Virando o Jogo da Imitação

Como ensinar raciocínio algorítmico apesar da Inteligência Artificial

Frank Coelho de Alcantara

2025-06-20

Sumário

1 Introdução

No coração da educação de disciplinas relacionadas à computação, especialmente nas disciplinas introdutórias, reside um desafio pedagógico fundamental percebido nos problemas encontrados para ensinar iniciantes a pensar como um programador. Essa habilidade, formalmente denominada **Raciocínio Algorítmico**, é a capacidade de decompor um problema complexo em uma sequência finita, não ambígua e executável de passos lógicos (1). O **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers, **IEEE**, define o **Raciocínio Algorítmico** como a habilidade de chegar a uma solução através da definição clara dos passos necessários (2). Não se trata de encontrar uma única resposta, mas de desenvolver um conjunto de regras ou instruções (um algoritmo) que, se seguido precisamente, resolve não apenas o problema original, mas também problemas similares (1). Como tal habilidade é essencial para a formação de profissionais competentes em Ciência da Computação e Engenharia, o ensino do **Raciocínio Algorítmico** deve ser uma prioridade curricular.

O Jogo da imitação é o título no Brasil de uma adaptação cinematográfica da vida de Alan Turing (3). O jogo da imitação também seria um bom nome para os problemas pedagógicos que encontrados no ensino de Ciência e Engenharia da Computação. O termo **imitação** é usado aqui para descrever a tendência de muitos alunos de tentarem replicar soluções sem compreender os princípios subjacentes, resultando em uma superficialidade no aprendizado(4).

Mesmo que o aprendizado por imitação seja eficaz para transmitir rapidamente técnicas e algoritmos já estabelecidos, especialmente em ambientes complexos onde a programação manual é inviável [Osa2018An], essa abordagem não apenas limita a capacidade dos alunos de resolver problemas complexos, mas também impede o desenvolvimento do **Pensamento Computacional**, uma habilidade essencial no século XXI (7).

A deficiência pedagógica provocada pela pedagogia da imitação é frequentemente exacerbada pela falta de uma metodologia estruturada que guie os alunos na construção de um entendimento profundo e duradouro dos conceitos fundamentais (4). Acrescente-se a este cenário em que o problema da imitação em detrimento da compreensão profunda está sendo substancialmente agravado com a crescente integração da Inteligência Artificial no processo de ensino-aprendizagem (9).

Ao adotar sistemas de IA que priorizam a reprodução de padrões e soluções já existentes, corre-se o risco de perpetuar uma pedagogia da imitação, dificultando ainda mais a construção do raciocínio algorítmico autêntico entre estudantes e profissionais (11). Isso pode resultar em uma geração menos preparada para criar, adaptar ou inovar algoritmos diante de novos desafios.

Observa-se que a ubiquidade das ferramentas de Inteligência Artificial Generativa, em particular dos **L**arge **L**anguage **M**odels[1], **LLM**s, instaura um novo conjunto de desafios cognitivos e pedagógicos que exacerbam essa dinâmica (15).

Este estudo propõe a implementação de uma disciplina introdutória de **Raciocínio Algorítmico** para cursos de Ciência e Engenharia da Computação, fundamentada em uma Metodologia **DAAD** (**D**ecomposição, **A**bstração, **A**lgoritmização e **D**epuração). Suportada pelos conceitos que levam ao desenvolvimento do **Pensamento Computacional**. Para tanto, além das definições e conceitos fundamentais, o trabalho explora a integração crescente do

[1] em tradução livre, **Modelos de Linguagem de Grande Escala**. Esses modelos são capazes de gerar texto coerente e relevante com base em prompts fornecidos pelos usuários, o que pode levar a uma dependência excessiva dessas ferramentas para resolver problemas algorítmicos sem o devido entendimento dos princípios subjacentes.

Pensamento Computacional e **Raciocínio Algorítmico** em currículos de universidades nos EUA, Reino Unido, Europa, destacando uma mudança do foco exclusivo na programação para uma compreensão mais ampla dos princípios computacionais.

Para a prova de conceito, foi criada uma estrutura curricular detalhada para uma disciplina chamada de **Raciocínio Algorítmico** com 80 horas. Este projeto foi criado com a Metodologia **DAAD**, a experiência de 20 anos de aulas do autor em Universidades da Cidade de Curitiba e uma pesquisa bibliográfica realizada nas universidades dos EUA, UK e Europa. A disciplina está dividida em 4 módulos de ensino com conteúdos e exercícios que progridem do conceitual ao prático, enfatizando a aplicação de cada componente **DAAD** em todos os módulos da disciplina na forma de um *framework* composto de exercícios, estratégias pedagógicas, sugestões de avaliações, rubricas e objetivos. Além disso, o estudo sugere que essa metodologia pode ser generalizada para outras disciplinas de graduação em ciências, matemática e engenharia. Neste *framework* as estratégias pedagógicas incluem aprendizagem ativa, atividades *unplugged* e *plugged*, e oportunidades para o fomento da colaboração.

Além do *framework* pedagógico, inédito, este estudo também propõe uma técnica de aplicação de exercícios, também inédito, que usa a aplicação e repetição de conceitos para reforçar o aprendizado e a compreensão dos alunos. Esta técnica, baseado na Sequência de Fibonacci, projetado para ser flexível e adaptável, permitindo que os alunos pratiquem e apliquem os conceitos de **Raciocínio Algorítmico** em diferentes contextos e níveis de complexidade. O formato da **T**écnica da **S**equência de **F**ibonacci, **TSF** se baseia nos estudos de (19).

Finalmente, complementando o *framwork* e a técnica de aplicação de exercícios, **TSF**, aqui propostos, este estudo também apresenta uma estrutura de **D**ecomposição para a divisão de tarefas complexas entre algunos de um grupo de trabalho, visando o ensino de técnicas de divisão de projetos e tarefas complexas em partes menores e mais gerenciáveis, para fomentar as competências necessárias ao gerenciamento de projetos, trabalho em grupo e colaboração.

Este documento começa com esta singela introdução, seguida por uma seção que apresenta os conceitos fundamentais do **Raciocínio Algorítmico** e do **Pensamento Computacional**. Em seguida, uma análise sobre a aplicação do ensino de **Raciocínio Algorítmico** em universidades dos EUA, Reino Unido e Europa que precede uma seção sobre a fundamentação teórica da Metodologia **DAAD**, específica para este estudo. A seguir, é apresentada a estrutura curricular da disciplina de **Raciocínio Algorítmico**, incluindo os módulos de ensino, objetivos, conteúdos e exercícios. O documento conclui com uma discussão sobre as implicações pedagógicas e sugestões para futuras pesquisas.

Parte I

Contextualização e Pesquisa

2 Referencial Teórico: Pensamento Computacional, Raciocínio Algorítmico e DAAD

Este estudo apresenta uma análise sobre a criação de uma disciplina de **Raciocínio Algorítmico** como prova de conceito para o uso da Metodologia **DAAD** (**D**ecomposição, **A**bstração, **A**lgoritmização e **D**epuração)[1] por professores de graduação em cursos de Ciência e Engenharia da Computação seguindo uma proposta da Universidade de Cambridge (20). Tal disciplina deverá ser introdutória, primeiro período destes cursos visando criar a estrutura cognitiva necessária ao entendimento da computação e sua aplicação para resolução de problemas. Contudo, o estudo ressalta que a Metodologia **DAAD** pode ser aplicada a qualquer disciplina de qualquer curso, e que o **Pensamento Computacional** é uma habilidade essencial para o século XXI. Tal esforço se mostra necessário devido à crescente importância do **Pensamento Computacional** e, consequentemente, do **Raciocínio Algorítmico**, além da popularização do acesso a ferramentas de Inteligência Artificial (21) (15) (22).

A motivação para a proposta de criação desta metodologia surgiu da identificação de lacunas no conhecimento dos alunos de Ciência e Engenharia da Computação, percebidas no estudo dos resultados das últimas avaliações do **E**xame **N**acional de **D**esempenho dos **E**studantes, **ENADE**, dos cursos de Ciência da Computação e Engenharia da Computação. Olhando as últimas edições destes exames e considerando apenas as médias das questões discursivas do componente de conhecimento específico com médias de 9,8 e 16,9 de 100, respectivamente (24).

Uma análise mais aprofundada revela que a dificuldade central no ensino da programação não reside na memorização da sintaxe, mas sim no desenvolvimento da capacidade de abstração (25). A sintaxe é apenas a manifestação superficial de um um desafio mais profundo para construir modelos mentais de processos computacionais (26). A programação exige que os alunos traduzam um problema do mundo real numa sequência de passos inequívocos que uma máquina possa executar. Este processo de tradução é, na sua essência, um ato de abstração (27): identificar os elementos essenciais do problema, ignorar os detalhes supérfluos e modelar o processo. Muitos alunos fracassam não porque não conseguem memorizar a sintaxe de um ciclo for, mas porque não conseguem abstrair o conceito de iteração a partir de um problema concreto (29). Além disso, parece haver algumas perspectivas conflitantes sobre quais habilidades cognitivas permitem o desenvolvimento eficaz de algoritmos. Principalmente, porque esta é uma questão em aberto debate na comunidade acadêmica.

2.1 A Jornada Inacabada: O Problema do Ensino do Raciocínio Algorítmico

As críticas ao ensino de **Raciocínio Algorítmico** aparecem de forma relevante na academia a partir das últimas décadas do século XX. Os estudos de Lithner (2008)(30) destacaram que o ensino tradicional prioriza **Raciocínio Algorítmico Imitativo**, no qual alunos reproduzem

[1] Algoritmização é um neologismo para traduzir o termo "Algorithmization" do inglês. A ideia é que seja um termo que remete à criação de algoritmos, ou seja, a construção de soluções sistemáticas e eficientes para problemas computacionais. É horrível, eu sei.

procedimentos memorizados, em inglês *Familiar Algorithmic Reasoning*[2], em detrimento do **Raciocínio Matemático Criativo**. Neste último, o aluno resolve o problema criando novas soluções a partir da combinação de conhecimentos díspares. Essa abordagem suprime a capacidade de decompor problemas e construir soluções originais, perpetuando uma cultura de superficialidade cognitiva (31). Os trabalhos de Lithner (2008)(30) e Harisman (2023) indicam que o ensino por imitação é parte significativa do problema. Mas, não é a única.

Pesquisas em educação matemática indicam que o ensino baseado em **Raciocínio Algorítmico Imitativo** pode comprometer a originalidade dos estudantes na solução de problemas novos. Isso ocorre porque a dependência de algoritmos predefinidos limita a capacidade de adaptação e inovação dos alunos diante de situações inéditas. Conforme Hurrell (2021)(32), o conhecimento procedimental, *como fazer*, caracterizado por sequências fixas de ações repetidas, não garante a compreensão necessária para gerar novas estratégias ou adaptar as conhecidas a novos problemas, o que reduz a originalidade na resolução.

Além disso, o *National Council of Teachers of Mathematics*, em seus Princípios e Padrões para a Matemática Escolar enfatiza que a aprendizagem matemática deve ir além do domínio de procedimentos algorítmicos e procedurais, promovendo a compreensão conceitual e a flexibilidade cognitiva(33). O documento alerta que a dependência excessiva de algoritmos e regras fixas pode inibir a transferência de conhecimentos para contextos inéditos e a capacidade de resolver problemas de forma criativa e inovadora (33).

Estudos empíricos, ainda mais antigos, indicam que estudantes treinados predominantemente com técnicas algorítmicas apresentam até 32% menos originalidade na solução de problemas novos, em comparação com aqueles que desenvolvem compreensão conceitual e estratégias flexíveis (5).

Finalmente, existem avaliações sobre a capacidade e eficiência da compreensão. O livro *Programming: The Derivation of Algorithms* (34) expôs como o ensino imperativo tradicional foca na sintaxe de laços de repetição e invariantes, usando a lógica de Hoare, negligenciando a construção de entendimento conceituais fundamentais. Essa metodologia foi criticada por substituir o "porquê" pelo "como", limitando o **Pensamento Computacional** profundo (35). A Table 2.1 resume as principais críticas ao ensino tradicional de **Raciocínio Algorítmico**.

Table 2.1: Estudos e possíveis razões por trás dos problemas persistentes no ensino de **Raciocínio Algorítmico**.

Fator	Impacto Histórico	Evidência Atual
Currículos Baseados em Eficiência	Priorização de otimização (ex: complexidade O(n)) sobre metacognição.	Algoritmos como *QuickSort* são ensinados como fórmulas pré-definidas, sem discussão ou cognição (35).
Falta de Formação Docente	Professores reproduzem métodos tradicionais por falta de treino em pedagogia criativa.	Estudos mostram que educadores não dominam técnicas para mitigar dependência de Inteligência Artificial (36)
Infraestrutura Cognitiva	Modelos mentais baseados em **Raciocínio Algorítmico Imitativo** dificultam a transição para **Raciocínio Matemático Criativo**.	Meta-análises confirmam baixa transferência de conceitos algorítmicos entre domínios (37)

[2]em tradução livre Raciocínio Algorítmico Familiar, ou Raciocínio Algorítmico Conhecido.

Um problema mais recente para a criação das competências que levem ao **Raciocínio Algorítmico** e ao **Pensamento Computacional** parece estar relacionado com a Inteligência Artificial(22).

O interesse na pesquisa de soluções de Inteligência Artificial, que segundo a Figure 2.1 se acentuou a partir de 2010.

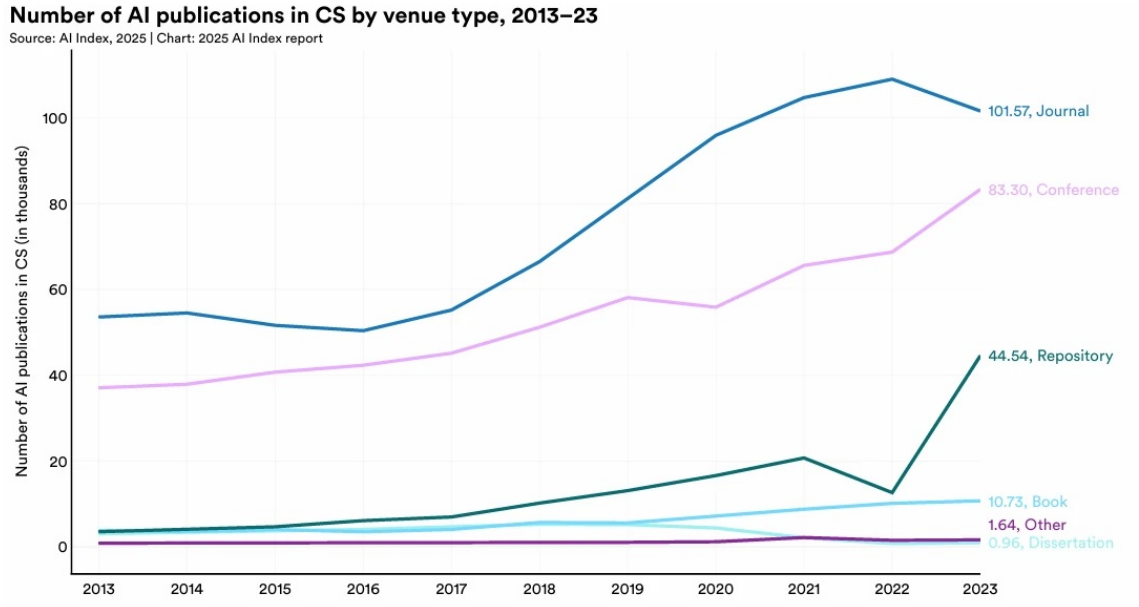


Figure 2.1: Gráfico mostrando a evolução de publicações sobre Inteligência Artificial por tipo de publicação (21).

Este crescimento teve impacto na disponibilidade destas tecnologias, o que pode ser corroborado se considerarmos o aumento do uso de dispositivos contendo soluções embarcadas de Inteligência Artificial como pode ser visto na Figure 2.2.

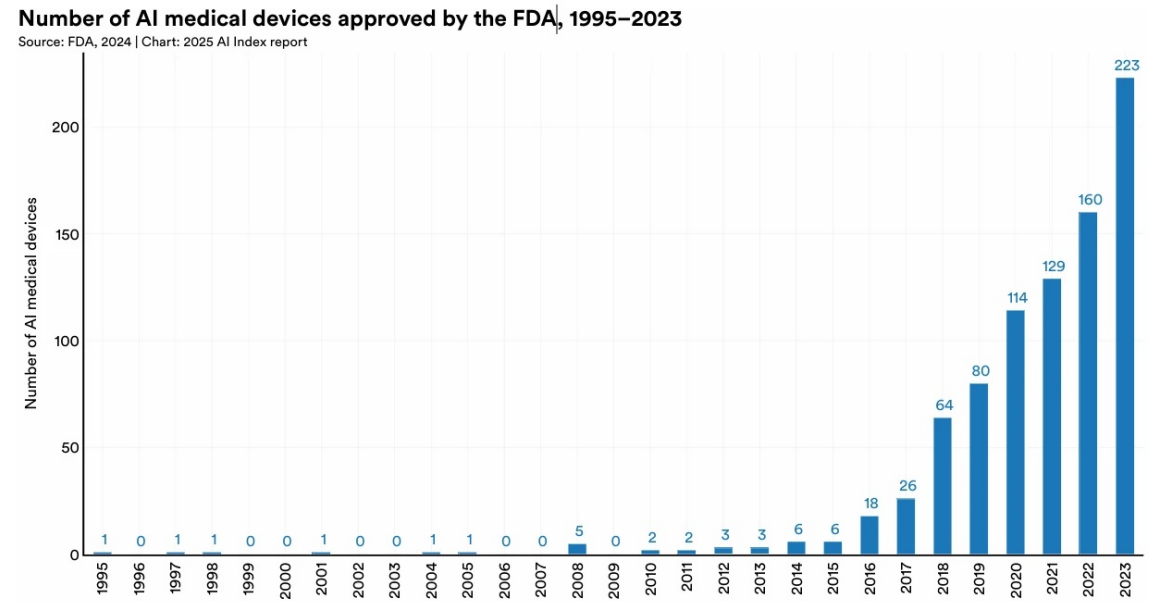


Figure 2.2: Gráfico mostrando a evolução do número de dispositivos médicos contendo Inteligência Artificial embarcada segundo os dados da (21).

Esta velocidade de adaptação parece ter impactos negativos no binômio ensino-aprendizagem de forma geral em todos os cursos e formações.

Um estudo recente conduzido pelo MIT Media Lab, intitulado *Your Brain on ChatGPT: Accumulation of Cognitive Debt when Using an AI Assistant for Essay Writing Task*[3], publicado em junho de 2025, investigou os impactos do uso de modelos de linguagem grandes (LLMs), como o ChatGPT, na cognição humana, especificamente no contexto da escrita de ensaios(12). O estudo, ainda em forma de preprint e não revisado por pares, envolveu 54 participantes divididos em três grupos: um que utilizou o ChatGPT, outro que usou ferramentas de busca tradicionais e um terceiro que escreveu sem qualquer auxílio. Os resultados indicaram que os participantes que utilizaram o ChatGPT apresentaram menor atividade cerebral, menor retenção de memória e menor originalidade em seus escritos em comparação com os outros grupos(12). Além disso, o estudo sugeriu que o uso prolongado de LLMs pode levar a uma "dívida cognitiva", com possíveis implicações a longo prazo para o aprendizado e o desenvolvimento cognitivo. No entanto, devido às limitações do estudo, como o tamanho da amostra e o foco específico no ChatGPT, os achados devem ser interpretados com cautela, e mais pesquisas são necessárias para generalizar os resultados (12).

O estudo de SILVA QUINTO, W. A. et al. (38) investiga o impacto da Inteligência Artificial no desenvolvimento do pensamento crítico entre estudantes de Tecnologia da Informação na Região Norte do Brasil. Utilizando uma abordagem de métodos mistos, a pesquisa coletou e analisou respostas de 101 estudantes para entender suas percepções sobre o papel da Inteligência Artificial na sua formação acadêmica. Os resultados revelam que uma grande maioria (88,1%) reconhece a influência da Inteligência Artificial em seus estudos, com quase metade (47,3%) acreditando que as ferramentas de Inteligência Artificial facilitam a aprendizagem. No entanto, uma menor porção (13,2%) expressa preocupações de que a Inteligência Artificial pode impedir o desenvolvimento do pensamento crítico(38).

O artigo *Teaching AI with games: the impact of generative AI drawing on computational thinking skills*[4], publicado em 2025, investiga o impacto do uso de ferramentas generativas de Inteligência Artificial, como ferramentas generativas para desenho, no desenvolvimento de habilidades de **Pensamento Computacional** em 56 alunos do sexto ano de escolas no norte de Taiwan(13). Divididos em dois grupos: o experimental usando estas ferramentas com Inteligência Artificial embarcada e programação baseada em blocos para criar jogos; e o grupo de controle, usando ferramentas de busca disponíveis na internet. Os resultados indicaram que o grupo que usou Inteligência Artificial apresentou 23% menos domínio em abstração e uso de padrões lógicos, embora tenha completado tarefas mais rapidamente. A pesquisa sugere que a Inteligência Artificial (IA) acelera a execução, mas reduz a cognição lógica(22). Ainda que os autores tenham usado o termo **Pensamento Algorítmico** como sinônimo do conceito que neste estudo será representado, muitas vezes, por uma de suas partes o **Raciocínio Algorítmico**. É preciso ressaltar que o artigo destaca a necessidade de integrar essas ferramentas de forma equilibrada para promover habilidades que estão intimamente relacionadas com o **Pensamento Computacional**.

Nem tudo são avaliações negativas; ao longo do tempo foram realizadas tentativas de resolver os problemas do ensino de **Raciocínio Algorítmico**. Duas abordagens que se destacam: a *Constructive Algorithmics*, que em português poderia ser traduzida como algorítmica construtiva, e a Computação *unplugged*.

A abordagem *Constructive Algorithmics*, desenvolvida por Richard Bird e Oege de Moor em 1997@BirdDeMoor1997, fundamenta-se no uso de **raciocínio equacional** e princípios da álgebra de programas por meio de uma abordagem matemática para a construção de programas de computador, tratando a programação como uma disciplina de engenharia e não

[3] em tradução livre "Seu Cérebro no ChatGPT: Acúmulo de Dívida Cognitiva ao Usar um Assistente de Inteligência Artificial para Tarefa de Escrita de Ensaio".

[4] em tradução livre "Ensinando Inteligência Artificial com jogos: o impacto da Inteligência Artificial generativa no desenvolvimento de habilidades de pensamento computacional".

como uma arte baseada em tentativa e erro(36). Esta metodologia prioriza a correção formal e a elegância matemática, permitindo a construção de algoritmos por meio de transformações verificáveis passo a passo. Um exemplo emblemático é a derivação de algoritmos de ordenação, como *quicksort*, via composição funcional, onde propriedades matemáticas garantem robustez lógica. Jeremy Gibbons 2020@GibbonsHaskell2020 aplicou esses mesmos princípios em *Algorithm Design with Haskell*[5].

A computação *unplugged* emerge como contraponto pedagógico, utilizando atividades manuais, como jogos de tabuleiro para decomposição de problemas e atividades com lápis e papel, para desenvolver bases conceituais do pensamento computacional. Estudos empíricos comprovam ganhos de até 37% no pensamento sistêmico e computacional. Os alunos internalizam conceitos abstratos, recursão, paralelismo, por meio de manipulação física e erro reflexivo (39). Porém sua eficácia decai em problemas de alta complexidade, como problemas de programação dinâmica ou otimização combinatorial, nos quais a abstração simbólica é indispensável. Por outro lado, estudos comparativos, como *Unplugged versus plugged-in: examining basic programming achievement and computational thinking of 6th-grade students*[6] (40), demonstram que a falta de transição para ferramentas digitais limita a aplicação prática: alunos dominam puzzles com blocos, mas falham em traduzir lógica para código em problemas do mundo real, expondo uma fratura escalar no modelo (41).

A Table 2.2 resume as principais diferenças entre as abordagens *Constructive Algorithmics* e Computação *unplugged*.

Table 2.2: Comparação entre as abordagens *Constructive Algorithmics* e Computação *unplugged*.

Abordagem	**Descrição**	**Pontos Fortes**	**Limitações**
Algorítmica Construtiva	Usa raciocínio equacional e álgebra de programas para derivar algoritmos em linguagens funcionais como Haskell (42).	Rigor matemático, correção formal, elegância na derivação de algoritmos (ex.: quicksort).	Descontextualização ética, negligencia impactos sociais.
Computação *Unplugged*	Utiliza atividades manuais (ex.: jogos de tabuleiro) para ensinar **Pensamento Computacional**(40).	Ganhos de até 37% no pensamento sistêmico, internalização intuitiva de conceitos (recursão, paralelismo).	Ineficaz em problemas complexos (ex.: programação dinâmica), limitada transição para ferramentas digitais.

Da ambiguidade, surge a integração híbrida como caminho promissor: iniciam-se com atividades não digitais, ou eletrônicas, para fundamentos e migra-se para *Constructive Algorithmics* com extensões que incluem análise como análise de vieses em algoritmos reais. Bird e de Moor em 1997@BirdDeMoor1997 já forneceram base para isso: seu método equacional pode ser estendido enquanto Gibbons (2020)(42) oferece ferramentas para essa transição ao usar Haskell em exemplos aplicados.

Das pesquisas citadas é possível inferir que o ensino de **Raciocínio Algorítmico** e **Pensamento Computacional** permanece irresoluto porque supervaloriza eficiência em detrimento

[5]em tradução livre "Projeto de Algoritmos com Haskell".

[6]em tradução livre Desplugado versus Plugado: examinando o desempenho em programação básica e o**Pensamento Computacional**de alunos do 6º ano

da análise: nas décadas de 1980 a 2000, pedagogias formaram gerações capazes de implementar *BubbleSort*, mas não de questionar *por que usá-lo* (42). Hoje, sistemas baseados em Inteligência Artificial agravam a superficialidade cognitiva (22).

Hoje, em julho de 2025, o cenário educacional ainda carece de uma abordagem unificada e estruturada para o ensino do **Pensamento Computacional** e do **Raciocínio Algorítmico**. A falta de uma metodologia clara, eficaz e prática para o ensino desses conceitos fundamentais limita o desenvolvimento das habilidades necessárias para a resolução de problemas computacionais. Dessa forma, este estudo propõe a criação de uma disciplina introdutória que utiliza a Metodologia **DAAD** como base para o desenvolvimento dessas competências, visando preencher essa lacuna no ensino superior. Uma disciplina que poderá ser chamada de **Raciocínio Algorítmico**.

2.2 Definições Fundamentais

O **Pensamento Computacional** é uma estrutura que descreve um conjunto de habilidades de pensamento crítico e resolução de problemas, que tem ganhado significativa relevância como uma forma eficaz de ensinar essas habilidades em ambientes educacionais formais (43). Embora não seja a única abordagem para desenvolver essas competências, o **Pensamento Computacional** oferece uma maneira de analisar problemas para gerar soluções automatizadas ou semi-automatizadas que utilizam as capacidades únicas das tecnologias computacionais (6).

É necessário considerar que o termo **Pensamento Computacional** parece ter sido criado por Papert (1985)(44), mas foi popularizado por Jeannette Wing, em 2006@Wing2006, argumentando que o **Pensamento Computacional** deveria ser considerado uma habilidade indispensável para todos, comparável à leitura, escrita e aritmética. Conceito no qual ela é apoiada por Saidin (2021)(45) e pela *Computer Science Teachers Association*@CSTA2011a. O **Pensamento Computacional** é reconhecido globalmente como uma das habilidades vitais para o século XXI, essencial para a resolução de problemas complexos em diversas áreas (46). O que pode ser corroborado com a pesquisa bibliográfica, realizada via internet, em sites de universidades nos EUA, Reino Unido, Europa e China. Esta pesquisa demonstrou uma tendência crescente na integração do **Pensamento Computacional** nos currículos de graduação, com ênfase em abordagens práticas e baseadas em projetos cujas características veremos na Chapter 3.

Jeannette Wing define o **Pensamento Computacional** como *"os processos de pensamento envolvidos na formulação de problemas e suas soluções de forma que as soluções possam ser efetivamente executadas por um agente de processamento de informações"* (28). Ela também o descreve como a resolução de problemas, o design de sistemas e a compreensão do comportamento humano, baseando-se em conceitos estruturantes da ciência da computação (14). O **Pensamento Computacional** é uma combinação de hábitos mentais disciplinados, atitudes de perseverança e habilidades interpessoais essenciais (29). Ele se distingue de outras abordagens de resolução de problemas, como o **Pensamento Científico** ou o *design thinking*, principalmente por seu foco em soluções algorítmicas e no desenvolvimento de sistemas que envolvem processamento de informações (47).

De acordo com Wing (2006) (28), o **Pensamento Computacional** estrutura-se em componentes fundamentais que transcendem a mera programação: a **decomposição**, fragmentação de problemas complexos em partes gerenciáveis, o **reconhecimento de padrões**, identificação de similaridades e tendências em diferentes contextos, a **abstração**, seleção de elementos essenciais e eliminação de detalhes irrelevantes, e o **design algorítmico**, criação de instruções passo a passo para solução automatizada, complementados por processos de **generalização**, aplicação de soluções a problemas análogos e **modelagem**, representação de sistemas do mundo real através de estruturas computacionais, formando uma abordagem

holística para a resolução eficiente de problemas (28).

As características e a definição de **Pensamento Computacional** criam o cenário necessário ao entendimento do problema do ensino de **Raciocínio Algorítmico**.

Em essência, o raciocínio algorítmico pode ser definido como uma forma de pensamento lógico e organizado utilizada para desmembrar um objetivo complexo em uma série de etapas ordenadas, empregando as ferramentas disponíveis (29). Um método de pensamento que guia processos de raciocínio por meio de procedimentos passo a passo, que exigem entradas e produzem saídas, demandando decisões sobre a qualidade e adequação das informações, e monitorando os próprios processos de pensamento para controlá-los e dirigi-los (29). É, portanto, simultaneamente um método de pensar e um meio para pensar sobre o próprio pensamento (29).

Este constructo envolve um conjunto de habilidades cognitivas cruciais para a construção, compreensão e avaliação de algoritmos muito similares às habilidades necessárias para o **Pensamento Computacional**. As principais habilidades cognitivas que compõem o **Raciocínio Algorítmico** incluem (29):

- **Decomposição**: a capacidade de fragmentar um problema ou sistema complexo em partes menores ou sub-objetivos mais gerenciáveis (29).
- **Abstração**: habilidade de identificar os componentes essenciais de um problema ou sistema, o que implica coletar informações relevantes e descartar as irrelevantes, construindo representações que demonstrem como o problema ou sistema funciona (29).
- **Algoritmização** (ou Design Algorítmico): o desenvolvimento de uma sequência de passos lógicos e ordenados para resolver um problema ou alcançar um objetivo (29). As regras que especificam o algoritmo devem ser precisas e determinadas (29).
- **Compreensão, Análise e Especificação Precisa de Problemas**: inclui a formulação de problemas e o desenvolvimento de uma sequência de passos composta por ações básicas que os resolverão, considerando casos simples e complexos, normais e especiais (48).
- **Avaliação e Otimização**: a capacidade de avaliar, aprimorar ou otimizar a eficiência de um algoritmo, considerando abordagens alternativas (48).

Historicamente, o termo **Raciocínio Algorítmico** antecedeu e se fundiu com o **Pensamento Computacional** (49). Alan Perlis, na década de 1960, cunhou o termo *algoritmizar* para descrever uma prática e um pensamento que levariam à automação de procedimentos em todas as indústrias (49). Edsger Dijkstra, um dos pilares da Ciência da Computação, delineou características do raciocínio algorítmico, como a capacidade de transitar entre níveis semânticos e de utilizar a língua materna como ponte entre problemas informais e soluções formais (49).

No contexto do **Pensamento Computacional**, a automação é um elemento indispensável para caracterizar a essência do raciocínio algorítmico, distinguindo-o de outras formas de pensamento, como o matemático, ao focar na implementação de instruções em dispositivos digitais por meio de programação (7). Este é, portanto, um pilar que permite a concepção de soluções que exploram as capacidades únicas das tecnologias computacionais.

