ALGORITMOS E LÓGICA DE PROGRAMAÇÃO

PROFESSORES

Dra. Gislaine Camila Lapasini Leal
Me. Pietro Martins de Oliveira

ACESSE AQUI O SEU LIVRO NA VERSÃO **DIGITAL!**

EXPEDIENTE

Coordenador(a) de Conteúdo
Danillo Xavier Saes

Projeto Gráfico e Capa
Arthur Cantareli, Jhonny Coelho e Thayla Guimarães

Editoração
Luciana Pinheiro Grandizoli

Design Educacional
Amanda Peçanha

Revisão Textual
Nágela Neves da Costa

Ilustração
Marta Sayuri Kakitani

Fotos
Shutterstock

FICHA CATALOGRÁFICA

N964 Núcleo de Educação a Distância. **LEAL,** Gislaine Camila Lapasini; **OLIVEIRA,** Pietro Martins de.

Algoritmos e Lógica de Programação / Gislaine Camila Lapasini Leal; Pietro Martins de Oliveira. - Florianópolis, SC: Arqué, 2024.

256 p.

1. Algoritmos 2. Lógica 3. Programação EaD. I. Título.

CDD - 005.1

Bibliotecária: Leila Regina do Nascimento - CRB- 9/1722.

Ficha catalográfica elaborada de acordo com os dados fornecidos pelo(a) autor(a).

Impresso por:

Unicesumar

NEAD - Núcleo de Educação a Distância
Av. Guedner, 1610, Bloco 4Jd. Aclimação - Cep 87050-900 | Maringá - Paraná
www.unicesumar.edu.br | 0800 600 6360

AVALIE ESTE LIVRO!

CRIAR MOMENTOS DE APRENDIZAGENS INESQUECÍVEIS É O NOSSO OBJETIVO E POR ISSO, **GOSTARÍAMOS DE SABER COMO FOI SUA EXPERIÊNCIA.**

Conta para nós! leva ***menos de 2 minutos*** *Vamos lá?!*

DIGITE O CÓDIGO

02510794

RESPONDA A PESQUISA

Dra. Gislaine Camila Lapasini Leal

Doutora em Engenharia Elétrica e Informática Industrial pela Universidade Tecnológica Federal do Paraná. Mestre em Ciência da Computação pela Universidade Estadual de Maringá (2010). Graduada em Engenharia de Produção – Software, pela Universidade Estadual de Maringá (2007) e em Processamento de Dados pelo Centro Universitário de Maringá (2004). Atualmente, é professora do Departamento de Engenharia de Produção da Universidade Estadual de Maringá, atuando na área de Apoio à Tomada de Decisão (Pesquisa Operacional e Gestão de Tecnologia da Informação). Dedica-se à seguintes áreas de pesquisa: Otimização, Apoio à Tomada de Decisão em Sistemas de Produção, Startups de Software, Melhoria da Qualidade e Produtividade no Desenvolvimento de Software e Melhoria de Processos.

http://lattes.cnpq.br/7810321373328408

Me. Pietro Martins de Oliveira

Mestre em Ciência da Computação na área de Visão Computacional, pela Universidade Estadual de Maringá (2015). Graduado em Engenharia de Computação pela Universidade Estadual de Ponta Grossa (2011). Atuou como analista de sistemas e programador nas empresas Siemens Enterprise Communications (Centro Internacional de Tecnologia de Software – CITS, 2011-2012) e Benner Saúde Maringá (2015). Tem experiência como docente e coordenador dos cursos de Bacharelado em Engenharia de Software, Sistemas de Informação e Engenharia de Produção.

http://lattes.cnpq.br/1793084774574585

ALGORITMOS E LÓGICA DE PROGRAMAÇÃO

Seja bem-vindo(a) à disciplina de Algoritmos e Lógica de Programação. Sou a professora Gislaine Camila e, juntamente com o professor Pietro Martins, estudaremos, nesta disciplina, o conceito central da programação, os algoritmos e a lógica de programação.

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

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

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

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

Na Unidade 1, estudaremos os conceitos e princípios básicos de lógica de programação e veremos que, empiricamente, já conhecemos algoritmos e utilizamos vários deles em nosso dia a dia. A Unidade 2 abordará o conceito de estrutura condicional, que nos possibilita impor condições para a execução de determinada instrução ou conjunto de instruções. Na Unidade 3, aprenderemos a construir algoritmos utilizando a estrutura de repetição para repetir um trecho de código quantas vezes forem necessárias, sem ter que reescrever trechos idênticos. A Unidade 4 trata das estruturas de dados homogêneas e heterogêneas, as quais permitem agrupar diversas informações em uma única variável. Por fim, na Unidade 5, estudaremos a modularização de algoritmos, utilizando sub-rotinas. Conheceremos as sub-rotinas do tipo procedimento e função, sua sintaxe, peculiaridades e como utilizá-las. Ao longo deste livro, você encontrará indicações de leituras complementares, as quais enriquecerão o seu conhecimento com mais exemplos de algoritmos. É importante que você desenvolva as atividades de autoestudo, pois, neste momento, você poderá colocar em prática os conhecimentos adquiridos e identificar eventuais dificuldades.

Desejamos a você um bom estudo!

ÍCONES

pensando juntos

Ao longo do livro, você será convidado(a) a refletir, questionar e transformar. Aproveite este momento!

explorando Ideias

Neste elemento, você fará uma pausa para conhecer um pouco mais sobre o assunto em estudo e aprenderá novos conceitos.

quadro-resumo

No fim da unidade, o tema em estudo aparecerá de forma resumida para ajudar você a fixar e a memorizar melhor os conceitos aprendidos.

conceituando

Sabe aquela palavra ou aquele termo que você não conhece? Este elemento ajudará você a conceituá-la(o) melhor da maneira mais simples.

conecte-se

Enquanto estuda, você encontrará conteúdos relevantes online e aprenderá de maneira interativa usando a tecnologia a seu favor.

Quando identificar o ícone de **QR-CODE**, utilize o aplicativo **Unicesumar Experience** para ter acesso aos conteúdos online. O download do aplicativo está disponível nas plataformas: Google Play App Store

CONTEÚDO

PROGRAMÁTICO

UNIDADE 01 — 8
CONCEITOS BÁSICOS

UNIDADE 02 — 52
ESTRUTURA CONDICIONAL

UNIDADE 03 — 92
ESTRUTURAS DE REPETIÇÃO

UNIDADE 04 — 137
ESTRUTURAS DE DADOS HOMOGÊNEAS E HETEROGÊNEAS

UNIDADE 05 — 178
SUB-ROTINAS

FECHAMENTO — 211
CONCLUSÃO GERAL

UNIDADE

CONCEITOS BÁSICOS

PROFESSORES
Dra. Gislaine Camila Lapasini Leal
Me. Pietro Martins de Oliveira

PLANO DE ESTUDO

A seguir, apresentam-se as aulas que você estudará nesta unidade: • Conceituando algoritmos • Estudando variáveis • Expressões • Funções Intrínsecas • Atribuição • Entrada de dados • Saída de dados • Construindo algoritmos.

OBJETIVOS DE APRENDIZAGEM

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

INTRODUÇÃO

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

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

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

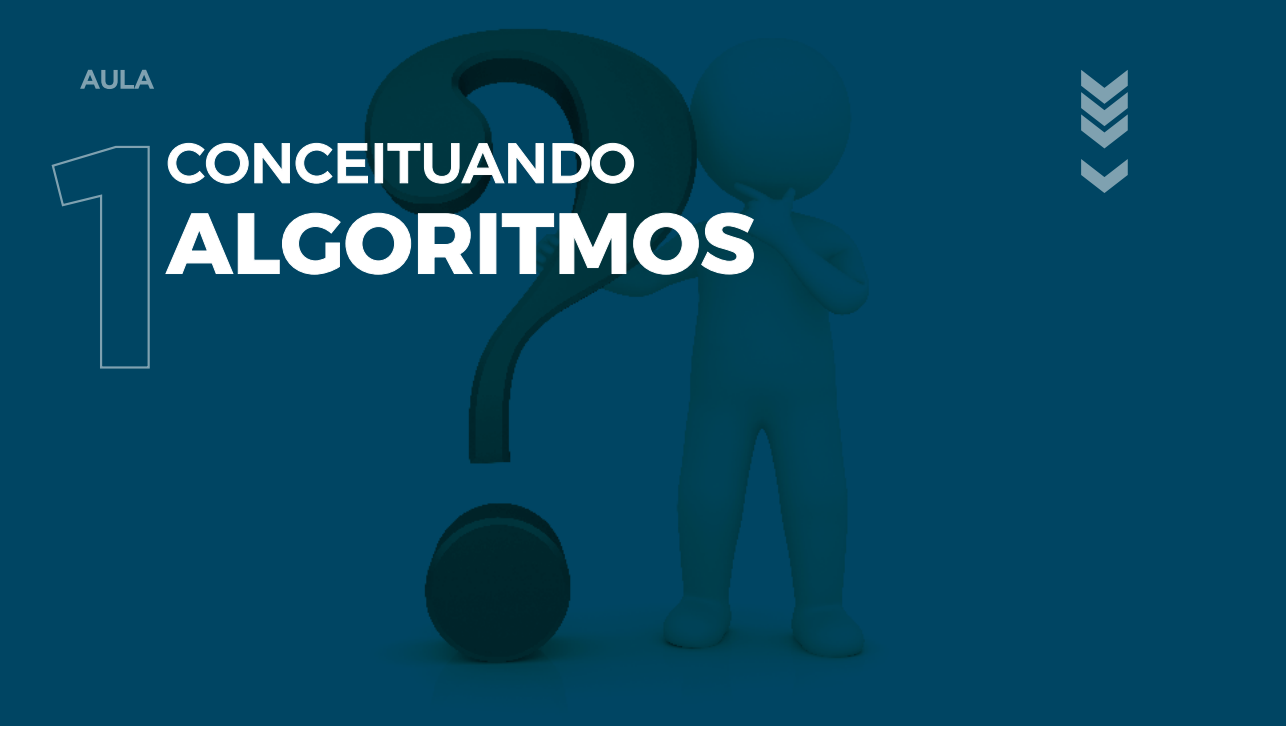
Ao formular algoritmos, precisamos guardar algumas informações do problema. Para isso, veremos o conceito de variáveis e constantes. Conheceremos os tipos de variáveis numéricas, textuais e lógicas e a característica da informação que cada uma delas pode armazenar. Além de guardar informações, precisaremos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

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

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

AULA

1 CONCEITUANDO ALGORITMOS



Para iniciarmos a nossa disciplina de Algoritmos e Lógica de Programação I, precisamos entender o conceito de algoritmos. Você já ouviu este termo? Um algoritmo consiste em uma sequência finita de passos (instruções) não ambíguos para solucionar um problema. Podemos ter vários algoritmos que resolvem um mesmo problema, dessa forma, um algoritmo não é a única solução. Um algoritmo é um caminho para a solução de um problema, visto que podem existir diversos caminhos que conduzem a ela (LOPES; GARCIA, 2002).

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

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

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

Quadro 1 - Algoritmo para ir ao trabalho / Fonte: os autores.

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

Agora que estamos mais familiarizados com o termo algoritmo, podemos perceber que eles fazem parte do nosso dia a dia, por exemplo, quando seguimos instruções para o uso de medicamentos, realizamos uma ligação telefônica, trocamos uma lâmpada, montamos um móvel ou aparelho e quando seguimos uma receita culinária (ZIVIANI, 2004). O próximo quadro apresenta um conjunto de instruções a serem seguidas para tomar sal de frutas (uso de medicamento).

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

Quadro 2 - Algoritmo para tomar sal de frutas / Fonte: os autores.

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

1. Inserir o número.
2. Apertar o botão para fazer a ligação.
3. Esperar atender.
4. Falar ao telefone.
5. Apertar o botão para desligar a ligação.

Quadro 3 - Algoritmo para fazer ligação telefônica / Fonte: adaptado de Lopes e Garcia (2002).

Uma solução para o problema de trocar uma lâmpada pode ser representada por cinco instruções, como é mostrado no próximo quadro.

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

Quadro 4 - Algoritmo para trocar lâmpada / Fonte: Lopes e Garcia (2002).

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

Como construir algoritmos

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

- **Compreender o problema**: definir qual o objetivo do algoritmo.

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

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

Tipos de algoritmos

Ascencio e Campos (2010) destacam que as formas mais utilizadas para representar os algoritmos são: descrição narrativa, fluxograma e pseudocódigo. A **descrição narrativa** consiste na representação do problema por meio da linguagem natural, descrevendo os passos que devem ser seguidos para a resolução de um problema, conforme visto em "Conceituando Algoritmos". Como vantagem dessa representação, destaca-se a facilidade de aprendizado. No entanto esse tipo de descrição pode ser ambíguo, gerar diversas interpretações e dificultar a conversão do algoritmo em um programa de computador (ASCENCIO; CAMPOS, 2010). Tomaremos como exemplo a soma de dois números. Seguindo os passos descritos em "Como construir algoritmos", temos:

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

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

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

Quadro 5 - Descrição Narrativa – Algoritmo soma / Fonte: os autores.

O **fluxograma** consiste em uma notação gráfica que permite indicar as ações e decisões que devem ser seguidas para resolver o problema. Os símbolos utilizados para construir o fluxograma são apresentados a seguir.

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

Quadro 6 - Simbologia de fluxograma / Fonte: adaptado de Manzano e Oliveira (1997).

Para melhor compreensão, apresentamos, a seguir, o fluxograma para o problema de somar dois números.

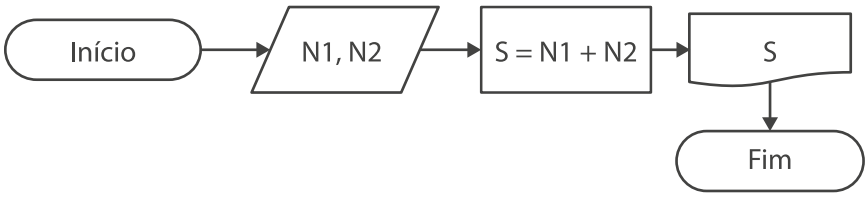


Figura 1 - Fluxograma – Algoritmo soma / Fonte: os autores.

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

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

O quadro que se apresenta, posteriormente, mostra o pseudocódigo do algoritmo que efetua a soma de dois números. As palavras em negrito são termos que possuem significado específico. Chamam-se "palavras reservadas" da linguagem. Notem que todo algoritmo possui um nome, é uma sequência finita e delimitado por início e fim.

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

Quadro 7 - Pseudocódigo – Algoritmo soma / Fonte: os autores.

O pseudocódigo é muito próximo à linguagem de programação, de tal modo que a "tradução" do algoritmo do pseudocódigo para o código-fonte, em linguagem de programação, é quase imediata, basta conhecer as palavras reservadas da linguagem, conforme destacam Ascencio e Campos (2010).

Neste livro, os pseudocódigos não serão compilados nem interpretados diretamente pela máquina, ou seja, não utilizaremos software para executar os algoritmos, mas realizaremos "teste de mesa". Este tem como finalidade atestar a veracidade do resultado emitido pelo processamento do algoritmo, simulando sua execução como se fosse a própria máquina.

AULA

2 ESTUDANDO VARIÁVEIS

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

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

Na figura a seguir, temos uma variável com o nome Idade. Esta variável pode guardar apenas valores numéricos inteiros. Com isso, temos que o valor "casa" não pode ser armazenado nessa caixa, visto que se trata de um conjunto de caracteres.

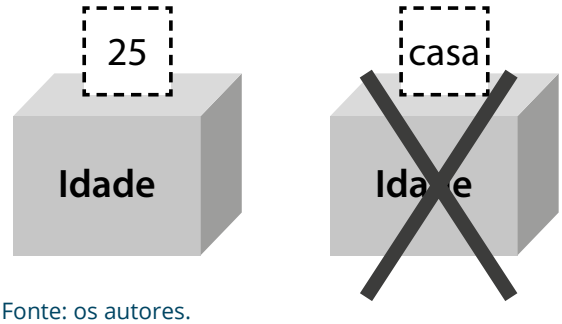


Figura 2 - Variável / Fonte: os autores.

Em um algoritmo, o conteúdo de uma variável pode ser modificado, consultado ou apagado quantas vezes forem necessárias. É importante ter ciência de que a variável armazena apenas um conteúdo por vez. Na próxima figura, nossa caixa, denominada `Idade,` recebe uma sequência de valores, sendo eles 25, 60, 13 e, por fim, o valor 36. Sabemos que uma variável armazena apenas um conteúdo por vez, dessa forma, quando realizarmos uma consulta, obteremos o valor 36 para a variável `Idade`, pois este foi o último dado armazenado na variável.

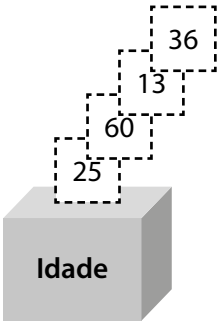


Figura 3 - Variável / Fonte: os autores.

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

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

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

Os identificadores são os nomes das variáveis, dos programas, das rotinas, constantes etc. Para nomear esses identificadores, devemos seguir algumas regras (ASCENCIO; CAMPOS, 2010):

- O nome sempre deve iniciar com uma letra, obrigatoriamente. Assim, poderíamos ter variáveis com nomes como "media", "altura", "idade", "cidade". Todavia não podemos ter identificadores como "2cidade", "4x", e assim por diante.
- O nome não pode conter espaços, ou seja, não podemos denominar uma variável de "altura media".

- O identificador de uma variável não pode conter caracteres especiais ($, #, @, ?, !, *).
- Nenhuma palavra reservada poderá ser nome de variável. As palavras reservadas têm uso específico no pseudocódigo. Alguns exemplos são: `Var`, `tipo`, `início`, `fim`, `se`, `então`, `senão`, `enquanto`, `repita`, `faça`, `caso`, `até`, `procedimento`, `função` e outros. Não se assuste com todos esses termos, os veremos no decorrer desta unidade e, com a prática, logo vocês estarão familiarizados com todos eles.

O quadro a seguir apresenta alguns exemplos de identificadores válidos e inválidos, exemplificando cada uma das regras descritas anteriormente. É importante lembrar que seguiremos essas mesmas regras para nomear tanto variáveis quanto constantes, algoritmos, rotinas e outros.

Identificadores válidos	Identificadores inválidos
A	2ª
a	b@
media	media idade
altura2	caso
media_idade	se
x36	x*y

Quadro 8 – Exemplos de identificadores / Fonte: os autores.

Até o momento, sabemos o que é um algoritmo; o que são variáveis e qual a sua importância. Em seguida, entenderemos quais os tipos de dados que dispomos para armazenar nas variáveis de nossos algoritmos.

Tipos de variáveis

Como sabemos, as nossas variáveis só podem armazenar dados do mesmo tipo. Os tipos de variáveis são: `inteiro, real, caractere` e `lógico`. Uma variável **inteira** armazena dados numéricos, que não possuem componentes decimais ou fracionários. O número pode ser positivo ou negativo (ASCENCIO; CAMPOS, 2010). Alguns exemplos de números inteiros são: -3, -1, 0, 4, 7. Dessa forma, a declaração de uma variável inteira é realizada da seguinte maneira:

```
Var

Idade : inteiro
```

Outro tipo de variável numérica é a `real`, que armazena componentes decimais ou fracionários, podendo estes ser positivos ou negativos. Alguns exemplos de números reais são: 15.02, 4.5, 6.07 e outros (repare que a notação para a separação entre as partes inteira e decimal é feita pelo ponto, e não pela vírgula, como estamos acostumados). A declaração de uma variável do tipo real é dada por:

```
Var

Altura : real
```

Neste ponto, vocês podem se perguntar: que diferença faz declarar uma variável como inteira ou real? Declaramos todas as variáveis como real, já que o conjunto dos inteiros está incluso nos reais?

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

Até o momento, conhecemos as variáveis numéricas, no entanto existem dados textuais que devemos processar, para tanto, utilizamos as variáveis do tipo `caractere`, também conhecidos como alfanuméricos, *strings* ou cadeia de caracteres (MANZANO; OLIVEIRA, 1997).

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

As variáveis do tipo caractere possuem um comprimento associado a elas, por exemplo, "Maria" tem comprimento 5, "21/06/1984" tem comprimento 10. O comprimento corresponde ao número de caracteres que a variável possui, como veremos nas próximas unidades, ela se comporta como um vetor. Não precisa se preocupar com o conceito de vetores por enquanto, apenas saiba que, ao declarar uma variável do tipo `caractere`, devemos definir o tamanho máximo de caracteres que ela poderá armazenar. Para tanto, basta inserir o tamanho entre os colchetes, ao lado da palavra reservada vetor. A declaração de uma variável do tipo `caractere` é realizada da seguinte forma:

```
Var
Nome : vetor[1..10] de caractere
```

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

```
Var
Ocupado: lógico
```

Constante

Uma constante armazena informações que não variam com o tempo, ou seja, o seu conteúdo é um valor fixo. Dessa forma, uma vez que o valor da constante foi definido, não é possível alterar seu conteúdo durante a execução do algoritmo. Da mesma forma, as variáveis e todas as constantes devem ser definidas no início do algoritmo. O comando utilizado para definir constantes é o `Const` e sua definição é dada por:

```
Const
<nome da constante> = <valor>
```

AULA

3 EXPRESSÕES

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

As expressões **aritméticas** são aquelas em que o resultado consiste em um valor numérico. Dessa forma, apenas operadores aritméticos e variáveis numéricas (`inteiro` e `real`) podem ser utilizadas em expressões desse tipo. O quadro a seguir apresenta os operadores aritméticos, destacando suas representações, forma de uso e prioridade. A prioridade entre operadores define a ordem de precedência em que as operações devem ser avaliadas (resolvidas) dentro de uma mesma expressão.

Operação	Operador	Operador	Prioridade
Soma	+	Utilizado para efetuar a soma entre duas ou mais variáveis. Dada uma variável `A` e outra `B`, temos que a soma delas é representada por `A + B`.	4

Operação	Operador	Operador	Prioridade
Subtração	-	Simboliza a subtração do valor entre duas variáveis. Supondo uma variável `A` e `B`, a diferença entre elas é dada por: `A - B`.	4
Multiplicação	*	O produto entre duas variáveis `A` e `B` é representado por `A * B`.	3
Divisão	/	A divisão entre duas variáveis `A` e `B` é dada por: `A / B`. Em relação à divisão, é importante lembrar que não existe divisão por 0.	3
Exponenciação	**	É representada por `**` mais o número que se quer elevar. Se quisermos elevar o valor da variável `A` ao quadrado, representamos por: `A ** 2`.	2
Resto	`MOD`	Usado quando se deseja encontrar o resto da divisão entre duas variáveis `A` e `B`. A representação é dada por `A MOD B`. Supondo `A = 3` e `B = 2`, temos que `A MOD B = 1`.	3
Divisão inteira	`DIV`	Representa o quociente da divisão entre dois números. Tomando `A = 7 e B = 2`, temos que `A div B` resulta em 3.	3

Quadro 9 - Operadores aritméticos / Fonte: os autores.

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

Operador	Símbolo	Exemplo
Igual	=	A = 1
Diferente	<>	A <> B
Maior	>	A > 5
Menor que	<	B < 12
Maior ou igual a	>=	A >= 6
Menor ou igual a	<=	B <= 7

Quadro 10 – Operadores relacionais / Fonte: adaptado de Ascencio e Campos (2010).

As expressões lógicas são aquelas cujo resultado consiste em um valor lógico `verdadeiro` ou `falso`. Nesse tipo de expressão, usa-se os operadores relacionais, os operadores lógicos ou as expressões matemáticas. No quadro posterior, são descritos cada um dos operadores lógicos: conjunção, disjunção e negação.

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

Quadro 11 - Operadores lógicos / Fonte: adaptado de Lopes e Garcia (2002).

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

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

Fiquem tranquilo(as)! Colocaremos em prática as questões relacionadas à precedência de operadores em "Construindo algoritmos".

AULA 4

FUNÇÕES INTRÍNSECAS

As funções intrínsecas são fórmulas, em sua maioria, matemáticas, que já estão prontas para utilizarmos em nossos algoritmos. O quadro a seguir apresenta as principais funções, destacando o comando associado e o que elas fazem.

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

Quadro 12 - Funções / Fonte: adaptado de Ascencio e Campos (2010); Lopes e Garcia (2002).

Observe, no quadro anterior, que as funções recebem parâmetros entre os parênteses. O parâmetro de uma função é o valor que ela recebe como entrada, podendo ser um valor literal (número, de maneira direta), uma constante, uma variável ou até uma expressão. Com base no valor do parâmetro é que a função realiza o processamento automaticamente e produz, como saída, o resultado esperado.

AULA

5 ATRIBUIÇÃO

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

O símbolo utilizado para a atribuição é `<-`, e a representação é dada por:

```
<identificador> <- <expressão>
```

Na sintaxe de atribuição anterior, o campo `identificador` representa a variável a qual se atribui um valor, e o termo `expressão` é o conteúdo que será atribuído à variável, podendo ser uma expressão aritmética, lógica ou textual, ou mesmo uma constante, um literal etc. Tomemos como exemplo atribuir a uma variável denominada `idade` o valor 26. Para isso, temos que:

```
idade <- 26
```

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

AULA

6 ENTRADA DE DADOS

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

```
Leia(<variável>)
```

Ao utilizarmos este comando, o computador fica "aguardando" uma ação do usuário, que é digitar o conteúdo que será, posteriormente, associado à variável indicada, dentro dos parênteses.

AULA

7 SAÍDA DE DADOS

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

```
Escreva(<variável> ou <texto>)
```

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

```
Escreva("Estou estudando Algoritmos")
```

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

```
Escreva("A idade é:", idade)
```

```
Escreva(n1, "x", n2, "é igual a", produto)
```

AULA

3 CONSTRUINDO ALGORITMOS

Vimos o conceito de variáveis, constantes, expressões; conhecemos os operadores, as funções, os comandos de atribuição, a entrada e a saída de dados. Nesta aula, reuniremos todos estes conhecimentos na construção de algoritmos.

Problema 1

Formular um algoritmo que leia e apresente as seguintes informações de uma pessoa: nome, idade, peso, altura e telefone. Antes de partir para a elaboração do algoritmo, dividiremos o problema, tal como aprendemos em "Como construir algoritmos", em objetivo do algoritmo, da entrada, do processamento e da saída. Desse modo, temos:

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

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

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

```
Algoritmo problema1
```

Em seguida, temos a declaração da lista de variáveis, que é dada por:

```
Var
    idade        : inteiro
    peso, altura : real
    nome         : vetor[1..50] de caractere
    telefone     : vetor[1..15] de caractere
```

Note que as variáveis do mesmo tipo podem ser declaradas na mesma linha, como é o caso de `peso` e `altura`. Se as variáveis do tipo `caractere` tivessem o mesmo tamanho, poderíamos declará-las de modo análogo. Declarar como `vetor[1..50] de caractere` indica que o tamanho máximo que a variável armazenará é de 50 caracteres.

Todo algoritmo é delimitado pela instrução de `Início`, logo:

```
Início
```

Na entrada de dados, precisamos obter as informações definidas por meio do comando `Leia`. No entanto é importante enviar sempre uma mensagem a fim de informar ao usuário o dado solicitado. Desse modo, utilizamos o comando `Escreva`. Com isto, temos:

```
Escreva("Informe o nome:")

Leia(nome)

Escreva("Informe a idade:")

Leia(idade)

Escreva("Informe o peso:")

Leia(peso)

Escreva("Informe a altura:")

Leia(altura)

Escreva("Informe o telefone:")

Leia(telefone)
```

Como não há processamento, realizaremos a saída de dados por meio do comando `Escreva`:

```
Escreva("O nome é:", nome)

Escreva("A idade é:", idade)

Escreva("O peso é:", peso)

Escreva("A altura é:", altura)

Escreva("O telefone é:", telefone)
```

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

```
Fim
```

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

```
Algoritmo problema1
Var
    idade          : inteiro
    peso, altura   : real
    nome           : vetor[1..50] de caractere
    telefone       : vetor[1..15] de caractere
Início
    Escreva("Informe o nome:")
    Leia(nome)
    Escreva("Informe a idade:")
    Leia(idade)
    Escreva("Informe o peso:")
    Leia(peso)
    Escreva("Informe a altura:")
    Leia(altura)
    Escreva("Informe o telefone:")
    Leia(telefone)
    Escreva("O nome é:", nome)
    Escreva("A idade é:", idade )
    Escreva("O peso é:", peso)
    Escreva("A altura é:", altura)
    Escreva("O telefone é:", telefone)
Fim
```

Quadro 13 - Pseudocódigo - Problema 1 / Fonte: os autores.

