Lógica de Programação

Liluyoud Cury de Lacerda
José Marcio Benite Ramos
Sara Luize Oliveira Duarte

Cuiabá-MT
2014

Presidência da República Federativa do Brasil
Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
Diretoria de Integração das Redes de Educação Profissional e Tecnológica

© Este caderno foi elaborado pelo Instituto Federal de Educação, Ciência e Tecnologia de Rondônia-RO, para a Rede e-Tec Brasil, do Ministério da Educação em parceria com a Universidade Federal de Mato Grosso.

Equipe de Revisão

Universidade Federal de Mato Grosso – UFMT

Coordenação Institucional
Carlos Rinaldi

Coordenação de Produção de Material Didático Impresso
Pedro Roberto Piloni

Designer Educacional
Neusa Blasques

Ilustração
Tatiane Hirata

Diagramação
Tatiane Hirata

Revisão de Língua Portuguesa
Verônica Hirata

Revisão Final
Neusa Blasques

Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO

Campus Porto Velho Zona Norte

Direção-Geral
Miguel Fabrício Zamberlan

Direção de Administração e Planejamento
Gilberto Laske

Departamento de Produção de EaD
Ariádne Joseane Felix Quintela

Coordenação de Design Visual e Ambientes de Aprendizagem
Rafael Nink de Carvalho

Coordenação da Rede e-Tec
Ruth Aparecida Viana da Silva

Projeto Gráfico
Rede e-Tec Brasil / UFMT

Lógica de Programação - Informática para Internet

L131l Lacerda, Liluyoud Cury de.

Lógica de programação /Liluyoud Cury de Lacerda, José Marcio Benite Ramos, Sara Luize Oliveira Duarte. – Cuiabá: Ed.UFMT, 2014.

117 p.
Curso Técnico – Rede E- Tec. (IFRO)

1. Lógica - Informática. 2. Algoritmos – Informática. I. Título.

CDU 004.42

Ficha Catalográfica Elaborada por Almira de Araújo Medeiros – CRB1 2.327)

Apresentação Rede e-Tec Brasil

Prezado(a) estudante,

Bem-vindo(a) à Rede e-Tec Brasil!

Você faz parte de uma rede nacional de ensino que, por sua vez, constitui uma das ações do Pronatec - Programa Nacional de Acesso ao Ensino Técnico e Emprego. O Pronatec, instituído pela Lei nº 12.513/2011, tem como objetivo principal expandir, interiorizar e democratizar a oferta de cursos de Educação Profissional e Tecnológica (EPT) para a população brasileira propiciando caminho de acesso mais rápido ao emprego.

É neste âmbito que as ações da Rede e-Tec Brasil promovem a parceria entre a Secretaria de Educação Profissional e Tecnológica (Setec) e as instâncias promotoras de ensino técnico, como os institutos federais, as secretarias de educação dos estados, as universidades, as escolas e colégios tecnológicos e o Sistema S.

A educação a distância no nosso país, de dimensões continentais e grande diversidade regional e cultural, longe de distanciar, aproxima as pessoas ao garantir acesso à educação de qualidade e ao promover o fortalecimento da formação de jovens moradores de regiões distantes, geográfica ou economicamente, dos grandes centros.

A Rede e-Tec Brasil leva diversos cursos técnicos a todas as regiões do país, incentivando os estudantes a concluir o ensino médio e a realizar uma formação e atualização contínuas. Os cursos são ofertados pelas instituições de educação profissional e o atendimento ao estudante é realizado tanto nas sedes das instituições quanto em suas unidades remotas, os polos.

Os parceiros da Rede e-Tec Brasil acreditam em uma educação profissional qualificada – integradora do ensino médio e da educação técnica – capaz de promover o cidadão com capacidades para produzir, mas também com autonomia diante das diferentes dimensões da realidade: cultural, social, familiar, esportiva, política e ética.

Nós acreditamos em você!
Desejamos sucesso na sua formação profissional!

Ministério da Educação
Abril de 2014

Nosso contato
etecbrasil@mec.gov.br



Indicação de Ícones

Os ícones são elementos gráficos utilizados para ampliar as formas de linguagem e facilitar a organização e a leitura hipertextual.



Atenção: indica pontos de maior relevância no texto.



Saiba mais: oferece novas informações que enriquecem o assunto ou "curiosidades" e notícias recentes relacionadas ao tema estudado.



Glossário: indica a definição de um termo, palavra ou expressão utilizada no texto.



Mídias integradas: remete o tema para outras fontes: livros, filmes, músicas, *sites*, programas de TV.



Atividades de aprendizagem: apresenta atividades em diferentes níveis de aprendizagem para que o estudante possa realizá-las e conferir o seu domínio do tema estudado.



Reflita: momento de uma pausa na leitura para refletir/escrever sobre pontos importantes e/ou questionamentos.



Palavra dos Professores-autores

Caro(a) estudante,

Você está prestes a entrar em um mundo fantástico onde homens e máquinas interagem para resolver problemas que antes eram difíceis e até mesmo impossíveis. Ele tem evoluído de maneira alucinante e alguns o consideram mágico: é o mundo das Tecnologias de Informação e Comunicação – TICs, ou, simplesmente, TI.

Tecnologias de Informação é um termo abrangente que engloba várias áreas, e neste caderno vamos abordar uma das mais importantes, que é a Programação.

A importância dessa área é justificada pelo fato de a lógica de programação ser a base de toda a informática. Por meio dela os circuitos lógicos são programados, são criados os sistemas operacionais e todos os aplicativos que usamos quando estamos em frente a qualquer dispositivo eletrônico, sejam celulares, *tablets*, computadores e muitos outros.

Neste caderno serão abordados os conceitos básicos da Lógica e da Linguagem de Programação, como fazer com que o computador execute os comandos que desejamos para executar uma tarefa específica e, principalmente, entender como é todo esse processo.

A Lógica de Programação pode ser considerada como a porta de entrada para esse mundo fantástico que permite uma profunda interação entre o homem e a máquina, no caso, o computador. Sendo assim, fica fácil entender por que muitos ficam apaixonados e maravilhados com essa área específica da informática.

Bons estudos.

Professores-autores.



Apresentação da Disciplina

A disciplina de Lógica de Programação compreende o fundamento teórico e prático necessário para desenvolver programas, apresentando as principais estruturas de dados e de linguagem necessárias para a resolução de problemas computacionais.

As aulas estão estruturadas de forma a capacitá-lo(a) na construção de algoritmos e programas a partir do entendimento da lógica para resolução de problemas de complexidade limitada.

O objetivo da disciplina é fazê-lo(a) compreender os mecanismos lógicos e matemáticos da programação, relacionando problemas e elaborando soluções através do uso de metodologias e ferramentas que envolvam os elementos básicos da construção de algoritmos e programas de computador.

Para alcançar esse objetivo, a disciplina foi estrutura em dez aulas. A primeira aborda a lógica no dia a dia e na informática. Na segunda, trataremos dos algoritmos e as formas de representá-lo. Na terceira aula, mostraremos as ferramentas usadas para a criação dos algoritmos. Já na quarta aula, discorreremos sobre a estrutura básica de um algoritmo, em pseudocódigo com VisualG, e em Java com Eclipse. Na sequência, veremos o conceito de variáveis, sua nomenclatura e os tipos primitivos de dados. Na sexta aula, estudaremos os conceitos de operadores e expressões, como empregar operadores aritméticos, relacionais e lógicos para criar expressões complexas e, finalmente, nas quatro últimas aulas, trataremos das estruturas de sequência, decisão, seleção e repetição.

Por fim, podemos dizer que um programa nada mais é do que instruções para armazenar, processar e recuperar dados em computador, e as técnicas usadas para escrever essas instruções influi diretamente na qualidade do programa.

É essencial que você dedique uma parcela significativa de seu tempo para assimilar todo o conteúdo abordado, pois esta disciplina é a base para as próximas disciplinas relacionadas a programação, e mais ainda, é a porta de entrada do mundo das Tecnologias de Informação e Comunicação.

Bons estudos!



Sumário

Aula 1. Lógica **15**
1.1 O que é lógica? 15
1.2 Lógica no dia a dia 16
1.3 Lógica na Informática 17
Aula 2. Algoritmos **21**
2.1 O que é um algoritmo? 21
2.2 Tipos de algoritmos 22
Aula 3. Ferramentas de criação de algoritmos **27**
3.1 Introdução 27
3.2 Plataforma de estudo 28
Aula 4. Estrutura básica de um algoritmo **33**
4.1 Introdução 33
4.2 Como definir um algoritmo 34
Aula 5. Variáveis e tipos de dado **43**
5.1 Introdução 43
5.2 Variáveis 43
5.3 Tipos de dado 45
5.4 Sintaxe e exemplos de declaração 46
Aula 6. Operadores e expressões **51**
6.1 Introdução 51
6.2 Operadores e expressões 52
6.3 Ordem de precedência 55
Aula 7. Estruturas sequenciais **61**
7.1 Introdução 61
7.2 Estrutura sequencial 62
7.3 Finalmente um algoritmo completo 67

Aula 8. Estruturas de decisão **71**
8.1 Introdução 71
8.2 Tipos de estruturas de decisão 71

Aula 9. Estruturas de seleção **83**
9.1 Introdução 83
9.2 Sintaxe do comando escolha 83

Aula 10. Estruturas de repetição **89**
10.1 Introdução 89
10.2 Tipos de estrutura de repetição 90
10.3 Comparação entre as estruturas de repetição 99

Palavras Finais **101**
Guia de Soluções **102**
Referências **120**
Obras Consultadas **120**
Bibliografia Básica **121**
Currículo dos Professores-autores **122**

Aula 1. Lógica

Objetivos:

- conceituar raciocínio lógico;
- diferenciar argumentos indutivos de argumentos dedutivos; e
- utilizar lógica no desenvolvimento de programas.

Caro(a)estudante,

Usamos a razão e o senso comum, mesmo que intuitivamente, na execução das tarefas do dia a dia, seja nas tomadas de decisão ou na resolução de problemas. No decorrer da aula, iremos apresentar os conceitos fundamentais por trás dessa razão, que a partir de agora chamaremos de lógica. Veremos também como aplicá-la na resolução de problemas computacionais.

1.1 O que é lógica?

Segundo o dicionário da Porto Editora, a palavra *lógica* é srcinária da palavra grega *logiké*, que significa "a arte de raciocinar". Disponível em: <http://www.infopedia.pt/lingua-portuguesa/> Acesso em: 03 set. 2013.

A lógica pode ser obtida a partir do encadeamento regular ou coerente das ideias e das coisas.

Em outras palavras, a lógica se preocupa com a maneira que os pensamentos e as ideias são organizados e apresentados, possibilitando-se concluir algo por meio do encadeamento de argumentos.

Segundo Puga e Rissetti (2009), os argumentos podem ser dedutivos ou indutivos. Os argumentos indutivos são aqueles que, a partir dos dados, conduzem a uma resposta ou conclusão por meio da analogia. Por exemplo:

Na última prova estudei apenas duas horas e me dei bem. Logo, se estudar duas horas para a prova de hoje, certamente me darei bem também.

Observe que esse tipo de raciocínio não dá certeza de que o resultado será de fato o esperado.

Os argumentos dedutivos são aqueles cuja resposta é obtida a partir de uma sequência de premissas, dadas pela análise de situações e fatos. Por exemplo:

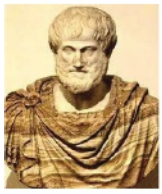
O autor do livro é um professor. O professor trabalha muito. Logo, o autor do livro trabalha muito.

Podemos perceber, agora, que, a partir de uma série de premissas (análise de fatos), podemos concluir algo verdadeiro.

O exemplo anterior identifica um silogismo que representa um argumento composto por duas premissas e uma conclusão. Estabelece uma relação em que, mesmo que haja uma aparente coerência no encadeamento lógico, pode ou não ser válida.

Então, podemos dizer que o objetivo da lógica é deduzir, analisar, formalizar argumentos e verificar sua validade com base na razão.

O filósofo grego Aristóteles (384 a 322 a. C.), aluno de Platão, é considerado o criador da lógica, mas ele não a chamava assim -- chamava-a de razão. O termo lógica só passou a ser utilizado bem depois.



Fonte: www.ahistoria.com.br

1.2 Lógica no dia a dia

Faz parte da natureza humana analisar e executar ações passo a passo para a realização de suas atividades – na maioria das vezes nem percebemos que estamos "pensando logicamente".

O ordenamento natural do pensamento para a execução de uma tarefa é chamado de raciocínio lógico. Como exemplo, podemos citar:

Tarefa: vestir-se.
O armário está fechado.
A roupa está dentro do armário.
Primeiro preciso abrir o armário para pegar a roupa.
Depois de pegar a roupa, posso me vestir.

Perceba, que mesmo inconscientemente, tomamos decisões e realizamos ações seguindo uma ordem lógica para conseguir o que queremos. A capacidade humana de raciocinar logicamente é que o tornou capaz de criar tantas maravilhas com as quais convivemos hoje, inclusive o computador. Agora, vamos nos ater ao estudo da lógica aplicada na informática.

1.3 Lógica na Informática

A lógica pode ser aplicada em qualquer ciência, tal como Filosofia, Psicologia, Sociologia, Matemática, Física etc., mas o nosso foco é como aplicá-la na Informática, mas especificamente na construção de programas.

Os seres humanos têm a capacidade de expressar a lógica através da palavra falada ou escrita, obedecendo às regras gramaticais da linguagem usada – no nosso caso, o português.

Entretanto, a linguagem natural pode ser um tanto cansativa, repetitiva, ambígua e suscetível a diversas interpretações e argumentações informais e, por esse motivo, faz-se necessária a criação de uma representação mais formal da lógica.

Na formalização da lógica, é possível trabalharmos com variáveis, o que possibilita aplicar o mesmo raciocínio lógico a problemas diferentes. Por exemplo:

João passou no vestibular.
Todos que passam no vestibular são estudiosos.
Logo, João é estudioso.

Substituindo "João" e "estudioso", pelas variáveis "X" e "Y", temos que:

X passou no vestibular.
Todos que passam no vestibular são Y.
Logo, X é Y.

Perceba que X e Y, por serem termos genéricos, podem ser aplicados em outros contextos:

Para X = José e Y = inteligente, temos:
José passou no vestibular.
Todos que passam no vestibular são inteligentes.
Logo, José é inteligente.

Ou,

Para X = Pedro e Y = sortudo, temos:
Pedro passou no vestibular.
Todos que passam no vestibular são sortudos.
Logo, Pedro é sortudo.



O conceito de variáveis será amplamente discutido na Aula 5.

Para a resolução de problemas computacionais, usamos esse tipo de raciocínio, em que informações a serem analisadas são representadas por termos genéricos (variáveis), os quais posteriormente serão substituídos por valores específicos, possibilitando a resolução de diversos problemas com a mesma lógica.

Resumo

Você teve o primeiro contato com os conceitos de lógica e constatou que ela é a base para a resolução de problemas computacionais. Vimos, também, que, para realizarmos uma atividade com sucesso (resolução de um problema), executamos vários passos de acordo com uma ordem, seguindo um raciocínio lógico, raciocínio esse que pode ser representado formalmente para resolver outros problemas semelhantes.



Atividades de aprendizagem

1. A partir do que leu nesta aula, explique com suas palavras a importância da lógica para a Informática.

2. Dê um exemplo de um argumento indutivo cujo resultado é verdadeiro e de outro que seja falso.

3. Dê um exemplo de um argumento dedutivo diferente do que foi mostrado nesta aula.

4. De acordo com os silogismos a seguir, assinale a afirmação que apresente uma conclusão válida:

I – Gatos são mamíferos. Mamíferos não botam ovo. Logo:

a) Todos os mamíferos são gatos.

b) Todos os gatos não botam ovo.

c) Os gatos têm mais de um filhote.

II – Hexágonos são figuras que possuem ângulos. Temos uma figura que não tem ângulo. Logo:

a) Essa figura pode ser uma linha.

b) Essa figura não é um hexágono.

c) Não é possível tirar conclusões.

III – Você está dirigindo sua moto. Se frear rapidamente, um carro baterá em sua traseira. Se não frear rapidamente, você atropelará uma senhora que está atravessando a rua. Logo:

a) As senhoras não devem andar na rua.

b) O carro ou você estão em alta velocidade.

c) O carro baterá em sua traseira ou você atropelará a senhora.

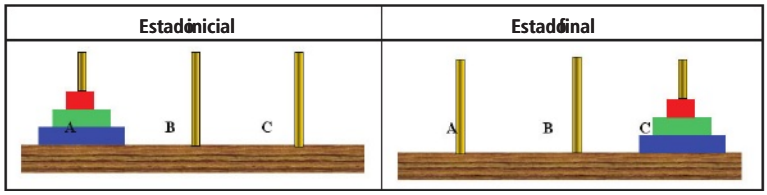
5. Três senhoras: Dona Branca, Dona Rosa e Dona Violeta, passeavam pelo parque, quando Dona Rosa disse:

– Não é curioso que estejamos usando vestidos de cores branca, rosa e violeta, embora nenhuma de nós esteja usando um vestido de cor igual ao seu próprio nome?

– Uma simples coincidência – respondeu a senhora com vestido violeta.

Qual a cor do vestido de cada senhora?

6. Analise e descreva uma maneira de mover os discos do estado inicial para o estado final, considerando que só é possível movimentar um disco por vez, e em hipótese alguma um disco maior poderá ficar sobre um menor.



Fonte: autores

Caro(a) estudante,

Finalizamos a nossa primeira aula. Nela, trabalhamos conceitos importantes para o entendimento das próximas. Esperamos que esteja motivado(a) para seguir em frente. Dando continuidade aos estudos, na próxima aula trataremos de um assunto muito importante para a lógica de programação: os algoritmos. Vamos lá!

Aula 2. Algoritmos

Objetivos:

- conceituar algoritmos;
- identificar os três tipos de algoritmos estudados, suas vantagens e desvantagens; e
- escrever algoritmo nas formas de descrição narrativa, fluxograma e pseudocódigo.

Caro(a) estudante,

A base do desenvolvimento dos programas modernos é o algoritmo. Nesta aula, você irá conhecer os principais conceitos relacionados a ele e terá a oportunidade de aprender a resolver os mais diversos problemas computacionais. Para tanto, deverá aprender a pensar de forma algorítmica, isto é, pensar em ordem, da maneira que vimos na aula anterior.

2.1 O que é um algoritmo?

De acordo com Forbellone (2005), um **algoritmo** pode ser definido como uma sequência de passos que visam atingir um objetivo bem definido. Segundo o dicionário Porto Editora, o sentido matemático e lógico da palavra **algoritmo** é um conjunto de regras bem definidas para resolver um problema com um número finito de passos.

Os algoritmos são amplamente utilizados em disciplinas ligadas à área de ciências exatas, por exemplo:

É possível descrever a resolução de uma equação de 2º grau (Bhaskara) em uma sequência de passos que levam ao resultado esperado.

A descrição dessa resolução de forma genérica é o que chamamos de algoritmo.

Geralmente, utilizamos o pensamento algorítmico de maneira intuitiva. Por exemplo, quando vamos trocar uma lâmpada, que é uma tarefa corriqueira e sem muita dificuldade, é necessário que sigamos uma série de passos para atingir o objetivo, que poderia ser:

Passo 1: pegar uma escada.
Passo 2: colocar a escada perto da lâmpada.
Passo 3: buscar uma lâmpada nova.
Passo 4: subir a escada.
Passo 5: retirar a lâmpada velha.
Passo 6: colocar a lâmpada nova.
Passo 7: descer a escada.
Passo 8: jogar a lâmpada velha no lixo.
Passo 9: guardar a escada.

Você pode estar pensando agora: eu troco a lâmpada de maneira diferente, primeiro pego a lâmpada nova para depois buscar a escada. Na maioria das vezes, existem várias maneiras de se chegar a um mesmo objetivo, isto é, podem existir vários algoritmos para solucionar o mesmo problema. O importante, nesse caso, é que o algoritmo descreva os passos a serem seguidos em cada uma das maneiras possíveis.



Peça para um(a) amigo(a) descrever os passos que faz para trocar o pneu do carro. Compare com os passos que você faria e verifique se são iguais. É bem possível que os algoritmos sejam diferentes.

