Coleção Desenvolvimento de Software na Prática

Linha 1 — Fundamentos de Programação

Lógica de Programação para Iniciantes

Algoritmos, raciocínio computacional e primeiros programas na prática

Apostila 1

```
INÍCIO
  LEIA X
  SE X >= 10 ENTÃO
     ESCREVA "MAIOR OU IGUAL"
  SENÃO
     ESCREVA "MENOR"
  FIMSE
FIM
```

ALGORITMOS
LÓGICA
PROGRAMAÇÃO

Prof. Douglas Ribeiro

R484l

RIBEIRO, Douglas.

Lógica de programação para iniciantes: algoritmos, raciocínio computacional e primeiros programas na prática / Douglas Ribeiro. - 1. ed. – Santa Ernestina, SP: Edição do autor, 2026.

114 p. : il. ; recurso digital. - (Coleção Desenvolvimento de Software na Prática ; v. 1. Linha 1 - Fundamentos de Programação)

Apostila digital de distribuição gratuita. Inclui referências, apêndices, exercícios, projeto final e recursos complementares de estudo.

ISBN: 978-65-02-18300-7

1. Lógica de programação. 2. Algoritmos. 3. Fluxogramas. 4. Pseudocódigo. 5. Portugol. 6. Pensamento computacional. 7. Programação de computadores - Estudo e ensino. I. Título. II. Série.

CDD 005.1
CDU 004.42

Prefácio

Aprender programação costuma parecer assustador para quem nunca teve contato com computadores como ferramenta de criação. Muitas pessoas imaginam que programar é decorar códigos, conhecer palavras difíceis ou dominar uma linguagem antes mesmo de entender o problema. Esta apostila parte de outra ideia: antes de escrever programas, o estudante precisa aprender a pensar de forma organizada.

A lógica de programação é uma ponte entre o problema do mundo real e a solução que poderá ser executada por um computador. Essa ponte não nasce pronta. Ela é construída com perguntas, tentativas, erros, correções, exemplos simples e muita prática orientada. Por isso, este material foi escrito com calma, de forma progressiva, buscando acolher estudantes que estão começando do zero ou migrando de outras áreas para a computação.

A apostila acompanha uma pequena narrativa fictícia. Lia, Caio, Nina e a professora Helena aparecem ao longo dos capítulos para representar dúvidas, descobertas e caminhos possíveis. A história não substitui a teoria; ela serve para aproximar o conteúdo da realidade do estudante. Ao observar os personagens enfrentando problemas, o leitor pode perceber que aprender programação é um processo humano, gradual e possível.

Esta versão também foi ampliada para incluir mais explicações, exemplos lógicos, pseudocódigos, testes de mesa, exercícios e soluções comentadas. A intenção é que o material sirva tanto para aulas presenciais e remotas quanto para estudo individual. Não se trata de um resumo, mas de uma referência introdutória para quem deseja construir uma base sólida em lógica de programação.

Apresentação da apostila

Esta apostila é o primeiro volume da coleção Desenvolvimento de Software na Prática. Seu objetivo é introduzir o estudante aos fundamentos da lógica de programação por meio de problemas simples, explicações detalhadas, exemplos comentados e atividades progressivas.

O conteúdo foi organizado para quem ainda não sabe programar. Por isso, conceitos aparentemente simples, como variável, entrada de dados, decisão, repetição e teste de mesa, são explicados com cuidado. A apostila também apresenta fluxogramas, linguagem natural e pseudocódigo, mostrando que existem diferentes formas de representar uma solução antes de implementá-la em uma linguagem de programação.

Ao longo da obra, a turma fictícia orientada pela professora Helena desenvolverá partes de um Sistema de Controle Acadêmico Simplificado. O projeto aparece aos poucos, para que o estudante perceba como cada conceito contribui para resolver problemas maiores. Primeiro surgem ideias simples, como calcular uma média. Depois aparecem decisões, repetições, vetores, matrizes, funções e testes.

A proposta é que o estudante leia, acompanhe a história, pratique os exemplos, resolva os exercícios e compare suas respostas com o apêndice de soluções comentadas. O erro faz parte do processo. O mais importante é aprender a raciocinar, revisar e melhorar a própria solução.

Personagens da jornada

Personagem	Papel na apostila
Professora Helena	Conduz as explicações, organiza as soluções e mostra que programar começa com clareza de pensamento.
Lia	Estudante curiosa, cuidadosa e organizada. Costuma perceber detalhes importantes do problema.
Caio	Estudante participativo e impulsivo. Representa dúvidas comuns, erros de pressa e soluções incompletas.
Nina	Profissional de outra área que está migrando para tecnologia. Mostra que ninguém precisa ter começado cedo para aprender computação.
Turma	Representa o grupo de estudantes que testa ideias, propõe caminhos e aprende coletivamente.

Orientações para estudar esta apostila

A melhor forma de estudar lógica de programação é combinar leitura e prática. Leia cada capítulo com calma, copie os pseudocódigos, altere pequenos detalhes e observe o resultado esperado. Quando encontrar um exercício, tente resolvê-lo antes de consultar o apêndice de soluções.

Não avance apenas porque entendeu a explicação. Em programação, entender um exemplo pronto é diferente de conseguir construir uma solução sozinho. Por isso, a apostila traz exercícios de fixação e desafios. Eles foram pensados para transformar leitura em aprendizagem ativa.

Sempre que aparecer um algoritmo, procure responder a quatro perguntas: qual problema ele resolve? Quais dados ele recebe? O que ele processa? Qual resultado ele apresenta? Essas perguntas simples ajudam a evitar soluções confusas.

Use o apêndice de soluções como apoio, não como atalho. Uma boa prática é tentar resolver, comparar com a solução proposta, identificar diferenças e reescrever o algoritmo com suas próprias palavras.

Orientações ao professor ou instrutor

Este material pode ser usado em disciplinas introdutórias de programação, cursos livres, formação técnica, graduação tecnológica ou nivelamento para estudantes de áreas não computacionais. A sequência foi pensada para aulas expositivas, práticas em laboratório e estudo autônomo.

Recomenda-se que cada capítulo seja trabalhado com leitura orientada, discussão da situação-problema, construção coletiva do algoritmo e prática individual. A narrativa dos personagens pode ser usada como ponto de partida para perguntas em sala: qual personagem percebeu melhor o problema? Qual erro Caio cometeu? Que alternativa Nina propôs?

Os exemplos foram escritos em pseudocódigo para evitar dependência de uma linguagem específica. O professor pode adaptar os algoritmos para Portugol, C, Python, JavaScript ou outra linguagem conforme o curso. O foco, entretanto, deve permanecer no raciocínio lógico.

Sumário

APRESENTAÇÃO DA APOSTILA 4
ORIENTAÇÕES PARA ESTUDAR ESTA APOSTILA 6
ORIENTAÇÕES AO PROFESSOR OU INSTRUTOR 7
PENSAR ANTES DE PROGRAMAR 9
ALGORITMOS EM LINGUAGEM NATURAL 17
FLUXOGRAMAS DO ZERO 24
PSEUDOCÓDIGO E PORTUGOL 33
DADOS, VARIÁVEIS E TIPOS 40
OPERADORES E EXPRESSÕES 47
ENTRADA, PROCESSAMENTO E SAÍDA 54
ESTRUTURAS CONDICIONAIS 60
ESTRUTURAS DE REPETIÇÃO 67
VETORES E MATRIZES 74
MODULARIZAÇÃO 80
TESTES, ERROS E BOAS PRÁTICAS 87
PROJETO FINAL ORIENTADO 93
APÊNDICE A - GUIA RÁPIDO DE PSEUDOCÓDIGO 101
APÊNDICE B - MODELOS VISUAIS DE FLUXOGRAMAS 102
APÊNDICE C - SOLUÇÕES COMENTADAS DOS EXERCÍCIOS E DESAFIOS 105

CAPÍTULO 1

Pensar antes de programar

lógica, problemas e pensamento computacional

Identificação do capítulo

Item	Descrição
Tema	Pensar antes de programar
Objetivo geral	Compreender que programação começa pela análise do problema e pela organização do raciocínio.
Palavras-chave	lógica; problema; algoritmo; pensamento computacional; decomposição
Projeto em construção	Sistema de Controle Acadêmico Simplificado

Objetivos de aprendizagem

Ao final deste capítulo, espera-se que o estudante seja capaz de:

- explicar, com suas palavras, o que é lógica de programação;
- diferenciar problema, solução e instrução;
- identificar entradas, processamentos e saídas em situações simples;
- reconhecer a importância de decompor problemas antes de escrever pseudocódigo;
- perceber que errar, testar e corrigir fazem parte do processo de aprendizagem.

Introdução

Na primeira aula, Caio entrou na sala dizendo que estava preocupado. Ele havia pesquisado vídeos sobre programação e encontrou telas cheias de símbolos, comandos e mensagens em inglês. Lia, mais silenciosa, comentou que também estava insegura, mas por outro motivo: ela gostava de resolver problemas, porém não sabia se isso era suficiente para começar. Nina, que vinha da área administrativa, fez uma pergunta direta: “professora, eu preciso saber matemática avançada para aprender a programar?”

A professora Helena sorriu e respondeu que a primeira habilidade de um programador não é decorar comandos. A primeira habilidade é organizar o pensamento. Antes de escrever um programa, é preciso entender o problema, identificar o que se sabe, perceber o que se deseja obter e descrever um caminho lógico para chegar ao resultado.

Para mostrar isso, ela propôs uma situação simples: a secretaria de uma escola precisava saber se um aluno estava aprovado, em recuperação ou reprovado com base em sua média. A turma ainda não conhecia variáveis, operadores ou estruturas condicionais, mas já podia começar a pensar: quais dados entram? Que cálculo precisa ser feito? Que regra define cada situação? Que resposta deve aparecer ao final?

BOA PRÁTICA
Quando receber um problema, não comece perguntando “qual comando eu uso?”. Comece perguntando “o que está acontecendo?” e “qual decisão precisa ser tomada?”.

O que é lógica de programação?

Lógica é a organização coerente de ideias. No cotidiano, usamos lógica quando seguimos uma receita, planejamos uma viagem, calculamos se o dinheiro é suficiente para uma compra ou decidimos qual caminho seguir para chegar a um lugar. Programação usa essa

mesma capacidade, mas exige que as instruções sejam escritas de forma precisa, porque o computador não interpreta intenções vagas.

Lógica de programação é o estudo das formas de construir soluções que possam ser executadas por um computador. Ela envolve analisar problemas, criar sequências de passos, tomar decisões, repetir ações, organizar dados e testar resultados. A linguagem de programação vem depois. Antes de escrever em C, JavaScript, Python, Portugol ou qualquer outra linguagem, o estudante precisa conseguir explicar a solução.

Um erro comum de iniciantes é acreditar que o computador "entende" o que queremos. Na verdade, o computador executa instruções. Se as instruções forem incompletas, ambíguas ou contraditórias, o resultado será incorreto. Por isso, programar exige clareza.

Problema, dados e solução

Todo algoritmo nasce de um problema. Um problema, em programação, é uma situação que possui uma necessidade de resposta. Essa resposta pode ser um cálculo, uma classificação, uma busca, uma organização de dados ou uma decisão.

Ao analisar um problema, procure separar três partes: entrada, processamento e saída. A entrada representa os dados necessários. O processamento representa as operações realizadas com esses dados. A saída representa o resultado esperado.

No problema da média do aluno, as entradas são as notas. O processamento é o cálculo da média. A saída é a média calculada e a situação do aluno. Mesmo sem código, a turma já consegue desenhar a lógica geral.

Parte	Pergunta orientadora	Exemplo no problema da média
Entrada	Quais informações preciso receber?	Nota 1 e Nota 2 do aluno.
Processamento	O que preciso fazer com essas informações?	Somar as notas e dividir por 2.
Saída	Qual resultado deve ser apresentado?	Média final e situação do aluno.

Pensamento computacional

Pensamento computacional é uma forma de resolver problemas usando estratégias que também aparecem na computação. Ele não pertence apenas aos programadores. Um professor que organiza uma aula, um enfermeiro que segue um protocolo, um administrador que monta uma planilha ou um comerciante que calcula descontos também usam formas de decompor, organizar e automatizar raciocínios.

Quatro ideias aparecem com frequência: decomposição, reconhecimento de padrões, abstração e criação de algoritmos. Decompor é dividir um problema grande em partes menores. Reconhecer padrões é perceber semelhanças entre problemas. Abstrair é focar no que importa e deixar de lado detalhes que não influenciam a solução. Criar algoritmo é descrever uma sequência de passos para resolver o problema.

Ideia	Explicação simples	Exemplo
Decomposição	Dividir para entender melhor.	Separar cadastro, cálculo de média e relatório.
Padrões	Perceber situações parecidas.	Todo aluno precisa de nome, notas e média.
Abstração	Focar no essencial.	Para calcular média, a cor da camiseta do aluno não importa.
Algoritmo	Descrever passos ordenados.	Ler notas, calcular média e mostrar situação.

Exemplo lógico comentado: rotina de aprovação

A professora Helena pediu que cada estudante propusesse, em linguagem natural, uma solução para classificar a situação de um aluno. Caio respondeu rapidamente: “é só ver se ele passou”. A professora explicou que a frase estava correta como ideia, mas ainda era vaga. O computador precisaria de critérios.

Lia sugeriu a primeira versão: receber duas notas, calcular a média e comparar com a regra da escola. Nina acrescentou que a resposta deveria ser exibida de modo claro para a secretaria. A professora então organizou a solução:

```
Algoritmo em linguagem natural:
1. Receber o nome do aluno.
2. Receber a primeira nota.
3. Receber a segunda nota.
4. Calcular a média das duas notas.
5. Se a média for maior ou igual a 7, informar "Aprovado".
6. Se a média for menor que 7 e maior ou igual a 5, informar
"Recuperação".
7. Se a média for menor que 5, informar "Reprovado".
8. Mostrar o nome, a média e a situação do aluno.
```

Ao final, a professora observou que Lia chegou perto da solução completa, porque separou os dados e o cálculo. Caio percebeu o objetivo geral, mas ainda não havia definido regras. Nina contribuiu lembrando que a saída precisa ser compreensível para quem usa o sistema. A turma aprendeu que uma boa solução nasce da combinação entre objetivo, dados, regras e apresentação do resultado.