O resultado da execução do algoritmo é apresentado no quadro:

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

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

Quadro 14 - Resultado da execução – Problema 1 / Fonte: os autores.

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

Problema 2

Um quadrado é uma figura geométrica com quatro lados de mesmo comprimento (`L`) e quatro ângulos retos. Elabore um algoritmo para calcular a área e o perímetro de um quadrado. Observando a figura a seguir, a área é dada pela superfície (preenchimento entre o pontilhado), e o perímetro é a medida do contorno do objeto, a soma dos quatro lados (pontilhado).

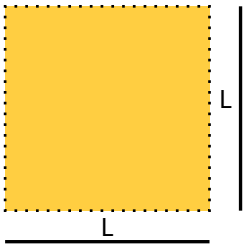


Figura 4 - Quadrado / Fonte: os autores.

Não elaboraremos o pseudocódigo diretamente. Primeiro, dividiremos o problema, tal como aprendemos em "Como construir algoritmos", em objetivo do algoritmo, da entrada, do processamento e da saída. Com isso, temos:

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

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

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

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

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

```
Algoritmo quadrado
```

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

```
Var
L, A, P: real
```

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

```
Início
```

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

```
Escreva("Informe o valor do lado do quadrado:")
Leia(L)
```

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

```
A <- L * L
P <- 4 * L
```

Estas instruções indicam que a variável `A` e a variável `P` armazenarão o resultado das expressões que definimos. Passamos pela entrada e pelo processamento, agora, temos que informar o resultado, ou seja, a saída de dados, realizada pelo comando `Escreva`. O objetivo é informar o valor da área e do perímetro, portanto:

```
Escreva("O perímetro é:", P)
Escreva("A área é:", A)
```

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

```
Fim
```

O algoritmo completo que formulamos encontra-se no quadro a seguir. Note que as palavras-chave estão em negrito. Lembre-se que essas palavras são reservadas, dessa forma, não podemos utilizá-las para nomear os identificadores (programas, variáveis, constantes e outros).

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

Quadro 15 - Algoritmo completo / Fonte: os autores.

Simularemos, agora, o funcionamento do nosso algoritmo, ou seja, realizaremos o **teste de mesa**. Esta etapa é importante, pois auxilia na compreensão, na verificação e validação do algoritmo. Supondo que `L` = 3, logo, temos:

`A = 3 * 3`

`P = 4 * 3`

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

O perímetro é: 12
A área é: 9

No quadro posterior, apresenta-se o resultado da execução do algoritmo, mostrado em vídeo.

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

Quadro 16 - Resultado da execução - Problema 2 / Fonte: os autores.

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

pensando juntos

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

Problema 3

Elaborar um algoritmo que apresente salário bruto, salário líquido, INSS e FGTS de um funcionário, sabendo que o salário bruto é dado pelo seguinte cálculo: (Horas Trabalhadas + Horas Extras*(1+Porcentagem Hora Extra))*-Valor por Hora. A Porcentagem de Hora Extra é 100%. O valor do INSS é dado por Salário Bruto * 9%, e o FGTS é o Salário Bruto * 8%. Por fim, o salário líquido é dado pelo salário bruto – INSS. Ao estruturarmos o problema em objetivo, entrada, processamento e saída, temos:

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

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

O pseudocódigo para o problema descrito é apresentado no quadro a seguir.

```
Algoritmo salario
Var
     htrabalhadas, hextra, vhora, sbruto, inss:real
     fgts, sliquido : real
Const
     phoraextra = 100
Início
     Escreva("Informe o número de horas traba-
     lhadas:")
     Leia(htrabalhadas)
     Escreva("Informe o número de horas extras:")
     Leia(hextra)
     Escreva("Informe o valor da hora:")
     Leia(vhora)
     sbruto <- (htrabalhadas + extra *
     (1+(phoraextra/100)))* vhora
     inss <- sbruto * (9/100)
     FGTS <- sbruto * (8/100)
     sliquido <- sbruto - inss
     Escreva("O salário bruto é:", sbruto)
     Escreva("O salário líquido é:", sliquido)
     Escreva("O INSS é:", inss)
     Escreva("O FGTS é:", FGTS)
Fim
```

Quadro 17 - Pseudocódigo – Problema 3 / Fonte: os autores.

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

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

```
sbruto <- (20 + 5*(1+(100/100)))*10
```

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

Ao final da execução do algoritmo, obteremos os resultados, conforme apresentado no quadro a seguir.

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

Quadro 18 - Resultado da execução - Problema 3 / Fonte: os autores.

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

Problema 4

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

- **Objetivo do algoritmo**: apresentar o antecessor e o sucessor de um número.
- **Entrada**: ler um número inteiro.
- **Processamento**: calcular o valor do número menos um (antecessor) e o valor do número mais um (sucessor).
- **Saída**: apresentar o antecessor, o número e o sucessor.

```
Algoritmo numero

Var

     num: inteiro

Início

     Escreva("Informe um número inteiro:")

     Leia(num)

     Escreva("O antecessor é:", num-1)

     Escreva("O número é:", num )

     Escreva("O sucessor é:", num +1)

Fim
```

Quadro 19 - Pseudocódigo – Problema 4 / Fonte: os autores.

Problema 5

Escreva um algoritmo que calcule a área de um triângulo.

- **Objetivo do algoritmo**: calcular a área de triângulo.
- **Entrada**: obter o valor da base e da altura do triângulo.
- **Processamento**: calcular o valor da área que é dado por: $A = \frac{base\ x\ altura}{2}$.
- **Saída**: imprimir o valor da área.

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

Quadro 20 - Pseudocódigo – Problema 5 / Fonte: os autores.

Problema 6

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

- **Objetivo do algoritmo**: calcular o valor a pagar de um produto.
- **Entrada**: ler o preço de um produto e o percentual de desconto.
- **Processamento**: calcular o valor do desconto e subtrair do preço do produto.
- **Saída**: imprimir o valor a pagar do produto e o valor do desconto.

```
Algoritmo produto
Var
     preco, pdesconto,  vdesconto, vpagar: real
Início
     Escreva("Informe o preço do produto:")
     Leia(preco)
     Escreva("Informe o percentual do desconto:")
     Leia(pdesconto)
     vdesconto  <- (preco * pdesconto)/100
     vpagar  <- preco - vdesconto
     Escreva("O valor a pagar é:", vpagar)
     Escreva("O valor do desconto é:", vdesconto)
Fim
```

Quadro 21 - Pseudocódigo – Problema 6 / Fonte: os autores.

Problema 7

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

- **Objetivo do algoritmo**: calcular o faturamento mensal e o anual de uma locadora de carros.
- **Entrada**: ler a quantidade de carros e o valor do aluguel mensal.
- **Processamento**: calcular o faturamento mensal, considerando que todos os carros foram locados, e o faturamento anual, considerando que 20% dos carros não serão locados.
- **Saída**: imprimir o valor do faturamento mensal e o valor do faturamento anual.

```
Algoritmo faturamento
Var
    valor, fmensal, fanual: real
    qtdade: inteiro
Início
    Escreva("Informe a quantidade de carros:")
    Leia(qtdade)
    Escreva("Informe o valor da locação:")
    Leia(valor)
    fmensal  <- qtdade * valor
    fanual  <- ((qtdade * 0,8) * valor)*12
    Escreva("O faturamento mensal é de:", fmensal)
    Escreva("O faturamento anual considerando a
    locação de 80% dos carros é:", fanual)
Fim
```

Quadro 22 - Pseudocódigo – Problema 7 / Fonte: os autores.

Problema 8

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

- **Objetivo do algoritmo**: calcular o salário líquido e o salário bruto de um professor.
- **Entrada**: ler o valor da hora-aula, o número de aulas dadas no mês, e o percentual de desconto do INSS.
- **Processamento**: calcular o salário bruto, dado pelo produto do número de horas-aula pelo valor da hora-aula, e o salário líquido, que é o salário bruto menos o desconto do INSS.
- **Saída**: imprimir o valor do salário bruto e o valor do salário líquido.

```
Algoritmo salario
Var
     vhora, naulas,  pdesconto, sliquido, sbruto:
real
Início
     Escreva("Informe o valor da hora-aula:")
     Leia(vhora)
     Escreva("Informe o número de horas-aula dadas
     no mês:")
     Leia(naulas)
     Escreva("Informe o percentual de desconto do
     INSS:")
     Leia(pdesconto)
     sbruto <- vhora * naulas
     sliquido <- sbruto - (sbruto * pdesconto)/100
     Escreva("O valor do salário bruto é:", sbruto)
     Escreva("O valor do salário líquido é:", sli-
     quido)
Fim
```

Quadro 23 - Pseudocódigo – Problema 8 / Fonte: os autores.

Problema 9

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

- **Objetivo do algoritmo**: calcular a área e o perímetro de um círculo.
- **Entrada**: ler o valor do raio.

- **Processamento**: calcular a área que é dada por: $A = \pi r^2$ e o perímetro que é dado por $A = \pi r$, onde `A` é o valor da área, `P` é o valor do perímetro, e `r` é o raio do círculo.
- **Saída**: imprimir o valor da área e do perímetro.

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

Quadro 24 - Pseudocódigo – Exercício 9 / Fonte: os autores.

Problema 10

Elabore um algoritmo que leia um número inteiro e, posteriormente, apresente a raiz quadrada e o valor desse número elevado ao quadrado.

- **Objetivo do algoritmo**: calcular a raiz quadrada e o valor do número elevado ao quadrado.
- **Entrada**: ler um número inteiro.
- **Processamento**: utilizar as funções SQR e SQRT para elevar ao quadrado e obter a raiz quadrada, respectivamente.
- **Saída**: imprimir a raiz quadrada e o valor do número elevado ao quadrado.

```
Algoritmo funcoes
Var
      raiz: real
      num, quadrado: inteiro
Início
      Escreva("Informe um número inteiro:")
      Leia(num)
      raiz <- sqrt(num)
      quadrado <- sqr(num)
      Escreva("A raiz quadrada é:", raiz)
      Escreva("O número elevado ao quadrado é:",
      quadrado)
Fim
```

Quadro 25 - Pseudocódigo – Problema 10 / Fonte: os autores.

conecte-se

História da Programação: Como tudo começou! Disponível em: http://www.techtudo.com.br/platb/desenvolvimento/2011/06/20/historia-da-programacao-como-tudo-comecou/

Para entender melhor como elaborar o teste de mesa ou a simulação de um algoritmo, veja: https://youtu.be/qV-5KqEOIrQ

CONSIDERAÇÕES FINAIS

Nesta unidade, introduzimos o conceito central da disciplina. Desse modo, vimos que um algoritmo consiste em um conjunto de instruções para solucionar um problema. Nos problemas, verificamos a possibilidade de vários algoritmos distintos que o resolvem, ou seja, um problema pode ter várias soluções algorítmicas.

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

Podemos representar algoritmos de três formas: descrição narrativa, fluxograma e pseudocódigo. Vimos que, na descrição narrativa, a solução é dada por meio da linguagem natural, o fluxograma apresenta uma notação gráfica para representação, e o pseudocódigo utiliza uma linguagem estruturada com regras bem definidas.

Aprendemos a armazenar os dados em variáveis, que são espaços na memória principal, esta pode ter seu conteúdo alterado ao longo do tempo. Vimos os principais tipos de dados: `inteiro, real, lógico e caractere`. Também tivemos contato com as regras para criar identificadores (nomes de variáveis, funções etc.).

Tivemos contato com operadores aritméticos, lógicos e relacionais, que podem ser utilizados para compor expressões. Na maioria das vezes, armazenamos os resultados de tais expressões em variáveis, utilizando o comando de atribuição: `<-`.

Por fim, descobrimos como utilizar funções intrínsecas para realizar operações pré-programadas. Dentre as mais importantes, destacam-se as funções matemáticas e as funções que permitem que o algoritmo interaja com o usuário por meio de entrada e saída de dados.

1. Construa um algoritmo, utilizando a descrição narrativa para pagar uma conta de luz, em um caixa eletrônico.

2. Para os nomes de variáveis a seguir, marque (C) para os corretos e (I) para os incorretos. Para cada nome incorreto, explique o que está errado.

( ) cidade	( ) media idade	( ) nome2
( ) endereço_nr	( ) a3	( ) 4cidade
( ) média_peso%	( ) endereço.cep	( ) cliente_nome
( ) aluno-nota	( ) B5	( ) 1234P

3. Formule um algoritmo para ler um número positivo qualquer e apresentar o quadrado e a raiz quadrada deste número. Dica: utilize as funções intrínsecas apresentadas.

4. Analise os algoritmos a seguir e escreva o resultado de sua execução.

```
Algoritmo exe5a
Var
   A, B, C: inteiro
Início
  A <- 5
  B <- 30
  C <- A
  B <- C
  A <- B
  Escreva(A)
  Escreva(B)
  Escreva(C)
Fim
```

```
Algoritmo exe5b
Var
   A, B, C: inteiro
Início
  A <- 5
  B <- 30
  C <- A
  A <- B
  B <- C
  Escreva(A)
  Escreva(B)
  Escreva(C)
Fim
```

COMO AGILIZAR SUA APRENDIZAGEM EM ALGORITMOS

Você está com dificuldade em aprender algoritmos? Acha realmente que não é para você? Se acalme! Em um primeiro momento, pode parecer, realmente, que algoritmos é um bicho de sete cabeças, todavia, nesta leitura complementar, debateremos maneiras eficazes de como aprender esta matéria!

Cada pessoa costuma aprender de formas distintas. Por exemplo, algumas pessoas aprendem melhor de forma passiva: pela leitura, por meio de áudios ou vídeos, demonstrações, aulas e palestras. Todavia, **independentemente** da pessoa, quando se trata do assunto algoritmos, é imprescindível que haja aprendizagem **ativa**: grupos de discussão, atividades práticas, construção de algoritmos e, em um estágio mais avançado, ajudar os colegas, ensinando-os a resolver problemas algorítmicos.

Algoritmos se aprende por meio da prática. É preciso resolver o máximo de problemas possível! Inicialmente, quando não se domina muito bem a sintaxe e a semântica do pseudocódigo e não se sabe bem como desenvolver uma lógica algorítmica, é preciso analisar exemplos de problemas já resolvidos.

Os problemas resolvidos são uma ferramenta muito importante para você começar a compreender melhor as regras "gramaticais" (sintáticas) que compõem o nosso pseudocódigo. Além disso, quando interpretamos o enunciado de um problema e, logo a seguir, temos contato com o algoritmo que resolve os requisitos daquele enunciado, é possível identificar quais conjuntos de instruções resolvem determinados problemas. Com isso, em problemas futuros, você poderá adaptar as lógicas já existentes à realidade de novos problemas.

Outra ferramenta fundamental é o **teste de mesa**. É imprescindível realizá-lo em todo e qualquer algoritmo com o qual você tiver contato. O teste de mesa é aquele momento em que você executa o algoritmo como se fosse um computador de verdade. Somente por meio do teste de mesa você pode verificar o resultado de um algoritmo do ponto de vista de usuário e, com isso, analisar a corretude da lógica desenvolvida. Ou seja, sempre que você tiver contato com um exercício resolvido, ou sempre que você desenvolver seu próprio algoritmo, **realize o teste de mesa**.

Por fim, e não menos importante, é interessante que você desenvolva seus próprios algoritmos para todo e qualquer problema proposto. Só assim você será capaz de treinar seu cérebro para criar um passo a passo lógico, um raciocínio algorítmico e resolver todo e qualquer problema que possua uma solução computacional. Sendo assim, mãos na massa, não perca tempo!

Fonte: os autores.

filme

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

Ano: 2005

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

filme

Lógica e Design de Programação

Ano: 2009

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

UNIDADE

ESTRUTURA CONDICIONAL

PROFESSORES
Dra. Gislaine Camila Lapasini Leal
Me. Pietro Martins de Oliveira

PLANO DE ESTUDO

A seguir, apresentam-se os tópicos que você estudará nesta unidade: • Estrutura condicional • Estrutura condicional simples • Estrutura condicional composta • Estrutura condicional aninhada • Estrutura de decisão múltipla • Exercícios resolvidos.

OBJETIVOS DE APRENDIZAGEM

Conhecer a estrutura condicional simples • Conhecer a estrutura condicional composta • Conhecer a estrutura condicional aninhada • Conhecer a estrutura de decisão múltipla • Elaborar algoritmos utilizando estrutura condicional • Resolver exercícios a fim de aprimorar os conhecimentos adquiridos.

INTRODUÇÃO

Nesta unidade, estudaremos a estrutura condicional, conhecendo a estrutura condicional simples, composta, aninhada e de decisão múltipla. Com os conhecimentos adquiridos, na Unidade 1, conseguimos construir algoritmos que, a partir da entrada, processavam os dados e apresentavam algumas informações na saída, isto é, algoritmos sequenciais.

Nos algoritmos sequenciais da Unidade 1, o fluxo era seguido de modo sequencial, ou seja, todas as instruções eram executadas uma a uma. Se pararmos para pensar, em diversas situações, temos que realizar uma verificação e analisar o resultado dela para saber que caminho seguir, ou seja, temos desvios de fluxo. Isso acontece quando compramos um carro, por exemplo, dependendo do valor da entrada, a taxa de juros pode ser maior ou menor. Quando temos que apresentar a média escolar de um aluno e tomar a decisão de aprová-lo, reprová-lo ou de permitir-lhe o exame. Assim, precisamos de uma estrutura que nos permita impor condições para a execução de determinada instrução, ou ainda, criar condições que possibilitem desviar o fluxo.

Estudaremos a estrutura condicional simples, que nos permite tomar uma decisão. A estrutura condicional composta, pela qual, a partir de uma expressão, podemos seguir dois caminhos: um, quando o resultado do teste é verdadeiro, e outro, quando o resultado é falso. A estrutura condicional aninhada ou encadeada, que nos permite estabelecer verificação de condições sucessivas. E a estrutura de decisão múltipla, uma generalização da estrutura condicional composta, em que pode haver uma ou mais condições a serem testadas e cada uma delas pode ter uma instrução diferente associada. Ao final desta unidade, poderemos formular algoritmos mais elaborados, com desvios de fluxo, o que nos possibilitará desenvolver soluções para uma gama maior de problemas.

AULA

1 ESTRUTURA CONDICIONAL

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

Desse modo, analisaremos adiante, a retirada de dinheiro em um caixa eletrônico. Após inserir o cartão, é solicitado que a senha seja digitada. Se ela estiver correta, poderemos efetuar o saque. Caso a senha esteja errada, receberemos mensagem informando que a senha é inválida. Notem que, nessa situação, não conseguimos representar apenas com o conhecimento adquirido na Unidade 1. Em situações como essa, precisamos utilizar uma estrutura que nos permita fazer verificações para, então, saber quais instruções devem ser executadas.

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

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

AULA 2

ESTRUTURA CONDICIONAL simples

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

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

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

Analisaremos, agora, o algoritmo apresentado no quadro. Consideremos o valor de `A` como 15, dessa forma, ao testar a condição dada pela expressão `A > 10`, retorna um valor `verdadeiro`. Com isso, temos a execução do comando `Escreva`, que está compreendido entre o `Se` e o `Fim_se`. Agora, tomemos `A` com valor 3. Ao testar a condição `A >10`, o valor retornado é `falso`. Deste modo, não é executada a instrução entre o `Se` e o `Fim_se`.

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

Quadro 1 - Pseudocódigo - Exemplo estrutura condicional simples / Fonte: os autores.

De acordo com Lopes e Garcia (2002), a condição é uma expressão lógica, portanto, ao ser testada, devolve como resposta o valor `verdadeiro` ou `falso`. Uma condição pode ser representada por uma expressão relacional ou por uma expressão lógica, formada por, pelo menos, duas expressões relacionais. Os operadores relacionais vistos na Unidade 1 são `>, <, =, >=, <= e <>`. Já os operadores lógicos são `E, OU e NÃO`.

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

```
X > 16

A < B

Sexo = "F"

Resposta <> "Sim"
```

Quando nossa condição é uma expressão lógica, temos, pelo menos, duas expressões relacionais que estão ligadas por um operador lógico. Você se recorda do funcionamento dos operadores lógicos? O operador `E` resulta em `verdadeiro` somente quando as duas condições são verdadeiras, como pode ser visto na tabela verdade, apresentada a seguir.

Operador E		
Condição 1	**Condição 2**	**Resultado**
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Falso
Falso	Verdadeiro	Falso
Falso	Falso	Falso

Quadro 2 – Tabela verdade: conjunção / Fonte: adaptado de Manzano e Oliveira (1997).

O operador OU resulta em verdadeiro quando, pelo menos, uma das condições é verdadeira. Sendo o resultado falso apenas quando as duas condições são falsas simultaneamente, como pode ser visualizado a seguir.

Operador OU		
Condição 1	**Condição 2**	**Resultado**
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Falso	Verdadeiro	Verdadeiro
Falso	Falso	Falso

Quadro 3 - Tabela verdade: disjunção / Fonte: adaptado de Manzano e Oliveira (1997).

O operador NÃO, por sua vez, funciona como a negação do resultado, ou seja, inverte o resultado lógico. Segundo Manzano e Oliveira (1997), esse operador é utilizado em situações em que se necessita estabelecer que uma dada condição deve não ser verdadeira ou deve não ser falsa. A tabela verdade para esse operador é apresentada no próximo quadro. Observe que o NÃO é um operador unário, ou seja, tem apenas uma condição (expressão relacional) associada a ele.

Operador NÃO	
Condição	**Resultado**
Verdadeiro	Falso
Falso	Verdadeiro

Quadro 4 - Tabela verdade: negação / Fonte: adaptado de Manzano e Oliveira (1997).

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

```
(X >= 1) E (X <=20)

(Sexo = "F") OU (Sexo = "f")

NÃO (X>5)
```

Note que as expressões lógicas são compostas, utilizando operadores relacionais e lógicos. Agora que conhecemos a sintaxe da estrutura condicional simples e sabemos como montar condições, formularemos nosso primeiro algoritmo, contendo desvio de fluxo. O problema consiste em identificar se um número inteiro é um número par e, então, imprimir a metade do número. Retomando nossos passos para a construção de algoritmos, temos:

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

O que é um número par? É um número inteiro múltiplo de 2, ou seja, um número cuja divisão por 2 resulte em resto igual a 0. A entrada de dados consiste em obter um número inteiro, o qual denominaremos de `N`. O processamento consiste em encontrar o resto da divisão deste número por 2 e verificar se é igual a 0. Como faremos isso? Você se recorda de alguma função que o faz? Na Unidade 1, vimos o operador `MOD`, que retorna o resto da divisão de dois números inteiros.

Se o resto for igual a 0, calcularemos a metade desse número. E a saída consiste em imprimir a metade do número. Se o resto for diferente de 0, não será executada nenhuma instrução e também não haverá saída. Uma solução para esse problema é apresentada no quadro seguinte, em que podemos visualizar o uso da estrutura condicional simples.

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

Quadro 5 - Pseudocódigo da primeira versão do algoritmo par / Fonte: os autores.

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

Em relação ao algoritmo apresentado, podemos colocar uma instrução `Escreva`, após o `Fim_se`, dizendo que o número é ímpar? Não, pois, para qualquer número obtido na entrada, indiferentemente de ser par ou ímpar, a mensagem seria impressa. Devemos lembrar que após o `Fim_se`, o fluxo do algoritmo segue, normalmente, executando a instrução.

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

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

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

Quadro 6 - Pseudocódigo - Algoritmo par (versão 2) / Fonte: os autores.

AULA

3 ESTRUTURA CONDICIONAL composta

Na estrutura condicional composta, realiza-se a avaliação de uma única expressão lógica. Se o resultado dessa avaliação for `verdadeiro`, é executada a instrução ou o conjunto de instruções compreendido entre o comando `Se` e o `Senão`. Por outro lado, se o resultado da avaliação for `falso`, é executada a instrução ou o conjunto de instruções entre o `Senão` e o `Fim_se` (MANZANO; OLIVEIRA, 1997).

A sintaxe da estrutura condicional composta é:

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

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

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

Quadro 7 - Pseudocódigo - Algoritmo parimpar / Fonte: os autores.

Se o resultado do teste da expressão relacional `n MOD 2 = 0` for `verdadeiro`, executa-se a instrução que se encontra entre o `Se` e o `Fim_se`, ou seja, `Escreva("o número é par")`. Caso o resultado do teste seja `falso`, executa-se a instrução que se encontra entre o `Senão` e o `Fim_se`, `Escreva("O número é ímpar")`. Por exemplo, se o número obtido na entrada for 5, temos que `5 MOD 2` é igual a 1, ou seja, o teste da expressão resulta em `falso`, logo, será impresso que "o número é ímpar".

Na estrutura condicional composta, utiliza-se expressões relacionais e expressões lógicas, tal como na estrutura condicional simples. Antes de conhecer outros tipos de estrutura condicional, praticaremos a construção de algoritmos, utilizando expressões lógicas. O problema consiste em: dado um número inteiro, verificar se ele compreende entre 20 e 90.

No processo de resolução, seguiremos esses passos:

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

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

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

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

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

Quadro 8 - Pseudocódigo – Algoritmo `faixa` / Fonte: os autores.

Considerando `n` com valor 27, temos uma expressão lógica composta por duas expressões relacionais (condições) unidas pelo operador `E`. Você se recorda como este operador funciona? Dadas duas condições, o resultado é `verdadeiro` se, e somente se, as duas expressões lógico-relacionais forem verdadeiras. O próximo quadro representa a expressão lógica da estrutura condicional, em que `27 > 20`

é `verdadeiro` e `27 < 90` também é `verdadeiro`, o que resulta em `verdadeiro`. Como o resultado do teste lógico é verdadeiro, temos a execução do fluxo compreendido entre o `Se` e o `Senão`, ou seja, a mensagem "o número está na faixa entre 20 e 90".

Operador `E`		
Condição 1	**Condição 2**	**Resultado**
`27 > 20`	`27 < 90`	`Verdadeiro`

Quadro 9 - Resultado das comparações primeiro teste de mesa no algoritmo `faixa`
Fonte: os autores.

E se, na entrada, recebêssemos o número 20? Analisaremos, agora, cada uma das expressões relacionais. Temos `20 > 20`, é uma expressão que resulta em `verdadeiro` ou `falso`? É `falso`, pois 20 é igual a 20 e não maior. Na segunda expressão, temos `20 < 90`, que resulta em `verdadeiro`. Essas expressões estão unidas pelo operador `E`, sendo uma delas falsa, e a outra, verdadeira, assim, o resultado do teste lógico é `falso`. Com isto, temos a execução da instrução, compreendida entre o `Senão` e o `Fim_se`, "o número está fora da faixa".

Operador `E`		
Condição 1	**Condição 2**	**Resultado**
`27 > 20`	`27 < 90`	`Falso`

Quadro 10 - Resultado das comparações primeiro teste de mesa no algoritmo `faixa`
Fonte: os autores.

conecte-se

Para facilitar a leitura de um algoritmo, normalmente, adiciona-se espaços em branco no início de um bloco de comandos para mostrar quais instruções estão dentro dele, a este recurso denominamos indentação. Para saber um pouco mais, leia o conteúdo disponível em: http://pt.wikipedia.org/wiki/Indenta%C3%A7%C3%A3o

AULA

4 ESTRUTURA CONDICIONAL aninhada

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

Lopes e Garcia (2002) destacam que essa estrutura é utilizada quando sentimos a necessidade de tomar decisões dentro de uma das alternativas de uma condição. Visualizar a estrutura condicional aninhada em um problema que consiste em encontrar o maior número dentre três números. Seguindo o método de estruturação do problema, visto na Unidade 1, temos que:

- **Objetivo do algoritmo**: encontrar o maior número.
- **Entrada**: obter três números inteiros.
- **Processamento**: comparar os números e armazenar o valor do maior.
- **Saída**: imprimir o maior número.

A entrada de dados consiste em ler três números inteiros, os quais armazenaremos em variáveis denominadas `a`, `b` e `c`. A fim de encontrar o maior número,

precisamos realizar comparações, para isso, utilizaremos expressões relacionais do tipo: `A > B` e armazenaremos o valor do maior número em uma variável, a qual chamaremos de `max`. A saída consiste em enviar uma mensagem contendo o valor do maior número, que está armazenado na variável `max`. O algoritmo para o problema é apresentado no próximo quadro.

```
Algoritmo maior
Var
     a, b, c, max: inteiro
Início
    Escreva("Digite o primeiro número inteiro:")
    Leia(a)
    Escreva("Digite o segundo número inteiro:")
    Leia(b)
    Escreva("Digite o terceiro número inteiro:")
    Leia(c)
    Se (a > b) então
         Se (a > c) então
           max <-  a
         Senão
           max <- c
         Fim_se
    Senão
        Se (b > c) então
             max <- b
        Senão
```

```
            max <- c
        Fim_se
    Fim_se
    Escreva("O maior número é: ", max)
Fim
```

Quadro 11 - Pseudocódigo - Algoritmo `maior` / Fonte: os autores.