Em sua pesquisa, Ribeiro, L. et al (2017)(50) diferenciam **raciocínio lógico**, fundamentado em premissas e inferências, do **raciocínio computacional**, focado em *abstração*, *automação* e *análise*, enquanto fazem avaliações sobre a redução do **Pensamento Computacional** à criação de algoritmos, ressaltando que o **Raciocínio Algorítmico** é apenas um dos pilares do **Pensamento Computacional**, não sua totalidade. KONG, S. et al.@Kong2019 destacam que o **Raciocínio Algorítmico** é frequentemente confundido com o **Pensamento Computacional** por ser seu elemento mais tangível, especialmente em contextos de programação. Mais tarde, estes mesmos autores alertam que essa visão reduz o **Pensamento Computacional** a habilidades técnicas, negligenciando pilares como a decomposição de

problemas e o pensamento analítico(51). O estudo de MEDEIROS (2024)(52) analisou 38 trabalhos brasileiros sobre **Pensamento Computacional** (2018–2024) e identificou que 76% deles equiparavam **Pensamento Computacional** ao ensino de algoritmos, via Scratch ((53)) ou aplicações de robótica. Concluindo que a ênfase excessiva no **Raciocínio Algorítmico** subestima outros pilares do **Pensamento Computacional**, tais como pensamento algébrico e resolução criativa de problemas (54).

Uma questão interessante surge do trabalho de Hora (2022)(55) que destaca que no Brasil o **Pensamento Computacional** é frequentemente restrito a disciplinas de exatas, Matemática e Computação, nas quais o **Raciocínio Algorítmico** domina as práticas pedagógicas, fazendo análises sobre o viés tecnicista que ignora dimensões como abstração contextualizada e análise sociotécnica. Destacando a abrangência do **Pensamento Computacional**. Dessa forma Hora (2022)(55) sustenta os trabalhos de Wing (2006) (28) e (29). Além disso, Felipussi e Padua (2023)(56) perceberam que os professores tendem a associar **Pensamento Computacional** diretamente à programação de robôs, tratando algoritmo como sinônimo de **Pensamento Computacional**.

A própria Sociedade Brasileira de Computação (2017)(54) adverte que a **BNCC** (**B**ase **N**acional **C**omum **C**urricular) não distingue claramente **Raciocínio Algorítmico** e **Pensamento Computacional**, gerando ambiguidade em propostas pedagógicas e recomenda equilibrar o ensino de algoritmos com atividades de **decomposição de problemas não lineares**.

Finalmente, existem trabalhos que indicam que a ambiguidade entre **Pensamento Computacional** e **Raciocínio Algorítmico** pode ter consequências negativas na formação do indivíduo.

Do trabalho de Medeiros (2024)(52) podemos inferir que os estudantes desenvolvem habilidades técnicas (ex.: codificar em blocos) mas falham em aplicar **Pensamento Computacional** em contextos interdisciplinares. As humanidades e artes raramente integram **Pensamento Computacional** em seus currículos, pois o **Raciocínio Algorítmico** é visto como não aplicável (57). Por fim, as avaliações de **Pensamento Computacional** focam em correção algorítmica e eficiência do código, não em processos mentais como abstração (58).

2.3 Primeiro Contato com a Metodologia DAAD

A Metodologia **DAAD** (**D**ecomposição, **A**bstração, **A**lgoritmização e **D**epuração), aqui proposta, estende uma discussão da Cambridge Mathmatics (20) e a análise feita por LEHMANN (2024)(48), sob os efeitos da algoritmização. O que este estudo propõe difere dos estudos anteriores por apresentar um *framework* prático iterativo de ações, exercícios e temas de estudo com o intuito de permitir a criação de **Pensamento Computacional** e **Raciocínio Algorítmico** em alunos de graduação, baseado nos estudos teóricos e empíricos de Wing (2006)(28), Lehmann (2023, 2024)(29),(48), Kirwan (2022)(59), KONG, Siu-Cheung et al. (2019)(58) e muitos outros que estão citados ao longo deste documento, sobre temas que abordam o **Raciocínio Algorítmico** e o **Pensamento Computacional**. Para tanto, este *framework* está estruturado em quatro estágios iterativos:

1. **Decomposição (D)**: processo de fragmentação de problemas complexos em subproblemas gerenciáveis, seguindo princípios de divisão funcional ou estrutural. Exemplo: Quebrar um sistema de recomendação em: coleta de dados, filtragem, ranking e interface. Wing (2006)(28), SBC (2017)(54) e Lehmann (2024)(48) estabelecem decomposição como pilar cognitivo essencial.

2. **Abstração (A)**: identificação seletiva de padrões e invariantes essenciais, com descarte consciente de detalhes irrelevantes ao núcleo do problema. Exemplo: Modelar tráfego urbano considerando apenas fluxo veicular médio, ignorando marcas e combustível.

Baseia-se no conceito de "abstração progressiva" perceptível na Linguagem LOGO de Papert (1985)(44).

3. **Algoritmização (A)**: formulação de soluções por meio de sequências lógico-operacionais, garantindo completude e implementabilidade. Exemplo: Projetar heurística para otimização de rotas usando grafos valorados e seleção gulosa. Alinha-se ao paradigma de descrição de estados finais. A essência do projeto algorítmico reside na descrição formal de estados finais, onde a corretude é verificada pelo atendimento inequívoco de pós-condições (36). Incluímos aqui, os conceitos:

    - **Algoritmização** (ou Design Algorítmico): o desenvolvimento de uma sequência de passos lógicos e ordenados para resolver um problema ou alcançar um objetivo (29). As regras que especificam o algoritmo devem ser precisas e determinadas (29).
    - **Compreensão, Análise e Especificação Precisa de Problemas**: inclui a formulação de problemas e o desenvolvimento de uma sequência de passos composta por ações básicas que os resolverão, considerando casos simples e complexos, normais e especiais (48).

Notadamente porque o segundo é parte integrante do primeiro, e ambos são fundamentais para a construção de soluções computacionais eficazes. A **Algoritmização** é o processo de criação de algoritmos, enquanto a **Compreensão, Análise e Especificação Precisa de Problemas** envolve a formulação e decomposição de problemas em etapas lógicas que podem ser resolvidas por meio de algoritmos.

4. **Depuração (D)**: refinamento iterativo mediante validação empírica, incluindo testes de robustez, análise de falhas e otimização pós-implementação. Exemplo: Injeção de dados corrompidos em *pipelines* de processamento para validar resiliência. Incorpora princípios de "engenharia resiliente" de Leveson (2012) (61). Ou seja, a **Depuração** abrange os conceitos relativos a **Avaliação e Otimização**: a capacidade de avaliar, aprimorar ou otimizar a eficiência de um algoritmo, considerando abordagens alternativas (48).

Em um ambiente no qual existem ambiguidades e avaliações sobre a formação fundamental dos alunos de graduação, a Metodologia **DAAD** foca prioritariamente nos processos matemáticos e lógicos envolvidos na criação e depuração de soluções computacionais como ferramentas indispensáveis para a solução de problemas.

A busca bibliográfica realizada para este estudo indicou que a Metodologia **DAAD** como está proposta parece ser a única ferramenta disponível que oferece uma estrutura prática iterativa para o ensino do **Raciocínio Algorítmico**, abordando tanto a decomposição de problemas quanto a abstração necessária para a criação de algoritmos eficazes e, principalmente, depuração. A relação e integração conceitual entre os componentes da Metodologia **DAAD** (**D**ecomposição, **A**bstração, **A**lgoritmização e **D**epuração) está ilustrada na Figure 2.3

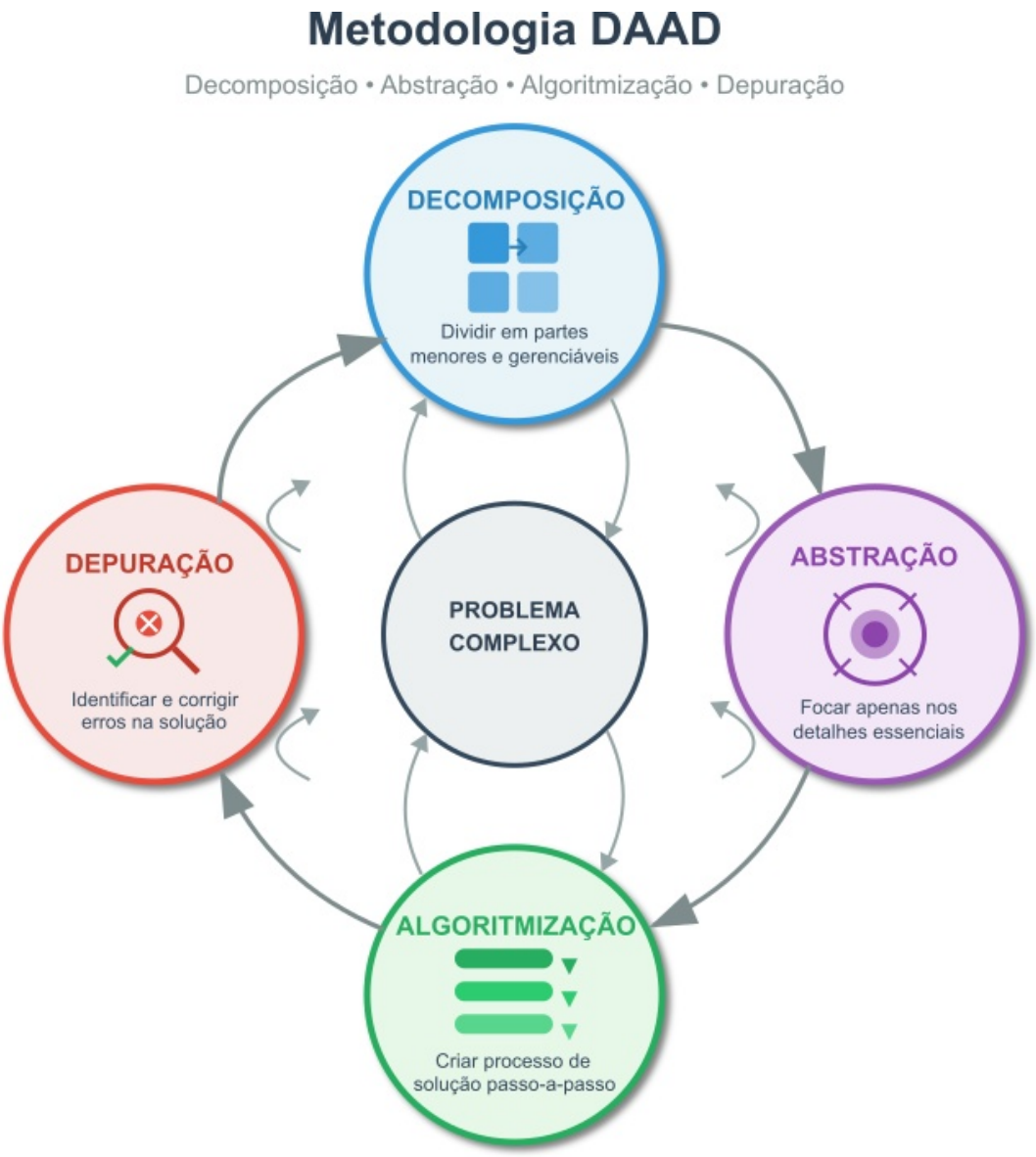


Figure 2.3: Ilustração da inter-relação entre Decomposição, Abstração, Algoritmização e Depuração. Fonte: o autor.

A Metodologia **DAAD** modifica os tradicionais "Quatro Pilares do Pensamento Computacional" de Wing@Wing2006 ao substituir "Reconhecimento de Padrões" por "Depuração", concordando com Lehmann@Lehmann2024 e com a Cambridge Mathematics@CambridgeMaths. Essa mudança adequa o **Pensamento Computacional** ao **Raciocínio Algorítmico**, enfatizando o processo ativo de criação de algoritmos e a habilidade essencial de detecção e correção de erros, a depuração. Criando uma metodologia iterativa adequada à solução de problemas em um ciclo de entendimento, abstração, construção da solução e depuração.

2.4 Estratégias Pedagógicas e Materiais Didáticos para o Ensino de DAAD

O ensino eficaz dos componentes do **Raciocínio Algorítmico** usando a Metodologia **DAAD**, **D**ecomposição, **A**bstração, **A**lgoritmização, **D**ebugging, exige estratégias pedagógicas que transcendem as aulas expositivas tradicionais, promovendo o engajamento ativo e a aplicação prática. E deve superar o Jogo da Imitação 1 como metodologia de ensino. O que está proposto neste estudo é uma abordagem progressiva e integrada entre atividades *unplugged* ou desconectadas, *plugged* ou conectadas e o estudo independente para desenvolver o **Pensamento Computacional**.

Iniciar cada tema possível com atividades *unplugged* permite que os alunos construam uma compreensão conceitual sólida sem a barreira da sintaxe de programação (63). Posteriormente, essas compreensões conceituais são reforçadas e aplicadas por meio de desafios de programação *plugged* (63). Essa progressão didática pretende que os alunos compreendam os porquês antes de se aprofundarem nos comos, otimizando o processo e os resultados do aprendizado(32).

A **Metodologia *Flipped Classroom*** (64), aplicada pela ETH Zurich (65), é uma estratégia pedagógica que inverte a dinâmica tradicional de ensino. A metodologia *Flipped Classroom* utiliza a expressão em inglês para sala de aula invertida para representar uma abordagem pedagógica que reverte a sequência do processo de ensino-aprendizagem. Nas últimas décadas, a introdução de novos conceitos passou a ocorrer em sala de aula majoritariamente, seguida por atividades de reforço em casa o que Bergmann (2012)(64) considera ineficiente. Segundo a metodologia *Flipped Classroom*, o estudo inicial do conteúdo deve ser realizado pelos alunos fora do ambiente de aula, por meio de materiais previamente fornecidos, de forma independente. Esse preparo prévio permite que o tempo em sala de aula seja otimizado para a prática, discussão e resolução de problemas, com o professor atuando como um orientador(64).

A **Aprendizagem Baseada em Projetos (PBL)** (66) é uma estratégia amplamente utilizada e eficaz para o ensino de **Pensamento Computacional** (67). A metodologia **PBL** envolve os alunos na resolução de problemas complexos e autênticos do mundo real, permitindo-lhes construir conhecimento significativo por meio da investigação, cooperação e prática (58). Essa abordagem fomenta a autonomia do aluno, a criatividade e a capacidade de resolver problemas (67). Exemplos incluem a criação de modelos computacionais sobre furacões usando Scratch (53) ou o design de aplicativos para necessidades sociais (67).

As atividades *unplugged* ensinam o **Pensamento Computacional** sem o uso de dispositivos digitais, utilizando materiais como papel, cartas ou atividades físicas (62). Estas atividades permitem que os alunos se concentrem nos conceitos subjacentes do **Pensamento Computacional** sem se prenderem à sintaxe de uma linguagem de programação (62). Exemplos incluem programar um colega para realizar uma tarefa simples, usar mapas para ilustrar a abstração e a criação de fluxogramas da resolução do problema(7). O uso de atividades *unplugged* antes da programação baseada em computador pode levar a melhores resultados de aprendizagem (7).

A **Aprendizagem Ativa e Colaborativa** é altamente incentivada (38). Discussões em equipe, resolução de problemas em grupo e a comunicação de soluções alternativas são práticas que fomentam o **Pensamento Computacional**. A aprendizagem colaborativa, facilitada por tecnologias simples, pode melhorar as habilidades de resolução de problemas e o engajamento dos alunos(38).

A **Abordagem Heurística e Baseada em Problemas** envolve apresentar problemas desafiadores e guiar os alunos a explorar e inovar, escalando gradualmente a complexidade, o que é indispensável para aprofundar a compreensão do **Pensamento Computacional** (67). A abordagem socrática, na qual os conceitos são desenvolvidos por meio de questionamento dos alunos, também se mostra eficaz (69). Ao serem desafiados com perguntas como "Como

você pode quebrar esse problema em etapas menores?" ou "Que padrões você nota que podem ser úteis para resolver este problema?", os estudantes desenvolvem habilidades de pensamento analítico (69).

No contexto dessas abordagens, o papel do professor se transforma de um mero transmissor de conhecimento para um facilitador, diagnosticador e curador do conhecimento (51). O educador deve atuar como gerador de questionamento, não fornecendo todas as soluções, mas guiando os alunos para que resolvam os problemas por si mesmos (67). A capacidade do professor de diagnosticar dificuldades e oferecer orientação direcionada é determinante para o sucesso do processo de aprendizagem (67). Estas características parecem indicar a necessidade de formação e desenvolvimento profissional para os educadores, capacitando-os não apenas no conhecimento do **Pensamento Computacional**, mas também nas estratégias pedagógicas para a sua implementação eficaz, incluindo a criação de ambientes de aprendizagem ativos e a oferta de retorno avaliativo construtivo.

2.4.1 Materiais Didáticos e Ferramentas

A literatura e as práticas educacionais atuais oferecem uma variedade de recursos didáticos e ferramentas para o ensino do **Pensamento Computacional** e do **Raciocínio Algorítmico**. Estes recursos são fundamentais para a implementação eficaz da Metodologia **DAAD** e incluem:

- **Linguagens de Programação**: Python(71) é frequentemente citada como uma linguagem de programação ideal para o ensino de **Pensamento Computacional**, especialmente em disciplinas introdutórias, devido à sua estrutura direta e facilidade de uso (72). Outras linguagens como OCaml(73), Java(74), C(75), C++(76) e Prolog (77) também são utilizadas em currículos mais avançados (78) (72).

Ambientes de Desenvolvimento e Ferramentas Visuais: Ambientes como Scratch(53) (para crianças e iniciantes), XLogo4Schools(79) para Logo(44) e Python, são empregados para reduzir a carga cognitiva e permitir que o foco recaia nos conceitos, notadamente para crianças e adolescentes (80). Além disso, ferramentas de depuração são essenciais para o aprendizado (67).

2.5 Melhores Práticas e Desafios na Implementação de Metodologias de Resolução de Problemas

A implementação de metodologias de resolução de problemas, como o **Raciocínio Algorítmico**, como parte integrante do **Pensamento Computacional**, no ensino superior no Brasil apresenta complexidades significativas. Um mapeamento sistemático da literatura nacional revelou um crescente interesse em iniciativas que estimulam o **Pensamento Computacional**(43). No entanto, a tradução da relevância teórica em práticas educacionais eficazes e amplamente disseminadas enfrenta desafios. A pesquisa aponta para uma grande diversidade nos instrumentos e artefatos utilizados para a avaliação (43). A discussão sobre a granularidade das avaliações, seja em alto nível desempenho final, ou granularidade fina, habilidades específicas, e a complexidade de sua mensuração eficiente, somada à questão da replicabilidade dos estudos empíricos devido à frequente ausência de detalhes metodológicos, indicam lacunas na padronização e na generalização das práticas avaliativas e de implementação (48). Desse modo, embora o reconhecimento da importância do **Raciocínio Algorítmico** no cenário educacional brasileiro seja evidente, sua plena e homogênea integração ainda é permeada por desafios de ordem prática e metodológica.

2.5.1 Melhores Práticas Identificadas na Literatura

O **Pensamento Computacional** é mais eficaz quando integrado em diversas disciplinas, não apenas nas disciplinas relacionadas a Ciência da Computação (51). Para tanto é necessário a contextualização dos problemas em cenários do mundo real ou em suas próprias pesquisas aumenta o engajamento dos alunos (67). Essa abordagem permite que os alunos apliquem as estruturas cognitivas desenvolvidas à solução de problemas relevantes para suas áreas de estudo, desde a análise de dados históricos até a otimização de processos logísticos(51).

O ensino integral, voltado para a formação do indivíduo presupõe que. além das habilidades técnicas, o ensino de **Pensamento Computacional** deve cultivar atitudes como confiança na resolução de problemas, persistência diante de desafios, tolerância a diferenças e capacidade de colaborar [Huang2020ComputationalThinking.]. A aprendizagem baseada em projetos e atividades *unplugged* são eficazes para fomentar essa mentalidade (62). Além disso, as práticas de **Raciocínio Algorítmico**, especialmente a depuração, a detecção sistemática de erros, exigem tentativas múltiplas e processos iterativos de revisão e refinamento, visto que programas raramente funcionam corretamente na primeira execução (58). Isso fomenta a resiliência frente a obstáculos. O **Raciocínio Algorítmico** exige a avaliação e otimização contínua de soluções (48,29).

Do ponto de vista da formação profissional, será indispensável garantir que os professores tenham um alto nível de conhecimento e prontidão em **Pensamento Computacional** e **Raciocínio Algorítmico** (67). Programas de desenvolvimento profissional que ofereçam ferramentas e ideias aprofundadas para a integração são essenciais. Isso inclui o apoio aos professores para criar novos materiais instrucionais e adaptar suas estratégias pedagógicas (67).

Finalmente, introduzir conceitos de **Pensamento Computacional** de forma gradual, revisitando e expandindo a compreensão ao longo do currículo, parece ser uma prática eficaz (47). Começar com atividades *unplugged* e progredir para desafios de programação *plugged* pode otimizar o aprendizado (62).

2.5.2 Desafios Comuns

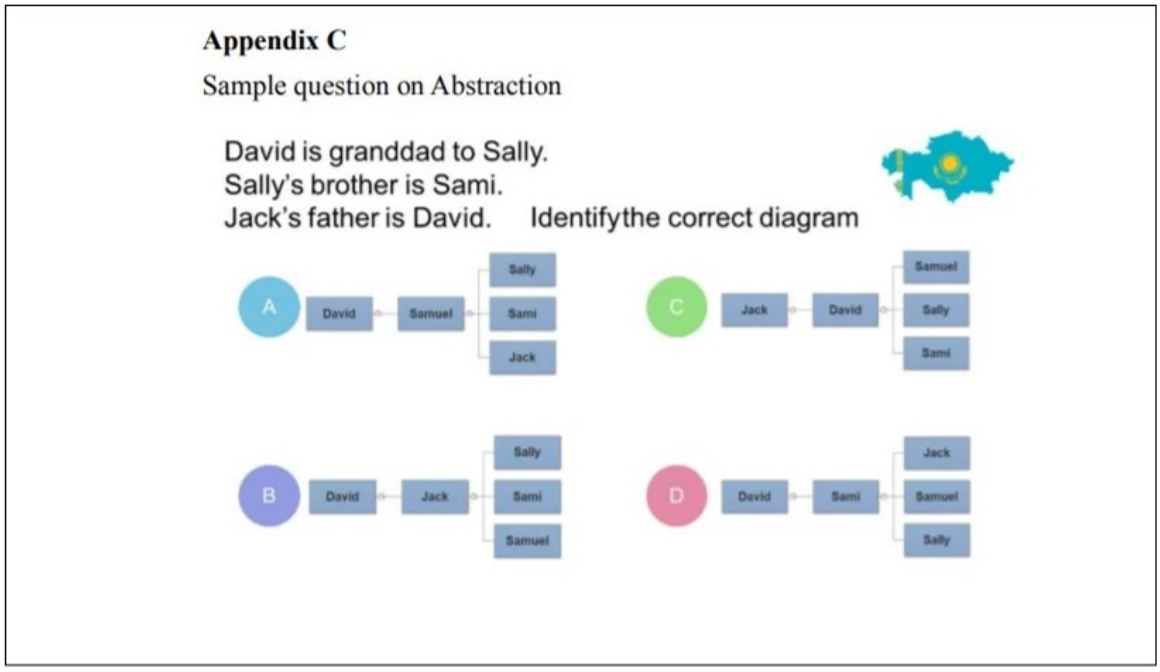
O primeiro desafio surge da falta de consenso na definição e na operacionalização do ensino de **Pensamento Computacional**. Neste cenário, uma das maiores dificuldades é a ausência de uma definição única e universalmente aceita (62). Essa ambiguidade afeta o design do currículo, a seleção de tópicos e, principalmente, a avaliação das habilidades dos alunos (58). Além disso, muitos educadores carecem de compreensão, confiança e habilidades necessárias para implementar o **Pensamento Computacional** de forma eficaz (45), ainda que suas disciplinas sejam diretamente ligadas a computação (7). Por fim, a expectativa de que as habilidades de **Pensamento Computacional** se desenvolvam naturalmente em alunos de ciência da computação é muitas vezes inadequada e irreal (47).

A integração do **Pensamento Computacional** em currículos já estabelecidos, especialmente em disciplinas não diretamente relacionadas com ciência, matemática e as engenharias, pode ser um desafio devido à falta de uma abordagem padronizada (58). Alguns conceitos e práticas de **Raciocínio Algorítmico**, como condicionais, dados, operadores, testes e depuração, e abstração/modularização, são identificados como difíceis de aprender para alunos iniciantes (57). O que diminui a autoestima e estudantes com baixa autoeficácia ou menor confiança em suas habilidades, como as de escrita, tendem a depender mais de ferramentas de IA, priorizando soluções imediatas em vez de desenvolver habilidades cognitivas e criativas (12). Por outro lado, a percepção de que o processo de aprendizado é *sem esforço* devido à disponibilidade instantânea de soluções pode desencorajar tentativas de resolução independente de problemas e reduzir a motivação dos estudantes para realizar pesquisas ou gerar suas próprias soluções (12). Além disso, a falta de estímulo mental decorrente do uso excessivo de

ferramentas que simplificam o processo pode, consequentemente, diminuir o desenvolvimento cognitivo e afetar negativamente a memória (12).

2.5.3 Avaliação e Mensuração de Habilidades de Pensamento Computacional

A avaliação das capacidades de realização do **Pensamento Computacional** é complexa e as ferramentas existentes são limitadas, muitas vezes focando em percepção-atitude ou testes de múltipla escolha simples. Alguns autores destacam que testes de múltipla escolha e questões de resposta curta são utilizados para medir o conhecimento e a compreensão de conceitos de **Pensamento Computacional** (62). Alguns testes, como o *Computational Thinking Performance Test*[7], focam em pensamento lógico, generalização e abstração (62). A Figure 2.4 exemplifica o tipo de testes de avaliação de **Pensamento Computacional** que podem ser aplicados, com questões que avaliam a capacidade de decomposição, abstração e algoritmização.



(a) Exemplo de Testes de Avaliação de Pensamento Computacional

Figure 2.4: Exemplo de Testes de Avaliação de **Pensamento Computacional** [Smith2021].

Em sala de aula, a avaliação de projetos e portfólios se destaca como ferramenta para discernir como os alunos aplicam seus conhecimentos para pensar computacionalmente e projetar artefatos computacionais criativos (58). Ao contrário dos testes e exames, que podem ser facilmente automatizados, a avaliação de projetos e portfólios requer avaliação manual ou o suporte de ferramentas de Inteligência Artificial.

As **Rubricas** de avaliação são ferramentas valiosas para guiar os professores na avaliação de atividades e objetivos de **Pensamento Computacional**, articulando critérios com descritores de desempenho que demonstram níveis progressivamente mais sofisticados de domínio (51). A Universidade de Delaware (81), por exemplo, desenvolveu uma rubrica para **Pensamento Computacional** com quatro dimensões: decomposição, algoritmos, dados e abstração que pode servir de exemplo e inspiração. A Table 2.3 apresenta um exemplo de rubrica para avaliação de habilidades de **Pensamento Computacional**.

[7]em tradução livre: Teste de Desempenho de Pensamento Computacional.

Table 2.3: Exemplo de Rubrica para Avaliação de Habilidades de Pensamento Computacional. Adaptado de (81).

Dimensão	**Excelente** 4	**Proficiente** 3	**Em Desenvolvimento** 2	**Iniciante** 1
Decomposição *Divide um problema em seus subproblemas constituintes*	Divide um problema complexo em subproblemas claramente descritos, bem definidos e distintos, mas relacionados, que são mais fáceis de resolver do que o problema original, mas quando combinados resolvem eficientemente o problema original.	Divide um problema complexo em subproblemas claramente descritos que são distintos, mas relacionados, porém carecem de eficiência, embora resolvam o problema original.	Divide um problema complexo em subproblemas que carecem de eficiência, falham em ter descrições suficientes e se sobrepõem, embora resolvam o problema original.	Divide um problema complexo em subproblemas que são ineficientes, descritos de forma inadequada, se sobrepõem ou são intimamente relacionados, e falham em resolver completamente o problema original.
Algoritmos *Cria uma série de passos ordenados para resolver um problema ou alcançar um objetivo*	Cria uma sequência lógica, eficiente e bem descrita de passos ou instruções para resolver um problema ou alcançar um objetivo.	Cria uma sequência lógica de passos que são bem descritos (por exemplo, não ambíguos, precisos) e resolvem um problema ou alcançam um objetivo, mas os passos são ineficientes, por exemplo, não em uma sequência ótima, sobrepostos, duplicados ou desnecessários.	Cria uma sequência lógica de passos que resolvem um problema ou alcançam um objetivo, mas os passos são mal descritos (por exemplo, ambíguos, vagos).	Cria uma sequência de passos que não resolvem um problema ou alcançam um objetivo. Os passos carecem de eficiência, descrições suficientes e não são descritos ou documentados.

Dimensão	**Excelente** 4	**Proficiente** 3	**Em Desenvolvimento** 2	**Iniciante** 1
Dados *Avalia um conjunto de dados para garantir que facilite a descoberta de padrões e relações*	Avalia um conjunto de dados para garantir que seja suficientemente abrangente, eficientemente organizado, significativamente rotulado e completamente descrito para que possa ser analisado para descobrir padrões e relações significativos.	Avalia um conjunto de dados para garantir que seja suficientemente abrangente, significativamente rotulado e completamente descrito, mas falha em garantir que seja eficientemente organizado.	Avalia um conjunto de dados para garantir que seja suficientemente abrangente e significativamente rotulado, mas falha em garantir que seja completamente descrito e eficientemente organizado.	Avalia um conjunto de dados, mas falha em garantir que seja suficientemente abrangente, eficientemente organizado, significativamente rotulado e completamente descrito, de modo que padrões e relações não sejam obscurecidos.
Abstração *Reduz a complexidade para criar uma representação geral de um processo ou grupo de objetos, de modo que seja não apenas apropriada para o propósito ou objetivo imediato, mas também possa ser usada em diferentes contextos*	Cria uma representação precisa, mas simplificada, de um processo ou grupo de objetos para resolver o problema ou atingir o objetivo. Seleciona características essenciais filtrando informações desnecessárias. Pode ser usada para resolver outros problemas ou objetivos.	Cria uma representação precisa, mas simplificada, de um processo ou grupo de objetos para resolver o problema ou atingir o objetivo. Seleciona características essenciais filtrando informações desnecessárias. Não pode ser usada para resolver outros problemas ou objetivos.	Cria uma representação precisa, mas simplificada, de um processo ou grupo de objetos para resolver o problema ou atingir o objetivo. Falha em selecionar todas as características essenciais filtrando informações desnecessárias. Não pode ser usada para resolver outros problemas ou objetivos.	Cria uma representação de um processo ou grupo de objetos que não é precisa, não é suficientemente simplificada, ou falha em resolver o problema ou atingir o objetivo.

A rubrica proposta pela Universidade do Dellaware (81) demonstra clareza e foco. As quatro dimensões avaliadas, Decomposição, Algoritmos, Dados e Abstraçã, são os pilares essenciais e bem definidos do **Pensamento Computacional**. Dentro de cada dimensão, os critérios para cada nível são descritos com especificidade, utilizando exemplos concretos como "subproblemas claramente descritos" ou "sequência lógica", o que os torna observáveis. Além disso, a linguagem é consistente em toda a tabela; termos-chave como "eficiente", "claramente descrito", "suficientemente abrangente", "preciso" e "simplificado" são aplicados de maneira uniforme, facilitando a comparação entre os níveis e dimensões.