Prática orientada: identificar entrada, processamento e saída

Antes de programar, pratique a análise. Imagine que uma lanchonete deseja calcular o valor final de uma compra. O cliente informa a quantidade de lanches e o preço unitário. O sistema deve calcular o total e mostrar o valor a pagar.

A entrada é a quantidade e o preço unitário. O processamento é a multiplicação da quantidade pelo preço. A saída é o valor total. Se houver

desconto, imposto ou taxa de entrega, essas regras entram no processamento.

```
Análise do problema:
Entrada: quantidade, preco_unitario
Processamento: total = quantidade * preco_unitario
Saída: total a pagar
```

Síntese ampliada do capítulo

Neste capítulo, o ponto mais importante foi compreender que programação começa antes do código. O estudante iniciante muitas vezes quer abrir um editor e digitar comandos, mas a qualidade do programa depende da qualidade do raciocínio anterior. Uma solução confusa em linguagem natural dificilmente se tornará um bom programa.

Também vimos que todo problema pode ser analisado por meio de entrada, processamento e saída. Essa divisão é simples, mas muito poderosa. Quando um algoritmo não funciona, muitas vezes o erro está em uma dessas três partes: dados insuficientes, processamento incorreto ou saída mal definida.

A partir daqui, os exercícios vão treinar exatamente essa capacidade de análise. Não tenha pressa. Em cada questão, escreva primeiro o que entra, depois o que será feito e, por fim, o que deve aparecer como resultado. Esse hábito acompanhará você em toda a programação.

PENSE NISSO
A síntese não é apenas uma repetição. Ela deve ajudar você a perceber se entendeu o caminho do raciocínio. Antes de ir aos exercícios, volte mentalmente ao problema inicial e tente explicar a solução com suas próprias palavras.

Exercícios de fixação

1. Explique, com suas palavras, a diferença entre problema e algoritmo.
2. Identifique entrada, processamento e saída em um sistema que calcula o troco de uma compra.

3. Escreva uma sequência lógica para preparar um café simples.
4. Por que a frase “ver se o aluno passou” ainda não é suficiente para um computador?
5. Crie uma análise de entrada, processamento e saída para calcular o valor de uma corrida de aplicativo.

Desafio ou atividade prática

Escolha uma situação real da sua rotina e descreva a solução em linguagem natural, separando entrada, processamento e saída. Depois, explique qual parte poderia gerar erro se fosse mal definida.

Referências de apoio do capítulo

WING, Jeannette M. Computational thinking. Communications of the ACM, New York, v. 49, n. 3, p. 33-35, mar. 2006. DOI: 10.1145/1118178.1118215.

PÓLYA, George. A arte de resolver problemas: um novo aspecto do método matemático. Tradução e adaptação de Heitor Lisboa de Araújo. Rio de Janeiro: Interciência, 1995. ISBN 978-85-7193-136-7.

CAPÍTULO 2

Algoritmos em linguagem natural

a primeira forma de organizar uma solução

Identificação do capítulo

Item	Descrição
Tema	Algoritmos em linguagem natural
Objetivo geral	Construir algoritmos descritivos antes do pseudocódigo, usando passos claros e ordenados.
Palavras-chave	algoritmo; sequência; clareza; ambiguidade; refinamento
Projeto em construção	Sistema de Controle Acadêmico Simplificado

Objetivos de aprendizagem

Ao final deste capítulo, espera-se que o estudante seja capaz de:

- definir algoritmo e reconhecer suas características;
- escrever soluções em linguagem natural com passos claros;
- identificar ambiguidades em instruções do cotidiano;
- refinar uma solução genérica até torná-la executável;
- preparar o raciocínio para a escrita em pseudocódigo.

Introdução

No segundo encontro, a professora Helena entrou na sala com uma folha em branco e escreveu no quadro: “fazer matrícula de aluno”. Em seguida, perguntou à turma: isso é um problema de programação? Caio respondeu que não, porque ninguém havia falado em computador. Lia discordou: se a matrícula segue regras, etapas e decisões, talvez possa virar um algoritmo.

A professora explicou que Lia estava certa. Um algoritmo não nasce necessariamente dentro de um computador. Um algoritmo é uma sequência finita e ordenada de passos para resolver um problema. Uma receita culinária, um manual de instalação e um roteiro de atendimento são exemplos de algoritmos em linguagem natural.

A diferença é que, em programação, precisamos reduzir ambiguidades. Instruções como “faça rápido”, “organize direito” ou “calcule quando der” podem fazer sentido para pessoas em uma conversa, mas não servem para um computador. O algoritmo precisa dizer o que fazer, em que ordem, com quais dados e em quais condições.

Características de um bom algoritmo

Um bom algoritmo deve ser finito, claro, ordenado e eficaz. Ser finito significa que ele precisa terminar. Ser claro significa que suas instruções não devem gerar dúvida. Ser ordenado significa que os passos precisam aparecer em uma sequência lógica. Ser eficaz significa que, se executado corretamente, deve resolver o problema proposto.

Imagine uma instrução como “adicione açúcar até ficar bom”. Para uma pessoa, isso depende do gosto. Para um algoritmo, essa instrução é vaga. Melhor seria dizer “adicione duas colheres de açúcar”. Em programação, quanto mais objetiva for a instrução, menor a chance de erro.

Característica	Significado	Problema quando falta
Finito	Deve ter começo e fim.	O algoritmo pode entrar em repetição infinita.
Claro	As instruções não devem ser ambíguas.	Duas pessoas podem executar de formas

Característica	Significado	Problema quando falta
		diferentes.
Ordenado	Os passos seguem uma sequência lógica.	O resultado pode ser incoerente.
Eficaz	Resolve o problema proposto.	Mesmo executado, não entrega o resultado desejado.

Da ideia geral ao passo detalhado

A professora Helena pediu que a turma criasse um algoritmo para matricular um aluno em um curso. A primeira tentativa de Caio foi: "pegar os dados e matricular". A ideia estava correta, mas faltavam detalhes. Que dados? Em qual curso? Existe vaga? O aluno já está cadastrado? A matrícula deve gerar comprovante?

A partir das perguntas, a turma percebeu que uma frase geral precisa ser refinada. Refinar significa transformar uma instrução ampla em passos menores e mais precisos.

```
Primeira versão, ainda vaga:
1. Pegar os dados do aluno.
2. Matricular o aluno.
3. Entregar comprovante.

Versão refinada:
1. Receber nome do aluno.
2. Receber CPF do aluno.
3. Receber curso desejado.
4. Verificar se há vaga no curso.
5. Se houver vaga, registrar matrícula.
6. Se não houver vaga, informar que não foi possível
matricular.
7. Se a matrícula foi registrada, emitir comprovante.
```

Sequência, decisão e repetição já aparecem no cotidiano

Mesmo antes de estudar estruturas formais, já podemos perceber três ideias fundamentais: sequência, decisão e repetição. Sequência é executar passos em ordem. Decisão é escolher um caminho conforme

uma condição. Repetição é executar passos várias vezes até atingir um objetivo.

No exemplo da matrícula, há sequência quando os dados são recebidos em ordem. Há decisão quando o sistema verifica se existe vaga. Poderia haver repetição se a secretaria cadastrasse vários alunos em sequência.

ATENÇÃO
Quando você escreve um algoritmo em linguagem natural, já está treinando estruturas que depois aparecerão no pseudocódigo: sequência, decisão e repetição. A diferença é que, por enquanto, elas ainda aparecem em frases do português.

Exemplo comentado: controle de senha de atendimento

A professora apresentou outro problema: uma clínica atende pacientes por senha. O sistema deve entregar a próxima senha e mostrar o número no painel. A turma precisava escrever uma solução em linguagem natural.

```
Algoritmo em linguagem natural:
1. Iniciar o número da senha em 0.
2. Quando um paciente chegar, aumentar a senha em 1.
3. Entregar ao paciente a nova senha.
4. Quando o atendente chamar o próximo paciente, mostrar no
painel a senha chamada.
5. Repetir o processo enquanto houver pacientes aguardando.
```

Caio percebeu a repetição: enquanto houver pacientes, o processo continua. Nina percebeu que a senha precisa começar em algum valor. Lia observou que o sistema faz duas coisas diferentes: emitir senha e chamar senha. A professora destacou que essa separação será importante quando a turma aprender modularização.

Síntese ampliada do capítulo

Neste capítulo, vimos que algoritmos não são apenas códigos. Eles podem ser escritos em linguagem natural e usados para organizar o raciocínio antes da programação. Essa etapa é especialmente importante para iniciantes, porque permite focar na solução sem se preocupar com a sintaxe de uma linguagem.

Também vimos que instruções vagas precisam ser refinadas. Quando dizemos “matricular aluno”, escondemos várias etapas: receber dados, verificar vagas, registrar informações e emitir comprovante. Um bom programador aprende a revelar essas etapas.

Nos exercícios, pratique a transformação de situações simples em sequências claras. Você não precisa escrever código ainda. Precisa aprender a fazer boas perguntas sobre o problema.

PENSE NISSO
A síntese não é apenas uma repetição. Ela deve ajudar você a perceber se entendeu o caminho do raciocínio. Antes de ir aos exercícios, volte mentalmente ao problema inicial e tente explicar a solução com suas próprias palavras.

Exercícios de fixação

1. Escreva um algoritmo em linguagem natural para calcular o troco de uma compra.
2. Refine a instrução “organizar uma fila” em pelo menos seis passos.
3. Identifique uma decisão no processo de matrícula de um aluno.
4. Identifique uma repetição no atendimento de uma clínica.
5. Explique por que “preparar café até ficar bom” é uma instrução ambígua.

Desafio ou atividade prática

Crie um algoritmo em linguagem natural para uma biblioteca emprestar livros. Considere cadastro do usuário, disponibilidade do livro e emissão de comprovante.

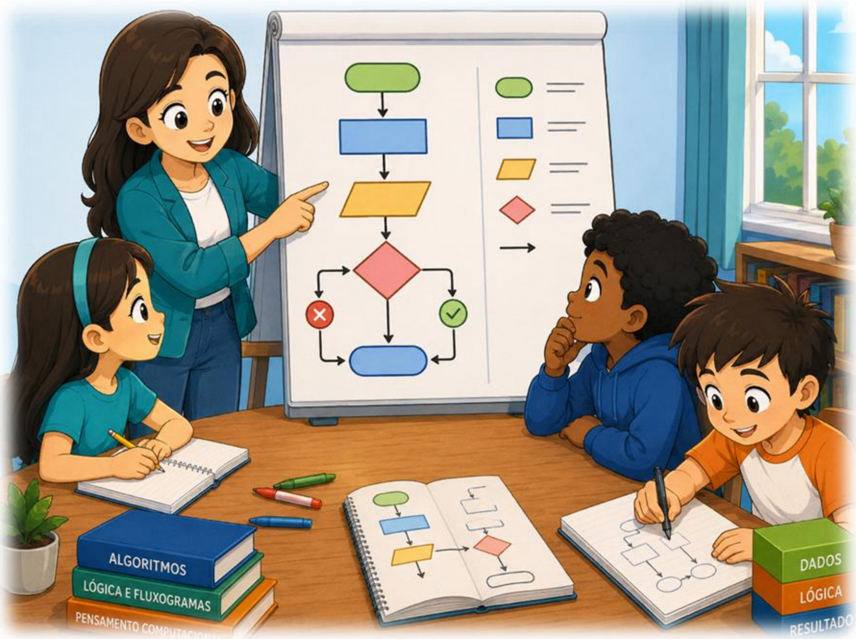
Referências de apoio do capítulo

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. Lógica de programação: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo: Pearson Prentice Hall, 2005. ISBN 978-85-7605-024-7.

CAPÍTULO 3

Fluxogramas do zero

representando visualmente a lógica



Identificação do capítulo

Item	Descrição
Tema	Fluxogramas do zero
Objetivo geral	Compreender o que é um fluxograma, quando utilizá-lo, quais símbolos são mais comuns e como ele ajuda a visualizar a lógica.
Palavras-chave	fluxograma; símbolo; decisão; processo; início; fim; visualização
Projeto em construção	Sistema de Controle Acadêmico Simplificado

Objetivos de aprendizagem

Ao final deste capítulo, espera-se que o estudante seja capaz de:

- explicar o que é um fluxograma e sua utilidade;
- reconhecer os principais símbolos usados em fluxogramas;
- interpretar um fluxograma simples;
- converter uma descrição em linguagem natural para fluxograma textual;
- comparar fluxograma, linguagem natural e pseudocódigo.

Introdução

Ao revisar o algoritmo da matrícula, Nina comentou que conseguia entender os passos, mas ainda se perdia quando havia decisões. “Se houver vaga, faço uma coisa; se não houver, faço outra. Quando leio em parágrafos, confundo os caminhos”, disse ela.

A professora Helena explicou que essa dificuldade é comum. Algumas pessoas entendem melhor uma solução quando a enxergam visualmente. Para isso, existe o fluxograma: uma representação gráfica de um algoritmo, usando símbolos padronizados conectados por setas.

Um fluxograma não é obrigatório em todo projeto, mas é muito útil quando queremos visualizar a sequência das ações, identificar decisões, explicar uma solução para outras pessoas ou planejar um algoritmo antes de escrever pseudocódigo.

O que é um fluxograma?

Fluxograma é um diagrama que representa o fluxo de execução de um processo ou algoritmo. A palavra “fluxo” indica movimento: uma etapa leva à próxima, uma decisão cria caminhos alternativos e as setas mostram a direção que a execução deve seguir.

Na lógica de programação, o fluxograma ajuda o estudante a enxergar a solução. Enquanto a linguagem natural usa frases e o pseudocódigo usa comandos, o fluxograma usa formas geométricas. Cada forma possui um significado: início, fim, entrada, saída, processamento ou decisão.

Ele pode ser usado em sala de aula, em documentação de sistemas, em reuniões com usuários, em análise de processos administrativos e em planejamento de programas. Para iniciantes, sua maior vantagem é tornar visível a estrutura do raciocínio.