2.2 Tipos de algoritmos

Um algoritmo nada mais é do que uma linha de raciocínio lógico que pode ser descrita de várias maneiras, seja de forma textual, gráfica e até mesmo oralmente.

Apesar das diversas maneiras de se representar um algoritmo, segundo Ascencio e Campos (2005), os três tipos mais utilizados são:

- descrição narrativa;
- fluxograma;
- pseudocódigo.

2.2.1 Descrição narrativa

A descrição narrativa consiste em descrever a solução do problema utilizando uma linguagem natural, como, por exemplo, a língua portuguesa. Esse foi o meio que usamos até agora para descrever nossos algoritmos.

Vantagens:

Muito simples de usar já que é bem conhecida pela pessoa que analisa o problema.

Desvantagens:

A linguagem natural abre espaço para várias interpretações e tende a se tornar prolixa.

Exemplo:

Vamos descrever o algoritmo que some dois números.

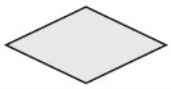
> Passo 1: receber o primeiro e segundo número.
> Passo 2: somar os dois números.
> Passo 3: mostrar o resultado obtido na soma.

2.2.2 Fluxograma

O fluxograma é uma representação gráfica de um algoritmo. Utiliza alguns símbolos pré-definidos, mostrados na Tabela 2.1, para identificar os passos a serem seguidos para chegar ao resultado.

TABELA 2.1 - Conjunto de símbolos utilizados no fluxograma

	Indica o início e o fim do algoritmo.
	Conecta os símbolos e indica o sentido do fluxo de dados.
	Indica cálculos e atribuições de valores.

	Representa entrada de dados.
	Representa saída de dados.
	Representa uma tomada de uma decisão.

Fonte: (ASCENCIO; CAMPOS, 2005)

Vantagens:

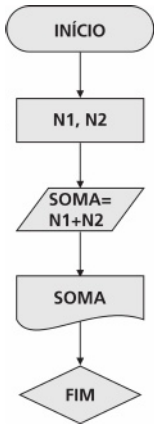
A lógica usada é mais precisa não dando margem a múltiplas interpretações, e consequentemente, fica mais fácil a migração do algoritmo para uma linguagem de programação real.

Desvantagens:

É muito mais trabalhoso fazer um desenho do que simplesmente escrever, mesmo considerando o auxílio de programas especializados. Além disso, a representação gráfica dos algoritmos tende a ficar muito extensa e, consequentemente, mais difícil de ser analisada e modificada (imagine refazer todo o desenho por causa de uma mudança na lógica).

Exemplo:

Vamos descrever o algoritmo que some dois números na forma de fluxograma:



Fonte: Autores.

2.2.3 Pseudocódigo

O pseudocódigo descreve a solução de um problema algorítmico de maneira textual e por meio de regras predefinidas. Ele utiliza uma linguagem estruturada que lembra o Pascal, uma linguagem de programação que já foi muito usada.

A palavra pseudocódigo significa "falso código". Alguns autores chamam o pseudocódigo de **português estruturado** ou **portugol**, bem sugestivo, já que podemos considerá-lo como uma linguagem intermediária entre a linguagem natural e a de programação.

Vantagens:

Além de a sintaxe ser bem próxima à da linguagem natural, a migração de um pseudocódigo para uma linguagem de programação é quase que imediata.

Desvantagens:

Pseudocódigos, assim como linguagens de programação, possuem algumas regras (mesmo que sejam bem mais simples do que a de uma linguagem de programação completa). É necessário aprendê-las e aplicá-las corretamente.

Exemplo:

Vamos descrever um pseudocódigo que some dois números:

```
ALGORITMO
DECLARE N1, N2, SOMA: NUMÉRICO;
ESCREVA "Digite dois números";
LEIA N1, N2;
SOMA ← N1 + N2;
ESCREVA "O resultado da soma é igual a: " , SOMA;
FIM_ALGORITMO.
```

Resumo

No desenvolvimento desta aula, você teve a oportunidade de aprender o que é um algoritmo. Discorremos sobre os tipos de algoritmo. Os três mais utilizados são a descrição narrativa, o fluxograma e o pseudocódigo. Mostramos também as vantagens e as desvantagens de cada um desses algoritmos. Esperamos que você esteja apto a escrevê-los ou desenhá-los, mesmo sem

ter conhecimento sobre todos os recursos e regras existentes.

Agora é hora de praticar. Vamos lá!

Atividades de aprendizagem

1. Descreva qual é a principal função dos algoritmos.

2. Faça um algoritmo na forma de "descrição narrativa" que descreva os passos de uma pessoa para abrir um documento do Word armazenado no seu e-mail.

3. Faça um algoritmo na forma de "fluxograma" que mostre o salário total de um funcionário, dado o salário que ele recebe, o número de filhos que possui e sabendo que, para cada filho, ele recebe uma ajuda de custo de R$ 32.

4. Faça um algoritmo na forma de "pseudocódigo" que calcule o cubo de um determinado número informado.

Caro(a) estudante,

Esperamos que tenha achado interessante o estudo dos algoritmos, pois eles facilitam muito a solução de alguns problemas. Vamos continuar? Ainda temos muito assunto legal para estudar. Na próxima aula, abordaremos as ferramentas de criação de algoritmos.

Aula 3. Ferramentas de criação de algoritmos

Objetivos:

- identificar os comandos de entrada, processamento e saída de dados; e
- desenvolver algoritmos com pseudocódigos/VisualG e Java/Eclipse.

Caro(a) estudante,

Atualmente existem centenas, senão milhares de programas especializados em criar outros programas. Eles são chamados de ambiente integrados de desenvolvimento (do inglês: *Integrated Development Environment* – IDE). Iremos utilizar em nossas aulas uma ferramenta bem simples, o VisualG, que não chega a ser uma IDE completa, mas é o suficiente para aprender e exercitar a criação de algoritmos.

Veremos que, para cada exemplo dado em VisualG, teremos o equivalente em Java, que é uma linguagem de programação completa e tem várias IDEs profissionais que dão suporte a essa linguagem.

3.1 Introdução

Qualquer dispositivo eletrônico que manipula informações, seja de pequeno porte, como um celular; de médio porte, como um computador *desktop*; ou de grande porte, como os servidores de datacenters, todos eles executam basicamente três ações:

- entrada de dados;
- processamento de dados;
- saída de dados.

Como esses dispositivos são controlados essencialmente por algoritmos específicos para cada cenário ou função, podemos dizer que um algoritmo é composto por comandos de entrada de dados, comandos de processamento de dados e comandos de saída de dados.

Mas o que seria, na prática, entrada, processamento e saída de dados?

- **Entrada de dados** é quando o computador recebe dados do mundo externo, como, por exemplo, do teclado, do microfone, da *webcam* ou de muitos outros. O algoritmo armazena essas informações na memória para posterior processamento.

- **Processamento de dados**é o momento em que o algoritmo que está rodando em um computador recebe os dados e os transforma de acordo com uma lógica predefinida, gerando, assim, algum tipo de informação que pode ser utilizado posteriormente, tanto pelo próprio algoritmo quanto pelo mundo externo.

- **Saída de dados** é quando o computador envia os dados resultantes do processamento de dados para o mundo externo. O computador pode enviar essa informação de diversas maneiras, tais como mostrar no monitor, imprimir no papel, enviar o áudio pela caixa de som, gravar no disco rígido, entre muitos outros.

Em resumo, o algoritmo é uma sequência de passos para se chegar a um objetivo, sendo que esses passos, quando executados em um computador, podem ser chamados de **comandos** ou **instruções**, os quais podem ser de entrada, processamento ou saída de dados.

Agora que sabemos o suficiente acerca de algoritmos, vem a pergunta: como criá-los de maneira eficiente?

3.2 Plataforma de estudo

Foi visto na aula anterior que podemos representar um algoritmo de várias maneiras diferentes, mas, no decorrer de nossas aulas, vamos adotar duas:

- pseudocódigo; e

- linguagem de programação Java.

A justificativa para essas escolhas é que o pseudocódigo possui a sintaxe mais amigável para principiantes, e o Java é, hoje, um padrão de programação no mercado.

3.2.1 Desenvolvendo algoritmos com pseudocódigos e VisualG

Para desenvolver pseudocódigos, basta um papel, um lápis e muita lógica de programação. Entretanto, não é possível "executar" esse pseudocódigo, dificultando, assim, testar a lógica usada.

Uma alternativa é usar um programa desenvolvido no Brasil que usa a língua portuguesa para descrever o pseudocódigo, que é o VisualG.

Esta é a janela do VisualG

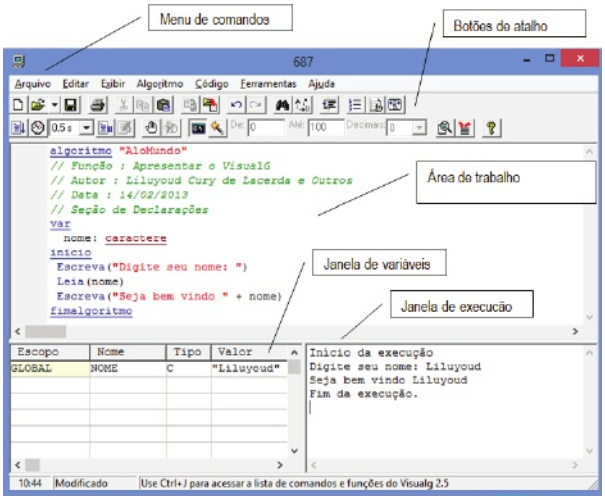


Figura 3.1 - Janela do VisualG
Fonte: Autores.

A interface do VisualG é bem intuitiva e permite a você criar algoritmos de maneira bem simples e direta. Vamos descrever alguns de seus itens.

- **Menu de comandos** – tem todas as opções disponíveis no programa, desde salvar e recuperar algoritmos até executá-los e testá-los.



Além do Java, outras linguagens de programação são bastante usadas. Podemos citar o C# e Visual Basic para desenvolvimento com produtos Microsoft e Web, e o PHP, específico para desenvolvimento para Web. Outro padrão que está em constante transformação e hoje já é capaz de desenvolver aplicações completas na Web é o HTML 5 + CSS 3 e Javascript.



O VisualG roda no sistema operacional Windows XP ou posterior (inclusive no Windows 8). Para baixá-lo, basta digitar VisualG 2.5 no Google e baixar as versões disponíveis. Entre elas temos:
http://www.guanabara.info/2007/09/visualg-v25/
http://www.ifro.br/downloads/visualgv25.exe

Para conhecer mais sobre o VisualG, assista ao vídeo: http://www.youtube.com/watch?v=sU17rbKEPUA

Ou leia o manual do VisualG do Professor Arley Rodrigues: http://www.facom.ufu.br/~claudio/Cursos/PP/Docs/Visualg2.pdf

- **Botões de atalho** – como o próprio nome já diz, são atalhos para os principais comandos do menu.
- **Área de trabalho** – é o local onde escrevemos nossos algoritmos em pseudocódigo/portugol.
- **Janela de variáveis** – mostra todas as variáveis que estão sendo usadas no algoritmo.
- **Janela de execução** – serve para mostrar a saída do sistema.

3.2.2 Desenvolvendo algoritmos com Java e Eclipse

Recomendamos que use o Eclipse, uma IDE que permite o desenvolvimento rápido de aplicações (RAD - Rapid Application Development) em Java.

O Eclipse é a IDE Java mais usada e também é gratuita. Você pode baixá-la em: http://www.eclipse.org/downloads/

Ao se criar um algoritmo para solucionar determinado problema, o objetivo é poder aplicá-lo na vida real, em situações reais. Para tanto, faz-se necessário transcrever o algoritmo para uma linguagem de programação completa.

A linguagem escolhida para este caderno é o pseudocódigo com VisualG, mas o Java, embora mais complexo, é o mais usado e apresenta muitos detalhes que, por si sós, já dariam um livro (na verdade existem inúmeros livros dedicados à linguagem Java). Por isso, iremos mostrar apenas o necessário para que consigamos transcrever nossos algoritmos de VisualG para o Java.

Esta é a janela do Eclipse

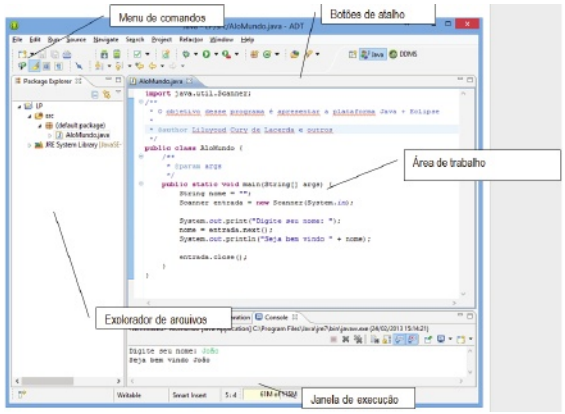


Figura 3.2 - Janela do Eclipse
Fonte: Autores.

Perceba que a interface do Eclipse é bem mais complexa, com muitas opções e janelas, isso porque o Eclipse é uma ferramenta de uso profissional que permite desde a criação de programas bem simples até programas para Web, celulares, *tablets* e muitos outros.

A priori, para você conseguir trabalhar com o Eclipse, vamos descrever apenas algumas áreas do mesmo:



Para conhecer mais sobre o Eclipse, assista ao vídeo: http://www.youtube.com/watch?v=Vf1AiGlaRQQ

- **Menu de comandos** – tem todas as opções disponíveis no programa, desde salvar e recuperar projetos em Java até executa-los e testa-los.
- **Botões de atalho** – como o próprio nome já diz, são atalhos para os principais comandos do menu.
- **Área de trabalho** – é o local onde escrevemos os programas em Java.
- **Explorador de arquivos** – permite gerenciar todos os arquivos que fazem parte do meu projeto de *software* (isso mesmo, um programa em Java pode ser composto por vários arquivos).
- **Janela de execução** – serve para mostrar a saída do sistema. É usualmente chamado de console ou *output*.

Para cada problema dado, a partir de agora vamos resolvê-los de duas formas, com as dobradinhas VisualG/pseudocódigo e Eclipse/Java.

Resumo

No decorrer da aula, tratamos das ferramentas que iremos usar para criar nossos algoritmos ao longo da disciplina. Desenvolvemos algoritmos com pseudocódigos usando o visual G e em Java com o Eclipse, uma ferramenta de uso profissional que permite desde a criação de programas bem simples até os mais complexos.

Agora, vamos às atividades.

Atividades de aprendizagem

1. Além das IDEs citadas na aula, pesquise mais quatro usadas para programação e identifique as linguagens que elas utilizam.

2. Instale as IDEs citadas nesta aula e escreva um algoritmo em cada uma

delas (pode ser uma variação dos que foram utilizados como exemplo).

3. Salve e recupere os algoritmos criados no exercício anterior. Analise as estruturas das pastas e dos arquivos criados por cada uma das plataformas (pseudocódigo com VisualG e Java com Eclipse).

4. Dê sua opinião sobre escrever algoritmos em pseudocódigos e em uma linguagem de programação verdadeira. Discorra sobre sua experiência nas atividades anteriores.

Caro(a) estudante,

Então, gostou do VisualG e do Eclipse? Com esses dois programas, você irá fazer coisas incríveis. Na Aula 4, mostraremos as estruturas básicas de um algoritmo. Continue estudando com afinco nossas aulas. Até a próxima!

Aula 4. Estrutura básica de um algoritmo

Objetivo:

- criar uma estrutura básica para desenvolver um algoritmo em pseudocódigo e um algoritmo em Java.

Caro(a) estudante,

A partir de agora, vamos começar a criar nossos próprios algoritmos, espaço onde acontece toda a mágica do mundo da programação. Mas precisamos primeiramente entender como funciona a estrutura básica do mesmo, que é muito simples e fácil, para depois testá-los nos ambientes integrados de desenvolvimentos vistos na aula passada.

4.1 Introdução

Os algoritmos são feitos para serem executados por máquinas, e sabemos que as máquinas ainda estão muito longe de ter uma "inteligência" comparada à do ser humano.

Você, ao estabelecer uma conversa com outra pessoa, usando a língua portuguesa, pode não seguir todas as regras gramaticais que ela exige, mas, mesmo assim, o diálogo acontece.

Veja o exemplo:

– E aí, João, tudo em riba?
– Cara, tudo na boa.
– Quando nós vai jogar aquela pelada?
– Mano, vamu marcar nesse fim de semana agora com a turma.

Nesse exemplo de comunicação entre duas pessoas, mesmo com os inúmeros erros (gírias, concordância, abreviações etc.), o diálogo aconteceu, isto é, os dois se entenderam.

O algoritmo é uma maneira de se "falar" com o computador, mas, nessa comunicação, um é provido de inteligência (o ser humano, no caso, o programador) e o outro... bem, vamos dizer que ainda não temos um HAL 9000 à nossa disposição.

É por isso que essa comunicação deve ser bem definida, sem erros, em uma linguagem bem limitada e simples (poucas regras, pequeno vocabulário). Cada linguagem de programação tem suas próprias regras e seu próprio vocabulário, mas elas, em sua maioria, são muito semelhantes.

Na comunicação com o computador, isto é, na criação do algoritmo, se você errar uma única palavra, usar uma estrutura errada ou até mesmo esquecer uma vírgula, o computador não vai entender, gerando erros na hora de compilar (traduzir do código-fonte para o código que a máquina entende) ou na hora de executar.

Por essa razão, nesta aula, vamos explicar detalhadamente essa estrutura de comunicação com o computador, que, como já dissemos antes, por ser limitada e simples, fica fácil de aprender – é só não se esquecer dos detalhes.

4.2 Como definir um algoritmo

Um algoritmo mínimo tem pelo menos um nome, uma descrição, a identificação do início e do fim do mesmo e, lógico, um comando (um passo a ser executado), pelo menos.

4.2.1 Nomenclatura de algoritmos

Para criar um algoritmo precisamos saber o seu objetivo e, a partir deste, damos-lhe um nome. Se vamos criar um algoritmo para calcular a média, poderíamos chamar o algoritmo de "Calcular Média".

Mas existe um problema com o nome "Calcular Média": ele não segue os padrões internacionais de nomenclatura de símbolos para linguagens de programação. Nesse padrão, recomenda-se:

- Não usar espaço na definição dos nomes;
- Não usar caracteres especiais, tais como: **! ? $ # * & - /** e outros;
- Usar apenas letras e números;

- Não iniciar o nome com um número;
- Evitar uso de acentos e cedilhas.

A seguir temos alguns exemplos de nomes de algoritmos válidos e não válidos:

TABELA 4.1 - Exemplos de nomes de algoritmos

Objetivo do algoritmo	Nome não válido	Nome válido
Calcular média das notas	Calcular média	CalcularMedia
Verificar a maior nota	Verificar maior nota	VerificarMaiorNota
Verificar endereço	Verificar-Endereço	VerificarEndereco
Somar dois números	2NúmerosSoma	Soma2Numeros
Calcular $ da venda	Calcular $Venda	CalcularValorVenda
Mostra Alô Mundo na tela	AlôMundo	AloMundo

Fonte: Autores.

A sintaxe usada pelo pseudocódigo e pelo Java para definir o nome de um algoritmo foi visto nas figuras 3.1 e 3.2, respectivamente, mas vamos revê-los lado a lado agora:

TABELA 4.2 - Definindo o nome do algoritmo

Pseudocódigo	Java
`algoritmo "AloMundo"`	`public class AloMundo`

Fonte: Autores.

4.2.2 Escopo de um algoritmo

Todo algoritmo tem início e fim. Para identificar essas partes, usamos palavras ou símbolos. O conceito de escopo vem justamente da necessidade de delimitar o intervalo em que determinadas informações processadas pelo algoritmo são válidas.

A seguir, temos um exemplo de definição do escopo de um algoritmo (o início e o fim dele) tanto em pseudocódigo quanto em Java.

TABELA 4.3 - Definindo o início e o fim do algoritmo

Pseudocódigo		Java	

`algoritmo "AloMundo"`
`inicio`
`fimalgoritmo`
	} Escopo do algoritmo	`public class AloMundo`
		`{`
		`}`
			} Escopo do algoritmo

Fonte: Autores.