Outra força significativa é a progressão lógica e hierárquica entre os níveis, de "Excelente" (4) a "Iniciante" (1). Cada degrau representa uma diferença clara na qualidade ou domínio da habilidade avaliada. Os níveis superiores não apenas cumprem os requisitos básicos, mas adicionam critérios mais sofisticados ou exigentes. Por exemplo, na Decomposição, o nível Excelente acrescenta eficiência e distinção clara aos subproblemas em relação ao Proficiente. Nos Algoritmos, a eficiência é o diferencial do Excelente. Na dimensão de Dados, os níveis inferiores perdem progressivamente critérios como organização e descrição completa. Já na Abstração, a capacidade de reutilização da representação em outros contextos é exclusiva do nível Excelente, marcando um salto qualitativo.

Finalmente, a rubrica se destaca pela ênfase em critérios mensuráveis e observáveis, focando nos resultados tangíveis do trabalho do aluno – como os subproblemas foram divididos, como os passos do algoritmo foram descritos, como os dados foram organizados ou como a representação abstrata foi construída –, evitando termos excessivamente subjetivos. Essa objetividade, aliada à sua abordagem multidimensional, que avalia habilidades distintas porém inter-relacionadas, fornece um retrato mais completo do aluno do que uma nota única. Isso se traduz em uma utilidade pedagógica prática: ela serve como um guia claro de ensino, estabelecendo metas de aprendizagem; permite ao professor dar feedback específico e direcionado por dimensão; e promove maior consistência e equidade na avaliação, reduzindo a subjetividade.

A aplicação eficaz da rúbrica da Universidade Delaware (81) exige atenção a certas considerações para garantir uma avaliação precisa. Primeiramente, a subjetividade residual em termos como "eficiente" ou "suficientemente abrangente" requer que o avaliador mantenha consistência, sendo o uso de "trabalhos âncora" uma prática recomendada para calibrar a avaliação. Essa necessidade de julgamento é particularmente notável ao diferenciar os níveis "Proficiente" (3) e "Excelente" (4), que dependem de conceitos avançados como eficiência e reusabilidade, exigindo que a tarefa proposta seja complexa o suficiente para permitir que o aluno demonstre tais habilidades. O avaliador deve estar ciente da forte interdependência prática entre as dimensões, por exemplo, uma decomposição fraca quase sempre levará a um algoritmo ineficiente, compreendendo que uma falha em uma área impacta diretamente o desempenho em outra, mesmo que sejam avaliadas separadamente. Esta rúbrica (81) serve apenas como referência. Ela requer adaptações para o uso com o **Raciocínio Algorítmico**.

Em um contexto de domínio do **Pensamento Computacional** e do **Raciocínio Algorítmico**, a autoavaliação surge como uma estratégia interessante. Existem escalas psicométricas e questionários de autoavaliação são empregados para medir a percepção dos alunos sobre suas próprias habilidades de **Pensamento Computacional** e sua autoeficácia. Algumas estão academicamente validadas, como o *Computational Thinking Self-Efficacy Survey*[8] (83)), que medem a percepção do estudante sobre sua capacidade de pensar computacionalmente. Essas ferramentas são úteis para entender como os alunos se veem em relação ao **Pensamento Computacional** e podem ajudar a identificar áreas de melhoria que poderão ser usadas no ensino de **Raciocínio Algorítmico**.

Algumas destas estratégias de avaliação podem ser realizadas por meio de ferramentas online como:

- **UniCTCheck (Ainda em Desenvolvimento)**: um método inovador para avaliar habilidades de **Pensamento Computacional** em alunos universitários de Ciência da Computação. O **CTScore** é um aplicativo web interativo que mede sete componentes de **Pensamento Computacional**, reconhecimento de padrões, pensamento criativo, pensamento algorítmico, resolução de problemas, pensamento crítico, decomposição e abstração, por meio de 12 questões. O **CTProg** é uma escala psicométrica que mede a compreensão conceitual de cinco fundamentos de programação, direções básicas e sequências, condicionais, laços de repetição, funções, estruturas de dados (49).

[8] em tradução livre: Questionário de Autoeficácia de Pensamento Computacional.

- **Bebras Questions**: utilizadas como ferramenta de teste para analisar melhorias nas capacidades de **Pensamento Computacional** para crianças e adolescentes (84).
- **Dr. Scratch, LOGO, C, C++, AgentCubes**: linguagens, ferramentas de depuração, e teste mencionadas como ferramentas de avaliação que podem ser utilizadas para aprimorar o **Pensamento Computacional**, com foco principal no treinamento de habilidades de programação (67).

Apesar do crescimento na pesquisa sobre avaliação de **Pensamento Computacional**, como ainda existe uma falta de consenso na descrição ou nas partes constituintes do **Pensamento Computacional**, que dificulta a padronização e validação de instrumentos de avaliação (49), muitos estudos apontam para a necessidade de mais pesquisas para desenvolver e validar uma conceituação teórica robusta do **Pensamento Computacional** e, consequentemente, ferramentas de avaliação mais eficazes (83).

Para o avanço da educação, parece ser importante que a comunidade acadêmica continue a colaborar na definição e operacionalização do **Pensamento Computacional**, o que, por sua vez, permitirá o desenvolvimento de instrumentos de avaliação mais rigorosos e comparáveis. Isso parece ser indispensável para medir o impacto das intervenções pedagógicas e garantir a qualidade da educação em **Pensamento Computacional**. Além disso, para a definição de uma disciplina de **Raciocínio Algorítmico** torna-se pertinente a análise deste tema em disciplinas de graduação ao redor do planeta.

3 Analisando Currículos Internacionais

A integração do **Pensamento Computacional** nos currículos do ensino superior tem sido uma tendência crescente, com universidades em todo o mundo revendo os conteúdos e estruturas dos seus cursos e disciplinas para incorporar esses conceitos (28). Na maior parte das universidades estudadas, o objetivo é incluir o **Pensamento Computacional** e, consequentemente, o **Raciocínio Algorítmico** além das disciplinas de programação (6). Esta evolução curricular representa uma transição de um modelo que ensinava primariamente programar para o computador para um que enfatiza o pensar computacionalmente. Instituições de prestígio estão se afastando de um foco exclusivo na sintaxe de programação para priorizar uma compreensão mais profunda de como os sistemas computacionais funcionam e como aplicar o raciocínio computacional para resolver problemas complexos (47). A ênfase em modelos algorítmicos ou modelos de solução apresentados em manuais (como é o caso de livros didáticos focados em procedimentos) pode gerar aprendizado de curto prazo, mas falha em aprimorar o conhecimento conceitual de longo prazo dos alunos (32). Estas preocupações indicam uma maturidade crescente do campo da educação em ciência da computação, na qual a alfabetização computacional é vista como uma competência mais ampla e conceitual, que transcende a mera proficiência numa linguagem de programação específica (29).

A comparação entre o ensino no Brasil e universidades internacionais é complexa e desafiadora.

Existem diferenças de metodologia entre as universidades da Europa, EUA e Brasil. A forma como o tempo de aula presencial, trabalhos e pesquisa é contabilizado para validação da disciplina tem impacto direto no formato dessas disciplinas para inclusão da Metodologia **DAAD** (**D**ecomposição, **A**bstração, **A**lgoritmização e **D**epuração) (29).

Nos EUA, o sistema predominante é o de *Credit Hours* ou *Semester Credit Hours*. Uma hora de crédito engloba, normalmente, tanto o tempo despendido em sala de aula quanto o estudo independente. Geralmente, **cada hora de crédito corresponde a uma hora de instrução semanal em sala de aula, complementada por duas horas de estudo individual, ao longo de um semestre que dura aproximadamente 15 semanas**. Isso significa que uma disciplina comum de 3 créditos terá aproximadamente 3 horas de aula por semana e 6 horas de estudo fora da sala, totalizando cerca de 45 horas de contato direto com o professor ao longo do semestre e 90 horas de estudo independente. Para a obtenção de um diploma de bacharelado, os estudantes normalmente precisam acumular entre 120 e 130 horas de crédito no curso (85). A Table 3.1 resume este conceito.

Table 3.1: Exemplo de Cálculo de horas, considerando o Sistema de Créditos Acadêmicos nos EUA.

Créditos da Disciplina	Horas em Sala de Aula (por semestre)	Horas de Estudo Fora da Sala (por semestre)	Horas Totais de Estudo (por semestre)
1 crédito	15 horas	30 horas	45 horas
2 créditos	30 horas	60 horas	90 horas
3 créditos	45 horas	90 horas	135 horas

Por outro lado, na Europa, a maioria das instituições acadêmicas adota o Sistema Europeu de Transferência e Acumulação de Créditos, em inglês **E**uropean **C**redit **T**ransfer and

Accumulation **S**ystem, **ECTS**. É um sistema que prioriza o volume de trabalho total que um estudante precisa dedicar para alcançar os resultados de aprendizado esperados (86). Isso abrange não somente as horas de aula, mas também atividades como estudo autônomo, preparação para avaliações, desenvolvimento de projetos e estágios. O padrão estabelece que 60 créditos **ECTS** equivalem ao volume de trabalho de um estudante em tempo integral durante um ano acadêmico. A conversão de 1 crédito **ECTS** geralmente representa entre 25 e 30 horas de trabalho total do estudante, embora essa proporção possa ter pequenas variações entre os países europeus. Diferentemente do modelo americano, não há uma correlação direta entre os créditos **ECTS** e as horas de contato em sala de aula, pois o foco está no esforço total do estudante, o que pode resultar em disciplinas com muitos créditos **ECTS** e poucas horas de aula se o trabalho independente for intenso. O **ECTS** foi concebido para otimizar a mobilidade estudantil e o reconhecimento de qualificações entre as universidades europeias. Um programa de bacharelado com duração de 3 anos geralmente corresponde a 180 **ECTS**, enquanto um de 4 anos pode totalizar 240 **ECTS** (86). A Table 3.2 resume este conceito.

Table 3.2: Exemplo de Cálculo de horas, considerando o Sistema Europeu de Transferência e Acumulação de Créditos, ECTS.

Créditos **ECTS** da Disciplina	Horas Totais de Trabalho do Estudante (por semestre/módulo)
1 ECTS****	25-30 horas
2 ECTS****	50-60 horas
3 ECTS****	75-90 horas

As universidades chinesas geralmente adotam um sistema de créditos que está, na maioria das vezes, vinculado às **horas de contato** semanais. Embora não exista um sistema de crédito totalmente unificado em todas as instituições de ensino superior chinesas, um padrão comum é que **1 crédito corresponda a uma hora de aula por semana**; uma hora aula corresponde a 45 minutos, ao longo de um semestre (87).

No Brasil, o sistema de créditos nas universidades públicas e privadas é predominantemente baseado na **carga horária de aulas** teóricas e práticas. A **L**ei de **D**iretrizes e **B**ases da Educação Nacional, **LDB** - Lei nº 9.394/1996, (88) estabelece a regulamentação geral para a educação superior no país. Tradicionalmente, a conversão mais comum é que **1 crédito equivale a 15 horas-aula**. Contudo, essa equivalência pode variar um pouco entre as instituições e tipos de atividades. Em linhas gerais, para disciplinas teóricas, 1 crédito geralmente corresponde a 15 horas de aula e para disciplinas práticas, estágios ou atividades de laboratório, a proporção de horas-aula por crédito pode ser diferente, frequentemente exigindo mais horas de contato para 1 crédito (por exemplo, 30 horas de prática para 1 crédito). A carga anual típica varia entre 750-1.200 horas de contato, equivalente a 50-80 créditos americanos.

O **Ministério da Educação (MEC)**, por meio de resoluções do Conselho Nacional de Educação, define cargas horárias mínimas para a integralização de diferentes cursos de graduação. Essas cargas horárias são, então, desdobradas pelas universidades em disciplinas com seus respectivos créditos. Um ponto relevante é que a **LDB**, no artigo 47, exige um **mínimo de duzentos dias letivos** anuais e, nas instituições públicas de educação superior (88).

Nas universidades brasileiras, a determinação de cargas horárias e créditos é um sistema de duas camadas, combinando diretrizes nacionais com a autonomia de cada instituição. O **M**inistério da **E**ducação e **C**ultura, **MEC**, e o **C**onselho **N**acional de **E**ducação, **CNE**, estabelecem a base legal para a carga horária dos cursos de graduação, principalmente por meio de resoluções e pareceres. A autonomia das universidades se manifesta na criação de disciplinas por cursos.

Para cada curso de bacharelado, o **MEC** define uma carga horária total mínima que precisa

ser cumprida para que o diploma seja válido. A Resolução CNE/CES nº 2(89), por exemplo, agrupa os cursos em diferentes faixas de carga horária. Um curso de Administração tem no mínimo 3.000 horas, enquanto um curso de Direito exige no mínimo 3.700 horas. A carga horária não se resume apenas ao tempo em sala de aula. Ela inclui as atividades que compõem o trabalho acadêmico do estudante cujo peso e duração dependem da universidade. Conforme o Parecer **CNE/CES** nº 261 (90), a hora-aula é um conceito amplo, e a carga horária total do curso pode abranger: aulas teóricas e práticas; horas de estudo individual ou em grupo; trabalhos de campo e de laboratório; estágios supervisionados; atividades de extensão e trabalho de conclusão de curso.

O crédito é, essencialmente, uma unidade de medida do trabalho do estudante. No modelo mais comum, um crédito pode equivaler a 15 horas de atividades em sala de aula. Assim, uma disciplina de 60 horas/aula ao longo de um semestre valeria 4 créditos. Para atividades práticas ou de laboratório, a proporção pode ser diferente. Por exemplo, 30 horas de atividades práticas podem equivaler a 1 crédito (91). Imagine uma disciplina com a seguinte estrutura em uma universidade hipotética:

- **Carga horária total**: 90 horas;
- **Aulas teóricas**: 60 horas;
- **Aulas práticas**: 30 horas;

Se a universidade define que 1 crédito teórico é igual a 15 horas e 1 crédito prático corresponde a 30 horas, essa disciplina teria:

- 60 / 15 = 4 créditos teóricos
- 30 / 30 = 1 crédito prático
- Total de 5 créditos para a disciplina.

As universidades particulares, notadamente as que têm curso superior noturno, ou não consideram horas de trabalho independente, ou não abordam estes trabalhos nas avaliações. A Table 3.3 permite a comparação entre as disciplinas avaliadas neste estudo e o equivalente em horas-aula no Brasil, considerando que todo esforço de aprendizado seja feito em sala e apenas as universidades utilizadas neste estudo.

Table 3.3: Comparação entre as disciplinas relacionadas com pensamento computacional e raciocínio algorítmico e a carga horária no Brasil.

País	Instituição	Disciplina	Créditos Originais	Horas-Aula Brasil*
Estados Unidos	Carnegie Mellon University	Princípios da Computação	10 créditos US	**150 horas-aula**
Estados Unidos	Carnegie Mellon University	Estrutura de Dados e Algoritmos	12 créditos US	**180 horas-aula**
Estados Unidos	Carnegie Mellon University	Codificação e Pensamento Computacional com VEX V5	Curso de extensão	Curso de extensão
Estados Unidos	MIT	Introdução ao Pensamento Computacional em Python (6.100A)	12 créditos US	**180 horas-aula**

País	Instituição	Disciplina	Créditos Originais	Horas-Aula Brasil*
Estados Unidos	MIT	Introdução ao Pensamento Computacional e Ciência de Dados (6.100B)	12 créditos US	**180 horas-aula**
Estados Unidos	Penn State University	Disciplinas de Ciência da Computação (integradas)	N/D	N/D
Estados Unidos	University of Wisconsin-Madison	Resolução de Problemas Usando Computadores (COMP SCI 310)	3 créditos US	**45 horas-aula**
Estados Unidos	Harvard University	Introdução à Ciência da Computação (CS50)	4 créditos US	**60 horas-aula**
Estados Unidos	Stanford University	Programação Introdutória (CS106A)	5 créditos US	**75 horas-aula**
Estados Unidos	Stanford University	Estruturas de Dados e Algoritmos (CS106B)	5 créditos US	**75 horas-aula**
Reino Unido	Cambridge University	Algoritmos 1	Parte do curso geral	Parte do curso geral
Reino Unido	Cambridge University	Computational Thinking Challenge	Projeto	Projeto
Reino Unido	Cambridge University	Lógica e Algoritmos	Parte do curso geral	Parte do curso geral
Reino Unido	Oxford University	Ciência da Computação	N/D	N/D
Reino Unido	Oxford University	Matemática e Ciência da Computação	N/D	N/D
Reino Unido	Oxford University	Ciência da Computação e Filosofia	N/D	N/D
Reino Unido	Imperial College London	Matemática Discreta	7,5 ECTS	**83 horas-aula**
Reino Unido	Imperial College London	Matemática, Lógica e Raciocínio	7,5 ECTS	**83 horas-aula**
Reino Unido	Imperial College London	Grafos e Algoritmos	7,5 ECTS	**83 horas-aula**
Reino Unido	Imperial College London	Raciocínio Simbólico (optativa)	6 ECTS	**66 horas-aula**
Reino Unido	Imperial College London	Técnicas Computacionais (optativa)	6 ECTS	**66 horas-aula**

País	Instituição	Disciplina	Créditos Originais	Horas-Aula Brasil*
Reino Unido	Imperial College London	Projeto e Análise de Algoritmos	7,5 ECTS	**83 horas-aula**
Reino Unido	Cardiff University	Pensamento Computacional (CM1101)	10 ECTS	**110 horas-aula**
Reino Unido	Cardiff University	Resolução de Problemas com Python	5 ECTS	**55 horas-aula**
Reino Unido	Cardiff University	Programação Orientada a Objetos com Java	10 ECTS	**110 horas-aula**
Escócia	Edinburgh University	Introdução a Computação	10 ECTS	**110 horas-aula**
Escócia	Edinburgh University	Programação de Computadores	10 ECTS	**110 horas-aula**
Escócia	Edinburgh University	Como Resolver Problemas Usando Computadores	N/D	N/D
Suíça	ETH Zurich	Pensamento Computacional	N/D	N/D
Suíça	ETH Zurich	Introdução a Programação	8 ECTS	**88 horas-aula**
Suíça	ETH Zurich	Estruturas de Dados e Algoritmos	8 ECTS	**88 horas-aula**
Suíça	ETH Zurich	Projeto Digital e Arquitetura de Computador	6 ECTS	**66 horas-aula**
Alemanha	Technical University of Munich	Pensamento Computacional (pesquisa)	N/D	N/D
Suíça	École hôtelière de Lausanne (EHL)	Pensamento Computacional	3,5 ECTS	**38,5 horas-aula**

Na Table 3.3, N/D significa Não Disponível. Foram usadas as seguintes fórmulas de conversão:

- N/D significa Não Disponível.
- Fórmulas de conversão:
  - **EUA → Horas-aula**: Créditos US × 15 horas
  - **ECTS → Horas-aula**: ECTS × 11 horas (média europeia de contato)
  - **Reino Unido → Horas-aula**: Créditos UK × 10 horas

Além das diferenças na expectativa de estudo independente por parte dos alunos, que podem influenciar o resultado das disciplinas, existem diferenças importantes na forma como as

universidades integram o **Pensamento Computacional** e o **Raciocínio Algorítmico** em seus currículos. Universidades de diferentes regiões do mundo implementam o **Pensamento Computacional**, com foco na descoberta de elementos **DAAD**, de forma diferente em suas disciplinas. A Table 3.4 oferece uma visão estruturada e comparativa de como universidades líderes em diferentes regiões abordam a educação em Pensamento Computacional, com foco nos elementos **DAAD**. Esta análise permite identificar padrões comuns e pontos fortes únicos, fornecendo informações valiosas para o design de um currículo de 80 horas, ao apresentar diversos modelos de integração e ênfase pedagógica.

Table 3.4: Exemplos de Universidades e a Integração de **Pensamento Computacional/DAAD** em seus Currículos de Graduação.

Universidade	País/Região	Tipo de Programa/Integração	Disciplinas Principais	Créditos/Carga Horária	Elementos DAAD/PC Enfatizados	Linguagens/Ferramentas	Destaques Pedagógicos
Carnegie Mellon University	EUA	Centro dedicado de PC, Currículo específico	Princípios da Computação; Estrutura de Dados e Algoritmos; Codificação e PC com VEX V5	10 créditos US; 12 créditos US; curso de extensão	Decomposição, Abstração, Algoritmização, Depuração	VEX V5 Robotics	PROBEs (explorações orientadas a problemas), aplicação em contextos do mundo real
MIT	EUA	Requisito mínimo para todos os estudantes	Introdução ao PC em Python (6.100A); PC e Ciência de Dados (6.100B)	12 créditos US cada	Raciocínio rigoroso, comunicação, impacto transformador	Python	Desenvolvimento de confiança para programas úteis independente da área de estudo
Penn State University	EUA	Integrado no currículo de Ciência da Computação	Disciplinas do curso de CS	Não especificado	Conceituação e implementação, abstração em múltiplos níveis, análise de código	Não especificado	Ênfase em comportamento, eficiência e correção do código
University of Wisconsin-Madison	EUA	Disciplina específica	Resolução de Problemas Usando Computadores (COMP SCI 310)	3 créditos US	Abstração, decomposição de problemas	Linguagens de manipulação simbólica, pacotes de software	Métodos computacionais para resolução de problemas

Universidade	País/Região	Tipo de Programa/Integração	Disciplinas Principais	Créditos/Carga Horária	Elementos CA/AD/PC Enfatizados	Linguagens/Ferramentas	Destaques Pedagógicos
Harvard University	EUA	Disciplina introdutória	Introdução à Ciência da Computação (CS50)	4 créditos US (estimado)	Raciocínio algorítmico, decomposição, abstração, depuração	Scratch, C, Python	Progressão estruturada (Scratch → C → Python), ênfase em fundamentos computacionais
Stanford University	EUA	Disciplinas introdutórias com aprendizagem por maestria	Programação Introdutória (CS106A); Estruturas de Dados e Algoritmos (CS106B)	5 créditos US cada	Decomposição, abstração, controle de fluxo, análise de complexidade	Python, C++, Karel the Robot	Uso de Karel the Robot, suporte pedagógico extensivo (LaIR Hours), progressão de Python para C++
Cambridge University	Reino Unido	Instituto Cambridge Mathematics, currículo integrado	Algoritmos 1; Lógica e Algoritmos; Computational Thinking Challenge	Curso geral	Abstração, lógica, algoritmos, representação de dados	Não especificado	Relação profunda entre PC e pensamento funcional, projeto de pesquisa e avaliação digital
Oxford University	Reino Unido	Integrado em múltiplos cursos	Cursos de CS, Matemática e CS, CS e Filosofia	Não especificado	Design de programas, conceitos de alto nível	Não especificado	UK Bebras Challenge, Oxford University Computing Challenge, ênfase em leitura de materiais conceituais

Universidade	País/Região	Tipo de Programa/Unidade	Disciplinas Principais	Créditos/Carga Horária	Elementos CA/PC Enfatizados	Linguagens/Ferramentas	Destaques Pedagógicos
Imperial College London	Reino Unido	Programa de graduação em computação	Matemática Discreta; Matemática, Lógica e Raciocínio; Grafos e Algoritmos; Projeto e Análise de Algoritmos; Raciocínio Simbólico; Técnicas Computacionais	7,5 ECTS; 7,5 ECTS; 7,5 ECTS; 7,5 ECTS; 6 ECTS; 6 ECTS	Princípios fundamentais, pensamento lógico, algoritmos	Não especificado	Laboratórios, resolução de problemas, projetos de design
Cardiff University	Reino Unido	Disciplina dedicada no 1º ano	Pensamento Computacional (CM1101); Resolução de Problemas com Python; POO com Java	10 ECTS; 5 ECTS; 10 ECTS	PC dedicado, resolução de problemas	Python, Java	Foco precoce distribuído por diversas disciplinas
ETH Zurich	Suíça	Paradigma de sala de aula invertida	Introdução a Programação; Estruturas de Dados e Algoritmos; Projeto Digital	8 ECTS; 8 ECTS; 6 ECTS	Abstração (“PC é mais que programar”)	Python, C++, Java	Flipped classroom, vídeos e leituras online, autoestudo
Edinburgh University	Escócia	Programa de Ciência da Computação	Introdução a Computação; Programação de Computadores; Como Resolver Problemas	10 ECTS; 10 ECTS; N/D	Compreensão, design, implementação de sistemas	Não especificado	Trabalho prático, construção de sistemas, trabalho experimental
École Hôtelière de Lausanne (EHL)	Suíça	Módulo específico em Administração Hoteleira	Pensamento Computacional	3,5 ECTS (88h: 30h contato + 58h estudo)	Base para resolução estruturada, análise de dados	Não especificado	Aplicação transdisciplinar, integração em área não-tecnológica

Uma análise mais detalhada das universidades citadas nas tabelas Table 3.3 e Table 3.4

revela as nuances na implementação do **Pensamento Computacional** e do **Raciocínio Algorítmico** em seus currículos.

3.1 Detalhamento das Universidades nos Estados Unidos

A Carnegie Mellon University (CMU) é uma instituição pioneira na promoção do **Pensamento Computacional**, com Jeannette Wing como figura central nesta iniciativa. A universidade possui um centro específico para **Pensamento Computacional** dedicado (92). A abordagem da CMU envolve o *Project Olympus* de inovação, no centro do qual estão os desafios ***PR**oblem-**O**riented **B**usiness **E**xplorations*, **PROBES**, uma sigla em inglês para exploração de problemas orientados a negócio. Alunos que resolvem estes problemas aplicam conceitos computacionais inovadores a problemas práticos para aprenderem na prática o valor do **Pensamento Computacional** (93). A disciplina Princípios da Computação (10 Créditos), introduzida em 2005, foca no estudo do processo de computação, não necessariamente na operação de um computador (94). A disciplina Estrutura de Dados e Algoritmos (12 Créditos) espera que os alunos aprendam a decompor problemas. Além disso, busca desenvolver habilidades para acompanhar o progresso da solução, avaliar a correção do código e corrigir falhas por meio de depuração automatizada (95).

Em disciplinas diretamente relacionadas com Ciência e Engenharia da Computação, a disciplina Codificação e **Pensamento Computacional** com VEX V5 (96), na forma de um curso de extensão, utiliza atividades de programação estruturadas em contextos de projetos do mundo real, ensinando explicitamente a decomposição para simplificar a codificação complexa (97). A ênfase da CMU na aplicação do **Pensamento Computacional** para resolver problemas do mundo real transcende a mera compreensão. Pensamento Computacional significa pensar algoritmicamente e com a habilidade de aplicar conceitos matemáticos, tais como indução, para desenvolver soluções mais eficientes, justas e seguras (92). Essa abordagem cria uma relação causal: problemas autênticos fornecem a motivação e o contexto para que os alunos se engajem profundamente, desenvolvam e apliquem as habilidades **DAAD**.

O MIT, Massachusetts Institute of Technology, reconhece o **Pensamento Computacional** como um tipo distinto de raciocínio rigoroso de importante valor intelectual. Este raciocínio requer e desenvolve modos importantes de comunicação e a necessidade de compreender o impacto transformador da computação em outras disciplinas (98). O MIT recomenda um requisito mínimo de computação para todos os estudantes de graduação, enfatizando que o **Pensamento Computacional** é mais amplo do que a proficiência em programação (100). As disciplinas Introdução ao Pensamento Computacional em Python (6.100A, 12 Créditos) e Introdução ao Pensamento Computacional e Ciência de Dados (6.100B, 12 Créditos) utilizam Python como linguagem de programação e visam dar aos alunos a confiança para escrever pequenos programas úteis, independentemente de sua área principal de estudos (100).

A **Penn State** integra o **Pensamento Computacional** em seu currículo do curso de Ciência da Computação (72). As disciplinas enfatizam a conceituação e implementação de soluções computacionais, o raciocínio sobre problemas em múltiplos níveis de abstração e a análise de código quanto ao comportamento, eficiência e correção (72). Também abordam habilidades de desenvolvimento e manutenção de programas, como depuração e teste. Já na **University of Wisconsin-Madison**, a disciplina Introdução à Programação para Ciências Computacionais (CMPSC 204, 3 créditos) introduz a abstração e decomposição de problemas, discute métodos de uso de computadores para resolver problemas, incluindo técnicas elementares de programação, linguagens de manipulação simbólica e pacotes de software (72).

A **Harvard University** adota uma estratégia pedagógica distintiva na sua disciplina Introdução à Ciência da Computação, CS50, caracterizada pela progressão no ensino do **Pensamento Computacional**. O curso, com duração de 11 semanas e exigindo dedicação semanal

de 10 a 20 horas, tem aproximadamente 4 créditos. A disciplina inicia com a linguagem C (75) antes de avançar para Python, demonstrando uma filosofia que prioriza a compreensão fundamental dos mecanismos computacionais (101,102). Esta abordagem se alinha com os princípios estabelecidos por Jeannette Wing sobre **Pensamento Computacional**(28), enfatizando que Pensamento Computacional significa pensar algoritmicamente e com a habilidade de aplicar conceitos matemáticos, tais como indução, para desenvolver soluções mais eficientes, justas e seguras (102).

A estrutura curricular do CS50 revela uma progressão cuidadosamente planejada: início com Scratch(53) para conceitos fundamentais, transição para Linguagem C(75) para compreensão de baixo nível, seguida por Python(71) para aplicações de alto nível (102). A introdução explícita de algoritmos na Semana 3, incluindo **Notação Assintótica**, estabelece uma base sólida para análise de eficiência que complementa o aprendizado prático da programação imperativa (102). Esta metodologia reflete uma compreensão de que o raciocínio algorítmico deve ser construído sobre fundamentos sólidos de como os computadores operam, incluindo gerenciamento de memória e ponteiros, conceitos essenciais para otimização e depuração avançadas (102).

A **Stanford University** implementa uma filosofia de aprendizagem por maestria em suas disciplinas CS106A (5 créditos) e CS106B (5 créditos), caracterizada por um sistema extensivo de suporte pedagógico e progressão conceitual estruturada (78,103). O CS106A utiliza *Karel the Robot* (104) como ferramenta introdutória, permitindo que os alunos desenvolvam intuição sobre decomposição de problemas e controle de fluxo antes de enfrentar a complexidade sintática de linguagens de propósito geral (78,105). Esta abordagem reconhece que a abstração computacional pode ser ensinada de forma mais eficaz quando separada das complexidades de implementação.

O sistema de suporte de Stanford, incluindo *LaIR Hours*, horas de ajuda presencial noturna, seções semanais obrigatórias e políticas flexíveis de avaliação, reflete uma compreensão profunda de que o desenvolvimento do **Pensamento Computacional** requer prática intensiva com retorno contínuo (78,103). A transição do CS106A (Python) para o CS106B, Linguagem C++ (106), demonstra uma progressão estratégica: estabelecimento inicial de confiança em resolução de problemas por meio de Python, seguido pelo aprofundamento em conceitos de baixo nível por meio de C++ (103). A introdução da Notação Big O no CS106B integra-se naturalmente com o aprendizado de estruturas de dados e algoritmos avançados, demonstrando como a análise de complexidade se conecta diretamente com decisões de implementação (103). A Figure 3.1 resume a estrutura do curso CS106A e CS106B.