Se você não entendeu o funcionamento do algoritmo, fique tranquilo, posteriormente, veremos, passo a passo, a estrutura condicional do problema em questão. Na figura a seguir, temos a representação da estrutura condicional aninhada, os colchetes mais à esquerda (externos) representam a estrutura condicional composta, em que o trecho 1 é executado quando o resultado da expressão relacional `a > b` é `verdadeiro`. O trecho 2 apresenta o conjunto de instruções, executado quando o resultado da expressão relacional é igual a `falso`. Note que tanto dentro do trecho 1 quanto do trecho 2, temos outra estrutura condicional composta, representada pelos trechos 3 e 4. A estrutura condicional do trecho 3 só é executada se `a > b` resultar em `verdadeiro`. Em seguida, verifica-se a > c, em caso verdadeiro, executa-se a instrução, em que se atribui o valor de `a` para a variável `max`. Se `a > c` for avaliado como `falso`, executa-se a instrução em que o valor de `c` é atribuído para a variável `max`.

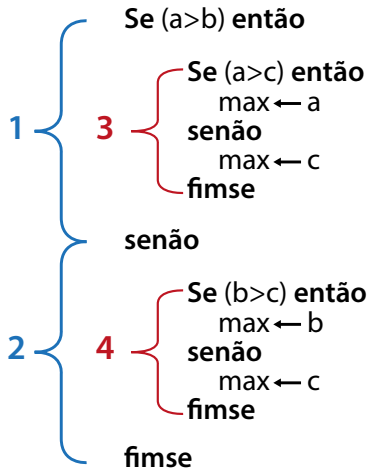


Figura 1 - Estrutura condicional aninhada / Fonte: os autores.

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

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

Quadro 12 - Pseudocódigo - Algoritmo `maior` / Fonte: os autores.

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

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

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

```
Caso <variável>
     Seja <valor 1> faça <instruções 1>
     Seja <valor 2> faça <instruções 2>
     ...
     Seja <valor N> faça <instruções N>
     Senão <instruções padrão>
Fim_caso
```

Nesta estrutura, o campo `variável` indica a variável a ser controlada, o campo `valor` indica o conteúdo da variável analisado e os campos `instruções` referem-se às instruções que serão executadas caso a `variável` seja igual ao respectivo `valor` associado àquele caso. O funcionamento dessa estrutura consiste em, ao entrar-se em uma construção do tipo Caso, a comparação "`variável = valor 1`" é testada: se for verdadeira, o bloco de comandos "`instruções 1`" é executado e, após seu término, o fluxo de execução prossegue pela primeira instrução após o final da construção `(Fim_caso)`; se "`variável = valor 1`" for falsa, então, a comparação "`variável = valor 2`" é testada: se esta for verdadeira, o bloco "instruções 2" é executado e, ao seu término, a execução prossegue, normalmente, pela instrução seguinte, ao final da construção `(Fim_caso)`. De modo análogo, ocorre para os demais casos da estrutura. Lopes e Garcia (2002) destacam que esse tipo de estrutura é bastante utilizado na construção de menus, tornando-os mais claros.

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

- **Objetivo do algoritmo**: escrever o mês equivalente ao número digitado.
  Entrada: obter um número inteiro.
- **Processamento**: verificar se o número digitado está entre 1 e 12.
- **Saída**: imprimir o mês equivalente por extenso.

Na entrada de dados, temos um número inteiro, que nomearemos com variável `num`. O processamento consiste em verificar se o número digitado está entre 1 e

12 e, como saída, escrever por extenso o mês correspondente. No quadro posterior, temos o pseudocódigo para o problema em questão.

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

Quadro 13 - Pseudocódigo – Algoritmo `mes` / Fonte: os autores.

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

Ainda, analisando o algoritmo que utiliza a estrutura de decisão múltipla, você acha que podemos escrevê-lo de outra forma? Como ficaria o algoritmo se utilizássemos apenas a estrutura condicional simples? Há, pelo menos, mais duas formas de resolvê-lo, utilizando a estrutura condicional simples ou a estrutura encadeada. A construção do algoritmo utilizando apenas a estrutura condicional simples pode ser visualizada a seguir.

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

```
Se (num = 3) então
      Escreva("Março")
    Fim_se
    Se (num = 4) então
      Escreva("Abril")
    Fim_se
    Se (num = 5) então
      Escreva("Maio")
    Fim_se
    Se (num = 6) então
      Escreva("Junho")
    Fim_se
    Se (num = 7) então
      Escreva("Julho")
    Fim_se
    Se (num = 8) então
      Escreva("Agosto")
    Fim_se
    Se (num = 9) então
      Escreva("Setembro")
    Fim_se
    Se (num = 10) então
      Escreva("Outubro")
Fim_se
```

```
Se (num = 11) então
      Escreva("Novembro")
    Fim_se
    Se (num = 12) então
      Escreva("Dezembro")
    Fim_se
    Se (num <> 1) E (num <> 2) E (num <> 3) E  (num <>
    4) E
    (num <> 5) E (num <> 6) E (num <> 7) E (num <>
    8) E
    (num <> 9) E (num <> 10) E (num <> 11) E
    (num <> 12)
    então
       Escreva("Mês inválido")
    Fim_se
Fim
```

Quadro 14 - Pseudocódigo – Algoritmo `mes2` / Fonte: os autores.

Os algoritmos apresentados nos quadros, anteriormente, são soluções para o problema. Você sabe qual a diferença entre eles? Supondo que o número digitado tenha sido 3. No primeiro algoritmo, acontece a sequência de verificações "`num = 1`", "`num = 2`" e "`num = 3`". Como "`num = 3`" é `verdadeiro`, ocorre a execução da instrução Escreva ("Março") e o fluxo de execução salta para o `Fim_caso`, isto é, os demais casos não são testados.

No segundo algoritmo, temos que todas as instruções serão verificadas. Dessa forma, a diferença entre os dois algoritmos está no desempenho, no número de instruções que será executada, e isso impacta no tempo de execução dos nossos programas. Portanto, é importante que você escreva algoritmos eficientes.

Você já se perguntou como fazer um algoritmo eficiente? Ele utiliza apenas

o número necessário de variáveis, já que cada variável consiste em um espaço de memória reservado. Além disso, podemos associar a eficiência com o número de instruções que o algoritmo executa para resolver um problema. Com isso, concluímos que não devemos declarar variáveis que não serão utilizadas e dar preferência a estruturas condicionais aninhadas ou de escolha múltipla, ao invés de construir várias de estruturas simples.

AULA 6

EXERCÍCIOS RESOLVIDOS

Agora, para sedimentar nossos conhecimentos, teremos contato com problemas que necessitam de tomada de decisão para serem resolvidos. Dessa forma, os algoritmos propostos a seguir, deverão utilizar estruturas de decisão simples, composta ou múltipla.

pensando juntos

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

Problema 1

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

- **Objetivo do algoritmo**: informar se a pessoa é menor de idade, maior ou idoso.
- **Entrada**: ler nome e idade.
- **Processamento**: verificar se a idade é menor que 18 (menor de idade), entre 18 e 64 anos (maior de idade) ou maior ou igual a 65 anos (idoso).
- **Saída**: escrever se a pessoa é menor de idade, maior ou idoso.

```
Algoritmo verificaidade
Var
     idade: inteiro
     nome: vetor[1..30] de caractere
Início
     Escreva("Digite o nome:")
     Leia(nome)
     Escreva("Digite a idade:")
     Leia(idade)
     Se (idade > = 65) então
        Escreva("Idoso")
```

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

Quadro 15 - Pseudocódigo – Problema 1 / Fonte: os autores.

Problema 2

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

- **Objetivo do algoritmo**: verificar a classe eleitoral de uma pessoa.
- **Entrada**: ler idade.
- **Processamento**: verificar se a idade é menor que 16 (não eleitor), entre 18 e 65 anos (eleitor obrigatório) ou entre 16 e 18 ou maior que 65 anos (eleitor facultativo).
- **Saída**: escrever a classe eleitoral.

```
Algoritmo classeeleitoral
Var
     idade: inteiro
Início
     Escreva("Digite a idade:")
     Leia(idade)
```

```
Se (idade < 16) então
        Escreva("Não eleitor")
     Senão
        Se (idade > 65) ou (idade < 18) então
           Escreva("Eleitor facultativo")
        Senão
           Escreva("Eleitor obrigatório")
        Fim_se
     Fim_se
Fim
```

Quadro 16 - Pseudocódigo – Problema 2 / Fonte: os autores.

Problema 3

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

- **Objetivo do algoritmo**: verificar a faixa de risco de uma pessoa a partir do IMC.
- **Entrada**: ler peso e altura.
- **Processamento**: verificar se o IMC é menor que 20 (abaixo do peso), entre 20 e 25 (normal), entre 26 e 30 (excesso de peso), entre 31 e 35 (obesidade) ou acima de 35 (obesidade mórbida).
- **Saída**: imprimir a faixa de risco da pessoa.

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

```
                Fim_se
            Fim_se
        Fim_se

Fim
```

Quadro 17 - Pseudocódigo – Problema 3 / Fonte: os autores.

Problema 4

Elabore um algoritmo que receba o salário de um funcionário e o código do cargo e apresente o cargo, o valor do aumento e o novo salário. A tabela a seguir apresenta os cargos.

Código	Cargo	Percentual do aumento
1	Serviços gerais	50%
2	Vigia	30%
3	Recepcionista	25%
4	Vendedor	15%

Tabela 1 - Apresentação de dados / Fonte: os autores.

- **Objetivo do algoritmo**: calcular o aumento de salário, de acordo com o cargo.
- **Entrada**: ler salário e código do cargo.
- **Processamento**: calcular o aumento salarial de acordo com o cargo da pessoa.
- **Saída**: imprimir o nome do cargo, o valor do aumento e o novo salário.

```
Algoritmo reajuste
Var
      cargo: inteiro
      salario, reajuste, aumento: real
Início
     Escreva("Digite o cargo:")
     Leia(cargo)
     Escreva("Digite o salário:")
     Leia(salario)
     Se (cargo = 1) então
        Escreva("Serviços gerais")
        aumento <- salario * (50/100)
        reajuste <- salario + aumento
        Escreva("O aumento é de:", aumento)
        Escreva("O novo salário é:", reajuste)
     Senão
        Se (cargo = 2) então
           Escreva("Vigia")
           aumento <- salario * (30/100)
           reajuste <- salario + aumento
           Escreva("O aumento é de:", aumento)
           Escreva("O novo salário é:", reajuste)
        Senão
           Se (cargo = 3) então
```

```
                Escreva("Recepcionista")
                aumento <- salario * (25/100)
                reajuste <- salario + aumento
                Escreva("O aumento é de:", aumento)
                Escreva("O novo salário é:",
                reajuste)
            Senão
                Se (cargo = 4) então
                    Escreva("Vendedor")
                    aumento <- salario * (15/100)
                    reajuste <- salario + aumento
                    Escreva("O aumento é de:",
                    aumento)
                    Escreva("O novo salário é:",
                    reajuste)
                Fim_se
            Fim_se
        Fim_se
    Fim_se
Fim
```

Quadro 18 - Pseudocódigo – Problema 4 / Fonte: os autores.

Problema 5

Escreva um algoritmo para resolver equações do segundo grau $\left(ax^2+bx+c\right)$. Sendo que: a variável a deve ser diferente de zero; $\Delta = b^2 - 4 \times a \times c$; Se $\Delta < 0$, então, não existe raiz real; se $\Delta = 0$, então, existe uma raiz real, que é dada por $x = \frac{(-b)}{2 \times a}$; se $\Delta > 0$, existem duas raízes reais $x1 = \frac{\left(-b+\sqrt{\Delta}\right)}{2 \times a}$ e $x1 = \frac{\left(-b-\sqrt{\Delta}\right)}{2 \times a}$.

- **Objetivo do algoritmo**: resolver a equação do segundo grau.
- **Entrada**: ler os valores de a, b e c.
- **Processamento**: calcular o delta (Δ) e o valor das raízes.
- **Saída**: imprimir o valor da raiz real.

```
Algoritmo raiz
Var
      a, b, c, delta, x1, x2: real
Início
   Escreva("Digite o valor de a:")
   Leia(a)
   Escreva("Digite o valor de b:")
   Leia(b)
   Escreva("Digite o valor de c:")
   Leia(c)
   Se (a = 0) então
     Escreva("Não é uma equação do segundo grau")
   Senão
      delta <- sqr(b) - 4 * a * c
      Se (delta < 0) então
```

```
            Escreva("Não existe raiz real")
        Senão
            Se (delta = 0) então
                Escreva("Existe uma raiz real")
                x1 <- (-b) / (2 * a)
                Escreva("A raiz é:", x1)
            Senão
                Se (delta > 0) então
                 Escreva("Existem duas raízes reais")
                   x1 <- (-b + sqrt(delta)) / (2 * a)
                    Escreva("A raiz x1 é:", x1)
                   x2 <- (-b - sqrt(delta)) / (2 * a)
                    Escreva("A raiz x2 é:", x2)
                Fim_se
            Fim_se
        Fim_se
    Fim_se
Fim
```

Quadro 19 - Pseudocódigo – Problema 5 / Fonte: os autores.

CONSIDERAÇÕES FINAIS

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

Estudamos quatro formas de estrutura condicional: estrutura condicional simples, estrutura condicional composta, estrutura condicional encadeada (ou aninhada) e estrutura de decisão múltipla. Na simples, vimos que as instruções só serão executadas se a condição avaliada for verdadeira.

Na condicional composta, vimos que é realizada a avaliação de uma única condição. No entanto temos dois caminhos para seguir, um quando o resultado da instrução é `verdadeiro`, e outro quando é `falso`. Aprendemos que a estrutura aninhada é usada quando temos que estabelecer verificações sucessivas, isto é, quando uma ação só poderá ser executada se um conjunto anterior de condições for satisfeito. Além disso, estudamos que o uso desta estrutura pode tornar o algoritmo mais rápido devido ao fato de executar um menor número de passos para chegar à solução do problema.

Vimos, também, a estrutura de decisão múltipla, uma generalização da estrutura `Se`, em que pode haver uma ou mais condições a serem testadas e um comando associado a cada uma delas. Além da estrutura condicional, trabalhamos a construção de expressões relacionais e lógicas, as quais são utilizadas na construção de condições. Entendemos, também, a tabela verdade dos operadores lógicos `e`, `ou` e `não`.

Ao longo desta unidade, construímos algoritmos, utilizando todos os conceitos aprendidos. Também discutimos a estrutura condicional mais adequada para cada situação.

1. Formule um algoritmo que leia a matrícula e o nome do vendedor, o seu salário fixo e o total de vendas e calcule a comissão do vendedor. Se o total de vendas é inferior a R$ 1500,00, o percentual de comissão é 2%, e se for maior, o percentual é de 4%. Apresente o nome do vendedor, a matrícula, o salário fixo e o salário total.

2. Escreva um algoritmo que leia um número e informe se ele é divisível por 3 e por 7, simultaneamente.

3. Formule um algoritmo que leia cinco números e conte quantos deles são negativos.

4. De acordo com uma tabela médica, o peso ideal está relacionado com a altura e o sexo. Elabore um algoritmo que receba altura e sexo de uma pessoa, calcule e imprima o seu peso ideal, sabendo que:

Para homens	(72.7 x altura) -58
Para mulheres	(62.1 x altura) -44.7

5. Elabore um algoritmo que leia o percurso em quilômetros, o tipo de moto e informe o consumo estimado de combustível, sabendo que uma moto do tipo A faz 26 km com um litro de gasolina, uma moto do tipo B faz 20 km, e o tipo C faz 7 km.

6. Uma instituição financeira concederá crédito a uma taxa de juros de 3% aos seus clientes, de acordo com o saldo médio do período. Elabore um algoritmo que calcule o valor que pode ser concedido ao cliente e imprima-o. Os clientes com saldo médio inferior a R$ 500,00 não têm direito a crédito. Já os clientes com saldo que vai de R$ 500,00 até R$ 1000,00 podem obter créditos de 35% em relação ao saldo médio. Clientes com saldo maior que R$ 1000,00 e que vá até R$ 3000,00 podem obter créditos de 50% em relação ao saldo médio. E, para aqueles clientes com saldo superior a R$ 3000,00, pode ser concedido crédito de 75% do valor do saldo.

Algoritmos estão, a todo tempo, tomando decisões que afetam nosso dia a dia. Sendo assim, acompanhe este trecho de um artigo da BBC que ilustra casos reais nos quais os algoritmos são legítimos tomadores de decisão.

NOVE ALGORITMOS QUE PODEM ESTAR TOMANDO DECISÕES SOBRE SUA VIDA - SEM VOCÊ SABER

[...].

1. O computador decide se você fará ou não uma entrevista de emprego

Currículos são cada vez mais descartados sem sequer passar por mãos humanas. Isso porque as empresas estão empregando sistemas automatizados em seus processos seletivos, principalmente na análise de centenas de milhares de inscrições. Seu próximo empréstimo provavelmente será aprovado (ou recusado) por um algoritmo.

Nos Estados Unidos, estima-se que mais de 70% dos candidatos sejam eliminados antes de serem avaliados por pessoas. Para as companhias, isso economiza dinheiro e tempo, mas alguns questionam a neutralidade dos algoritmos. [...].

2. Quer dinheiro emprestado? Seu perfil na rede social pode afetar isso

Historicamente, quando alguém pede dinheiro a uma instituição financeira, a resposta vai depender da análise das chances do empréstimo ser pago, com base na proporção entre a dívida e a renda desta pessoa e seu histórico de crédito.

Não é mais assim: agora, algoritmos reúnem e analisam dados de múltiplas fontes, que vão desde padrões de compra a buscas na internet e sua atividade em redes sociais.

O problema é que esse método usa informações coletadas sem o conhecimento ou colaboração de quem pede o dinheiro. Também há uma questão em torno da transparência do código do algoritmo e seu comportamento tendencioso.

3.Um algoritmo pode te ajudar a achar um amor, mas pode não ser quem você espera

Não é uma surpresa que sites de namoro usam algoritmos para identificar duas pessoas compatíveis. É um dos seus principais apelos para o público, na verdade.

[...].

[...] Até mesmo em alternativas como o aplicativo Tinder, em que as variáveis são bem menos complexas (geografia, idade e orientação sexual), as combinações não são tão simples assim. Quem usa o serviço recebe uma nota secreta sobre o quanto essa pessoa é "desejável", calculada para "permitir melhores combinações", segundo o Tinder. A fórmula é mantida em segredo, mas os executivos da empresa por trás do aplicativo já indicaram que o número de vezes que o perfil de alguém é curtido ou rejeitado tem um papel crucial sobre isso.

4.Um programa pode determinar se você é viciado em drogas e se conseguirá contratar um plano de saúde

O mal uso de medicamentos e drogas é a principal causa de mortes acidentais nos Estados Unidos, e especialistas com frequência se referem a esse problema como uma epidemia.

Para lidar com isso, cientistas e autoridades estão se unindo em projetos baseados em dados. Existe aí uma questão ética: os dados analisados incluem o histórico médico e até mesmo o endereço residencial. O argumento a favor desse tipo de intervenção é que isso pode salvar vidas e mitigar prejuízos ao sistema de saúde - viciados em opioides têm, por exemplo, 59% mais chances de serem usuários de alto custo.

[...].

5. Eles determinam até se um filme será feito

Essa não é a primeira vez que alguém dirá que Hollywood tem uma fórmula para produzir sucessos. Mas é diferente do processo baseado na experiência e instinto de produtores ao selecionar um roteiro ou elenco.

Algoritmos são usados para analisar não só as chances de um filme ir bem nas bilheterias, mas também quanto dinheiro ele fará.

[...].

Além de comparar um novo filme com uma base de dados de produções passadas, esses serviços afirmam que podem detectar o impacto de mudanças na história e até mesmo entre os atores.

6. Algoritmos influenciam em quem você vota e quem será presidente

Em uma época em que dados tornaram-se mais importantes do que empatia e carisma no mundo da política, algoritmos são cruciais para candidatos em busca de votos.

[...].

7. A polícia usa algoritmos para prever se você será um criminoso

O sistema de vigilância da China sobre seus 1,3 bilhão de habitantes é bem conhecido, mas parece haver espaço para expandi-lo. Grupos de direitos humanos acusaram as autoridades chinesas de violar a privacidade dos cidadãos, dizendo que o real propósito do sistema é monitorar dissidentes.

[...].

8. Um computador pode te mandar para a prisão

Juízes em ao menos dez Estados americanos estão tomando decisões em casos criminais com a ajuda de um sistema automatizado chamado COMPAS, baseado em um algoritmo de análise de risco para prever a probabilidade de uma pessoa cometer um novo crime.

[...].

9. Eles podem influenciar seu dinheiro

Esqueça as imagens de pessoas gritando na bolsa de valores com telefones nos ouvidos. Transações no mercado de ações estão se tornando, cada vez mais, um produto de cálculos feitos por algoritmos, que são mais rápidos do que qualquer humano e compram e vendem papéis em questão de segundos.

[...].

Fonte: Duarte (2018, on-line)[1].

livro

Algoritmos e Estruturas de Dados

Autores: Newton Alberto de Castilho Lages e Ângelo de Moura Guimarães

Editora: LTC

Sinopse: os princípios da lógica de programação para computadores são apresentados por meio de uma linguagem de programação didática denominada Portugol, de modo a enfatizar a metodologia de criação de um algoritmo, independentemente da linguagem de programação utilizada pelo futuro programador.

UNIDADE

5 ESTRUTURAS DE REPETIÇÃO

PROFESSORES
Dra. Gislaine Camila Lapasini Leal
Me. Pietro Martins de Oliveira

PLANO DE ESTUDO

A seguir, apresentam-se as aulas que você estudará nesta unidade: • Estrutura de repetição • Estrutura `Para` • Estrutura `Enquanto` • Estrutura `Repita` • Estruturas de repetição encadeadas • Exercícios resolvidos.

OBJETIVOS DE APRENDIZAGEM

Compreender as estruturas de repetição • Estudar as estruturas de repetição controladas e condicionais • Conhecer as estruturas de repetição encadeadas • Construir algoritmos utilizando estruturas de repetição • Estudar os exercícios resolvidos para melhor entendimento.

INTRODUÇÃO

Nesta unidade, você aprenderá a construir algoritmos com repetição de um trecho de código. A repetição permite executar um conjunto de instruções quantas vezes forem necessárias, sem precisar reescrever trechos de códigos idênticos.

Abordaremos, em seguida, o conceito de estrutura de repetição e sua aplicação. Estudaremos as estruturas de repetição com laços contados e laços condicionais. O uso de laços contados é indicado às situações em que sabemos, previamente, quantas vezes as instruções precisam ser executadas. Nesse caso, abordaremos, especificamente, a Estrutura Para.

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

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

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

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

AULA

1 ESTRUTURA DE REPETIÇÃO

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

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

Nas estruturas de repetição, o número de repetições pode ser fixo ou relacionar-se a uma condição. Isto é, os laços de repetição podem ser classificados em laços contados e laços condicionais (ASCENCIO; CAMPOS, 2010). Os laços contados são aqueles que utilizamos quando sabemos, previamente, quantas vezes um trecho de código precisa ser repetido. Por exemplo, realizar a leitura de 100 números, efetuar o somatório dos números entre 500 e 700 e outros. A estrutura utilizada para representar os laços contados é a Estrutura `Para`.

Os laços condicionais são utilizados quando não conhecemos o número de vezes que o trecho de código precisa ser repetido. A repetição atrela-se a uma condição que pode ser alterada dentro do laço. Um exemplo dessa situação seria solicitar que o usuário informe um número até que ele digite um número entre 1 e 12. Com isso, podemos efetuar a validação dos dados de entrada.

Você se recorda do algoritmo que criamos na Unidade 2 para escrever por extenso o nome do mês correspondente ao número digitado pelo usuário? Naquele caso, se o usuário informasse um número fora da faixa, entre 1 e 12, emitiríamos uma mensagem informando que não havia mês correspondente ao número digitado e a execução do algoritmo era encerrada. Agora, utilizando laços, podemos "forçar" a digitação de um dado de entrada que seja válido, ou seja, enquanto o usuário não digitar um número dentro da faixa definida, continuamos solicitando que ele informe um novo número, até que ele faça o que o algoritmo pede.

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

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

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

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

AULA

2 ESTRUTURA PARA

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

```
Para <var> de <início> até <fim> passo <incremento>
faça

     <instruções>

Fim_para
```

Em que:

- `<var>`: é a variável contadora utilizada para controlar a estrutura de repetição. Esta variável, geralmente, é do tipo `inteiro`, mas também pode ser do tipo `real`.
- `<início>` e `<fim>`: esses termos delimitam o intervalo para a execução do laço de repetição. Podem ser constantes inteiras, funções ou expres-

sões que retornem números inteiros (SALVETTI; BARBOSA, 1998). Isso significa que a variável indicada por <var> iniciará com um valor igual a <início> e será incrementada até atingir o valor <fim>.

- <incremento> representa o valor que será incrementado ou decrementado (se for um valor negativo) a cada passagem do laço, isto é, como será a magnitude da variação da variável de controle <var> a cada iteração do laço. Esse termo pode ser representado por uma constante ou uma variável.

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

Agora que você conheceu a teoria da estrutura de repetição Para, resolveremos um problema, utilizando-a para tornar mais clara a sua aplicação prática. Você se lembra de tabuada? A figura seguinte apresenta a tabuada do número 5, em que temos o produto entre o número 5 e os números inteiros compreendidos entre 0 e 10.

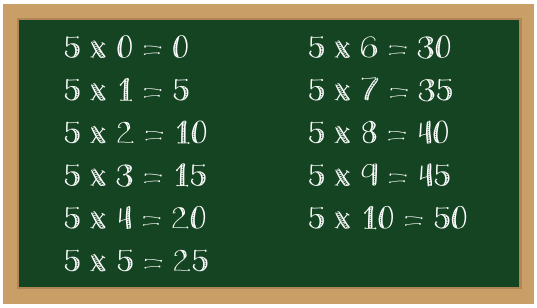


Figura 1 - Tabuada do número 5 / Fonte: os autores.

Que tal construir um algoritmo para efetuar a tabuada de um número qualquer? O primeiro passo é estruturar o nosso problema, seguindo os passos descritos na Unidade 1:

- **Objetivo do algoritmo**: calcular a tabuada de um número inteiro.
- **Entrada**: obter um número inteiro.
- **Processamento**: efetuar as operações de multiplicação entre o número informado pelos valores inteiros, compreendidos entre 1 e 10.
- **Saída**: imprimir a tabuada de 1 a 10 do número informado na entrada.

Na entrada de dados, temos que ler um número inteiro, isso implica declarar uma variável do tipo `inteiro` para armazenar o número digitado pelo usuário. Denominaremos essa variável de `num`. O processamento consiste em multiplicar o número recebido na entrada (armazenado na variável `num`) pelos valores 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 e 10, como pode ser visualizado na próxima figura. Criaremos uma variável chamada `mult` para armazenar o resultado da multiplicação. Observe que temos a repetição de uma expressão aritmética de multiplicação (`num` x `i`) em que sabemos, previamente, o número de repetições. Portanto, podemos utilizar a estrutura `Para`. Lembre-se que, ao utilizar essa estrutura, precisamos declarar uma variável contadora que, nesse caso, deve ser do tipo `inteiro`, a qual nomearemos de `i`. A variável `i` deve ter início em 0 e fim em 10, pois queremos mostrar a tabuada de 0 a 10. O `passo` a ser utilizado é 1. Como saída do algoritmo, temos que imprimir o resultado da operação de multiplicação.

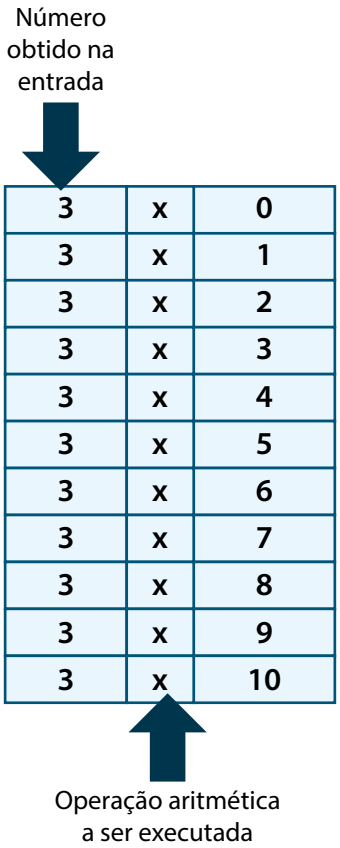


Figura 2 - Processamento a ser executado / Fonte: os autores.