Em algoritmos, é muito comum haver estruturas de programação dentro de outras estruturas. Assim como o algoritmo em si tem um escopo (um espaço válido), cada estrutura existente no algoritmo também pode ter seu próprio escopo. Nesse caso, informações tratadas em um escopo mais interno, a *priori*, não serão acessíveis pelo externo. Esse conceito mais abrangente de escopo será tratado nas próximas aulas.

Perceba que no pseudocódigo está explícita a delimitação do início e do fim do algoritmo. Já em Java foram usadas as chaves para definir o escopo.

4.2.3 Comandos ou instruções

Comandos ou instruções são mecanismos usados para dizer ao computador o que deve ser feito. Cada instrução dada é um passo sendo executado dentro do algoritmo, e os mesmos devem estar em ordem, da primeira instrução até a última, quando o objetivo do algoritmo deve ser alcançado.

Os comandos devem estar dentro do escopo do algoritmo – no caso do pseudocódigo, entre as palavras `inicio` e `fimalgoritmo`, e, no caso do Java, entre as chaves { }. A seguir, temos um exemplo de um comando de saída:

TABELA 4.4 - Inserindo comandos no algoritmo

Pseudocódigo	Java

`Algoritmo "AloMundo"`
`inicio`
`  Escreva("Alô mundo!")`
`Fimalgoritmo`
	`public class AloMundo`
	`{`
	`  public static void main(String[] args)`
	`  {`
	`  System.out.print("Alô mundo!");`
	`  }`
	`}`

Fonte: Autores.

Perceba que o equivalente à instrução `Escreva()` do pseudocódigo é o `System.out.print()` no Java. Além disso, verifica-se que ambos usam os parênteses ( ) ao final do comando: isso denota a execução de uma função dentro da linguagem de programação.

No pseudocódigo, cada instrução fica em uma linha. Já no Java, o final da instrução é delimitado por ponto e vírgula, sendo assim, podemos ter mais de uma instrução em uma mesma linha, desde que separadas por esse sinal.

É importantíssimo destacar um detalhe na linguagem Java: observe que temos duas estruturas no código apresentado:

- `public class AloMundo`
- `public static void main(String[] args)`

A primeira define o nome **classe**, mas esse conceito de classe só será visto na disciplina de Programação Orientada a Objetos. Neste momento, vamos entender que seja apenas a identificação do programa, no caso `AloMundo`.

A segunda estrutura é um método cujo conceito também será abordado a fundo somente em Orientação a Objetos. Neste momento, vamos considerar como sendo um subprograma, isto é, um programa dentro de outro programa.

Então, no programa `AloMundo` em Java, mostrado na Tabela 4.4, temos um subprograma chamado `main` (principal, em inglês) interno a ele e dois escopos distintos, já que ambos têm um início e fim (delimitados por { }).

É considerada uma boa prática de programação que, ao se definir um escopo interno a outro, como aconteceu no exemplo, dar um espaçamento à direita (3 espaços ou mais) para deixar explícito que a estrutura `main` pertence ao escopo de `class`. Isso é o que chamamos de indentação:

TABELA 4.5 - Escopos e indentação em um código Java

```
public class AloMundo
{
   public static void main(String[] args)
   {
      System.out.print("Alô mundo!");
   }
}
```

Indentação

Escopo do subprograma main

Escopo do programa AloMundo

Fonte: Autores.

4.2.4 Comentários

Imagine que você tenha desenvolvido um algoritmo bem complexo há um ano, e, por acaso, você precisa usá-lo para resolver outro problema. Se o algoritmo não estiver documentado, você certamente terá um trabalho enorme em compreendê-lo novamente, perdendo assim um tempo precioso. Mas, caso você o tenha documentado, será possível identificar o que ele faz, as estruturas de programação que foram usadas e o porquê, entre muitas outras informações úteis acerca dele, de suas alterações (histórico) e dos autores do mesmo.

Uma das maneiras mais simples (mas não é a única) de documentar seu có-

digo é usando comentários, possibilitando assim um melhor entendimento do algoritmo implementado.

Podemos usar comentários para:

- Identificar o responsável e data de criação daquele código.
- Identificar todas as alterações feitas no código, seus responsáveis e datas de alteração.
- Identificar o objetivo daquele código.
- Explicar de maneira mais fácil a lógica de seu algoritmo.
- Entre muitos outros, inclusive para a documentação profissional de seu código-fonte através de marcadores específicos.

Existem três tipos de comentários:

- **Comentários de uma linha:** texto antecedido por // vira comentário de uma única linha.
- **Comentários de muitas linhas:** este tipo de comentário não existe em pseudocódigos, só em linguagens de programação. Em pseudocódigo você pode criar comentários com muitas linhas de código desde que cada linha tenha o marcador // na frente. No Java, todo o texto delimitado por /* */ se transforma em um comentário de muitas linhas.
- **Comentários de documentação de código:** este tipo de comentário não existe em pseudocódigos, só em linguagens de programação. No Java, todo o texto delimitado por /** */, se transforma em uma documentação de código.

Os comentários são ignorados na hora de executar o algoritmo, por isso você pode usar à vontade. Apenas tenha cuidado em não abusar deles, o que, em vez de ajudar, pode "contaminar" o seu código com muitas informações inúteis.

Para mostrar a funcionalidade da documentação de seu código, vamos inserir esses três tipos de comentário no algoritmo AloMundo:

TABELA 4.6 - Comentários em pseudocódigo

Pseudocódigo

```
Algoritmo "AloMundo"
// Função : Apresentar o VisualG
// Autor :Liluyoud Cury de Lacerda e Outros
// Data : 14/02/2013
// Seção de Declarações
Inicio
   // Essa função escreve Alô mundo na tela do computador
   Escreva("Alô mundo!")
Fimalgoritmo
```

Fonte: Autores.

TABELA 4.7 - Comentários em Java

Pseudocódigo

```
/**
 * Esse programa tem como função apresentar os comentários em Java
 * @Author: Liluyoud Cury de Lacerda
 * @Version: 1.0
 */
public classAloMundo
{
   /* O método main cria um ponto inicial de execução
      para o código em java. */
   public static voidmain(String[] args)
   {
      // O método main cria um ponto inicial de execução
      System.out.print("Alô mundo!");
   }
}
```

Fonte: Autores.

Como foi dito antes, o pseudocódigo só usa um tipo de estrutura de comentários, o //, mas o Java apresenta três tipos, com finalidades distintas, o //, /* */e o /** */.

Em relação às cores usadas pelos comentários, que são diferentes no VisualG (verde) e no Eclipse (cinza), é apenas a maneira como o ambiente integrado de desenvolvimento trata determinados tipos de estruturas dentro do código. Esse tipo de configuração pode ser mudado pelo programador.

Resumo

Você acaba de adquirir novos conhecimentos, pois, nesta aula, apontamos como definir um algoritmo. Vimos que, para criar um algoritmo, precisamos dar um nome a ele dependendo do seu objetivo, e que esse nome deve estar de acordo com os padrões internacionais de nomenclatura de símbolos para

linguagens de programação. Discorremos sobre esses padrões. Abordamos o escopo, isto é, o espaço entre o início e fim de um algoritmo. Mostramos, na tabela 4.3, o escopo de um algoritmo em pseudocódigo e em Java. Na sequência, tratamos dos comandos, instruções dadas ao computador para serem executadas, as quais devem estar inseridas no escopo. Apontamos também como documentar o algoritmo, ou seja, inserir informações úteis, tais como o que ele faz, as estruturas de programação que foram usadas, suas alterações (histórico) e o nome de seus(suas) autores(as), o que facilitará caso venhamos a utilizá-lo para resolver outro problema. Enfim, mostramos como criar uma estrutura para desenvolver um algoritmo básico tanto em pseudocódigo com VisualG, quanto em Java com Eclipse.

Atividades de aprendizagem

1. Crie um algoritmo que forneça a seguinte saída na tela do computador (seria interessante para seu aprendizado, criar o algoritmo tanto em pseudo-código quanto em Java):

```
+----------------------------------+
| Programa Alô Mundo               |
+----------------------------------+
| Seja bem-vindo ao maravilhoso    |
| mundo da programação.            |
|                                  |
| Aproveite!!!                     |
+----------------------------------+
```

2. Documente o código criado acima usando comentários.

3. Baseado(a) nas funções de algoritmos descritas abaixo, determine o pos-sível nome para cada um deles:

Funçãodalgoritmo	Nome
Transformar dólar em real	
Calcular o índice de massa corporal de uma pessoa	
Calcular a área de um triângulo	
Efetuar um saque na conta-corrente	

4. Crie um algoritmo mínimo para cada dos apresentados acima, usando os nomes que você sugeriu, (seria interessante, para seu aprendizado, criar o algoritmo tanto em pseudocódigo quanto em Java).

Caro(a) estudante,

Como se sentiu criando e executando seu primeiro programa? Na próxima aula, trataremos das variáveis e dos tipos de dado. Esperamos você lá!



Aula 5. Variáveis e tipos de dado

Objetivos:

- conceituar variável e tipos de dados;
- diferenciar dado de informação;
- aplicar as regras para a nomenclatura de variáveis; e
- identificar os tipos de dados primitivos existentes e utilizá-los em algoritmos, seja em pseudocódigo, seja em Java.

Caro(a) estudante,

Nesta aula, veremos um conceito primordial para o processamento de dados, que é o de variáveis. Vamos entender os mecanismos de funcionamento das variáveis nas linguagens de programação, mostrar como dar nome a elas e definir o tipo de dado que elas armazenam. Também vamos demonstrar o uso de variáveis tanto em pseudocódigo quanto em Java.

5.1 Introdução

Antes de iniciar os estudos sobre variáveis, precisamos entender o conceito de dado na computação. Segundo Puga e Rissetti (2009), dados são valores que serão utilizados pelo algoritmo para a resolução de um problema. Esses valores podem ser fornecidos pelo usuário através de dispositivos de entrada de dados, ou srcinados a partir de outros programas (inclusive ele próprio).

5.2 Variáveis

Os dados de um algoritmo, para serem utilizados e processados, precisam estar previamente armazenados na memória do computador. A variável então seria o nome dado a um espaço da memória que contém o dado a ser utilizado pelo algoritmo.

Qual a diferença entre um dado e uma informação? Em um contexto de programas de computador, podemos dizer que os dados correspondem aos valores fornecidos ao programa, que depois de processados, geram informação.

Quando usamos uma variável, na verdade, estamos buscando aquela informação da memória e passando-a para o processador do computador, para que este realize as operações necessárias com aquele dado.

Podemos afirmar, então, que as variáveis são utilizadas para representar valores genéricos, que poderão ser acessados e modificados de acordo com a lógica do algoritmo.

5.2.1 Nomenclatura de variáveis

Assim como é importante saber dar nome aos nossos algoritmos, é primordial saber nomear uma variável também. As regras para a nomenclatura de variáveis seguem os mesmos princípios dos algoritmos. Vejamos:

- Não usar espaço;
- Não usar caracteres especiais tais como: **! ? $ # * & - /** e outros;
- Usar apenas letras e números;
- Não iniciar o nome com um número;
- Evitar uso de acentos e cedilhas.

A seguir, temos alguns exemplos de nomes de variáveis:

TABELA 5.1 - Exemplos de nomes de variáveis

O que vai ser armazenado na variável?	Proposta de nome	Situação
O nome de uma pessoa	`Nome`	Nomenclatura válida
A nota de um aluno	`n`	Nome válido, mas inapropriado, já que não identifica o contexto da variável. O certo seria: `nota` ou `notaAluno`.
Endereço de um cliente	`endereço`	Nome válido, mas inapropriado, já que o nome contém cedilha. O certo poderia ser: `endereco`, `enderecoCliente` ou `clienteEndereco`.
Saldo da conta	`contaSaldo`	Correto. Poderia ser também `saldoConta`.
Nome do 2º dependente	`2dependente`	Nome inválido, já que não pode ser iniciado com um número. O certo poderia ser: `segundoDependente` ou `dependente2`.
Email do funcionário	`e-mail`	Nome inválido, já que não pode ter caracteres especiais, no caso o "-". O certo poderia ser: `email` ou apenas `mail`.

Fonte: Autores.

5.3 Tipos de dado

A linguagem nativa do computador é a linguagem binária, composta apenas por dois dígitos, ou seja, por 0 e 1. Um valor/dado é armazenado na memória do computador em forma binária. Se o computador for de 8 bits, cada fração da memória é composto de oito zeros ou uns, como os exemplos:

TABELA 5.2 – Raio X da memória de um computador com 1 GByte de RAM

Endereço de memória	Exemplo de dado	Exemplo de nomenclatura
1	00000000	
2	10010101	Variável X
3	11000010	Variável Z
4	00000000	
5	11111111	Variável
...	...	
1.073.741.824 bytes ou 1 Gbyte	10101010	

Fonte: Autores.

Segundo a Tabela 5.2, o endereço 2 da memória armazena a variável cujo nome é X. Esse endereço contém apenas *bits* (0 e 1), mas, para nós, humanos, o que eles representam? Um conjunto de *bits* pode representar um número, um texto, um som, uma imagem, entre muitos outros tipos de dado.

É por esse motivo que, quando definimos uma variável, é necessário dizer de que tipo ela é para que o programa consiga converter esses *bits* no dado que desejamos. O *byte* 10010101 pode tanto representar a letra A quanto o número 149 ou o número -21 (o primeiro *bit* pode representar o sinal positivo ou negativo).

Isto é, no exemplo acima, um único *byte* pode representar três valores diferentes, dependendo do tipo de dado.

A menor unidade de informação reconhecida pelo computador é o bit, que pode ser 0 (representa o estado desligado) ou 1 (representa o estado ligado). Um conjunto de oito bits (oito zeros ou uns) é chamado de byte. Em um computador padrão de 64 bits, significa que cada endereço de memória contem oito bytes (8 x 8 bits).

5.3.1 Tipos de dados primitivos

A definição do tipo de dado de uma variável é primordial para garantir a resolução do algoritmo. As linguagens de programação têm alguns tipos já definidos em sua estrutura – eles são chamados de tipos de dados primitivos.

Os tipos de dados primitivos são:

- **Literal:** tipo de dado que pode receber letras, números e símbolos. Esses tipos de dados não podem ser usados para fazer cálculos, apenas para

armazenar informação. Ex.: "Programadores & Associados", "Professor Universitário", "2ª chamada", etc.

- **Inteiro:** tipo de dado que pode receber números inteiros positivos ou negativos. Ex.: 10, 1000, -23, 0.

- **Real:** tipo de dado que pode receber números reais, isto é, com casas decimais, positivos ou negativos. Ex.: 3,1415; 9,8; 123,45.

- **Lógico:** tipo de dado que pode receber apenas dois tipos de informação – verdadeiro (1) ou falso (0).

5.3.2 Tipos de dados primitivos em Java

Cada linguagem de programação pode definir seus próprios tipos de dados, desdobramentos dos tipos primitivos citados anteriormente. No Java, os tipos primitivos são:

TABELA 5.3 - Tipos de dados primitivos em Java

Tipo de dados	Em Java	Capacidade
Literal	char	Armazena um único caractere
	String	Armazena um conjunto de caracteres
Inteiro	byte	Armazena números de -2^7 a 2^7-1 (28 a 127)
	short	Armazena números de -2^{15} a $2^{15}-1$ (32.768 a 32.767)
	int	Armazena números de -2^{31} a $2^{31}-1$ (2.147.483.648 a 2.147.483.647)
	long	Armazena números de -2^{65} a $2^{65}-1$
Real	float	Armazena números de -3,4E-38 a +3,4E+38
	double	Armazena números de -1,7E-308 a +1,7E+308
Lógico	boolean	Armazena 0 (falso) ou diferente de 0 (verdadeiro)

Fonte: http://docs.oracle.com/javase/7/docs/.

5.4 Sintaxe e exemplos de declaração

A seguir veremos a sintaxe e exemplos de declaração em pseudocódigo e em Java.

5.4.1 Pseudocódigo

Em pseudocódigo:

- A declaração de variáveis deve ser feita antes de se iniciar o algoritmo;

- Há uma palavra reservada para identificar a declaração das variáveis. Em VisualG é **var**;
- O nome da variável que deve seguir as regras de nomenclatura explicadas anteriormente;
- O sinal de dois pontos (:) serve para separar o tipo do nome da variável;
- Temos o tipo da variável;
- Cada variável deve estar declarada em uma linha.

Seguindo essas premissas, temos a seguinte sintaxe e alguns exemplos para se declarar variável em pseudocódigo:

TABELA 5.3 - Sintaxe e exemplos de declaração de variáveis em pseudocódigo.

Sintaxe	Exemplos
var nomeDaVariável: **tipoDaVariavel**	**var** nome: **caractere** idade: **inteiro** salario: **real** casado: **logico**

Fonte: autores

5.4.2 Java

Em Java:

- Não existe uma palavra específica para definir a declaração de uma variável. Uma variável pode ser criada em qualquer parte do código, mas as boas práticas de programação recomendam que as variáveis principais sejam criadas anteriormente e documentadas;
- Ao se declarar uma variável, o tipo de dado vem antes do nome da variável;
- O nome da variável deve seguir as regras de nomenclatura explicadas anteriormente;
- É possível declarar diversas variáveis em uma linha, desde que sejam separadas por ponto e vírgula;

- Se diversas variáveis possuem o mesmo tipo, basta defini-lo e enumerar o nome de cada uma das variáveis separado por vírgula.

Seguindo essas premissas, temos a seguinte sintaxe e alguns exemplos para se declarar variável em Java:

TABELA 5.4 - Sintaxe e exemplos de declaração de variáveis em Java.

Sintaxe	Exemplos

`tipoDaVariavel n omeDaVariavel;`	`String nome;`
	`byte idade;`
	`float salario;`
	`boolean casado;`
	`int populacao, rebanho;`
	`double resultado; int dias;`

Fonte: autores

Resumo

Acabamos de estudar o conceito de variáveis e tipos de dado, primordiais para a elaboração de algoritmos. Juntamente com eles, mostramos como nomear uma variável, distinguir os tipos de dados primitivos existentes e, principalmente, como usá-los em algoritmos, seja em pseudocódigo, seja em Java.

Vamos praticar!

Atividades de aprendizagem

1. Assinale os nomes de variáveis válidas e inválidas. Quando inválida, informar o porquê:

a) XXX

b) a123

c) (A)

d) "NOME"

e) #66

f) dia/ano/mês

g) 1abc

h) Olá Pessoal

i) a.b

j) c*d

2. Baseado(a) na representação da informação do que será armazenado em uma variável, sugira um nome para a mesma e para o tipo de dado. Para cada um, use a sintaxe do pseudocódigo e do Java:

Oqueseráarmazenado?	Sintaxepseudocódigo Sintaxe Java
O salário de um funcionário	
A nota e a média de um aluno	
Os dias de atraso de uma prestação	
O CPF de uma pessoa	
Se a família tem casa própria	
Uma carta	
O endereço completo do cliente	
Se o produto é novo ou usado	

Caro(a) estudante,

Parabéns! Esta foi a quinta aula da disciplina. Esperamos ter contribuído para seu aprendizado. Na próxima aula, abordaremos os operadores e expressões que envolvem as variáveis. Não perca!



Aula 6. Operadores e expressões

Objetivos:

- reconhecer operadores aritméticos, relacionais e lógicos;
- empregar corretamente esses operadores;
- distinguir expressões aritméticas de expressões lógicas; e
- criar e resolver expressões aritméticas e lógicas.

Caro(a) estudante,

O uso de expressões matemáticas e lógicas é muito importante para se resolver determinados problemas algorítmicos. As expressões são formadas por valores, variáveis (operando) e operadores. Vamos explicar como criar essas expressões, definir os operandos e os operadores em um algoritmo e, principalmente, como analisar previamente tais expressões.

6.1 Introdução

Um processador basicamente é uma "máquina" que processa instruções de armazenamento e recuperação de valores, faz cálculos matemáticos e efetua comparação de valores.

Mas, para dizer ao computador como executar essas instruções é necessário utilizar operadores específicos para criar expressões que possam ser calculadas pelo processador.

A seguir, iremos apresentar todos os tipos de operadores e como usá-los para formar expressões.

6.2 Operadores e expressões

As operações e expressões podem ser:

- Operação de atribuição;
- Expressões aritméticas; e
- Expressões lógicas.