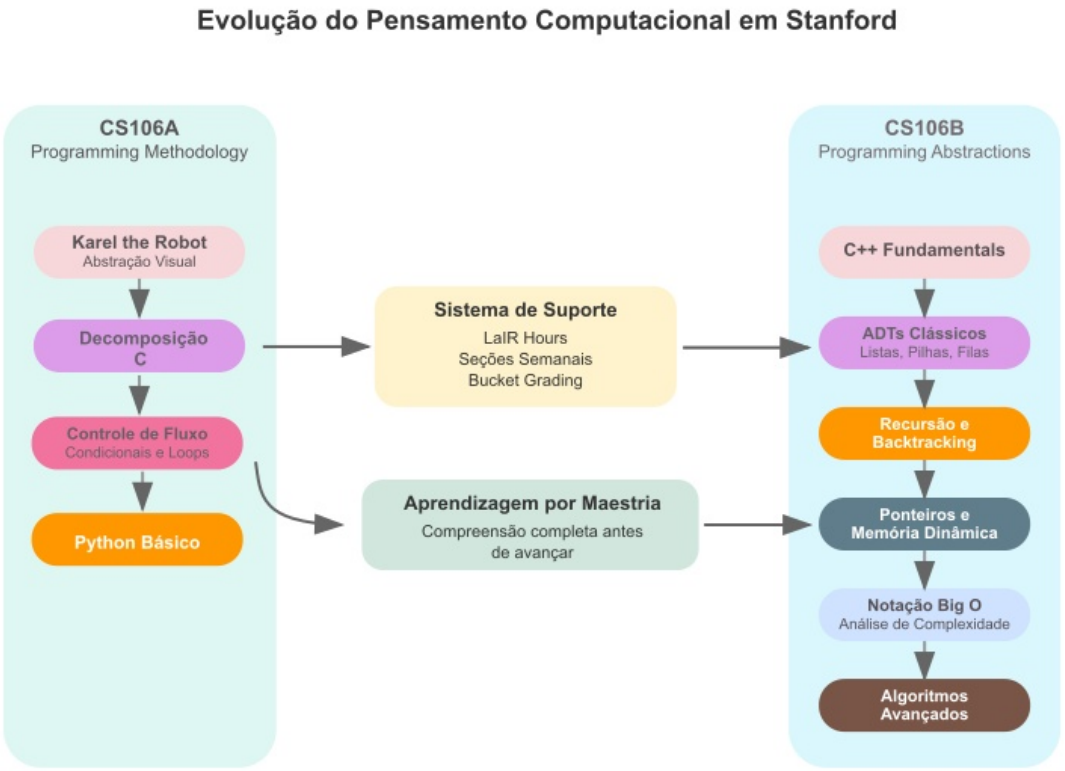


Figure 3.1: Estrutura do curso CS106A e CS106B da Stanford University.

Síntese da estrutura do curso CS106A e CS106B da Stanford University. A figura ilustra a progressão do ensino de **Pensamento Computacional** desde conceitos básicos até estruturas de dados avançadas, destacando a transição de C para Python e deste para C++ e a introdução da Notação Big O.

3.1.1 Síntese das Abordagens Pedagógicas Americanas

A análise das metodologias adotadas pelas principais universidades americanas revela um espectro diversificado de estratégias para o desenvolvimento do **Pensamento Computacional**, cada uma refletindo filosofias educacionais distintas mas complementares. Harvard prioriza a exposição inicial aos fundamentos computacionais por meio da progressão "do baixo para o alto", argumentando que a compreensão profunda dos mecanismos de baixo nível facilita a maestria posterior de abstrações de alto nível (102). Stanford implementa uma abordagem de "aprendizagem por maestria", enfatizando a construção gradual de confiança por meio de ferramentas acessíveis como **Karel the Robot** antes de introduzir complexidades técnicas (78).

A Carnegie Mellon University distingue-se pela integração de problemas autênticos do mundo real por meio do *Project Olympus* e dos desafios **PROBES**, demonstrando como o **Pensamento Computacional** pode ser aplicado em contextos práticos de negócio (92). O MIT enfatiza uma abordagem estruturada que reconhece o **Pensamento Computacional** como um tipo distinto de raciocínio rigoroso, implementando disciplinas específicas como Introdução ao Pensamento Computacional em Python (6.100A, 12 Créditos) que visam desenvolver confiança na programação independentemente da área de estudos principal. As demais instituições, incluindo Penn State, University of Wisconsin-Madison e University of Texas at Austin, convergem na ênfase da decomposição de problemas, análise de eficiência e desenvolvimento de habilidades de depuração como componentes fundamentais do currículo introdutório.

Esta diversidade metodológica demonstra que o **Pensamento Computacional** transcende a mera proficiência em programação, exigindo desenvolvimento de habilidades de decomposição, abstração e reconhecimento de padrões que são fundamentais para a resolução eficaz de problemas computacionais complexos. As variações nas estratégias pedagógicas refletem diferentes caminhos para alcançar o mesmo objetivo: capacitar os alunos a pensar algoritmicamente e aplicar conceitos matemáticos para desenvolver soluções eficientes, justas e seguras, conforme preconizado por Jeannette Wing e implementado de forma pioneira na Carnegie Mellon University (92).

3.2 Detalhamento das Universidades no Reino Unido

A **Cambridge University** estabelece uma abordagem distintiva para o desenvolvimento do **Raciocínio Algorítmico** por meio do instituto *Cambridge Mathematics*, uma iniciativa colaborativa criada pela união da editora universitária com as faculdades de educação e matemática (20). O instituto produz cursos, seminários e material didático especializado para a formação de competências computacionais em alunos de diferentes cursos de graduação, demonstrando uma visão integrada do **Pensamento Computacional** que transcende as fronteiras tradicionais da ciência da computação corroborando as ideias de Wing (2006) (28) e Lehmann (2033) (29). Nos documentos gerados pelo Cambridge Mathematics ((107) e (108)) é possível observar uma relação profunda e estruturada entre o **Pensamento Computacional** e o pensamento funcional matemático, estabelecendo uma base teórica sólida (20).

O material desenvolvido pelo instituto, (107) e (108), é sistematicamente integrado no currículo universitário, com ênfase particular em abstração, lógica, algoritmos e representação de dados (20), estabelecendo a base para o desenvolvimento do raciocínio computacional em múltiplas disciplinas.

O currículo de disciplinas introdutórias relacionadas ao desenvolvimento do **Raciocínio Algorítimico** em Cambridge revela uma progressão estruturada e rigorosa que combina fundamentos teóricos com aplicação prática. A disciplina Algoritmos 1 (14 créditos ECTS), parte do curso geral de Ciência da Computação (109), abrange tópicos fundamentais como ordenação, análise de complexidade, paradigmas de design incluindo divisão e conquista, programação dinâmica e algoritmos gulosos, estruturas de dados, e análise formal da eficiência dos algoritmos (110). Esta disciplina estabelece a base algorítmica essencial para o desenvolvimento do raciocínio computacional, enfatizando não apenas a implementação de algoritmos, mas também a compreensão profunda de sua eficiência e aplicabilidade.

Esta base algorítmica é complementada pelo *Computational Thinking Challenge*, um projeto de pesquisa e avaliação digital desenvolvido pela faculdade de educação, especificamente focado na avaliação de competências de **Pensamento Computacional** voltado para estudantes e educadores (111). Esta iniciativa serve como evidência do compromisso de Cambridge não apenas com o ensino, mas também com a pesquisa e avaliação sistemática das metodologias de desenvolvimento do **Pensamento Computacional**.

Finalmente, em Cambridge, a disciplina Algoritmos 2 (14 créditos ECTS)complementa a formação em **Raciocínio Algorítmico**, abordando conteúdos avançados. Esta disciplina inclui grafos e algoritmos de busca de caminho, grafos e subgrafos, com foco em fluxo máximo, emparelhamentos bipartidos, árvores geradoras mínimas de Kruskal e Prim, e ordenação topológica, e estruturas de dados avançadas (como heap binomial, análise amortizada, heaps de Fibonacci e conjuntos disjuntos)(112).

Analisando apenas as duas dicisplinas diretamente relacionadas com Ciência da Computação a uma das iniciativas de fomento ao desenvolvimento de **Pensamento Computacional**, é possível observar que a Cambridge University adota uma abordagem estruturada e progressiva, enfatizando a importância de fundamentos teóricos sólidos combinados com

aplicação prática. A Figure 3.2 apresenta a evolução e relação entre os tópicos de Algoritmos 1 e 2, destacando a progressão do ensino de **Pensamento Computacional** desde conceitos básicos até estruturas de dados avançada

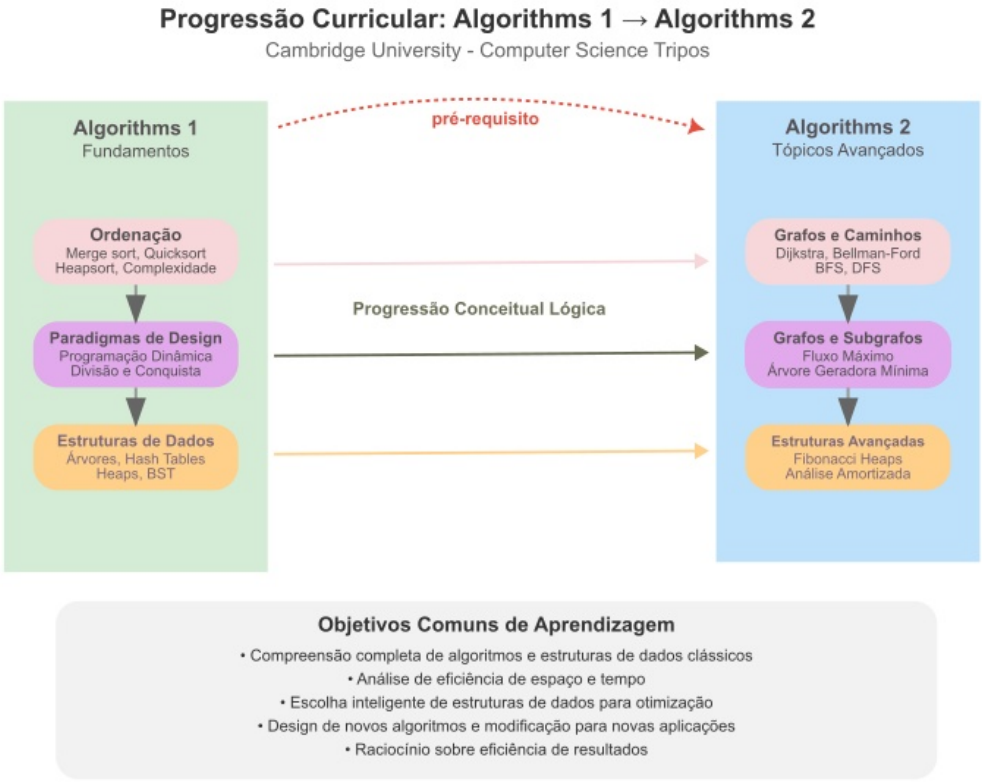


Figure 3.2: Relação entre as disciplinas relacionadas a Raciocínio Algorítmico na Universidade de Cambridge

Comparação das estruturas curriculares das disciplinas de Algoritmos 1 e 2 da Cambridge University. A figura ilustra a progressão do ensino de **Raciocínio Algorítmico** desde conceitos básicos até estruturas de dados avançadas, destacando a relação entre os tópicos abordados nas duas disciplinas.

A **Oxford University** implementa uma filosofia pedagógica que estimula o **Pensamento Computacional** desde os estágios iniciais por meio do design de programas de computador, oferecendo cursos integrados de Ciência da Computação(113), Matemática e Ciência da Computação(114), e Ciência da Computação e Filosofia(113). Esta diversidade de programas interdisciplinares parece refletir uma compreensão de que o **Pensamento Computacional** e o **Raciocínio Algorítmico** podem ser aplicados e desenvolvidos em múltiplos contextos acadêmicos, oferecendo perspectivas únicas e complementares.

Oxford parece estender seu compromisso com o **Pensamento Computacional** por meio de iniciativas específicas de divulgação e engajamento que demonstram uma visão ampla da responsabilidade educacional da universidade. A universidade participa ativamente do *UK Bebras Challenge*, um desafio nacional de **Pensamento Computacional** que visa introduzir esses conceitos aos alunos nos níveis de ensino anterior à graduação (115). A competição, que envolve mais de 100.000 crianças e adolescentes anualmente, promove o desenvolvimento de habilidades de **Pensamento Computacional** por meio de problemas desafiadores e interativos, refletindo uma abordagem prática e aplicada ao ensino desses conceitos fundamentais. Além disso, a universidade também participa do "Oxford University Computing Challenge", um evento de participação por convite que encoraja o desenvolvimento de habilidades de **Pen-**

samento Computacional por meio da aplicação de algoritmos e programas para resolver problemas complexos (116).

Complementarmente, o *Oxford University Computing Challenge (OUCC)*, uma competição por convite que encoraja o desenvolvimento de habilidades de **Pensamento Computacional** por meio da aplicação de algoritmos e programas para resolver problemas complexos (117). Em 2023, mais de 20.000 participantes do *UK Bebras* foram convidados a participar do **OUCC**(84). Estas iniciativas demonstram o reconhecimento de que o **Pensamento Computacional** deve ser cultivado de forma ampla e sistemática, estendendo-se além dos limites tradicionais do ensino superior para impactar toda a sociedade.

Diferenciando-se da Universidade de Cambridge e das universidades dos EUA, a Oxford University não permite acesso prático às ementas dos seus cursos, impedindo uma análise de conteúdo e duração para aqueles que não estão, de alguma forma, associados à instituição.

O **Imperial College London** adota uma abordagem fundamentada em princípios de engenharia para o desenvolvimento do **Pensamento Computacional**, enfatizando princípios fundamentais, pensamento lógico e considerações de engenharia no design de sistemas (118). Esta perspectiva de engenharia distingue a abordagem do Imperial College, focando na aplicação prática do **Pensamento Computacional** em contextos de design e implementação de sistemas reais, preparando os alunos para os desafios práticos que enfrentarão em suas carreiras profissionais. Para corroborar esta integração entre teoria e prática, o Imperial College London promove a iniciativa I-X (119) para fomentar aprendizado e aplicação de tecnologia computacional em contextos práticos e reais.

O programa de graduação em computação integra teoria e prática por meio de uma combinação estruturada de aulas de laboratório, resolução de problemas e trabalhos de projeto e design, criando um ambiente de aprendizagem que espelha as demandas profissionais reais (118). O objetivo desta estrutura parece ser a formação progressiva dos conceitos de **Raciocínio Algorítmico** e **Pensamento Computacional** por meio da aplicação prática em problemas autênticos de engenharia, onde os conceitos emergem naturalmente das necessidades práticas, ao invés de serem ensinados como abstrações isoladas.

Os módulos centrais revelam uma progressão cuidadosamente estruturada que constrói sistematicamente as competências fundamentais: no primeiro ano **Matemática Discreta**, **Lógica e Raciocínio** (7,5 ECTS) e **Grafos e Algoritmos** (7,5 ECTS); no segundo ano **Projeto e Análise de Algoritmos** (7,5 ECTS), além de disciplinas opcionais como **Raciocínio Simbólico** (6 ECTS) e **Técnicas Computacionais** (6 ECTS) (118). Esta estrutura curricular parece induzir a criação do **Raciocínio Algorítmico**. E isso é tudo que se pode afirmar sobre o Imperial College London sobre **Pensamento Computacional** e **Raciocínio Algorítmico** já que não foi possível encontrar estes termos em nenhuma ementa de disciplina online.

A **Cardiff University** distingue-se por sua abordagem explícita e dedicada ao **Pensamento Computacional**, oferecendo uma disciplina específica **Pensamento Computacional** (10 ECTS) no primeiro ano de seu curso de Ciência da Computação (120). Esta disciplina dedicada representa uma abordagem direta e sem ambiguidades para o desenvolvimento dessas competências fundamentais, garantindo que todos os alunos recebam uma base sólida e explícita em **Pensamento Computacional** desde o início de seus estudos. O ensino de **Raciocínio Algorítmico** é complementado por **Resolução de Problemas com Python** (5 ECTS) (121). A estrutura curricular de Cardiff indica um foco precoce e dedicado ao **Pensamento Computacional**, distribuído estrategicamente por meio de múltiplas disciplinas para garantir reforço e aplicação consistente dos conceitos fundamentais.

3.3 Detalhamento das Universidades na Europa Continental

A Comissão Europeia, por meio do seu Plano de Ação para a Educação Digital 2021-2027, reforça a importância da educação em computação como uma prioridade para aprimorar as habilidades e competências digitais (122). Desde 2014, muitos países europeus têm revisado seus currículos de educação obrigatória para introduzir conceitos básicos de Ciência da Computação, preparando o terreno para o desenvolvimento de habilidades de **Pensamento Computacional** (122).

A **ETH Zurich** (Suíça) enfatiza que o **Pensamento Computacional** vai além da programação e envolve as capacidades de abstração (123). Para o ensino exclusivo de **Pensamento Computacional**, a universidade adota um paradigma de sala de aula invertida, *flipped classroom* (64), com vídeos e leituras para autoestudo, disponíveis online para seus alunos. O modelo de sala de aula invertida da ETH Zurich (65) indica que a instituição tem consciência de que as disciplinas convencionais não são capazes de dividir o tempo, de forma justa, entre a aquisição do conhecimento, a consolidação supervisionada e a aplicação concreta deste conhecimento (65). Além disso, existem declarações indicando que o **Pensamento Computacional** é mais do que programar um computador, significa pensar em abstrações (123).

Talvez o comprometimento mais evidente da ETH Zurich com o **Pensamento Computacional** seja a inclusão de um conjunto de tutoriais de programação para iniciantes com a indicação de qual linguagem de programação será mais relevante por curso, incluindo, Python, C++ e Java para cursos que vão da ciência da computação até ciências farmacêuticas, de biologia até ciência de alimentos (124).

A **École hôtelière de Lausanne (EHL)**, renomada instituição suíça de ensino superior, integra um módulo específico de "Pensamento Computacional" em seu **programa de Bacharelado em Administração Hoteleira (EHL HOSPITALITY BUSINESS SCHOOL, 2024)**. Esta disciplina, oferecida no segundo semestre, representa 3,5 ECTS e uma carga horária total de 88 horas, sendo 30 horas de contato direto com o professor e 58 horas destinadas ao estudo independente (125). A inclusão de **Pensamento Computacional** em um programa de administração hoteleira demonstra o reconhecimento crescente de que essas competências transcendem as áreas tradicionais de ciência da computação. A oferta no segundo semestre indica seu papel como disciplina estruturante para o desenvolvimento de habilidades de resolução de problemas. Esta abordagem da EHL alinha-se com as tendências globais de integração explícita do **Pensamento Computacional** nos currículos de graduação, independentemente da área de formação principal (28).

4 Definição e Princípios da Metodologia DAAD

O **Pensamento Computacional**, incluindo os elementos da Metodologia **DAAD**, é considerado uma habilidade básica e estruturante para o desenvolvimento científico, tecnológico e econômico no século XXI (47). Sua relevância se estende além das fronteiras da ciência da computação, sendo essencial para a resolução de problemas em diversas áreas *STEM* (Ciência, Tecnologia, Engenharia e Matemática) e *No-STEM* (51). Profissionais da computação, por sua própria natureza, utilizam intrinsecamente essas habilidades: pensar abstratamente, operar em múltiplos níveis de abstração, abstrair ideias para gerenciar a complexidade, e empregar a iteração, a depuração e o teste de software como práticas rotineiras (29). Diversos estudos e frameworks listam o **Pensamento Algorítmico** ou o Design de Algoritmos como um dos principais componentes ou habilidades do **Pensamento Computacional** (49). Finalmente, o **Raciocínio Algorítmico** é uma habilidade fundamental e um componente central do **Pensamento Computacional** (7). Ele implica a capacidade de compreender, testar, aprimorar ou projetar um algoritmo (49). Este tipo de raciocínio é, de fato, a orientação mental e o processo de pensamento que alicerça a aplicação da Metodologia **DAAD** (14).

Para entender a Metodologia **DAAD**, acrônimo de **P**roblem Decomposition, **A**bstraction, Algorithm **D**esign e **D**ebugging, estrutura-se como uma aplicação pedagógica direta dos componentes fundamentais do **Pensamento Computacional** (29) adaptados para a realidade do ensino de **Raciocínio Algorítmico** nos cursos de graduação no Brasil, especificamente aqueles dos cursos relacionados às áreas de ciências, matemática, computação e engenharia.

Na Metodologia **DAAD** a fase **Decomposição**, decomposição do problema, é a manifestação prática da ação de **Decomposição**, fragmentando desafios complexos em subproblemas gerenciáveis e facilmente descritíveis em estruturas de comandos ou fluxogramas. A fase **Abstração** reflete diretamente o pilar homônimo do modelo **Pensamento Computacional**, filtrando detalhes irrelevantes para isolar padrões essenciais, incorporando implicitamente o *reconhecimento de padrões*. A fase **Algoritmização** materializa o pilar *design algorítmico* do trabalho de (29), transformando abstrações em sequências executáveis, por meio de ferramentas de apoio ao **Raciocínio Algorítmico** como fluxogramas e pseudocódigos, enquanto promove a generalização ao transferir soluções para contextos análogos, porém mais amplos. Finalmente, A fase **Depuração** substitui o processo de modelagem computacional através de refinamento iterativo, validando soluções contra comportamentos esperados do sistema (29).

Nesta proposta, o termo **Depuração** será mantido ampliando seu sentido e uso. Nesta metodologia, a fase **Depuração** irá além do sentido computacional de encontrar erros em programas. A **Depuração** tem um sentido mais amplo e representa purificar substâncias e sistemas, corrigir erros e eliminar elementos indesejados. Assim, ainda que esta seja uma fase própria do ciclo completo, a **Depuração** representa o processo de correção e adaptação que será empregado em todos os processos e fases de ensino usando a Metodologia **DAAD**, **D**ecomposição, **A**bstração, **A**lgoritmização e **D**epuração.

Esta sinergia, entre **Pensamento Computacional** e **Raciocínio Algorítmico** posiciona a Metodologia **DAAD** como uma estrutura operacional para consolidar a teoria do **Raciocínio Algorítmico** em práticas educacionais tangíveis, mantendo a essência holística da proposta original de Wing (2006)(28) e Lehmann (2023)(29).

4.1 Metodologias Semelhantes ao DAAD

A Metodologia **DAAD** se distingue de outras abordagens pedagógicas de **Pensamento Computacional** por sua ênfase na **Depuração** como etapa essencial do processo de aprendizado. A seguir, são apresentadas algumas metodologias concorrentes e suas diferenças em relação ao **DAAD**.

A abordagem dos **Quatro Pilares do Pensamento Computacional**, desenvolvida pela Universidade de York, estrutura-se em decomposição, reconhecimento de padrões, abstração e pensamento algorítmico. Essa metodologia enfatiza a fragmentação de problemas complexos e a identificação de regularidades, mas não formaliza etapas de validação, limitando-se à concepção teórica de soluções (127).

A **Abordagem por Atividades Desplugadas**, proposta por (41), prioriza intervenções pedagógicas sem uso de tecnologia, utilizando recursos físicos, como quebra-cabeças, para desenvolver abstração e decomposição. Embora eficaz em contextos com infraestrutura limitada, sua aplicação tende a focar em modelagem de *hardware* em detrimento do design sistemático de algoritmos, além de carecer de sequencialidade didática clara (127).

A metodologia de Wing (2006) (28), base teórica seminal, define pilares como decomposição, abstração, design algorítmico e generalização. Wing enfatiza que a generalização de soluções permite aplicação em múltiplos contextos, porém não integra explicitamente mecanismos de depuração ou validação, sejam eles iterativos ou não, concentrando-se na fase de concepção em vez do refinamento prático (128). Essas lacunas são supridas pela Metodologia **DAAD**, que incorpora a **D**epuração como etapa fundamental para testar robustez, eficiência e escalabilidade de algoritmos em cenários reais, como em sistemas embarcados ou otimização NP-Difícil[1], consolidando um ciclo completo de desenvolvimento (128). A Table 4.1 resume estas metodologias.

Table 4.1: Propostas metodológicas de aplicação pedagógica dos conceitos de Pensamento Computacional

Metodologia	Instituição/Referência	Componentes-Chave	Diferenças para o DAAD
Quatro Pilares do PC	University of York	1. Decomposição2. Reconhecimento de padrões3. Abstração4. Pensamento algorítmico	• Substitui *Decomposição* por *Reconhecimento de padrões* • Não inclui refinamento iterativo após implementação
Abordagem por Atividades Desplugadas	Brackmann (2022)	• Problemas físicos (ex.: quebra-cabeças) • Modelagem conceitual sem código • Ênfase em abstração	• Foco em *hardware* vs. *solução algorítmica* • Menos estruturada em etapas sequenciais
Metodologia de Wing (2006)	Wing (2006)	• Decomposição • Abstração • Design algorítmico • Generalização	• Não formaliza *Decomposição* como pilar independente • *Generalização* *Validação prática iterativa*

[1] Problema NP-Difícil: Classe de problemas computacionais considerados altamente complexos, cuja solução exata em tempo razoável é improvável. Resolver tais problemas frequentemente requer algoritmos heurísticos ou aproximados.

4.2 Inovações do DAAD

A diferença *sine qua non* do **DAAD** em relação aos outros modelos é a integração explícita de **Depuração** (29). Uma fase do processo de aprendizado dedicada à eliminação e correção de problemas, mas também valida soluções. Essa abordagem de Lehmann (2023) (29) corrige a lacuna de modelos que ignoram o refinamento pós-implementação, promovendo uma compreensão mais profunda e prática do desenvolvimento de algoritmos.

Na Abordagem de York (2020) (126), o reconhecimento de padrões é estático, enquanto no **DAAD** a abstração é dinâmica, preparação para depuração. O modelo original de Wing (2006) (28) trata generalização como resultado, enquanto no **DAAD** a **Depuração** torna-a processual através de iteração. Finalmente, a Abordagem **Desplugada** de Brackmann (2022) (41) enfatiza modelagem física, mantém abstração como conceito teórico, já na Metodologia **DAAD** vincula-se à rastreabilidade durante validação, enquanto a Metodologia **DAAD** prioriza soluções algorítmicas e validação prática.

A estrutura do **DAAD** também inclui uma sequência pedagógica otimizada em ordenação lógica: **Problema → Abstração → Algoritmo → Validação**. O que contrasta com os modelos fragmentados, como as metodologias de York (126) e Wing (28) que não articulam nenhuma ordem de transição entre etapas. Além disso, como o objetivo é criar um *framework* pedagógico para a disciplina de **Raciocínio Algorítmico**, inicialmente, e depois progredir para todas as disciplinas dos cursos de Ciência e Engenharia da Computação, a Metodologia **DAAD** enfatiza a criação de algoritmos robustos e eficientes, alinhando-se às demandas do mercado de trabalho. Portanto, a Metodologia **DAAD** inclui a aplicação de técnicas avançadas de depuração, como testes de estresse e análise de casos limites, que vão além da simples correção de erros sintáticos. Suportando processos de depuração sistemática alinhados às demandas industriais à medida que a complexidade das disciplinas aumenta ao longo do curso.

A relação entre as fases do **DAAD** e a solução de problemas pode ser validada por meio da análise de problemas simples típicos das disciplinas introdutórias dos cursos de Ciência e Engenharia da Computação. A Figure 4.1 apresenta exemplos de problemas simples que podem ser resolvidos usando a Metodologia **DAAD**. Esses problemas são comuns em disciplinas introdutórias e ilustram como cada fase do **DAAD** pode ser aplicada para chegar a uma solução eficaz.

Figure 4.1: Exemplos de problemas simples que podem ser resolvidos usando a metodologia DAAD.

A Metodologia **DAAD** foi concebida para ir além das disciplinas introdutórias e além dos

cursos de Ciência e Engenharia da Computação. O objetivo é que a Metodologia **DAAD** ajude a criar a capacidade de resolver problemas complexos de forma sistemática, promovendo uma compreensão profunda dos processos envolvidos na criação de soluções em qualquer área do conhecimento. Entretanto, como a prova de conceito aqui proposta diz respeito aos cursos de Ciência e Engenharia da Computação, a seguir são apresentados exemplos de aplicação do **DAAD** em contextos mais avançados, que demonstram sua versatilidade e aplicabilidade em problemas complexos e multidisciplinares.

1. **Otimização Algorítmica para Problemas NP-Difíceis**: usando um modelo de decomposição baseado usando o Teorema Cook-Levin para problemas NP-completos, com métricas de validação de Dolan-Moré (129).

- **Decomposição**: fragmentação de problemas de otimização combinatorial (ex.: problema do caixeiro viajante) em subproblemas de roteamento local e conexões inter-regionais, permitindo abordagens *divide-et-conquer*, dividir e conquistar.

- **Algoritmização**: implementação de meta-heurísticas para busca de soluções ótimas em espaços de estado complexos.

- **Depuração**: validação via análise de falhas entre soluções heurísticas e limites teóricos, utilizando perfis de desempenho para avaliação estatística .

2. **Engenharia de Sistemas Embarcados**: usando *framework* de abstração conforme definido por Wilhelm (2008) para sistemas tempo-real, com validação por Henzinger (2007), teremos:

- **Abstração**: modelagem de invariantes temporais e de consumo energético através de **W**orst-**C**ase **E**xecution **T**ime, **WCET**, e máquinas de estado de consumo, filtrando variáveis não essenciais ao cumprimento de restrições rígidas de tempo real.

- **Algoritmização**: síntese de algoritmos com garantias formais (ex.: controle PID com provas de estabilidade usando a Função de Lyapunov (130)) atendendo a requisitos de segurança.

- **Depuração**: verificação **H**ardware-**I**n-the-**L**oop, **HIL**, com injeção de falhas sistêmicas e análise de violações de datas limites via acompanhamento e registro, em tempo real.

3. **Desenvolvimento de Pipeline de Aprendizado de Máquina**: usando com referência o trabalho de Geron (2019) (131), que descreve a construção de sistemas de aprendizado de máquina como um processo iterativo e sistemático, teremos:

- **Decomposição**: fragmentação do fluxo de trabalho em coleta de dados, pré-processamento, seleção de modelo, treinamento e avaliação.

- **Abstração**: identificação de características essenciais nos dados (ex.: seleção de funcionalidades via análise de componente principal, PCA, ou análise de importância) .

- **Algoritmização**: projeto de arquitetura de redes neurais ou ajuste de hiperparâmetros de algoritmos de classificação.

- **Depuração**: validação cruzada, análise de superajuste, *overfitting* e ajuste de modelos com base em métricas de precisão.

4. **Desenvolvimento de Sistemas Concorrentes**: usando como referência o trabalho de Herlihy (2015)(132), que aborda a construção de sistemas concorrentes e distribuídos, teremos:

- **Decomposição**: divisão do sistema em tarefas concorrentes (ex.: *threads* para E/S, processamento e comunicação).

- **Abstração**: isolamento de seções críticas e recursos compartilhados (ex.: *buffer* de mensagens).

- **Algoritmização**: criação de protocolos de sincronização (ex.: semáforos, *mutexes*) e algoritmos de exclusão mútua.

- **Depuração**: teste de estresse para detecção de *deadlocks* e condições de corrida com ferramentas como *Valgrind* ou *TSan* .

5. **Desenvolvimento de Sistemas Distribuídos**: este caso requer a aplicação dos princípios de decomposição hierárquica e depuração em falhas não determinísticas, seguindo o modelo de Tanenbaum (2015) (133) para sistemas distribuídos.

- **Decomposição**: divisão de sistemas complexos em microsserviços independentes (ex.: separação de módulos de autenticação, processamento de pagamentos e gerenciamento de dados) para tratamento paralelo.
- **Algoritmização**: implementação de protocolos de consenso como *Raft* ou *Paxos* para garantir consistência entre nós distribuídos.
- **Depuração**: simulação de falhas em cascata e testes de partição de rede com ferramentas como *Chaos Monkey* para validar resiliência(134).

4.3 Metodologia DAAD: Estrutura e Fases

A Metodologia **DAAD** é composta por quatro fases principais: **Decomposição**, **Abstração**, **Algoritmização** e **Depuração**. Cada fase desempenha um papel importante na resolução de problemas complexos, permitindo que os alunos desenvolvam habilidades de raciocínio e análise. A seguir, são apresentadas as fases da Metodologia **DAAD** e suas respectivas definições.

A Metodologia **DAAD** pode ser representada visualmente como um ciclo contínuo, no qual cada fase se conecta à próxima, formando um processo iterativo de resolução de problemas. A Figure 4.2 ilustra essa representação, destacando as fases de **Decomposição**, **Abstração**, **Algoritmização** e **Depuração**.