O quadro a seguir apresenta o algoritmo para o problema da tabuada. Observe que o processamento (expressão aritmética dada por `mult <- num * i`) e a saída de dados `Escreva(num, "x", i, "=", mult)` se encontram dentro do laço de repetição. Por que isso acontece? Porque temos que imprimir o resultado de várias operações de multiplicação e não apenas uma. O que acontece se colocarmos a saída de dados fora do laço? Na tela, apresenta-se apenas a mensagem, contendo o resultado da última operação, em que `num` é igual a 3, `i` igual a 10. Portanto, será exibida a mensagem "3 x 10 = 30".

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

```
Algoritmo tabuada
Var
     num, i, mult: inteiro
Início
    Escreva("Digite um número:")
    Leia(num)
    Para i de 0 até 10 passo 1 faça
       mult <-  num * i
       Escreva(num, "x", i, "=", mult)
   Fim_para
 Fim
```

Quadro 1 - Pseudocódigo - Algoritmo `tabuada` / Fonte: os autores.

A figura a seguir ilustra a simulação do algoritmo apresentado no quadro anterior. Observe.

```
Algoritmo tabuada

Var

  num, i, mult: inteiro

Início

  Escreva "Digite um número"

  Leia (num)        3

Para i de 0 até 10 passo 1 faça

   mult ← num*i

   Escreva num, "x", i, "=", mult

   fimpara

Fim
```

```
i = 0
  mult = 3 * 0
  Escreva 3 x 0 = 0
i = 1
  mult = 3 * 1
  Escreva 3 x 1 = 3
i = 2
  mult = 3 * 2
  Escreva 3 x 2 = 6
i = 0
  mult = 3 * 3
  Escreva 3 x 3 = 9
i = 0
  mult = 3 * 4
  Escreva 3 x 4 = 12
i = 0
  mult = 3 * 5
  Escreva 3 x 5 = 15
i = 0
  mult = 3 * 6
  Escreva 3 x 6 = 18
i = 0
  mult = 3 * 7
  Escreva 3 x 7 = 21
i = 0
  mult = 3 * 8
  Escreva 3 x 8 = 24
i = 0
  mult = 3 * 9
  Escreva 3 x 9 = 27
i = 0
  mult = 3 * 10
  Escreva 3 x 10 = 30
```

Figura 3 - Simulação - Algoritmo tabuada / Fonte: os autores.

Será que não existem formas de deixar um algoritmo mais eficiente? Podemos melhorar o algoritmo construído para a tabuada? A resposta é: sim. Podemos economizar uma variável, no caso, a variável `mult`, e com isso, retirar a instrução de atribuição, realizando a operação aritmética diretamente no comando `Escreva`. Isso é possível, pois, no comando `Escreva`, podemos colocar uma expressão. No quadro seguinte, podemos visualizar outro modo de escrever o algoritmo para a tabuada.

```
Algoritmo tabuada
Var
     num, i: inteiro
Início
    Escreva ("Digite um número:")
    Leia (num)
    Para i de 0 até 10 passo 1 faça
       Escreva (num, "x", i, "=", num * i )
    Fim_para
 Fim
```

Quadro 2 - Pseudocódigo - Algoritmo tabuada / Fonte: os autores.

Nesta aula, estudamos a estrutura de repetição por contagem, que utiliza uma variável contadora para controlar o laço. Essa estrutura é recomendada em situações nas quais sabemos, previamente, quantas vezes o comando deve ser repetido.

UNICESUMAR

AULA

3 ESTRUTURA ENQUANTO

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

A sintaxe da estrutura `Enquanto` é:

```
Enquanto <condição> faça
        <instruções>
Fim_enquanto
```

Na estrutura **`Para`**, tínhamos uma variável de controle (contador) para ditar o número de repetições do bloco de comandos. Na estrutura Enquanto, não há, necessariamente, variável de controle, impõe-se uma condição para controlar a repetição do algoritmo. Devemos garantir que, em algum momento, a condição seja quebrada, senão o algoritmo pode entrar em loop infinito (não parar nunca). Para impedir o loop infinito do algoritmo, utilizamos uma expressão condicional, tal como vimos na Unidade 2, para interromper a repetição (LOPES; GARCIA, 2002).

Outra situação é que, como o teste condicional é executado no início, podem ocorrer casos em que as instruções da estrutura de repetição nunca sejam execu-

tadas. Isso acontece quando o teste condicional da estrutura resulta em `falso`, logo, na primeira comparação (ASCENCIO; CAMPOS, 2010).

Agora que conhecemos os conceitos relacionados à estrutura Enquanto, construiremos um algoritmo que resolva o seguinte problema: ler vários números inteiros e informar quantos se encontram no intervalo de 100 a 300. Se for digitado o valor 0, o algoritmo encerra sua execução. Seguindo o método de estruturação de problemas visto na Unidade 1, temos:

- **Objetivo do algoritmo**: ler vários números e informar quantos estão no intervalo entre 100 e 300.
- **Entrada**: ler números inteiros, enquanto o número 0 não for digitado pelo usuário.
- **Processamento**: contar quantos números estão no intervalo entre 100 e 300.
- **Saída**: imprimir a quantidade de números entre 100 e 300.

Na entrada de dados, realizamos a leitura de números inteiros repetidas vezes até que o valor 0 seja digitado. O processamento consiste em contar a quantidade de números digitados que estão na faixa entre 100 e 300, para isso, utilizaremos uma variável do contadora, que nomearemos como `cont`. Para saber quantos valores estão dentro da faixa, utilizaremos a estrutura condicional `Se`, conforme visto na Unidade 2. Como saída, temos que informar o valor da variável `cont`.

Na construção de algoritmos, utilizando a estrutura `Enquanto`, o teste lógico é realizado no início do bloco de comandos, desse modo, precisamos ter um valor atribuído à variável usada na condição, antes de entrar na estrutura `Enquanto`. Além disso, no conjunto de instruções, dentro do laço de repetição, deve haver uma instrução que modifique o valor dessa variável, senão entraremos em um loop. Isso nos indica que, ao utilizar laços do tipo `Enquanto`, temos que ler a variável, tanto fora da estrutura de repetição quanto dentro. Observe o algoritmo do próximo quadro.

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

Quadro 3 - Pseudocódigo - Algoritmo `conta` / Fonte: os autores.

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

A próxima linha é a instrução, enquanto temos o teste lógico que analisa se o número é diferente de 0. Se o resultado for `verdadeiro`, temos a execução das instruções que estão dentro do laço, senão vai para a instrução após o `Fim_enquanto`. No laço de repetição, temos a verificação se o número está ou não na

faixa estabelecida. Para isso, é usada a estrutura condicional `Se`, em que temos duas expressões relacionais unidas por uma expressão lógica com o operador `e`. Se o resultado do teste lógico for `verdadeiro`, temos que `cont` recebe o valor que ele tem mais um, ou seja, é incrementado em uma unidade. Se o teste lógico resultar em `falso`, a execução segue para a linha posterior ao `Fim_se`. Note que, após o `Fim_se`, temos a leitura da variável novamente. Por que isso acontece? Se a leitura da variável fosse realizada apenas fora do laço de repetição, teríamos que o laço entraria em **loop infinito**, uma vez que o valor de `num` não se alteraria mais. As instruções dentro do laço serão repetidas até que na entrada seja obtido o valor 0. Quando este valor for obtido, tem-se a execução do comando após o `Fim_enquanto`, que exibe na tela o valor armazenado na variável `cont`. Na figura seguinte, podemos visualizar a simulação do algoritmo descrito.

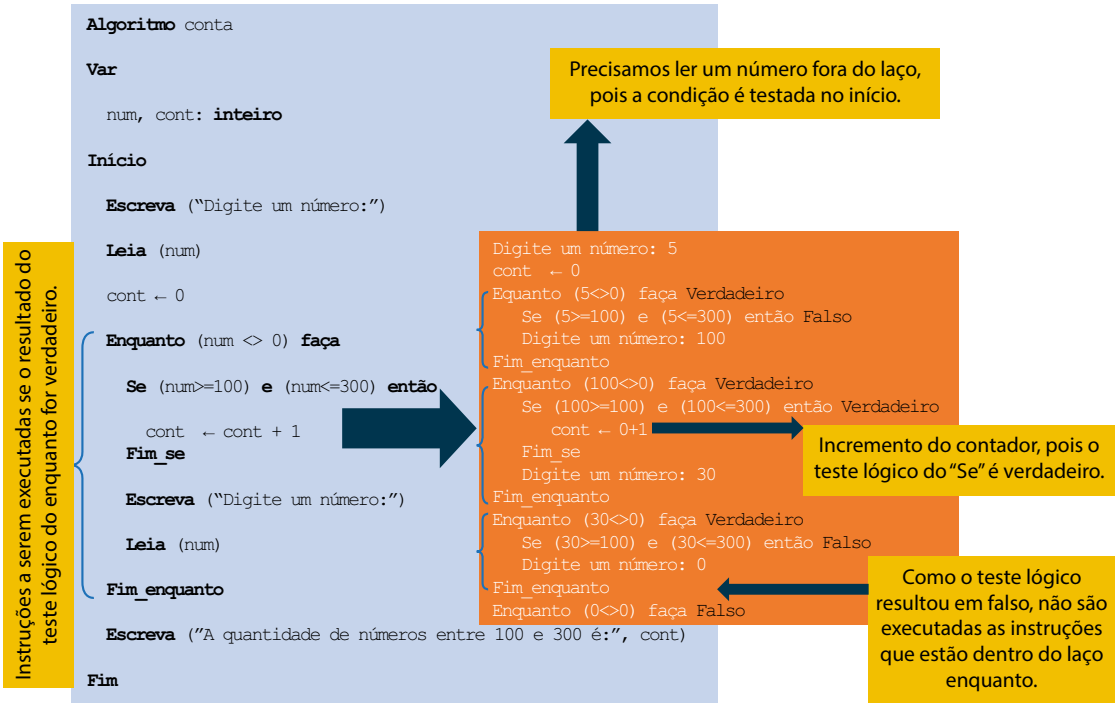


Figura 4 - Simulação - Algoritmo conta / Fonte: os autores.

Analisaremos, neste momento, o comportamento do algoritmo sem a entrada de dados dentro da estrutura de repetição. Note que a leitura é realizada apenas antes da estrutura de repetição.

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

Quadro 4 - Pseudocódigo - Algoritmo `conta` / Fonte: os autores.

A figura posterior apresenta um esquema que representa a simulação do algoritmo. A partir da entrada de dados, em que foi obtido o valor 130, tem-se a inicialização da variável `cont` e, em seguida, a estrutura enquanto. No início do enquanto, há um teste lógico que analisa se `num` é diferente de 0. Como 130 é diferente de 0, a avaliação do teste resulta em `verdadeiro`, com isto, tem-se a execução das instruções que estão dentro da estrutura `Enquanto`. Internamente, há um teste lógico que verifica se o número é maior ou igual a 100 e, simultaneamente, menor ou igual a 300, a avaliação deste teste é, nesse caso, verdadeira. Portanto, tem-se o incremento da variável `cont`, que passa a armazenar o valor 1.

Após o `Fim_se`, não temos nenhuma instrução, desse modo, a execução é retomada para a linha do `Enquanto`, em que, novamente, o teste lógico é executado. A variável `num` continua, ainda, com o valor 130, resultando em `verdadeiro` o teste lógico, pois 130 é diferente de 0. Na verificação da estrutura condicional, o teste também resulta em `verdadeiro`, com isso, se executa a

instrução que incrementa a variável `cont`. Novamente, a execução retorna para a linha do `Enquanto`, e o funcionamento do algoritmo será o mesmo, de modo que apenas a variável `cont` é incrementada a cada iteração do laço de repetição. Perceba que se não efetuarmos a leitura da variável `num` dentro da estrutura enquanto, também seu valor não será alterado e o código entra em **loop infinito**, pois sempre a condição `num <> 0` é satisfeita.

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

```
Digite um número: 130
cont ← 0
Equanto (130<>0) faça Verdadeiro
   Se (130>=100) e (130<=300) então Verdadeiro
      cont ← 0+1
   Fim_se
Fim_enquanto
Enquanto (130<>0) faça Verdadeiro
   Se (130>=100) e (130<=300) então Verdadeiro
      cont ← 1+1
   Fim_se
Fim_enquanto
Enquanto (130<>0) faça Verdadeiro
   Se (130>=100) e (130<=300) então Verdadeiro
      cont ← 2+1
   Fim_se
Fim_enquanto
...

Equanto (130<>0) faça Verdadeiro
   Se (130>=100) e (130<=300) então Verdadeiro
      cont ← n+1
   Fim_se
Fim_enquanto
...
```

Figura 5 - Simulação - Algoritmo `conta` / Fonte: os autores.

pensando juntos

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

AULA

4 ESTRUTURA REPITA

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

A sintaxe da estrutura **`Repita`** é:

```
Repita

    <instruções>

Até_que <condição>
```

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

Com o conhecimento que temos da estrutura `Repita`, reescreveremos o algoritmo que lê vários números e informa quantos estão no intervalo de 100 a 300.

Se for digitado o valor 0, o algoritmo encerra sua execução. Descrevemos os passos da estruturação desse problema na aula anterior, "Estrutura `Enquanto`".

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

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

Quadro 5 - Pseudocódigo – Algoritmo `conta` / Fonte: os autores.

Estudaremos cada linha do algoritmo para entender melhor o funcionamento dessa estruturação de repetição. Na primeira linha, temos a inicialização da variável `cont`, que conta o número de valores que estão na faixa entre 100 e 300. Afinal, por que inicializamos essa variável? Por exemplo, se efetuamos a leitura

de vários números e nenhum deles estava na faixa entre 100 e 300, qual o valor de `cont`? Não há como garantir que o valor será 0. Como uma variável é um espaço em memória, devemos inicializá-lo para que não fique nenhum "lixo".

Após a inicialização de `cont`, temos o início da estrutura `Repita`. Internamente a essa estrutura, temos a leitura do número, o qual é armazenado na variável `num`. Em seguida, temos a estrutura condicional `Se`, que analisa se o número é maior ou igual a 100 e menor ou igual a 300. O resultado do teste lógico do `Se` é `verdadeiro`, quando as duas expressões relacionais são verdadeiras e, então, `cont` é incrementado em 1. Se o resultado do teste da condição do `Se` for `falso`, a execução do algoritmo salta, diretamente, para a linha que impõe a condição para o laço de repetição. Nessa linha, temos a verificação se o número é igual a 0, isto é, se isso ocorrer, a repetição do laço finaliza e é executada a instrução `Escreva`, que imprime o conteúdo da variável `cont`.

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

AULA

5 ESTRUTURAS DE REPETIÇÃO Encadeadas

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

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

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

```
Para <var1> de <início> ate <fim> passo <incremento> faça
   Para <var2> de <início> ate <fim> passo <incremento>
  faça
      <instruções>
  Fim_para
Fim_para
```

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

```
Para <var> de <início> até <fim> passo <incremento>
faça
     Enquanto (<condição>) faça
          <instruções>
     Fim_enquanto
Fim_para
```

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

```
Para <var> de <início> até <fim> passo <incremento>
faça
       Repita
              <instruções>
        Até_que (<condição>)
Fim_para
```

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

```
Enquanto (<condição1>) faça
    Enquanto (<condição2>) faça
        <instruções>
    Fim_enquanto
Fim_enquanto
Enquanto (<condição1>) faça
    Repita
        <instruções>
    Até_que (<condição2>)
Fim_enquanto
Enquanto (<condição>) faça
   Para <var> de <início> até <fim> passo <incr.> faça
```

```
        <instruções>
    Fim_para
Fim_enquanto
```

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

```
Repita
    Para <var> de <início> até <fim> passo <incr.> faça
        <instruções>
    Fim_para
Até_que (<condição>)
Repita
    Enquanto (<condição2>) faça
        <instruções>
        Fim_enquanto
Até_que (<condição1>)
Repita
    Repita
        <instruções>
    Até_que (<condição>)
Até_que (<condição>)
```

Fique tranquilo(a)! Veremos a aplicação de estruturas encadeada no Problema 2.

explorando Ideias

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

Fonte: os autores.

AULA

6 EXERCÍCIOS RESOLVIDOS

De acordo com o que foi exposto até o momento, trabalharemos os exercícios resolvidos. Alguns deles contam com uma descrição detalhada. É mandatório que executemos os testes de mesa nesses algoritmos, como forma de praticar e, também, validar as soluções propostas.

Problema 1

Você se recorda da função matemática fatorial? O fatorial de um número `n` é o produto dos números naturais, começando em `n` e decrescendo até 1, isto é, o produto de todos os inteiros positivos menores ou igual a `n`. O fatorial de um número é representado por `n!` Tomemos como exemplo o fatorial de 7, representado por 7! O cálculo de 7! é: $7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$, igual a 5040.

Agora que recordamos o que é o fatorial, escreveremos um algoritmo para calcular o fatorial de um número qualquer. Seguindo o método de estruturação de problemas, conforme visto na Unidade 1, temos que:

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

Na entrada do problema, temos a leitura de um número inteiro, do qual queremos obter o fatorial. Precisamos armazenar este número em uma variável inteira que chamaremos de `num`. O processamento consiste em efetuar a operação aritmética de multiplicação de todos os números inteiros positivos menores ou igual a `num`. Observe que temos operações de multiplicação sucessivas, partindo de 7, conforme ilustra a figura posterior.

Aqui, temos uma informação importante, a operação de multiplicação tem que começar em 7 e ir até 1, ou seja, sabemos o número de vezes que a multiplicação precisa ser executada. Desse modo, podemos utilizar a estrutura **`Para`** em nosso algoritmo. Nessa estrutura, há uma variável de controle que delimita o intervalo para a execução do laço, no caso, temos que partir de 7 e ir até 1, isto é, teremos um "incremento" de -1 (decremento). Não podemos esquecer de declarar mais uma variável, a variável de controle, que é do tipo `inteiro`, chamaremos de `cont`. Como saída, temos que informar o valor do cálculo do fatorial. Para isso, utilizaremos o comando `Escreva`.

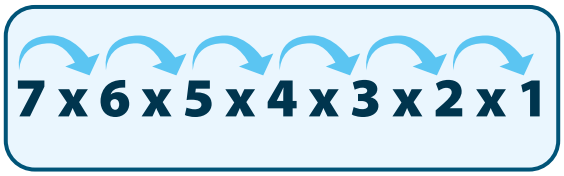


Figura 6 - Representação do cálculo de 7 / Fonte: os autores.

No quadro que segue, temos o algoritmo para o problema do fatorial. Faça você um teste de mesa, efetuando o cálculo do fatorial de 5.

```
Algoritmo fatorial
Var
     num, fat, cont: inteiro
Início
     Escreva("Digite o número que deseja calcu
     lar o fatorial:")
     Leia (num)
     fat <- 1
```

```
    Para cont de 7 até 1 passo -1 faça
        fat <- fat * cont
    Fim_para
    Escreva("O fatorial é: ", fat)
  Fim
```

Quadro 6 - Pseudocódigo - Algoritmo `fatorial` / Fonte: os autores.

Ao analisar o algoritmo, surge o questionamento: Por que a variável `fat` foi inicializada com 1? Podemos inicializá-la com 0? Se a inicialização da variável for 0, e considerando `num` como 5, teríamos `fat` igual a 0, pois qualquer número multiplicado por 0 é o próprio 0. Confira a ilustração a seguir.

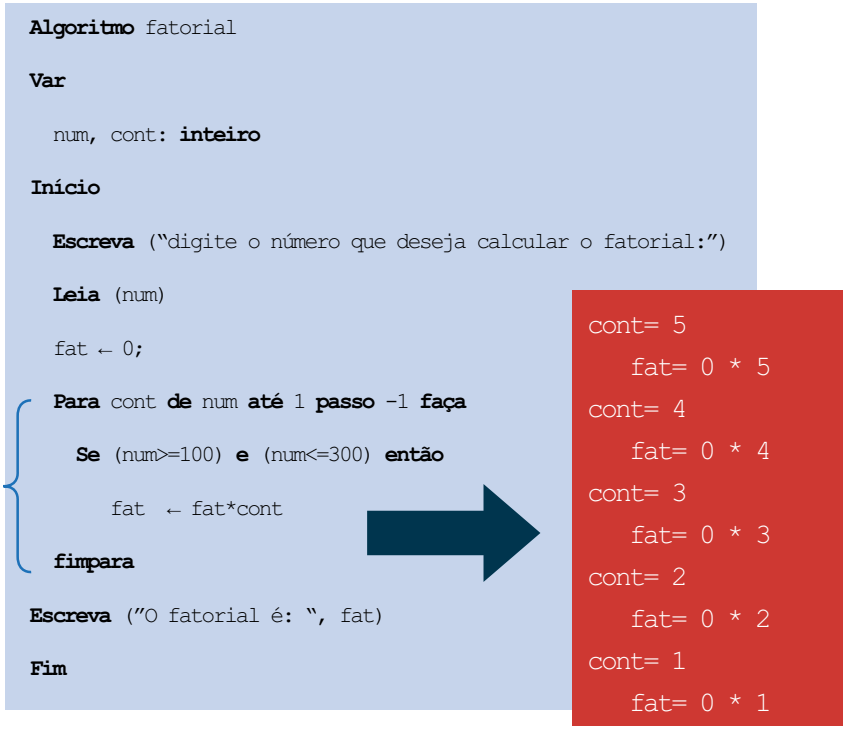


Figura 7 - Simulação - Algoritmo `fatorial` / Fonte: os autores.

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

Problema 2

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

- **Objetivo do algoritmo**: contabilizar o número de alunos aprovados e reprovados.
- **Entrada**: ler o número de provas da disciplina, os códigos e as notas dos alunos.
- **Processamento**: calcular a média de cada e contar o número de aprovados e reprovados.
- **Saída**: imprimir o número de aprovados e reprovados.

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

No próximo quadro, apresenta-se o algoritmo para o problema descrito. Neste algoritmo, temos a aplicação de estruturas de repetição encadeadas. Por quê? Precisamos saber o número de notas que devem ser lidas para cada aluno e, além disso, devem ser lidas as notas de vários até que o código do aluno seja 0. Como não sabemos de quantos alunos teremos que ler as notas, é interessante utilizar uma estrutura do tipo laço condicional. No primeiro momento, optamos por utilizar a estrutura **`Enquanto`**. Nessa estrutura, o teste condicional é realizado logo no início, desse modo, temos que realizar a leitura da informação utilizada no teste

condicional fora da estrutura. No caso, a informação obtida foi o código do aluno.

Se o teste condicional resulta em `verdadeiro`, ou seja, o código do aluno é diferente de 0, teremos de ler as notas. Em seguida, temos a inicialização das variáveis `soma` e `media`, pois a `soma` deve acumular as notas de um aluno por vez, e não de todos, simultaneamente. Para a leitura das notas, utilizamos a estrutura **Para**, pois sabemos, previamente, o número de notas que precisa ser lido. Dentro da estrutura `Para`, calcula-se o somatório das notas do aluno. Após ser finalizada a leitura de todas as notas (acumuladas na variável `soma`), efetua-se o cálculo da média. Em seguida, temos o teste condicional que analisa se o aluno está aprovado ou reprovado. Se a média é maior ou igual a 7, temos o incremento da variável `aprovados`, senão, a variável `reprovados` é incrementada. As instruções que se seguem efetuam a leitura do código do aluno. Depois do `Fim_enquanto`, são apresentados os números de alunos aprovados e reprovados.

```
Algoritmo fatorial
Var
     nprovas, cod, aprovados, reprovados, cont:
     inteiro media, nota, soma: real
Início
   Escreva ("Informe o número de notas da disci
   plina:")
   Leia (nprovas)
   aprovados <- 0
   reprovados <- 0
   Escreva ("Informe o código do aluno:")
   Leia (cod)
   Enquanto (cod <> 0) faça
```

```
      soma <- 0
      media <- 0
      Para cont de 1 até nprovas passo 1 faça
          Escreva ("Informe a nota:")
          Leia (nota)
          soma <- soma + nota;
      Fim_para
      media <- soma/nprovas;
      Se (media >= 7) então
         aprovados <- aprovados + 1
      Senão
         reprovados <- reprovados + 1
      Fim_se
      Escreva("Informe o código do aluno:")
      Leia(cod)
    Fim_enquanto
    Escreva("O número de aprovados é:", aprovados )
    Escreva("O número de reprovados é:", reprova-
dos)
  Fim
```

Quadro 7 - Pseudocódigo - Algoritmo `notas` / Fonte: os autores.

Em relação ao algoritmo apresentado, o que aconteceria se o cálculo da média fosse executado dentro da estrutura **Para**? Tomemos como exemplo que foram obtidas quatro notas 7, 8, 7 e 9, respectivamente. Na primeira passagem

da estrutura Para, teríamos média igual a 1,75. Na segunda, passada a média, seria 2, na terceira, 1,75, e na quarta, 3. Isto é, a cada nota obtida, efetuar-se-ia a divisão dessa nota pelo número de `notas` (`nota/nprovas`). Portanto, no interior da estrutura **`Para`**, devemos apenas acumular o valor das notas e realizar o cálculo da média, depois que todas as notas forem lidas.

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

No quadro a seguir, temos a solução do problema, utilizando o encadeamento com estruturas `Repita` e `Para`.

```
Algoritmo fatorial
Var
     nprovas, cod, aprovados, reprovados, cont:
     inteiro media, nota, soma: real
Início
   Escreva ("Informe o número de notas da disci
   plina:")
   Leia (nprovas)
   aprovados <- 0
   reprovados <- 0
   Repita
     Escreva ("Informe o código do aluno:")
     Leia (cod)
```

Quadro 8 - Pseudocódigo - Algoritmo `notas` / Fonte: os autores.

```
    soma <- 0
    media <- 0
    Para cont de 1 até nprovas passo 1 faça
        Escreva ("Informe a nota:")
        Leia (nota)
        soma <- soma + nota
    Fim_para
    media <- soma/nprovas
    Se (media >= 7) então
       aprovados <- aprovados + 1
    Senão
       reprovados <- reprovados + 1
    Fim_se
  Até_que (cod = 0)
 Escreva("O número de aprovados é:", aprovados)
 Escreva("O número de reprovados é:", reprova
 dos)
Fim
```

Observe que, quando utilizamos o **`Repita`**, não há a leitura da variável utilizada no teste condicional antes dessa mesma estrutura de repetição, diferentemente do `Enquanto`. Isso é possível, pois a estrutura **Repita** realiza o teste somente ao final, o que obriga a execução do bloco de comandos por, pelo menos, uma vez.

Problema 3

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

- **Objetivo do algoritmo**: imprimir "ALGORITMOS", tantas vezes quanto o usuário desejar.
- **Entrada**: ler um número inteiro.
- **Processamento**: não há.
- **Saída**: imprimir a palavra "Algoritmos", o número de vezes informado.

```
Algoritmo algoritmos
Var
     num, i: inteiro
Início
     Escreva("Informe o número de vezes que
     deseja imprimir:")
     Leia(num)
     Para i de 1 até num passo 1 faça
          Escreva("ALGORITMOS")
     Fim_para
Fim
```

Quadro 9 - Pseudocódigo - Problema 3 / Fonte: os autores.

Problema 4

Elabore um algoritmo que leia 100 números inteiros e conte quantos são pares e quantos são ímpares.

- **Objetivo do algoritmo**: ler 100 números e contar os pares e ímpares.
- **Entrada**: ler 100 números inteiros.

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

```
Algoritmo conta
Var
     num, npares, nimpares, i: inteiro
Início
      npares <- 0
      nimpares <- 0
      Para i de 1 até 100 passo 1 faça
          Escreva("Digite um número:")
          Leia(num)
          Se (num MOD 2 = 0) então
               npares <- npares + 1
          Senão
               nimpares <- nimpares + 1
          Fim_se
      Fim_para
      Escreva("A quantidade de números pares é:",
      npares)
      Escreva("A quantidade de números ímpares
      é:", nimpares)
Fim
```

Quadro 10 - Pseudocódigo - Problema 4 / Fonte: os autores.

Problema 5

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

- **Objetivo do algoritmo**: ler vários números, enquanto forem positivos, e contar quantos foram digitados.
- **Entrada**: ler números enquanto forem positivos.
- **Processamento**: contar a quantidade de números digitada.
- **Saída**: imprimir a quantidade de números positivos digitada.

```
Algoritmo conta
Var
     num, qtdade: inteiro
Início
     qtdade <- 0
     Escreva("Informe um número:")
     Leia(num)
     Enquanto (num > 0) faça
           qtdade <- qtdade + 1
           Escreva("Informe um número:")
           Leia(num)
     Fim_enquanto
     Escreva("O total de números positivos infor
     mado é:", qtdade)
Fim
```

Quadro 11 - Pseudocódigo - Problema 5 / Fonte: os autores.

