

o conceito de modularização, suas vantagens e desvantagens, entendendo que se trata de uma técnica altamente recomendável, tanto pela eficiência no projeto e desenvolvimento, quanto pela confiabilidade e manutenibilidade do produto elaborado.

Analizamos, ainda, as principais ferramentas de modularização, que são as funções e procedimentos. Diversos exemplos de como esses recursos são utilizados na estrutura dos algoritmos foram apresentados, objetivando proporcionar uma visão mais prática de sua implementação. A partir de agora, você se torna capaz de modularizar programas, criar funções e procedimentos, recursivos ou não.

## Conclusão

Chegamos ao final da disciplina de Lógica de Programação. Como você deve ter notado, saímos diferentes da forma como começamos esta jornada. Iniciamos nosso aprendizado conhecendo um pouco dos tipos de linguagens utilizadas pelos computadores, como o computador trabalha com linguagens de baixo nível e alto nível, destacamos como as linguagens são classificadas e entendidas pelos processadores. Logo, tivemos a oportunidade de perceber que é possível programar os computadores através de algoritmos e praticar a lógica para o desenvolvimento de programas computacionais, e para isso aprendemos uma linguagem que lançou as bases do nosso aprendizado.

Compreender os principais tipos de dados operadores lógicos e expressões foi o primeiro passo para podermos aprender a estrutura de uma linguagem estruturada e suas sintaxes. Entramos em contato com comandos de entrada e saída em linguagem de programação estruturada, onde iniciamos a construção de programas simples, mas que ampliou nosso conhecimento de como é possível realizar a programação de computadores.

Aplicamos estruturas de seleção e suas características em problemas computacionais através de comandos que retornam valores verdadeiros ou falsos, que podem enriquecer mais nosso conhecimento e ampliar as soluções de algoritmos, mas que ainda faltavam outros conhecimentos para podermos alcançar.

Analizamos e aplicamos estruturas de repetição e suas características em problemas computacionais, onde aprendemos que é possível repetir um determinado código quantas vezes o programador desejar, otimizando assim a resolução de problema computacional.

Aplicamos o uso de vetores para armazenar, classificar e pesquisar listas e tabelas de valores. O aprendizado dessa técnica permitiu que pudéssemos armazenar uma quantidade de dados maiores nas variáveis criadas nos algoritmos.

Outra técnica muito importante para o programador foi a utilização do uso de funções e procedimentos para aplicação de recursividade em programação estruturada, o que garante a otimização dos programas, o que chamamos de recursividade.

Em todas as atividades foram utilizadas a ferramenta VisuAlg para facilitar a prática e aprendizado dos comandos e sequências lógicas que aprendemos durante todo o percurso da disciplina

Por fim, lembre-se: você, que é aluno de lógica de programação, mantenha-se sempre atualizado, ampliando constantemente seus estudos para obter um melhor desempenho no mercado de trabalho, com a base aqui aprendida é possível ampliar seus horizontes para novos conhecimentos.





## Referências

ALVES, R. Recursividade em Java. Desenvolvimento. **Linha de Código**. [201?]. Disponível em: <<http://www.linhadecodigo.com.br/artigo/3316/recursividade-em-java.aspx>>. Acesso em: 19 jan. 2016.

ASCENCIO, A. F. G; CAMPOS, E. A. V. **Fundamentos da programação de computadores**: algoritmos, Pascal, C/C ++. 3. ed. São Paulo: Pearson Education do Brasil, 2012. Disponível em: <<https://fael.bv3.digitalpages.com.br/reader#0>>. Acesso em: 21 dez. 2015.

BORENSZTEJN, P. **Registros en Python**: introducción a la computación. [201?]. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. Disponível em: <<http://www.dc.uba.ar/materias/int-com/2011/cuat1/Descargas/RegistrosP.pdf>>. Acesso em: 4 fev. 2016.

CORMEN, T. et al. **Algoritmos**: teoria e prática. 2. ed. Rio de Janeiro: Elsevier, 2002.

DANIELE, A. Mercado de TI brasileiro cresce e pode ficar acima do PIB em 2015. **EXAME.com**. 6 fev. 2015. Disponível em: <<http://exame.abril.com.br/tecnologia/noticias/mercado-de-ti-brasileiro-cresce-e-pode-e-ficar-acima-do-pib-em-2015>>. Acesso em: 17 dez. 2015.

FARRER, H. et al. **Algoritmos estruturados**. 3. ed. Rio de Janeiro: Guanabara, 1996.

FORBELLONE, A. L. V. **Lógica de programação**: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo: Prentice Hall, 2005.

GOMES, B. E. G.. **Linguagem C++**: registros. Fundamentos de programação. Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte (IFRN). [201?]. Disponível em: <<https://docente.ifrn.edu.br/brunogurgel/disciplinas/2012/fprog/aulas/cpp/aula10registros.pdf>>. Acesso em: 4 fev. 2016.

GOMES, D. F. Estruturas (struct) – C#: programação e estrutura de dados. **Slidshare.net**. 16 maio 2013. Disponível em: <<http://pt.slideshare.net/denisfg/explicando-estruturasregistros-no-c>>. Acesso em: 4 fev. 2016.