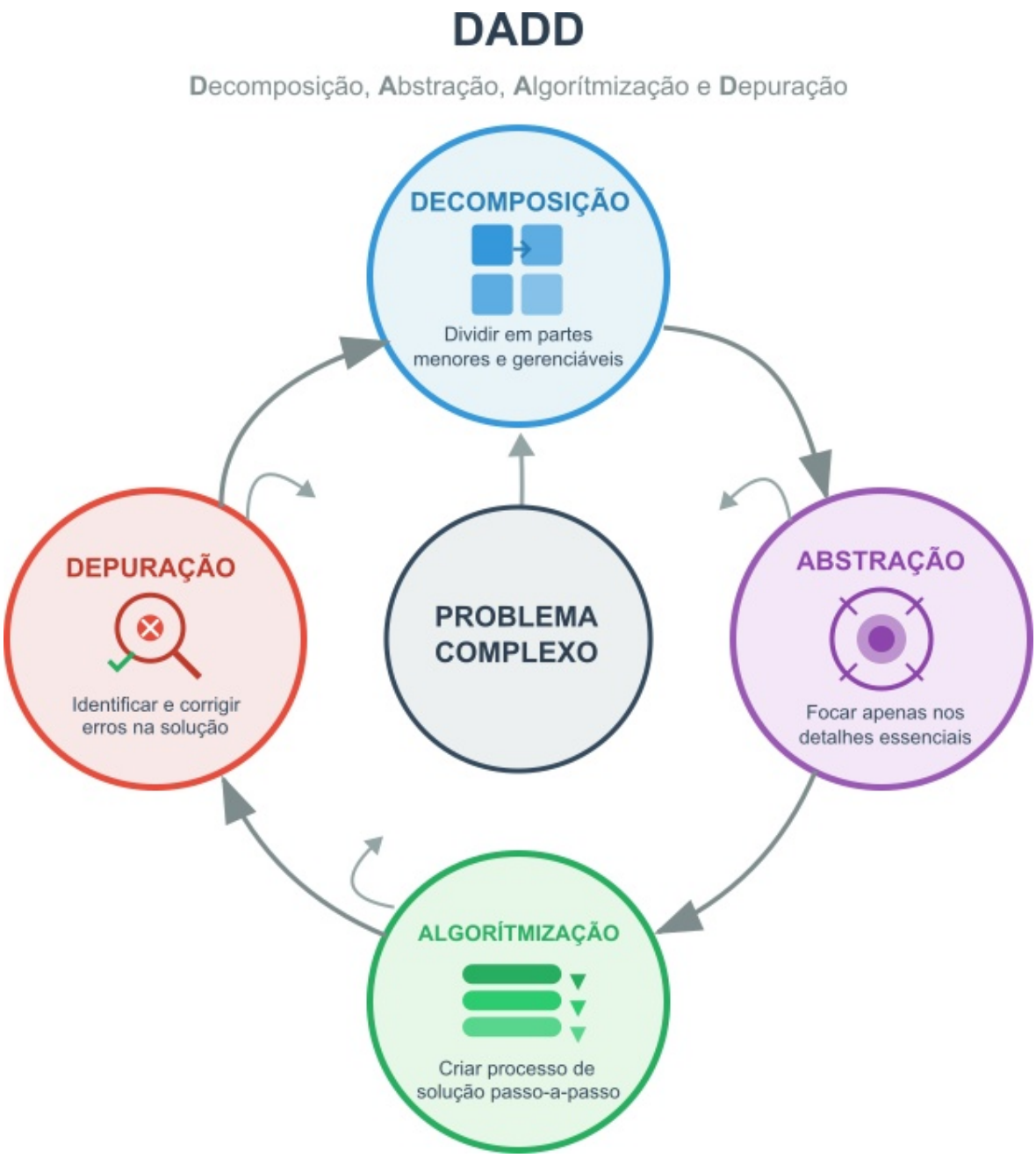


Figure 4.2: Representação visual da metodologia DAAD.

A representação visual da Metodologia **DAAD** enfatiza a natureza cíclica e interconectada do processo de resolução de problemas. Porém, contém uma deficiência, pois não representa a natureza iterativa da metodologia; talvez uma espiral fosse mais adequada. Contudo, como a espiral é, em si própria, uma representação complexa, a representação em ciclo parece mais adequada para o ensino inicial da Metodologia **DAAD**. A característica iterativa e cíclica da Metodologia **DAAD** pode ser percebida por meio do detalhamento das fases da Metodologia **DAAD**.

4.3.1 Decomposição: Quebrando a Complexidade

A **D**ecomposição é o processo analítico pelo qual problemas ou conceitos complexos são divididos em partes menores e mais gerenciáveis (29). Esse passo é necessário para a construção de uma resolução eficaz de problemas, transformando situações que inicialmente parecem

complexas e limitantes em elementos mais simples e acessíveis (58). Miller (1956) descobriu que a memória humana está limitada a 7 ± 2 elementos, o que implica que a decomposição de informações complexas em partes menores pode facilitar a compreensão e a retenção (135). Ao dividir um problema grande em subproblemas, torna-se mais fácil analisar cada componente isoladamente e, posteriormente, integrar as soluções para resolver o problema original (29).

Dividir o problema em partes menores, fragmentos, é o primeiro problema que o aluno precisa resolver usando a própria Metodologia **DAAD**.

Uma das formas de transformar os fragmentos resultantes do processo de decomposição é a modelagem e análise. Um modelo funciona como uma representação abstrata de situações do mundo real (136) e é útil em soluções de engenharia e computação. O processo de modelagem começa com o reconhecimento de padrões e a correspondência de padrões.

Estes processos envolvem a identificação de similaridades e diferenças entre as partes menores do problema. Esta identificação de similaridades e diferenças ajuda a categorizar os fragmentos do problema em uma sequência específica (47). No contexto da **Decomposição**, a correspondência de padrões oferece oportunidades interessantes. Os padrões descobertos podem ser transferidos para problemas similares, evitando reiniciar processos do zero.

O próximo passo é a generalização dos fragmentos, mediante a eliminação de detalhes irrelevantes (98). Aqui, dentro da **Decomposição** surge um processo de **Abstração** baseado na lógica indutiva. A lógica indutiva baseia-se no princípio de que conclusões gerais podem ser derivadas de observações e experiências empíricas. Como essas conclusões indutivas contêm incerteza inerente, seu grau de confiabilidade depende do número de observações realizadas (137). Contudo, o uso da indução permite a formulação de regras gerais a partir de casos individuais e permite a definição apurada dos fragmentos em que o problema deve ser resolvido. Neste ponto, a **Decomposição** exige a dedução. A dedução requer conhecimento das regras e limites do problema que está sendo resolvido. A essência do uso da **Abstração** na decomposição reside na eliminação de detalhes irrelevantes e no foco em elementos essenciais, simplificando estruturas complexas e permitindo ênfases específicas (51).

A definição das partes resultantes da **Decomposição** requer que os fragmentos encontrados sejam validados. A validação é o processo de garantir que as partes do problema sejam adequadas para a solução do problema original e que funcionem em conjunto. Para isso, é importante organizar as partes em uma forma que permita a avaliação. Esta organização é chamada de **Algoritmização**. A validação pode ser feita por meio de testes, simulações ou outras técnicas que garantam que as partes sejam adequadas para a solução do problema original (29). Este processo de validação é, na verdade, a aplicação dos conceitos de **Depuração**.

No contexto da engenharia, a decomposição é empregada para desmembrar tarefas complexas, como a simulação de construção, em segmentos menores e mais fáceis de gerenciar (49). Na ciência da computação, a **Decomposição** visa quebrar um problema ou sistema complexo em partes que são mais fáceis de conceber, entender, programar e manter (83).

Exemplos clássicos *unplugged* incluem a divisão da tarefa de fazer um bolo em etapas menores, como preparar a massa, assar o bolo, fazer a cobertura e aplicá-la. Todos os exemplos de tarefas diárias, como fazer um sanduíche, montar um quebra-cabeça ou organizar uma festa, podem ser vistos como exemplos de **Decomposição**. Desmembrar uma tarefa como fazer um bolo, ou outras tarefas diárias, em etapas menores e mais gerenciáveis busca facilitar a execução e o entendimento do processo como um todo, o que reflete diretamente o propósito da **Decomposição**: quebrar um problema ou sistema complexo em partes que são mais fáceis de conceber, entender, programar ou manter (28).

4.3.2 Abstração: Focando no Essencial

A **A**bstração é uma técnica central do **Pensamento Computacional** que se concentra em identificar informações importantes e relevantes, enquanto ignora detalhes desnecessários ou irrelevantes, facilitando uma compreensão mais clara das questões essenciais (47) referentes ao problema que precisa ser resolvido. A abstração é o processo de generalizar detalhes concretos para direcionar a atenção para aspectos de maior importância (98).

A abstração é considerada o processo de pensamento de mais alto nível no **Pensamento Computacional**, conferindo a capacidade de escalar e gerenciar a complexidade (98). A sua presença é tão difundida na ciência da computação que a sua descrição concisa é um desafio, e embora haja um consenso sobre a sua centralidade (47), as definições exatas podem variar entre os investigadores. Porém, unindo as definições, podemos afirmar que:

A **Abstração** é o processo cognitivo de redução da complexidade que envolve a criação de representações gerais de processos ou grupos de objetos mediante a remoção sistemática de informações desnecessárias e o foco seletivo nos elementos essenciais de uma situação específica (69). Este processo permite a formulação de afirmações gerais que resumem exemplos particulares sobre conceitos, procedimentos, relações e modelos subjacentes (29), constituindo uma estratégia fundamental para lidar com a complexidade (138).

A **Abstração** caracteriza-se pela capacidade de identificar padrões e realizar generalizações a partir de instâncias específicas, criando representações que sejam não apenas apropriadas para o propósito imediato, mas também reutilizáveis em contextos diferentes (58). Esta habilidade cognitiva permite que soluções desenvolvidas para problemas específicos possam ser adaptadas e aplicadas a situações análogas, otimizando o processo de resolução de problemas e promovendo a transferência de conhecimento entre domínios distintos.

No contexto do **Raciocínio Algorítmico** e do **Pensamento Computacional**, a **Abstração** funciona como uma ferramenta indispensável para a simplificação de estruturas complexas e terá impacto na **Decomposição** e **Algoritmização**. A **Abstração** permite que os solucionadores de problemas concentrem seus recursos cognitivos nos aspectos mais relevantes e significativos de uma situação, descartando detalhes que não contribuem para a compreensão ou resolução do problema em questão (51).

Como a **abstração** também é caracterizada como a habilidade de construir representações utilizando os componentes essenciais que demonstram o funcionamento do problema ou sistema (29). Exemplos ilustrativos, *unplugged*, de **Abstração** incluem o uso de mapas, que simplificam o mundo real ao omitir detalhes desnecessários e, principalmente **fluxogramas**.

Os fluxogramas funcionam como ferramentas de abstração porque simplificam processos complexos em representações visuais que eliminam detalhes de implementação específicos. Além disso, os fluxogramas identificam padrões estruturais (sequências, decisões, repetições) e removem informações desnecessárias como sintaxe específica, permitindo que programadores se concentrem no essencial do problema. Finalmente, fluxogramas podem ser realizados completamente *unplugged*, sem o uso de computadores, o que os torna uma ferramenta acessível para ensinar abstração e raciocínio algorítmico utilizando as vantagens cognitivas do ensino *unplugged*(63). O uso de fluxogramas induz a **Algoritmização**.

4.3.3 Algoritmização: Desenvolvendo Soluções Sistemáticas

A **A**lgoritmização, ou Design de Algoritmos para Wing (2006) (28), envolve a criação de instruções passo a passo ou procedimentos para resolver um problema. Um algoritmo é definido como uma sequência finita de instruções matematicamente rigorosas, tipicamente utilizadas para resolver uma classe específica de problemas ou para realizar uma computação (67). Para fazer um algoritmo, é necessário decompor o problema em partes menores, identificar padrões e abstrair os detalhes irrelevantes, seguindo as etapas da Metodologia **DAAD**. A **Algoritmização** é a fase em que as soluções são formalizadas e estruturadas, permitindo que sejam

implementadas de forma sistemática e eficiente (29).

A **Algoritmização** baseia-se na premissa de que as soluções para os problemas não se limitam a respostas pontuais, mas sim a algoritmos que podem fornecer respostas sempre que necessário para casos gerais (45). É o processo de construir um esquema de passos ordenados que podem ser seguidos para fornecer soluções para todos os problemas constituintes necessários para resolver o problema original. A capacidade de expressar uma solução na forma de um algoritmo demonstra uma compreensão mais profunda do problema (48). A eficiência, em termos de velocidade ou uso de memória, é uma consideração importante na criação de algoritmos (48) sempre que o objetivo final for transformar o algoritmo em uma solução computacional.

As abstrações necessárias ao entendimento de algoritmos podem ser expressas de várias formas, incluindo linguagem natural, pseudocódigo, fluxogramas e diagramas (67). Neste contexto, destacam-se os fluxogramas e o pseudocódigo como ferramentas de apoio.

O pseudocódigo atua como uma ponte entre a abstração visual e a escrita de código real, descrevendo algoritmos em linguagem natural estruturada. O pseudocódigo facilita a compreensão de estruturas condicionais (14). Expressar um processo como um algoritmo, o que o pseudocódigo permite, força a precisão na formulação e leva a uma compreensão mais profunda do problema do que outros meios de pensamento tradicionais (14).

O fluxograma permite que os alunos visualizem estruturas de controle como condicionais e laços, além do fluxo de dados e pontos de parada. Um exemplo prático envolve representar a tomada de decisão para identificar se um número é positivo ou negativo, conectando etapas como leitura do valor, verificação da condição e saída do resultado em uma sequência gráfica. A Figure 6.1 ilustra um exemplo de fluxograma que pode ser usado para ensinar a **Algoritmização** em uma ferramenta que remove o fator *unplugged* porém permite a **Depuração** independente (139).

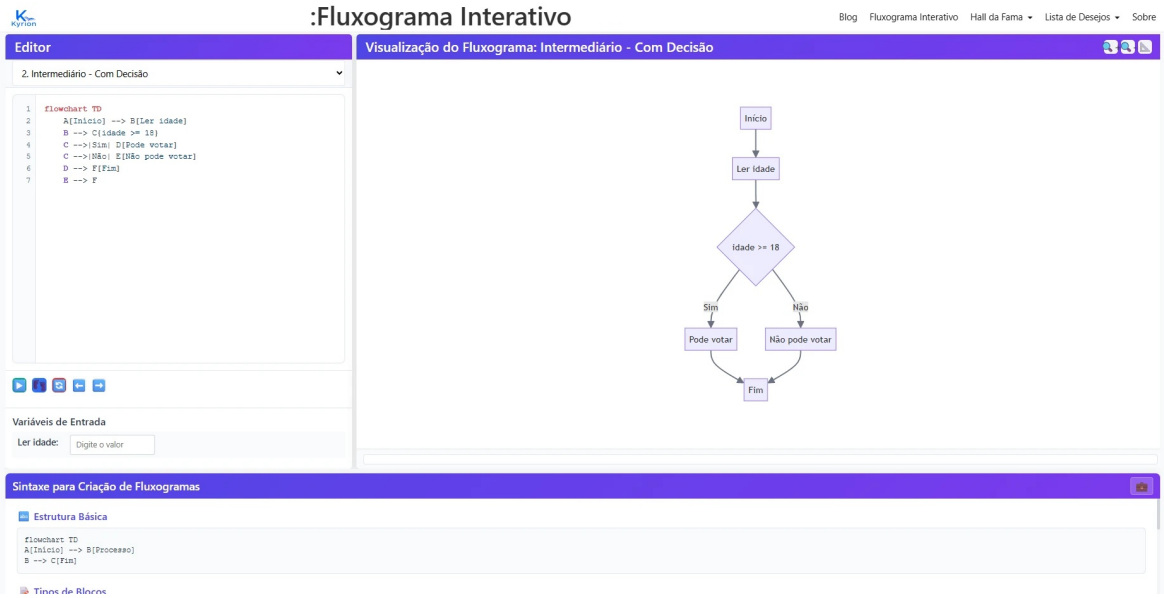


Figure 4.3: Exemplo de fluxograma que pode ser usado para ensinar a Algoritmização.

4.3.4 Depuração: Identificando e Corrigindo Erros

A **D**epuração é a fase responsável por encontrar e remover erros, imperfeições ou elementos indesejáveis durante todas as fases da Metodologia **DAAD**. Na computação, a depuração, *debugging*, é o processo de identificar, isolar e corrigir erros em programas de computador, garantindo que eles funcionem conforme o esperado (67).

O processo de depuração envolve a identificação do erro, a análise de sua causa, registrando as mudanças de estado do programa e os valores dos dados, a correção do problema e a validação da correção por meio de testes (138). É uma atividade iterativa que contribui para a construção do conhecimento e o aprendizado de estratégias de resolução de problemas (140).

O trabalho de Brennan e Resnick (2012) (140) estabelece a **Depuração** como uma das práticas computacionais fundamentais. Esta abordagem reconhece debugging não como habilidade isolada, mas como componente integrado do processo de resolução de problemas. A **Depuração** pode ser sintetizada em cinco etapas: validar; identificar; representar; localizar e corrigir. Esta sistematização oferece framework pedagógico aplicável a diferentes faixas etárias, desde educação infantil até ensino superior, com adaptações apropriadas para cada nível de desenvolvimento cognitivo (140).

Nas fases de **Decomposição** e **Abstração**, a **Depuração** atua como um processo de refinamento contínuo. Cada fase requer uma técnica específica para identificar e corrigir problemas, garantindo que o processo de resolução de problemas seja robusto e eficiente. Por exemplo, na **Decomposição**, a depuração pode envolver a verificação da clareza e viabilidade dos subproblemas identificados. Na **Abstração**, a depuração pode se concentrar na eliminação de detalhes irrelevantes que possam obscurecer a compreensão do problema. Nestes dois casos, a representação de conceitos em palavras simples pode ser a forma mais eficiente.

Neste cenário, fluxogramas oferecem representação visual clara da lógica algorítmica, facilitando identificação de caminhos de execução problemáticos. Estratégias eficazes incluem a análise visual através do rastreamento do fluxo do algoritmo identificando pontos de falha. Tanto o fluxograma quanto o pseudocódigo são ferramentas valiosas para a **Depuração**, pois permitem que os alunos visualizem a lógica do algoritmo e identifiquem possíveis erros de forma mais intuitiva. Tabelas de rastreamento, onde os alunos registram valores de variáveis e estados do programa durante a execução, são úteis para identificar erros lógicos e comportamentais.

4.4 Ferramentas de Abstração na Metodologia DAAD

As ferramentas de abstração e apoio à Metodologia **DAAD** incluem uma variedade de recursos visuais e interativos que facilitam a compreensão e aplicação das fases do processo. Algumas dessas ferramentas são:

4.4.1 Fluxogramas

Representações visuais que ilustram a lógica do algoritmo, permitindo identificar facilmente os passos e decisões envolvidas. Historicamente, o fluxograma — uma representação visual padronizada de processos — emergiu como uma das principais ferramentas para construir essa ponte entre o pensamento humano e a lógica computacional na década de 1960. No entanto, sua prevalência e utilidade são temas de um debate contínuo e vigoroso na academia, refletindo uma tensão mais ampla entre a teoria da carga cognitiva, a necessidade de aprendizado e a rápida evolução dos paradigmas de programação. A persistência dos fluxogramas em contextos educacionais é sustentada por vantagens pedagógicas claras, validadas por estudos empíricos.

- **Redução da Carga Cognitiva**: A principal vantagem é a separação da lógica da sintaxe. Ferramentas visuais como os fluxogramas reduzem a carga extrínseca, permitindo que os alunos dediquem seus recursos cognitivos à tarefa mais fundamental de estruturar uma solução (141).
- **Clareza Visual e Desempenho Superior em Compreensão**: Um estudo empírico de 2022, que utilizou rastreamento ocular, descobriu que, ao analisar algoritmos, os

participantes foram significativamente mais rápidos, cometeram menos erros e tiveram maior confiança em suas soluções ao usar fluxogramas estruturados em comparação com o pseudocódigo (142).

- **Alavanca para a Programação Textual (Scaffolding)**: Pesquisas recentes demonstram que a habilidade de resolver problemas usando fluxogramas é um forte preditor da habilidade de resolver os mesmos problemas em uma linguagem textual como o Python (141).
- **Aplicações Modernas e Inovadoras**: A defesa moderna dos fluxogramas não se concentra no desenho manual, mas em sua integração em ambientes de software interativos. Ferramentas como o *Progranimate* permitem que os alunos construam fluxogramas dinâmicos que são sincronizados visualmente com o código gerado e uma animação da execução do programa (141).

A crítica mais contundente ao uso de fluxogramas surge da sua inadequação aos paradigmas modernos. Os fluxogramas são inerentemente procedurais e sequenciais. Eles são inadequados para representar paradigmas como Programação Orientada a Objetos (POO), recursividade e concorrência (143). Para modelar sistemas complexos e orientados a objetos, a indústria e a academia utilizam a Linguagem de Modelagem Unificada (UML). Apesar disso, a evolução histórica das ferramentas de ensino de programação, desde os fluxogramas e pseudocódigo, passando pelas linguagens baseadas em blocos como o Scratch, até aos modernos assistentes de IA, pode ser vista como uma busca contínua por um ponto ótimo de abstração. Cada geração de ferramentas tenta minimizar a carga cognitiva extrínseca (relacionada com a sintaxe e o ambiente de desenvolvimento) sem sacrificar a carga cognitiva essencial, o próprio **Raciocínio Algorítmico** (22). Esta trajetória não é aleatória; é uma otimização pedagógica impulsionada pela teoria da carga cognitiva, que procura libertar os recursos mentais do aluno para que se possam concentrar no que é verdadeiramente fundamental: a arte de resolver problemas.

A principal vantagem pedagógica do fluxograma reside na sua clareza visual. Ao representar um algoritmo como um diagrama, ele transforma um processo temporal e abstrato num artefacto espacial e concreto, permitindo que os alunos visualizem um constructo físico que representa o fluxo de controle, os pontos de decisão e a sequência de operações. Para isso, na Metodologia **DAAD**, como proposta neste documento, os alunos poderão utilizar apenas 4 dos símbolos usados para fluxogramas (144), como pode ser visto na Figure 4.4.

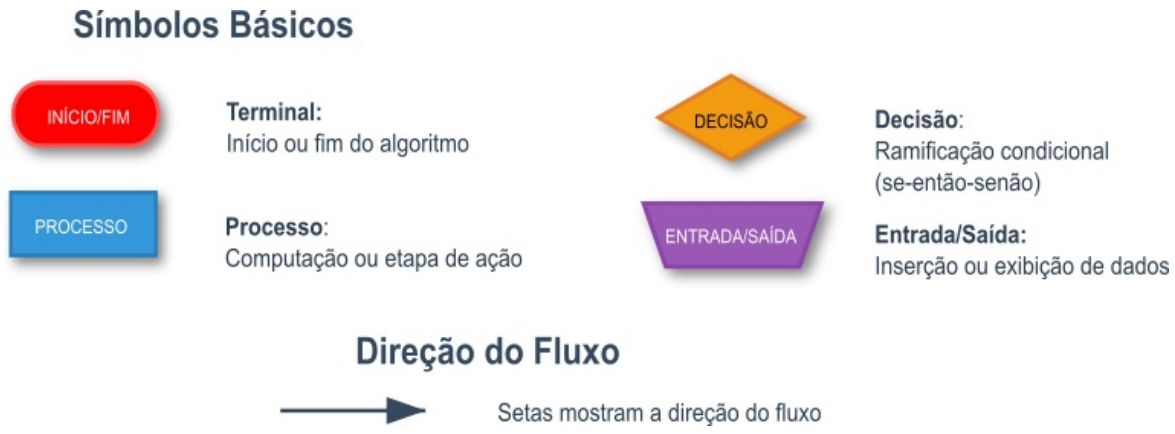


Figure 4.4: Símbolos usados para fluxogramas na metodologia DAAD.

A simplicidade dos símbolos utilizados na Metodologia **DAAD**, em relação aos símbolos tradicionais de fluxogramas(145), permite que os alunos se concentrem na lógica do algoritmo,

e internalizem as abstrações básicas para a solução de problemas usando o **Raciocínio Algorítmico**. Algumas pesquisas corroboram a eficácia dos fluxogramas na educação de programação:

- **Eficiência na Compreensão**: uma quantidade significativa de evidências empíricas sugere que os fluxogramas são superiores ao pseudocódigo em tarefas de compreensão de algoritmos. Estudos controlados, alguns utilizando tecnologia de *eye-tracking* para medir objetivamente a atenção, descobriram que os participantes analisavam algoritmos representados por fluxogramas de forma significativamente mais rápida e com menos erros do que quando representados por pseudocódigo (146).
- **Impacto da Complexidade**: a vantagem dos fluxogramas parece aumentar com a complexidade do algoritmo. Para problemas mais complexos, a diferença no tempo de análise e na taxa de erro a favor dos fluxogramas torna-se ainda mais pronunciada (146).
- **Confiança e Preferência do Aluno**: os alunos não só têm um desempenho melhor com fluxogramas, como também relatam sentir-se mais confiantes nas suas soluções e expressam uma clara preferência pela representação visual. Num estudo seminal de Scanlan (1989) (147), os alunos que usaram fluxogramas demonstraram uma melhor compreensão, maior confiança, cometeram menos erros e necessitaram de menos tempo de aprendizagem em comparação com os que usaram pseudocódigo (148).
- **Base Cognitiva**: acredita-se que a representação visual e espacial dos fluxogramas estimula ambos os hemisférios cerebrais, enquanto o pseudocódigo, sendo textual, estimula predominantemente o hemisfério esquerdo, associado ao processamento lógico e linguístico. Esta estimulação mais holística pode tornar os fluxogramas mais acessíveis a uma gama mais vasta de estilos de aprendizagem (149).

A Figure 4.5 ilustra um exemplo de fluxograma que pode ser usado para ensinar a **Abstração** e a **Algoritmização** em uma ferramenta que remove o fator *unplugged* porém permite a **Depuração** independente.

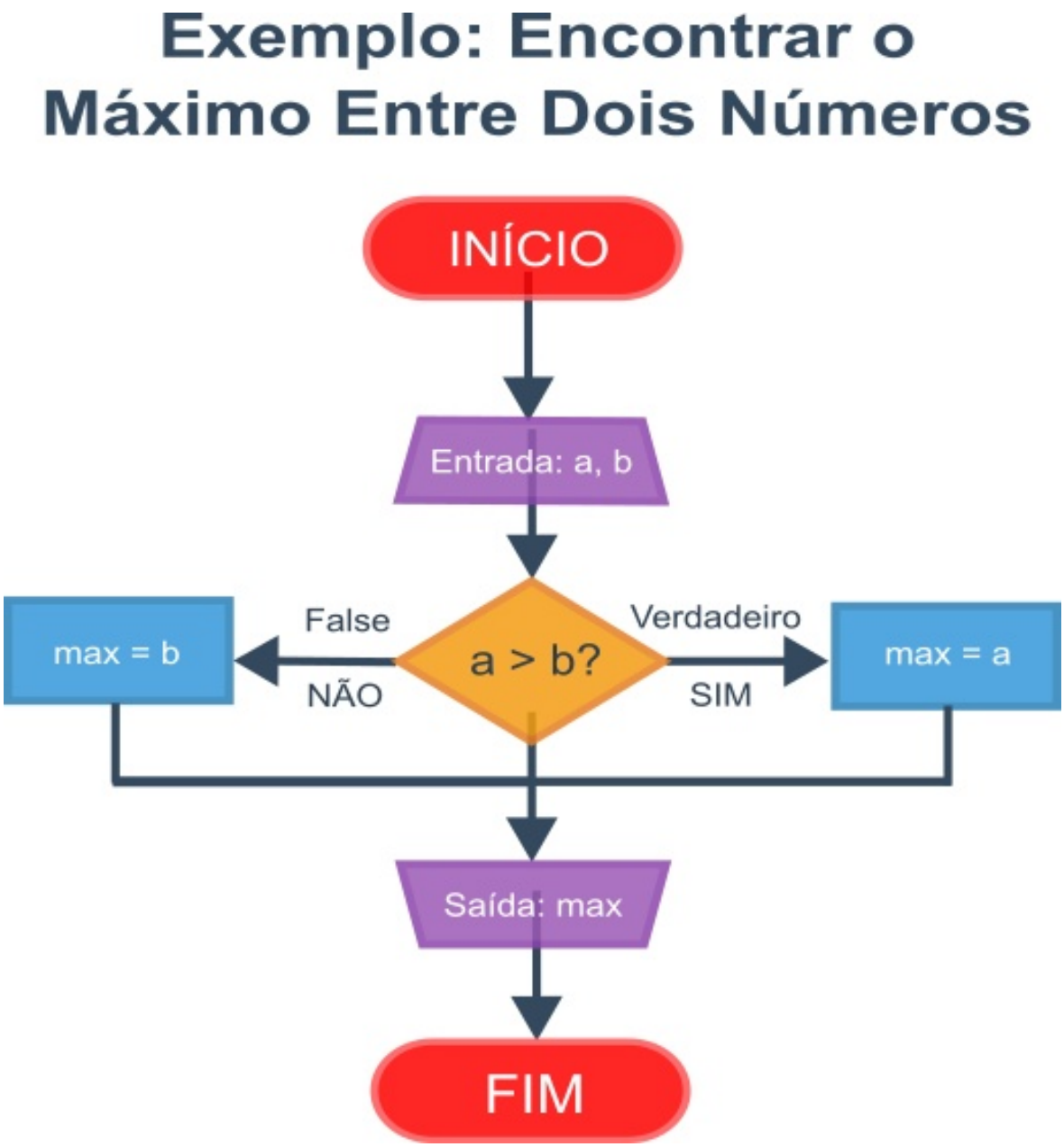


Figure 4.5: Exemplo de fluxograma que pode ser usado para ensinar a Abstração e a Algoritmização.

A análise da literatura e da prática curricular revela um paradoxo notável. Por um lado, a investigação empírica, particularmente estudos controlados recentes, demonstra que os fluxogramas podem ser uma ferramenta de abstração eficaz para programadores iniciantes. Eles superam consistentemente o pseudocódigo em tarefas de compreensão visual, reduzem o tempo de aprendizagem e a taxa de erros, e aumentam a confiança dos alunos (146). Por outro lado, esta eficácia teórica contrasta fortemente com sua ausência quase total na prática pedagógica das principais universidades pesquisadas neste estudo (95). Estas instituições, que definem as tendências na educação em ciência da computação, relegaram os fluxogramas a um papel secundário em favor da codificação direta em linguagens de alto nível como Python (71), ou do uso de linguagens baseadas em blocos como o Scratch (53) como um passo introdutório. E aqui reside o dilema que o *framework* proposto por este estudo para a Metodologia **DAAD** pretende resolver usando as vantagens cognitivas dos fluxogramas sem sacrificar a carga cognitiva necessária aos paradigmas modernos.

A solução está no equilíbrio. A pedagogia de **Raciocínio Algorítmico** não deve adotar uma rejeição total dos fluxogramas, mas sim um reposicionamento estratégico do seu papel no currículo. Em vez de serem vistos como uma ferramenta universal para o design de algoritmos, devem ser empregados como uma ferramenta de nicho, com um propósito pedagógico específico e limitado. O seu uso ideal é como uma ferramenta introdutória e transitória para alunos iniciantes. O objetivo deve ser o de visualizar explicitamente as três estruturas de controlo fundamentais da programação estruturada: atribuição, decisão e iteração (laços).

4.4.2 Pseudocódigo

O pseudocódigo é uma forma de descrever algoritmos em linguagem natural, que ajuda os alunos a entender a lógica sem se preocupar com a sintaxe de uma linguagem de programação específica. A sua estrutura textual do pseudocódigo assemelha-se à estrutura de uma linguagem de programação real, o que pode, teoricamente, facilitar a transição final para a codificação. Além disso, o pseudocódigo parece ser mais adequado para descrever algoritmos complexos em detalhe, onde um fluxograma se tornaria visualmente poluído e ilegível (150).