Problema 6

Escreva um algoritmo que leia um conjunto de pedidos e calcule o total da compra. O pedido possui os seguintes campos: número, data (dia, mês e ano), preço unitário e quantidade. A cada entrada de um pedido, é preciso informar o custo daquele pedido que é a multiplicação entre o preço unitário e a quantidade. A entrada é encerrada quando o usuário informa 0 como número do pedido.

- **Objetivo do algoritmo**: ler vários pedidos e calcular o total da compra.
- **Entrada**: ler pedidos de compra (número, data, preço e quantidade) até que o número do pedido seja 0.
- **Processamento**: calcular o preço por pedido e o total da compra.
- **Saída**: a cada pedido cadastrado, imprimir seu valor e, ao final, imprimir o valor total da compra.

```
Algoritmo compras
Var
     pedido,  qtdade: inteiro
     preco, total: real
     data: Vetor[1..10] de caractere
Início
     total <- 0
     Escreva("Informe o número do pedido:")
     Leia(pedido)
     Enquanto (pedido <> 0) faça
           Escreva("Informe a data:")
           Leia(data)
           Escreva("Informe o preço unitário:")
           Leia(preco)
```

```
            Escreva("Informe a quantidade:")
            Leia(qtdade)
            total <- total + (preco * qtdade)
            Escreva("O valor da compra é:", pre
            co*qtdade)
            Escreva("Informe o número do novo pe
            dido ou 0 para finalizar:")
            Leia(pedido)
      Fim_enquanto
      Escreva("O total da compra é:", total)
Fim
```

Quadro 12 - Pseudocódigo - Problema 6 / Fonte: os autores.

Problema 7

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

- **Objetivo do algoritmo**: ler informações sobre vários atletas e apresentar o atleta mais alto, o mais pesado e a média de idade.
- **Entrada**: ler nome, sexo, idade, peso e altura até que seja digitado o nome "FIM".
- **Processamento**: identificar o atleta do sexo masculino mais alto, a atleta do sexo feminino mais pesada e a média de idade dos atletas.
- **Saída**: imprimir o nome do atleta mais alto e da atleta mais pesada e a média de idade.

```
Algoritmo atletas
Var
     idade, cont: inteiro
     peso, altura, media, alto, pesado, soma: real
     nome, nalto, npesado: Vetor[1..30] de caracter
     sexo: caractere
Início
    media <- 0
    cont <- 0
    alto <- 0
    pesado <- 0
    nalto <- “”
    npesado <- “”
    Escreva(“Informe o nome do atleta ou FIM para
    encerrar:”)
    Leia(nome)
    Enquanto (nome <> “FIM”) faça
        Escreva(“Informe a idade:”)
        Leia(idade)
        Escreva(“Informe o peso:”)
        Leia(peso)
        Escreva(“Informe a altura:”)
        Leia(altura)
```

```
        Escreva("Informe o sexo:")
        Leia(sexo)
        soma <- soma + idade
        cont <- cont + 1
        Se (sexo = 'M') ou (sexo = 'm') então
            Se (altura > alto) então
                alto <- altura
                nalto <- nome
            Fim_se
        Fim_se
        Se (sexo = 'F') ou (sexo = 'f') então
            Se (peso > pesado) então
                pesado <- peso
                npesado <- nome
            Fim_se
        Fim_se
        Escreva("Informe o nome do novo atleta ou
        FIM para encerrar:")
        Leia(nome)
    Fim_enquanto
    Escreva("O nome do atleta mais alto é:", nalto)
    Escreva("O nome da atleta mais pesada é:",
    npesado)
Fim
```

Quadro 13 - Pseudocódigo - Problema 7 / Fonte: os autores.

Problema 8

Faça um algoritmo que calcula a área de um triângulo e que não permita a entrada de dados inválidos, ou seja, as medidas devem ser maiores ou iguais a 0.

- **Objetivo do algoritmo**: calcular a área de um triângulo.
- **Entrada**: ler a base e a altura do triângulo.
- **Processamento**: calcular a área.
- **Saída**: imprimir o valor da área.

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

Quadro 14 - Pseudocódigo - Problema 8 / Fonte: os autores.

Problema 9

Construa um algoritmo que receba a idade e o estado civil de várias pessoas e imprima a quantidade de pessoas casadas (C), solteiras (S), divorciadas (D) e viúvas (V). O algoritmo finaliza quando for informado o valor 0 para idade.

- **Objetivo do algoritmo**: ler idade e estado civil e contabilizar a quantidade de pessoas por estado civil.
- **Entrada**: ler idade e estado civil de várias pessoas.
- **Processamento**: contabilizar o número de pessoas casadas, solteiras, divorciadas e viúvas.
- **Saída**: imprimir a quantidade de pessoas casadas, solteiras, divorciadas e viúvas.

```
Algoritmo pessoas
Var
   idade, ncasado, ndivorciado, nsolteiro: inteiro
    nviuva:  inteiro
    estado: caractere
Início
     ncasado <- 0
     nsolteiro <- 0
     nviuva <- 0
     ndivorciado <- 0
     Repita
          Escreva("Informe a idade da próxima pes-
          soa, ou zero para encerrar:")
         Leia(idade)
     Até_que (idade >= 0)
     Enquanto (idade <> 0) faça
     Repita
         Escreva("Informe o estado civil:")
```

```
        Leia(estado)
        Até_que ((estado = 'C') ou (estado = 'D')
        ou estado = 'S') ou (estado = 'V'))
        Se (estado = 'C') então
            ncasado <- ncasado + 1
        Fim_se
        Se (estado = 'S') então
            nsolteiro <- nsolteiro + 1
        Fim_se
        Se (estado = 'D') então
            ndivorciado <- ndivorciado + 1
        Fim_se
        Se (estado = 'V') então
            nviuva <- nviuva + 1
        Fim_se
        Repita
            Escreva("Informe a idade da próxima
            pessoa, ou zero para encerrar:")
            Leia(idade)
       Até_que (idade >= 0)
    Fim_enquanto
   Escreva("O número de casados é:", ncasados)
   Escreva("O número de solteiros é:", nsolteiro)
   Escreva("O número de separados é:", ndivorciado)
   Escreva("O número de viúvas é:", nviuva)
Fim
```

Quadro 15 - Pseudocódigo - Problema 9 / Fonte: os autores.

CONSIDERAÇÕES FINAIS

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

Nos laços de repetição condicionais, vimos as estruturas `Enquanto` e `Repita`. Vimos que a estrutura **`Enquanto`** é utilizada quando não sabemos, previamente, o número de repetições que deve ser executado e impomos uma condição que é testada logo no início do bloco de comandos. Aprendemos que, no uso desta estrutura, podemos utilizar um comando `Leia`, ou de atribuição, antes de um `Enquanto`, para satisfazer a condição de repetição. Também poderíamos utilizar um comando `Leia`, ou de atribuição (geralmente, a última instrução dentro da estrutura de repetição) para quebrar a condição de parada e interromper as repetições.

Do mesmo modo que a estrutura **`Enquanto`**, a estrutura **`Repita`** é utilizada quando temos um número indefinido de repetições, no entanto, o teste lógico é realizado no final da execução do bloco de comandos. Além de estudar as estruturas de repetição, conhecemos os conceitos e as aplicações de variáveis contadoras e acumuladoras.

Vimos que o conceito de encadeamento pode ser aplicado às estruturas de repetição, de modo análogo à estrutura condicional, vista na Unidade 2. Ao longo desta unidade, construímos algoritmos, utilizando todos os conceitos aprendidos. Também discutimos as particularidades de cada estrutura de repetição, enfatizando a forma de uso de cada uma delas e o seu uso encadeado. Para aperfeiçoar o raciocínio lógico e fixar os conceitos vistos, faça as atividades de autoestudo.

1. Escreva um algoritmo que leia a quantidade de números que se deseja digitar. Em seguida, leia esses números e encontre o maior e o menor.

2. Construa um algoritmo que leia números inteiros até que seja digitado 0. Calcule e escreva o número de valores lidos, a média aritmética, a quantidade de números pares e a quantidade de números ímpares.

3. Elabore um algoritmo para apresentar todos os números divisíveis por 5 e menores que 200.

4. Construa um algoritmo que leia números inteiros até que seja digitado um valor negativo. Ao final, informe a média dos números, o maior e o menor valor.

5. Uma indústria produz e vende vários produtos, para cada um deles, têm-se os seguintes dados: nome, quantidade produzida e quantidade vendida. Formule um algoritmo que:

   - Leia a quantidade de produtos que a empresa possui.
   - Imprima nome e quantidade em estoque para os produtos com estoque menor que 30.
   - Imprima o nome do produto com maior quantidade em estoque.

6. Elabore um algoritmo que imprima todas as tabuadas do 1 ao 10.

Computadores e máquinas, em geral, realizam trabalhos repetitivos de forma mais rápida, assertiva e eficaz que seres humanos. Estruturas de repetição são um exemplo dessa prerrogativa. Todavia existem outras formas de instruir máquinas a realizarem nosso trabalho. Confira o trecho de artigo a seguir, publicado no portal Universo Racionalista.

DEEP LEARMIMG VAI NOS ENSINAR A LIÇÃO DE NOSSAS VIDAS: O TRABALHO É PARA AS MÁQUINAS

[...].

Como um sussurro em uma língua estrangeira, foi como se você até o tivesse ouvido, mas talvez não tenha compreendido completamente seu significado. No entanto, é vital que nós entendamos esta nova língua, e o que ela vem nos contando cada vez mais, pois suas ramificações irão alterar tudo o que consideramos certezas sobre como nossa economia globalizada funciona, e o modo em que nós existimos nesse contexto.

Essa "língua" é uma nova classe de aprendizado de máquinas chamado deep learning, e o "sussurro" foi um computador utilizando disso para, aparentemente do nada, vencer Fan Hui no jogo Go, no qual ele foi 3 vezes campeão europeu. Não só uma vez, mas cinco seguidas, sem nenhuma derrota. Muitos dos que leram esta notícia acharam isso impressionante, mas de nenhuma maneira comparável com uma possível partida com Lee Se-dol, que muitos consideram como um dos melhores jogadores de Go vivos, se não o melhor. Imaginando um duelo tão grande de homem versus máquina, o melhor jogador da China previu que Lee não iria perder nenhuma das partidas, e o próprio Lee confiantemente esperava perder uma, no máximo.

E o que aconteceu quando eles se enfrentaram? Lee ganhou apenas uma das cinco partidas. Uma inteligência artificial chamada AlphaGo agora é um jogador de Go melhor que qualquer humano do planeta, e recebeu o rank "divino" de 9 dan. Em outras palavras, o nível de habilidade da máquina chega a ser considerado 'divino'. Go foi oficialmente dominado por uma máquina, assim como Jeopardy foi pelo Watson, e o xadrez pelo Deep Blue.

"A vitória histórica da AlphaGo é um sinal claro de que as máquinas saíram do crescimento linear para o parabólico".

Mas o que é Go? Colocando de um jeito simples, pense em Go como um Super Ultra Mega xadrez. Isto pode ainda parecer uma conquista pequena, apenas outra coroa de louros para as máquinas, enquanto elas continuam a provar serem superiores nos nossos passa-tempos, mas isto não é algo pequeno, e o que está acontecendo não é brincadeira.

A vitória histórica da AlphaGo é um sinal claro de que as máquinas saíram do crescimento linear para o parabólico. Os avanços tecnológicos se tornaram tão visivelmente exponenciais, que nós podemos ver muitos outros marcos sendo realizados muito antes do que esperávamos. Esses avanços exponenciais, mais notavelmente em forma de inteligência artificial limitada a tarefas específicas, levantam a questão de estarmos completamente despreparados para o futuro, caso continuemos a insistir nos empregos como nossa fonte primária de renda.

[...]

Fonte: César (2016, on-line)[2].

livro

Algoritmos: Programação para Iniciantes

Autor: Gilvan Vilarim

Editora: Ciência Moderna

Sinopse: este livro tem por objetivo apresentar os fundamentos da lógica para a programação de computadores, capacitando o leitor a construir algoritmos estruturados, traz assuntos e exercícios baseados em experiências reais do autor, vivenciadas em sala de aula. A leitura não exige tanto conhecimento do assunto por parte do leitor, mas apenas familiaridade com a operação do microcomputador. Voltado para estudantes dos cursos técnicos introdutórios em programação de computadores, aplicados nas áreas de Informática, Ciência da Computação, Engenharias, técnico profissionalizante de nível médio e outras, o livro apresenta como características principais uma nova linguagem estruturada para a construção de algoritmos; texto renovado e atual, fortemente baseado na didática utilizada em aulas de programação; intensa carga de exercícios resolvidos e problemas propostos; curiosidades sobre computadores, algoritmos e programação, estimulando o interesse do leitor pelo assunto.

UNIDADE

ESTRUTURAS DE DADOS HOMOGÊNEAS e Heterogêneas

PROFESSORES
Dra. Gislaine Camila Lapasini Leal
Me. Pietro Martins de Oliveira

PLANO DE ESTUDO

A seguir, apresentam-se as aulas que você estudará nesta unidade: • Estrutura de dados homogênea • Vetores • Matrizes • Estrutura de dados heterogênea • Exercícios resolvidos.

OBJETIVOS DE APRENDIZAGEM

Estudar as estruturas de dados homogêneas • Estudar vetores e matrizes • Conhecer estruturas de dados heterogêneas (registros) • Construir algoritmos utilizando estruturas de dados homogêneas e heterogêneas • Aprender com os exercícios resolvidos.

INTRODUÇÃO

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

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

Estudaremos as estruturas de dados heterogêneas, registros que agregam informações de diversos tipos. Abordaremos, especificamente, como realizar atribuição, entrada e saída de dados, utilizando registros. Na construção de algoritmos, faremos uso das estruturas homogêneas e heterogêneas a partir dos conceitos vistos nas demais unidades, principalmente as estruturas de repetição, pois elas são utilizadas nas operações de atribuição, leitura e escrita.

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

AULA

1 ESTRUTURAS DE DADOS HOMOGÊNEAS

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

AULA

2 VETORES

As estruturas unidimensionais, conhecidas como vetores, consistem em um arranjo de elementos armazenados na memória principal, sequencialmente, todos com o mesmo nome (LOPES; GARCIA, 2002). O vetor segue o mesmo princípio de uma matriz linha da matemática, como ilustra a figura a seguir. Em cada coluna da linha, temos uma variável com o mesmo nome, no entanto em diferentes posições no arranjo. A posição de cada variável no arranjo é indicada por um índice. No caso apresentado, temos cinco posições, de modo que, na posição 1, temos o valor 4; na posição 2, o valor 7; na posição 3, o valor 10; na posição 4, o valor 1, e na posição 5, o valor 5.

4	7	10	1	5
1	2	3	4	5

Figura 1 - Vetor / Fonte: adaptada de Lopes e Garcia (2002).

O uso de vetores associa-se à criação de tabelas, em que temos a definição de uma única variável, que é dimensionada com determinado tamanho. A dimensão deve

ser uma constante inteira e positiva (MANZANO; OLIVEIRA, 1997). A declaração de um vetor é realizada na seção de variáveis (`Var`), do seguinte modo:

```
<nome>: Vetor [<dimensão>] de <tipo de dado>
```

Em que:

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

Um exemplo de declaração de vetor é:

```
Media: Vetor[1..5] de real
```

O nome da variável é `media`, ela possui cinco posições e é do tipo `real`. O acesso a cada posição do vetor é realizado por meio da especificação do índice, isto é, `media[1]` indica o elemento que está na posição 1 do vetor, que, no caso da figura anterior, é o valor 4.

As operações de atribuição, leitura e escrita são realizadas para cada elemento do vetor, de modo que não é possível operar sobre o vetor como um todo (exceto para vetores de `caractere`). Desse modo, ao utilizar os comandos de atribuição, leitura e escrita, devemos utilizar o nome da variável mais o índice. Note que, como as operações devem ser realizadas elemento a elemento, temos que utilizar uma estrutura de repetição para a entrada e saída de dados. Das estruturas de repetição, vistas na Unidade 3, qual a mais adequada? Como um vetor possui um tamanho fixo, informado no momento de sua declaração, sabemos, previamente, o número de repetições que precisa ser realizado. Portanto, a estrutura **`Para`** é adequada.

Se você não entendeu como funciona um vetor, fique tranquilo(a), pois construiremos um algoritmo para aplicar os conceitos estudados até aqui. Desse modo, considere o seguinte problema: ler o nome e as quatro notas de dez alunos e imprimir uma relação contendo o nome do aluno, as notas e a média final. Seguindo o método de estruturação de problemas, temos que:

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

Na entrada de dados, temos que efetuar a leitura do nome do aluno e quatro notas. Observe que a leitura tem que ser realizada para dez alunos. Sem utilizar a estrutura de vetor, precisaríamos de dez variáveis para armazenar nome, 40 variáveis para armazenar as quatro notas dos dez alunos e dez variáveis para armazenar a média. Com o uso de vetores, precisamos declarar uma variável para armazenar o nome, quatro variáveis para armazenar as notas e uma para armazenar a média.

O tipo de dado utilizado para armazenar o nome é `caractere`, e para as notas e médias, utilizamos o `real`. Podemos utilizar `inteiro`? Não, pois tanto para nota quanto para média admitem-se valores decimais. A entrada de dados é realizada por meio do comando `Leia`, no entanto devemos lembrar que não é possível obter todos os valores do vetor de uma única vez, precisamos ler elemento a elemento. O processamento consiste em calcular a média das quatro notas para cada aluno e armazenar na variável `media`. Na saída de dados, temos que informar nome, notas e médias dos dez alunos. Para isso, utilizaremos o comando `Escreva` em uma estrutura de repetição, pois temos que escrever elemento a elemento do vetor. No quadro seguinte, apresenta-se o algoritmo para o problema descrito.

```
Algoritmo media_notas
Var
    nome: Vetor[1..10] de caractere
    n1, n2, n3, n4, media: Vetor[1..10] de real
    cont: inteiro
Início
   Para cont de 1 até 10 passo 1 faça
       Escreva("Informe nome do aluno:")
       Leia(nome[cont])
       Escreva("Informe a nota 1:")
       Leia(n1[cont])
```

```
        Escreva("Informe a nota 2:")
        Leia(n2[cont])
        Escreva("Informe a nota 3:")
        Leia(n3[cont])
        Escreva("Informe a nota 4:")
        Leia(n4[cont])
        media[cont] <- (n1[cont] + n2[cont]+n3
        [cont] + n4[cont])/4
   Fim_para
    Para cont de 1 até 10 passo 1 faça
       Escreva("Aluno:", nome[cont])
       Escreva("Nota 1:", n1[cont])
       Escreva("Nota 2:", n2[cont])
       Escreva("Nota 3:", n3[cont])
       Escreva("Nota 4:", n4[cont])
       Escreva("Média:", media[cont])
    Fim_para
Fim
```

Quadro 1 – Pseudocódigo – Algoritmo `media_notas` / Fonte: os autores.

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

Fique tranquilo(a)! A seguir, construiremos mais algoritmos, utilizando vetores.

Ordenação em vetor

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

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

O método da bolha não é o método mais eficiente, contudo é bastante simples. A seguir, veremos seu funcionamento. O método consiste em percorrer o vetor repetidas vezes, comparando os elementos vizinhos. Se eles estão fora de ordem, efetua-se uma troca de posição. O número de repetições de "varredura" é dado pelo número de elementos do vetor menos 1. Tomaremos, como exemplo, a ordenação (crescente) do vetor apresentado na figura seguinte. Primeiramente, veremos o funcionamento do método passo a passo e, em seguida, a implementação do algoritmo. Como podemos observar, o vetor não está ordenado.

4	7	10	1	5

Figura 2 - Vetor inicial / Fonte: os autores.

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

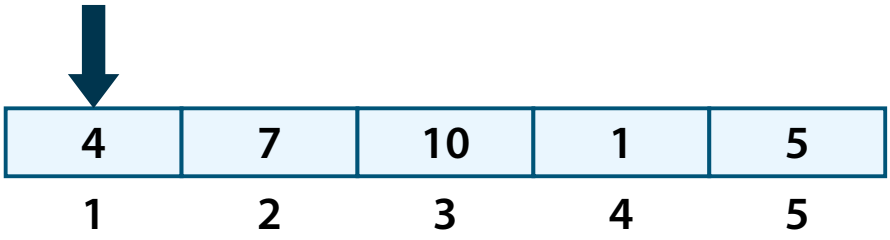


Figura 3 - Primeira repetição / Fonte: os autores.

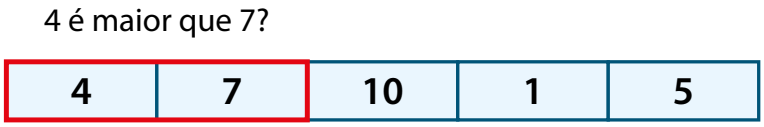


Figura 4 - Comparação entre elementos / Fonte: os autores.

Agora, comparamos o elemento da posição 1 com o elemento da posição 3 (figura seguinte). Como 4 não é maior que 10, não há troca. Depois, comparamos o elemento da posição 1 com o elemento da posição 4 (figura posterior).

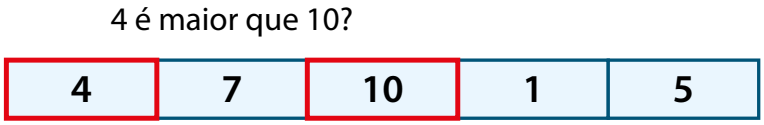


Figura 5 - Comparação entre elementos / Fonte: os autores.

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

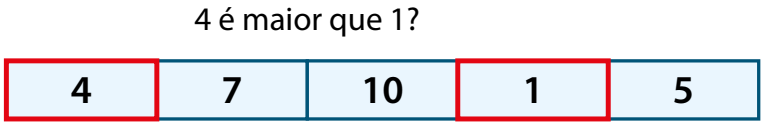


Figura 6 - Comparação entre elementos / Fonte: os autores.

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

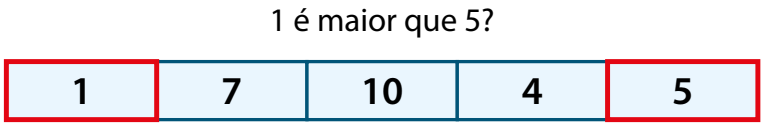


Figura 7 - Comparação entre elementos / Fonte: os autores.

Dessa forma, garantimos que o menor elemento entre todos os elementos do vetor esteja na primeira posição. De maneira análoga, temos que repetir o processo de comparação, agora, comparando o elemento da posição 2 com os demais elementos (figura seguinte).

1	7	10	4	5

Figura 8 - Segunda repetição / Fonte: os autores.

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

7 é maior que 10?

1	7	10	4	5

Figura 9 - Comparação entre elementos / Fonte: os autores.

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

7 é maior que 4?

1	7	10	4	5

Figura 10 - Comparação entre elementos / Fonte: os autores.

Prosseguimos com a comparação, agora, temos que analisar o elemento da posição 2 e o da posição 5.

1	4	10	7	5

Figura 11 - Ordenação do vetor / Fonte: os autores.

4 é maior que 5?

1	4	10	7	5

Figura 12 - Comparação entre elementos / Fonte: os autores.

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

1	4	10	7	5

Figura 13 - Terceira repetição / Fonte: os autores.

Ao compararmos o elemento da posição 3 com o elemento da posição 4 (figura seguinte), temos que 10 é maior que 7. Portanto, há troca (figura posterior).

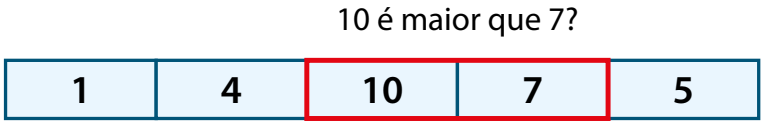


Figura 14 - Comparação entre elementos / Fonte: os autores.

Figura 15 - Ordenação do vetor / Fonte: os autores.

Ainda, temos que comparar o elemento da posição 3 com o da posição 5 (figura seguinte).

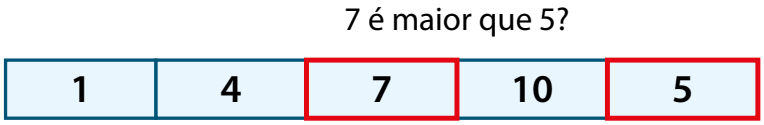


Figura 16 - Comparação entre elementos / Fonte: os autores.

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

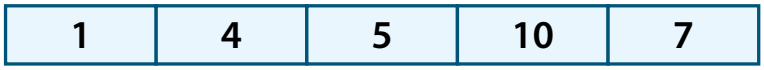


Figura 17 - Ordenação do vetor / Fonte: os autores.

Repetimos, agora, o processo de comparações, iniciando na quarta posição, conforme ilustra a figura a seguir.

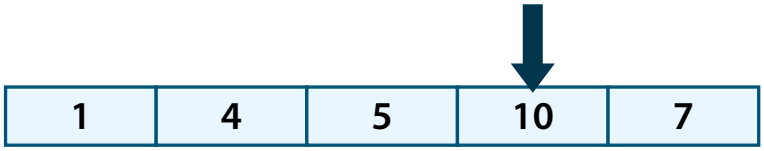


Figura 18 - Quarta repetição / Fonte: os autores.

Ao comparar o elemento da posição 4 com o da posição 5, temos que 10 é maior que 7. Os elementos trocam de posição e, como não há mais comparações, temos o vetor ordenado. Observe as figuras a seguir.

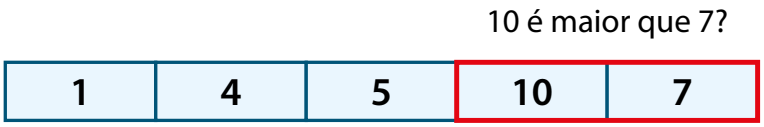


Figura 19 - Comparação entre elementos / Fonte: os autores.

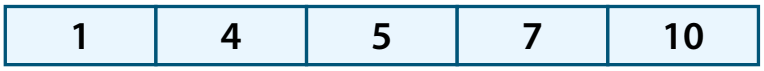


Figura 20 - Vetor ordenado / Fonte: os autores.

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

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

```
                aux <- num[i]
                num[i] <- num[j]
                num[j] <- aux
            Fim_se
        Fim_para
    Fim_para
    Para i de 1 até 5 passo 1 faça
        Escreva(num[i])
     Fim_para
Fim
```

Quadro 2 - Pseudocódigo - Algoritmo `ordena` / Fonte: os autores.

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

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

11	3	8	1	15

Figura 21 - Vetor / Fonte: os autores.

Busca em vetor

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

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

Método sequencial

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

O algoritmo que realiza a busca sequencial de um número qualquer em um vetor é apresentado a seguir.

```
Algoritmo Busca_Sequencial
Var
    vnum: vetor[1..20] de inteiro
    num: inteiro
```

```
    acha: lógico
Início
    Para i de 1 até 20 passo 1 faça
        Escreva("Informe um número:")
        Leia(vnum[i])
    Fim_para
    Escreva("Informe o número que deseja bus
    car:")
    Leia(num)
    i <- 1
    acha <- falso
    Enquanto (i <= 20) e (acha = falso) faça
        Se (num = vnum[i]) então
            acha <- verdadeiro
        Senão
             i <- i + 1
        Fim_se
   Fim_enquanto
   Se (acha = verdadeiro) então
        Escreva("O elemento foi encontrado na
        posição", i)
   Senão
        Escreva("O elemento não foi encontrado")
   Fim_se
Fim
```

Quadro 3 - Pseudocódigo - Algoritmo **Busca_Sequencial** / Fonte: os autores.

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

AULA

3 MATRIZES

Uma matriz é uma variável homogênea multidimensional, formada por uma sequência de variáveis do mesmo tipo, com o mesmo nome e alocadas, sequencialmente, na memória. Para acessar cada elemento da matriz, utilizam-se índices; para cada dimensão, um índice.

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

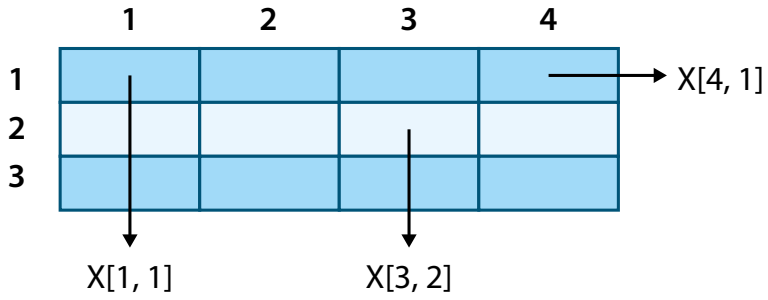


Figura 22 - Matriz / Fonte: adaptada de Ascencio e Campos (2010).