Por que o fluxograma ajuda?

Quando um algoritmo possui muitas decisões, o texto pode ficar longo. O fluxograma permite ver os caminhos de forma mais clara. Se

uma condição for verdadeira, a seta vai para um lado; se for falsa, vai para outro. Essa visualização facilita a descoberta de erros.

Outra vantagem é a comunicação. Pessoas que ainda não conhecem pseudocódigo podem entender um fluxograma simples. Por isso, ele é útil quando precisamos explicar a lógica para colegas, professores, usuários ou equipes multidisciplinares.

No entanto, fluxogramas também têm limites. Para algoritmos muito grandes, o desenho pode ficar extenso e difícil de manter. Por isso, em projetos reais, fluxogramas costumam ser usados para explicar partes importantes do processo, não necessariamente cada detalhe do sistema.

Símbolos principais de um fluxograma

Fluxograma: símbolos principais

Representação gráfica usada para visualizar algoritmos e processos

Início / Fim
Indica onde o algoritmo começa e termina.

Processo
Representa um processamento ou cálculo.

Entrada / Saída
Usado para ler dados ou mostrar resultados.

Decisão
Representa uma pergunta com caminhos possíveis.

Seta
Mostra a direção do fluxo de execução.

Símbolo	Nome	Uso principal
Oval	Início/Fim	Indica onde o algoritmo começa e onde termina.
Paralelogramo	Entrada/Saída	Representa leitura de dados ou exibição de informações.
Retângulo	Processamento	Representa cálculo, atribuição ou ação executada.
Losango	Decisão	Representa uma pergunta com caminhos alternativos, geralmente Sim/Não.
Seta	Fluxo	Mostra a direção da execução.
Círculo pequeno	Conector	Liga partes do fluxograma quando o desenho fica grande.

ATENÇÃO
O nome dos símbolos pode variar um pouco conforme a ferramenta, mas a ideia central permanece: formas diferentes comunicam funções diferentes no algoritmo.

Exemplo visual: média do aluno

Na figura 2, observe um fluxograma completo representando a verificação da situação final do aluno com base em sua média.

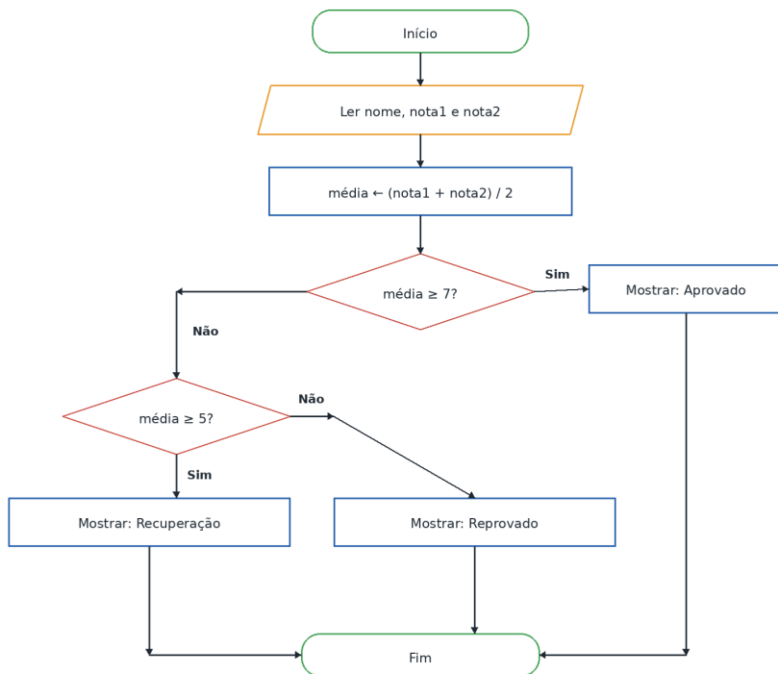


Figura 2 — Fluxograma de verificação da situação do aluno.

Observe que os colchetes representam ações de entrada, processamento ou saída, enquanto as chaves representam decisões. Em um fluxograma gráfico, “INÍCIO” e “FIM” estariam em ovais; leitura e exibição estariam em paralelogramos; cálculo estaria em retângulo; e as perguntas estariam em losangos.

O fluxograma deixa evidente que existem três situações possíveis: aprovado, recuperação e reprovado. Também mostra que a segunda pergunta só acontece se a primeira resposta for “Não”. Esse detalhe poderia passar despercebido em uma explicação apressada.

Comparando linguagem natural, fluxograma e pseudocódigo

Forma	Vantagem	Limitação
Linguagem natural	Fácil para começar e explicar ideias.	Pode ficar ambígua se os passos não forem precisos.
Fluxograma	Mostra visualmente sequência e decisões.	Pode ficar grande em problemas complexos.
Pseudocódigo	Aproxima o raciocínio da programação.	Exige aprender uma forma mais estruturada de escrever.

Exemplo comentado: caixa de supermercado

A professora propôs um segundo exemplo. Um caixa de supermercado precisa calcular o valor a pagar. Se o cliente tiver cupom, aplica 10% de desconto. Caso contrário, cobra o valor normal.

```
Fluxograma textual:
[INÍCIO]
   ↓
[LER valor_compra]
   ↓
[LER tem_cupom]
   ↓
{tem_cupom = "sim"?}
   ├── Sim → [total ← valor_compra * 0.90]
   └── Não → [total ← valor_compra]
   ↓
[MOSTRAR total]
   ↓
[FIM]
```

Caio inicialmente queria calcular o desconto sempre. A professora mostrou que isso geraria erro para clientes sem cupom. Lia percebeu que a decisão precisa acontecer antes do cálculo final. Nina comentou que o fluxograma deixou mais fácil enxergar os dois caminhos. A professora

concluiu que o fluxograma é especialmente útil quando existe uma pergunta que muda o rumo da execução.

Como desenhar um fluxograma na prática

- Comece pelo símbolo de início.
- Liste as entradas necessárias.
- Coloque os processamentos em ordem.
- Sempre que houver uma pergunta, use um símbolo de decisão.
- Identifique claramente os caminhos de saída da decisão, como Sim e Não.
- Garanta que todos os caminhos cheguem a um fim.
- Revise se não há setas sem destino ou decisões sem resposta.

Síntese ampliada do capítulo

Neste capítulo, aprendemos que fluxograma é uma representação visual de um algoritmo. Ele ajuda a enxergar a sequência dos passos, as decisões e os caminhos possíveis. Para quem está começando, isso é muito importante, porque a lógica deixa de ser apenas texto e passa a ser vista como fluxo.

Também vimos que cada símbolo possui função. O erro mais comum é desenhar tudo com o mesmo formato. Isso enfraquece o fluxograma, porque o leitor não consegue distinguir entrada, processamento e decisão. Outro erro comum é criar decisões sem indicar claramente os caminhos de saída.

Nos próximos exercícios, pratique desenhar ou escrever fluxogramas textuais antes do pseudocódigo. Esse treino ajudará você a perceber caminhos alternativos e evitar algoritmos incompletos.

PENSE NISSO
A síntese não é apenas uma repetição. Ela deve ajudar você a perceber se entendeu o caminho do raciocínio. Antes de ir aos exercícios, volte mentalmente ao problema inicial e tente explicar a solução com suas próprias palavras.

Exercícios de fixação

1. Explique o que é um fluxograma e cite uma situação em que ele pode ser usado.
2. Qual símbolo representa uma decisão? Que tipo de resposta normalmente sai dele?
3. Crie um fluxograma textual para verificar se uma pessoa é maior de idade.
4. Crie um fluxograma textual para calcular o total de uma compra com desconto apenas se o valor for maior que R$ 100,00.
5. Compare fluxograma e pseudocódigo: em que situação cada um pode ser mais útil?

Desafio ou atividade prática

Desenhe, em papel ou ferramenta digital, um fluxograma para o empréstimo de livro em uma biblioteca. Considere: usuário cadastrado, livro disponível, registro do empréstimo e mensagem final.

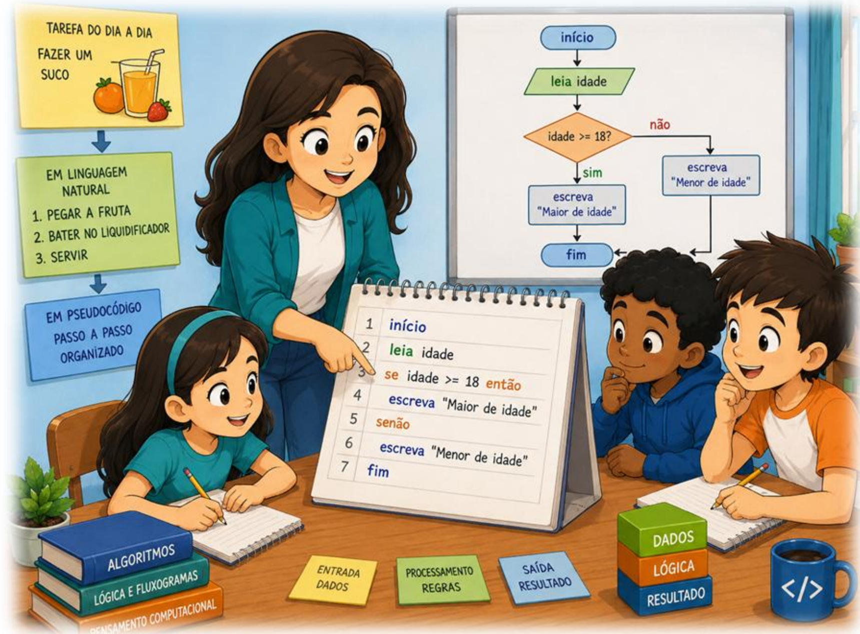
Referências de apoio do capítulo

FARRER, Harry et al. Programação estruturada de computadores: algoritmos estruturados. 3. ed. Rio de Janeiro: LTC, 1999. ISBN 978-85-216-1180-6.

CAPÍTULO 4

Pseudocódigo e Portugol

aproximando o raciocínio da programação



Identificação do capítulo

Item	Descrição
Tema	Pseudocódigo e Portugol
Objetivo geral	Escrever algoritmos em formato estruturado, usando comandos simples de leitura, escrita, atribuição e controle.
Palavras-chave	pseudocódigo; Portugol; algoritmo; leia; escreva; se; enquanto
Projeto em construção	Sistema de Controle Acadêmico Simplificado

Objetivos de aprendizagem

Ao final deste capítulo, espera-se que o estudante seja capaz de:

- entender a função do pseudocódigo no aprendizado de programação;
- reconhecer comandos básicos de entrada e saída;
- usar atribuição para guardar resultados intermediários;
- converter algoritmos em linguagem natural para pseudocódigo;
- interpretar pseudocódigos simples.

Introdução

Depois de trabalhar com linguagem natural e fluxogramas, a turma estava pronta para dar um passo em direção ao código. A professora Helena explicou que ainda não usariam uma linguagem profissional. Primeiro, usariam pseudocódigo: uma forma intermediária de escrever algoritmos com estrutura parecida com programação, mas em linguagem mais simples.

O pseudocódigo não pertence a uma linguagem única. Ele é uma convenção didática. Alguns professores usam “leia” e “escreva”; outros usam “entrada” e “saída”; alguns adotam Portugol. O importante é que o estudante compreenda a lógica e consiga transformá-la depois em uma linguagem real.

Nina gostou da ideia: “então o pseudocódigo é como um rascunho técnico?”. A professora respondeu que sim. É um rascunho mais preciso que a linguagem natural e menos rígido que uma linguagem de programação profissional.

Estrutura básica de um pseudocódigo

```
algoritmo "Nome do algoritmo"
var
   // declaração de variáveis
inicio
   // comandos executados pelo algoritmo
fim
```

Nem todo pseudocódigo precisa seguir exatamente esse formato, mas ele ajuda a organizar. A área “var” é usada para declarar variáveis, isto é, nomes que guardarão valores. A área entre “inicio” e “fim” contém os comandos que serão executados.

Comandos de entrada e saída

O comando de entrada permite que o algoritmo receba dados. Em muitos materiais, usamos “leia”. O comando de saída permite mostrar informações ao usuário. Usamos “escreva”.

Quando o algoritmo precisa perguntar o nome do aluno, usa entrada. Quando precisa mostrar a média final, usa saída. Essa diferença parece simples, mas é essencial: entrada vem de fora para dentro do algoritmo; saída vai do algoritmo para fora.

```
algoritmo "Saudacao"
var
   nome: caractere
inicio
   escreva("Digite seu nome: ")
   leia(nome)
   escreva("Olá, ", nome, "! Seja bem-vindo.")
fim
```

Atribuição

Atribuição é o ato de guardar um valor em uma variável. Usamos normalmente o símbolo de seta ou sinal de igual, dependendo do padrão adotado. Nesta apostila, usaremos a seta para deixar claro que o valor da direita é armazenado no nome da esquerda.

No comando media ← (nota1 + nota2) / 2, o algoritmo calcula a expressão e guarda o resultado na variável media. A variável passa a conter esse valor até que receba outro.

```
total ← quantidade * preco_unitario
media ← (nota1 + nota2) / 2
idade_futura ← idade_atual + 1
```

Exemplo comentado: cálculo de média

```
algoritmo "MediaAluno"
var
   nome: caractere
   nota1, nota2, media: real
inicio
   escreva("Nome do aluno: ")
   leia(nome)
```

```
   escreva("Primeira nota: ")
   leia(nota1)

   escreva("Segunda nota: ")
   leia(nota2)

   media ← (nota1 + nota2) / 2

   escreva("Aluno: ", nome)
   escreva("Média: ", media)
fim
```

A professora leu o algoritmo com a turma, linha por linha. Primeiro, o algoritmo declara quais informações serão usadas. Depois, solicita o nome e as notas. Em seguida, calcula a média. Por fim, mostra o resultado.

Caio perguntou por que era necessário declarar “media” se ela poderia ser calculada direto na saída. A professora explicou que guardar a média facilita usar o valor novamente em outras partes, por exemplo, para decidir se o aluno foi aprovado. Lia observou que a variável também torna o algoritmo mais legível.