Tipicamente, o pseudocódigo utiliza uma linguagem simples e direta, evitando jargões técnicos. Muitas vezes, é escrito em uma mistura de inglês simples, ou português, com uma estrutura semelhante à de programação, o que o torna acessível tanto para programadores quanto para não programadores. Como o exemplo podemos criar um pseudocódigo baseado no fluxograma da Figure 4.5:

Listing 4.1

```
ALGORITMO EncontrarMaximo
INÍCIO
    // Entrada dos dados
    LEIA a
    LEIA b

    // Processo de decisão
    SE a > b ENTÃO
        max ← a
    SENÃO
        max ← b
    FIM SE

    // Saída do resultado
    ESCREVA max
FIM
```

O pseudocódigo, como o apresentado na Listing 4.1, deve ser independente de linguagem, o que significa que pode ser convertido para qualquer linguagem de programação. Foca na lógica do algoritmo, não na sintaxe de uma linguagem de programação específica, mas permite a inclusão de estruturas de controle como loops, condicionais e variáveis, facilitando a compreensão dos conceitos fundamentais de programação. Necessariamente precisa ser fácil de ler e entender.

Além da vantagem clara de interpretação, sendo baseado em texto, o pseudocódigo é inerentemente mais fácil e rápido de escrever e modificar (150). Apesar da forte evidência a favor dos fluxogramas em tarefas de compreensão, alguns estudos não encontraram diferenças estatisticamente significativas no desempenho dos alunos, sugerindo que a eficácia de cada ferramenta pode ser dependente da tarefa específica (146).

O *framework* proposto por este estudo para a Metodologia **DAAD** propõe o uso do pseudocódigo como uma ferramenta de abstração complementar aos fluxogramas. O pseudocódigo pode ser usado para descrever algoritmos mais complexos, onde a clareza visual dos fluxogramas pode não ser suficiente. A combinação de fluxogramas e pseudocódigo permite que os alunos desenvolvam uma compreensão mais profunda dos conceitos de programação, aproveitando as vantagens de ambas as ferramentas. O objetivo é usar o pseudocódigo para fazer a ponte para o pensamento textual; e, finalmente, introduzir uma linguagem de programação completa.

4.4.3 Tabelas de Rastreio

As Tabelas de Rastreio são estruturas que permitem aos alunos registrar e acompanhar os valores das variáveis e os estados do programa durante a execução, facilitando a identificação de erros. Também são conhecidas como tabelas de rastreio, tabelas de execução, tabelas de acompanhamento ou teste de bancada. Elas são uma ferramenta valiosa para o ensino do **Raciocínio Algorítmico** e da **Depuração**, pois permitem que os alunos visualizem o fluxo de controle e as mudanças de estado do programa de forma sistemática. Estas tabelas funcionam como uma simulação manual da execução de um programa, permitindo acompanhar o estado das variáveis e o fluxo de controle passo a passo.

Basicamente, uma tabela de rastreio é composta por:

- **Colunas para variáveis**: cada variável do programa tem sua própria coluna;
- **Coluna de linha/instrução**: indica qual linha do código está sendo executada;
- **Colunas de saída**: registram valores impressos ou exibidos;
- **Linhas**: cada linha representa um momento na execução.

O processo de construção de uma tabela de rastreio envolve a execução do código linha por linha, registrando as mudanças nas variáveis e quaisquer saídas produzidas. Isso permite que os alunos vejam como o estado do programa evolui ao longo do tempo e identifiquem onde ocorrem erros ou comportamentos inesperados. Para entender tabelas de rastreio, considere que o fluxograma da Figure 4.5 representado pelo pseudocódigo da Listing 4.1 pode ser executado por um programa com os dados de entrada `a = 15`, `b = 8`. Neste caso, a tabela de rastreio seria:

Table 4.2: Tabela de rastreio de exemplo para o pseudocódigo da Listing 4.1.

Linha	a	b	a > b	max	Saída	Observações
1	-	-	-	-	-	Início do algoritmo
2	15	-	-	-	-	Leitura do primeiro valor
3	15	8	-	-	-	Leitura do segundo valor
4	15	8	True	-	-	Avaliação da condição: 15 > 8
5	15	8	True	15	-	Executa ramo verdadeiro
9	15	8	True	15	15	Exibe o resultado
10	15	8	True	15	15	Fim do algoritmo

A pesquisa do grupo de trabalho ***I**nnovation and **T**echnology in **C**omputer **S**cience*

***E**ducation*, **ITiCSE**[2], forneceu evidências interessantes sobre visualização e engajamento. O estudo marco *Exploring the Role of Visualization and Engagement in Computer Science Education*[3] estabeleceu que tabelas de rastreio requerem engajamento ativo do aluno para serem educacionalmente valiosas, com resultados de aprendizado correlacionados diretamente aos níveis de engajamento (151). Além disso, O artigo IEEE de 2023 *Generating Trace Table for Java Programs*[4] estendeu conceitos de tabela de rastreio usando implementações baseadas em Excel projetadas em torno de diagramas de sequência UML, facilitando a compreensão de comportamento dinâmico incluindo polimorfismo e ligação dinâmica (152). As duas pesquisas parecem destacar a eficiência de tabelas de rastreio como ferramenta de ensino, notadamente para **Depuração** e **Abstração**. Finalmente os cursos de introdução de Stanford combinam tabelas de rastreio com abordagens sistemáticas de depuração(103). A metodologia de Stanford advoga por rastreamento estratégico em vez de exaustivo, reconhecendo que rastreamentos completos de programa são frequentemente impraticáveis para sistemas complexos.

Essas ferramentas são essenciais para apoiar a Metodologia **DAAD**, pois proporcionam representações claras e acessíveis dos conceitos envolvidos, facilitando a compreensão e a aplicação das fases do processo de resolução de problemas e permitem a interação *unplugged* com os conceitos de **Raciocínio Algorítmico**. A combinação de fluxogramas, pseudocódigo e tabelas de rastreio oferece uma abordagem abrangente para o ensino do **Pensamento Computacional**, permitindo que os alunos desenvolvam habilidades essenciais de resolução de problemas e programação adequadas às suas capacidades individuais.

[2]em tradução livre, Inovação e Tecnologia na Educação em Ciência da Computação, ITiCSE é um grupo de trabalho da ACM SIGCSE, que visa melhorar a educação em ciência da computação por meio de visualização e engajamento.

[3]em tradução livre, Explorando o Papel da Visualização e do Engajamento na Educação em Ciência da Computação.

[4]em tradução livre, Gerando Tabela de Rastreio para Programas Java.

Parte II

Disciplina Raciocínio Algorítmico

5 Projeto de Disciplina de Raciocínio Algorítmico

Uma disciplina de **Raciocínio Algorítmico**, nomeada assim para que exista aderência com os conceitos de **Pensamento Computacional** e da Metodologia **DAAD**. Este será o projeto de uma disciplina de 80 horas-aula que podem ser distribuídas de formas diferentes, dependendo do formato acadêmico. O projeto aqui proposto considera 17 semanas com 4h aula por semana e 12h de trabalhos acadêmicos independentes, o que equivale a aproximadamente 1,78 créditos nos EUA e aproximadamente 3,2 **ECTS**. Trata-se de uma disciplina introdutória, que pode ser oferecida no primeiro ou segundo ano dos cursos de graduação em Ciência e Engenharia da Computação. Este projeto tem aderência às práticas adotadas em Curitiba, mas diverge das práticas adotadas nos EUA, Reino Unido e Europa. O objetivo é introduzir um *framework* de temas, métodos pedagógicos e exercícios que ajude a criar o hábito da estruturação da solução de problemas com a Metodologia **DAAD** (**D**ecomposição, **A**bstração, **A**lgoritmização e **D**epuração), preparando-os para resolver problemas computacionais complexos em qualquer contexto.

A estrutura do currículo foi projetada para construir progressivamente as habilidades e competências necessárias, começando com problemas e exercícios básicos e, paulatinamente, avançando para aplicações mais complexas. A limitação de 80 horas exige um equilíbrio cuidadoso entre a amplitude e a profundidade do conteúdo. As iniciativas semelhantes das universidades estudadas nesta pesquisa demonstraram que estes temas e técnicas podem ser aplicados em uma disciplina autônoma, ETH Zurich (123), ou integrados em várias disciplinas, como no Imperial College (118) ou Stanford (78), (103). A diferença entre estas duas abordagens reside na quantidade e profundidade do conteúdo referente à computação e não à prática de **Raciocínio Algorítmico** ou **Pensamento Computacional**. O que difere, essencialmente, do intuito da disciplina aqui proposta. O desafio reside em cobrir os elementos da Metodologia **DAAD** com profundidade suficiente para criar os constructos cognitivos necessários à aplicação natural do **Raciocínio Algorítmico**, usando para isso os temas introdutórios dos cursos de Ciência e Engenharia da Computação.

A progressão de conceitos para aplicações, e de problemas mais simples para mais complexos, parece ser uma estratégia eficaz — conclusão baseada no resultado da pesquisa realizada. Isso implica que, embora conceitos amplos sejam introduzidos, a profundidade será alcançada de forma incremental por meio da aplicação repetida e da crescente complexidade dos projetos. Portanto, a prática pedagógica deve priorizar a prática intensiva sobre a cobertura teórica exaustiva, garantindo que os alunos pratiquem o **Raciocínio Algorítmico** em vez de apenas aprenderem sobre ele ou repetirem técnicas apresentadas como solução.

Caberá ao professor da disciplina analisar as características particulares de cada turma e optar pelo uso, ou não, da metodologia de sala de aula invertida. Contudo, é necessário destacar que a opção sem a metodologia de sala de aula invertida terá impacto na criação dos constructos necessários à criação das competências relacionadas ao **Raciocínio Algorítmico**. Esta opção implica que o tempo dedicado à resolução de problemas, exercícios e projetos em sala será reduzido. Este projeto foi desenvolvido com a intenção de ser aplicado em uma disciplina de **Raciocínio Algorítmico** que não adote a metodologia de sala de aula invertida, mas que ainda assim permita ao aluno desenvolver as competências necessárias para a resolução de problemas computacionais complexos.

Finalmente, o autor optou por começar com a Linguagem C++ 23 (106). Esta escolha é

proposital para que o aluno possa criar em código, exatamente a mesma estrutura de solução que ele terá criado usando fluxogramas. Esta opção por uma linguagem mais flexível irá permitir que o aluno aumente o entendimento da complexidade computacional gradualmente e de acordo com suas características individuais. No primeiro contato com o **Raciocínio Algorítmico** o aluno irá criar fluxogramas simples, usando apenas os módulos básicos da computação (Início, Fim, Entrada, Saída, Atribuição, Decisão). Esta estrutura exige que a linguagem de programação escolhida possa utilizar `goto`.

5.1 EMENTA DE DISCIPLINA: RACIOCÍNIO ALGORÍTMICO

Carga Horária: 68 horas-aula presenciais + 12 horas de trabalho independente
Créditos: 4,53 (sistema brasileiro)
Pré-requisitos: Nenhum
Modalidade: Presencial

Desenvolvimento do **Raciocínio Algorítmico** por meio da Metodologia **DAAD** (**D**ecomposição, **A**bstração, **A**lgoritmização e **D**epuração). A disciplina é estruturada em quatro módulos principais interativos com atividades práticas e projetos que reforçam os conceitos de Introdução ao **Raciocínio Algorítmico** e técnicas de decomposição de problemas, desenvolvimento de habilidades de abstração e reconhecimento de padrões, design de algoritmos e implementação de estruturas de dados, técnicas de depuração sistemática e desenvolvimento de projetos integrados. Usando os conceitos fundamentais de resolução de problemas computacionais, desde atividades *unplugged* até implementação de algoritmos e estruturas de dados. Desenvolvimento de habilidades de análise, design e implementação de soluções computacionais utilizando fluxogramas, pseudocódigo e programação em C++ e Python. Aplicação de técnicas de depuração, controle de versão e desenvolvimento de projetos práticos. Complementa as atividades presenciais com três atividades de pesquisa e solução de problemas realizadas fora de sala.

5.1.1 OBJETIVOS

Objetivo Geral: capacitar o estudante a aplicar o **Raciocínio Algorítmico** para resolver problemas complexos por meio da decomposição, abstração, algoritmização e depuração sistemática, desenvolvendo competências práticas através de módulos interativos e projetos aplicados.

Objetivos Específicos:

- compreender e aplicar os princípios fundamentais do **Raciocínio Algorítmico** através da Metodologia **DAAD**;
- desenvolver habilidades de decomposição de problemas complexos em subproblemas gerenciáveis;
- criar e interpretar fluxogramas seguindo uma versão simplificada para representação algorítmica;
- representar algoritmos usando pseudocódigo;
- implementar soluções computacionais utilizando linguagem C++ e Python;
- aplicar técnicas de abstração na criação de componentes reutilizáveis e estruturas de dados;
- analisar eficiência e complexidade de algoritmos fundamentais;
- utilizar técnicas sistemáticas de depuração e teste de software;
- desenvolver projetos práticos aplicando metodologias de controle de versão;

Ao final da disciplina, os alunos deverão ser capazes de: decompor problemas computacionais complexos em subproblemas menores e gerenciáveis, identificando interfaces claras

entre eles; identificar características essenciais e informações relevantes de um problema, criando modelos ou representações generalizadas, e aplicar múltiplos níveis de abstração para gerenciar a complexidade; projetar procedimentos passo a passo eficientes e corretos, algoritmos, para resolver problemas decompostos, e expressá-los usando pseudocódigo e uma linguagem de programação escolhida; identificar, localizar e corrigir sistematicamente erros em seus próprios códigos e nos de outros, empregando diversas estratégias de depuração e teste; aplicar a Metodologia **DAAD** de forma iterativa para resolver desafios computacionais e de engenharia novos, demonstrando pensamento analítico; trabalhar eficazmente em equipes em projetos de programação, utilizando ferramentas e práticas colaborativas; articular claramente definições de problemas, soluções algorítmicas e processos de depuração para públicos técnicos e não técnicos.

5.2 CONTEÚDO PROGRAMÁTICO

A organização modular da disciplina combina teoria e prática através de atividades presenciais dinâmicas e projetos aplicados. Cada módulo é projetado para reforçar os conceitos aprendidos por meio de: atividades *unplugged* na forma de exercícios e atividades realizadas sem uso de máquinas de processamento; laboratórios Práticos permitindo a implementação progressiva de fluxogramas para código C++; projetos aplicados incluindo o desenvolvimento de sistemas especialistas e soluções do mundo real. Usando uma metodologia progressiva na evolução gradual de atividades simples para complexas; e incentivando a prática da aprendizagem Colaborativa nas atividades em sala e no uso de repositórios públicos e controle de versão.

6 Módulo 1: Semanas 1-3 (12 Horas-Aula): Introdução ao Raciocínio Algorítmico e Decomposição

Este módulo foca na compreensão conceitual do **Raciocínio Algorítmico** e sua importância, explorando a **Decomposição** segundo a definição: decompor é saber como quebrar problemas complexos em subproblemas menores. Em resumo:

- Conceitos fundamentais do **Raciocínio Algorítmico** e introdução à Metodologia **DAAD**;
- Atividades *unplugged* para desenvolvimento de raciocínio algorítmico;
- Técnicas de decomposição: quebrar problemas complexos em subproblemas menores e identificáveis;
- Fluxogramas: padrão IEEE, estruturas básicas (Início, Fim, Entrada, Saída, Atribuição, Decisão);
- Estruturas sintáticas modulares: laços de repetição (for, while, do while) e funções;
- Exercícios práticos de algoritmos em fluxogramas aplicando princípios de decomposição.

1. **Atividades *unplugged* (`unplugged`)**: serão utilizadas atividades sem computador para ilustrar a decomposição. Exemplos incluem "fazer um bolo", "fazer um sanduíche de presunto e queijo" ou "Programar o Professor". A descrição destas atividades está disponível no .

2. **Fluxogramas e Algoritmos Simples**: introdução às técnicas de fluxogramas para a definição de algoritmos e solução de problemas simples. Serão sugeridos exercícios de algoritmos para uso de fluxogramas, segundo uma simplificação do padrão ISO (144) contendo apenas os módulos referentes a Início, Fim, Entrada, Saída, Atribuição e Decisão.

3. **Estruturas Sintáticas Modulares**: introdução aos conceitos de artefatos sintáticos para laços de repetição (`for`, `while`, `do while`) e funções, na forma de módulos reutilizáveis criados por blocos de fluxogramas. Exercícios de fluxogramas para induzir a reutilização de conjuntos de blocos. Estes exercícios estão disponíveis no . A Figure 6.1 resume o padrão IEEE e as estruturas básicas e os blocos necessários para `for`, `while`, `do while`.

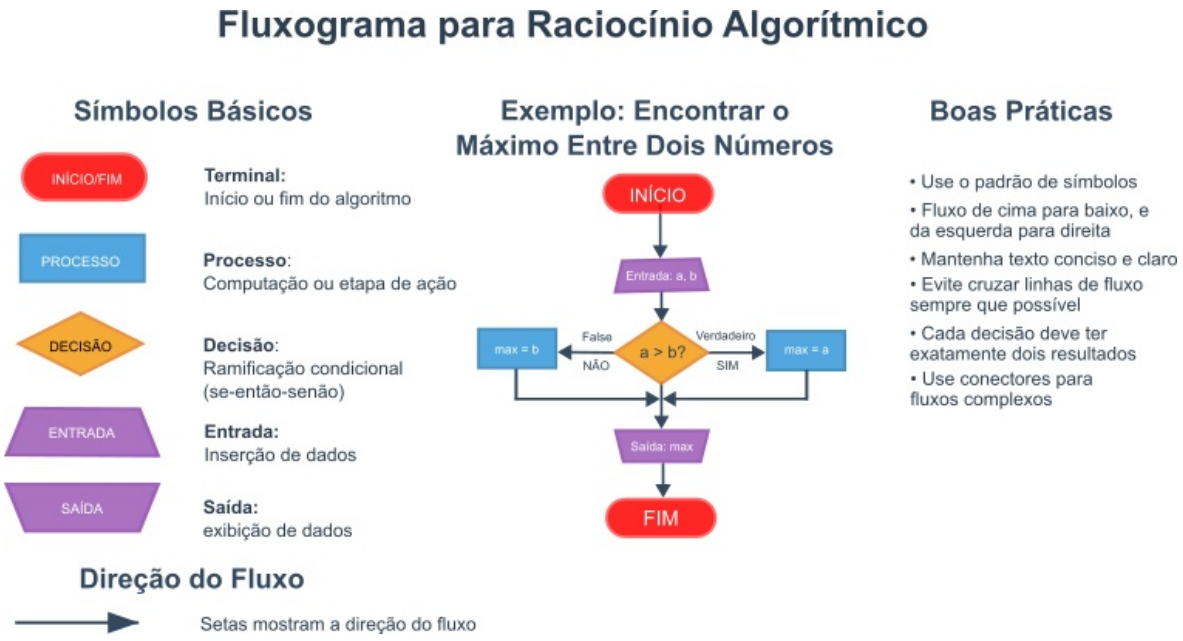


Figure 6.1: Estruturas básicas de fluxogramas adaptadas do padrão ISO 5807:1985.

6.1 Atribuição, Condicional, Repetição

Estas são as sugestões para atividades a serem realizadas no primeiro dia de aula, logo após a apresentação da disciplina. O professor deve escolher entre as atividades sugeridas aquelas que são adequadas ao seu perfil e as características da turma. Alternativamente, o professor pode usar suas próprias atividades, desde que elas possuam o mesmo grau de aderência às práticas da Metodologia **DAAD**.

6.1.1 Atividades *Unplugged*: Decomposição e Abstração

O importante é que o professor seja capaz de fazer atividades lúdicas que envolvam toda a turma na criação de listas de instruções para resolver tarefas simples, relacionadas com atividades cotidianas para criar os constructos cognitivos necessários ao entendimento da divisão de problemas grandes em problemas menores. Cada lista completa é, na verdade, um algoritmo. Ao final das tarefas, o professor deve enfatizar que a lista de tarefas é um algoritmo, e que cada tarefa é uma etapa do algoritmo.

Para as duas tarefas sugeridas a seguir, o professor deve enfatizar que o objetivo é criar uma lista de tarefas que permita a qualquer pessoa, mesmo aquelas que nunca fizeram a tarefa, completá-la com sucesso.

1. **Fazer um bolo / Fazer um sanduíche de presunto e queijo**:

Nos dois casos, antes de começar a tarefa, o professor deve distribuir uma página com pautas contendo a receita do bolo ou do sanduíche. Divida a sala em grupos e forneça *uma receita para cada aluno do grupo*. A tarefa consiste em escrever, o mais detalhadamente possível, todas as instruções necessárias para que alguém que nunca fez um sanduíche ou um bolo possa fazê-lo. O professor deve enfatizar a importância de incluir *todas as tarefas possíveis*. Cada grupo deve entregar apenas uma lista de tarefas, contendo uma versão das listas criadas pelos integrantes do grupo.

Material necessário: lápis e papel.

Entrega: a entrega será a lista de tarefas, escritas à mão.

Avaliação por Pares: o professor deve criar uma tabela com 10 notas no quadro em ordem descendente, embaralhar as listas, remover a relação grupo-lista, para que a avaliação seja anônima. Ler cada lista de tarefas destacando os pontos que não foram indicados. Por exemplo: é impossível colocar o queijo no pão sem pegar a fatia de queijo antes. Caberá ao professor destacar as instruções faltantes. Uma vez que a lista tenha sido lida, o professor irá pedir que os alunos levantem a mão para a nota que eles acham que o grupo merece, começando em 10, e marcar a nota que for mais votada. Finalmente, o professor deve separar a lista que tiver a maior nota. Esta lista será útil para explicar fluxogramas.

Objetivo: entender o que é dividir um problema em problemas menores e enfatizar a necessidade de detalhamento. Trabalhar em grupo. Avaliação por pares.

2. **Programar o Professor**:

Nesta atividade, o professor é um autômato que só sabe ler e identificar símbolos gráficos e usar o teclado. Além disso, o professor tem acesso exclusivo aos seus segredos, como se fosse uma memória interna. Esta tarefa deve ser realizada com a participação de todos os alunos da sala de forma voluntária. A tarefa que será realizada deve ser uma tarefa simples, como enviar um e-mail para si mesmo.

Antes de começar, o professor deve certificar-se de que seja possível operar o seu sistema operacional apenas com as teclas `tab` e `enter` para conseguir realizar a tarefa que será proposta. É importante que o professor não possa usar o mouse ou qualquer outro dispositivo de entrada para realizar a tarefa.

Em sala, divida a turma em grupos e sugira que dois ou mais alunos de cada grupo anotem todas as instruções que foram realizadas com sucesso. Sucesso significa que a instrução foi bem construída, segue a sequência correta e foi executada sem erros.

Para começar a atividade, explique os limites da capacidade do professor, compartilhe o desktop do seu computador sem nenhum aplicativo aberto. Informe qual aplicativo ou site será necessário para completar a tarefa e pare com as mãos no teclado. Neste ponto, a tarefa começou.

O professor deve ter apenas duas reações: ou faz o que os alunos pediram ou fica parado e imóvel. Esta é uma tarefa lúdica que tem o potencial de provocar interesse e engajamento.

Assim que a tarefa for concluída, o professor deve pedir que os alunos analisem as instruções escritas no seu grupo e verifiquem se elas estão corretas. Neste momento, o professor pode interagir com os grupos e fazer perguntas para guiá-los na análise. Guarde uma das listas de instruções para a definição de fluxogramas.

Material necessário: lápis e papel.

Entrega: lista de tarefas executadas criada pelos alunos.

Avaliação: a avaliação das listas será feita apenas pelos grupos.

Objetivo: mostrar os limites de entrada e saída de dados, os limites dos sistemas computacionais, a divisão de problemas em problemas menores.

6.1.2 Atividades *Unplugged* Fluxogramas: Decomposição, Abstração, Algoritmização e Depuração

Após as atividades lúdicas, o professor deve introduzir os conceitos de fluxogramas e estruturas de decisão e repetição. A ideia é que os alunos comecem criando fluxogramas para representar as tarefas realizadas nas atividades lúdicas.

1. **Criar fluxogramas para as instruções escritas pelos grupos**:

Esta é uma tarefa para a prática de criação de fluxogramas. Estas tarefas focam em **Decomposição** e **Abstração**. O professor deve propor problemas simples que podem ser resolvidos com fluxogramas, como calcular a média de três números, verificar se um número

é par ou ímpar, ou encontrar o maior de três números. Os alunos devem criar fluxogramas para resolver esses problemas, praticando a decomposição e abstração.

O professor irá apresentar o conceito de fluxogramas, utilizando uma adaptação dos módulos da ISO 5807:1985 (144), mostrando apenas os quatro módulos destacados na Figure 6.2. Cabe ao professor escolher se deve apresentar a ISO 5807:1985 (144) em mais detalhes, evidenciando a diferença entre o padrão completo e a versão reduzida a ser utilizada.

Figure 6.2: Módulos de Fluxogramas que serão usados para resolver os problemas básicos de algoritmização. Adaptados do padrão ISO 5807:1985.

O professor pode começar apresentando a ideia de fluxograma e os módulos que poderão ser utilizados para resolver as tarefas que ele irá apresentar. Para explicar os módulos, o professor pode usar uma das listas de atividades que resultaram das atividades da Section 6.1.1. Este é um bom momento para explicar que a lista de tarefas é um algoritmo, que cada tarefa corresponde a uma etapa do algoritmo, e aproveitar essa lista para criar o primeiro fluxograma.

As listas de instruções criadas pelos alunos para a solução das atividades geralmente não contêm módulos de decisão. Neste ponto, o professor pode escolher um problema diferente que inclua um módulo de decisão para explicar este processo. A explicação do módulo de decisão deve necessariamente incluir a ideia de que ele representa uma bifurcação do algoritmo, permitindo que o fluxo siga dois caminhos distintos, conforme a condição avaliada. Assim, há dois fluxos possíveis para a solução e para a execução. A tomada de decisão implica controle de fluxo, e o professor deve enfatizar que esse controle é parte essencial do **Raciocínio Algorítmico** e da construção de programas.

Divida a turma em grupos e entregue uma página impressa com as tarefas para cada grupo. Ao contrário da primeira atividade, quando entregamos uma página em branco para cada aluno, nesta atividade certifique-se de entregar apenas uma lista de tarefas por grupo. Isso induz o convívio e o compartilhamento.

Finalmente, o professor deve indicar que o sistema só consegue resolver problemas usando as regras da aritmética aplicando apenas operações binárias e unárias. Ou seja, para fazer a divisão do resultado de um produto, precisamos primeiro fazer o produto e depois dividir o resultado. O professor deve enfatizar que o sistema não consegue fazer operações de ordem superior, como calcular a média de três números, sem decompor o problema em subproblemas menores. O objetivo desta restrição é criar os constructos cognitivos necessários para que os alunos entendam a importância da decomposição de problemas, variáveis temporárias e controle de fluxo para a resolução de problemas.

Material necessário: lápis, papel.

Entrega: fluxogramas criados pelos alunos.

Apresentação e Avaliação: nesta tarefa, a avaliação consiste apenas na devolutiva da tarefa de forma coletiva e interativa no quadro. O professor pode escolher entre resolver apenas alguns problemas ou resolver todos os problemas no quadro. Além disso, o ritmo é importante. Avalie o resultado de acordo com o progresso da turma. Uma prática interessante é usar a Técnica da Sequência de Fibonacci para a apresentação da tarefa e devolutiva.

Para usar a Técnica da Sequência de Fibonacci na aplicação de tarefa siga os seguintes passos:

1. atribua tempo para a solução de 1 problema, ao final deste tempo faça a devolutiva deste problema;

2. atribua tempo para a solução de 1 problema, ao final deste tempo faça a devolutiva deste problema;
3. atribua tempo para a solução de 2 problemas, ao final deste tempo faça a devolutiva destes 2 problemas;
4. atribua tempo para a solução de 3 problemas, ao final deste tempo faça a devolutiva destes 3 problemas;
5. atribua tempo para a solução de 5 problemas, ao final deste tempo faça a devolutiva destes 5 problemas;

O tempo disponível para cada problema individual deve diminuir progressivamente. Assim, no último estágio, o tempo total para a solução dos 5 problemas propostos será igual ao tempo disponível para a solução do primeiro problema. A Figure 6.3 resume a Técnica da Sequência de Fibonacci.

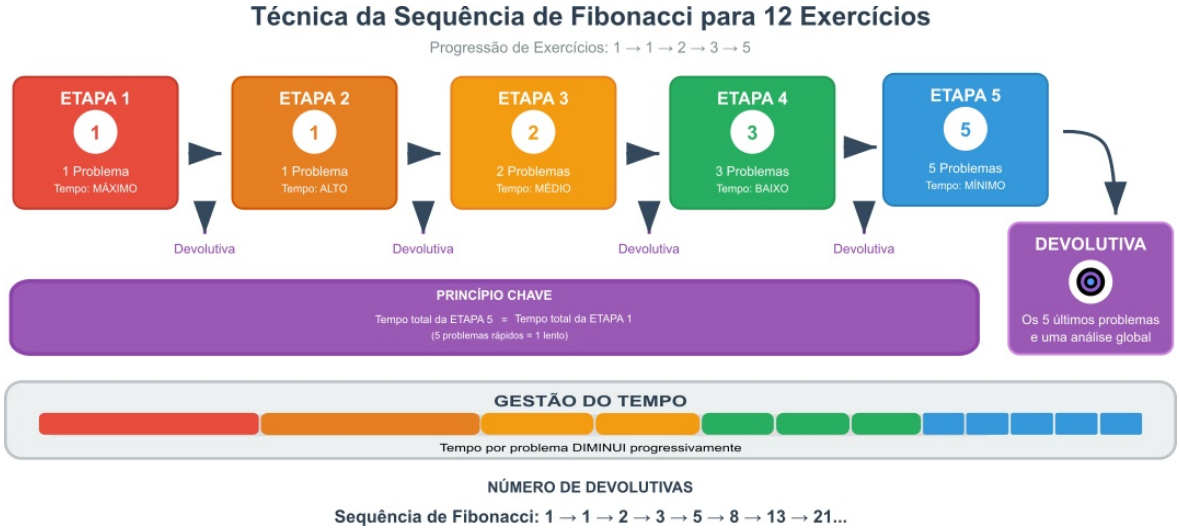


Figure 6.3

Se escolher a Técnica da Sequência de Fibonacci para apresentação e devolutiva de atividades, o professor deverá escolher o número de problemas que serão resolvidos em cada etapa de acordo com o tamanho e o resultado da turma, mas deve se manter na Técnica da Sequência de Fibonacci. A devolutiva deve ser feita no quadro, e o professor deve explicar cada passo do algoritmo, enfatizando a importância da decomposição e controle de fluxo. A seguir está uma lista de problemas que podem ser usados para esta atividade:

A. Elegibilidade para Eventos Cívicos: a Câmara Municipal de "Vila Nova" está organizando um evento cívico importante e precisa determinar rapidamente quais cidadãos são elegíveis para participar em atividades que exigem uma idade mínima de 16 anos. O Sr. João, responsável pelo registro, precisa de um sistema que, ao receber a `idade` de um cidadão, indique se este pode ou não participar nas atividades restritas. Crie o fluxograma de um sistema capaz de automatizar esta tarefa.