6.2.1 Operador e expressão de atribuição

A operação de atribuição é utilizada para armazenar um valor em uma variável. Esse valor pode ser predefinido (fixo), proveniente de outra variável ou resultado de outra operação.

A representação algorítmica de atribuição tem a seguinte notação:

```
variavel ← valor
```

O símbolo ← pode ser trocado por := no VisualG. No Java, atribuição é dada pelo caractere =. A tabela a seguir mostra ambos os casos.

TABELA 6.1 - Exemplos de operações de atribuição

AtribuiçãoemVisualG	AtribuiçãoemJava

var
  nome: caractere
  idade: inteiro
  salario: real
  casado: logico
inicio
  nome := "João & Silva"
  idade := 40
  salario := 620.37
  casado := VERDADEIRO
fimalgoritmo
	String nome;
	byte idade;
	float salario;
	boolean casado;
	int populacao, rebanho;
	double resultado; int dias;
	
	nome = "João & Silva";
	idade = 40;
	salario = 620.37;
	casado = true;
	populacao = 1200; rebanho = 80;
	resultado = 8.5; dias = 10;

Fonte: autores

Agora, vamos fazer algumas considerações sobre o código acima:

- **Por que o tipo literal (caractere e *String*) recebe valores entre aspas duplas?**

O uso de aspas duplas serve para delimitar um texto, que pode ter espaços ou caracteres especiais. Usar um delimitador de texto é uma prática comum nas linguagens de programação. No caso do Pascal são usadas as aspas simples.

- **Por que não uso aspas no número?**

Se você usar aspas no número, o computador vai entender que é um texto e não um número, dois tipos de dado completamente diferentes.

- **Por que o salário tem um ponto (.) e não uma vírgula (,) no número?**

Geralmente as linguagens de programação usam a língua inglesa como base, e, lá, o caractere que define o decimal é o ponto e não a vírgula.

- **O que aconteceria se tentasse atribuir um texto a um número ou vice-versa?**

Variáveis de um tipo de dado só devem receber valores daquele tipo. Na maioria das linguagens de programação existem comandos que permitem converter um tipo para outro, por exemplo, converter um texto em um número e vice-versa (nós iremos ver isso mais adiante).



Existem linguagens de programação que possuem o conceito de variáveis dinâmicas, isto é, podem receber valores de diversos tipos fazendo, inclusive, a conversão automática de um para outro. Entretanto, via de regra, é recomendado definir tipos nas variáveis. Linguagens que obrigam a definição de tipos em variáveis são chamadas de fortemente tipificadas.

6.2.2 Operadores e expressões aritméticas

Os operadores e as expressões aritméticas são responsáveis pelas operações matemáticas realizadas no computador.

O conjunto de símbolos que representa as operações básicas de matemática é chamado de operadores aritméticos. Veja a tabela 6.2.

TABELA 6.2 - Operadores aritméticos

Operador	Exemplo de expressão	Função
+	a + b	Soma o valor de a e b
-	a - b	Do valor de a é subtraído o valor de b
*	a * b	Multiplica a por b
/	a / b	Divide a por b
mod	a MOD b (pseudocódigo) a % b (Java)	Retorna o resto da divisão inteira de a por b. Ex: 5 % 2 é igual a 1

++	a++ ou ++a (Java)	Incrementa o valor de a em 1. a++ retorna o valor de a antes de incrementar. ++a primeiro incrementa para depois retornar o valor de a.
--	a-- ou --a (Java)	Subtrai 1 do valor de a. a-- retorna o valor de a antes de subtrair. --a primeiro subtrai para depois retornar o valor de a.

Fonte: autores

Nem todos os operadores aritméticos podem ser representados por um símbolo em linguagens de programação. Funções como exponenciação, raiz, seno, cosseno, tangente, entre muitos outros, são definidas por instruções específicas em cada linguagem.

6.2.3 Operadores e expressões lógicas

Os operadores relacionais são utilizados para comparar valores ou outras expressões. O resultado dessa comparação é do tipo lógico (booleano), isto é, só pode ser verdadeiro ou falso.

Na tabela 6.3, veja os símbolos que representam os operadores relacionais3.

TABELA 6.3 - Operadores relacionais

Operador	Exemplo de expressão	Função
>	b > a	Se o valor de a for maior que o de b, a expressão é verdadeira, senão é falsa.
>=	b >= a	Se o valor de a for maior ou igual ao de b, a expressão é verdadeira, senão é falsa.
<	b < a	Se o valor de a for menor que o de b, a expressão é verdadeira, senão é falsa.
<=	b <= a	Se o valor de a for menor ou igual ao de b, a expressão é verdadeira, senão é falsa.
=	a = b (pseudocódigo) a == b (Java)	Se o valor de a for igual ao de b, a expressão é verdadeira, senão é falsa.
<>	a <> b (pseudocódigo) a != b (Java)	Se o valor de a for diferente do valor de b, a expressão é verdadeira, senão é falsa.

Fonte: autores

É possível concatenar expressões lógicas criando outras expressões lógicas mais complexas. Para concatenar é necessário utilizar operadores lógicos, que podem ser:

TABELA 6.4 - Operadores lógicos

Operador	Exemplo de expressão	Função
E	a E b (pseudocódigo) a && b (Java)	A expressão a E b só será verdadeira se a e b forem verdadeiros.
OU	a OU b (pseudocódigo) a || b (Java)	A expressão a OU b só será falsa se a e b forem falsos.
NÃO	NAO a (pseudocódigo) ! a (Java)	Se a for verdadeiro, NAO a será falso. Se a for falso, NAO a será verdadeiro.

Fonte: autores



Para melhor conhecimento das funções dos operadores lógicos, acesse a página http://pt.wikipedia.org/wiki/Tabela_verdade.

6.3 Ordem de precedência

Em uma expressão com muitos operandos e operadores, existe uma ordem para estabelecer qual delas (a relação de dois operandos e um operador) será executada primeiro – é o que chamamos de ordem de precedência.

Para exemplificar melhor o conceito, vamos imaginar que queremos calcular a média de duas notas: n1 e n2.

```
media = n1 + n2 / 2
```

Se **n1** fosse igual a **6** e **n2** igual a **8**, a média deveria ser **7**, isso porque em nossa cabeça primeiro vamos calcular a expressão **n1 + n2** (operandos **n1** e **n2** e o operador +) e, depois de obter o resultado, iríamos dividi-lo por **2**.

Mas na programação não é bem assim que acontece: para o computador a média seria 10. Como assim? Como uma pessoa tira 6 e 8 e a média fica 10?

É que o computador analisa toda a expressão, subdivide em expressões mínimas (apenas 2 operandos e 1 operador) e verifica qual parte ele vai calcular primeiro, como **a divisão tem uma ordem de precedência maior que a soma**, o computador primeiro vai calcular **n2 / 2**, que pelo exemplo seria **4**, e depois somar com a **n1**, fazendo assim o resultado ser **10**.

Para garantir que uma determinada expressão seja executada antes da outra, basta colocá-la entre parênteses. No exemplo da média, para o computador calcular corretamente, a expressão deveria ser:

```
media = (n1 + n2) / 2
```

A ordem de precedência entre os operadores aritméticos e lógicos é:

TABELA 6.5 - Ordem de precedência

Prioridade	Operadoraritmético	Operadorlógico
1º	*	NÃO
2º	/	E
3º	MOD	OU
4º	+	
5º	-	

Fonte: autores

Como uma expressão pode conter vários operadores aritméticos, relacionais e lógicos, existe uma ordem geral entre eles, que é:

TABELA 6.6 - Ordem de precedência geral

Prioridade	Operadores
1º	Parênteses
2º	Operadoresaritméticos
3º	Operadoresrelacionais
4º	Operadoreslógicos

Fonte: autores

Sabendo como funciona a escolha do computador sobre a ordem em que ele calcula as partes de uma expressão, vamos tentar simular o comportamento do mesmo analisando as seguintes expressões lógicas (cujo resultado somente pode ser verdadeiro ou falso):

```
EXEMPLO 1
2 * 4 = 4 + 4
8 = 4 + 4
8 = 8
verdadeiro
```

```
EXEMPLO 2
3 + 3 * 2 <= 3 * 3 E 3 + 2 <= 15 MOD 3
3 + 6 <= 3 * 3 E 3 + 2 <= 15 MOD 3
9 <= 3 * 3 E 3 + 2 <= 15 MOD 3
9 <= 9 E 3 + 2 <= 15 MOD 3
9 <= 9 E 5 <= 15 MOD 3
9 <= 9 E 5 <= 0
verdadeiro E 5 <= 0
verdadeiro E falso
falso
```

EXEMPLO 3

```
(3 + 3) * 2 <= 9 E (3 + 2 <= 15) OU 15 MOD 3 = 0
   6 * 2 <= 9 E (3 + 2 <= 15) OU 15 MOD 3 = 0
     6 * 2 <= 9 E (5 <= 15) OU 15 MOD 3 = 0
       6 * 2 <= 9 E falso OU 15 MOD 3 = 0
         12 <= 9 E falso OU 15 MOD 3 = 0
             12 <= 9 E falso OU 0 = 0
               falso E falso OU 0 = 0
           falso E falso OU verdadeiro
               falso OU verdadeiro

                verdadeiro
```

EXEMPLO 4

```
NÃO (5 + 3 <> 8 OU (2 + 3 * 2 <= 3 * 3 / 2 OU (5 MOD 2 = 1 E 4 MOD 2 = 0)))
   NÃO (5 + 3 <> 8 OU (2 + 3 * 2 <= 3 * 3 / 2 OU (1 = 1 E 4 MOD 2 = 0)))
      NÃO (5 + 3 <> 8 OU (2 + 3 * 2 <= 3 * 3 / 2 OU (1 = 1 E 0 = 0)))
    NÃO (5 + 3 <> 8 OU (2 + 3 * 2 <= 3 * 3 / 2 OU (verdadeiro E 0 = 0)))
 NÃO (5 + 3 <> 8 OU (2 + 3 * 2 <= 3 * 3 / 2 OU (verdadeiro E verdadeiro)))
       NÃO (5 + 3 <> 8 OU (2 + 3 * 2 <= 3 * 3 / 2 OU (verdadeiro)))
        NÃO (5 + 3 <> 8 OU (2 + 3 * 2 <= 3 * 3 / 2 OU verdadeiro))
          NÃO (5 + 3 <> 8 OU (2 + 6 <= 3 * 3 / 2 OU verdadeiro))
            NÃO (5 + 3 <> 8 OU (8 <= 3 * 3 / 2 OU verdadeiro))
              NÃO (5 + 3 <> 8 OU (8 <= 9 / 2 OU verdadeiro))
               NÃO (5 + 3 <> 8 OU (8 <= 4,5 OU verdadeiro))
                 NÃO (5 + 3 <> 8 OU (falso OU verdadeiro))
                     NÃO (5 + 3 <> 8 OU (verdadeiro))
                   NÃO (5 + 3 <> 8 OU verdadeiro)
                     NÃO (8 <> 8 OU verdadeiro)
                     NÃO (falso OU verdadeiro)

                       NÃO (verdadeiro)
                      NÃO verdadeiro
                         falso
```

Os conceitos de operadores, operando e expressões serão de suma importância para a elaboração de algoritmos mais complexos, que preveem inúmeras possibilidades de execução dependendo de cada caso analisado. Vamos ao resumo.

Resumo

Nesta aula, estudamos os conceitos de operadores, operandos e expressões, que podem ser aritméticas, relacionais e lógicas, e o mais importante: mostramos como resolvê-las. Agora é sua vez praticar! Vamos lá!

Atividades de aprendizagem

1. O que é operando, operador e uma expressão? Exemplifique.

2. Quais os tipos de expressão existentes? Dê um exemplo de cada uma.

3. Sabendo que a = 2, b = 5 e c = 3, calcule o valor resultante das seguintes expressões:

a) `b * 2 – a * c`

b) `b + a * 2 – a + c * 3`

c) `(b + a) * (2 – a) + c * 3`

d) `(b + a) * ((2 – a) + c) * 3`

e) `(c + a * a – a + 1) / (c * b * a + c – a * c)`

4. Sabendo v = verdadeiro e f = falso, calcule o valor resultante das seguintes expressões:

a) `v E f OU f E v`

b) `(v E f) OU (f E v)`

c) `(NÃO (v E f)) E (f E v)`

d) `NÃO ((v E f) OU (f E v))`

e) `NÃO ((NÃO(v E f)) E (NÃO(f E v)))`

5. Sabendo que a = 1, b = 2 e c = 3, demonstre a execução passo a passo da seguinte expressão:

```
NÃO (b * c <> 8 OU ((a * c) - a <= c * c / a)) OU (b MOD a
= 1 E 4 MOD a = 0)
```

Caro(a) estudante,

Finalizamos mais uma aula da disciplina Lógica de Programação.

Na próxima, abordaremos as estruturas sequenciais.



Aula 7. Estruturas sequenciais

Objetivos:

- criar uma estrutura sequencial em um algoritmo; e
- reconhecer e usar corretamente os comandos de entrada e de saída.

Caro(a) estudante,

Finalmente lhe será oportunizada a criação de um programa completo, usando variáveis, atribuindo e lendo seus respectivos valores, definindo expressões aritméticas e lógicas e, por fim, você aprenderá a estruturar sequencialmente um algoritmo em VisualG e Java, a ler dados do usuário e a mostrar o resultado do processamento para o mesmo (a informação resultante). Aproveite bem a aula e se prepare, pois teremos muitas atividades.

7.1 Introdução

A arquitetura básica dos computadores, baseada na arquitetura de John von Neumann, até hoje tem influência direta nas linguagens de programação. Entretanto, o estilo (modelo ou paradigma) de programação sofreu algumas evoluções, de um formato simples (sem muitos recursos e de difícil percepção por parte dos leigos) da década de 50, para um formato mais completo (com muitos recursos e de fácil entendimento) nos dias atuais.

De acordo com Manzano e Oliveira (2009, p. 437), "os paradigmas de programação passaram por cinco fases evolucionárias", sendo elas:

- Programação tradicional;
- Programação estruturada;
- Programação modular;

- Programação com abstração de dados; e
- Programação orientada a objetos.

Nesta disciplina, vamos abordar exclusivamente aspectos da programação estruturada, que tem como base as estruturas de controle, que são:

- Estruturas sequenciais;
- Estruturas de decisão; e
- Estruturas de repetição.

Nesta aula, você verá como funciona uma estrutura sequencial.

A programação estruturada é uma das variantes da linguagem imperativa, que é baseada na arquitetura von Neumann, na qual tanto os dados quanto os programas são armazenados na mesma memória, que, por sua vez, é separada da CPU (Unidade de Processamento de Dados), necessitando, assim, que tanto os dados quanto as instruções sejam transmitidos da memória para a CPU e vice-versa.

Segundo Sebesta (2011, p. 38):

> por causa da arquitetura de von Neumann, os recursos centrais das linguagens imperativas são:
> • As variáveis, que modelam as células de memória;
> • As sentenças de atribuição, baseadas na operação de envio de dados e instruções; e
> • A forma iterativa de repetição nessa arquitetura.
>
> Os operandos em expressões são enviados da memória para a CPU e o resultado da avaliação da expressão é enviado de volta à célula de memória representada pelo lado esquerdo da atribuição.
>
> A iteração é rápida em computadores von Neumann porque as instruções são armazenadas em células adjacentes de memória, e repetir a execução de uma seção de código requer apenas uma simples instrução de desvio.

7.2 Estrutura sequencial

A estrutura sequencial em um algoritmo se dá quando um conjunto de ações é executado em uma sequência linear de cima para baixo, e da esquerda para a direita, na mesma ordem em que foram escritas.

A estrutura sequencial é a base do algoritmo, pois, como vimos anteriormente, são ações executadas passo a passo, em uma sequência definida para atender a um determinado objetivo.

Em uma estrutura sequencial, podemos definir instruções de declaração de variáveis, atribuição, operações aritméticas e lógicas, de entrada e de saída. Todas essas instruções já foram vistas, menos as duas últimas, que veremos a seguir.

7.2.1 Saída de dados

Para que serviria a capacidade de processar dados do computador se não fosse possível mostrar os resultados? É exatamente isso que os comandos de saída fazem, enviam informações processadas do algoritmo.

O dispositivo mais comum para se mostrar o resultado de um processamento é o monitor de vídeo, mas existem diversos outros dispositivos de saída, tais como o disco rígido, a impressora, a caixa de som etc. Um algoritmo também pode gerar uma saída que serviria de entrada para outro algoritmo, sem precisar passar por nenhum dispositivo tradicional de saída, mas isso não é

muito comum.

Nesse momento, subtende-se que o comando de saída é o monitor do computador.

O nome do comando de saída padrão em pseudocódigo é `Escreva()` e em Java é `System.out.print()`. Existem outras variantes desse comando. As principais são:

TABELA 7.1 - Variantes do comando Escreva

Função	Pseusocódigo	Java
Escreve o dado	`Escreva()`	`print().`
Escreve o dado e pula uma linha	`Escreval()`	`println().`
Escreve usando parâmetros		`printf().`

Fonte: autores

Com esses comandos é possível:

- Escrever um texto;
- Escrever o valor de uma variável;
- Escrever um conjunto de textos e valores de variáveis concatenados. Vamos a um exemplo do comando `Escreva` e `Escreval`

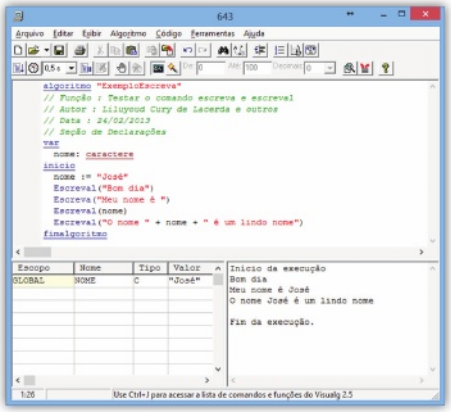


Figura 7.1: Exemplo do comando Escreva e Escreval.
Fonte: Autores.

Veja o código e analise as instruções passadas na área de trabalho e a saída do mesmo na janela de execução. Perceba que:

- Quando usei o comando `Escreval`, ele escreve o texto na saída e pula uma linha;
- Quando usei o comando `Escreva`, ele escreve o texto na saída, mas não pula uma linha;
- Para escrever o texto, delimitei o mesmo com aspas duplas (“ ”);
- Quando não uso as aspas duplas, significa que quero escrever o valor de uma variável; e
- Quando quero mesclar texto com variáveis, faço a concatenação deles com o sinal de adição (+).

Agora, vamos escrever esse mesmo código em Java, no Eclipse.

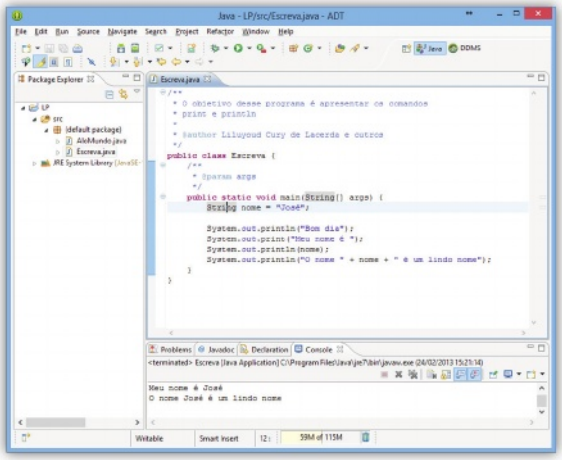


Figura 7.2: Exemplo dos comandos print e println.
Fonte: Autores.

O exemplo usando `printf` você verá na figura 7.4.

7.2.2 Entrada de dados

No exemplo anterior, perceba que não houve interação entre o(a) usuário(a) e o programa – o algoritmo simplesmente executou.

Mas, na maioria dos algoritmos, a entrada de dados é de suma importância. É assim que conseguimos interagir com o programa passando dados que serão utilizados para a resolução de problemas.

O dispositivo mais comum de entrada de dados é o teclado, mas existem diversos outros, tais como, a tela *touchscreen* (tela sensível ao toque), o microfone, o *scanner*, o disco rígido etc. Um algoritmo também pode receber dados diretamente de outro algoritmo, sem precisar passar por nenhum dispositivo tradicional de saída, mas isso não é muito comum.