Ao final, a professora destacou que ninguém precisa decorar todos os comandos no primeiro contato. O mais importante é compreender a função de cada parte: declarar, ler, calcular e escrever.

Erros comuns em pseudocódigo inicial

- Usar uma variável sem declará-la.
- Ler dados depois de tentar usá-los em um cálculo.
- Confundir texto com nome de variável.
- Esquecer de mostrar o resultado ao usuário.
- Fazer atribuição no sentido errado.

BOA PRÁTICA
Escreva mensagens claras antes de cada leitura. Em vez de apenas usar leia(nota1), mostre ao usuário o que ele deve digitar: escreva("Primeira nota: ").

Síntese ampliada do capítulo

Neste capítulo, o pseudocódigo apareceu como ponte entre o pensamento e o programa. Ele permite escrever soluções com mais precisão do que a linguagem natural, mas sem exigir ainda a sintaxe completa de uma linguagem real.

Vimos que um algoritmo em pseudocódigo costuma declarar variáveis, receber entradas, realizar processamentos e mostrar saídas. Essa organização será repetida muitas vezes ao longo da apostila. Quanto mais você praticar, mais natural ela se tornará.

Nos exercícios, procure escrever pseudocódigos completos. Não basta calcular mentalmente. Declare as variáveis, leia os dados, processe e mostre o resultado. Esse cuidado evita lacunas na solução.

PENSE NISSO
A síntese não é apenas uma repetição. Ela deve ajudar você a perceber se entendeu o caminho do raciocínio. Antes de ir aos exercícios, volte mentalmente ao problema inicial e tente explicar a solução com suas próprias palavras.

Exercícios de fixação

1. Escreva um pseudocódigo que leia o nome de uma pessoa e mostre uma mensagem de boas-vindas.
2. Escreva um pseudocódigo que leia o preço de um produto e mostre o valor com 10% de desconto.
3. Explique a diferença entre leia e escreva.
4. Explique o que significa a atribuição total ← quantidade * preco.
5. Reescreva o algoritmo de cálculo de média para usar três notas.

Desafio ou atividade prática

Crie um pseudocódigo para calcular o salário bruto de um funcionário com base nas horas trabalhadas e no valor da hora.

Referências de apoio do capítulo

ASCENCIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Veneruchi de. Fundamentos da programação de computadores: algoritmos, Pascal, C/C++ e Java. 3. ed. São Paulo: Pearson Education do Brasil, 2012. ISBN 978-85-64574-16-8.

CAPÍTULO 5

Dados, variáveis e tipos

guardando informações para resolver problemas

Identificação do capítulo

Item	Descrição
Tema	Dados, variáveis e tipos
Objetivo geral	Compreender como algoritmos armazenam e manipulam informações por meio de variáveis, constantes e tipos de dados.
Palavras-chave	dados; variável; constante; inteiro; real; caractere; lógico
Projeto em construção	Sistema de Controle Acadêmico Simplificado

Objetivos de aprendizagem

Ao final deste capítulo, espera-se que o estudante seja capaz de:

- definir variável e constante;
- reconhecer tipos de dados básicos;
- escolher nomes significativos para variáveis;
- identificar erros causados por tipos inadequados;
- usar variáveis em expressões simples.

Introdução

Na aula seguinte, a professora Helena colocou três caixas sobre a mesa. Em uma, escreveu “nome”; em outra, “idade”; em outra, “média”. Em seguida, perguntou: o que essas caixas têm a ver com programação? Lia respondeu que pareciam lugares para guardar informações.

A professora explicou que variáveis funcionam como espaços nomeados para armazenar dados durante a execução de um algoritmo. O valor pode mudar, mas o nome da variável ajuda o programador a saber o que está sendo guardado.

Caio perguntou se poderia chamar uma variável de “x”. A professora respondeu que poderia, mas nem sempre deveria. Em problemas pequenos, x pode funcionar. Em sistemas reais, nomes significativos ajudam muito na leitura e manutenção do algoritmo.

O que são dados?

Dados são valores que representam informações. Um nome, uma nota, uma idade, um preço, uma resposta “sim” ou “não” e uma data podem ser dados. O computador manipula dados, mas precisa saber como tratá-los.

Tipos de dados mais comuns

Tipo	Uso	Exemplos
inteiro	Números sem parte decimal.	0, 1, 18, -5
real	Números com parte decimal.	7.5, 10.0, 99.90
caractere/cadeia	Texto, nomes e mensagens.	"Lia", "Aprovado"
lógico	Valores de verdade.	verdadeiro, falso

Escolher o tipo correto evita erros. Uma idade normalmente pode ser inteira. Uma nota pode ser real. Um nome deve ser texto. Uma resposta como “o aluno está aprovado?” pode ser lógica: verdadeiro ou falso.

Quando o tipo é inadequado, o algoritmo pode fazer operações indevidas. Não faz sentido somar nomes como se fossem números. Também não faz sentido guardar uma média com casas decimais em uma variável inteira se a precisão for importante.

Variáveis e constantes

Variável é um espaço cujo valor pode mudar durante a execução. Constante é um valor que permanece fixo. Se um sistema usa uma taxa padrão de desconto de 10%, essa taxa pode ser representada como constante. Se o valor da compra muda para cada cliente, deve ser variável.

A distinção ajuda a proteger a lógica. Valores fixos importantes ficam mais claros quando nomeados como constantes. Isso evita “números mágicos”, isto é, números soltos no algoritmo sem explicação.

```
constante
   DESCONTO_PADRAO ← 0.10

var
   valor_compra, valor_desconto, total: real
```

Boas práticas para nomes de variáveis

- Use nomes que indiquem o conteúdo: media, idade, total_compra.
- Evite nomes genéricos como a, b, coisa, valor1 quando o problema for maior.
- Não use acentos ou espaços, para facilitar adaptação a linguagens reais.
- Mantenha um padrão: total_compra ou totalCompra, mas não misture sem necessidade.
- Prefira clareza a abreviações excessivas.

Exemplo comentado: cálculo de desconto

```
algoritmo "DescontoProduto"
constante
   TAXA_DESCONTO ← 0.10
var
   preco, desconto, preco_final: real
inicio
```

```
   escreva("Preço do produto: ")
   leia(preco)

   desconto ← preco * TAXA_DESCONTO
   preco_final ← preco - desconto

   escreva("Desconto: ", desconto)
   escreva("Preço final: ", preco_final)
fim
```

A professora explicou que TAXA_DESCONTO foi escrita como constante porque, naquele problema, o desconto padrão não muda. Já o preço, o valor do desconto e o preço final são variáveis, pois dependem do produto informado. Nina percebeu que o algoritmo ficou mais fácil de alterar: se o desconto mudar para 15%, basta trocar o valor da constante.

Teste de mesa introdutório

Teste de mesa é uma simulação manual do algoritmo. Ainda estudaremos o tema com mais profundidade, mas já podemos observar como os valores mudam.

Passo	preco	desconto	preco_final	Observação
Ler preço	100.00	-	-	Usuário informou o preço.
Calcular desconto	100.00	10.00	-	100 * 0.10.
Calcular final	100.00	10.00	90.00	100 - 10.
Mostrar	100.00	10.00	90.00	Saída apresentada.

Síntese ampliada do capítulo

Neste capítulo, aprendemos que algoritmos precisam guardar informações. Variáveis são nomes associados a valores que podem mudar, e constantes representam valores fixos importantes para a solução.

Também vimos que os tipos de dados orientam o tipo de operação possível. Números inteiros e reais podem participar de cálculos. Textos representam nomes e mensagens. Valores lógicos representam verdadeiro ou falso. Escolher mal o tipo pode gerar erros de lógica.

Nos exercícios, pratique a escolha de nomes claros. Um algoritmo não deve ser compreensível apenas para quem o escreveu. Ele deve poder ser lido por colegas, professores e, no futuro, por você mesmo.

PENSE NISSO
A síntese não é apenas uma repetição. Ela deve ajudar você a perceber se entendeu o caminho do raciocínio. Antes de ir aos exercícios, volte mentalmente ao problema inicial e tente explicar a solução com suas próprias palavras.

Exercícios de fixação

1. Classifique os dados nome, idade, nota e aprovado como tipos adequados.
2. Crie nomes de variáveis para guardar preço de venda, quantidade em estoque e média final.
3. Explique a diferença entre variável e constante.
4. Por que uma nota escolar deve ser normalmente do tipo real?
5. Crie um pseudocódigo que leia largura e altura de um retângulo e calcule sua área.

Desafio ou atividade prática

Crie um algoritmo que leia o valor de uma compra, aplique uma taxa fixa de entrega e mostre o total final. Use constante para a taxa de entrega.

Referências de apoio do capítulo

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. Lógica de programação: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo: Pearson Prentice Hall, 2005. ISBN 978-85-7605-024-7.

ASCENCIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Veneruchi de. Fundamentos da programação de computadores: algoritmos, Pascal, C/C++ e Java. 3. ed. São Paulo: Pearson Education do Brasil, 2012. ISBN 978-85-64574-16-8.

CAPÍTULO 6

Operadores e expressões

transformando dados em resultados

Identificação do capítulo

Item	Descrição
Tema	Operadores e expressões
Objetivo geral	Utilizar operadores aritméticos, relacionais e lógicos para calcular, comparar e combinar condições.
Palavras-chave	operadores; expressão; aritmético; relacional; lógico; precedência
Projeto em construção	Sistema de Controle Acadêmico Simplificado

Objetivos de aprendizagem

Ao final deste capítulo, espera-se que o estudante seja capaz de:

- usar operadores aritméticos em cálculos simples;
- interpretar operadores relacionais;
- combinar condições com operadores lógicos;
- entender a ordem de precedência em expressões;
- aplicar operadores em problemas reais.

Introdução

Com as variáveis compreendidas, a professora Helena perguntou: “guardar dados basta para resolver problemas?” A turma percebeu que não. É preciso transformar os dados. Notas precisam virar média. Preços precisam virar total. Idades precisam ser comparadas. Respostas precisam ser combinadas.

Operadores são símbolos ou palavras usados para realizar operações. Eles aparecem em cálculos, comparações e condições. Em lógica de programação, os operadores mais comuns são aritméticos, relacionais e lógicos.

Operadores aritméticos

Operador	Significado	Exemplo
+	adição	total ← a + b
-	subtração	troco ← pago - valor
*	multiplicação	area ← largura * altura
/	divisão	media ← soma / quantidade
% ou mod	resto da divisão	resto ← numero mod 2

O operador de resto da divisão é muito usado para descobrir se um número é par. Se numero mod 2 for igual a 0, o número é par; caso contrário, é ímpar.

```
resto ← numero mod 2
se resto = 0 então
   escreva("Número par")
senão
   escreva("Número ímpar")
fimse
```

Operadores relacionais

Operador	Significado	Exemplo
<>	diferente de	senha <> "123"
>	maior que	idade > 18
<	menor que	estoque < minimo
>=	maior ou igual	media >= 7
<=	menor ou igual	faltas <= 10

Operadores relacionais produzem respostas lógicas: verdadeiro ou falso. A expressão media >= 7 será verdadeira se a média for 7 ou mais; caso contrário, será falsa.

Operadores lógicos

Operador	Uso	Exemplo
e	Todas as condições precisam ser verdadeiras.	media >= 7 e faltas <= 10
ou	Basta uma condição ser verdadeira.	pagamento = "pix" ou pagamento = "cartao"
não	Inverte o valor lógico.	não aprovado

Operadores lógicos são muito importantes em decisões. Um aluno pode ser aprovado apenas se tiver média suficiente e frequência adequada. Nesse caso, as duas condições precisam ser verdadeiras.

```
se media >= 7 e faltas <= 10 então
   escreva("Aprovado")
senão
   escreva("Não aprovado")
fimse
```

Precedência e parênteses

Assim como na matemática, algumas operações são feitas antes de outras. Multiplicação e divisão costumam vir antes de soma e subtração. Para evitar dúvida, use parênteses. Eles tornam a intenção mais clara.

A expressão media ← nota1 + nota2 / 2 está errada para calcular média de duas notas, porque a divisão acontece antes da soma. O correto é media ← (nota1 + nota2) / 2.

ATENÇÃO
Quando houver dúvida, use parênteses. Eles não são sinal de fraqueza; são sinal de clareza.

Exemplo comentado: aprovação com frequência

```
algoritmo "AprovacaoComFrequencia"
var
   media: real
   faltas: inteiro
inicio
   escreva("Média do aluno: ")
   leia(media)
   escreva("Número de faltas: ")
   leia(faltas)

   se media >= 7 e faltas <= 10 então
      escreva("Aluno aprovado")
   senão
      escreva("Aluno não aprovado")
   fimse
fim
```

Caio achou estranho um aluno com média 9 ser não aprovado por excesso de faltas. A professora explicou que o operador “e” exige as duas condições. Lia percebeu que isso representa a regra da escola: não basta nota; também é preciso frequência. Nina comentou que muitas regras administrativas funcionam assim, combinando critérios.

Síntese ampliada do capítulo

Neste capítulo, vimos que operadores permitem transformar e comparar dados. Operadores aritméticos calculam; operadores relacionais comparam; operadores lógicos combinam condições.

Atenção especial deve ser dada aos parênteses e à ordem de precedência. Muitos erros de lógica surgem de expressões aparentemente simples, mas mal organizadas. Quando o resultado parecer estranho, revise a expressão antes de culpar o restante do algoritmo.

Nos exercícios, procure montar expressões aos poucos. Primeiro faça o cálculo, depois a comparação, e só então combine condições com “e” ou “ou”.

PENSE NISSO
A síntese não é apenas uma repetição. Ela deve ajudar você a perceber se entendeu o caminho do raciocínio. Antes de ir aos exercícios, volte mentalmente ao problema inicial e tente explicar a solução com suas próprias palavras.

Exercícios de fixação

1. Corrija a expressão media ← nota1 + nota2 / 2 para calcular a média corretamente.
2. Escreva uma condição para verificar se idade é maior ou igual a 18.
3. Escreva uma condição para verificar se um produto tem estoque menor que o mínimo.
4. Explique a diferença entre usar “e” e “ou”.
5. Crie um pseudocódigo que leia um número e informe se ele é par ou ímpar.