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

```
<Variável>: Vetor[<dimensão1>, <dimensão2>] de
                    <tipo>
```

Em que:

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

Um exemplo de declaração de matriz é:

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

Nesse exemplo, temos uma matriz com oito linhas e quatro colunas (próxima figura). Na primeira dimensão, representamos o número de linhas e, na segunda, o número de colunas.

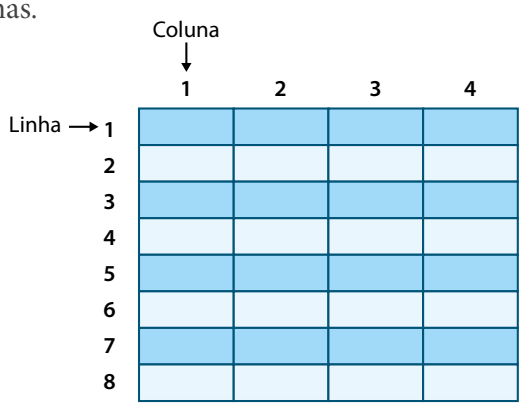


Figura 23 - Exemplo de matriz / Fonte: os autores.

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

Com base nos conceitos de matrizes, escreveremos um algoritmo para o seguinte problema: efetuar a leitura de quatro notas de 20 alunos, calcular a média de cada aluno e a média da turma. Desse modo, sistematizando o problema, temos que:

- **Objetivo do algoritmo**: calcular a média de cada aluno e a média da turma.
- **Entrada**: ler quatro notas de 20 alunos.
- **Processamento**: calcular a média das quatro notas dos alunos e a média da turma.
- **Saída**: imprimir a média de cada aluno e a média da turma.

Na entrada de dados, temos que efetuar a leitura de quatro notas de 20 alunos. Para isso, utilizaremos uma matriz com 20 linhas e quatro colunas, ou seja, a primeira dimensão (linhas) representa o número de alunos, e a segunda dimensão (colunas), as notas. O tipo de dado utilizado para armazenar essas informações é `real`. Como temos uma matriz bidimensional, utilizaremos dois laços de repetição para efetuar a operação de leitura.

O processamento consiste no cálculo da média das quatro notas para cada aluno, que armazenaremos em uma variável vetor de dimensão 20. Para o cálculo da média da turma, utilizaremos uma variável para acumular os valores das médias de cada aluno e, ao sair da estrutura de repetição, calculamos a média. Na saída, temos que informar a média de cada aluno e da turma. No quadro seguinte, é apresentado o algoritmo para o problema descrito.

```
Algoritmo media_notas
Var
    media: Vetor[1..20] de real
    notas: Vetor[1..20, 1...4] de real
    somat, soma, mediat: real
```

```
    i, j: inteiro
Início
   somat <- 0
   soma <- 0
   Para i de 1 até 20 passo 1 faça
        Para j de 1 até 4 passo 1 faça
             Escreva("Informe a nota", j, "do
             aluno" i)
             Leia(notas[i,j])
             soma <- soma + notas[i,j]
        Fim_para
       media[i] <- soma/4
       somat <- somat + soma
       soma <- 0
   Fim_para
   mediat <- somat/20
   Para i de 1 até 20 passo 1 faça
        Escreva("A média do aluno", i , "é :",
        media[i])
   Fim_para
   Escreva("A média da turma é:", mediat)
 Fim
```

Quadro 4 - Pseudocódigo - Algoritmo **media_notas** / Fonte: os autores.

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

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

pensando juntos

Será que a variável caractere é armazenada na memória principal como um vetor? Na realidade, é possível criar tanto variáveis caracteres simples quanto vetores. A diferença reside apenas na quantidade de caracteres a serem armazenados.

AULA

4 ESTRUTURAS DE DADOS HETEROGÊNEAS

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

Registros

Os registros são estruturas de dados que agregam diversas informações, que podem ser de diferentes tipos. Com essa estrutura, é possível gerar novos tipos de dados, além dos definidos pelas linguagens de programação (ASCENCIO; CAMPOS, 2010). Em um registro, cada informação é denominada campo, o qual pode ser de diferentes tipos, ou ainda, representar outros registros. A definição de um tipo registro deve ocorrer antes da declaração de variáveis registro, pois pode ocorrer a necessidade de declarar uma variável com tipo registro dentro do próprio registro. A sintaxe para declaração é dada por (MANZANO; OLIVEIRA, 1997):

```
Tipo
        <identificador> = registro
        <lista dos campos e seus tipos>
```

```
Fim_registro
...
Var
<variáveis> : <identificador>
```

Em que:

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

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

As operações de atribuição, leitura e escrita são realizadas utilizando o nome da variável registro e o seu campo correspondente, separado por um caractere "." (ponto) (MANZANO; OLIVEIRA, 1997). Conhecemos os conceitos e a sintaxe de registros, mas qual a estrutura de um registro e como utilizá-los?

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

A figura seguinte ilustra o layout de um registro, contendo os campos: nome, primeira nota, segunda nota, terceira nota e quarta nota.

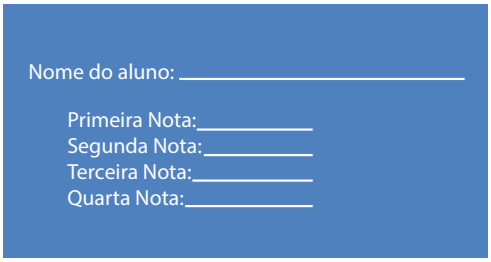


Figura 24 - Registro aluno / Fonte: adaptada de Manzano e Oliveira (1997).

Um algoritmo que realiza a leitura de nome e quatro notas de um aluno, utilizando o conceito de registros, apresenta-se no quadro a seguir. Com esse exemplo, fica mais claro como deve ser realizada a declaração, a atribuição, a leitura e a escrita, utilizando registros.

```
Algoritmo registroaluno
Tipo
     Cad_aluno = registro
         nome: Vetor[1..50] de caractere
         nota1: real
         nota2: real
         nota3: real
         nota4: real
Fim_registro
Var
    aluno: Cad_aluno
Início
       Escreva("Informe nome do aluno:")
       Leia(aluno.nome)
       Escreva("Informe a nota 1:")
       Leia(aluno.nota1)
       Escreva("Informe a nota 2:")
       Leia(aluno.nota2)
       Escreva("Informe a nota 3:")
       Leia(aluno.nota3)
       Escreva("Informe a nota 4:")
       Leia(aluno.nota4)
   Fim
```

Quadro 5 - Pseudocódigo - Algoritmo **registroaluno** / Fonte: os autores.

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

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

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

```
Algoritmo registroaluno
Tipo
    Cad_aluno = registro
        notas: Vetor[1..4] de real
        nome: Vetor[1..50] de caractere
Fim_registro
Var
    aluno: Cad_aluno
    i: inteiro
Início
      Escreva("Informe nome do aluno:")
      Leia(aluno.nome)
      Para i de 1 até 4 passo 1 faça
            Escreva("Informe a nota", i, ":")
            Leia(aluno.notas[i])
      Fim_para
 Fim
```

Quadro 6 - Pseudocódigo – Algoritmo **registroaluno** / Fonte: os autores.

No exemplo apresentado na aula "Vetores", tínhamos a leitura de nome e notas para dez alunos. Podemos resolver o mesmo problema utilizando registros? Sim, pois podemos declarar vetores de registros, conforme apresentado no quadro seguinte.

```
Algoritmo registroaluno
Tipo
    Cad_aluno = registro
        notas: Vetor [1..4] de real
        nome: Vetor[1..50] de caractere
Fim_registro
Var
    alunos: Vetor[1..10] de Cad_aluno
    i, j: inteiro
Início
    Para i de 1 até 10 faça
        Escreva("Informe nome do aluno:")
        Leia(alunos[i].nome)
        Para j de 1 até 4 passo 1 faça
            Escreva("Informe a nota", j, ":")
            Leia(alunos[i].notas[j])
        Fim_para
    Fim_para
    Para i de 1 até 10 faça
        Escreva("Aluno:", alunos[i].nome)
        Para j de 1 até 4 passo 1 faça
```

```
                Escreva("Nota", j, ":", alunos[i].
                notas[j])
            Fim_para
        Fim_para
Fim
```

Quadro 7 - Pseudocódigo – Algoritmo **registroaluno** / Fonte: os autores.

conecte-se

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

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

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

Se você ficou com dúvidas, não se preocupe! Escreveremos mais um algoritmo, aplicando os conceitos aprendidos nesta unidade, na próxima aula.

AULA

5 EXERCÍCIOS RESOLVIDOS

Problema 1

O problema consiste em ler informações (código, descrição, preço e quantidade em estoque) de 20 produtos. Além disso, deve ser permitido executar quantas consultas o operador desejar. Desse modo, digitando o código do produto, apresentam-se a ele o preço e o saldo em estoque. Se o código digitado não existir, o usuário deve ser informado. Sistematizando o problema, temos que:

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

O quadro a seguir apresenta o algoritmo para o problema descrito. Observe que esse algoritmo envolve o conceito de registros, laço de repetição e busca em vetor.

```
Algoritmo registroproduto
Tipo
     Cad_produto = registro
         codigo: inteiro
         nome: Vetor[1..30] de caractere
         preco: real
         saldo: inteiro
Fim_registro
Var
    produto: Vetor[1..10] de Cad_produto
    i, codigo: inteiro
    acha: lógico
    resp: Vetor[1..3] de caractere
Início
    Para i de 1 até 20 faça
        Escreva("Informe código do produto:")
        Leia(produto[i].codigo)
        Escreva("Informe a descrição do produ
        to:")
        Leia(produto[i].nome)
        Escreva("Informe o preço do produto:")
        Leia(produto[i].preco)
        Escreva("Informe o saldo em estoque do
        produto:")
       Leia(produto[i].saldo)
   Fim_para
   resp <- "sim"
```

```
    Enquanto (resp = "sim") faça
        Escreva("Informe o código a ser pesqui-
sado:")
        Leia(codigo)
        i <- 1
        acha <- falso
        Enquanto (i < = 20) e (acha= falso) faça
             Se (produto[i].codigo = codigo) então
                 acha <- verdadeiro
             Senão
                  i <- i + 1
             Fim_se
       Fim_enquanto
       Se (acha = verdadeiro) então
           Escreva("O preço é:", produto[i].preco)
           Escreva("O saldo em estoque é:", pro-
           duto[i].saldo)
        Senão
             Escreva("Não há produto com o códi-
             go informado")
        Fim_se
        Escreva("Deseja continuar a pesquisa?")
       Leia(resp)
    Fim_enquanto
Fim
```

Quadro 8 - Pseudocódigo - Algoritmo **registroproduto** / Fonte: os autores.

Problema 2

Escreva um algoritmo que leia um vetor com 30 elementos inteiros e os escreva em ordem contrária à leitura.

- **Objetivo do algoritmo**: ler e imprimir um vetor com 30 elementos inteiros.
- **Entrada**: ler um vetor com 30 números inteiros.
- **Processamento**: não há.
- **Saída**: imprimir os 30 elementos em ordem contrária à leitura.

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

Quadro 9 - Pseudocódigo - Problema 2 / Fonte: os autores.

Problema 3

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

- **Objetivo do algoritmo**: ler duas matrizes e imprimir a diferença entre elas.
- **Entrada**: ler duas matrizes 5 x 5.
- **Processamento**: calcular a diferença entre as duas matrizes.
- **Saída**: imprimir a matriz diferença.

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

```
        Fim_para
    Fim_para
    Para i de 1 até 5 passo 1 faça
        Para j de 1 até 5 passo 1 faça
            Escreva(C[i,j])
        Fim_para
    Fim_para
Fim
```

Quadro 10 - Pseudocódigo - Problema 3 / Fonte: os autores.

Problema 4

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

- **Objetivo do algoritmo**: ler informações sobre 20 contatos e realizar consulta pelo nome.
- **Entrada**: ler nome, fone e e-mail de 20 contatos e nome para pesquisa.
- **Processamento**: encontrar o nome informado pelo usuário.
- **Saída**: imprimir o telefone se existir o nome informado. Senão, informar ao usuário que não há contato cadastrado com o nome digitado.

```
Algoritmo agenda
Tipo
    Cad_agenda = registro
        nome: Vetor[1..30] de caractere
        fone: Vetor[1..15] de caractere
```

```
        e-mail: Vetor[1..20] de caractere
Fim_registro
Var
    contato: Vetor[1..20] de Cad_agenda
    i: inteiro
    acha: lógico
    nomebusca: Vetor[1..30] de caractere
    resp: Vetor[1..3] de caractere
Início
    Para i de 1 até 20 faça
        Escreva("Informe nome do contato:")
        Leia(contato[i].nome)
        Escreva("Informe o telefone:")
        Leia(contato[i].fone)
        Escreva("Informe o e-mail:")
        Leia(contato[i].email)
        Escreva("Informe o e-mail:")
        Leia(contato[i].email)
    Fim_para
    resp <- "sim"
    Enquanto (resp = "sim") faça
        Escreva("Informe o nome a ser pesquisa-
        do:")
```

```
        Leia(nomebusca)
        i <- 1
        acha <- falso
        Enquanto (i <= 20) e (acha = falso) faça
            Se (contato[i].nome = nomebusca)
            então
                acha <- verdadeiro
            Senão
                i <- i + 1
            Fim_se
        Fim_enquanto
        Se (acha = verdadeiro) então
            Escreva("O fone é:", contato[i].fone)
        Senão
            Escreva("Não há contato com o nome
            informado")
        Fim_se
        Escreva("Deseja continuar a pesquisa?")
        Leia(resp)
    Fim_enquanto
Fim
```

Quadro 11 - Pseudocódigo - Problema 4 / Fonte: os autores.

Problema 5

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

- **Objetivo do algoritmo**: ler informações sobre 30 professores e realizar consulta pela matrícula.
- **Entrada**: ler matrícula, nome, lotação e salário e matrícula para pesquisa.
- **Processamento**: encontrar o professor com a matrícula informada pelo usuário.
- **Saída**: imprimir a lotação, se existir a matrícula informada. Senão, informar ao usuário que não há professor com a matrícula digitada.

```
Algoritmo registroprofessor
Tipo
     Cad_professor = registro
         matricula: inteiro
         nome: Vetor[1..30] de caractere
         lotacao: Vetor[1..30] de caractere
         salario: real
Fim_registro
Var
    professor: Vetor[1..30] de Cad_professor
    i, registro: inteiro
    acha: lógico
    resp: Vetor[1..3] de caractere
Início
```

```
Para i de 1 até 30 faça
    Escreva("Informe a matrícula:")
    Leia(professor[i].matricula)
    Escreva("Informe o nome:")
    Leia(professor[i].nome)
    Escreva("Informe a lotação:")
    Leia(professor[i].lotacao)
    Escreva("Informe o salário:")
    Leia(professor[i].salario)
Fim_para
resp <- "sim"
Enquanto (resp = "sim") faça
    Escreva("Informe a matricula a ser pes-
    quisado:")
    Leia(registro)
    i <- 1
    acha <- falso
    Enquanto (i <= 20) e (acha = falso) faça
        Se (professor[i].matricula = regis-
        tro) então
            acha <- verdadeiro
        Senão
            i <- i + 1
        Fim_se
```

```
        Fim_enquanto
        Se (acha = verdadeiro) então
            Escreva("A lotação é:", professor[i].
            lotacao)
            Escreva("O salário é:", professor[i].
            salario)
        Senão
            Escreva("Não há professor com a ma-
            trícula informada")
        Fim_se
        Escreva("Deseja continuar a pesquisa?")
        Leia(resp)
    Fim_enquanto
Fim
```

Quadro 12 - Pseudocódigo – Problema 5 / Fonte: os autores.

CONSIDERAÇÕES FINAIS

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

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

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

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

1. Escreva um algoritmo que leia 20 nomes e imprima o primeiro caractere de cada nome.

2. Formule um algoritmo que entre com o nome do aluno e as notas de quatro provas de 20 alunos. Imprima nome, nota1, nota2, nota3, nota4 e média de cada aluno e informe a média geral da turma.

3. Escreva um algoritmo que leia vários nomes até que seja digitado o valor "FIM". Imprima o primeiro caractere de cada nome.

4. Elabore um algoritmo que leia um vetor de 50 números inteiros e imprima o maior e o menor número.

5. Escreva um algoritmo que leia dois vetores A e B, com 20 números inteiros cada. Efetue a soma dos dois vetores em um vetor C e imprima o vetor C do primeiro ao último elemento.

6. Construa um algoritmo que leia o nome, o preço de compra e o preço de venda de 30 produtos distintos e imprima a quantidade de mercadorias que apresentam lucro < 15% e quantas apresentam lucro > 30%.

7. Formule um algoritmo que leia uma matriz 5 x 5 de números inteiros e imprima os elementos da diagonal principal.

8. Desenvolva um algoritmo que efetua a leitura dos nomes de cinco alunos e, também, de suas quatro notas bimestrais. Calcule a média de cada aluno e apresente os nomes classificados em ordem crescente de média.

9. Escreva um algoritmo que recebe duas matrizes inteiras de ordem 5 e imprima a soma e a diferença entre elas.

10. Formule um algoritmo que receba os valores de contas a pagar de uma empresa ao longo do ano, de modo que cada linha representa um mês do ano, e cada coluna, uma semana do mês. Com isso, temos uma matriz 12 x 4. Calcule e imprima: total de contas a pagar por mês e total anual.

Você sabia que os dados de um programa ou algoritmo, quando em execução, ficam armazenados na memória primária do computador? Em dispositivos como desktops, laptops, smartphones e tablets, a memória RAM é responsável por esse papel, funcionando como um grande vetor linear, pronto para receber e enviar dados aos programas. Para entender melhor esse componente eletrônico, confira trechos do artigo a seguir.

COMO FUNCIONA A MEMÓRIA RAM?

[...]

O que significa RAM

O termo é um acrônimo para Random Access Memory, isto é, memória de acesso aleatório. Isso implica que esta memória pode acessar os dados de forma não sequencial (ao contrário de uma fita cassete, por exemplo), acelerando em muito os processos de leitura e escrita. Qualquer setor livre ou já preenchido é imediatamente encontrado e processado.

Entretanto, ao contrário de um disco rígido, a memória RAM é totalmente volátil, o que significa que todos os dados armazenados podem ser perdidos quando o dispositivo não é devidamente alimentado. Mas se há este contratempo, saiba que ela é milhares de vezes mais rápida que a varredura do disco físico.

[...]

Antes de tudo, você precisa saber que existem dois tipos básicos de RAM, que são a memória estática e a memória dinâmica. A primeira pode ser menos reconhecida pelo público em geral, mas está presente em muitos componentes de nossos computadores, principalmente nos processadores, onde formam a memória cache (nós explicaremos o conceito mais abaixo, durante a ilustração do percurso de funcionamento da memória RAM).

A memória RAM estática é composta por flip-flops (montados com quatro a seis transistores) e não necessita ser atualizada constantemente, o que a torna muito mais rápida e eficiente para trabalhos que requerem baixa latência. Os estados de saída podem ser 0 ou 1, sendo perdidos apenas quando o fornecimento de energia é cortado.

Em contrapartida, ela ocupa um espaço físico muito maior, sendo impraticável a construção de pentes de memória para uso tradicional. Outro problema é o custo bem mais elevado.

Já a memória RAM dinâmica, ou DRAM, é a que vemos para comprar nas lojas e que equipam nossas placas mãe. Suas células são compostas por um capacitor e um transistor, sendo o transistor uma espécie de portão (que barra ou dá passagem ao pulso elétrico) e o capacitor o responsável por armazenar a informação (novamente, estados de 0 ou 1).

Fonte: Pankiewivcz (2009, on-line)[3].

UNIDADE

SUB-ROTINAS

5

PROFESSORES
Dra. Gislaine Camila Lapasini Leal
Me. Pietro Martins de Oliveira

PLANO DE ESTUDO

A seguir, apresentam-se as aulas que você estudará nesta unidade: • Sub-rotinas • Procedimentos • Escopo de variáveis • Passagem de parâmetros • Funções • Recursividade • Exercícios resolvidos.

OBJETIVOS DE APRENDIZAGEM

Conhecer e desenvolver sub-rotinas • Conhecer (procedimentos e funções) • Entender os mecanismos de passagem de parâmetros e escopo de variáveis • Compreender algoritmos recursivos • Absorver o conteúdo por meio dos exercícios resolvidos.

INTRODUÇÃO

Caro(a) aluno(a), chegamos à última unidade da disciplina de Algoritmos e Lógica de Programação I. Nesta unidade, você estudará as sub-rotinas. No dia a dia, um programador, geralmente, encontra problemas complexos e abrangentes. Para resolver esse tipo de problema, o primeiro passo consiste em decompô-lo em subproblemas para, assim, facilitar o processo de entendimento, análise e resolução. Na solução dos subproblemas, utilizam-se sub-rotinas, blocos de instruções que realizam tarefas específicas. Na literatura, encontramos, também, os termos subalgoritmo, subprograma e módulo, equivalentes a sub-rotinas. Existem dois tipos destas: os procedimentos e as funções. Estudaremos cada uma delas, destacando sua sintaxe e o modo de aplicação.

Ao trabalhar com sub-rotinas, surge a necessidade de entender o conceito de escopo de variáveis, compreender o que são variáveis locais e variáveis globais e como elas impactam o algoritmo. O escopo relaciona-se à visibilidade de uma variável, ao longo do algoritmo, sendo que uma variável local é aquela que podemos utilizar apenas dentro da sub-rotina, e uma variável global é aquela que está acessível de qualquer parte do algoritmo.

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

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

AULA

1 SUB-ROTINAS

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

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

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

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

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

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

- Divide o problema em partes coerentes.
- Facilita o teste.
- Evita repetição do código.
- Apresenta melhor legibilidade do algoritmo.

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

AULA

2 PROCEDIMENTOS

PROCEDIMENTO

Um procedimento consiste em um trecho de código (conjunto de instruções), com início e fim, identificado por um nome, usado para invocar a execução da rotina de qualquer parte do programa principal ou em uma sub-rotina qualquer. Quando uma sub-rotina é chamada, ela é executada e, ao seu término, o processamento retorna para a linha seguinte a da instrução que a chamou (MANZANO; OLIVEIRA, 1997). Uma sub-rotina de procedimento se caracteriza por não retornar valor para quem as chamou (ASCENCIO; CAMPOS, 2010). E sua sintaxe é:

```
Procedimento <nome do procedimento>

Var

     <variáveis>

Início

     <instruções>

Fim_procedimento
```

Outro ponto importante é que as sub-rotinas devem ser declaradas antes do programa ou das sub-rotinas que as chamam. Para exemplificar o funcionamento de sub-rotinas do tipo procedimento, construiremos um programa calculadora que apresenta um menu de opções no programa principal e possibilita ao usuário escolher as seguintes operações: adição, subtração, multiplicação, divisão e sair (MANZANO; OLIVEIRA, 1997).

Sistematizando as informações do problema, temos que:

- **Objetivo do algoritmo**: criar uma calculadora com as operações de adição, subtração, multiplicação e divisão.
- **Entrada**: ler a opção do usuário. E se for uma operação aritmética, efetuar a operação.
- **Processamento**: efetuar a operação aritmética selecionada.
- **Saída**: imprimir o resultado da operação aritmética.

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

O quadro a seguir apresenta o algoritmo que efetua as quatro operações aritméticas. Note que, para cada operação, há um procedimento associado.

```
Algoritmo calculadora
     Procedimento soma
     Var
          A, B, Resultado: real
     Início
          Escreva("Procedimento SOMA")
          Escreva("Informe o valor de A:")
          Leia(A)
```

```
            Escreva("Informe o valor de B:")
            Leia(B)
            Resultado <- A + B
            Escreva("O resultado da soma é:", Re-
            sultado)
      Fim_procedimento

      Procedimento subtracao
      Var
            A, B, Resultado: real
      Início
            Escreva("Procedimento SUBTRAÇÃO")
            Escreva("Informe o valor de A:")
            Leia(A)
            Escreva("Informe o valor de B:")
            Leia(B)
            Resultado <- A - B
            Escreva("O resultado da subtração
            é:", Resultado)
      Fim_procedimento

Procedimento multiplicacao
      Var
            A, B, Resultado: real
      Início
            Escreva("Procedimento MULTIPLICAÇÃO")
            Escreva("Informe o valor de A:")
            Leia(A)
```

```
Escreva("Informe o valor de B:")
          Leia(B)
          Resultado <- A * B
          Escreva("O resultado da multiplicação
          é:", Resultado)
     Fim_procedimento

     Procedimento divisao
     Var
          A, B, Resultado: real
     Início
          Escreva("Procedimento DIVISÃO")
          Escreva("Informe o valor de A:")
          Leia(A)
          Escreva("Informe o valor de B:")
          Leia(B)
          Resultado <- A / B
          Escreva("O resultado da divisão é:",
          Resultado)
     Fim_procedimento
Var
    opcao: inteiro
Início
    opcao <- 0
    Enquanto (opcao <> 5) faça
        Escreva("1 - Soma")
        Escreva("2 - Subtração")
        Escreva("3 - Multiplicação")
```

```
        Escreva("4 - Divisão")
        Escreva("5 - Sair")
        Escreva("Digite a opção:")
        Leia(opcao)
        Se (opcao <> 5) faça
            Caso (opcao)
                Seja 1 faça soma()
                Seja 2 faça subtracao()
                Seja 3 faça divisao()
                Seja 4 faça multiplicacao()
                Senão Escreva("Opção inválida")
            Fim_caso
        Fim_se
    Fim_enquanto
Fim
```

Quadro 1 - Pseudocódigo – Algoritmo calculadora / Fonte: os autores.

Na solução do problema, utilizaram-se quatro procedimentos, os quais foram declarados antes do programa principal. Cada procedimento possui início, fim e declaração de variáveis. As variáveis declaradas em uma sub-rotina são ditas variáveis locais. Na aula seguinte, estudaremos o escopo das variáveis. Observe que, no programa principal, ocorre a chamada para cada um dos procedimentos declarados. O problema "calculadora" foi dividido em subproblemas (cada operação aritmética), e para esses subproblemas, construíram-se sub-rotinas do tipo procedimento.

AULA

3 ESCOPO DE VARIÁVEIS

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

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

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

A figura a seguir ilustra o conceito de escopo de variáveis. Temos uma rotina principal que contém as rotinas 1 e 2. Internamente à rotina 1, temos duas rotinas (1.1 e 1.2) e, internamente à rotina 2, temos a rotina 2.1 Observe que a rotina principal possui duas variáveis `A`, `B`, que são variáveis globais em relação às sub-rotinas 1 e 2. No entanto, na rotina 1, temos a declaração de outra variável `A`, a qual assume um contexto local para esta sub-rotina, mas é global para as

sub-rotinas 1.1 e 1.2, do mesmo modo, a variável X. Já as variáveis W e Y são variáveis locais que pertencem às rotinas 1.1 e 1.2, respectivamente.

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

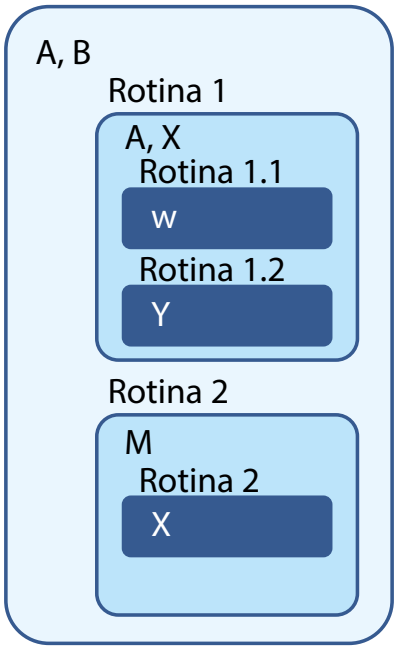


Figura 1 - Escopo de variáveis / Fonte: Manzano e Oliveira (1997).

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

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

AULA

4 PASSAGEM DE PARÂMETROS

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

Os parâmetros formais são aqueles declarados, por meio de variáveis, juntamente com a identificação do nome da sub-rotina. Os parâmetros reais, por sua vez, são aqueles que substituem os parâmetros formais, no uso da sub-rotina, por um programa principal ou por uma rotina chamadora.