A nome do comando de entrada padrão em pseudocódigo é **`Leia()`**. Em Java, o comando leia é feito em duas fases:

- A definição do dispositivo de entrada usando o comando:

```
Scanner entrada = new Scanner(System.in);
```

- A leitura do dado do dispositivo:

```
nome = entrada.next();
```

Com esses comandos é possível:

- Ler um texto;
- Ler um número inteiro; e
- Ler um número real.

Vamos ao exemplo dos comandos `Escreva` e `Escreval`

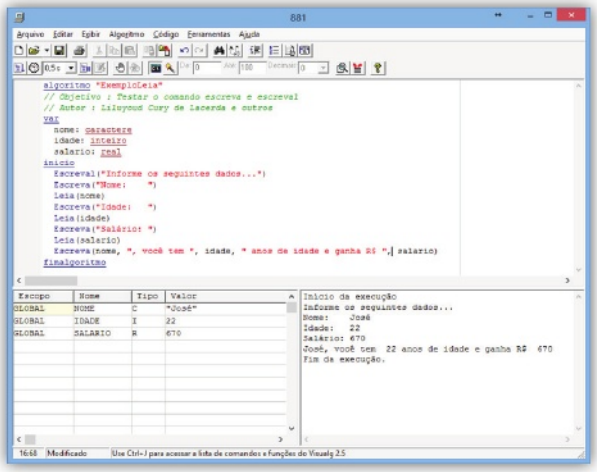


Figura 7.3: Exemplo dos comandos Escreva e Escreval.
Fonte: Autores.

Analise o código e perceba que o comando `Leia` é o mesmo para valores literais e numéricos. Entretanto, em Java, existem diferenças. Agora vamos escrever esse mesmo código em Java, no Eclipse.

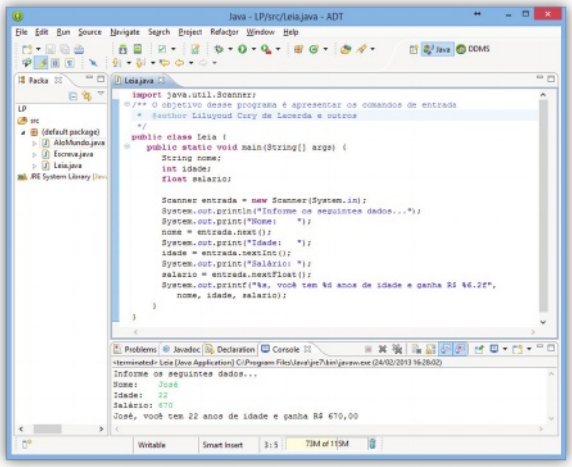


Figura 7.4: Exemplo dos comandos de entrada em Java e do printf.
Fonte: Autores.

Percebeu que, para cada tipo de dado que você for ler em Java, você precisa de um comando diferente? E o `printf`, que agora podemos definir em um único texto com parâmetros (%s para texto, %d para inteiro, %f para real) que serão substituídos por valores de variáveis.

7.3 Finalmente um algoritmo completo

Você se lembra de quando dissemos que um algoritmo nada mais é do que um mecanismo para entrada, processamento ou saída de dados? Agora você está preparado(a) para criar uma estrutura sequencial com todos eles.

Para exemplificar, vamos criar um algoritmo para descobrir o salário líquido de um funcionário, sabendo que é descontado do seu salário bruto 27,5%, referente ao imposto de renda. O algoritmo será implementado tanto em pseudocódigo quanto em Java. Vamos aos exemplos.

Algoritmo completo que calcula o salário líquido de um funcionário

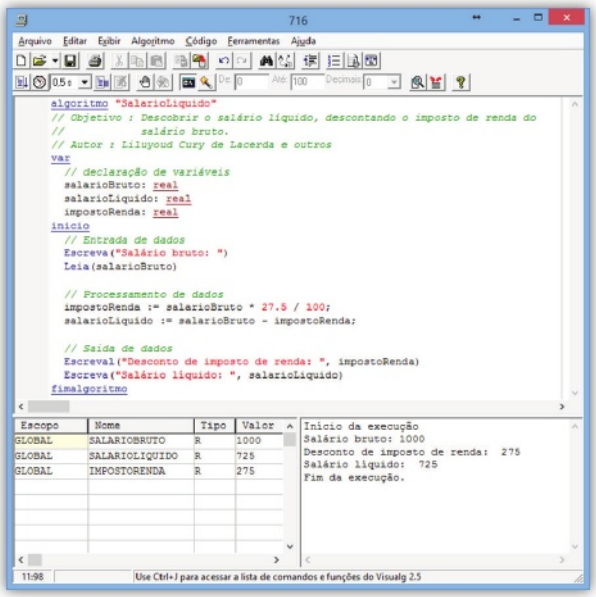


Figura 7.5: Algoritmo completo que calcula o salário líquido de um funcionário.
Fonte: Autores.

Algoritmo completo em Java que calcula o salário líquido de um funcionário

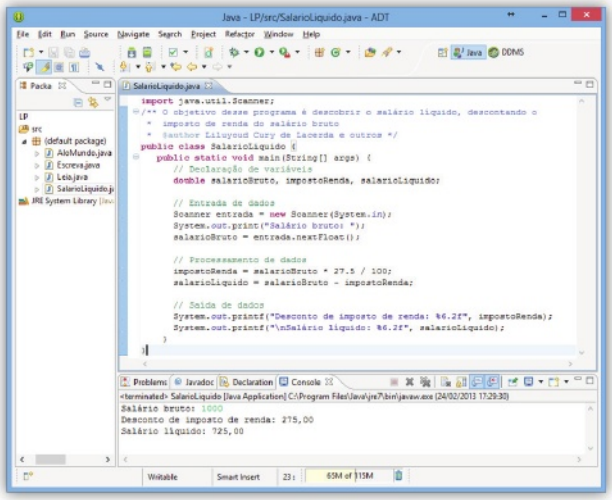


Figura 7.6: Algoritmo completo em Java que calcula o salário líquido de um funcionário.
Fonte: Autores.

Apenas um comentário no código da Figura 7.6: o caractere \n, no comando de saída `printf`, faz com que pule uma linha.

Agora, você poderá criar os algoritmos usando a estrutura de sequência (instruções em uma sequência linear) e aplicar todas as premissas necessárias de um programa padrão: a entrada, o processamento e a saída de dados.

Resumo

Você complementou seus conhecimentos básicos acerca da elaboração de algoritmos. Nesta aula, tratamos da estrutura sequencial em um algoritmo, ou seja, um conjunto de ações executadas em sequência linear. Vimos também os comandos de saída e entrada de dados em pseudocódigo e em Java. E, finalmente, mostramos um algoritmo completo que calcula o salário líquido de um funcionário, em pseudocódigo e em Java.

Nas atividades de aprendizagem desta aula, temos um número bem maior de exercícios para que você possa praticar. Lembre-se: "aprender a progra-

mar não é apenas leitura e estudo, é prática". Então vamos às atividades.

Atividades de aprendizagem

1. Faça um algoritmo em VisualG ou em Java que receba três números inteiros e calcule a soma deles.

2. Faça um algoritmo em VisualG ou em Java que receba quatro números reais e calcule a média deles.

3. Faça um algoritmo em VisualG ou em Java que receba duas notas de prova, seus respectivos pesos e calcule a média ponderada deles.

4. Faça um algoritmo em VisualG ou em Java que receba o salário inicial, a porcentagem de aumento e calcule o novo salário.

5. Faça um algoritmo em VisualG ou em Java que receba o salário-base e a quantidade de dependentes, e calcule o salário líquido sabendo que, para cada filho, o funcionário recebe o salário família de R$ 32,00 e, ao final, paga imposto de renda de 27,5% sobre o salário bruto.

6. Faça um algoritmo em VisualG ou em Java que receba os valores antigo e novo de um produto, e calcule a porcentagem de aumento que o mesmo teve.

7. Faça um algoritmo em VisualG ou em Java que receba o tamanho do raio de um círculo e faça o cálculo do diâmetro, do perímetro e da área do círculo. Calcule também o volume se esse círculo fosse a base para se criar uma esfera.

8. Faça um algoritmo em VisualG ou em Java que receba a velocidade de um veículo em Km/h (quilômetros por hora) e a transforme em m/s (metros por segundo).

Caro(a) estudante,

Parabéns por ter completado seus conhecimentos básicos acerca da elaboração de algoritmos. Agora, poderá criá-los usando a estrutura de sequência (instruções em uma sequência linear) e aplicar todas as premissas necessárias de um programa padrão: a entrada, o processamento e a saída de dados. Na próxima aula, abordaremos as estruturas de decisão. Até lá!



Aula 8. Estruturas de decisão

Objetivos:

- reconhecer uma estrutura de decisão;
- identificar os tipos de estrutura de decisão; e
- aplicar estruturas de decisão em algoritmos.

Caro(a) estudante,

Nas aulas anteriores resolvemos problemas de complexidade bem limitada, baseadas em estruturas sequenciais simples. Nesta aula, você irá estudar a elaboração de algoritmos mais complexos, saindo do tradicional passo a passo linear para fluxos de execução que podem sofrer desvios na sequência baseados em expressões lógicas, isto é, o computador será capaz de tomar uma decisão sobre o que será executado ou não. Interessante, não? Então vamos estudar com bastante afinco esta aula.

8.1 Introdução

Estrutura de decisão ou estrutura condicional é basicamente um mecanismo capaz de verificar previamente determinadas condições, baseadas em expressões lógicas, para a realização de uma ou mais instruções.

A expressão lógica em uma estrutura de decisão é o mecanismo usado para que o computador tome sua decisão. Caso ela seja verdadeira, o fluxo de execução do algoritmo segue um caminho; caso seja falsa, segue outro.

8.2 Tipos de estruturas de decisão

As estruturas de decisão podem ser de três tipos:

- Estruturas de decisão simples ou estrutura **Se**;
- Estruturas de decisão composta ou estrutura **Se** – **Senão**; e
- Estruturas de decisão encadeada ou estrutura **Se** – **Senão** – **Se**.

8.2.1 Estrutura de decisão "Se"

A estrutura de decisão Se é a mais simples, pois existe apenas um desvio de fluxo na execução do algoritmo. Esse desvio está condicionado a uma expressão booleana: quando verdadeira, não há desvio, mas caso seja falsa, o desvio é efetuado.

Para uma melhor visualização de como esse desvio acontece (como o computador toma decisões), vamos comparar o fluxograma de execução de um algoritmo com estrutura sequencial com o da estrutura condicional:

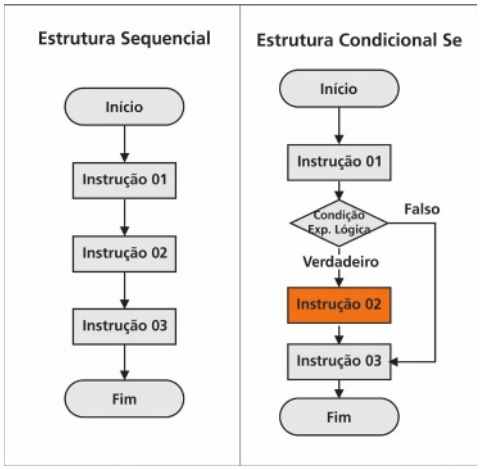


Figura 8.1: Estrutura sequencial *vs* estrutura condicional.
Fonte: Autores.

A sintaxe da estrutura condicional Se - Senão tanto em pseudocódigo quanto em Java é:

TABELA 8.1 - Sintaxe da estrutura condicional Se

Pseudocódigo	Java

Se (expressão lógica) Então
  Comando ou Bloco de Comandos
Fimse
	// Um único comando a ser controlado
	if(expressão lógica)
	  Comando;
	
	// Vários comandos a serem
	// controlados
	if(expressão lógica) {
	  Bloco de Comandos;
	}

Fonte: Autores.

- De acordo com a sintaxe, o `Comando` ou `Bloco de Comandos` só será executado se a expressão lógica for `verdadeira`.

- Em Java, quando temos apenas um comando para ser executado o uso das chaves é opcional.

Para entender melhor a estrutura de decisão Se, vamos a um exemplo prático: dado o salário de uma pessoa, o imposto de renda pode ser cobrado caso ele ganhe mais de R$ 1.500,00 mensais. Se isso acontecer, a porcentagem do imposto é de 15% sobre o salário bruto, caso contrário, o imposto a pagar é zero.

```
1  algoritmo "ExemploSe"
2  var
3    salarioBruto: real
4    salarioLiquido: real
5    impostoRenda: real
6  inicio
7    // Entrada de dados
8    Escreva("Salário bruto: ")
9    Leia(salarioBruto)
10
11   // Processamento de dados
12   impostoRenda := 0
13   Se (salarioBruto > 1500) entao
14     impostoRenda := salarioBruto * 0.15;
15   Fimse
16   salarioLiquido := salarioBruto - impostoRenda
17
18   // Saida de dados
19   Escreval("Desconto de imposto de renda: ", impostoRenda)
20   Escreva("Salário liquido: ", salarioLiquido)
21 fimalgoritmo
```

Figura 8.2: Exemplo da estrutura condicional Se em pseudocódigo.
Fonte: Autores.

Analise o código e verifique que:

- Caso a expressão lógica (`salarioBruto > 1500`) seja `verdadeira`, então o imposto de renda será calculado (linhas 13 a 15);

- Caso a expressão lógica (`salarioBruto > 1500`) seja `falsa`, então o imposto de renda não será calculado, e ele continuará sendo zero (da maneira que ele foi inicializado na linha 12).

No segundo caso, quando a expressão lógica é falsa, ocorre um desvio do fluxo do algoritmo, fazendo com que o comando da linha 14 não seja executado de acordo com a sequência do algoritmo. Muito legal esse mecanismo de controle de fluxo, não é? Agora vamos ver o mesmo algoritmo em Java:

```
1  import java.util.Scanner;
2
3  public class ExemploSe {
4     public static void main(String[] args) {
5        // Declaração de variáveis
6        double salarioBruto, impostoRenda, salarioLiquido;
7
8        // Entrada de dados
9        Scanner entrada = new Scanner(System.in);
10       System.out.print("Salário bruto: ");
11       salarioBruto = entrada.nextFloat();
12
13       // Processamento de dados
14       impostoRenda = 0;
15       if (salarioBruto > 1500) {
16          impostoRenda = salarioBruto * 0.15;
17       }
18       salarioLiquido = salarioBruto - impostoRenda;
19
20       // Saída de dados
21       System.out.printf("Desconto de imposto de renda: %6.2f", impostoRenda);
22       System.out.printf("\nSalário liquido: %6.2f", salarioLiquido);
23    }
24 }
```

Figura 8.3: Exemplo da estrutura condicional Se em Java.
Fonte: Autores.

Tanto o algoritmo em pseudocódigo quanto em Java deverão ter as mesmas saídas:

TABELA 8.2 - Exemplos de saída de um algoritmo com a estrutura condicional Se.

(salarioBruto > 1500) for:	
Falso	**Verdadeiro**
Salário bruto: 1000 Desconto de imposto de renda:0 Salário líquido:1000	Salário bruto: 1501 Desconto de imposto de renda: **225.15** Salário líquido:1275.85

Fonte: Autores.

8.2.2 Estrutura de decisão "Se – Senão"

A estrutura de decisão **Se - Senão** é mais completa, pois permite dois fluxos de execução no algoritmo, um para quando a expressão booleana for verdadeira e outro para quando for falsa. Sendo assim, o fluxo escolhido para a execução está condicionado a essa expressão, que, caso seja verdadeira, um conjunto de comandos é executado, mas caso seja falsa, executa-se outro conjunto.

Para você ter uma melhor visualização de como essa escolha de fluxo acontece, vamos comparar o fluxograma de execução de um algoritmo com estrutura Se com o da estrutura **Se - Senão**:

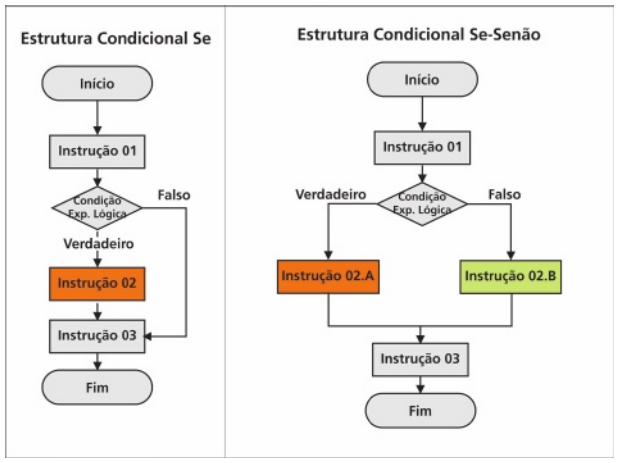


Figura 8.4: Estrutura Se *versus* Se – Senão.
Fonte: Autores.

A sintaxe da estrutura condicional **Se - Senão** tanto em pseudocódigo quanto em Java é:

TABELA 8.3 - Sintaxe da estrutura condicional Se - Senão.

Pseudocódigo	Java

`Se (expressão lógica) Então`
`  Comando 1 ou Bloco de Comandos 1`
`Senão`
`  Comando 1 ou Bloco de Comandos 1`
`Fimse`
	`if (expressão lógica)`
	`  Comando1;`
	`else`
	`  Comando2;`
	
	`if (expressão lógica) {`
	`  Bloco de Comandos 1;`
	`} else {`
	`  Bloco de Comandos 2;`
	`}`

Fonte: Autores.

De acordo com a sintaxe, o `Comando 1` ou `Bloco de Comandos 1` só será executado se a expressão lógica for `verdadeira`. Caso seja `falsa`, será executado o `Comando 2` ou `Bloco de Comandos 2`.

Para entender melhor a estrutura de decisão **Se - Senão**, vamos a um exemplo prático: dado um número, verifique se o mesmo é par ou ímpar.

```
1  algoritmo "ExemploSeSenao"
2  var
3    numero: inteiro
4    saida: caractere
5  inicio
6    // Entrada de dados
7    Escreva("Informe um número: ")
8    Leia(numero)
9
10   // Processamento de dados
11   Se (numero mod 2 = 0) entao
12     saida := "O número é par"
13   Senao
14     saida := "O número é impar"
15   Fimse
16
17   // Saida de dados
18   Escreva(saida)
19 fimalgoritmo
```

Figura 8.5: Exemplo da estrutura condicional Se - Senão em pseudocódigo.
Fonte: Autores.

Analise o código e verifique:

- Para saber se um número é par, basta usar uma propriedade matemática: caso o resto da divisão desse número por 2 seja 0, então ele é par.

- Analogicamente, se o resto da divisão de qualquer número por 2 for 1, significa que ele é ímpar.

```
import java.util.Scanner;

public class ExemploSeSenao {
    public static void main(String[] args) {
        // Declaração de variáveis
        int numero;
        String saida;

        // Entrada de dados
        Scanner entrada = new Scanner(System.in);
        System.out.print("Informe um número: ");
        numero = entrada.nextInt();

        // Processamento de dados
        if (numero % 2 == 0) {
            saida = "O número é par";
        } else {
            saida = "O número é impar";
        }

        // Saida de dados
        System.out.print(saida);
    }
}
```

Figura 8.6: Exemplo da estrutura condicional Se - Senão em pseudocódigo.
Fonte: Autores.

O mesmo algoritmo em Java seria:

8.2.3 Estrutura de decisão "Se – Senão – Se"

A estrutura de decisão **Se – Senão - Se** é a mais complexa de todas, pois permite o encadeamento de um **Se** logo após o **Senão** de outro **Se**, isto é, é possível encadear um **Se** dentro de outro, formando assim uma única estrutura.

Mesmo sendo complexo, às vezes é a solução mais fácil para alguns tipos de algoritmo, que precisam testar diversas condições e tomar uma decisão sobre qual fluxo de comandos seguir, que só pode ser um (ou nenhum), justamente aquele cuja condição for satisfeita, isto é, cuja expressão lógica for verdadeira.