Desafio ou atividade prática

Crie um algoritmo que leia média e frequência percentual. O aluno será aprovado se média >= 6 e frequência >= 75.

Referências de apoio do capítulo

ASCENCIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Veneruchi de. Fundamentos da programação de computadores: algoritmos, Pascal, C/C++ e Java. 3. ed. São Paulo: Pearson Education do Brasil, 2012. ISBN 978-85-64574-16-8.

CAPÍTULO 7

Entrada, processamento e saída

o ciclo básico de todo programa

Identificação do capítulo

Item	Descrição
Tema	Entrada, processamento e saída
Objetivo geral	Aplicar o modelo entrada-processamento-saída em problemas diversos, escrevendo pseudocódigos completos.
Palavras-chave	entrada; processamento; saída; usuário; mensagens; validação
Projeto em construção	Sistema de Controle Acadêmico Simplificado

Objetivos de aprendizagem

Ao final deste capítulo, espera-se que o estudante seja capaz de:

- identificar entradas necessárias em um problema;
- organizar o processamento antes da saída;
- escrever mensagens claras para o usuário;
- construir pseudocódigos completos com leitura, cálculo e exibição;
- perceber a importância de validar dados.

Introdução

A professora Helena desenhou no quadro três blocos: Entrada, Processamento e Saída. Em seguida, disse: “se vocês entenderem bem esse modelo, terão uma base para quase todo programa”.

A entrada é o que o programa recebe. O processamento é o que o programa faz. A saída é o que o programa entrega. Essa estrutura aparece em calculadoras, caixas eletrônicos, sistemas acadêmicos, aplicativos de transporte, lojas virtuais e planilhas.

Caio disse que parecia simples demais. A professora concordou que era simples, mas não superficial. Muitos erros em programas iniciantes acontecem porque o estudante esquece uma entrada, calcula algo fora de ordem ou não mostra o resultado de forma clara.

Mensagens para o usuário

Um algoritmo não deve apenas ler dados. Ele deve orientar o usuário. Mensagens como “Digite a primeira nota” são melhores do que uma tela silenciosa esperando entrada. A clareza da interação reduz erros.

```
escreva("Digite a primeira nota: ")
leia(nota1)

// Melhor que apenas:
leia(nota1)
```

Exemplo 1: conversão de temperatura

```
algoritmo "CelsiusParaFahrenheit"
var
   celsius, fahrenheit: real
inicio
   escreva("Temperatura em Celsius: ")
   leia(celsius)

   fahrenheit ← (celsius * 9 / 5) + 32

```

```
   escreva("Temperatura em Fahrenheit: ", fahrenheit)
fim
```

Neste exemplo, a entrada é a temperatura em Celsius. O processamento aplica a fórmula de conversão. A saída mostra a temperatura em Fahrenheit.

Exemplo 2: cálculo de IMC como exercício lógico

O IMC é usado aqui apenas como exemplo matemático de entrada, processamento e saída, sem finalidade médica. Para decisões de saúde, deve-se procurar profissional qualificado.

```
algoritmo "CalculoIMC"
var
   peso, altura, imc: real
inicio
   escreva("Peso em kg: ")
   leia(peso)
   escreva("Altura em metros: ")
   leia(altura)

   imc ← peso / (altura * altura)

   escreva("IMC calculado: ", imc)
fim
```

Exemplo 3: valor de venda com quantidade

```
algoritmo "TotalVenda"
var
   produto: caractere
   quantidade: inteiro
   preco_unitario, total: real
inicio
   escreva("Produto: ")
   leia(produto)
   escreva("Quantidade: ")
   leia(quantidade)
```

```
   escreva("Preço unitário: ")
   leia(preco_unitario)

   total ← quantidade * preco_unitario

   escreva("Produto: ", produto)
   escreva("Total da venda: ", total)
fim
```

Validação inicial

Validar é verificar se os dados fazem sentido antes de usá-los. Se um algoritmo calcula total de venda, quantidade negativa não faz sentido. Se calcula média, nota menor que zero ou maior que dez pode ser inválida, dependendo da regra da escola.

Neste momento, veremos apenas a ideia. A validação completa usará decisões e repetições nos próximos capítulos.

```
se quantidade <= 0 então
   escreva("Quantidade inválida")
senão
   total ← quantidade * preco_unitario
   escreva("Total: ", total)
fimse
```

Síntese ampliada do capítulo

Neste capítulo, reforçamos o modelo entrada-processamento-saída. Ele é uma das bases mais importantes da programação. Mesmo programas grandes podem ser entendidos como conjuntos de entradas, processamentos e saídas interligados.

Também vimos que mensagens claras fazem parte da qualidade do algoritmo. Um programa não é escrito apenas para o computador; ele também precisa ser compreendido pelo usuário e pelo programador que dará manutenção.

Os exercícios deste capítulo pedem que você construa algoritmos completos. Sempre revise se há entrada suficiente, se o cálculo está correto e se a saída responde ao problema.

PENSE NISSO
A síntese não é apenas uma repetição. Ela deve ajudar você a perceber se entendeu o caminho do raciocínio. Antes de ir aos exercícios, volte mentalmente ao problema inicial e tente explicar a solução com suas próprias palavras.

Exercícios de fixação

1. Crie um pseudocódigo para calcular a área de um triângulo.
2. Crie um pseudocódigo para calcular o consumo médio de combustível de um veículo.
3. Identifique entrada, processamento e saída no cálculo de salário bruto.
4. Por que mensagens claras antes do comando leia são importantes?
5. Adicione uma validação simples ao algoritmo de venda para impedir quantidade negativa.

Desafio ou atividade prática

Crie um algoritmo que leia nome do funcionário, horas trabalhadas e valor da hora. Mostre o salário bruto e uma mensagem final organizada.

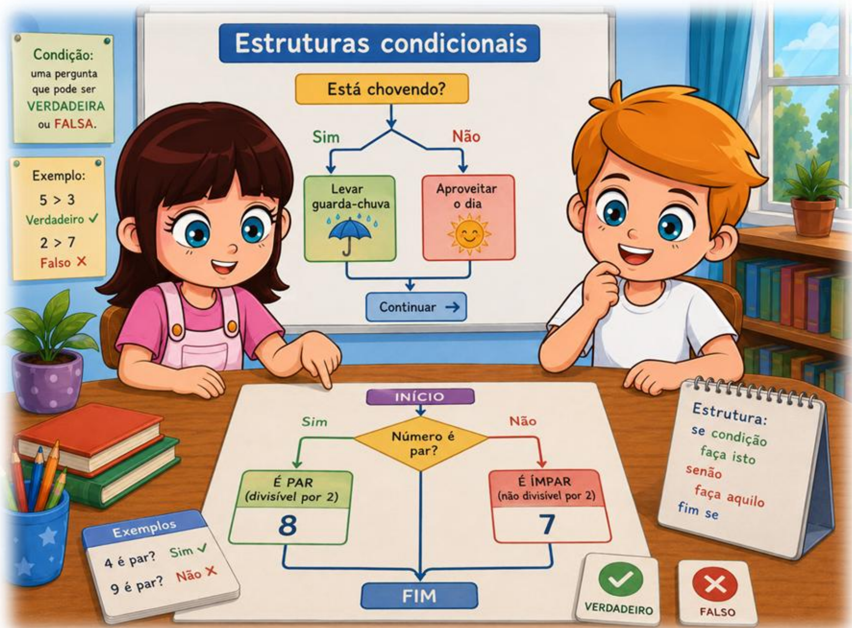
Referências de apoio do capítulo

BROOKSHEAR, J. Glenn; BRYLOW, Dennis. Computer science: an overview. 13. ed. Boston: Pearson, 2019. ISBN 978-0-13-487546-0.

CAPÍTULO 8

Estruturas condicionais

quando o algoritmo precisa escolher caminhos



Identificação do capítulo

Item	Descrição
Tema	Estruturas condicionais
Objetivo geral	Construir algoritmos que tomam decisões com base em condições simples, compostas e encadeadas.
Palavras-chave	se; senão; condição; decisão; operadores relacionais; operadores lógicos
Projeto em construção	Sistema de Controle Acadêmico Simplificado

Objetivos de aprendizagem

Ao final deste capítulo, espera-se que o estudante seja capaz de:

- usar a estrutura se-então-senão;
- diferenciar condição simples, composta e encadeada;
- combinar condições com operadores lógicos;
- evitar erros comuns em decisões;
- aplicar decisões em problemas acadêmicos e comerciais.

Introdução

Até agora, muitos algoritmos seguiram uma linha reta: ler, calcular e mostrar. Mas problemas reais nem sempre seguem um único caminho. Se a média for alta, o aluno é aprovado. Se for intermediária, fica em recuperação. Se for baixa, é reprovado. Essa mudança de caminho é uma decisão.

Estruturas condicionais permitem que o algoritmo escolha blocos de comandos conforme uma condição. A condição é uma expressão que resulta em verdadeiro ou falso.

A professora Helena explicou que a palavra "se" é uma das mais importantes da programação. Ela transforma uma sequência fixa em um algoritmo capaz de reagir a diferentes situações.

Condicional simples

```
se idade >= 18 então
   escreva("Maior de idade")
fimse
```

A condicional simples executa um comando apenas quando a condição é verdadeira. Se idade for menor que 18, nada será mostrado nesse exemplo.

Condicional composta

```
se idade >= 18 então
   escreva("Maior de idade")
senão
   escreva("Menor de idade")
fimse
```

A condicional composta define dois caminhos: um para condição verdadeira e outro para condição falsa.

Condicional encadeada

```
se media >= 7 então
   escreva("Aprovado")
```

```
senão
   se media >= 5 então
      escreva("Recuperação")
   senão
      escreva("Reprovado")
   fimse
fimse
```

A condicional encadeada permite mais de duas possibilidades. É importante organizar bem os intervalos para não criar classificações incorretas.

Exemplo comentado: desconto progressivo

A loja da situação-problema quer conceder descontos conforme o valor da compra: abaixo de R$ 100, sem desconto; de R$ 100 a R$ 499, desconto de 5%; a partir de R$ 500, desconto de 10%.

```
algoritmo "DescontoProgressivo"
var
   valor, desconto, total: real
inicio
   escreva("Valor da compra: ")
   leia(valor)

   se valor >= 500 então
      desconto ← valor * 0.10
   senão
      se valor >= 100 então
         desconto ← valor * 0.05
      senão
         desconto ← 0
      fimse
   fimse

   total ← valor - desconto
   escreva("Desconto: ", desconto)
   escreva("Total a pagar: ", total)
fim
```

Caio tentou testar primeiro se valor >= 100. A professora mostrou que isso capturaria também valores acima de 500, impedindo o desconto maior se a lógica fosse mal encadeada. Lia sugeriu começar pela maior faixa, e essa estratégia simplificou o raciocínio. A turma percebeu que a ordem das condições pode alterar o resultado.

Menu de opções

```
algoritmo "MenuSimples"
var
   opcao: inteiro
inicio
   escreva("1 - Cadastrar aluno")
   escreva("2 - Calcular média")
   escreva("3 - Sair")
   escreva("Escolha uma opção: ")
   leia(opcao)

   se opcao = 1 então
      escreva("Cadastro selecionado")
   senão
      se opcao = 2 então
         escreva("Cálculo de média selecionado")
      senão
         se opcao = 3 então
            escreva("Encerrando")
         senão
            escreva("Opção inválida")
         fimse
      fimse
   fimse
fim
```

Menus aparecem em muitos sistemas. Mesmo quando uma linguagem oferece estruturas específicas como escolha/caso, entender o menu com condicionais ajuda a compreender a lógica.

Síntese ampliada do capítulo

Neste capítulo, o algoritmo deixou de ser uma sequência única e passou a escolher caminhos. Essa é uma mudança fundamental. Programas úteis precisam reagir a dados diferentes, e isso acontece por meio de condições.

Vimos que a ordem das condições importa, especialmente em faixas de valores. Também vimos que operadores lógicos permitem combinar regras. Uma condição mal escrita pode aprovar quem deveria ser reprovado ou conceder desconto incorreto.

Nos exercícios, teste suas condições com valores diferentes. Para média, teste 4, 5, 6.9, 7 e 10. Para descontos, teste valores nas fronteiras. Os erros costumam aparecer exatamente nesses limites.

PENSE NISSO
A síntese não é apenas uma repetição. Ela deve ajudar você a perceber se entendeu o caminho do raciocínio. Antes de ir aos exercícios, volte mentalmente ao problema inicial e tente explicar a solução com suas próprias palavras.

Exercícios de fixação

1. Escreva uma condicional para verificar se uma pessoa pode votar considerando idade >= 16.
2. Crie um pseudocódigo que informe se um número é positivo, negativo ou zero.
3. Explique por que a ordem das condições importa no desconto progressivo.
4. Crie um menu com três opções e tratamento para opção inválida.
5. Escreva uma condição para aprovar aluno com média >= 6 e frequência >= 75.

Desafio ou atividade prática

Crie um algoritmo que leia idade e informe: criança, adolescente, adulto ou idoso, conforme faixas definidas por você. Explique suas faixas antes do pseudocódigo.

Referências de apoio do capítulo

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. Lógica de programação: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo: Pearson Prentice Hall, 2005. ISBN 978-85-7605-024-7.

CAPÍTULO 9

Estruturas de repetição

fazendo o algoritmo trabalhar várias vezes

Identificação do capítulo

Item	Descrição
Tema	Estruturas de repetição
Objetivo geral	Utilizar laços para repetir comandos com contador, condição inicial ou condição final.
Palavras-chave	repetição; enquanto; para; repita; contador; acumulador
Projeto em construção	Sistema de Controle Acadêmico Simplificado

Objetivos de aprendizagem

Ao final deste capítulo, espera-se que o estudante seja capaz de:

- entender por que repetições são necessárias;
- usar contadores e acumuladores;
- construir laços com quantidade conhecida;
- construir laços com critério de parada;
- evitar repetições infinitas.

Introdução

A secretaria da escola não calcula a média de apenas um aluno. Em uma turma, pode haver 30, 40 ou mais estudantes. Se o algoritmo tiver que repetir os mesmos passos para cada aluno, escrever tudo manualmente seria inviável.