O quadro seguinte apresenta o algoritmo para o problema da calculadora, utilizando o conceito de parâmetros.

```
Algoritmo calculadora
     Procedimento soma(A, B: real)
     Var
          Resultado: real
     Início
          Resultado <- A + B
```

```
            Escreva("O resultado da soma é:", Re-
            sultado)
      Fim_procedimento

      Procedimento subtracao(A, B: real)
      Var
            Resultado: real
      Início
            Resultado <- A - B
            Escreva("O resultado da subtração é
            :", Resultado)
      Fim_procedimento

      Procedimento multiplicação(A, B: real)
      Var
            Resultado: real
      Início
            Resultado <- A * B
            Escreva("O resultado da multiplicação
            é :", Resultado)
      Fim_procedimento

      Procedimento divisão(A, B: real)
      Var
            Resultado: real
      Início
            Resultado <- A / B
            Escreva("O resultado da divisão é :",
            resultado)
      Fim_procedimento
Var
```

```
     opcao: inteiro
     X, Y: real
Início
    opcao <- 0
    Enquanto (opcao <> 5) faça
        Escreva("1 - Soma")
        Escreva("2 - Subtração")
        Escreva("3 - Multiplicação")
        Escreva("4 - Divisão")
        Escreva("5 - Sair")
        Escreva("Digite a opção:")
        Leia(opcao)
        Se (opcao <> 5) faça
               Escreva("Informe o valor de X:")
               Leia(X)
               Escreva("Informe o valor de Y:")
               Leia(Y)
               Caso (opcao)
                    Seja 1 faça soma(X,Y)
                    Seja 2 faça subtracao(X,Y)
                    Seja 3 faça divisao(X,Y)
                    Seja 4 faça multiplica-
                    cao(X,Y)
                    Senão Escreva("Opção inváli-
                    da")
            Fim_caso
        Fim_se
    Fim_enquanto
Fim
```

Quadro 2 - Pseudocódigo – Algoritmo calculadora / Fonte: os autores.

Observe que, agora, temos a leitura das variáveis X e Y, no programa principal, e na identificação dos procedimentos, temos os parâmetros que são exigidos pelos mesmos. Os parâmetros reais são os valores de X e Y obtidos na entrada de dados, e os parâmetros formais são A e B.

A passagem de parâmetro ocorre quando é realizada a substituição dos parâmetros formais pelos reais, no momento da execução da sub-rotina. A passagem ocorre de duas formas: por valor e por referência (MANZANO; OLIVEIRA, 1997).

A passagem de parâmetro **por valor** é caracterizada pela não alteração do valor do parâmetro real, quando o parâmetro formal é manipulado na sub-rotina. Isto é, qualquer alteração na variável local da sub-rotina não afetará o valor do parâmetro real correspondente. Na passagem de parâmetros por valor, a sub-rotina trabalha com cópias dos valores passados no momento de sua chamada (MANZANO; OLIVEIRA, 1997; ASCENCIO; CAMPOS, 2010).

Na passagem de parâmetro **por referência**, os parâmetros passados para a sub-rotina consistem em endereços de memória ocupados por variáveis. O acesso a determinado valor é realizado por apontamento do endereço. Na passagem por referência, o valor do parâmetro real é alterado, quando o parâmetro formal é manipulado dentro da sub-rotina (ASCENCIO; CAMPOS, 2010).

Exemplificaremos a passagem de parâmetro por valor e por referência, utilizando o problema de calcular o fatorial de um número inteiro. No quadro a seguir, apresenta-se o algoritmo, utilizando a passagem de parâmetros por valor.

```
Algoritmo calculafatorial
     Procedimento Fatorial(X: inteiro)
     Var
          i, fat: inteiro
     Início
          fat <- 1
          Para i de 1 até X passo 1 faça
               fat <- fat * i
          Fim_para
          Escreva("O fatorial é:", fat)
```

```
     Fim_procedimento
Var
    n: inteiro
Início
    Escreva("Informe o número que deseja calcu-
    lar o fatorial:")
    Leia(n)
    Fatorial(n)
Fim
```

Quadro 3 - Pseudocódigo – Algoritmo fatorial / Fonte: Manzano e Oliveira (1997).

Nesse caso, temos que a variável `X` é o parâmetro formal que receberá o valor da variável `n`, por meio do procedimento fatorial. Ao término do laço de repetição, o conteúdo da variável `fat` é impresso. Lembre-se: a variável `fat` é válida apenas dentro do procedimento fatorial.

No quadro que segue, temos a passagem de parâmetros por referência, no caso, a variável `fat`. Sempre que a passagem for por referência, devemos inserir a instrução `Var` na declaração da sub-rotina. A variável `X` continua a passar por valor.

```
Algoritmo calculafatorial
     Procedimento Fatorial(X: inteiro, Var fat:
     inteiro)
     Var
           i: inteiro
     Início
          Para fat <- 1
                i de 1 até X passo 1 faça
                  fat <- fat * i
            Fim_para
     Fim_procedimento
Var
```

```
     n, resultado: inteiro
Início
     Escreva("Informe o número que deseja calcular
     o fatorial:")
     Leia(n)
     Fatorial(n, resultado)
     Escreva("O fatorial é:", resultado)
Fim
```

Quadro 4 - Pseudocódigo – Algoritmo fatorial / Fonte: Manzano e Oliveira (1997).

Ao final do laço de repetição, o resultado do fatorial está armazenado na variável `fat`, que foi passada para a sub-rotina, como um parâmetro por referência, e ao retornar ao programa principal, este conteúdo é armazenado, diretamente, na variável `resultado` do programa principal (parâmetro real), a qual apresenta o valor recebido dentro da sub-rotina, por meio da variável `fat`. Esse tipo de passagem de parâmetro é, geralmente, utilizado para obter determinado valor de dentro de uma sub-rotina (MANZANO; OLIVEIRA, 1997).

A figura que se apresenta a seguir ilustra a passagem de parâmetros por referência no cálculo do fatorial. No algoritmo principal, temos a leitura do número que devemos calcular o fatorial. Este valor é armazenado na variável `n`. Em seguida, há a chamada do procedimento `Fatorial` com os parâmetros reais n sendo passado por valor e `resultado` por referência, respectivamente. O valor de `n` é copiado para a variável `X`. Para calcular o fatorial, utiliza-se o valor de `X` para controlar o laço de repetição. Ao término da execução do procedimento, o valor da variável `fat`, que foi passado por referência, modifica, automaticamente, o conteúdo da variável `resultado`.

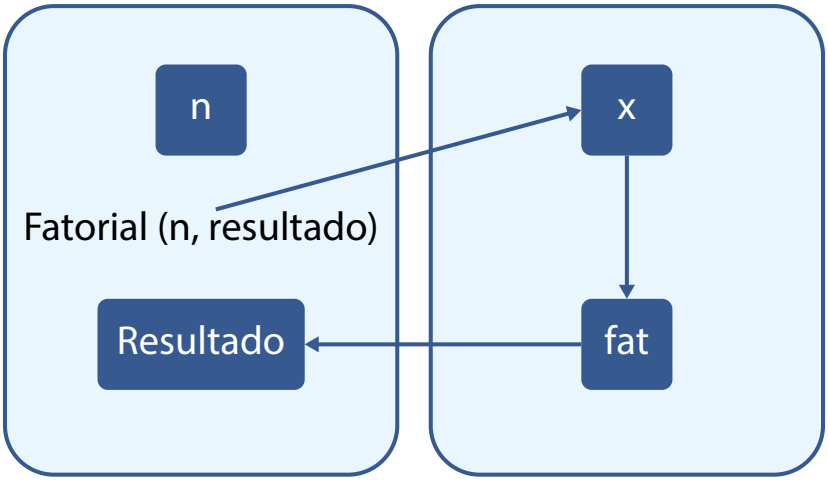
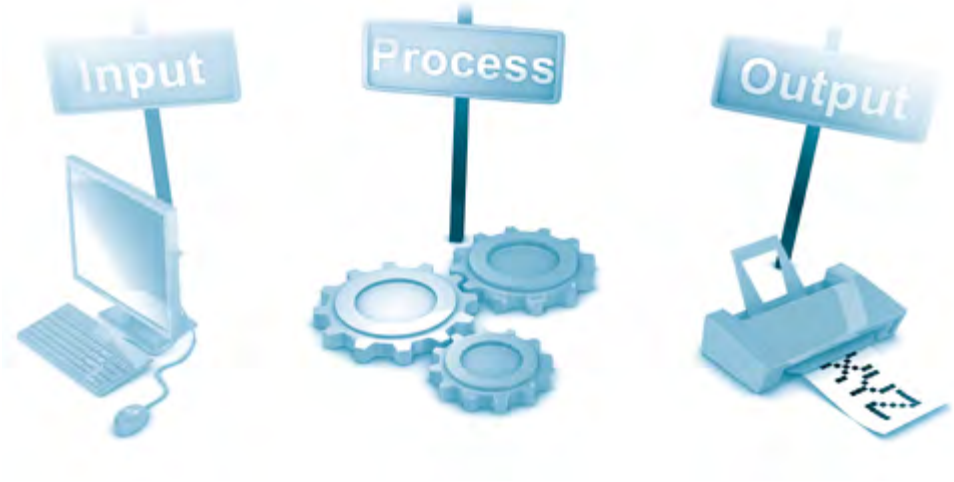


Figura 2 - Representação da passagem por referência / Fonte: os autores.

pensando juntos

Na passagem de parâmetros por referência, o parâmetro é um endereço de memória e não o conteúdo do parâmetro propriamente dito.



AULA

5 FUNÇÕES

Uma função também é uma sub-rotina, que tem como objetivo desviar a execução do programa principal para realizar uma tarefa específica, **com uma única diferença: sempre retorna um valor** (ASCENCIO; CAMPOS, 2010).

A sintaxe é dada por:

```
Função <nome_func> (<parâmetros>): <tipo_func>

Var

     <variáveis locais>

Início

     <instruções>

     Retorne <valor>

Fim_função
```

Um exemplo de uso de função é apresentado no quadro a seguir. Observe que a função tem um tipo associado, no caso, inteiro. O tipo da função deve, obrigatoriamente, ser compatível com o tipo do dado retornado, por meio do comando `Retorne`, ou seja, nesse caso, a variável `fat` é do tipo `inteiro`, assim como a função `Fatorial`.

```
Algoritmo calculafatorial
     Função Fatorial(X: inteiro): inteiro
     Var
          i, fat: inteiro
     Início
          fat <- 1
          Para i de 1 até X passo 1 faça
              fat <- fat * i
          Fim_para
          Retorne fat
     Fim_função
Var
    n, resultado: inteiro
Início
    Escreva("Informe o número que deseja calcu-
    lar o fatorial:")
    Leia(n)
    resultado <- Fatorial(n)
    Escreva("O fatorial é:", resultado)
Fim
```

Quadro 5 - Pseudocódigo - Algoritmo calcula fatorial / Fonte: os autores.

No algoritmo principal, o resultado da chamada da função é atribuído à variável `resultado`, pois toda função retorna um valor. Nesse caso, a função `Fatorial` retorna o conteúdo de `fat`, que é, justamente, o resultado do cálculo.

AULA

6 RECURSIVIDADE

Um objeto é dito recursivo se ele for definido em termos de si próprio. Wirth (1999) destaca que o conceito de recursão não é encontrado apenas na matemática, podemos encontrá-lo no nosso dia a dia.

A recursividade é um mecanismo que permite uma função chamar a si mesma, direta ou indiretamente. Uma função é dita como recursiva quando possui uma chamada a si própria (ZIVIANI, 2004; GUIMARÃES; LAGES, 1994). O princípio da recursividade consiste em diminuir, sucessivamente, o problema em um problema menor, até que a simplicidade permita resolvê-lo de modo direto, isto é, sem recorrer a si mesmo. Desse modo, uma função recursiva possui um passo básico e um passo recursivo. O passo básico é, imediatamente, conhecido, e o passo recursivo tenta resolver um subproblema do problema inicial (GUIMARÃES; LAGES, 1994).

Você conhece algum problema recursivo? Retomaremos o problema do fatorial, que é um exemplo clássico de recursão. A definição recursiva é dada por:

$$n = \begin{cases} 1 & se\ n = 0 \\ n * (n-1)! & se\ n > 0 \end{cases}$$

O quadro a seguir apresenta a função recursiva para o cálculo do fatorial.

```
Função Fatorial(X: inteiro): inteiro
Início
    Se (X = 0) então
        Retorne 1
    Senão
        Retorne X * Fatorial(X-1)
    Fim_se
Fim_função
```

Quadro 6 - Pseudocódigo – Função **`Fatorial`** recursiva / Fonte: os autores.

A figura posterior ilustra a simulação da função fatorial para o cálculo do `Fatorial` de 5. Observe que são realizadas chamadas sucessivas à função fatorial até chegar na condição de parada, `Se (X = 0) então`.

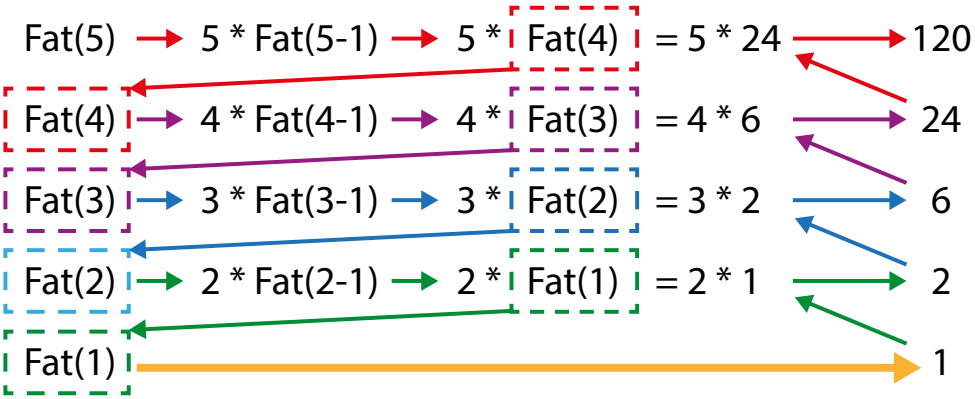


Figura 3 - Simulação da função fatorial / Fonte: os autores.

Outro exemplo clássico de problema recursivo é a série de Fibonacci, dada pela seguinte sequência numérica: 0, 1, 1, 2, 3, 5, 8, 13, 21, [...]. Sua definição matemática recursiva é dada por:

$$Fib(n) = \begin{cases} 0 & se\ n = 0 \\ 1 & se\ n = 1 \\ Fib(n-2) + Fib(n-1) & se\ n > 1 \end{cases}$$

No quadro a seguir, apresenta-se a função que calcula a série de Fibonacci.

```
Função Fibonacci(X: inteiro): inteiro
Início
    Se (X = 0) ou (X = 1) então
        Retorne X
    Senão
        Retorne Fibonacci(X-2) + Fibonacci(X-1)
    Fim_se
Fim_função
```

Quadro 7 - Pseudocódigo – Função **`Fibonacci`** recursiva / Fonte: os autores.

Ziviani (2004) destaca que a recursividade permite uma descrição mais concisa dos algoritmos, especialmente, quando o problema tem natureza recursiva ou utiliza estruturas recursivas. Algumas desvantagens da recursão são: algoritmos recursivos, geralmente, consomem mais recursos, especialmente, memória, e são mais difíceis de serem depurados.

explorando Ideias

Para saber um pouco mais sobre recursividade, leia o artigo "A História da Ciência e o ensino da recursividade: as torres de Hanói", disponível em: https://revistas.pucsp.br/hcensino/article/download/5489/5768. O artigo destaca a importância da recursividade e apresenta o problema clássico das Torres de Hanói.

Fonte: os autores.

AULA

7 EXERCÍCIOS RESOLVIDOS

Problema 1

Escreva uma sub-rotina que apresente o somatório dos N primeiros números inteiros.

- **Objetivo do algoritmo**: ler um número inteiro e apresentar o somatório dos números no intervalo de 1 até N.
- **Entrada**: ler um número inteiro.
- **Processamento**: efetuar o somatório de 1 até N.
- **Saída**: imprimir o valor do somatório.

```
Algoritmo somatorio
    Função soma(X: inteiro): inteiro
    Var
        i, total: inteiro
    Início
        total <- 0
        Para i de 1 até X passo 1 faça
```

```
                total <- total + i
            Fim_para
            Retorne total
        Fim_função
Var
        n, resultado: inteiro
Início
        Escreva("Informe o número:")
        Leia(n)
        resultado <- soma(n)
        Escreva("O somatório é:", resultado)
Fim
```

Quadro 8 - Pseudocódigo – Problema 1 / Fonte: os autores.

Problema 2

Elabore uma sub-rotina que receba um número e apresente sua raiz quadrada.

- **Objetivo do algoritmo**: ler um número e apresentar sua raiz quadrada.
- **Entrada**: ler um número inteiro.
- **Processamento**: calcular a raiz quadrada.
- **Saída**: imprimir o valor da raiz quadrada.

```
Algoritmo raizquadrada
        Função raiz (X: real): real
        Início
                Retorne X*X
```

```
     Fim_função
Var
    num, resultado: real
Início
    Escreva("Informe o número:")
    Leia(num)
    resultado <- sqrt(num)
    Escreva("A raiz quadrada é :", resultado)
Fim
```

Quadro 9 - Pseudocódigo – Problema 2 / Fonte: os autores.

Problema 3

Elabore uma sub-rotina que receba três valores de entrada e retorne o maior valor.

- **Objetivo do algoritmo**: ler três números inteiros e retornar o maior valor.
- **Entrada**: ler três números inteiros.
- **Processamento**: comparar os números e selecionar o maior.
- **Saída**: imprimir o valor do maior número.

```
Algoritmo raizquadrada
     Função maior(X, Y, Z: inteiro): inteiro
     Var
          num: inteiro
     Início
          Se (x > y) então
               num <- x
          Senão
```

```
                num <- y
            Fim_se
            Se (z > num) então
                num <- z
            Fim_se
            Retorne num
        Fim_função
Var
        resultado, a, b, c: inteiro
Início
        Escreva("Informe o valor de A:")
        Leia(a)
        Escreva("Informe o valor de B:")
        Leia(b)
        Escreva("Informe o valor de C:")
        Leia(c)
        resultado <- maior(a, b, c)
        Escreva("O maior valor é:", resultado)
Fim
```

Quadro 10 - Pseudocódigo – Problema 3 / Fonte: os autores.

Problema 4

Elabore um procedimento que receba um valor em segundos e converta para horas, minutos e segundos.

- **Objetivo do algoritmo**: ler um valor em segundos e converter para horas, minutos e segundos.
- **Entrada:** ler um número inteiro.

- **Processamento:** converter os segundos para horas, dividindo por 3600. Depois, deve ser dividido por 60 (minutos) e, posteriormente, por 60, novamente, para obter os segundos.
- **Saída:** imprimir as horas, os minutos e segundos resultantes da conversão.

```
Algoritmo calculatempo
     Procedimento tempo(X: inteiro, Var h, m, s:
inteiro)
     Var
          resto: inteiro
     Início
          h <- X/3600
          resto <- X MOD 3600
          m <- resto/60
          s <- resto MOD 60
     Fim_procedimento
Var
    n, hora, minuto, segundo: inteiro
Início
    Escreva("Informe os segundos que deseja con-
    verter:")
   Leia(n)
    tempo(n, hora, minuto, segundo)
    Escreva("Horas:", hora)
    Escreva("Minutos:", minuto)
    Escreva("Segundos:", segundo)
Fim
```

Quadro 11 - Pseudocódigo – Problema 4 / Fonte: os autores.

Problema 5

Elabore uma sub-rotina que receba o valor antigo de um produto e o percentual de reajuste e retorne o valor reajustado.

- **Objetivo do algoritmo**: reajustar o valor de um produto.
- **Entrada**: ler o preço do produto e o percentual de reajuste.
- **Processamento**: calcular o reajuste do produto.
- **Saída**: imprimir o valor do produto reajustado.

```
Algoritmo produto
     Função reajuste(precop, percentualp: real):
real
     Var
          valor: real
     Início
          valor <- precop + (precop * (percen-
          tualp/100))
          Retorne valor
     Fim_função
Var
   valor, percentual, resultado: real
Início
    Escreva("Informe o preço do produto:")
    Leia(valor)
    Escreva("Informe o percentual de reajuste:")
    Leia(percentual)
    resultado <- reajuste(valor, percentual)
    Escreva("O valor reajustado é:", resultado)
Fim
```

Quadro 12 - Pseudocódigo – Problema 5 / Fonte: os autores.

CONSIDERAÇÕES FINAIS

Nesta unidade, você aprendeu a modularizar os algoritmos, utilizando sub-rotinas. A modularização é uma técnica bastante utilizada em programação devido à complexidade e ao tamanho dos problemas que temos que resolver no dia a dia. O princípio dessa técnica consiste em decompor um problema em subproblemas de menor complexidade com o objetivo de facilitar o entendimento, a análise e a resolução.

Vimos que, na resolução dos subproblemas, utilizam-se sub-rotinas. Estas consistem em blocos de instruções que realizam tarefas específicas. Aprendemos que as sub-rotinas podem ser de dois tipos (procedimentos e funções) e que, quando uma sub-rotina é chamada, ela é executada e, ao seu término, o processamento retorna para a linha seguinte à da instrução que a chamou. Estudamos que uma sub-rotina do tipo procedimento não retorna valor para quem a chamou, já uma função sempre retorna um valor.

Conhecemos o conceito de escopo de variáveis e vimos que elas podem ser locais ou globais. Uma variável local é aquela que está acessível apenas dentro da sub-rotina, enquanto que uma variável global é acessível de qualquer parte do algoritmo. Além disso, estudamos o funcionamento da passagem de parâmetros por valor e por referência. Na passagem de parâmetros por valor, não há alteração do conteúdo do parâmetro real, pois a sub-rotina trabalha com cópias dos valores passados, no momento de sua chamada. Já na passagem de parâmetros por referência, esses conteúdos são alterados, pois os parâmetros passados são endereços de memória.

Estudamos o conceito de recursividade, este é um mecanismo que permite uma função chamar a si mesma, e aprendemos que toda função recursiva é formada por um passo básico e um recursivo. Além disso, construímos funções recursivas para calcular o fatorial e a série de Fibonacci, que são problemas clássicos de recursão.

1. Desenvolva um cadastro de produtos que contenha código, descrição, unidade e preço para 20 produtos. Defina um menu com as seguintes opções:
   I - Cadastrar os 20 registros.
   II - Pesquisar um produto pelo código.
   III - Classificar por ordem de descrição os registros cadastrados.
   IV - Classificar por ordem de código.
   V - Apresentar todos os registros.
   VI - Sair do programa de cadastro.

2. Elabore um algoritmo para cadastro de dez pessoas, contendo as seguintes informações: nome, idade, altura e peso. O menu deve apresentar as seguintes opções:
   I - Cadastrar os dez registros.
   II - Ordenar os registros por idade.
   III - Apresentar a média de peso das pessoas.
   IV - Apresentar o nome das pessoas com peso acima de 50 kg.
   V - Sair do programa.

3. Escreva uma função que receba um caractere e retorne 1 se for uma consoante, número ou caractere especial, e retorne 0 se for vogal.

4. Elabore um programa com uma sub-rotina que apresente o somatório dos N primeiros números pares, definidos por um operador. O valor de N será informado pelo usuário.

5. Escreva um procedimento que efetue a leitura de um número inteiro e apresente uma mensagem informando se o número é par ou ímpar.

6. Desenvolva um programa que, por meio de uma sub-rotina, apresente como resultado o valor absoluto de um número.

7. Elabore um algoritmo que, por meio de função, efetue e apresente o valor da conversão de um valor em real de um valor lido em euros. Deverá ser solicitado, por meio do programa principal, o valor da cotação do euro e a quantidade de euro disponível.

8. Escreva um algoritmo utilizando função que converta dada temperatura lida em Celsius para Fahrenheit.

Forbellone e Eberspacher (2005) defendem que um problema muito complexo pode ser decomposto, em subproblemas menores e mais simples, no intuito de facilitar tanto o entendimento do problema quando a construção de sua respectiva solução. Assim, em seu livro "Lógica de Programação: a construção de algoritmos e estruturas de dados", os autores apontam o seguinte:

A decomposição de um problema é fator determinante para a redução da complexidade. Lembramos que Complexidade é sinônimo de Variedade, ou seja, a quantidade de situações diferentes que um problema pode apresentar. Assim, quando decompomos um problema em subproblemas, estamos invariavelmente dividindo também a complexidade e, por consequência, simplificando a resolução. Outra grande vantagem da decomposição é que permite focalizar a atenção em um problema pequeno de cada vez, o que ao final produzirá uma melhor compreensão do todo.

É conveniente que adotemos, então, um critério para orientar o processo de decomposição:

- Dividir o problema em partes principais.
- Analisar a divisão obtida para garantir coerência.
- Se alguma parte ainda permanecer complexa, decompô-la também.
- Analisar o resultado para garantir o entendimento e coerência.

Fazendo uma analogia, entender o funcionamento do corpo humano não é nada trivial. Isso por que nosso corpo é uma máquina complexa. Porém, podemos dividir esse problema em partes menores: sistema digestivo, respiratório, nervoso, cardiovascular, etc. e assim, tentando compreender cada parte separadamente, podemos compreender melhor o todo. Caso essas partes ainda seja muito complexas, podemos continuar dividindo em partes ainda menores, por exemplo, o sistema respiratório pode ser dividido em nariz, faringe, laringe, traquéia e os pulmões que, por sua vez, podem ser divididos em brônquios, bronquíolos e alvéolos, e assim por diante.

Esse processo de decomposição contínua também é conhecido como refinamentos sucessivos, porque se parte de um problema complexo e abrangente, que é sucessivamente dividido até resultar em problemas mais simples e específicos.

Nota: à técnica de Refinamentos Sucessivos também se dá o nome de Top-Down (de cima para baixo), uma vez que se parte de conceitos mais abrangentes (abstra-

tos) até atingir o nível e detalhamento desejado. Também existe uma técnica exatamente inversa, conhecida por Bottom-Up (de baixo para cima). Consiste em partir dos conceitos mais detalhados e ir agrupando-os sucessivamente em níveis mais abrangentes (abstratos) até atingir o nível de abstração desejado. O processo de compreensão é frequentemente mais natural quando se utiliza a técnica Top-Down. Por exemplo, é mais fácil compreender um automóvel partindo-se do todo até o último parafuso do que do parafuso até o todo. Certamente existem exceções. Por exemplo, é mais fácil entender operações aritméticas mais abstratas, como potenciação e radiciação, se antes soubermos somar e subtrair.

Fonte: adaptado de Forbellone e Eberspacher (2005, p. 127).

conclusão geral

Chegamos ao fim do nosso livro!

Diante dos conceitos apresentados, você deve estar mais bem preparado(a) para construir algoritmos e entender a importância que eles têm no desenvolvimento de software. Iniciamos o nosso aprendizado construindo algoritmos simples, que tinham apenas entrada e saída de dados e, conforme avançamos, passamos a construir algoritmos mais complexos, que envolviam entrada, processamento, saída de dados e modularização.

Na Unidade 1, discutimos o conceito de algoritmos, um conjunto de passos para solucionar um problema, e estudamos que, dado um problema, não há apenas um único algoritmo como solução, podemos ter vários. A Unidade 2 tratou das estruturas condicionais, também denominadas estruturas de seleção, de decisão ou estrutura de controle, que possibilitam a construção de algoritmos com desvios de fluxos. Revimos a construção de expressões lógicas e a tabela verdade de cada um dos operadores.

Na Unidade 3, aprendemos a construir algoritmos, utilizando as estruturas de repetição, laços contados e laços condicionais. A Unidade 4 discutiu as estruturas de dados homogêneas e heterogêneas. Nas estruturas de dados homogêneas, conhecemos os vetores (unidimensionais) e as matrizes (multidimensionais), que agrupam diversas informações do mesmo tipo em uma única variável. Tratamos, também, de como realizar ordenação e buscar em vetores.

Por fim, na Unidade 5, aprendemos a construir algoritmos modularizados, utilizando sub-rotinas. Estudamos o conceito de escopo de variáveis e vimos as variáveis globais e locais. Discutimos o conceito de recursividade, um mecanismo que permite uma função chamar a si mesma, e as situações em que pode ser aplicado.

Em cada uma das unidades, apresentamos e construímos algoritmos que possibilitaram a aplicação prática dos conceitos e estimularam o desenvolvimento do raciocínio lógico. Nas cinco unidades, consolidamos a visão de Algoritmos e Lógica de Programação, imprescindível para o desenvolvimento de software.

Muito sucesso a você!

ASCENCIO, A. F. G.; CAMPOS, E. A. V. **Fundamentos da Programação de Computadores**. 5. ed. São Paulo: Prentice Hall, 2010.