Para você ter uma melhor visualização de como essa estrutura encadeada de Se é organizada, a seguir temos um exemplo de fluxograma de execução de um algoritmo com estrutura **Se – Senão - Se**:

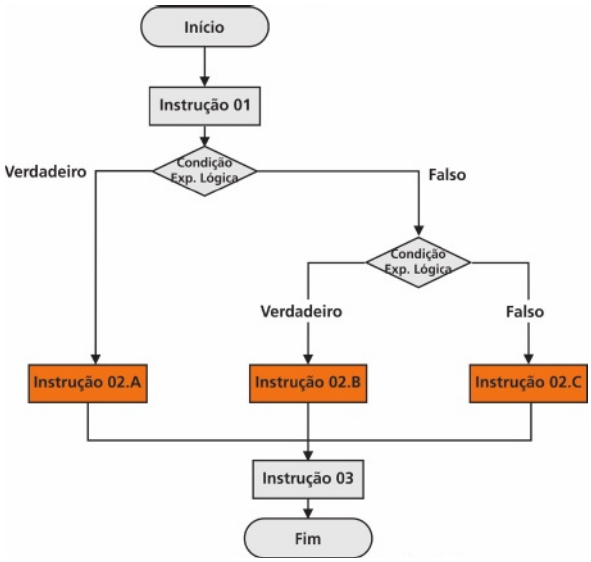


Figura 8.7: Fluxograma da Estrutura de decisão Se – Senão – Se.
Fonte: Autores.

Em pseudocódigo, essa estrutura encadeada nada mais é do que um Se dentro de outro Se, mas, em Java, é possível aninhar as estruturas Se como se fossem um só comando. Veja o exemplo a seguir:

TABELA 8.4 - Sintaxe da estrutura condicional Se – Senão - Se.

Pseudocódigo	Java

Se (expressão lógica 1) Então
Bloco de Comandos 1
Senão
Se (expressão lógica 2) Então
Bloco de Comandos 2
Senão
...
...
Se (expressão lógica N) Então
Bloco de Comandos N
Senão
Bloco de Comandos Senão
Fimse
Fimse
Fimse
	if (expressão lógica 1) {
	Bloco de Comandos 1;
	} else if (expressão lógica 2) {
	Bloco de Comandos 2;
	} else if (expressão lógica 3) {
	Bloco de Comandos 3;
	} else if (expressão lógica 4) {
	Bloco de Comandos 4;
	} else if... {
	...
	} else if(expressão lógica N) {
	Bloco de Comandos N;
	} else {
	Bloco de Comandos Senão
	}

Fonte: Autores.

Vejamos algumas propriedades importantes da estrutura de decisão Se – Senão – Se:

- O `Bloco de Comandos 1` será somente executada se a `expressão lógica 1` for `verdadeira`. Após a execução do `Bloco de Comandos 1`, o fluxo de execução sai da estrutura de decisão **Se – Senão – Se**, mesmo que haja outras expressões verdadeiras após a `expressão lógica 1`.

- O `Bloco de Comandos 2` será somente executado se a `expressão lógica 1` for `falsa` e a `expressão lógica 2` for `verdadeira`. Após a execução do `Bloco de Comandos 2`, o fluxo de execução sai da estrutura de decisão **Se – Senão – Se**, mesmo que haja outras expressões verdadeiras após a `expressão lógica 2`.

- O `Bloco de Comandos 3` somente será executado se as `expressões lógicas 1 e 2` forem `falsas` e a `expressão lógica 3` for `verdadeira`. Após a execução do `Bloco de Comandos 3`, o fluxo de execução sai da estrutura de decisão **Se – Senão – Se**, mesmo que haja outras expressões verdadeiras após essa expressão.

- O `bloco de comandos N` será somente executado se as `expressões lógicas 1 a N-1` forem `falsas` e a `expressão lógica N` for `verdadeira`. Após a execução do `Bloco de Comandos N`, o fluxo de execução sai da estrutura de decisão **Se – Senão – Se**.

- Os comandos do bloco `else` serão somente executados se nenhuma expressão lógica anterior for verdadeira.

Para entender melhor a estrutura de decisão **Se – Senão - Se**, vamos a um exemplo prático: dadas duas notas, caso a média delas seja maior que 7, o aluno estará aprovado; caso seja menor do que 3, o aluno estará reprovado; e se estiver entre 3 e 7, o aluno vai para a prova final.

O mesmo algoritmo em Java seria:

```
algoritmo "ExemploSeSenaoSe"
var
  nota1, nota2, media: real
  saida: caractere
inicio
  // Entrada de dados
  Escreva("Informe a 1a nota do aluno: ")
  Leia(nota1)
  Escreva("Informe a 2a nota do aluno: ")
  Leia(nota2)

  // Processamento de dados
  media := (nota1 + nota2) / 2
  Se (media >= 7) Entao
    saida := "Aluno APROVADO"
  Senao
    Se (media < 3) Entao
      saida := "Aluno REPROVADO"
    Senao
      saida := "Aluno para PROVA FINAL"
    Fimse
  FimSe

  // Saida de dados
  Escreva(saida)
fimalgoritmo
```

Figura 8.8: Exemplo da estrutura condicional Se – Senão - Se em pseudocódigo.
Fonte: Autores.

```
import java.util.Scanner;

public class ExemploSeSenaoSe {
   public static void main(String[] args) {
      // Declaração de variáveis
      double nota1, nota2, media;
      String saida;

      // Entrada de dados
      Scanner entrada = new Scanner(System.in);
      System.out.print("Informe a 1a nota do aluno: ");
      nota1 = entrada.nextDouble();
      System.out.print("Informe a 2a nota do aluno: ");
      nota2 = entrada.nextDouble();

      // Processamento de dados
      media = (nota1 + nota2) / 2;
      if (media >= 7)
         saida = "Aluno APROVADO";
      else if (media < 3)
         saida = "Aluno REPROVADO";
      else
         saida = "Aluno para PROVA FINAL";

      // Saida de dados
      System.out.print(saida);
      entrada.close();
   }
}
```

Figura 8.9: Exemplo da estrutura condicional Se - Senão - Se em Java.
Fonte: Autores.

Graças às estruturas de decisão, podemos fazer o computador tomar decisões e executar um conjunto de códigos que você quiser, de acordo com a sua lógica. Agora vamos ao resumo da aula.

Resumo

Adicionamos nesta aula conhecimentos importantíssimos para a elaboração de algoritmos mais complexos. Tratamos das estruturas de decisão simples, compostas e encadeadas. Vimos que a estrutura "**Se**" é a mais simples, pois há apenas um desvio de fluxo na execução do algoritmo. Esse desvio está condicionado a uma expressão booleana: quando verdadeira, não há desvio, mas, caso seja falsa, o desvio é efetuado. A estrutura de decisão composta "**Se Então**" é mais completa, pois permite dois fluxos de execução do algoritmo, um quando a expressão booleana for verdadeira e outra para quando for falsa. Já a estrutura encadeada "**Se - Senão - Se**" é a mais complexa, pois permite o encadeamento de um se logo após o **Senão** de outro **Se**, istoé, é possível encadear um **Se** dentro de outro, formando, assim, uma única estrutura.

Assim como na aula anterior, temos um número maior de exercícios. Vamos praticar!

Atividades de aprendizagem

1. Faça um algoritmo em VisualG ou em Java que, dados dois números, mostre qual é o maior deles.

2. Faça um algoritmo em VisualG ou em Java que, dado um número, verifique se o mesmo é divisível por 3.

3. Faça um algoritmo em VisualG ou em Java que, dado o tamanho de três retas, seja possível construir um triângulo com elas.

4. Faça um algoritmo em VisualG ou em Java que dado um número, verifique se o mesmo é divisível por 3 e 4, mas que não seja divisível por 5.

5. Faça um algoritmo em VisualG ou em Java que, dados três números, mostre-os -em ordem crescente.

6. Faça um algoritmo em VisualG ou em Java que, dado o salário bruto de uma pessoa, calcule o salário líquido sabendo que há o desconto de dois impostos, de acordo com a tabela abaixo:

Tabela IRPF	Tabela INSS
Salário até R$ 1.500,00: 0% de imposto, Salário até R$ 3.000,00: 15% de imposto, Salário acima de R$ 3.000,00: 27,5% de imposto,	Imposto fixo de 11% do salário, sendo que o total do imposto não pode ser superior a R$ 400,00.

Olá, viu como é possível um computador tomar decisões e assim executar tarefas mais complexas? Muito legal, não? Na próxima aula, você irá complementar os seus conhecimentos de algoritmos com as estruturas de seleção,

um conceito bem parecido com o da estrutura de decisão. Até a próxima!

Aula 9. Estruturas de seleção

Objetivos:

- reconhecer uma estrutura de seleção; e
- aplicar estruturas de seleção em algoritmos.

Caro(a) estudante,

Trataremos agora da estrutura de seleção, que é uma estrutura complementar às estruturas de decisão, isso porque ela é capaz de decidir qual fluxo de comandos executar, mas usando como parâmetro o valor de uma variável, em vez do resultado de uma expressão lógica.

9.1 Introdução

A estrutura de seleção é muito utilizada em programação, pois serve para controlar várias ações diferentes, de acordo com o valor de um parâmetro (variável) definido dentro dele. Diferentemente das estruturas de decisão, a de seleção tem número pré-definido de escolhas possíveis, e, consequentemente, de caminhos possíveis.

A instrução usada para definir uma estrutura de seleção é:

- Escolha no VisualG;
- switch no Java.



Toda estrutura de seleção pode ser substituída por uma estrutura de decisão equivalente. Geralmente, nesse caso, é usada a estrutura de decisão Se - Senão - Se. Entretanto, em relação ao desempenho, a análise de uma estrutura de seleção consome menos tempo de processamento que uma de decisão.

9.2 Sintaxe do comando escolha

A sintaxe do comando escolha, tanto no VisualG quanto no Java, é bem parecida:

TABELA 9.1: Sintaxe da estrutura de seleção.

Pseudocódigo	Java

Escolha(variavel)
  Caso "qualquer valor 1"
  Bloco de Comandos 1
  Caso "qualquer valor 2"
  Bloco de Comandos 2
  Caso "qualquer valor 3"
  Bloco de Comandos 3
  Caso ...
  Caso "qualquer valor N"
  Bloco de Comandos N
  OutroCaso
  Bloco de Comandos OutroCaso
FimEscolha
	switch (variavel) {
	  case valorInteiro1:
	  Bloco de Comandos 1;
	  break;
	  case valorInteiro2:
	  Bloco de Comandos 1;
	  break;
	  case ...:
	  case valorInteiroN:
	  Bloco de Comandos N;
	  break;
	  default:
	  Bloco de Comandos default;
	  break;
	}

Fonte: Autores.

Entretanto, temos algumas considerações a fazer:

- No pseudocódigo, a variável pode ser numérica e textual; no Java, somente numérico do tipo inteiro. Em ambos os casos ela não pode ser uma expressão lógica.

- No Java, para impedir que o código continue sendo executado, é necessário usar o comando break.

Explicando melhor o funcionamento da estrutura de seleção, temos:

- O comando Escolha recebe uma variável e abre um bloco de dados;

- Dentro desse bloco de dados há os cases;

- Cada case recebe um valor único, ou seja, que não pode ser repetido no mesmo bloco de dados;

- Após o valor único, colocamos todo código que deverá ser executado (pode haver quantas linhas de código forem necessárias);

- Baseado no valor da variável, ele verifica qual case tem esse valor e executa o código correspondente;

- Caso não ache nenhum valor correspondente nos cases, ele executa a

saída-padrão (`OutroCase` em VisualG e `default` em Java).

Para entender melhor a estrutura de seleção `Escolha - Caso`, vamos a um exemplo prático:

Criar um menu de comandos que mostre três mensagens, de acordo com a opção escolhida: Bom dia, Boa Tarde ou Boa Noite.

```
1  algoritmo "ExemploEscolha"
2  var
3    opcao: inteiro
4    saudacao: caractere
5  inicio
6     Escreval("+-------------------------------+")
7     Escreval("|         MENU DE OPÇÕES        |")
8     Escreval("+-------------------------------+")
9     Escreval("| [1] - Dizer bom dia           |")
10    Escreval("| [2] - Dizer boa tarde         |")
11    Escreval("| [3] - Dizer boa noite         |")
12    Escreval("+-------------------------------+")
13    Escreva("Digite sua opção: ")
14    Leia(opcao)
15    Escolha (opcao)
16      Caso 1
17        saudacao := "Bom Dia!"
18      Caso 2
19        saudacao := "Boa Tarde!"
20      Caso 3
21        saudacao := "Boa Noite!"
22      OutroCaso
23        saudacao := "Você selecionou uma opção errada"
24    FimEscolha
25    Escreva(saudacao)
26 fimalgoritmo
```

Figura 9.1: Exemplo de estrutura de seleção em VisualG.
Fonte: Autores.

Exemplo de estrutura de seleção em Java

```java
import java.util.Scanner;

public class ExemploEscolha {
   public static void main(String[] args) {
      int opcao;
      String saudacao;

      System.out.println("+--------------------------------+");
      System.out.println("|        MENU DE OPÇÕES          |");
      System.out.println("+--------------------------------+");
      System.out.println("| [1] - Dizer bom dia            |");
      System.out.println("| [2] - Dizer boa tarde          |");
      System.out.println("| [3] - Dizer boa noite          |");
      System.out.println("+--------------------------------+");
      System.out.print("Digite sua opção: ");

      Scanner entrada = new Scanner(System.in);
      opcao = entrada.nextInt();

      switch (opcao) {
         case 1:
            saudacao = "Bom Dia!";
            break;
         case 2:
             saudacao = "Boa Tarde!";
            break;
         case 3:
             saudacao = "Boa Noite!";
            break;
         default:
             saudacao = "Você selecionou uma opção errada";
            break;
      }

      System.out.print(saudacao);
      entrada.close();
   }
}
```

Figura 9.2: Exemplo de estrutura de seleção em Java.
Fonte: Autores.

Com essa estrutura, fica fácil manipular vários fluxos de comandos, desde que consigamos quantificar as opções possíveis de execução sem usar expressões lógicas, usando somente valores de variáveis. Agora, vamos ao resumo da aula.

Resumo

Nesta aula, vimos mais uma estrutura de linguagem, a de seleção. Mostramos que a estrutura de seleção permite manipular vários fluxos de comandos, usando como parâmetro o valor de uma variável. Agora é sua vez de praticar.

Atividades de aprendizagem



1. Faça um algoritmo em VisualG ou em Java que use a estrutura de seleção na seguinte situação: dado um número de 1 a 12, mostrar o nome do mês correspondente. Caso o usuário não digite esse número, mostrar uma mensagem de erro.

2. Faça um algoritmo em VisualG que use a estrutura de seleção neste caso:

dado o nome de uma Unidade Federativa do Brasil – UF, escreva a capital dessa UF. Caso o usuário informe uma UF não existente, mostrar uma mensagem de erro.

3. Faça um algoritmo em VisualG ou em Java que, sabendo sua posição em um campeonato, escreva as seguintes mensagens:

- 1º colocado: Parabéns, você é o melhor.
- 2º colocado: Parabéns, você conseguiu.
- 3º colocado: Parabéns, você quase chegou lá.
- 4º colocado em diante: Parabéns, o importante é competir.

4. Faça um algoritmo em VisualG ou em Java que dados dois números, mostre um menu de opções em que você possa fazer as seguintes escolhas:

- Somar os dois números;
- Multiplicar os dois números;
- Subtrair os dois números;
- Dividir os dois números; e
- Calcular o 1º número elevado ao 2º (possível somente em Java – é necessário pesquisar sobre as funções matemáticas na linguagem).

Caro(a) estudante,

Estamos chegando ao final da disciplina Lógica de Programação. Agora, só falta mais uma aula para fechar o conteúdo necessário para que você consiga desenvolver algoritmo de alta qualidade. Na próxima aula, veremos uma estrutura muito importante na programação, que permite executar um conjunto de comandos mais de uma vez. Até mais!

Aula 10. Estruturas de repetição

Objetivos:

- reconhecer uma estrutura de repetição;
- identificar os tipos de estruturas de repetição; e
- aplicar estruturas de repetição em algoritmos.

Caro(a)estudante,

Já tivemos a oportunidade de estudar algoritmos sofisticados. Vamos, neste momento, ampliar ainda mais seus conhecimentos, apresentando algumas técnicas de programação que possibilitem repetir um bloco de comandos várias vezes. Vamos ao conteúdo!

10.1 Introdução

Em nosso cotidiano é comum executarmos a mesma ação repetitivamente até chegar a um objetivo, por exemplo, fixar um prego na madeira, quando temos que bater no prego repetidas vezes até que ele esteja fixo o suficiente para não cair.

Esse tipo de situação também é muito comum na programação, onde se faz necessário executar o mesmo conjunto de comandos mais de uma vez para chegar ao resultado desejado.

Se precisássemos calcular a média mais de uma vez, por exemplo, teríamos que executar (reiniciar) o programa CalcularMedia várias vezes, uma ação inviável ou, no mínimo, inconveniente.

Para resolver problemas como esse é que as linguagens de programação oferecem as estruturas de repetição (ou *looping*).

A quantidade de vezes a ser repetido pode ser ou não previamente conhecida, mas certamente deve ter um limite (ser finito). Se você não garantir que a repetição termine em algum momento, o seu algoritmo entra em **loop infinito**, isto é, ele para de funcionar, podendo até travar o computador, já que consume muitos recursos do equipamento.

Há um tempo, o programa rodava no mesmo espaço de memória e consumia o mesmo conjunto de recursos que o sistema operacional. Essa arquitetura de gerenciamento de processos (programas) era complicada, pois um determinado problema no seu programa poderia afetar todo o computador.

Atualmente, nos sistemas operacionais modernos, os programas rodam em uma área protegida (máquinas virtuais) e, caso haja algum problema nele, apenas o programa em questão é afetado, deixando o resto do ambiente computacional seguro (pelo menos deveria ser assim o funcionamento, mas até hoje vemos alguns travamentos em nossos computadores).

10.2 Tipos de estrutura de repetição

As estruturas de repetição podem ser de três tipos:

- Estrutura de repetição `Para`;
- Estrutura de repetição `Enquanto`;
- Estrutura de repetição `Repita`.

10.2.1 Para

A principal característica da estrutura de repetição `Para` é que ela executa um laço de repetição por um número predefinido de vezes. Para tanto, a estrutura `Para` tem o auxílio da figura do **contador**, uma variável inteira, que tem um valor inicial e deve chegar a um valor final.

Para exemplificar esse conceito, imagine que seu treinador peça para que você faça 10 abdominais. O que é mais importante para você conseguir fazer esse exercício? Se você respondeu "estar em forma", apesar de ter certa relevância, essa não é a resposta correta. O mais importante nesse caso é saber contar até 10.

Outro aspecto importante do contador é a forma de contar. Na estrutura Para é possível contar em ordem crescente (1, 2, 3..., 10), em forma decrescente (10, 9, 8,..., 1), pulando alguns passos (0, 2, 4, 6), entre muitas outras maneiras.

A sintaxe da estrutura de repetição `Para` em pseudocódigo é:

```
para <contador> de <valorInicial> ate <valorFinal> [passo] faca
   <bloco de comandos>
fimpara
```

Figura 10.1: Sintaxe do comando Para.
Fonte: Autores.

Onde:

- **contador**: é uma variável do tipo inteiro que controla o número de repetições do laço.
- **valorInicial:** identifica o valor inicial do contador.
- **valorFinal:** identifica o valor final do contador, no momento que o laço é interrompido.
- **passo:** é opcional, mas, quando presente, precedida pela palavra passo, é uma expressão que especifica o incremento que será acrescentado à variável contadora em cada repetição do laço. Exemplo: `passo 1` (a cada passo incrementa 1 o valor do contador – esse é o padrão, por isso pode ser omitido; `passo 5` (a cada passo incrementa em 5 o valor do contador); ou `passo -2` (a cada passo diminui em 2 o valor do contador).

Como exemplos de sintaxe, temos:

```
Algoritmo "ExemplosPara"
var i: inteiro
inicio
   // contar de 1 até 10
   para i de 1 ate 10 faca
    escreva (j:3)
   fimpara

   // contar de 10 até 1
   para i de 10 ate 1 passo -1 faca
   escreva (j:3)
   fimpara

fimalgoritmo
```

Figura 10.2: Exemplos da estrutura Para em pseudocódigo.
Fonte: Autores.