Estruturas de repetição permitem executar um bloco de comandos várias vezes. A repetição pode acontecer por quantidade definida, como "para cada um dos 30 alunos", ou por condição, como "enquanto o usuário quiser continuar".

A professora Helena disse que repetição é uma das ideias mais poderosas da programação, porque permite automatizar tarefas cansativas e reduzir erros humanos.

Contadores e acumuladores

Contador é uma variável usada para contar quantas vezes algo aconteceu. Acumulador é uma variável usada para somar valores ao longo da repetição.

Se queremos calcular a média da turma, precisamos acumular as médias dos alunos e contar quantos alunos foram processados.

```
contador ← 0
soma ← 0

// a cada aluno processado:
contador ← contador + 1
soma ← soma + media_aluno
```

Laço para: quantidade conhecida

```
algoritmo "Tabuada"
var
   numero, i, resultado: inteiro
inicio
   escreva("Digite um número: ")
   leia(numero)
```

```
   para i de 1 até 10 faça
      resultado ← numero * i
      escreva(numero, " x ", i, " = ", resultado)
   fimpara
fim
```

Usamos "para" quando sabemos a quantidade de repetições. A tabuada repete de 1 até 10, portanto o laço tem início, fim e incremento claros.

Laço enquanto: condição inicial

```
algoritmo "Senha"
var
   senha: caractere
inicio
   escreva("Digite a senha: ")
   leia(senha)

   enquanto senha <> "1234" faça
      escreva("Senha incorreta. Tente novamente: ")
      leia(senha)
   fimenquanto

   escreva("Acesso liberado")
fim
```

No laço enquanto, a condição é testada antes de executar o bloco. Se a senha já estiver correta na primeira tentativa, o bloco de repetição não será executado.

Laço repita: condição final

```
algoritmo "SenhaRepita"
var
   senha: caractere
inicio
   repita
```

```
      escreva("Digite a senha: ")
      leia(senha)
   até senha = "1234"

   escreva("Acesso liberado")
fim
```

No repita, o bloco executa pelo menos uma vez, porque a condição é verificada ao final.

Exemplo comentado: média de uma turma

```
algoritmo "MediaTurma"
var
   quantidade, i: inteiro
   nota1, nota2, media_aluno, soma_medias, media_turma:
real
inicio
   escreva("Quantidade de alunos: ")
   leia(quantidade)

   soma_medias ← 0

   para i de 1 até quantidade faça
      escreva("Aluno ", i)
      escreva("Nota 1: ")
      leia(nota1)
      escreva("Nota 2: ")
      leia(nota2)

      media_aluno ← (nota1 + nota2) / 2
      soma_medias ← soma_medias + media_aluno

      escreva("Média do aluno: ", media_aluno)
   fimpara

   media_turma ← soma_medias / quantidade
```

```
   escreva("Média geral da turma: ", media_turma)
fim
```

Caio esqueceu de iniciar soma_medias com zero. A professora explicou que acumuladores precisam começar com um valor conhecido. Lia percebeu que a média da turma só pode ser calculada depois que todas as médias individuais foram somadas. Nina perguntou o que aconteceria se quantidade fosse zero; a professora disse que esse é um caso de validação importante, pois não podemos dividir por zero.

Repetição infinita

Uma repetição infinita acontece quando a condição de parada nunca é atingida. Por exemplo, se um laço enquanto espera que contador chegue a 10, mas o contador nunca aumenta, o algoritmo não termina.

```
contador ← 1
enquanto contador <= 10 faça
   escreva(contador)
   // erro: faltou contador ← contador + 1
fimenquanto
```

ATENÇÃO
Sempre verifique se a variável usada na condição de parada está sendo atualizada dentro do laço.

Síntese ampliada do capítulo

Neste capítulo, vimos que repetições evitam reescrever comandos e permitem processar muitos dados. Essa é uma habilidade essencial para qualquer programa que trabalhe com listas, turmas, vendas, tentativas ou cadastros.

Também aprendemos a diferença entre contador e acumulador. O contador conta eventos; o acumulador soma valores. Confundir esses papéis é um erro comum. Outro erro comum é esquecer de inicializar acumuladores ou atualizar contadores.

Nos exercícios, simule os laços manualmente. Anote o valor das variáveis a cada repetição. Esse cuidado ajuda a entender o funcionamento interno do algoritmo.

PENSE NISSO
A síntese não é apenas uma repetição. Ela deve ajudar você a perceber se entendeu o caminho do raciocínio. Antes de ir aos exercícios, volte mentalmente ao problema inicial e tente explicar a solução com suas próprias palavras.

Exercícios de fixação

1. Crie um algoritmo que mostre os números de 1 a 20.
2. Crie um algoritmo que leia 5 números e mostre a soma.
3. Explique a diferença entre contador e acumulador.
4. Crie um algoritmo de senha usando enquanto.
5. Identifique o erro em um laço que nunca atualiza a variável contador.

Desafio ou atividade prática

Crie um algoritmo que leia a idade de várias pessoas até que seja digitado -1. Ao final, mostre a quantidade de pessoas informadas e a média das idades.

Referências de apoio do capítulo

FARRER, Harry et al. Programação estruturada de computadores: algoritmos estruturados. 3. ed. Rio de Janeiro: LTC, 1999. ISBN 978-85-216-1180-6.

CAPÍTULO 10

Vetores e matrizes

organizando coleções de dados

Identificação do capítulo

Item	Descrição
Tema	Vetores e matrizes
Objetivo geral	Manipular conjuntos de valores relacionados usando estruturas indexadas simples.
Palavras-chave	vetor; matriz; índice; posição; lista; tabela
Projeto em construção	Sistema de Controle Acadêmico Simplificado

Objetivos de aprendizagem

Ao final deste capítulo, espera-se que o estudante seja capaz de:

- entender o conceito de vetor;
- preencher e percorrer vetores;
- buscar valores em vetores;
- entender o conceito de matriz;
- aplicar vetores e matrizes em notas de alunos.

Introdução

Até agora, quando queríamos guardar uma nota, criávamos uma variável. Mas e se uma turma tiver 40 alunos? Criar nota1, nota2, nota3 até nota40 não é uma boa solução. Precisamos de uma estrutura capaz de guardar vários valores relacionados.

Vetores e matrizes são estruturas de dados simples. Um vetor é como uma lista. Uma matriz é como uma tabela, com linhas e colunas. Elas permitem organizar coleções de dados e percorrê-las com repetição.

A professora Helena explicou que vetores e matrizes não são apenas “mais variáveis”. Eles mudam a forma de pensar, porque permitem trabalhar com conjuntos.

Vetor

Um vetor guarda vários valores do mesmo tipo, acessados por índices. Se notas[1] guarda a primeira nota, notas[2] guarda a segunda, e assim por diante.

```
var
   notas: vetor[1..5] de real
   i: inteiro
inicio
   para i de 1 até 5 faça
      escreva("Digite a nota ", i, ": ")
      leia(notas[i])
   fimpara

   para i de 1 até 5 faça
      escreva("Nota ", i, ": ", notas[i])
   fimpara
fim
```

Busca em vetor

```
algoritmo "BuscaNome"
var
   nomes: vetor[1..5] de caractere
```

```
    procurado: caractere
    i: inteiro
    encontrado: lógico
inicio
    para i de 1 até 5 faça
        escreva("Nome ", i, ": ")
        leia(nomes[i])
    fimpara

    escreva("Nome a procurar: ")
    leia(procurado)
    encontrado ← falso

    para i de 1 até 5 faça
        se nomes[i] = procurado então
            encontrado ← verdadeiro
        fimse
    fimpara

    se encontrado então
        escreva("Nome encontrado")
    senão
        escreva("Nome não encontrado")
    fimse
fim
```

Lia perguntou por que o algoritmo não parava quando encontrava o nome. A professora explicou que, dependendo do pseudocódigo, poderíamos interromper a busca. Nesta versão inicial, percorremos tudo para manter a estrutura simples. Em linguagens reais, há formas de parar o laço mais cedo.

Matriz

Matriz é uma estrutura com duas dimensões: linhas e colunas. Pode representar notas de alunos em várias avaliações. Por exemplo, notas[aluno, avaliacao].

```
algoritmo "NotasMatriz"
var
   notas: matriz[1..3, 1..2] de real
   aluno, avaliacao: inteiro
   soma, media: real
inicio
   para aluno de 1 até 3 faça
      para avaliacao de 1 até 2 faça
         escreva("Aluno ", aluno, " - Nota ",
avaliacao, ": ")
         leia(notas[aluno, avaliacao])
      fimpara
   fimpara

   para aluno de 1 até 3 faça
      soma ← 0
      para avaliacao de 1 até 2 faça
         soma ← soma + notas[aluno, avaliacao]
      fimpara
      media ← soma / 2
      escreva("Média do aluno ", aluno, ": ", media)
   fimpara
fim
```

Caio achou o laço dentro de outro laço estranho. A professora explicou que a matriz possui duas dimensões. O primeiro laço percorre os alunos; o segundo percorre as avaliações de cada aluno. Essa estrutura é chamada de laço aninhado.

Síntese ampliada do capítulo

Neste capítulo, aprendemos a guardar coleções de dados com vetores e matrizes. Vetores funcionam como listas; matrizes funcionam como tabelas. Essas estruturas permitem que o algoritmo trabalhe com muitos valores de forma organizada.

Também percebemos que vetores e matrizes quase sempre aparecem junto com repetições. Sem laços, seria trabalhoso preencher e

percorrer cada posição. Por isso, os capítulos anteriores são base para este conteúdo.

Nos exercícios, desenhe as posições. Visualizar os índices evita confusão. Lembre-se de que muitos erros acontecem por acessar posição errada ou esquecer o limite do vetor.

PENSE NISSO
A síntese não é apenas uma repetição. Ela deve ajudar você a perceber se entendeu o caminho do raciocínio. Antes de ir aos exercícios, volte mentalmente ao problema inicial e tente explicar a solução com suas próprias palavras.

Exercícios de fixação

1. Crie um vetor para armazenar 10 idades e depois mostre todas.
2. Crie um algoritmo que leia 5 números em um vetor e mostre o maior valor.
3. Explique a diferença entre vetor e matriz.
4. Crie uma matriz 2x3 para armazenar notas de 2 alunos em 3 avaliações.
5. Por que laços aninhados são úteis em matrizes?

Desafio ou atividade prática

Crie um algoritmo que leia o nome e a média de 5 alunos usando vetores. Ao final, mostre os alunos aprovados com média >= 7.

Referências de apoio do capítulo

CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. Introduction to algorithms. 4. ed. Cambridge, MA: The MIT Press, 2022. ISBN 978-0-262-04630-5.

CAPÍTULO 11

Modularização

dividindo problemas para resolver melhor

Identificação do capítulo

Item	Descrição
Tema	Modularização
Objetivo geral	Organizar algoritmos em procedimentos e funções, favorecendo clareza, reutilização e manutenção.
Palavras-chave	procedimento; função; parâmetro; retorno; módulo; reutilização
Projeto em construção	Sistema de Controle Acadêmico Simplificado

Objetivos de aprendizagem

Ao final deste capítulo, espera-se que o estudante seja capaz de:

- compreender por que dividir algoritmos em módulos;
- diferenciar procedimento e função;
- usar parâmetros simples;
- entender retorno de valores;
- aplicar modularização no projeto acadêmico.

Introdução

O algoritmo do sistema acadêmico estava crescendo. Já havia cadastro, cálculo de média, situação do aluno, relatórios e testes. Caio comentou que o pseudocódigo estava ficando grande demais para acompanhar. A professora disse que aquele era o momento ideal para falar sobre modularização.

Modularizar é dividir uma solução em partes menores, chamadas módulos. Cada módulo deve ter uma responsabilidade clara. Assim como uma escola possui secretaria, coordenação e sala de aula, um programa pode possuir partes responsáveis por tarefas específicas.

Em programação, os módulos mais comuns para iniciantes são procedimentos e funções. Procedimentos executam ações. Funções executam uma tarefa e devolvem um valor.

Procedimento

Procedimento é um bloco de comandos nomeado. Ele pode receber informações e executar uma ação, mas não precisa devolver um valor diretamente.

```
procedimento mostrarCabecalho()
inicio
   escreva("Sistema Acadêmico Simplificado")
   escreva("-------------------------------")
fimprocedimento
```

Função

Função é um bloco que realiza uma tarefa e retorna um valor. Ela é útil quando precisamos calcular algo e usar o resultado depois.

```
função calcularMedia(nota1: real, nota2: real): real
var
   media: real
inicio
   media ← (nota1 + nota2) / 2
```

```
   retorne media
fimfunção
```

Usando a função no algoritmo principal

```
algoritmo "UsoFuncaoMedia"
var
   n1, n2, media_final: real
inicio
   escreva("Nota 1: ")
   leia(n1)
   escreva("Nota 2: ")
   leia(n2)

   media_final ← calcularMedia(n1, n2)

   escreva("Média: ", media_final)
fim
```

Parâmetros

Parâmetros são informações enviadas para um procedimento ou função. No exemplo calcularMedia(nota1, nota2), os parâmetros permitem que a função seja reutilizada com diferentes notas.

Exemplo comentado: função para situação do aluno

```
função obterSituacao(media: real): caractere
inicio
   se media >= 7 então
      retorne "Aprovado"
   senão
      se media >= 5 então
         retorne "Recuperação"
      senão
         retorne "Reprovado"
      fimse
```

```
   fimse
fimfunção
```

A professora explicou que a regra de situação do aluno agora está concentrada em um único lugar. Se a escola mudar a regra, alteramos a função, não todas as partes do programa. Lia percebeu que isso reduz erros. Caio disse que o algoritmo principal fica mais curto. Nina observou que a função se parece com um procedimento administrativo reaproveitável.