Análise:

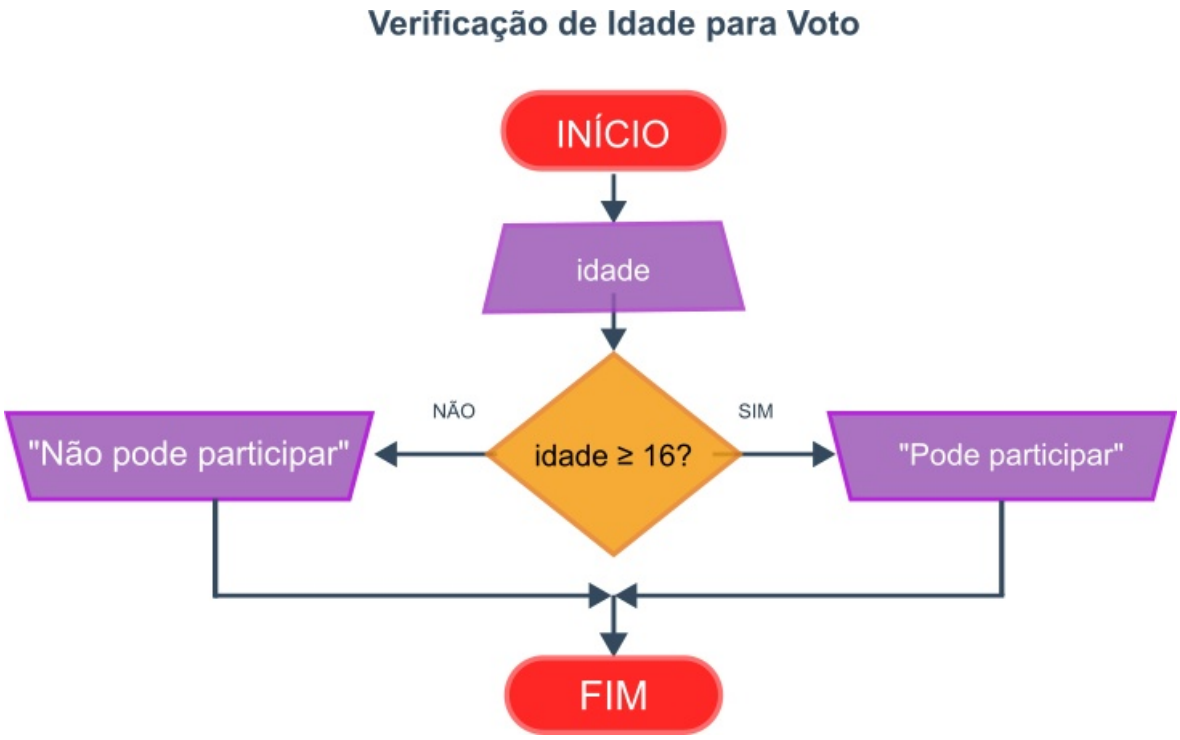
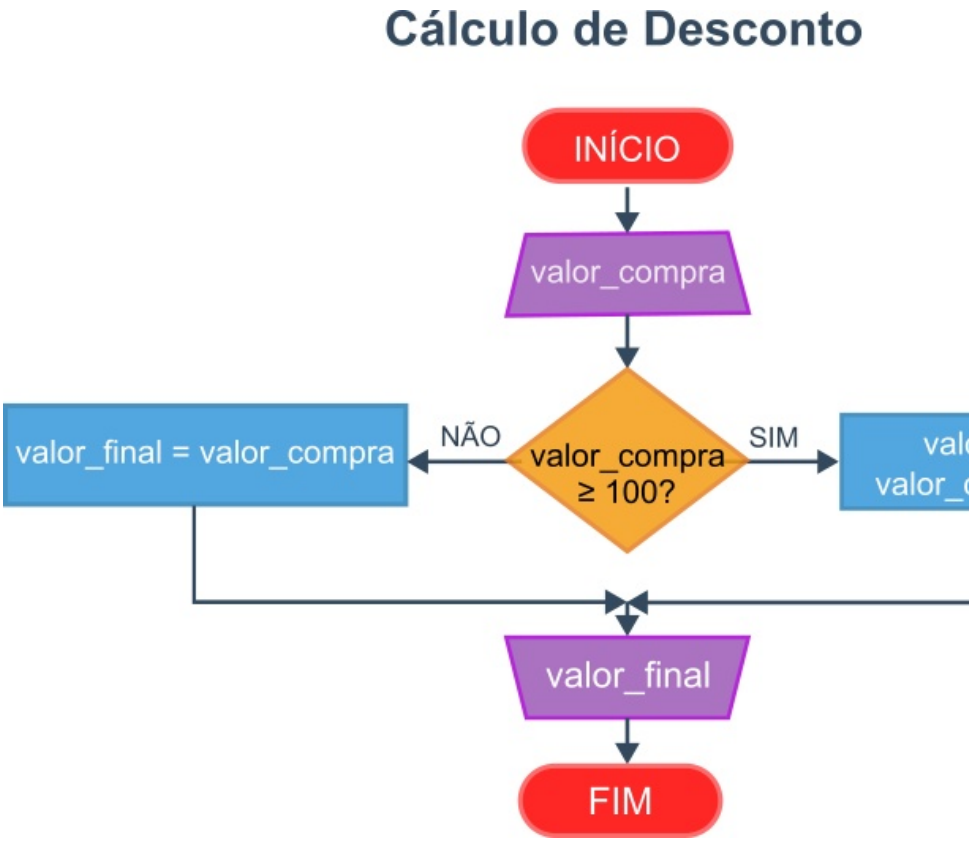


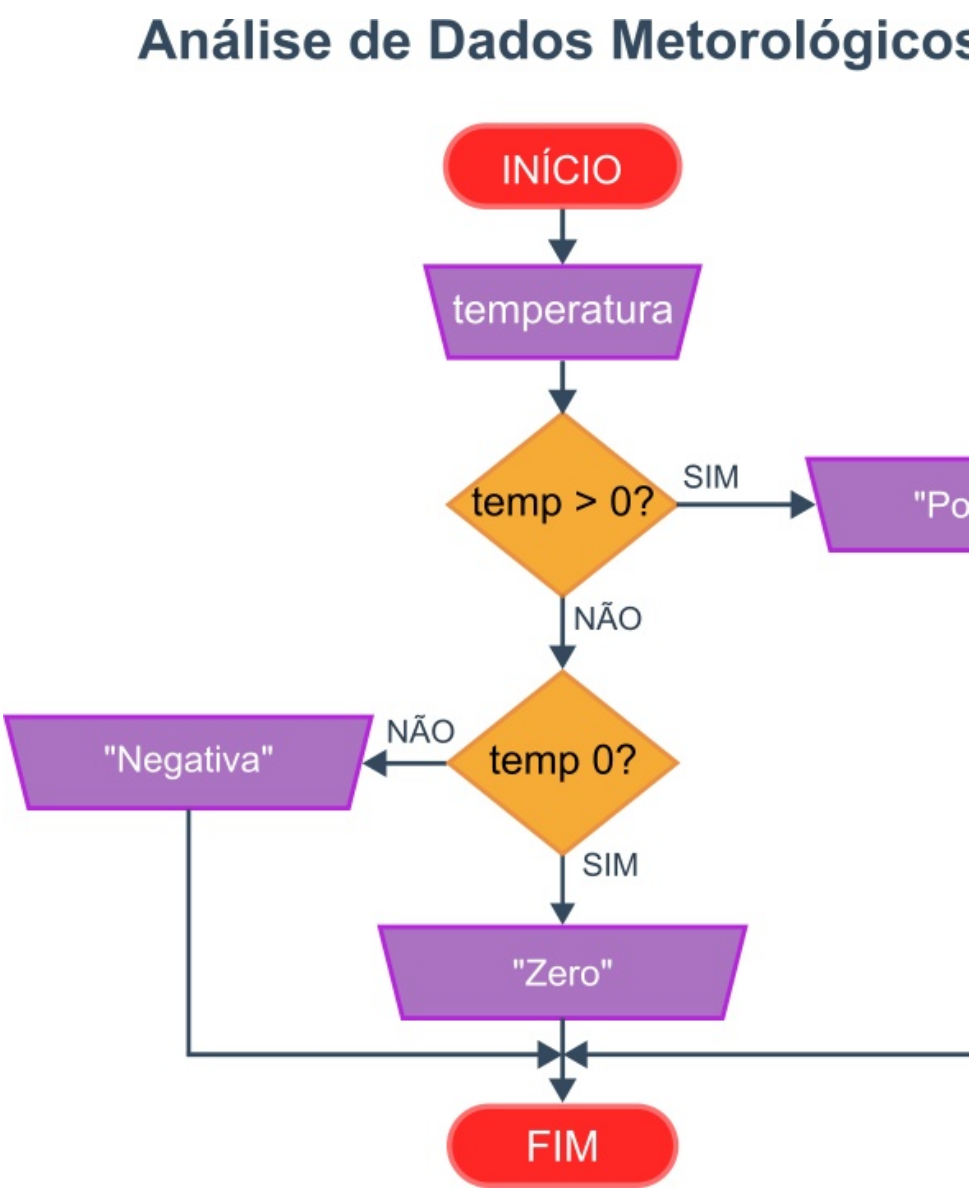
Figure 6.4: Resultado do Fluxograma de Elegibilidade para Eventos Cívicos.

B. Controle de Sistema de Descontos: a empresa "Varejo Inteligente" está implementando um novo sistema de descontos automáticos para os seus clientes mais fiéis. Se o valor total da compra de um cliente for de pelo menos 100 Reais, ele recebe um desconto de 10% sobre o valor total. Caso contrário, não há desconto. A Dona Alice, gerente da loja, precisa de uma forma rápida de calcular o valor final a ser pago por um cliente, dado o valor inicial da sua compra. Crie o fluxograma do algoritmo de um sistema que, ao receber o `valor_compra`, determine o `valor_final` com base nesta política de descontos.



Análise: ::: {#fig-fluxEX2}
Resultado do Fluxograma de Controle de Sistema de Descontos. :::

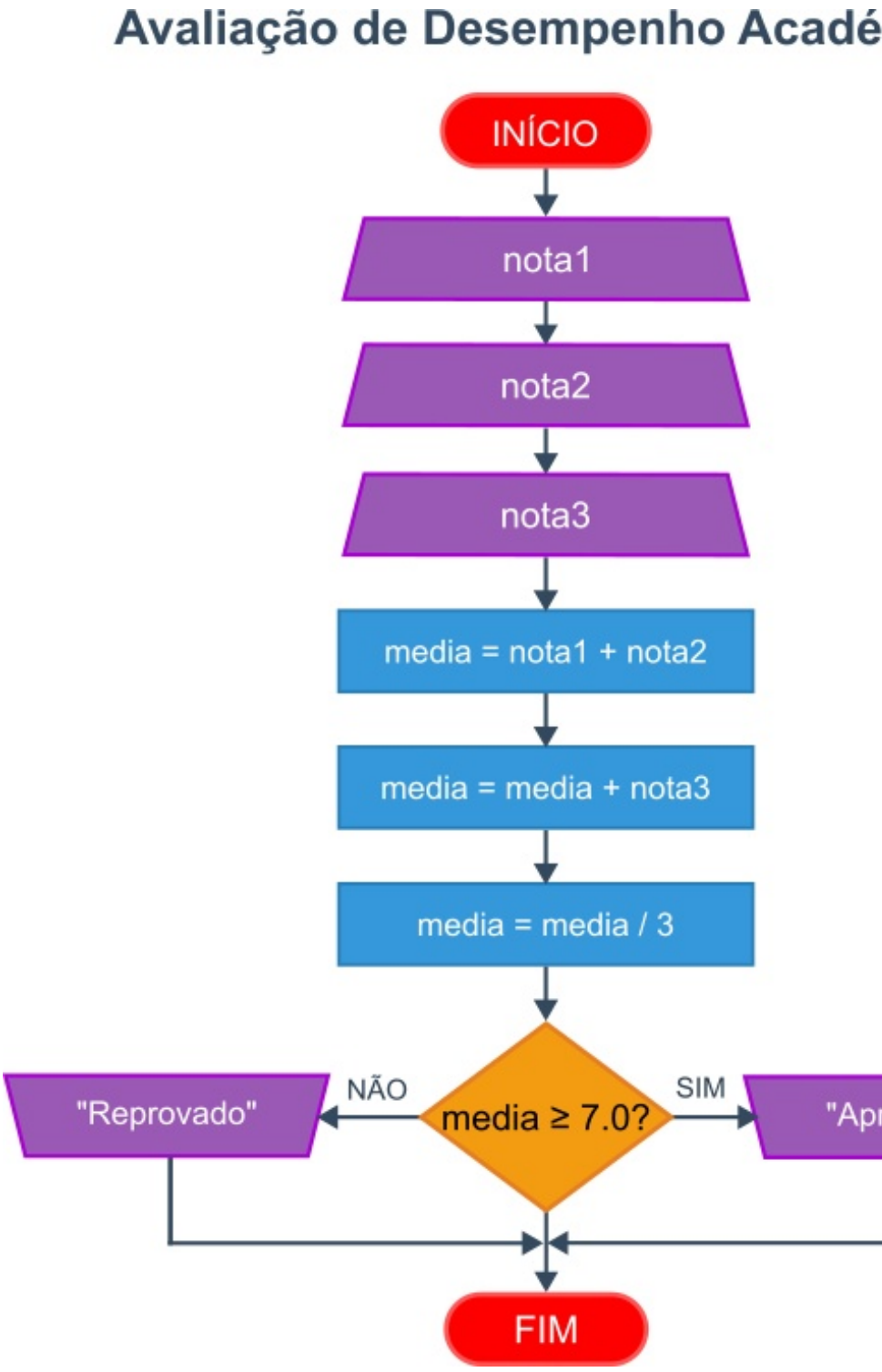
C. **Análise de Dados Meteorológicos**: no centro de meteorologia da cidade "Longe a Beça", o Dr. Silva está analisando dados de temperatura. Ele precisa classificar as leituras de temperatura para identificar padrões climáticos. Dado um valor de `temperatura` (em graus Celsius), o Dr. Silva precisa saber se este valor representa uma temperatura positiva (acima de zero), negativa (abaixo de zero) ou se é exatamente zero graus. Crie um fluxograma para um sistema capaz de classificar estas temperaturas.



Análise: ::: {#fig-fluxEX3}

Resultado do Fluxograma de Análise de Dados Meteorológicos. :::

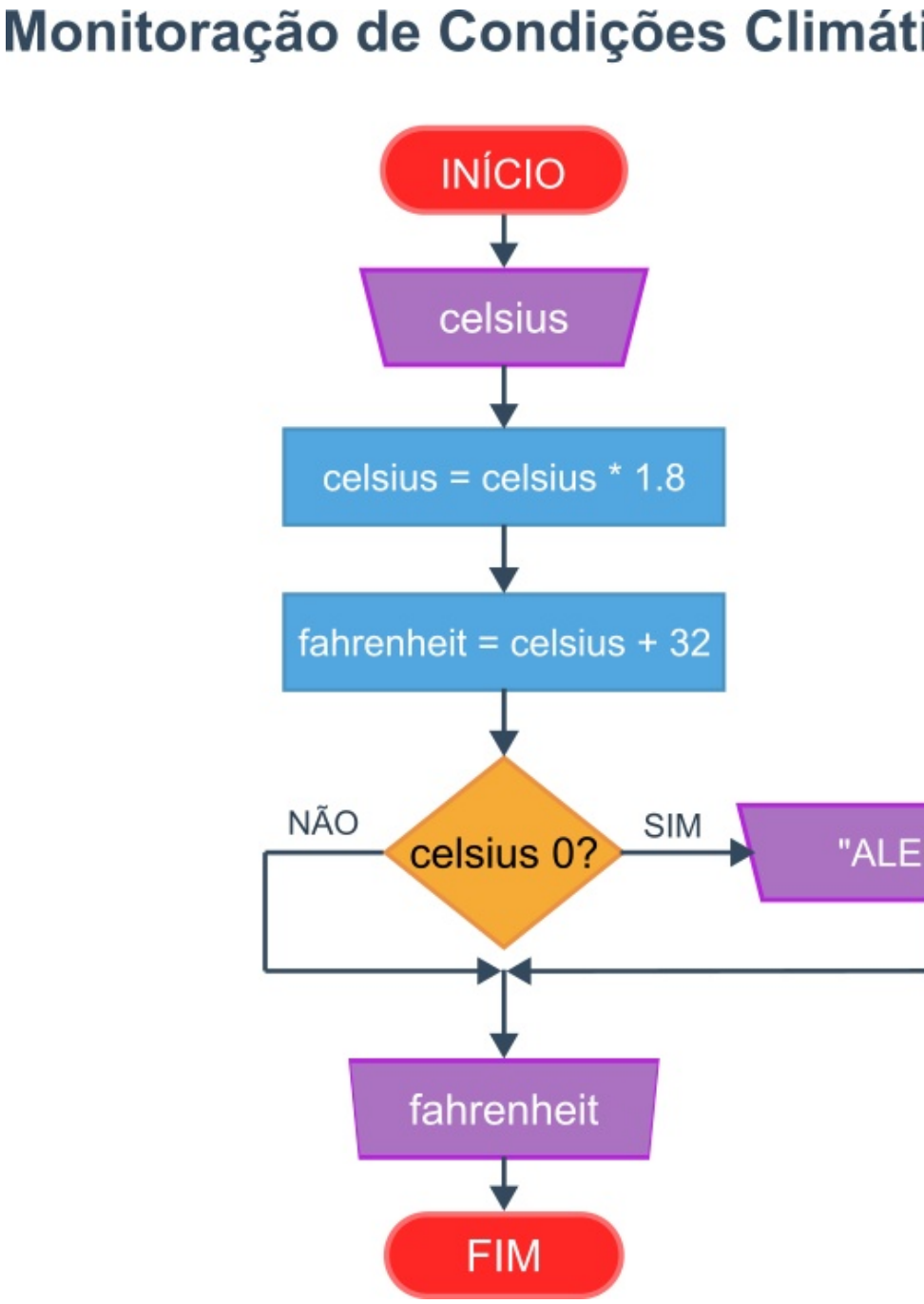
D. **Avaliação de Desempenho Acadêmico**: a Professora Sofia precisa avaliar o desempenho dos seus alunos na disciplina de "Raciocínio Algorítmico". Cada aluno realizará três avaliações, e a média aritmética desses testes determinará a sua aprovação. Para ser aprovado, a `media_final` deve ser igual ou superior a 7.0. Ajude a Professora Sofia a criar um algoritmo que, recebendo as três `notas` de um aluno, calcule a média e determine se o aluno foi "Aprovado" ou "Reprovado".



Análise: ::: {#fig-fluxEX4}

Resultado do Fluxograma de Avaliação de Desempenho Acadêmico. :::

E. Monitoramento de Condições Climáticas: a equipe de investigação do Ártico, liderada pela Dra. Anastácia, recolhe dados de temperatura em Celsius, mas precisa convertê-los para Fahrenheit a fim de comparar com dados históricos de outras estações. Além disso, para fins de segurança, é importante que seja emitido um alerta especial se a temperatura em Celsius for inferior a 0 graus. A fórmula de conversão é F = (C × 1,8) + 32. Crie um algoritmo, na forma de fluxograma, que realize esta conversão e emita o alerta quando necessário.



Análise: ::: {#fig-fluxEX5}
Resultado do Fluxograma de Monitoramento de Condições Climáticas. :::

F. **Calendário Empresarial**: a empresa "Eventos Globais" planeja os seus eventos com base em um calendário anual decidido com 10 anos de antecedência. Para certos cálculos de prazos, é útil saber se um determinado `ano` é bissexto. Sabendo que um ano é bissexto se satisfizer as seguintes condições:

1. O ano é **divisível por 4**: exemplo: 2024 é divisível por 4 (2024 / 4 = 506), então ele poderia ser bissexto.
2. **EXCETO** se o ano for **divisível por 100**: exemplo: 1900 é divisível por 100 (1900 / 100 = 19), então ele não seria bissexto, apesar de ser divisível por 4.
3. **MAS**, se o ano for **divisível por 400**, então ele **é bissexto**, mesmo que seja divisível por 100: exemplo: 2000 é divisível por 400 (2000 / 400 = 5), então ele é bissexto, mesmo sendo divisível por 100.

Análise:

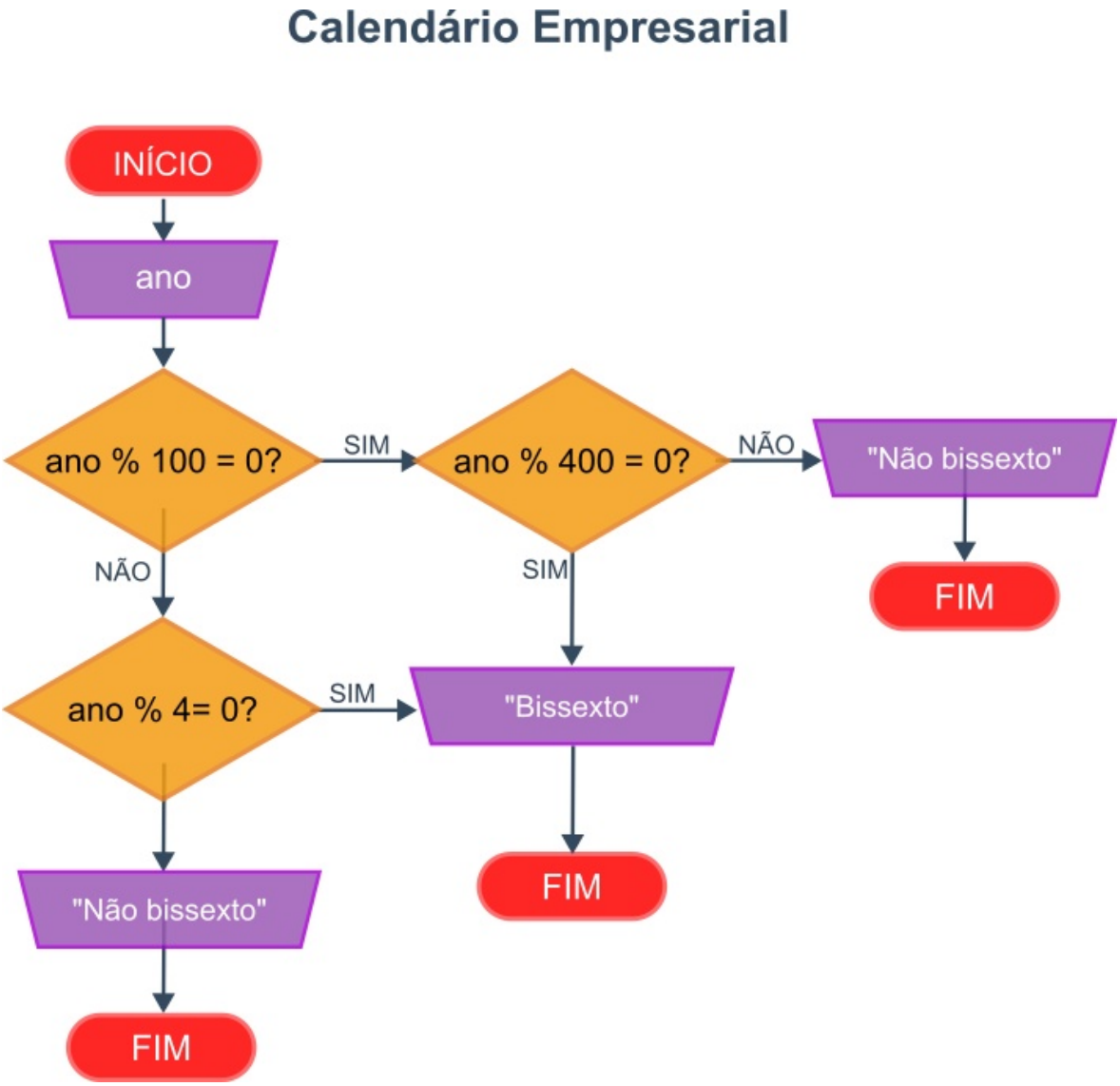


Figure 6.5: Resultado do Fluxograma de Calendário Empresarial.

G. Sistema de Acesso a Recursos: a biblioteca universitária "Conhecimento Aberto" está testando um novo sistema de acesso online aos seus tomos. Para a fase de testes, existem um `nome_usuário` ("admin") e uma `senha` ("123") pré-definidos. O objetivo é que os usuários insiram suas credenciais e o sistema verifique se o `login` foi bem-sucedido ou falhou. Usando apenas o `nome_usuário` ("admin") e a `senha` ("123"), desenvolva um algoritmo, na forma de fluxograma, que simule este processo de login.

Análise:

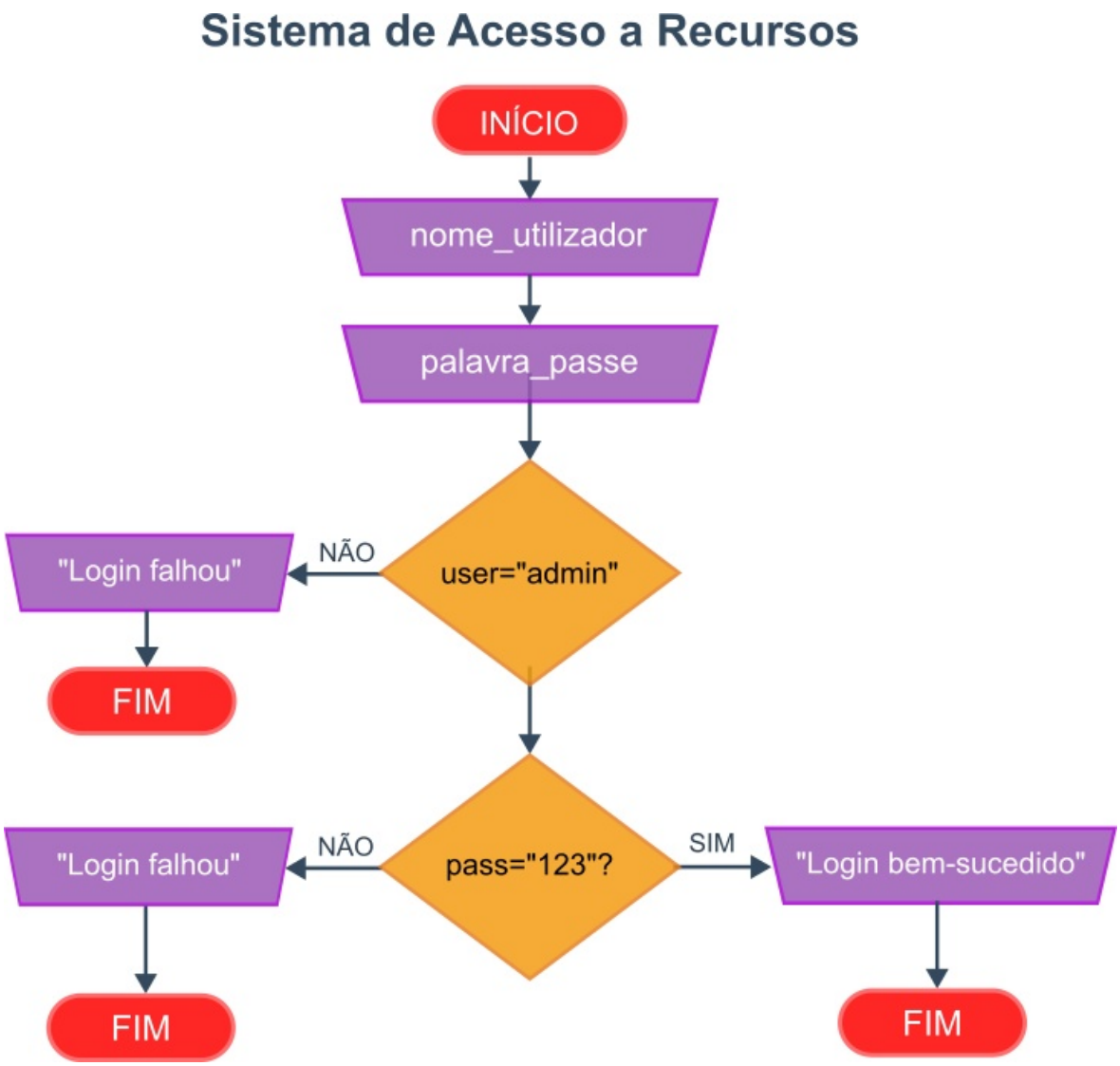


Figure 6.6: Resultado do Fluxograma de Sistema de Acesso a Recursos.

No exercício **G**, termina a Sequência de Fibonacci de aplicação (1, 1, 2, 3). Neste ponto, o professor pode escolher entre continuar a Sequência de Fibonacci, oferecendo 5 exercícios, sem alteração de conteúdo ou introduzir os conceitos de pseudocódigo e tabelas de rastreio. Se optar por esta abordagem, explique os conceitos de depuração e o uso das tabelas de rastreio usando o ao exercício **F** e solicite que os alunos apliquem estas mesmas técnicas ao exercício **G**.

Para o exercício F, Análise: o pseudocódigo poderia ser:

```
ALGORITMO VerificarAnoBissexto

ENTRADA:
    ano: inteiro

SAÍDA:
    ehBissexto: booleano

INÍCIO
    SE (ano % 400 == 0) ENTÃO
```

```
        ehBissexto ← VERDADEIRO
    SENÃO SE (ano % 100 == 0) ENTÃO
        ehBissexto ← FALSO
    SENÃO SE (ano % 4 == 0) ENTÃO
        ehBissexto ← VERDADEIRO
    SENÃO
        ehBissexto ← FALSO
    FIM SE

    RETORNAR ehBissexto
FIM
```

Finalmente, esta é uma oportunidade de discutir tanto a legibilidade do pseudocódigo quanto seu uso para testes e depuração. Neste caso, o professor pode sugerir os seguintes anos para validação do algoritmo: 2024, 1900, 2000, 2023. O professor pode pedir que os alunos verifiquem se o pseudocódigo funciona corretamente para esses anos usando as Tabelas de Rastreio.

Table 6.1: Teste 1: Ano 2024: resultado: 2024 é bissexto

Linha	Condição/Ação	Resultado	ehBissexto	Próximo Passo
ENTRADA	ano ← 2024	-	indefinido	Linha SE
SE	ano % 400 == 0	2024 % 400 = 24 0 → FALSO	indefinido	SENÃO SE (linha 2)
SENÃO SE	ano % 100 == 0	2024 % 100 = 24 0 → FALSO	indefinido	SENÃO SE (linha 3)
SENÃO SE	ano % 4 == 0	2024 % 4 = 0 → VERDADEIRO	VERDADEIRO	FIM SE
RETORNAR	ehBissexto	-	VERDADEIRO	FIM

Table 6.2: Teste 2: Ano 1900: resultado: 1900 não é bissexto

Linha	Condição/Ação	Resultado	ehBissexto	Próximo Passo
ENTRADA	ano ← 1900	-	indefinido	Linha SE
SE	ano % 400 == 0	1900 % 400 = 300 0 → FALSO	indefinido	SENÃO SE (linha 2)
SENÃO SE	ano % 100 == 0	1900 % 100 = 0 → VERDADEIRO	FALSO	FIM SE
RETORNAR	ehBissexto	-	FALSO	FIM

Table 6.3: Teste 3: Ano 2000: resultado: 2000 é bissexto

Linha	Condição/Ação	Resultado	ehBissexto	Próximo Passo
ENTRADA	ano ← 2000	-	indefinido	Linha SE
SE	ano % 400 == 0	2000 % 400 = 0 → VERDADEIRO	VERDADEIRO	FIM SE
RETORNAR	ehBissexto	-	VERDADEIRO	FIM

Table 6.4: Teste 4: Ano 2023: resultado: 2023 não é bissexto

Linha	Condição/Ação	Resultado	ehBissexto	Próximo Passo
ENTRADA	ano ← 2023	-	indefinido	Linha SE
SE	ano % 400 == 0	2023 % 400 = 23 0 → FALSO	indefinido	SENÃO SE (linha 2)
SENÃO SE	ano % 100 == 0	2023 % 100 = 23 0 → FALSO	indefinido	SENÃO SE (linha 3)
SENÃO SE	ano % 4 == 0	2023 % 4 = 3 0 → FALSO	indefinido	SENÃO
SENÃO	ehBissexto ← FALSO	-	FALSO	FIM SE
RETORNAR	ehBissexto	-	FALSO	FIM

Em resumo:

Table 6.5: Resumo: 2024 e 2000 são bissextos, enquanto 1900 e 2023 não são bissextos

Ano	Divisível por 4	Divisível por 100	Divisível por 400	É Bissexto
2024	Sim	Não	Não	Sim
1900	Sim	Sim	Não	Não
2000	Sim	Sim	Sim	Sim
2023	Não	Não	Não	Não

Em seguida, o professor pode apresentar uma versão mais próxima das linguagens de programação, como C++ ou Python, para que os alunos vejam a transição do pseudocódigo para o código real.

```
ALGORITMO VerificarAnoBissexto

ENTRADA:
    ano: inteiro

SAÍDA:
    ehBissexto: booleano

INÍCIO
    ehBissexto ← ((ano % 4 == 0) E (ano % 100 != 0)) OU (ano % 400 == 0)
```

```
    RETORNAR ehBissexto
FIM
```

Para o exercício G, Análise: o pseudocódigo poderia ser:

```
ALGORITMO SistemaLogin

CONSTANTES:
    USUARIO_VALIDO ← "admin"
    SENHA_VALIDA ← "123"

ENTRADA:
    nomeUsuário: texto
    senha: texto

SAÍDA:
    loginSucesso: booleano

INÍCIO
    ESCREVER "Digite o nome de Usuário:"
    LER nomeUsuário
    ESCREVER "Digite a senha:"
    LER senha

    SE (nomeUsuário == USUARIO_VALIDO) E (senha == SENHA_VALIDA) ENTÃO
        loginSucesso ← VERDADEIRO
        ESCREVER "Login bem-sucedido! Acesso autorizado."
    SENÃO
        loginSucesso ← FALSO
        ESCREVER "Login falhado! Credenciais inválidas."
    FIM SE

    RETORNAR loginSucesso
FIM
```

Neste caso, o professor pode sugerir os seguintes casos para validação do algoritmo: credenciais corretas (admin/123), nome correto com senha errada (admin/456), nome errado com senha correta (user/123), e ambas credenciais erradas (user/456). O professor pode pedir que os alunos verifiquem se o pseudocódigo funciona corretamente para esses casos usando as Tabelas de Rastreio.