A sintaxe da estrutura de repetição Para em Java é:

```
for(valor inicial; condição de parada; incremento) {
   <bloco de comandos>;
}
```

Figura 10.3: Sintaxe do comando for.
Fonte: Autores.

Onde:

- **valor inicial:** é uma variável (contador) do tipo inteiro que controla o número de repetições do laço.
- **condição de parada:** é uma expressão lógica que, enquanto for verdadeira, o lação (*loop*) acontece. Quando a expressão booleana for falsa, a repetição para.
- **incremento:** é uma expressão aritmética que indica o incremento ou decremento da variável contadora.

Como exemplos de sintaxe, temos:

```
// contar de 1 até 10
for (int i = 1; i <= 10; i = i + 1) {
   System.out.println(i);
}

// contar 10 vezes
for (int i = 0; i < 9; i++) {
   System.out.println(i);
}

// contar de 10 até 1
for (int i = 10; i >= 1; i = i - 1) {
   System.out.println(i);
}

// mostrar os números pares menores que 100
for (int i = 0; i < 100; i = i + 2) {
   System.out.println(i);
}
```

Figura 10.4: Exemplos da estrutura Para em Java.
Fonte: Autores.

Para entender melhor a estrutura de repetição `Para`, vamos a um exemplo prático:

Criar um algoritmo que permita somar os N primeiros números de uma sequência de inteiros. Exemplo, para N = 5, calcular 1 + 2 + 3 + 4 + 5.

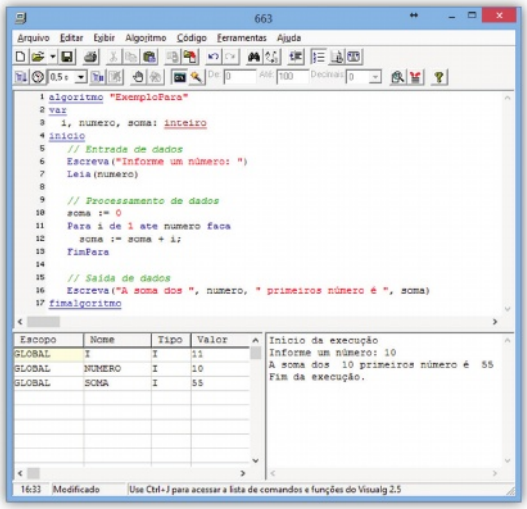


Figura 10.5: Exemplo prático da estrutura Para em pseudocódigo.
Fonte: Autores.

Em Java, o programa da figura 10.5 ficaria da seguinte maneira:

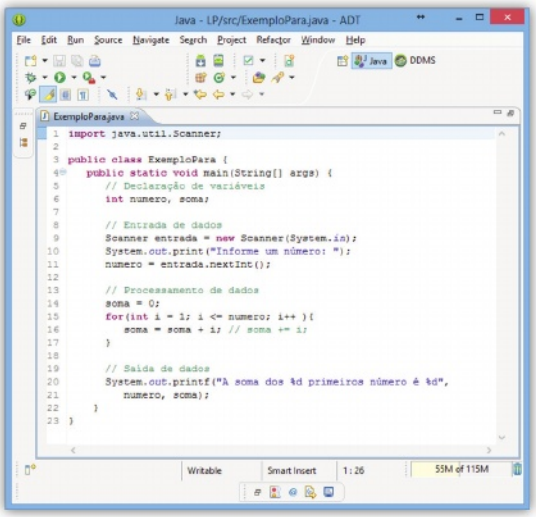


Figura 10.6: Exemplo prático da estrutura Para em Java.
Fonte: Autores.

10.2.2 Enquanto

A estrutura de repetição `Enquanto` é comumente usada quando o número de vezes que um bloco de comandos deverá ser executado não é conhecido, diferentemente do `Para`, que é predefinido.

Explicando melhor, o comando Para é do tipo "faça 10 abdominais" e o `Enquanto` é do tipo "faça abdominais enquanto não estiver cansado". Perceba que, no segundo tipo, não é possível precisar a quantidade de abdominais que a pessoa irá fazer.

Esse tipo de estrutura só é possível porque existe um teste lógico no início do comando `Enquanto`: esse comando primeiro avalia uma expressão lógica – caso ela seja verdadeira, faz a repetição do código; caso seja falsa, ela para a repetição, saindo, assim, do *loop*.

A sintaxe da estrutura de repetição `Enquanto` em pseudocódigo é:

```
Enquanto <expressão lógica> faca
   <bloco de comandos>
FimEnquanto
```

Figura 10.7: Sintaxe do comando Enquanto.
Fonte: Autores.

Onde:

- **Expressão lógica:** é uma expressão que, enquanto for verdadeira, permite a execução do bloco de comandos, mas, ao ficar falsa, o fluxo de execução do algoritmo sai da estrutura de repetição, parando essa atividade.

Como exemplos de sintaxe, temos:

```
Algoritmo "ExemplosEnquanto"
var a, b: inteiro
Inicio
   a := 10
   b := 20
    // o loop irá executar enquanto a for menor ou igual a b
    Enquanto (a <= b) faca
      escreval (a, " ", b)
      a = a + 1
      b = b - 1

    FimEnquanto
FimAlgoritmo
```

Figura 10.8: Sintaxe do comando Enquanto.
Fonte: Autores.

A sintaxe da estrutura de repetição `Enquanto` em Java é:

```
while (expressão lógica) {
   <bloco de comandos>;
}
```

Figura 10.9: Sintaxe do comando for.
Fonte: Autores.

O papel da expressão lógica no funcionamento da estrutura `while` segue os mesmos princípios mostrados na versão em pseudocódigo.

Como exemplos de sintaxe, temos:

```
int a = 10;
int b = 20;
// o loop irá executar enquanto a for menor ou igual a b
while(a <= b) {
   System.out.println(a + " " + i);
    a++; // incrementa o valor de a em 1
    b--; // decrementa o valor de b em 1
}
```

Figura 10.10: Exemplos da estrutura Enquanto em Java.
Fonte: Autores.

Para entender melhor a estrutura de repetição `Enquanto`, vamos a um exemplo prático:

Criar um algoritmo que mostre todos os elementos menores que 1000 na sequência de Fibonacci. Para conhecimento, a sequência de Fibonacci é aquela cujo próximo elemento é a soma dos dois anteriores. Ex.: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34...

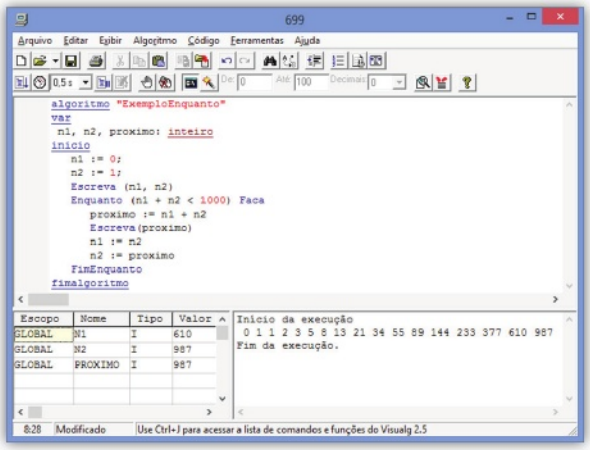


Figura 10.11: Exemplo prático da estrutura Enquanto em pseudocódigo.
Fonte: Autores.

Em Java, o programa da figura 10.11 ficaria da seguinte maneira:

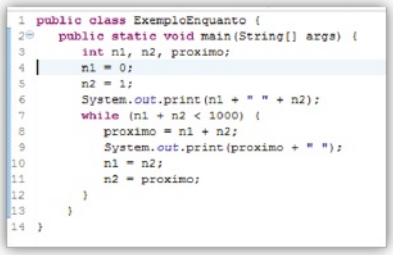


Figura 10.12: Exemplo prático da estrutura Enquanto em Java.
Fonte: Autores.

10.2.3 Faça ou Repita

As estruturas de repetição `Faça` ou `Repita`, assim como o `Enquanto`, são comumente usadas quando não é conhecido o número de vezes que um bloco de comandos deverá ser executado.

A única diferença entre eles é que o `Enquanto` testa a expressão lógica antes de executar o código e tanto o `Faça` quanto o `Repita` testam somente depois de executar o código. A principal consequência dessa diferença é que o bloco de comandos em `Faça` ou `Repita` será sempre executado pelo menos uma vez. Já no comando `Enquanto` pode acontecer que seu bloco de comandos não seja executado nem uma vez.

Não existe o equivalente da estrutura `Faça` no VisualG, apenas no Java e em outras linguagens mais modernas, tais como o C#, C++ e Javascript. Analogicamente, nessas linguagens, não existe o equivalente à estrutura `Repita`.

A principal diferença entre os comandos `Faça` e `Repita` é que o primeiro executa o *loop* enquanto a expressão lógica for verdadeira. O segundo executa o *loop* até que a expressão lógica seja verdadeira, isto é, no `Repita`, enquanto a expressão lógica for falsa, ele executa a repetição, no momento em que ela se tornar verdadeira, ela sai do *loop*.

A seguir temos um exemplo da sintaxe do comando `Repita` no VisualG.

```
Repita
   <bloco de comandos>
Ate <expressão logica>// repete até a expressão for ver-
dadeira
```

Figura 10.13: Sintaxe do comando Repita.
Fonte: Autores.

Onde:

- **Expressão lógica:** é uma expressão que, enquanto for falsa permite a execução do bloco de comandos, mas, ao ficar verdadeira, o fluxo de execução do algoritmo sai da estrutura de repetição.

A seguir temos um exemplo da sintaxe do comando `Faça` no Java.

```
do {
   <bloco de comandos>;
} while (expressão lógica); // repete enquanto a ex-
pressão for verdadeira
```

Figura 10.14: Sintaxe do comando do.
Fonte: Autores.

Onde:

- **Expressão lógica:** é uma expressão que, enquanto for verdadeira, permite a execução do bloco de comandos, mas, ao ficar falsa, é interrompida e o fluxo de execução do algoritmo sai da estrutura de repetição.

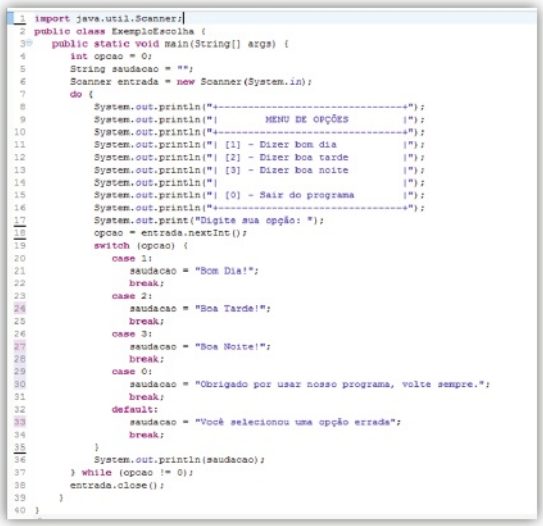
Para entender melhor as estruturas de repetição `Repita` e `Faça`, vamos a um exemplo prático:

Baseado no exemplo mostrado no item 9.2.2, crie um menu de comandos que mostre três mensagens, de acordo com a opção escolhida: Bom dia, Boa Tarde ou Boa Noite, e tenha uma 4ª opção perguntando se o usuário deseja sair do programa.

```
1  algoritmo "ExemploRepita"
2  var
3     opcao: inteiro
4     saudacao: caractere
5  inicio
6     Repita
7        Escreval("+--------------------------------+")
8        Escreval("|         MENU DE OPÇÕES         |")
9        Escreval("+--------------------------------+")
10       Escreval("| [1] - Dizer bom dia            |")
11       Escreval("| [2] - Dizer boa tarde          |")
12       Escreval("| [3] - Dizer boa noite          |")
13       Escreval("|                                |")
14       Escreval("| [0] - Sair do programa         |")
15       Escreval("+--------------------------------+")
16       Escreva("Digite sua opção: ")
17       Leia(opcao)
18       Escolha (opcao)
19          Caso 1
20             saudacao := "Bom Dia!"
21          Caso 2
22             saudacao := "Boa Tarde!"
23          Caso 3
24             saudacao := "Boa Noite!"
25          Caso 0
26             saudacao := "Obrigado por usar nosso programa, volte sempre."
27          OutroCaso
28             saudacao := "Você selecionou uma opção errada."
29       FimEscolha
30       Escreval(saudacao)
31    Ate (opcao = 0)
32 fimalgoritmo
```

Figura 10.15: Exemplo prático da estrutura Repita em VisualG.
Fonte: Autores.

Em Java, o programa ficaria da seguinte maneira:



```
import java.util.Scanner;
public class ExemploEscolha {
    public static void main(String[] args) {
        int opcao = 0;
        String saudacao = "";
        Scanner entrada = new Scanner(System.in);
        do {
            System.out.println("+--------------------------------+");
            System.out.println("|         MENU DE OPÇÕES         |");
            System.out.println("+--------------------------------+");
            System.out.println("| [1] - Dizer bom dia            |");
            System.out.println("| [2] - Dizer boa tarde          |");
            System.out.println("| [3] - Dizer boa noite          |");
            System.out.println("|                                |");
            System.out.println("| [0] - Sair do programa         |");
            System.out.println("+--------------------------------+");
            System.out.print("Digite sua opção: ");
            opcao = entrada.nextInt();
            switch (opcao) {
                case 1:
                    saudacao = "Bom Dia!";
                    break;
                case 2:
                    saudacao = "Boa Tarde!";
                    break;
                case 3:
                    saudacao = "Boa Noite!";
                    break;
                case 0:
                    saudacao = "Obrigado por usar nosso programa, volte sempre.";
                    break;
                default:
                    saudacao = "Você selecionou uma opção errada";
                    break;
            }
            System.out.println(saudacao);
        } while (opcao != 0);
        entrada.close();
    }
}
```

Figura 10.15: Exemplo prático da estrutura Faça em Java.
Fonte: Autores.

10.3 Comparação entre as estruturas de repetição

Se você ainda tem dúvidas sobre qual estrutura de repetição usar, não há problemas. Na verdade, você pode usar qualquer uma, pois todas conseguem resolver uma situação que envolva repetição de código.

A implementação de determinada solução às vezes pode ser mais simples em um tipo de estrutura do que em outro.

Mas, para conhecimento, a seguir nós temos uma tabela comparativa das estruturas de repetição. Assim, fica mais fácil tomar decisão sobre qual usar em cada caso.

TABELA 10.1 - Tabela comparativa entre as estruturas de repetição

Característica	Para	Enquanto	Faça	Repita
Usa contador	Sim	Opcional	Opcional	Opcional
Tem quantidade predefinida	Sim	Depende	Depende	Depende
Primeiro verifica depois executa	Sim	Sim	Não	Não
Primeiro executa depois verifica	Não	Não	Sim	Sim
Executa enquanto a expressão lógica é verdadeira	Sim	Sim	Sim	Não
Executa enquanto a expressão lógica é falsa	Não	Não	Não	Sim

Fonte: Autores.

Resumo

Essa foi a nossa última aula. Você estudou as estruturas de repetição para aperfeiçoar determinados tipos de processo. Mostramos que, na estrutura de decisão, o computador executa o mesmo bloco de comando quantas vezes forem necessárias, baseado em controles bem definidos de repetição para se chegar ao resultado pretendido.

Não se esqueça de realizar as atividades específicas desta aula para um melhor entendimento do assunto abordado.

Atividades de aprendizagem

1. Faça um algoritmo em VisualG ou em Java que calcule o saldo atual de uma poupança, dado o total depositado, a porcentagem do rendimento mensal e o número de meses que o valor está no banco.

2. Faça um algoritmo em VisualG ou em Java que calcule a soma dos N primeiros números, onde N é informado pelo usuário. Ex.: para N = 5, então calcular 1 + 2 + 3 + 4 + 5.

3. Faça um algoritmo em VisualG ou em Java que calcule o fatorial de um número inteiro.

4. Faça um algoritmo em VisualG ou em Java que leia A e B, ambos números inteiros e positivos, e calcule A^B.

5. Faça um algoritmo em VisualG ou em Java que leia um valor N inteiro e positivo, calcule e mostre o valor de E, conforme a fórmula a seguir:

$$E = 1 + 1/1! + 1/2! + 1/3! + ... + 1/N!$$

Palavras Finais

Caro(a) estudante,

Primeiramente, gostaríamos de parabenizá-lo(la) por ter concluído esta disciplina do curso Técnico em Informática. Estamos felizes por ter completado mais uma importante etapa na busca da aprendizagem e, consequentemente, do conhecimento. Chegamos ao final da disciplina e certamente não foi um percurso fácil, entretanto, durante nossa caminhada, adquirimos informações e pudemos realizar trocas de experiências, com isso estamos mais preparados para seguir em frente e superar os novos desafios que virão.

A lógica de programação é um ponto de partida. Não podemos esquecer que a busca pelo conhecimento é contínua, portanto, vamos praticar, e, como dica, por que não criar algoritmos para resolver problemas reais? A prática em programação é o diferencial da sua vida profissional – e nós a adquirimos com exercícios e simulações de problemas reais.

A você que chegou até aqui o nosso parabéns.

Guia de Soluções

Aula 1

1. Resposta pessoal.

2.

Resultado verdadeiro:	Resultado falso:
O ferro conduz eletricidade. O ouro conduz eletricidade. O chumbo conduz eletri cidade. A prata conduz eletricidade. Logo, todo metal conduz eletricidade.	Até agora, nenhuma mulher foi Presidente da República no Líbano. Logo, nenhuma mulher será Presidente da República no Líbano.

3. Só há movimento no carro se houver combustível.

O carro está em movimento.

Logo, há combustível no carro.

4.

I - B

II - B

III - C

5. Dona Rosa está de branco.

Dona Branca está de violeta.

Dona Violeta está de rosa.

6. http://www.youtube.com/watch?v=UUd3cucmL2A

Aula 2

1. Descrever passo a passo a resolução de um problema.

2. Ligar o computador.

Abrir o navegador.

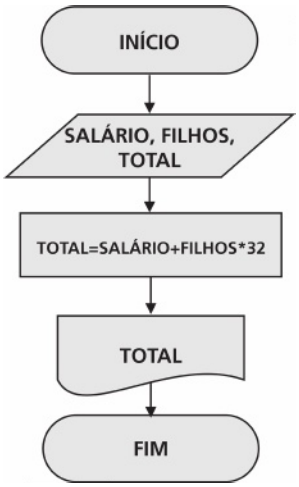
Acessar o site do e-mail.

Digitar o usuário e a senha.

Procurar e-mail com o documento.

Se encontrar o documento, abri-lo.

3.



4.
```
DECLARE numero, cubo: INTEIRO
INÍCIO
LEIA(numero)
cubo ← numero * numero * numero
ESCREVA(cubo)
FIM
```

Aula 3

1. Visual Studio – C#, VB, C++

NetBeans – Java

Delphi – Pascal

XCode – Objective C

2. Visual Studio com C#:

```
using System;
namespaceMeusProjetos
{
classProgram
    {
staticvoid Main(string[] args)
{
Console.Write("Digiteseunome: ");
string nome = Console.ReadLine();
Console.WriteLine("Seja bem vindo {0}", nome);
Console.ReadKey();
}
}
}
```

NetBeans com Java

Por ser um compilador Java, pode ser usado os mesmos algoritmos do Eclipse.

Delphi com Pascal

```
ProgramOla_Mundo;
begin
WriteLn('Olá, Mundo!');
end.
```

XCode com Objective-C

```
#import <stdio.h>

int main(void)
{
puts("Olá, Mundo!");
return 0;
}
```

Mais exemplos em: http://pt.wikipedia.org/wiki/Programa_Olá_Mundo

3. Os programas criados no VisualG e no Eclipse são simplesmente arquivos armazenados no disco que podem ser visualizados pelo Windows Explorer. Os arquivos em VisualG têm extensão .alg, e os arquivos em Java têm extensão .java.

No caso do Eclipse e outras IDEs mais modernas, são criados muitos outros arquivos que têm como objetivo descrever o projeto como um todo, mas o importante é saber onde estão armazenados os arquivos com os algoritmos criados.

4. É possível perceber que é bem mais fácil escrever um algoritmo usando o português (no caso do VisualG) do que em inglês (no caso do Java). Infelizmente, todas as maiores linguagens de programação foram criadas por pessoas cuja língua pátria era o inglês, por isso a predominância desse idioma nas linguagens de programação.