COSTA, E. B. L. da. A História da Ciência e o ensino da recursividade; as torres de Hanói. **História da Ciência e Ensino**, v. 4, p. 38-48, 2011.

FARRER, H. **Programação Estruturada de Computadores**. Rio de Janeiro: LTC, 1989.

FORBELLONE, A. L. V.; EBERSPACHER, H. F. **Lógica de Programação**. 3. ed. São Paulo: Makron Books, 2005.

GUIMARÃES, A. M.; LAGES, N. A. C. **Algoritmos e Estruturas de Dados.** Rio de Janeiro: LTC, 1994.

LOPES, A.; GARCIA, G. **Introdução à Programação**. Rio de Janeiro: Elsevier, 2002.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. **Estudo Dirigido de Algoritmos**. 3. ed. São Pauo: Érica, 1997.

SALVETTI, D. D.; BARBOSA, L. M. **Algoritmos**. São Paulo: Pearson Makron Books, 2004.

WIRTH, N. **Algoritmos e Estruturas de Dados**. Rio de Janeiro: LTC, 1999.

ZIVIANI, N. **Projeto de Algoritmos com Implementações em Pascal e C**. 2. ed. São Paulo: Pioneira Thomson Learning, 2004.

REFERÊNCIAS ON-LINE

1 Em: https://www.bbc.com/portuguese/geral-42908496. Acesso em: 22 nov. 2019.

2 Em: https://universoracionalista.org/deep-learning-vai-nos-ensinar-a-licao-de-nossas-vidas--o-trabalho-e-para-as-maquinas/. Acesso em: 25 nov. 2019.

3 Em: https://www.tecmundo.com.br/aumentar-desempenho/2667-como-funciona-a-memoria-ram-.htm. Acesso em: 26 nov. 2019.

UNIDADE 1

1.

- Entrar na fila.
- Aguardar até chegar a minha vez.
- Inserir o cartão no caixa eletrônico.
- Selecionar a opção de pagamento de conta.
- Ler o código de barras.
- Digitar a senha.
- Imprimir o comprovante.

2.

(C) cidade	
(I) endereço_nr	tem um caractere especial inválido 'ç'
(I) média_peso%	tem um caractere especial inválido '%'
(I) aluno-nota	sinal de operação aritmética '-'
(I) media idade	não pode conter espaço
(C) a3	
(I) endereço.cep	caracteres especiais inválidos 'ç' e '.'
(C) B5	
(C) nome2	
(I) 4cidade	não pode começar com número
(C) cliente_nome	
(I) 1234P	não pode começar com número

3.

```
Algoritmo uni1exe3
Var
      num: inteiro
Início
     Escreva("Digite o valor:")
     Leia(num)
     Escreva("O número é:", num)
     Escreva("A raiz quadrada é:", sqrt(num))
     Escreva("O quadrado é:", sqr(num))
Fim
```

4.

exec5a:	exec5b:
5	30
5	5
5	5

UNIDADE 2

1.

```
Algoritmo uni2exe1
Var
      matricula: inteiro
      nome: vetor[1..64] de caractere
      salario, vendas, total: real
Início
   Escreva("Informe a matrícula do funcionário:")
   Leia(matricula)
   Escreva("Informe o nome do funcionário:")
   Leia(nome)
   Escreva("Informe o salário fixo do funcionário:")
   Leia(salario)
   Escreva("Informe o valor das vendas do funcio-
   nário:")
   Leia(vendas)
   Se ( vendas < 1500 ) então
      total <- salario + (vendas * 0.02)
   Senão
      total <- salario + (vendas * 0.04)
   Fim_se
   Escreva("Vendedor: ", nome)
   Escreva("Matrícula: ", matricula)
   Escreva("Salário fixo: ", salario)
   Escreva("Salário total: ", total)
Fim
```

2.

```
Algoritmo uni2exe2
Var
     numero : inteiro
Início
     Escreva("Informe o número:")
     Leia(numero)
     Se (((numero MOD 3)=0) E ((numero MOD 7)=0))
então
          Escreva("O número ", numero, " é divisí-
          vel por 3 e por 7")
     Senão
          Escreva("O número ", numero, " NÃO é di-
          visível por 3 e por 7")
     Fim_se
Fim
```

3.

```
Algoritmo uni2exe3
Var
     num1, num2, num3, num4, num5, qtd : inteiro
Início
     Escreva("Informe o primeiro número:")
     Leia(num1)
     Escreva("Informe o segundo número:")
     Leia(num2)
     Escreva("Informe o terceiro número:")
     Leia(num3)
     Escreva("Informe o quarto número:")
     Leia(num4)
     Escreva("Informe o quinto número:")
     Leia(num5)
     qtd <- 0
     Se (num1 < 0) então
          qtd <- qtd + 1
     Fim_se
```

```
    Se (num2 < 0) então
        qtd <- qtd + 1
    Fim_se
    Se (num3 < 0) então
        qtd <- qtd + 1
    Fim_se
    Se (num4 < 0) então
        qtd <- qtd + 1
    Fim_se
    Se (num5 < 0) então
        qtd <- qtd + 1
    Fim_se
Fim
```

4.

```
Algoritmo uni2exe4
Var
     sexo: caractere
     altura, peso ideal: real
Início
     Escreva("Informe o sexo [M/F]:")
     Leia(sexo)
     Escreva("Informe a altura:")
     Leia(altura)
     Se (sexo = 'M') então
          peso_ideal <- (72.7 * altura) - 58
     Senão
          peso_ideal <- (62.1 * altura) - 44.7
     Fim_se
     Escreva("Seu peso ideal é: ", peso_ideal)
Fim
```

5.

```
Algoritmo uni2exe5
Var
     moto: caractere
     percurso, consumo: real
Início
     Escreva("Informe o tipo da moto [A/B/C]:")
     Leia(moto)
     Escreva("Informe o tamanho o percurso (em
     KM):")
     Leia(percurso)
     Caso (moto)
          Seja 'A' faça
               consumo <- percurso / 26
          Seja 'B' faça
               consumo <- percurso / 20
          Seja 'C' faça
               consumo <- percurso / 7
     Fim_caso
     Escreva("O consumo estimado de combustível é: ",
     consumo)
Fim
```

6.

```
Algoritmo uni2exe6
Var
     saldo: real
Início
     Escreva("Informe o saldo médio do correntis-
     ta:")
     Leia(saldo)
     Se (saldo < 500) então
          Escreva("Linha de crédito indisponí-
          vel.")
     Fim_se
     Se ((saldo >= 500) E (saldo <= 1000)) então
          Escreva("Linha de crédito disponível no
          valor de:", saldo * 0.35)
     Fim_se
     Se ((saldo > 1000) E (saldo <= 3000)) então
          Escreva("Linha de crédito disponível no
          valor de:", saldo * 0.50)
```

```
        Fim_se
        Se (saldo > 3000) então
            Escreva("Linha de crédito disponível no
            valor de:", saldo * 0.75)
        Fim_se
Fim
```

UNIDADE 3

1.

```
Algoritmo uni3exe1
Var
        qtd, contador, numero, maior, menor: inteiro
        primeiro: lógico
Início
        contador <- 0
        primeiro <- verdadeiro
       Escreva("Informe quantos números deseja ler:")
        Leia(qtd)
        Enquanto (contador < qtd) faça
            Leia(numero)
            Se (primeiro = verdadeiro) então
                maior <- numero
```

```
                menor <- numero
                primeiro <- falso
            Fim_se
            Se (numero > maior) então
                maior <- numero
            Fim_se
                menor <- numero
            Fim_se
            contador <- contador + 1
        Fim_enquanto
        Escreva("O maior número lido foi: ", maior)
        Escreva("O menor número lido foi: ", menor)
Fim
```

2.

```
Algoritmo uni3exe2
Var
    numero, contador, pares, impares: inteiro
    media: real
Início
    pares <- 0
    impares <- 0
    media <- 0
```

```
    contador <- 0
    Escreva("Digite um número para efetuar o cál
    culo ou 0 para finalizar:")
    Leia(numero)
    Enquanto (numero <> 0) faça
        Se ((numero MOD 2) = 0) então
            pares <- pares + 1
        Senão
            impares <- impares + 1
        Fim_se
        media <- media + numero
        contador <- contador + 1
        Escreva("Digite um novo número para efe
        tuar o cálculo ou 0 para finalizar:")
        Leia(numero)
    Fim_enquanto
    media <- media / contador
    Escreva("Foram lidos ", contador, " números,
    dos quais ", pares ", eram pares.")
    Escreva("Também foram encontrados ", impares,
    " números ímpares.")
    Escreva("E a média aritmética dos números li
    dos é: ",media)
Fim
```

3.

```
Algoritmo uni3exe3
Var
     num: inteiro
Início
     Para num de 0 até 199 passo 1 faça
          Se ((num MOD 5)= 0) então
               Escreva(num)
          Fim_se
     Fim_para
Fim
```

4.

```
Algoritmo uni3exe4
Var
     contador, numero, maior, menor: inteiro
     media: real
     primeiro: lógico
Início
     primeiro <- verdadeiro
     media <- 0
     contador <- 0
    Escreva("Informe o valor:")
```

```
    Leia(numero)
    Enquanto (numero >= 0) faça
        Se (primeiro = verdadeiro) então
            maior <- numero
            menor <- numero
            primeiro <- falso
        Fim_se
        Se (numero > maior) então
            maior <- numero
        Fim_se
        Se (numero < menor) então
            menor <- numero
        Fim_se
        media <- media + numero
        contador <- contador + 1
        Escreva("Informe o valor:")
        Leia(numero)
    Fim_enquanto
    media <- media / contador
    Escreva("O maior número lido foi: ", maior)
    Escreva("O menor número lido foi: ", menor)
    Escreva("A média dos números lidos é: ", media)
Fim
```

5.

```
Algoritmo uni3exe5
Var
     i, j: inteiro
Início
     Para i de 1 até 10 passo 1 faça
          Escreva("Tabuada do ", i)
          Para j de 1 até 10 passo 1 faça
               Escreva(i, " * ", j, " = ", i*j)
          Fim_para
     Fim_para
Fim
```

UNIDADE 4

1.

```
Algoritmo uni4exe1
Var
     nome: Vetor[1..20, 0..64] de caractere
     num: inteiro
Início
     Para num de 1 até 20 passo 1 faça
```

```
            Escreva("Informe o ", num, " nome: ")
            Leia(nome[num])
      Fim_para
      Para num de 1 até 20 passo 1 faça
            Escreva(nome[num][1])
      Fim_para
Fim
```

2.

```
Algoritmo uni4exe2
Var
      nome: Vetor[1..20, 1..64] de caractere
      notas: Vetor[1..20, 1..4] de real
      media: Vetor[1..20] de real
      i, j: inteiro
      media_turma: real
Início
      media_turma <- 0
      Para i de 1 até 20 passo 1 faça
            Escreva("Informe o ", i, " nome: ")
            Leia(nome[i])
            media[i] <- 0
```

```
            Para j de 1 até 4 passo 1 faça
                Escreva("Informe a nota", j)
                Leia(notas[i, j])
                media[i] <- media[i] + notas[i, j]
            Fim_para
            media[i] <- media[i] / 4
            media_turma <- media_turma + media[i]
        Fim_para
        media_turma <- media_turma / 20
        Para i de 1 até 20 passo 1 faça
            Escreva("Aluno", nome[i])
            Para j de 1 até 4 passo 1 faça
                Escreva("Nota", j, " é:", notas[i, j])
            Fim_para
            Escreva("A média do aluno", nome[i],
            "foi:", media[i])
        Fim_para
        Escreva("A média da turma é:", media_turma)
Fim
```

3.

```
Algoritmo uni4exe3
Var
     nome: Vetor[1..64] de caractere
Início
     Repita
          Escreva("Digite o nome, digite FIM para
          sair:")
          Leia(nome)
          Escreva(nome[1])
     Até_que (nome = "FIM")
Fim
```

4.

```
Algoritmo uni4exe4
Var
     num, maior, menor: inteiro
     numeros: Vetor[1..50] de inteiro
     primeiro: lógico
Início
     primeiro <- verdadeiro
     Escreva("Digite os 50 números")
```

```
    Para num de 1 até 50 passo 1 faça
        Leia(numeros[num])

        Se (primeiro = verdadeiro) então
            maior <- numeros[num]
            menor <- numeros[num]
            primeiro <- falso
        Fim_se
        Se (numeros[num] > maior) então
            maior <- numeros[num]
        Fim_se
        Se (numeros[num] < menor) então
            menor <- numeros[num]
        Fim_se
    Fim_para
    Escreva("O maior número lido foi: ", maior)
    Escreva("O menor número lido foi: ", menor)
Fim
```

5.

```
Algoritmo uni4exe5
Var
     num: inteiro
     a, b, c : Vetor[1..20] de inteiro
Início
     Escreva("Digite os 20 números do Vetor A")
     Para num de 1 até 20 passo 1 faça
          Escreva("Digite o número ", num)
          Leia(a[num])
     Fim_para
     Escreva("Digite os 20 números do Vetor B")
     Para num de 1 até 20 passo 1 faça
          Escreva("Digite o número ", num)
          Leia(b[num])
     Fim_para
     Escreva("Calculando o vetor C = A + B")
     Para num de 1 até 20 passo 1 faça
          c[num] <- a[num] + b[num]
     Fim_para
     Escreva("Os valores do vetor C = A + B são: ")
     Para num de 1 até 20 passo 1 faça
```

```
            Escreva(num, ") ", c[num])
      Fim_para
Fim
```

6.

```
Algoritmo uni4exe6
      Tipo
            cad_produto = registro
                  nome: Vetor[1..64] de caractere
                  p_compra: real
                  p_venda: real
      Fim_registro
Var
      num, lucro_maior, lucro_menor: inteiro
      produto: Vetor[1..30] de cad_produto
Início
      lucro_maior <- 0
      lucro_menor <- 0
      Escreva("Cadastre os 30 produtos ")
      Para num de 1 até 30 passo 1 faça
            Escreva("Digite o nome do produto ", num)
            Leia(produto[num].nome)
```

```
            Escreva("Digite o preço de compra do
            produto ", produto[num].nome)
            Leia(produto[num].p_compra)
            Escreva("Digite o preço de venda do
            produto ", produto[num].nome)
            Leia(produto[num].p_venda)
            Se (((produto[num].p_venda-produto[num].
            p_compra)/ produto[num].p_compra)
            < 0.15) então
                lucro_menor <- lucro_menor + 1
            Fim_se
            Se (((produto[num].p_venda-produto[num].
            p_compra)/ produto[num].p_compra)
            > 0.30) então
                lucro_maior <- lucro_maior + 1
            Fim_se
        Fim_para
        Escreva("Existem ", lucro_menor, " produtos
        com lucro menor do que 15%")
        Escreva("Existem ", lucro_maior, " produtos
        com lucro maior do que 30%")
Fim
```

7.

```
Algoritmo uni4exe7
Var
     i, j: inteiro
     matriz: Vetor[1..5, 1..5] de inteiro
Início
     Escreva("Entre com os valores da matriz 5x5 ")
     Para i de 1 até 5 passo 1 faça
          Para j de 1 até 5 passo 1 faça
               Escreva("Digite o valor ", i, " x ",j)
               Leia(matriz[i, j])
          Fim_para
     Fim_para
     Escreva("Os valores da diagonal da matriz ",
     i, " x ", j,"são:")
     Para i de 1 até 5 passo 1 faça
          Escreva("Posição ", i, " x ", i, " = "
          matriz[i, i])
     Fim_para
Fim
```

8.

```
Algoritmo uni4exe8
     Tipo
          cad_aluno = registro
               nome: Vetor[1..64] de caractere
               notas: Vetor[1..4] de real
               media: real
     Fim_registro
Var
     i, j: inteiro
     alunos: Vetor[1..5] de cad_aluno
     temp: cad_aluno
Início
     Escreva("Entre com os dados dos alunos")
     Para i de 1 até 5 passo 1 faça
          Escreva("Digite o nome do aluno: ")
          Leia(alunos[i].nome)
          alunos[i].media <- 0
          Para j de 1 até 4 passo 1 faça
               Escreva("Digite a nota ", j )
               Leia(alunos[i].notas[j])
```

```
            alunos[i].media <- alunos[i].media +
                 alunos[i].notas[j]
            Fim_para
            alunos[i].media <- alunos[i].media/4
      Fim_para
      Para i de 1 até 5 passo 1 faça
            Para j de i+1 até 5 passo 1 faça
                Se (alunos[i].media > alunos[j].
                media) então
                      temp <- alunos[i]
                      alunos[i] <- alunos[j]
                      alunos[j] <- temp
                Fim_se
            Fim_para
      Fim_para
      Escreva("A classifcação dos alunos por média
      crescente é:")
      Para i de 1 até 5 passo 1 faça
            Escreva(alunos[i].nome, "média: ",
            alunos[i].media)
      Fim_para
Fim
```

9.

```
Algoritmo uni4exe9
Var
     i, j: inteiro
     a, b, soma, dif : Vetor[1..5, 1..5] de inteiro
Início
     Escreva("Entre com os valores da primeira
     matriz")
     Para i de 1 até 5 passo 1 faça
          Para j de 1 até 5 passo 1 faça
               Escreva("Digite o valor ", i, " x ", j)
               Leia(a[i, j])
          Fim_para
     Fim_para
     Escreva("Entre com os valores da segunda matriz")
     Para i de 1 até 5 passo 1 faça
          Para j de 1 até 5 passo 1 faça
               Escreva("Digite o valor ", i, " x ", j)
               Leia(b[i, j])
               soma[i,j] <- a[i, j] + b[i, j]
               dif[i,j] <- a[i, j] - b[i, j]
          Fim_para
```

```
        Fim_para
        Escreva("A matriz resultante da soma das ma
        trizes:")
        Para i de 1 até 5 passo 1 faça
            Para j de 1 até 5 passo 1 faça
                Escreva("Posição ", i, " x ", j,
                soma[i, j])
            Fim_para
        Fim_para
        Escreva("A matriz resultante da diferença das
        matrizes:")
        Para i de 1 até 5 passo 1 faça
            Para j de 1 até 5 passo 1 faça
                Escreva("Posição ", i, " x ", j,
                dif[i, j])
            Fim_para
        Fim_para
Fim
```

10.

```
Algoritmo uni4exe10
Var
     i, j: inteiro
     contas: Vetor[1..12, 1..4] de real
     mensal: Vetor[1..12] de real
     total: real
Início
     Escreva("Entre com os valores das contas a
     pagar")
     total <- 0
     Para i de 1 até 12 passo 1 faça
          mensal[i] <- 0
          Para j de 1 até 4 passo 1 faça
               Escreva("Digite o valor do mês ",
               i, ", semana ", j)
               Leia(contas[i, j])
          mensal[i] <- mensal[i] + contas[i, j]
          Fim_para
          total <- total + mensal[i]
     Fim_para
     Escreva("O total de contas a pagar, por mês é:")
```

```
    Para i de 1 até 12 passo 1 faça
        Escreva("Total a pagar no mês ", i,
        " é : ", mensal[i])
    Fim_para
    Escreva("O total de contas a pagar no ano
    é:", total)
Fim
```

UNIDADE 5

1.

```
Algoritmo cadastros
    Tipo
        CadProduto = registro
            codigo: inteiro
            descricao: Vetor[1..50] de carac-
            tere
            unidade: Vetor[1..5] de caractere
            preco: real
    Fim_registro

    Procedimento mostrarMenu()
    Início
        Escreva("1 - Cadastrar registro")
```

```
        Escreva("2 - Pesquisar um produto pelo
        código")
        Escreva("3 - Classificar por ordem de
        descrição")
        Escreva("4 - Classificar por ordem de
        código")
        Escreva("5 - Apresentar todos os
        registros")
        Escreva("6 - Sair do programa de
        cadastro")
    Fim_procedimento

    Procedimento produtosCadastrar()
    Início
        Se (qtdProd < 20) então
            Escreva("Digite o código: ")
            Leia(produtos[qtdProd+1].codigo)
            Escreva("Digite a descrição: ")
            Leia(produtos[qtdProd+1].descricao)
            Escreva("Digite a unidade: ")
            Leia(produtos[qtdProd+1].unidade)
            Escreva("Digite o preço: ")
```

```
            Leia(produtos[qtdProd+1].preco)
            qtdProd <- qtdProd + 1
            Escreva("Produto cadastrado com
            sucesso!")
        Senão
            Escreva("Limite máximo atingido:
            20 produtos!")
        Fim_se
        opt <- 0
    Fim_procedimento

    Função produtosBuscar(cod: inteiro): inteiro
    Var
        i, achou: inteiro
    Início
        i <- 1
        achou <- 0
        Enquanto (i <= qtdProd) faça
            Se (produtos[i].codigo == cod) então
                Escreva("Produto: ", cod)
                Escreva("Descrição: ",
                produtos[i].descricao)
```

```
                    Escreva("Unidade: ",
                    produtos[i].unidade)
                    Escreva("Preço:   ",   produtos
                    [i].preco)
                    Retorne i
               Fim_se
               i <- i + 1
          Fim_enquanto
          Escreva("Produto não encontrado...")
          Retorne -1
     Fim_função
     Procedimento produtosOrdenarDescr()
     Var
          i, j, k: inteiro
          aux: CadProduto
     Início
          Para i de 1 até qtdProd passo 1 faça
               Para j de i+1 até qtdProd passo
               1 faça
                    k <- 1
                    Enquanto ((produtos[i].des-
                    cricao[k] = produtos[j].des-
                    cricao[k]) e (k <= 50)) faça
```

```
                    k <- k + 1
                Fim_enquanto
                Se ((produtos[i].descricao[k]
                > produtos[j].descricao[k]) e
                (k <= 50)) então
                    aux <- produtos[i]
                produtos[i] <- produtos[j]
                    produtos[j] <- aux
                Fim_se
            Fim_para
        Fim_para
    Fim_procedimento

    Procedimento produtosOrdenarCod()
    Var
        i, j: inteiro
        aux: CadProduto
    Início
        Para i de 1 até qtdProd passo 1 faça
            Para j de i+1 até qtdProd passo 1
            faça
                Se (produtos[i].codigo >
                produtos[j].codigo)
```

```
                    então
                         aux <- produtos[i]
                        produtos[i] <- produtos[j]
                         produtos[j] <- aux
                    Fim_se
               Fim_para
          Fim_para
     Fim_procedimento

     Procedimento produtosMostrar()
     Var
          i: inteiro
     Início
          Para i de 1 até qtdProd passo 1 faça
               Escreva("Produto:  ", produtos[i].
               codigo)
               Escreva("Descrição: ", produtos[i].
               descricao)
               Escreva("Unidade:  ", produtos[i].
               unidade)
               Escreva("Preço:   ", produtos[i].
               preco)
```

```
            Fim_para
      Fim_procedimento
Var
      produtos: Vetor[1..20] de CadProduto
      opt, qtdProd, i: inteiro
Início
      qtdProd <- 0
      Repita
            mostrarMenu()
            Leia(opt)
            Caso (opt)
                  Seja 1 faça
                        Para i de 1 até 20 passo 1 faça
                              produtosCadastrar()
                        Fim_para
                  Seja 2 faça
                        produtosBuscar()
                  Seja 3 faça
                        produtosOrdenarDescr()
                  Seja 4 faça
                        produtosOrdenarCod()
                  Seja 5 faça
                        produtosMostrar()
```

```
            Fim_caso
      Até_que (opt = 6)
Fim
```

2.

```
Algoritmo cadastroPessoas
      Tipo
            CadPessoa = registro
                  nome: Vetor[1..50] de caractere
                  idade: inteiro
                  altura: real
                  peso: real
      Fim_registro

      Procedimento mostrarMenu()
      Início
            Escreva("1 - Cadastrar pessoa")
            Escreva("2 - Classificar por ordem de
            idade")
            Escreva("3 - Calcular média de peso")
            Escreva("4 - Mostrar quem está acima do
            limite de 50")
            Escreva("5 - Sair do programa de cadas-
            tro")
```

```
Fim_procedimento

Procedimento pessoaCadastrar()
Início
    Se (qtdPessoas < 10) então
        Escreva("Insira o nome: ")
        Leia(pessoas[qtdPessoas+1].nome)
        Escreva("Insira a idade: ")
        Leia(pessoas[qtdPessoas+1].idade)
        Escreva("Insira a altura: ")
        Leia(pessoas[qtdPessoas+1].altura)
        Escreva("Insira o peso: ")
        Leia(pessoas[qtdPessoas+1].peso)
        qtdPessoas <- qtdPessoas + 1
    Senão
        Escreva("Limite máximo de 10
        pessoas atingido")
    Fim_se
Fim_procedimento
Procedimento pessoasOrdenarIdade()
Var
    i, j: inteiro
```

```
        aux: CadPessoa
    Início
        Para i de 1 até qtdPessoas passo 1 faça
            Para j de i+1 até qtdPessoas passo
            1 faça
                Se (pessoas[i].idade >
                pessoas[j].idade) então
                    aux <- pessoas[i]
                    pessoas[i] <- pessoas[j]
                    pessoas[j] <- aux
                Fim_se
            Fim_para
        Fim_para
    Fim_procedimento


    Função mediaPesos(): real
    Var
        i: inteiro
        media: real
    Início
        media <- 0
        Para i de 1 até qtdPessoas passo 1 faça
```

```
                media <- media + pessoas[i].peso
            Fim_para
            Retorne media / qtdPessoas
        Fim_função

        Procedimento   nomePessoasPesoMaiorQue(peso:
        real)
        Var
            i: inteiro
        Início
            Para i de 1 até qtdPessoas passo 1 faça
                Se (pessoas[i].peso > peso) então
                    Escreva(i, " - Nome: ",
                    pessoas[i].nome)
                Fim_se
            Fim_para
        Fim_procedimento

Var
    pessoas: Vetor[1..10] de CadPessoa
    opt, qtdPessoas: inteiro
```

```
Início
     qtdPessoas <- 0
     Repita
          mostrarMenu()
          Leia(opt)
          Caso (opt)
               Seja 1 faça
                    Para i de 1 até 10 passo 1
                    faça pessoaCadastrar()
                    Fim_para
               Seja 2 faça
                    pessoasOrdenarIdade()
               Seja 3 faça
                    Escreva("Média dos pesos: ",
                    mediaPesos())
               Seja 4 faça
                    nomePessoasPesoMaiorQue(50)
          Fim_caso
     Até_que (opt = 5)
Fim
```

3.

```
Função testaCaractere(c: caractere): inteiro
Início
     Se ((c = 'A') ou (c = 'a') ou (c = 'E') ou
     (c = 'e')
     ou (c = 'I') ou (c = 'i') ou (c = 'O') ou
     c = 'o')
     ou (c = 'U') ou (c = 'u')) então
          Retorne 0
     Senão
          Retorne 1
     Fim_se
Fim_função
```

4.

```
Algoritmo somatorio
     Função somador(limite: inteiro): inteiro
          Var soma, i: inteiro
     Início
          soma <- 0
          Para i de 1 até limite passo 1 faça
               Se ((i MOD 2) = 0) então
                    soma <- soma + i
```

```
                Fim_se
            Fim_para
            Retorne soma
        Fim_função
Var
        num: inteiro
Início
        Escreva("Gostaria de somar os inteiros pares
        até qual limite?")
        Leia(num)
        Escreva("O somatório é: ", somador(num))
Fim
```

5.

```
Procedimento parOuImpar(num: inteiro)
Início
        Se ((num MOD 2) = 0) então
            Escreva("O número ", num, " é par.")
        Senão
            Escreva("O número ", num, " é ímpar.")
        Fim_se
Fim_procedimento
```

6.

```
Algoritmo valorAbsoluto
     Função abs(num: real): real
     Início
          Se (num < 0) então
               Retorne num * (-1)
          Senão
               Retorne num
          Fim_se
     Fim
Var
     x: real
Início
     Escreva("Digite um número real qualquer:")
     Leia(x)
     Escreva("O valor absoluto é: ", abs(x))
Fim
```

7.

```
Algoritmo conversao
     Função converteParaReais(c: real, q: real):
real
     Início
          Retorne c * q
     Fim_função
Var
```

```
    cot, qtde: real
Início
    Escreva("Qual é a cotação atual do Euro?")
    Leia(cot)
    Escreva("Quantos Euros você deseja conver-
    ter?")
    Leia(qtde)
    Escreva("O valor em reais é: ",
    converteParaReais(cot, qtde))
Fim
```

8.

```
Algoritmo conversao
    Função converteParaFahrenheit(c: real): real
    Início
        Retorne (c * 1.8) - 32
    Fim_função
Var
    celcius: real
Início
    Escreva("Qual a temperatura em °C?")
    Leia(celcius)
    Escreva("Convertendo para °F: ",
    converteParaFahrenheit(celcius))
Fim
```