LUQUETA, G. Lógica de programação. **Professor Gerson**. [201?]. Disponível em: <[http://gerson.luqueta.com.br/index\\_arquivos/Algoritmos.pdf](http://gerson.luqueta.com.br/index_arquivos/Algoritmos.pdf)>. Acesso em: 4 fev. 2016.

OLIVEIRA, Wagner: Hardware: componentes básicos e funcionamento. **DOCPLAYER**. [201?]. Disponível em: <<http://docplayer.com.br/5615184-Hardware-componentes-basicos-e-funcionamento-wagner-de-oliveira.html>>. Acesso em: 18 jan. 2016.

PEREIRA, C. J. **Apostila sobre VisuAlg**. Joinville: Universidade do Estado de Santa Catarina; Centro de Ciências Tecnológicas. 2011. Disponível em: <[http://www2.joinville.udesc.br/~alp/arquivos/UDESC\\_Apostila\\_sobre\\_Visualg\\_2011.pdf](http://www2.joinville.udesc.br/~alp/arquivos/UDESC_Apostila_sobre_Visualg_2011.pdf)>. Acesso em: 13 dez. 2015.

PILHAS. Projeto de Algoritmos. **Instituto de Matemática e Estatística (IME)**. Universidade de São Paulo (USP), 29 jan. 2016. Disponível em: <<http://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html#pilhadeexecucao>>. Acesso em: 20 jan. 2016.

PUGA, S.; RISSETTI, G. **Lógica de programação e estruturas de dados**. 2 ed. São Paulo: Prentice Hall, 2005. Disponível em: <<https://fael.bv3.digitalpages.com.br/reader#0>>. Acesso em: 19 jan. 2016.

HAMANN, R. Quais as linguagens de programação mais usadas atualmente? **TECMUNDO**. 1º jul. 2015. Disponível em: <<http://www.tecmundo.com.br/programacao/82480-linguagens-programacao-usadas-atualmente-infografico.htm>>. Acesso em: 13 dez. 2015.

RECURSÃO e algoritmos recursivos. Projeto de Algoritmos. **Instituto de Matemática e Estatística (IME)**. Universidade de São Paulo (USP). [201?]. Disponível em: <<http://www.ime.usp.br/~pf/algoritmos/aulas/recu.html>>. Acesso em: 20 jan. 2016.

SANTANA JR., O. Algoritmos e estruturas de dados I: strings, registros e vetores (arrays). **Orivaldo Santana Jr.** [201?]. Disponível em: <[http://orivaldo.net/web/disciplinas/AEDI/apresentacoes/strings\\_registros\\_vetores.pdf](http://orivaldo.net/web/disciplinas/AEDI/apresentacoes/strings_registros_vetores.pdf)>. Acesso em: 4 fev. 2016.

SILVA, M. N. P. da. Fatorial e o principio fundamental da contagem. **Brasil Escola**. [201?]. Disponível em <<http://brasilecola.uol.com.br/matematica/fatorial-principio-fundamental-da-contagem.htm>>. Acesso em: 20 jan. 2016.

TANEMBAUM, A. S. **Organização estruturada de computadores**. 5. ed. São Paulo: Pearson Prentice Hall, 2007. Disponível em: <<https://fael.bv3.digitalpages.com.br/reader#127>>. Acesso em: 21 dez. 2015.

TEIXEIRA, C. B. Algoritmos e lógica de programação. **Cesarbt**. [201?]. Disponível em: <<http://www.cesarbt.xpg.com.br/arq/algoritmos.pdf>>. Acesso em: 4 fev. 2016.

VISUALG. Produtos. **Apoio Informática**. Disponível em: <<http://www.apoioinformatica.inf.br/produtos/visualg>>. Acesso em: 24 dez. 2015.

WERLICH, C. **Algoritmos**. Joinville: UniSociesc, 2007.

XAVIER, G. F. C. **Lógica de programação**. São Paulo: SENAC, 2007.

YEPES, I.; PADILHA, T. P. P. **Lógica de programação**. (Org. Faculdade Educacional da Lapa). Curitiba: Fael, 2013.

O material que você está recebendo trata dos conceitos básicos de lógica de programação. O objetivo é mostrar as estruturas de controle e de dados que você terá disponível para começar a trabalhar com o raciocínio lógico, que é a base para os ensinamentos de informática. Mesmo com a evolução constante da área de tecnologia, a estrutura lógica do programa de computador independente da arquitetura deste computador, e sim de uma visão de alto nível da área de lógica de programação, que se apresenta na forma de uma sequência de ações para a solução dos problemas, o algoritmo.

O conteúdo que você verá aqui é uma introdução para muitas disciplinas e de um mundo novo de desafios. E este conhecimento é a base para muitas outras que virão, por isso é muito importante que você compreenda a teoria e os exemplos que estão disponíveis neste material. Apesar de aqui você aprender a programar em uma linguagem específica, você terá toda a base necessária para criar tarefas que podem ser desenvolvidas em várias linguagens de programação. Para isto o livro tem vários exercícios com diferentes graus de complexidade. Aproveite!



ISBN 978-85-60531-53-0



9 788560 531530