Algoritmo principal modularizado

```
algoritmo "AlunoModularizado"
var
   nome, situacao: caractere
   nota1, nota2, media: real
inicio
   mostrarCabecalho()

   escreva("Nome do aluno: ")
   leia(nome)
   escreva("Nota 1: ")
   leia(nota1)
   escreva("Nota 2: ")
   leia(nota2)

   media ← calcularMedia(nota1, nota2)
   situacao ← obterSituacao(media)

   escreva("Aluno: ", nome)
   escreva("Média: ", media)
   escreva("Situação: ", situacao)
fim
```

Síntese ampliada do capítulo

Neste capítulo, aprendemos que modularização é uma estratégia para controlar a complexidade. Quando um algoritmo cresce, dividir em partes menores ajuda a ler, testar, corrigir e reaproveitar.

Procedimentos executam ações. Funções retornam valores. Parâmetros permitem enviar dados para o módulo. Esses conceitos aparecem em praticamente todas as linguagens de programação, ainda que com sintaxes diferentes.

Nos exercícios, procure identificar responsabilidades. Se uma parte calcula média, talvez possa ser função. Se uma parte apenas mostra um cabeçalho, talvez possa ser procedimento. Essa forma de pensar prepara você para projetos maiores.

PENSE NISSO
A síntese não é apenas uma repetição. Ela deve ajudar você a perceber se entendeu o caminho do raciocínio. Antes de ir aos exercícios, volte mentalmente ao problema inicial e tente explicar a solução com suas próprias palavras.

Exercícios de fixação

1. Explique a diferença entre procedimento e função.
2. Crie uma função que receba dois números e retorne a soma.
3. Crie um procedimento que mostre uma linha de separação na tela.
4. Por que modularizar ajuda na manutenção?
5. Crie uma função que receba a média e retorne verdadeiro se o aluno estiver aprovado.

Desafio ou atividade prática

Modularize um algoritmo de venda criando uma função calcularTotal e uma função calcularDesconto.

Referências de apoio do capítulo

ZIVIANI, Nivio. Projeto de algoritmos: com implementações em Pascal e C. 3. ed. rev. e ampl. São Paulo: Cengage Learning, 2011. ISBN 978-85-221-1050-6.

CAPÍTULO 12

Testes, erros e boas práticas

aprendendo a revisar algoritmos

Identificação do capítulo

Item	Descrição
Tema	Testes, erros e boas práticas
Objetivo geral	Testar algoritmos, identificar erros comuns e aplicar boas práticas de organização e legibilidade.
Palavras-chave	teste de mesa; erro de lógica; erro de sintaxe; depuração; legibilidade
Projeto em construção	Sistema de Controle Acadêmico Simplificado

Objetivos de aprendizagem

Ao final deste capítulo, espera-se que o estudante seja capaz de:

- explicar o que é teste de mesa;
- diferenciar erro de sintaxe, execução e lógica;
- simular algoritmos manualmente;
- identificar casos de teste importantes;
- aplicar boas práticas de escrita de pseudocódigo.

Introdução

Caio comemorou quando seu algoritmo “parecia pronto”. A professora Helena fez uma pergunta simples: “como você sabe que está certo?”. Ele respondeu que havia lido e parecia fazer sentido. A professora então explicou que ler é importante, mas não basta. É preciso testar.

Testar é executar ou simular o algoritmo com dados escolhidos para verificar se o resultado está correto. Em programação, erros são normais. O estudante precisa aprender a encontrá-los, compreendê-los e corrigi-los.

A apostila usa a expressão teste de mesa para a simulação manual de um algoritmo. É como acompanhar, linha por linha, o valor das variáveis.

Tipos de erro

Tipo de erro	Descrição	Exemplo
Sintaxe	Comando escrito fora do padrão da linguagem.	Esquecer fimse em uma estrutura condicional.
Execução	Erro ocorre durante a execução.	Dividir por zero.
Lógica	Programa executa, mas resultado está errado.	Calcular média sem parênteses.

Teste de mesa

```
algoritmo "Media"
var
   nota1, nota2, media: real
inicio
   nota1 ← 8
   nota2 ← 6
   media ← (nota1 + nota2) / 2
   escreva(media)
fim
```

Linha/ação	nota1	nota2	media	Comentário
nota1 ← 8	8	-	-	Primeira atribuição.
nota2 ← 6	8	6	-	Segunda atribuição.
media ← (8+6)/2	8	6	7	Cálculo correto.
escreva(media)	8	6	7	Saída esperada: 7.

Casos de teste

Caso de teste é um conjunto de dados usado para verificar o algoritmo. Bons testes incluem casos comuns, limites e situações inválidas.

No algoritmo de aprovação, testar apenas média 8 não basta. É necessário testar 7, 6.9, 5, 4.9 e outros valores nas fronteiras. Os limites revelam erros de comparação.

Média	Resultado esperado	Por que testar?
8.0	Aprovado	Caso comum.
7.0	Aprovado	Limite da aprovação.
6.9	Recuperação	Logo abaixo do limite.
5.0	Recuperação	Limite da recuperação.
4.9	Reprovado	Logo abaixo da recuperação.

Boas práticas de legibilidade

- Use nomes significativos para variáveis.
- Mantenha indentação em estruturas condicionais e laços.
- Escreva uma responsabilidade por bloco.
- Evite repetir regras em muitos lugares.
- Comente trechos difíceis, mas não comente o óbvio.
- Teste com valores de limite.

BOA PRÁTICA
Um algoritmo bem escrito não precisa ser explicado a cada linha. A própria organização, os nomes e a indentação devem ajudar o leitor.

Síntese ampliada do capítulo

Neste capítulo, vimos que testar é parte essencial da programação. Um algoritmo que parece correto ainda pode falhar em casos específicos. Por isso, devemos simular, testar limites e revisar as condições.

Também diferenciamos tipos de erro. Erros de lógica são especialmente perigosos porque o programa pode executar sem avisar, mas entregar resultado incorreto. O teste de mesa ajuda a encontrar esse tipo de problema.

Nos exercícios, não apenas escreva algoritmos. Escolha dados de teste e diga o resultado esperado. Esse hábito aproxima você de uma prática profissional.

PENSE NISSO
A síntese não é apenas uma repetição. Ela deve ajudar você a perceber se entendeu o caminho do raciocínio. Antes de ir aos exercícios, volte mentalmente ao problema inicial e tente explicar a solução com suas próprias palavras.

Exercícios de fixação

1. Explique o que é teste de mesa.
2. Dê um exemplo de erro de lógica em cálculo de média.
3. Crie três casos de teste para um algoritmo que verifica maioridade.
4. Por que testar valores de limite é importante?
5. Reescreva um algoritmo simples usando nomes de variáveis mais significativos.

Desafio ou atividade prática

Escolha um algoritmo de capítulo anterior e monte uma tabela de teste de mesa completa para pelo menos três entradas diferentes.

Referências de apoio do capítulo

DOWNEY, Allen B. Think Python: how to think like a computer scientist. 2. ed. Sebastopol: O'Reilly Media, 2016. ISBN 978-1-4919-3936-9.

CAPÍTULO 13

Projeto final orientado

Sistema de Controle Acadêmico Simplificado

Identificação do capítulo

Item	Descrição
Tema	Projeto final orientado
Objetivo geral	Integrar os conceitos estudados em um sistema acadêmico simples com cadastro, notas, médias, situação e relatório.
Palavras-chave	projeto; integração; requisitos; menu; vetor; função; teste
Projeto em construção	Sistema de Controle Acadêmico Simplificado

Objetivos de aprendizagem

Ao final deste capítulo, espera-se que o estudante seja capaz de:

- compreender requisitos de um projeto simples;
- planejar entradas, processamentos e saídas;
- usar decisões, repetições, vetores e funções;
- organizar uma solução maior em etapas;
- testar o projeto final com casos variados.

Introdução

Ao final da jornada, a professora Helena retomou o problema inicial da secretaria escolar. Agora a turma já conhecia variáveis, operadores, entrada e saída, decisões, repetições, vetores, matrizes, modularização e testes. Era hora de integrar tudo.

O projeto final não precisa ser um sistema comercial completo. Ele será um Sistema de Controle Acadêmico Simplificado, suficiente para consolidar a lógica. O objetivo é cadastrar alunos, registrar notas, calcular médias, informar situações e gerar relatórios simples.

A professora explicou que projetos não começam pelo código. Começam por requisitos. Requisitos são necessidades que o sistema deve atender.

Requisitos do projeto

- Permitir cadastrar o nome de até 10 alunos.
- Registrar duas notas para cada aluno.
- Calcular a média individual.
- Classificar a situação como Aprovado, Recuperação ou Reprovado.
- Mostrar relatório com nome, média e situação.
- Calcular a média geral da turma.
- Identificar a maior e a menor média.
- Usar menu de opções.
- Usar funções para cálculo da média e obtenção da situação.
- Apresentar testes de mesa ou casos de teste.

Planejamento das estruturas

Informação	Estrutura sugerida	Comentário
Nomes dos alunos	vetor nomes[1..10]	Cada posição guarda um nome.
Nota 1	vetor nota1[1..10]	Primeira nota de cada aluno.
Nota 2	vetor nota2[1..10]	Segunda nota de cada aluno.
Médias	vetor medias[1..10]	Resultado calculado.
Quantidade cadastrada	inteiro qtd	Controla quantos alunos existem.

Funções do projeto

```
função calcularMedia(n1: real, n2: real): real
inicio
   retorne (n1 + n2) / 2
fimfunção

função obterSituacao(media: real): caractere
inicio
   se media >= 7 então
      retorne "Aprovado"
   senão
      se media >= 5 então
         retorne "Recuperação"
      senão
         retorne "Reprovado"
      fimse
   fimse
fimfunção
```

Pseudocódigo completo proposto

```
algoritmo "SistemaAcademicoSimplificado"
var
   nomes: vetor[1..10] de caractere
   nota1, nota2, medias: vetor[1..10] de real
   qtd, opcao, i: inteiro
   maior, menor, soma, media_geral: real
inicio
   qtd ← 0

   repita
      escreva("===== MENU =====")
      escreva("1 - Cadastrar aluno")
      escreva("2 - Listar relatório")
      escreva("3 - Mostrar estatísticas")
      escreva("4 - Sair")
```

```
      escreva("Opção: ")
      leia(opcao)

      se opcao = 1 então
         se qtd < 10 então
            qtd ← qtd + 1
            escreva("Nome do aluno: ")
            leia(nomes[qtd])
            escreva("Nota 1: ")
            leia(nota1[qtd])
            escreva("Nota 2: ")
            leia(nota2[qtd])
            medias[qtd] ← calcularMedia(nota1[qtd],
nota2[qtd])
            escreva("Aluno cadastrado com sucesso.")
         senão
            escreva("Limite de alunos atingido.")
         fimse
      senão
         se opcao = 2 então
            se qtd = 0 então
               escreva("Nenhum aluno cadastrado.")
            senão
               para i de 1 até qtd faça
                  escreva(nomes[i], " - Média: ",
medias[i],
                          " - Situação: ",
obterSituacao(medias[i]))
               fimpara
            fimse
         senão
            se opcao = 3 então
               se qtd = 0 então
                  escreva("Nenhum dado para
estatísticas.")
               senão
                  soma ← 0
```

```
                 maior ← medias[1]
                 menor ← medias[1]

                 para i de 1 até qtd faça
                    soma ← soma + medias[i]
                    se medias[i] > maior então
                       maior ← medias[i]
                    fimse
                    se medias[i] < menor então
                       menor ← medias[i]
                    fimse
                 fimpara

                 media_geral ← soma / qtd
                 escreva("Média geral: ", media_geral)
                 escreva("Maior média: ", maior)
                 escreva("Menor média: ", menor)
              fimse
           senão
              se opcao <> 4 então
                 escreva("Opção inválida.")
              fimse
           fimse
        fimse
     fimse
   até opcao = 4
   escreva("Sistema encerrado.")
fim
```

Comentário da professora sobre o projeto

A professora Helena destacou que o projeto reúne praticamente todos os conceitos da apostila. Há variáveis simples, vetores, funções, decisões, repetições, acumulador, maior e menor valor, menu e tratamento de situações vazias.

Lia percebeu que o sistema não permite cadastrar mais de 10 alunos. Caio observou que o relatório só funciona se houver alunos cadastrados.

Nina comentou que a parte de estatísticas se parece com relatórios administrativos que ela já havia usado no trabalho.

A professora concluiu que o projeto ainda pode melhorar, mas já representa uma solução lógica consistente. Em uma linguagem real, seria possível acrescentar gravação em arquivo, interface gráfica, banco de dados e validações mais fortes.

Casos de teste recomendados

Teste	Dados	Resultado esperado
Listar sem cadastro	opção 2 antes de cadastrar	Mensagem: nenhum aluno cadastrado.
Cadastrar aprovado	notas 8 e 9	Média 8.5; Aprovado.
Cadastrar recuperação	notas 5 e 6	Média 5.5; Recuperação.
Cadastrar reprovado	notas 3 e 4	Média 3.5; Reprovado.
Estatísticas	três alunos cadastrados	Média geral, maior e menor média corretas.

Síntese ampliada do capítulo

O projeto final mostra que lógica de programação é cumulativa. Cada capítulo acrescentou uma ferramenta. Isoladamente, cada ferramenta parecia pequena; juntas, elas permitem construir um sistema com comportamento útil.

Também fica claro que programar não é apenas escrever comandos. É planejar, organizar dados, prever situações vazias, criar funções, testar casos e pensar na pessoa que usará o sistema.

Ao terminar este projeto, o estudante estará preparado para iniciar uma linguagem de programação com mais segurança, porque já entende a lógica por trás dos comandos.

PENSE NISSO
A síntese não é apenas uma repetição. Ela deve ajudar você a perceber se entendeu o caminho do raciocínio. Antes de ir aos exercícios, volte mentalmente ao problema inicial e tente explicar a solução com suas próprias palavras.

Exercícios de fixação

1. Explique quais conceitos da apostila aparecem no projeto final.
2. Por que o projeto verifica se qtd = 0 antes de listar relatório?
3. Qual é a função do vetor medias?
4. Por que maior e menor recebem medias[1] antes do laço?
5. Sugira duas melhorias para o projeto.

Desafio ou atividade prática

Implemente o projeto final em pseudocódigo, Portugol ou linguagem escolhida pelo professor. Entregue também uma tabela com pelo menos cinco casos de teste.