Aula 4

1 e 2.

```
algoritmo "Mensagem"

// Autor: Liluyoud, Sara e José Marcio
var
  // declaração de variáveis
Inicio
  // Algoritmo para mostrar uma mensagem
Escreva("+------------------------------+")
Escreva("| Programa Alô, Mundo          |")
Escreva("+------------------------------+")
Escreva("| Seja bem-vindo ao maravilhoso |")
Escreva("| mundo da programação.        |")
Escreva("|                              |")
Escreva("| Aproveite!!!                 |")
Escreva("+------------------------------+")
Fimalgoritmo
```

3.

Função do Algoritmo	Nome
Transforma dólar em real	Cambio
Calcular o índice de massa corporal de uma pessoa	CalculoIMC
Calcular a área de um triângulo	AreaTriangulo
Efetuar um saque na conta-corrente	ContaSaque

4.

```
algoritmo "Cambio"
var
valorReal, valorDolar, cotacao: real
Inicio
Escreva("Informe o valor em Dolar: ")
Leia(valorDolar)
Escreva("Informe a cotação do dia: ")
Leia(cotacao)
valorReal := valorDolar * cotacao
Escreva("O valor em real é ", valorReal)
Fimalgoritmo

algoritmo "CalculoIMC"
var
peso, altura, imc: real
Inicio
Escreva("Informe seu Peso: ")
Leia(peso)
Escreva("Informe sua Altura: ")
Leia(altura)
imc := peso / (altura * altura)
Escreva("Seu IMC é ", imc)
Fimalgoritmo

algoritmo "AreaTriangulo"
var
base, altura, area: real
Inicio
Escreva("Informe o largura da base do triângulo: ")
Leia(base)
Escreva("Informe sua Altura: ")
Leia(altura)
area := (base * altura) / 2
Escreva("A área do triângulo é ", area)
Fimalgoritmo

algoritmo "ContaSaque"
var
saldo, valorSaque: real
Inicio
Escreva("Informe o saldo atual da conta: ")
Leia(saldo)
Escreva("Informe o valor do Saque: ")
Leia(valorSaque)
saldo := saldo - valorSaque
Escreva("O novo saldo é de ", saldo)
fimalgoritmo
```

Aula 5

1. **a)** `XXX válido`

b) `a123 válido`

c) `(A) inválido`

Não podemos usar caracteres especiais no nome, nesse caso foram usados os ( ).

d) `"NOME" inválido`

Não podemos usar caracteres especiais no nome, nesse caso foram usadas as " ".

e) `#66 inválido`

Não podemos usar caracteres especiais no nome, nesse caso foi usado o #.

f) `dia/ano/mês inválido`

Não podemos usar caracteres especiais no nome, nesse caso foi usada a /.

g) `1abc inválido`

Não podemos iniciar uma variável com número.

h) `Olá Pessoal inválido`

Não pode haver espaços em uma variável, e recomenda-se que não sejam usado acentos também.

i) `a.b inválido`

Não podemos usar caracteres especiais no nome, nesse caso foi usado o ".".

j) `c*d inválido`

Não podemos usar caracteres especiais no nome, nesse caso foi usado o *.

2.

O que será armazenado?	Sintaxe Pseudocódigo Sintaxe Java
O salário de um funcionário	`salario: real` `double salario;`
A nota e a média de um aluno	`nota, media: real` `double nota, media;`
Os dias de atraso de uma prestação	`diasEmAtraso: inteiro` `intdiasEmAtraso;`
O CPF de uma pessoa	`cpf: caractere` `stringcpf;`
Se a família tem casa própria	`casaPropria: lógico` `booleancasaPropria;`
Uma carta	`carta: caractere` `string carta;`
O endereço completo do cliente	`endereco: caractere` `stringendereco;`
Se o produto é novo ou usado	`novo: lógico` `boolean novo;`

Aula 6

1. Operando: são os argumentos de um operador, isto é, as informações necessárias para um operador realizar sua função.

Operador: função com domínio específico.

Expressão: é uma combinação de operandos e operadores.

Exemplo: a * b

a e **b** são os operandos

* é o operador

a * **b** é a expressão

2. Expressões matemáticas ou aritméticas

(a + b) / 2

Expressões lógicas

(a + b) <= (a * b)

3. Sendo a = 2, b = 5 e c = 3, temos:

a) `b * 2 – a * c=>5 * 2 – 2 * 3=> 10 – 6 => 4`

b) `b + a * 2 – a + c * 3=>5 + 2 * 2 – 2 + 3 * 3=> 5 + 4 – 2 + 9 => 16`

c) `(b + a) * (2 – a) + c * 3=>(5 + 2) * (2 – 2) + 3 * 3 => 7 * 0 + 9 => 9`

d) `(b + a) * ((2 – a) + c ) * 3 =>(5 + 2) * ((2 – 2) + 3) * 3 => 7 + (0 + 3) * 3 => 7 + 3 * 3 => 16`

e) `(c + a * a – a + 1) / (c * b * a + c – a * c)=> (3 + 2 * 2 – 2 + 1) / (3 * 5 * 2 + 3 – 2 * 3) = > (3 + 4 – 2 + 1) / (30 + 3 – 6) => 6/27`

4.

a) `v E f OU f E v=>f OU f E v=> f E v => f`

b) `(v E f) OU (f E v)=> (f) OU (f) => f`

c) `(NÃO (v E f)) E (f E v)=>(NÃO (f)) E (f) =>(v) E (f) => f`

d) `NÃO ((v E f) OU (f E v))`

e) `NÃO ((NÃO(v E f)) E (NÃO(f E v)))`

5. Sabendo que a = 1, b = 2 e c = 3, demonstre a execução passo a passo da seguinte expressão:

```
NÃO (b * c <> 8 OU ((a * c) - a <= c * c / a)) OU (b MOD a = 1 E
                        4 MOD a = 0)
```

```
NÃO (2 * 3<> 8 OU ((1 * 3) - 1<= 3 * 3 / 1)) OU (2 MOD 1 = 1 E 4
                        MOD 1 = 0)
```

```
NÃO (6<> 8 OU (3 - 1<= 9)) OU (0 = 1 E 0 = 0)

NÃO (6<> 8 OU (2<= 9)) OU (f E v)

NÃO (6<> 8 OU v) OU (f)

NÃO (v OU v) OU (f)

NÃO (v) OU (f)

f OU f

f
```

Aula 7

1.

```
Algoritmo "Prog_7_1"
Var
n1, n2, n3, soma: Inteiro
Inicio
Leia(n1)
Leia(n2)
Leia(n3)
soma:= n1 + n2 + n3
Escreva(soma)
FimAlgoritmo
```

2.

```
Algoritmo "Prog_7_2"
Var
n1, n2, n3, n4, media: Real
Inicio
Leia(n1)
Leia(n2)
Leia(n3)
Leia(n4)
media := (n1 + n2 + n3 + n4) / 4
Escreva(media)
FimAlgoritmo
```

3.

```
Algoritmo "Prog_7_3"
Var
n1, n2, p1, p2, media: Real
Inicio
Leia(n1)
Leia(n2)
Leia(p1)
Leia(p2)
media := ((n1 * p1) + (n2 * p2))/ (p1 + p2)
Escreva(media)
FimAlgoritmo
```

4.

```
Algoritmo "Prog_7_4"
Var
salario, porcentagem: Real
Inicio
Leia(salario)
Leia(porcentagem)
salario := salario + salario * porcentagem / 100
Escreva(salario)
FimAlgoritmo
```

5.

```
Algoritmo "Prog_7_5"
Var
salario, imposto: Real
dependentes: Inteiro
Inicio
Leia(salario)
Leia(dependentes)
salario := salario + dependentes * 32
imposto := salario * 0.275
salario := salario - imposto
Escreva(salario)
FimAlgoritmo
```

6.

```
Algoritmo "Prog_7_6"
Var
valorAntigo, valorNovo, diferenca: Real
Inicio
Leia(valorAntigo)
Leia(valorNovo)
diferenca := (valorNovo - valorAntigo) / valorAntigo
diferenca := diferenca * 100
Escreva(diferenca, "%")
FimAlgoritmo
```

7.

```
Algoritmo "Prog_7_7"
Var
raio, diametro, perimetro, area, volume: Real
Inicio
Leia(raio)
diametro := 2 * raio
perimetro := 2 * 3.1415 * raio
area :=  3.1415 * raio * raio
volume := 4 / 3 * 3.1415 * raio * raio *raio
Escreval(diametro)
Escreval(perimetro)
Escreval(area)

Escreval(volume)
FimAlgoritmo
```

8.

```
Algoritmo "Prog_7_8"
Var
valocidadeKM, velocidadeMS: Real
Inicio
Leia(valocidadeKM)
velocidadeMS := valocidadeKM * 1000 / (60 * 60)
Escreval(velocidadeMS)
FimAlgoritmo
```

Aula 8

1.

```
Algoritmo "Prog_8_1"
Var
n1, n2: inteiro
Inicio
Leia(n1)
Leia(n2)
  Se (n1 > n2) Entao
Escreva("O primeiro número é maior")
Senao
Escreva("O segundo número é maior")
FimSe
FimAlgoritmo
```

2.

```
Algoritmo "Prog_8_2"
Var
  n: inteiro
Inicio
  Leia(n)
  Se (n mod 3 = 0) Entao
Escreva("É divisível por 3")
Senao
Escreva("Não é divisível por 3")
FimSe
FimAlgoritmo
```

3.

```
Algoritmo "Prog_8_3"
Var
lado1, lado2, lado3: inteiro
Inicio
Leia(lado1)
Leia(lado2)
Leia(lado3)
  Se ((lado1 + lado2 > lado3) E (lado2 + lado3 > lado1) E
(lado1 + lado3 > lado2)) Entao
Escreva("É possível criar um triângulo")
Senao
Escreva("Não é possível criar um triângulo")
FimSe
FimAlgoritmo
```

4.

```
Algoritmo "Prog_8_1"
Var
  n: inteiro
Inicio
   Leia(n)
   Se (n % 3 = 0) E (n % 4 = 0) E (n % 5 <> 0) Entao
Escreva("É divisível por 3 e 4 e não é por 5")
Senao
Escreva("Não obedece as regras")
FimSe
FimAlgoritmo
```

5.

```
Algoritmo "Prog_8_5"
Var
n1, n2, n3, aux: inteiro
Inicio
Leia(n1)
Leia(n2)
Leia(n3)
  Se (n1 > n2) Entao
aux := n1
n1 := n2
n2 := aux
FimSe
  Se (n2 > n3) Entao
aux := n2
n2 := n3
n3 := aux
FimSe
  Se (n1 > n2) Entao
aux := n1
n1 := n2
n2 := aux
FimSe
Escreva(n1, n2, n3)
FimAlgoritmo
```

6.

```
Algoritmo "Prog_8_6"
Var
salarioBruto, irpf, inss, salarioLiquido: Real
Inicio
Leia(salarioBruto)
irpf := 0
inss := salarioBruto * 0.11
  Se (inss> 400) Entao
inss := 400
FimSe
  Se (salarioBruto>= 3000) Entao
irpf := salarioBruto * 0.275
Senao
   Se (salarioBruto>= 1500) Entao
irpf := salarioBruto * 0.15
FimSe
FimSe
salarioLiquido := salarioBruto - irpf - inss;
Escreval(irpf)
Escreval(inss)
Escreval(salarioLiquido)
FimAlgoritmo
```

Aula 9

1.

```
Algoritmo "Prog_9_1"
Var
numero: Inteiro
Inicio
Leia(numero)
Escolha numero
   Caso 1
Escreva("Janeiro")
   Caso 2
Escreva("Fevereiro")
   Caso 3
Escreva("Março")
   Caso 4
Escreva("Abril")
   Caso 5
Escreva("Maio")
   Caso 6
Escreva("Junho")
   Caso 7
```

```
Escreva("Julho")
   Caso 8
Escreva("Agosto")
   Caso 9
Escreva("Setembro")
   Caso 10
Escreva("Outubro")
   Caso 11
Escreva("Novembro")
   Caso 12
Escreva("Dezembro")
OutroCaso
Escreva("Mês Inválido")
FimEscolha
FimAlgoritmo
```

2.

```
Algoritmo "Prog_9_2"
Var
uf: caractere
Inicio
Leia(uf)
  Escolha uf
   Caso "RO"
Escreva("Porto Velho")
   Caso "AC"
Escreva("Rio Branco")
   Caso "AM"
Escreva("Manaus")
   Caso "AP"
Escreva("Amapá")
OutroCaso
Escreva("UF Inexistente")
FimEscolha
FimAlgoritmo
```

3.

```
Algoritmo "Prog_9_3"
Var
posicao: Inteiro
Inicio
Leia(posicao)
  Escolha posicao
   Caso 1
Escreva("Parabéns, você é o melhor.")
```

```
   Caso 2
Escreva("Parabéns, você conseguiu.")
   Caso 3
Escreva("Parabéns, você quase chegou lá.")
OutroCaso
Escreva("Parabéns, o importante é competir.")
FimEscolha
FimAlgoritmo
```

4.

```
Algoritmo "Prog_9_4"
Var
n1, n2, resultado: Real
opcao: Caractere
Inicio
Leia(n1)
Leia(n2)
Leia(opcao)
  Escolha opcao
   Caso "+"
resultado := n1 + n2
   Caso "*"
resultado := n1 * n2
   Caso "-"
resultado := n1 - n2
   Caso "/"
resultado := n1 - n2

FimEscolha
Escreva(resultado)
FimAlgoritmo
```

Aula 10

1.

```
Algoritmo "Prog_10_1"
Var
valor, rendimento: Real
  i, meses: inteiro
Inicio
Leia(valor)
Leia(rendimento)
Leia(meses)
  Para i de 1 ate meses faca
valor := valor + valor * rendimento / 100
FimPara
Escreva(valor)
FimAlgoritmo
```

2.

```
Algoritmo "Prog_10_2"
Var
  i, n, soma: inteiro
Inicio
  Leia(n)
soma := 0
  Para i de 1 ate n faca
soma := soma + i
FimPara
Escreva(soma)
FimAlgoritmo
```

3.

```
Algoritmo "Prog_10_3"
Var
  i, n, fatorial: inteiro
Inicio
  Leia(n)
fatorial := 1
  Para i de n ate 2 passo -1 faca
fatorial := fatorial * i
FimPara
Escreva(fatorial)
FimAlgoritmo
```

4.

```
Algoritmo "Prog_10_4"
Var
  i, a, b, potencia: inteiro
Inicio
Leia(a)
  Leia(b)
potencia := 1
  Para i de 1 ate b faca
potencia := potencia * a
FimPara
Escreva(potencia)
FimAlgoritmo
```

5.

```
Algoritmo "Prog_10_5"
Var
  n, fatorial, i, j: inteiro
neper: real
Inicio
  Leia(n)
neper := 1
  Para i de 1 ate n faca
fatorial := 1
    Para j de 1 ate i faca
fatorial := fatorial * i
FimPara
neper := neper + (1 / fatorial)
FimPara
Escreva(neper)
FimAlgoritmo
```

Referências

ASCENCIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Veneruchida. **Fundamentos de programação de computadores:** algoritmos, Pascal e C/C++. 2. ed. São Paulo: Pearson Prentice Hall, 2005.

FORBELLONE, André Luiz Villar**Lógica de programação:** a construção de algoritmos e estrutura de dados. 3. ed. São Paulo: Brochura, 2005.

PUGA, Sandra; RISSETTI, Gerson.**Lógica de programação e estruturas de dados:** com aplicações em Java. 2. ed. São Paulo: Pearson Prentice Hall, 2009.

MANZANO, José Augusto N. G; OLIVEIRA, Jayr Figueiredo de**Algoritmos:** lógica para desenvolvimento de programação de computadores. 22. ed. São Paulo: Erica, 2009.

SEBESTA, Robert W.**Conceitos de linguagem de programação.** 9. ed. Porto Alegre: Bookman, 2011.

INFOPEDIA. Enciclopédias e Dicionários Porto Editora.**Dicionário de Língua Portuguesa da Porto Editora – com acordo ortográfico.** Disponível em <.http://www.infopedia.pt/lingua-portuguesa/lógica> Acesso em: 03 set. 2013.

Obras Consultadas

ARAÚJO, Everton Coimbra.**Algoritmos:** fundamento e prática. 3. ed. São Paulo:Visual Books, 2007.

CARBONI, Irenice de Fátima.**Lógica de programação.** São Paulo: Thomson, 2003.

GOODRICH, Michael T; TAMASSIA, Roberto**Estruturas de dados e algoritmos em Java.** 4. ed. Porto Alegre: Bookman, 2007.

LAFORE, R. **Estruturas de dados e algoritmos em Java.** [S.l.]: Ciência Moderna, 2005.

LOPES, Anita; GARCIA, Guto.**Introdução à programação:** 500 algoritmos resolvidos. Rio de Janeiro: Campus, 2002.

SOARES, Márcio Vieira; GOMES, Marcelo Marques; SOUZA, Marco Antôni**Algoritmos e Lógica de Programação**. 2. ed. São Paulo: Cengage Learning, 2011.

SZWARCFITER, J. L.; MARKENZON, L**Estruturas de dados e seus algoritmos.** 3. ed. [S.l.]: LTC, 2010.

Bibliografia Básica

CORMEN, Thomas H. et al. **Algoritmos:** teoria e prática. 3. ed. São Paulo: Érica, 2012.

FARRELL, Joyce. **Lógica e design de programação.** São Paulo: Cengage Learning, 2009.

Currículo dos Professores-autores



Liluyoud Cury de Lacerda

Mestre em Ciências da Computação pela Universidade Federal de Santa Catarina – UFSC.

Bacharel em Ciências da Computação pela Universidade Federal de São Carlos – UFSCar.

Coordenador do curso de Sistemas de Informação da Faculdade de Ciências Administrativas e Tecnologia – FATEC-RO.

Coordenador do curso de Sistemas para Internet da Faculdade de Ciências Administrativas e Tecnologia – FATEC-RO.

Coordenador do curso de Pós-Graduação em Desenvolvimento Web da Faculdade de Ciências Administrativas e Tecnologia – FATEC-RO.

Professor titular responsável pelas disciplinas de Algoritmos, Padrões de Projeto e Inteligência Artificial da Faculdade de Ciências Administrativas e Tecnologia – FATEC-RO.

Professor titular responsável pelas disciplinas de Programação Orientada a Objetos e Projeto Integrador II da Faculdade de São Mateus – FATESM-RO.

Analista Programador do Ministério Público do Estado de Rondônia - MP/RO.



José Marcio Benite Ramos

Mestre em Ciências da Computação pela Universidade Federal de Santa Catarina - UFSC.

Bacharel em Ciências da Computação pela Universidade Federal de São Carlos - UFSCar.

Professor titular responsável pelas disciplinas de Estrutura de Dados, Programação Web e Siste-

mas Comerciais da Faculdade de Ciências Administrativas e Tecnologia – FATEC-RO.

Professor titular responsável pela disciplina de Programação Web da Faculdade de São Mateus – FATESM-RO.

Analista de sistemas e diretor da divisão esc. de Porto Velho do Serviço Federal de Processamento de Dados – SERPRO.



Sara Luize Oliveira Duarte

Mestre em Gestão e Desenvolvimento Regional pela Universidade de Taubaté (UNITAU).

Graduada em Processamento de Dados e pós-graduada em Desenvolvimento Web e Metodologia do Ensino Superior, ambas pela FATEC-RO.

Professora titular responsável pela disciplina Informática Básica para Trabalhos Acadêmicos, ofertada na modalidade semipresencial, na Faculdade de Tecnologia São Mateus – FATESM-RO.

Supervisora de Tecnologia do Laboratório de Educação a Distância - LED e Supervisora das salas virtuais da modalidade semipresencial da FATESM.

Professora titular responsável pelas disciplinas TCC e Estágio Supervisionado I e II da Faculdade de Ciências Administrativas e Tecnologia – FATEC-RO.

Professora conteudista de alguns guias de estudo, como Informática Básica para Trabalhos Acadêmicos, Metodologia da Pesquisa Científica, Como Estudar na EAD, Informática Básica, entre outros.