Referências de apoio do capítulo

ASCENCIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Veneruchi de. Fundamentos da programação de computadores: algoritmos, Pascal, C/C++ e Java. 3. ed. São Paulo: Pearson Education do Brasil, 2012. ISBN 978-85-64574-16-8.

ZIVIANI, Nivio. Projeto de algoritmos: com implementações em Pascal e C. 3. ed. rev. e ampl. São Paulo: Cengage Learning, 2011. ISBN 978-85-221-1050-6.

APÊNDICE A - Guia rápido de pseudocódigo

Este apêndice reúne comandos usados ao longo da apostila. Ele não substitui os capítulos, mas serve como consulta rápida durante exercícios e projetos.

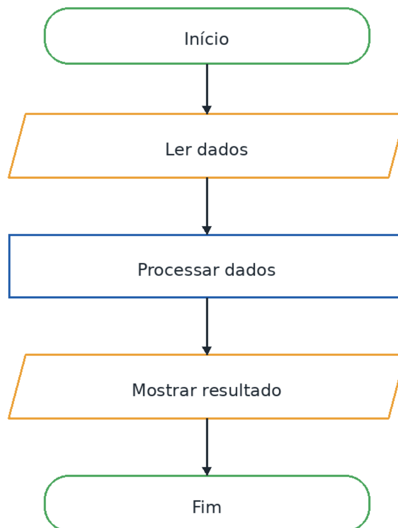
Como o pseudocódigo é uma linguagem didática, pequenas variações podem aparecer em outros materiais. O importante é manter coerência dentro do mesmo algoritmo.

Comando	Finalidade	Exemplo
leia	Receber dados.	leia(nome)
escreva	Mostrar dados.	escreva("Olá")
←	Atribuir valor.	total ← preco * qtd
se/senão/fimse	Tomar decisão.	se media >= 7 então
para/fimpara	Repetir com contador.	para i de 1 até 10 faça
enquanto/fimenquanto	Repetir enquanto condição for verdadeira.	enquanto senha <> "123" faça
repita/até	Repetir até condição final.	repita ... até opcao = 0
função/retorne	Criar módulo que devolve valor.	retorne media
procedimento	Criar módulo de ação.	procedimento mostrarMenu()

APÊNDICE B - Modelos visuais de fluxogramas

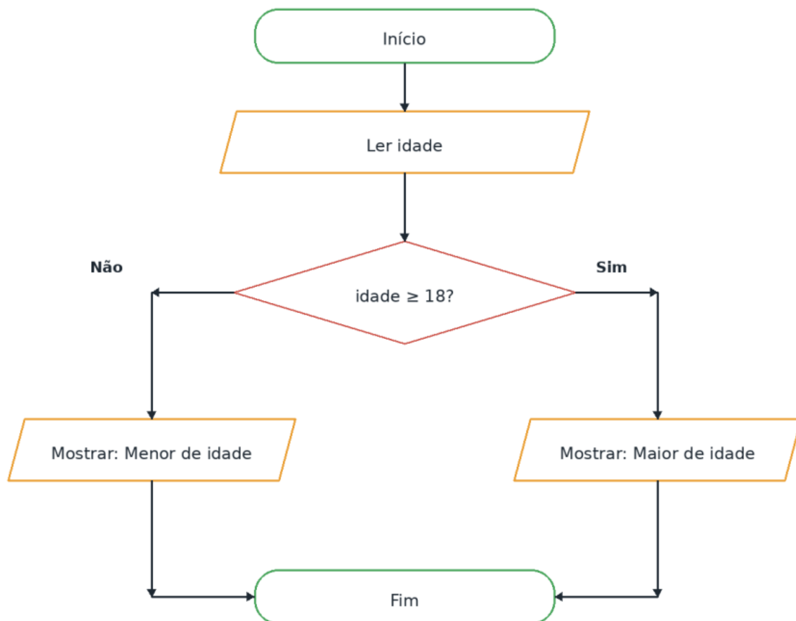
Apêndice B — Modelo 1: Sequência simples

Fluxo linear sem decisões nem repetições.



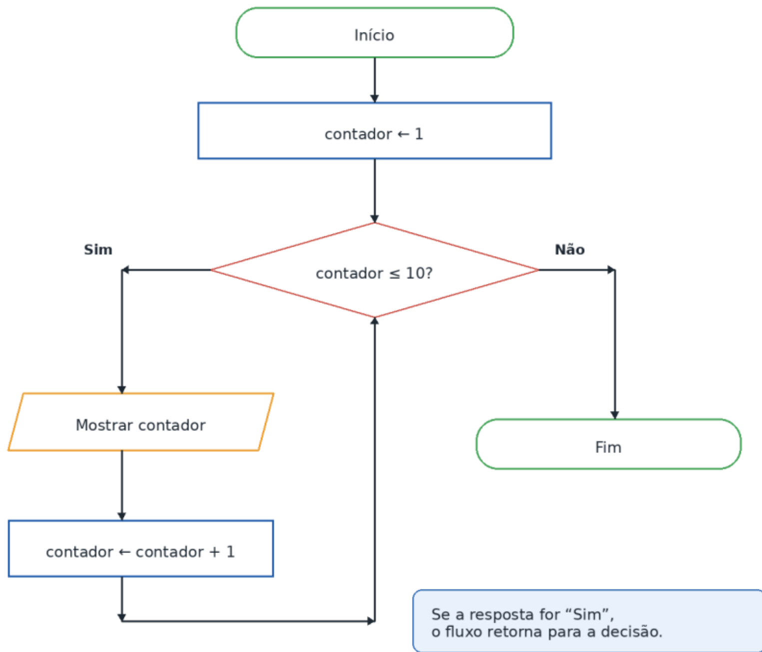
Apêndice B — Modelo 2: Decisão simples

Exemplo com dois caminhos possíveis a partir de uma pergunta.



Apêndice B — Modelo 3: Repetição

Exemplo de laço de repetição controlado por contador.



APÊNDICE C - Soluções comentadas dos exercícios e desafios

As soluções a seguir são propostas de resposta. Em lógica de programação, pode haver mais de uma forma correta de resolver um problema. Compare sua solução com a proposta, observe diferenças e tente explicar por que ambas funcionam ou por que uma delas precisa ser corrigida.

Capítulo 1 - Exercício 2

Solução proposta: Troco de compra

```
Entrada: valor da compra e valor pago.
Processamento: troco ← valor_pago - valor_compra.
Saída: mostrar o troco. Se o valor pago for menor que a
compra, informar pagamento insuficiente.
```

Capítulo 1 - Desafio

Solução proposta: Situação da rotina

```
Exemplo: controle de água diária.
Entrada: quantidade de copos bebidos.
Processamento: comparar com meta diária.
Saída: informar se a meta foi atingida.
Erro possível: não definir a meta antes da comparação.
```

Capítulo 2 - Exercício 1

Solução proposta: Troco em linguagem natural

```
1. Receber valor da compra.
2. Receber valor pago.
3. Se valor pago for menor que compra, informar valor
insuficiente.
4. Caso contrário, calcular troco.
5. Mostrar troco.
```

Capítulo 3 - Exercício 3

Solução proposta: Fluxograma textual de maioridade

Capítulo 4 - Exercício 2

Solução proposta: Produto com 10% de desconto

```
algoritmo "Desconto10"
var
   preco, final: real
inicio
   escreva("Preço: ")
   leia(preco)
   final ← preco * 0.90
   escreva("Valor com desconto: ", final)
fim
```

Capítulo 5 - Exercício 5

Solução proposta: Área do retângulo

```
algoritmo "AreaRetangulo"
var
   largura, altura, area: real
inicio
   leia(largura)
   leia(altura)
   area ← largura * altura
   escreva(area)
fim
```

Capítulo 6 - Exercício 5

Solução proposta: Par ou ímpar

```
algoritmo "ParImpar"
var
   numero: inteiro
inicio
   leia(numero)
```

```
   se numero mod 2 = 0 então
      escreva("Par")
   senão
      escreva("Ímpar")
   fimse
fim
```

Capítulo 7 - Exercício 1

Solução proposta: Área do triângulo

```
algoritmo "AreaTriangulo"
var
   base, altura, area: real
inicio
   leia(base)
   leia(altura)
   area ← (base * altura) / 2
   escreva(area)
fim
```

Capítulo 8 - Exercício 2

Solução proposta: Positivo, negativo ou zero

```
se numero > 0 então
   escreva("Positivo")
senão
   se numero < 0 então
      escreva("Negativo")
   senão
      escreva("Zero")
   fimse
fimse
```

Capítulo 9 - Exercício 2

Solução proposta: Somar 5 números

```
soma ← 0
para i de 1 até 5 faça
   leia(numero)
   soma ← soma + numero
fimpara
escreva(soma)
```

Capítulo 10 - Exercício 2

Solução proposta: Maior valor em vetor

```
maior ← numeros[1]
para i de 2 até 5 faça
   se numeros[i] > maior então
      maior ← numeros[i]
   fimse
fimpara
escreva(maior)
```

Capítulo 11 - Exercício 2

Solução proposta: Função soma

```
função somar(a: real, b: real): real
inicio
   retorne a + b
fimfunção
```

Capítulo 12 - Exercício 3

Solução proposta: Casos de teste para maioridade

```
Teste 1: idade 17 → Menor de idade.
Teste 2: idade 18 → Maior de idade.
Teste 3: idade 60 → Maior de idade.
O valor 18 é essencial por ser o limite.
```

Capítulo 13 - Desafio

Solução proposta: Casos de teste do projeto final

```
1. Listar sem alunos: mensagem de vazio.
2. Cadastrar Ana com 8 e 9: média 8,5, aprovada.
3. Cadastrar Bruno com 5 e 6: média 5,5, recuperação.
4. Cadastrar Carla com 3 e 4: média 3,5, reprovada.
5. Estatísticas: verificar média geral, maior e menor média.
```

Referências

ASCENCIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Veneruchi de. Fundamentos da programação de computadores: algoritmos, Pascal, C/C++ e Java. 3. ed. São Paulo: Pearson Education do Brasil, 2012. ISBN 978-85-64574-16-8.

BROOKSHEAR, J. Glenn; BRYLOW, Dennis. Computer science: an overview. 13. ed. Boston: Pearson, 2019. ISBN 978-0-13-487546-0.

CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. Introduction to algorithms. 4. ed. Cambridge, MA: The MIT Press, 2022. ISBN 978-0-262-04630-5.

DOWNEY, Allen B. Think Python: how to think like a computer scientist. 2. ed. Sebastopol: O'Reilly Media, 2016. ISBN 978-1-4919-3936-9.

FARRER, Harry et al. Programação estruturada de computadores: algoritmos estruturados. 3. ed. Rio de Janeiro: LTC, 1999. ISBN 978-85-216-1180-6.

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. Lógica de programação: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo: Pearson Prentice Hall, 2005. ISBN 978-85-7605-024-7.

KNUTH, Donald E. The art of computer programming: volume 1: fundamental algorithms. 3. ed. Reading, MA: Addison-Wesley Professional, 1997. ISBN 978-0-201-89683-1.

PAPERT, Seymour. Mindstorms: children, computers, and powerful ideas. New York: Basic Books, 1980. ISBN 0-465-04627-4.

PÓLYA, George. A arte de resolver problemas: um novo aspecto do método matemático. Tradução e adaptação de Heitor Lisboa de Araújo. Rio de Janeiro: Interciência, 1995. ISBN 978-85-7193-136-7.

SEBESTA, Robert W. Concepts of programming languages. 12. ed. Boston: Pearson, 2018. ISBN 978-0-13-499718-6.

WING, Jeannette M. Computational thinking. Communications of the ACM, New York, v. 49, n. 3, p. 33-35, mar. 2006. DOI: 10.1145/1118178.1118215.

WIRTH, Niklaus. Algorithms + data structures = programs. Englewood Cliffs, NJ: Prentice-Hall, 1976. ISBN 978-0-13-022418-7.

ZIVIANI, Nivio. Projeto de algoritmos: com implementações em Pascal e C. 3. ed. rev. e ampl. São Paulo: Cengage Learning, 2011. ISBN 978-85-221-1050-6.

Anotações

Notas sobre o Autor

Douglas Francisco Ribeiro é professor, pesquisador e profissional da área de tecnologia. Formado no Curso Superior em Tecnologia em Sistemas para Internet, pela Faculdade de Tecnologia de Taquaritinga (Fatec Taquaritinga). É Mestre em Bioengenharia (PPGIB) pela Universidade de São Paulo (USP), um programa Interunidades formada pela Escola de Engenharia de São Carlos (EESC), à Faculdade de Medicina de Ribeirão Preto (FMRP) e o Instituto de Química de São Carlos (IQSC). Atualmente, é doutorando pela Universidade Federal de São Carlos (UFSCar), com estudos na área de Engenharia de Software.

Atua na área de tecnologia há mais de 30 anos, acumulando experiência prática em desenvolvimento de sistemas, projetos tecnológicos, inovação e formação profissional. Na docência, possui mais de 10 anos de atuação pelo Centro Paula Souza, contribuindo para a formação de estudantes em cursos técnicos e superiores de tecnologia.

Atualmente, exerce a função de coordenador do Curso Superior em Desenvolvimento de Software Multiplataforma na Fatec Matão, desenvolvendo atividades acadêmicas, pedagógicas e de gestão voltadas à formação de profissionais para o mercado de tecnologia.

Como autor, dedica-se à produção de materiais didáticos com linguagem clara, aplicada e acessível, buscando aproximar teoria e prática no ensino de computação e desenvolvimento de software.

COLEÇÃO

DESENVOLVIMENTO DE SOFTWARE NA PRÁTICA

LINHA 1 - FUNDAMENTOS DE PROGRAMAÇÃO

APOSTILA 1

LÓGICA DE PROGRAMAÇÃO
PARA INICIANTES

ALGORITMOS, RACIOCÍNIO COMPUTACIONAL E PRIMEIROS PROGRAMAS NA PRÁTICA

Parabéns por concluir esta apostila. Continue praticando, criando soluções e desenvolvendo seu raciocínio computacional. O código é apenas o começo; a lógica é o caminho para transformar ideias em projetos reais.

PROF. DOUGLAS FRANCISCO RIBEIRO