Table 6.6: Teste 1: Credenciais corretas (admin/123): resultado: login bem-sucedido

Linha	Condição/Ação	Resultado	loginSucesso	Próximo Passo
ENTRADA	nomeUsuário ← "admin", senha ← "123"	-	indefinido	Linha SE

Linha	Condição/Ação	Resultado	loginSucesso	Próximo Passo
SE	(nomeUsuário == “admin”) E (senha == “123”)	(“admin” == “admin”) E (“123” == “123”) → VERDADEIRO E VERDADEIRO → VERDADEIRO	VERDADEIRO	Linha loginSucesso
loginSucesso	loginSucesso ← VERDADEIRO	-	VERDADEIRO	ESCREVER
ESCREVER	*Login bem-sucedido! Acesso autorizado.”	-	VERDADEIRO	FIM SE
RETORNAR	loginSucesso	-	VERDADEIRO	FIM

Table 6.7: Teste 2: Nome correto, senha errada (admin/456): resultado: login falhado

Linha	Condição/Ação	Resultado	loginSucesso	Próximo Passo
ENTRADA	nomeUsuário ← “admin”, senha ← “456”	-	indefinido	Linha SE
SE	(nomeUsuário == “admin”) E (senha == “123”)	(“admin” == “admin”) E (“456” == “123”) → VERDADEIRO E FALSO → FALSO	indefinido	SENÃO
SENÃO	loginSucesso ← FALSO	-	FALSO	ESCREVER
ESCREVER	*Login falhado! Credenciais inválidas.”	-	FALSO	FIM SE
RETORNAR	loginSucesso	-	FALSO	FIM

Table 6.8: Teste 3: Nome errado, senha correta (user/123): resultado: login falhado

Linha	Condição/Ação	Resultado	loginSucesso	Próximo Passo
ENTRADA	nomeUsuário ← “user”, senha ← “123”	-	indefinido	Linha SE

Linha	Condição/Ação	Resultado	loginSucesso	Próximo Passo
SE	(nomeUsuário == “admin”) E (senha == “123”)	(“user” == “admin”) E (“123” == “123”) → FALSO E VERDADEIRO → FALSO	indefinido	SENÃO
SENÃO	loginSucesso ← FALSO	-	FALSO	ESCREVER
ESCREVER	“Login falhado! Credenciais inválidas.”	-	FALSO	FIM SE
RETORNAR	loginSucesso	-	FALSO	FIM

Table 6.9: Teste 4: Ambas credenciais erradas (user/456): resultado: login falhado

Linha	Condição/Ação	Resultado	loginSucesso	Próximo Passo
ENTRADA	nomeUsuário ← “user”, senha ← “456”	-	indefinido	Linha SE
SE	(nomeUsuário == “admin”) E (senha == “123”)	(“user” == “admin”) E (“456” == “123”) → FALSO E FALSO → FALSO	indefinido	SENÃO
SENÃO	loginSucesso ← FALSO	-	FALSO	ESCREVER
ESCREVER	“Login falhado! Credenciais inválidas.”	-	FALSO	FIM SE
RETORNAR	loginSucesso	-	FALSO	FIM

Em resumo:

Table 6.10: Resumo: apenas o caso com credenciais corretas (admin/123) resulta em login bem-sucedido

Caso de Teste	Nome Usuário	Senha	Nome Válido	Senha Válida	Login Sucesso
Credenciais corretas	admin	123	Sim	Sim	Sim
Nome correto, senha errada	admin	456	Sim	Não	Não
Nome errado, senha correta	user	123	Não	Sim	Não

Caso de Teste	Nome Usuário	Senha	Nome Válido	Senha Válida	Login Sucesso
Ambas credenciais erradas	user	456	Não	Não	Não

Em seguida, o professor pode apresentar uma versão mais próxima das linguagens de programação, como C++ ou Python, para que os alunos vejam a transição do pseudocódigo para o código real.

```
ALGORITMO SistemaLoginConciso

CONSTANTES:
    USUARIO_VALIDO ← "admin"
    SENHA_VALIDA ← "123"

ENTRADA:
    nomeUsuário: texto
    senha: texto

SAÍDA:
    loginSucesso: booleano

INÍCIO
    LER nomeUsuário, senha
    loginSucesso ← (nomeUsuário == USUARIO_VALIDO) E (senha == SENHA_VALIDA)
    RETORNAR loginSucesso
FIM
```

Deste ponto em diante, o professor pode começar uma nova Sequência de Fibonacci, introduzindo exercícios mais complexos que envolvam estruturas de repetição e funções. A seguir estão alguns exemplos de exercícios que podem ser usados para continuar a prática de fluxogramas, pseudocódigo e tabelas de rastreio. Para estes exercícios caberá aos alunos escolher se usarão fluxogramas, pseudocódigo, mas devem usar as tabelas de rastreio:

H. **Avaliação de Saúde Ocupacional**: em uma clínica de saúde ocupacional, a enfermeira Joana Mãoleve realiza avaliações de rotina, incluindo o cálculo do Índice de Massa Corporal (IMC) dos funcionários. O IMC é calculado pela fórmula $IMC = \frac{peso}{altura \times altura}$, na qual o `peso` é em quilogramas e a `altura` em metros. Após o cálculo, o IMC é classificado para indicar o estado nutricional. Ajude a enfermeira Joana Mãoleve a criar um algoritmo, na forma de fluxograma, que, dados o `peso` e a `altura` de um funcionário, calcule o IMC e o classifique como:

- Menor que 18.5: “Abaixo do peso”
- Entre 18.5 e 24.9: “Peso normal”
- Entre 25 e 29.9: “Sobrepeso”
- 30 ou mais: “Obesidade”

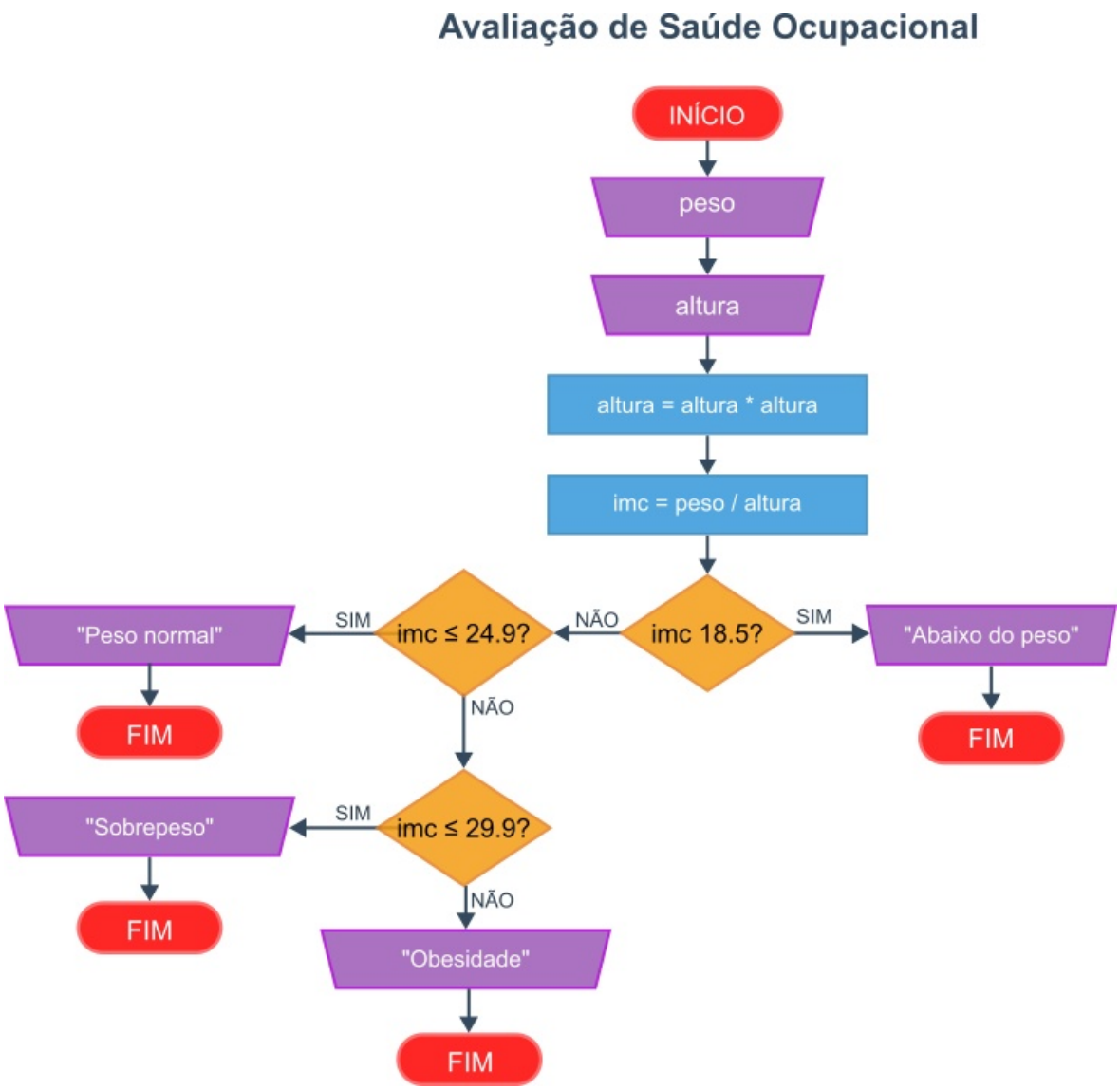


Figure 6.7: Resultado do Fluxograma de Avaliação de Saúde Ocupacional.

O pseudocódigo poderia ser:

```
ALGORITMO AvaliacaoIMC

ENTRADA:
    peso: real
    altura: real

SAÍDA:
    imc: real
    classificacao: texto

INÍCIO
    ESCREVER "Digite o peso (kg):"
    LER peso
    ESCREVER "Digite a altura (m):"
    LER altura
```

```
    imc ← peso / (altura * altura)

    SE (imc < 18.5) ENTÃO
        classificacao ← "Abaixo do peso"
    SENÃO SE (imc >= 18.5) E (imc <= 24.9) ENTÃO
        classificacao ← "Peso normal"
    SENÃO SE (imc >= 25.0) E (imc <= 29.9) ENTÃO
        classificacao ← "Sobrepeso"
    SENÃO
        classificacao ← "Obesidade"
    FIM SE

    ESCREVER "IMC: ", imc
    ESCREVER "Classificação: ", classificacao

    RETORNAR imc, classificacao
FIM
```

Neste caso, o professor pode sugerir os seguintes casos para validação do algoritmo: pessoa com baixo peso (45kg, 1.70m), peso normal (70kg, 1.75m), sobrepeso (80kg, 1.70m), e obesidade (100kg, 1.70m). O professor pode pedir que os alunos verifiquem se o pseudocódigo funciona corretamente para esses casos usando as Tabelas de Rastreio.

Table 6.11: Teste 1: Baixo peso (45kg, 1.70m): resultado: IMC 15.57 - Abaixo do peso

Linha	Condição/Ação	Resultado	imc	classificacao	Próximo Passo
ENTRADA	peso ← 45, altura ← 1.70	-	indefinido	indefinido	Cálculo IMC
imc	imc ← 45 / (1.70 * 1.70)	45 / 2.89 = 15.57	15.57	indefinido	Linha SE
SE	imc < 18.5	15.57 < 18.5 → VERDADEIRO	15.57	“Abaixo do peso”	FIM SE
ESCREVER	“IMC: 15.57”	-	15.57	“Abaixo do peso”	ESCREVER
ESCREVER	“Classificação: Abaixo do peso”	-	15.57	“Abaixo do peso”	RETORNAR
RETORNAR	imc, classificacao	-	15.57	“Abaixo do peso”	FIM

Table 6.12: Teste 2: Peso normal (70kg, 1.75m): resultado: IMC 22.86 - Peso normal

Linha	Condição/Ação	Resultado	imc	classificacao	Próximo Passo
ENTRADA	peso ← 70, altura ← 1.75	-	indefinido	indefinido	Cálculo IMC
imc	imc ← 70 / (1.75 * 1.75)	70 / 3.0625 = 22.86	22.86	indefinido	Linha SE
SE	imc < 18.5	22.86 < 18.5 → FALSO	22.86	indefinido	SENÃO SE

Linha	Condição/Ação	Resultado	imc	classificacao	Próximo Passo
SENÃO SE	(imc >= 18.5) E (imc <= 24.9)	(22.86 >= 18.5) E (22.86 <= 24.9) → VERDADEIRO E VERDADEIRO → VERDADEIRO	22.86	“Peso normal”	FIM SE
ESCREVER	“IMC: 22.86”	-	22.86	“Peso normal”	ESCREVER
ESCREVER	“Classificação: Peso normal”	-	22.86	“Peso normal”	RETORNAR
RETORNAR	imc, classificacao	-	22.86	“Peso normal”	FIM

Table 6.13: Teste 3: Sobrepeso (80kg, 1.70m): resultado: IMC 27.68 - Sobrepeso

Linha	Condição/Ação	Resultado	imc	classificacao	Próximo Passo
ENTRADA	peso ← 80, altura ← 1.70	-	indefinido	indefinido	Cálculo IMC
imc	imc ← 80 / (1.70 * 1.70)	80 / 2.89 = 27.68	27.68	indefinido	Linha SE
SE	imc < 18.5	27.68 < 18.5 → FALSO	27.68	indefinido	SENÃO SE
SENÃO SE	(imc >= 18.5) E (imc <= 24.9)	(27.68 >= 18.5) E (27.68 <= 24.9) → VERDADEIRO E FALSO → FALSO	27.68	indefinido	SENÃO SE
SENÃO SE	(imc >= 25.0) E (imc <= 29.9)	(27.68 >= 25.0) E (27.68 <= 29.9) → VERDADEIRO E VERDADEIRO → VERDADEIRO	27.68	“Sobrepeso”	FIM SE
ESCREVER	“IMC: 27.68”	-	27.68	“Sobrepeso”	ESCREVER
ESCREVER	“Classificação: Sobrepeso”	-	27.68	“Sobrepeso”	RETORNAR
RETORNAR	imc, classificacao	-	27.68	“Sobrepeso”	FIM

Table 6.14: Teste 4: Obesidade (100kg, 1.70m): resultado: IMC 34.60 - Obesidade

Linha	Condição/Ação	Resultado	imc	classificacao	Próximo Passo
ENTRADA	peso ← 100, altura ← 1.70	-	indefinido	indefinido	Cálculo IMC
imc	imc ← 100 / (1.70 * 1.70)	100 / 2.89 = 34.60	34.60	indefinido	Linha SE
SE	imc < 18.5	34.60 < 18.5 → FALSO	34.60	indefinido	SENÃO SE
SENÃO SE	(imc >= 18.5) E (imc <= 24.9)	(34.60 >= 18.5) E (34.60 <= 24.9) → VERDADEIRO E FALSO → FALSO	34.60	indefinido	SENÃO SE
SENÃO SE	(imc >= 25.0) E (imc <= 29.9)	(34.60 >= 25.0) E (34.60 <= 29.9) → VERDADEIRO E FALSO → FALSO	34.60	indefinido	SENÃO
SENÃO	classificacao ← “Obesidade”	-	34.60	“Obesidade”	FIM SE
ESCREVER	“IMC: 34.60”	-	34.60	“Obesidade”	ESCREVER
ESCREVER	“Classificação: Obesidade”	-	34.60	“Obesidade”	RETORNAR
RETORNAR	imc, classificacao	-	34.60	“Obesidade”	FIM

Em resumo:

Table 6.15: Resumo: algoritmo classifica corretamente todas as faixas de IMC conforme especificação médica

Caso de Teste	Peso (kg)	Altura (m)	IMC Calculado	Faixa IMC	Classificação
Baixo peso	45	1.70	15.57	< 18.5	Abaixo do peso
Peso normal	70	1.75	22.86	18.5-24.9	Peso normal
Sobrepeso	80	1.70	27.68	25.0-29.9	Sobrepeso
Obesidade	100	1.70	34.60	30.0	Obesidade

Em seguida, o professor pode apresentar uma versão mais próxima das linguagens de programação, como C++ ou Python, para que os alunos vejam a transição do pseudocódigo para o código real.

```
ALGORITMO AvaliacaoIMCConciso

ENTRADA:
    peso: real
    altura: real

SAÍDA:
    imc: real
    classificacao: texto

INÍCIO
    LER peso, altura
    imc ← peso / (altura * altura)

    SE (imc < 18.5) ENTÃO classificacao ← "Abaixo do peso"
    SENÃO SE (imc <= 24.9) ENTÃO classificacao ← "Peso normal"
    SENÃO SE (imc <= 29.9) ENTÃO classificacao ← "Sobrepeso"
    SENÃO classificacao ← "Obesidade"

    RETORNAR imc, classificacao
FIM
```

I. Sistema de Avaliação de Crédito Bancário: no banco "Futuro Seguro", o analista Roberto precisa automatizar a aprovação de empréstimos pessoais. O sistema deve considerar múltiplos critérios: `renda_mensal`, `idade`, `score_credito` e `tempo_emprego` (em anos). As regras são: (1) Renda mínima de 2000 reais; (2) Idade entre 21 e 65 anos; (3) Score de crédito de pelo menos 600; (4) Tempo de emprego de pelo menos 2 anos. Se todos os critérios forem atendidos, calcular o `valor_maximo_emprestimo` como renda_mensal $\times$ 10. Se o score for maior que 750, aplicar bônus de 20% no valor máximo. Se algum critério não for atendido, negar o empréstimo. Crie um fluxograma que implemente este sistema de avaliação.

Análise: o pseudocódigo poderia ser:

```
ALGORITMO AvaliacaoCreditoBancario

ENTRADA:
    rendaMensal: real
    idade: inteiro
    scoreCredito: inteiro
    tempoEmprego: real

SAÍDA:
    aprovado: booleano
    valorMaximoEmprestimo: real
    motivoRejeicao: texto

INÍCIO
    ESCREVER "Digite a renda mensal:"
    LER rendaMensal
    ESCREVER "Digite a idade:"
    LER idade
    ESCREVER "Digite o score de crédito:"
    LER scoreCredito
```

```
    ESCREVER "Digite o tempo de emprego (anos):"
    LER tempoEmprego

    aprovado ← VERDADEIRO
    motivoRejeicao ← ""
    valorMaximoEmprestimo ← 0

    // Verificar critério 1: Renda mínima de 2000 reais
    SE (rendaMensal < 2000) ENTÃO
        aprovado ← FALSO
        motivoRejeicao ← motivoRejeicao + "Renda insuficiente (mín. R$ 2000); "
    FIM SE

    // Verificar critério 2: Idade entre 21 e 65 anos
    SE (idade < 21) OU (idade > 65) ENTÃO
        aprovado ← FALSO
        motivoRejeicao ← motivoRejeicao + "Idade fora da faixa permitida (21-65 anos); "
    FIM SE

    // Verificar critério 3: Score de crédito de pelo menos 600
    SE (scoreCredito < 600) ENTÃO
        aprovado ← FALSO
        motivoRejeicao ← motivoRejeicao + "Score de crédito insuficiente (mín. 600); "
    FIM SE

    // Verificar critério 4: Tempo de emprego de pelo menos 2 anos
    SE (tempoEmprego < 2) ENTÃO
        aprovado ← FALSO
        motivoRejeicao ← motivoRejeicao + "Tempo de emprego insuficiente (mín. 2 anos); "
    FIM SE

    // Se todos os critérios forem atendidos, calcular valor máximo
    SE (aprovado == VERDADEIRO) ENTÃO
        valorMaximoEmprestimo ← rendaMensal * 10

        // Verificar bônus para score > 750
        SE (scoreCredito > 750) ENTÃO
            valorMaximoEmprestimo ← valorMaximoEmprestimo * 1.2
            ESCREVER "Bônus de 20% aplicado devido ao excelente score de crédito!"
        FIM SE

        ESCREVER "Empréstimo APROVADO!"
        ESCREVER "Valor máximo disponível: R$ ", valorMaximoEmprestimo
    SENÃO
        ESCREVER "Empréstimo NEGADO!"
        ESCREVER "Motivos: ", motivoRejeicao
    FIM SE

    RETORNAR aprovado, valorMaximoEmprestimo, motivoRejeicao
FIM
```

Neste caso, o professor pode sugerir os seguintes casos para validação do algoritmo: cliente

aprovado sem bônus (R$ 5000 renda, 35 anos, score 700, 5 anos emprego), cliente aprovado com bônus (R$ 8000 renda, 28 anos, score 800, 3 anos emprego), negação por renda baixa (R$ 1500 renda, 30 anos, score 650, 3 anos emprego), e negação por múltiplos critérios (R$ 1800 renda, 19 anos, score 550, 1 ano emprego). O professor pode pedir que os alunos verifiquem se o pseudocódigo funciona corretamente para esses casos usando as Tabelas de Rastreio.

Table 6.16: Teste 1: Aprovação sem bônus (5000 renda, 35 anos, score 700): resultado: Aprovado - R$ 50.000

Linha	Condição/Ação	Resultado	aprovado	valorMaximoEmprestimo	motivoRejeicao	Próximo Passo
ENTRADA	rendaMensal ← 5000, idade ← 35, score ← 700, tempoEmprego ← 5	-	VERDADEIRO	0	“ ”	Verificar renda
SE renda	rendaMensal < 2000 → 5000 < 2000 → FALSO	-	VERDADEIRO	0	“ ”	Verificar idade
SE idade	(idade < 21) OU (idade > 65) → (35 < 21) OU (35 > 65) → FALSO OU FALSO → FALSO	-	VERDADEIRO	0	“ ”	Verificar score
SE score	scoreCredito < 600 → 700 < 600 → FALSO	-	VERDADEIRO	0	“ ”	Verificar tempo
SE tempo	tempoEmprego < 2 → 5 < 2 → FALSO	-	VERDADEIRO	0	“ ”	Calcular valor
SE aprovado	aprovado == VERDADEIRO → VERDADEIRO	valorMaximoEmprestimo = 5000 * 10 = 50000	VERDADEIRO	50000	“ ”	Verificar bônus
SE bônus	scoreCredito > 750 → 700 > 750 → FALSO	-	VERDADEIRO	50000	“ ”	ESCREVER

Linha	Condição/Ação	Resultado	aprovado	valorMaximoEmprestimo	motivoRejeicao	Próximo Passo
RETORNAR	aprovado, valorMaximoEmprestimo, motivoRejeicao	-	VERDADEIRO	50000	“ ”	FIM

Table 6.17: Teste 2: Aprovação com bônus (8000 renda, 28 anos, score 800): resultado: Aprovado - R$ 96.000 com bônus

Linha	Condição/Ação	Resultado	aprovado	valorMaximoEmprestimo	motivoRejeicao	Próximo Passo
ENTRADA	renda ← 8000, idade ← 28, score ← 800, tempo ← 3	-	VERDADEIRO	0	“ ”	Verificar critérios
Verificações	Todas as condições (renda 2000, idade 21-65, score 600, tempo 2) são atendidas	-	VERDADEIRO	0	“ ”	Calcular valor
SE aprovado	aprovado == VERDADEIRO → VERDADEIRO	valorMaximoEmprestimo = 8000 * 10 = 80000	VERDADEIRO	80000	“ ”	Verificar bônus
SE bônus	scoreCredito > 750 → 800 > 750 → VERDADEIRO	valorMaximoEmprestimo = 80000 * 1.2 = 96000	VERDADEIRO	96000	“ ”	ESCREVER
ESCREVER	“Bônus de 20% aplicado...”	-	VERDADEIRO	96000	“ ”	ESCREVER
RETORNAR	aprovado, valorMaximoEmprestimo, motivoRejeicao	-	VERDADEIRO	96000	“ ”	FIM

Table 6.18: Teste 3: Negação por renda baixa (1500 renda): resultado: Negado - Renda insuficiente

Linha	Condição/Ação	Resultado	aprovado	valorMaximoEmprestimo	motivoRejeicao	Próximo Passo
ENTRADA	renda ← 1500, idade ← 30, score ← 650, tempo ← 3	-	VERDADEIRO	0	“ ”	Verificar renda
SE renda	rendaMensal < 2000 → 1500 < 2000 → VERDADEIRO	aprovado ← FALSO, motivoRejeicao ← “Renda insuficiente (mín. R$ 2000);”	FALSO	0	“Renda insuficiente (mín. R$ 2000);”	Verificar idade
SE idade	(30 < 21) OU (30 > 65) → FALSO OU FALSO → FALSO	-	FALSO	0	“Renda insuficiente (mín. R$ 2000);”	Verificar score
SE score	650 < 600 → FALSO	-	FALSO	0	“Renda insuficiente (mín. R$ 2000);”	Verificar tempo
SE tempo	3 < 2 → FALSO	-	FALSO	0	“Renda insuficiente (mín. R$ 2000);”	Verificar aprovação
SE aprovado	aprovado == VERDADEIRO → FALSO == VERDADEIRO → FALSO	-	FALSO	0	“Renda insuficiente (mín. R$ 2000);”	SENÃO
SENÃO	ESCREVER “Empréstimo NEGADO!”	-	FALSO	0	“Renda insuficiente (mín. R$ 2000);”	RETORNAR

Linha	Condição/Ação	Resultado	aprovado	valorMaximoEmprestimo	motivoRejeicao	Próximo Passo
RETORNAR	NÃO aprovado, valorMaximoEmprestimo, motivoRejeicao	-	FALSO	0	“Renda insuficiente (mín. R$ 2000);”	FIM

Table 6.19: Teste 4: Negação por múltiplos critérios (todos critérios falham): resultado: Negado - Múltiplos motivos

Linha	Condição/Ação	Resultado	aprovado	valorMaximoEmprestimo	motivoRejeicao	Próximo Passo
ENTRADA	renda ← 1800, idade ← 19, score ← 550, tempo ← 1	-	VERDADEIRO	0	“ ”	Verificar critérios
SE renda	1800 < 2000 → VERDADEIRO	aprovado ← FALSO, motivoRejeicao ← “Renda insuficiente (mín. R$ 2000);”	FALSO	0	“Renda insuficiente (mín. R$ 2000);”	Verificar idade
SE idade	(19 < 21) OU (19 > 65) → VERDADEIRO OU FALSO → VERDADEIRO	motivoRejeicao ← “...Idade fora da faixa permitida (21-65 anos);”	FALSO	0	“Renda insuficiente...; Idade fora da faixa...;”	Verificar score
SE score	550 < 600 → VERDADEIRO	motivoRejeicao ← “...Score de crédito insuficiente (mín. 600);”	FALSO	0	“...Score insuficiente...;”	Verificar tempo

Linha	Condição/Ação	Resultado	aprovado	valorMaximoEmprestimo	motivoRejeicao	Próximo Passo
SE tempo	$1 < 2 \rightarrow$ VERDADEIRO	motivoRejeicao ← “...Tempo de emprego insuficiente (mín. 2 anos);”	FALSO	0	“...Tempo insuficiente...;”	Verificar aprovação
SE aprovado	FALSO == VERDADEIRO → FALSO	-	FALSO	0	“Múltiplos motivos de rejeição”	SENÃO
RETORNAR	aprovado, valorMaximoEmprestimo, motivoRejeicao	-	FALSO	0	“Múltiplos motivos”	FIM

Em resumo:

Table 6.20: Resumo: algoritmo avalia corretamente todos os critérios bancários e aplica bônus para clientes premium

Caso de Teste	Renda	Idade	Score	Tempo Emp.	Critérios Atendidos	Valor Base	Bônus	Valor Final	Status
Aprovação básica	5000	35	700	5	Todos (4/4)	R$ 50.000	Não	R$ 50.000	Aprovado
Aprovação c/ bônus	8000	28	800	3	Todos (4/4)	R$ 80.000	Sim (+20%)	R$ 96.000	Aprovado
Renda baixa	1500	30	650	3	3/4 (renda falha)	-	-	R$ 0	Negado
Múltiplos problemas	1800	19	550	1	0/4 (todos falham)	-	-	R$ 0	Negado

A versão mais próxima:

```
ALGORITMO AvaliacaoCreditoBancarioConciso

ENTRADA:
    rendaMensal, idade, scoreCredito, tempoEmprego: numerico

SAÍDA:
    aprovado: booleano
    valorMaximoEmprestimo: real

INÍCIO
```

```
    LER rendaMensal, idade, scoreCredito, tempoEmprego

    aprovado ← (rendaMensal >= 2000) E (idade >= 21 E idade <= 65) E
               (scoreCredito >= 600) E (tempoEmprego >= 2)

    SE aprovado ENTÃO
        valorMaximoEmprestimo ← rendaMensal * 10
        SE (scoreCredito > 750) ENTÃO
            valorMaximoEmprestimo ← valorMaximoEmprestimo * 1.2
        valorMaximoEmprestimo ← valorMaximoEmprestimo
    SENÃO
        valorMaximoEmprestimo ← 0

    RETORNAR aprovado, valorMaximoEmprestimo
FIM
```

J. Classificação de Emergências Hospitalares: no Hospital "Santa Esperança", a enfermeira Carla utiliza o protocolo de Manchester para classificar a urgência dos pacientes. O sistema recebe `temperatura`, `pressao_sistolica`, `frequencia_cardiaca`, `nivel_dor` (escala 0-10) e `idade`. As classificações são: **Emergência** (vermelho): temperatura > 39°C OU pressão < 90 OU > 180 OU frequência < 50 OU > 120; **Muito Urgente** (laranja): temperatura entre 38-39°C OU pressão entre 90-100 OU 160-180 OU dor > 7; **Urgente** (amarelo): temperatura entre 37.5-38°C OU dor entre 4-7 OU idade > 65 anos; **Pouco Urgente** (verde): demais casos. Se múltiplas condições se aplicarem, usar a classificação mais alta. Desenvolva o algoritmo para esta triagem.

Análise: o pseudocódigo poderia ser:

```
ALGORITMO ClassificacaoEmergencia

ENTRADA:
    temperatura: real
    pressaoSistolica: inteiro
    frequenciaCardiaca: inteiro
    nivelDor: inteiro
    idade: inteiro

SAÍDA:
    classificacao: texto
    cor: texto

INÍCIO
    ESCREVER "Digite a temperatura (°C):"
    LER temperatura
    ESCREVER "Digite a pressão sistólica:"
    LER pressaoSistolica
    ESCREVER "Digite a frequência cardíaca:"
    LER frequenciaCardiaca
    ESCREVER "Digite o nível de dor (0-10):"
    LER nivelDor
    ESCREVER "Digite a idade:"
    LER idade
```

```
    // Verificar Emergência (vermelho) - prioridade máxima
    SE (temperatura > 39) OU (pressaoSistolica < 90) OU (pressaoSistolica > 180) OU
       (frequenciaCardiaca < 50) OU (frequenciaCardiaca > 120) ENTÃO
        classificacao ← "Emergência"
        cor ← "vermelho"
    // Verificar Muito Urgente (laranja)
    SENÃO SE (temperatura >= 38 E temperatura <= 39) OU
             (pressaoSistolica >= 90 E pressaoSistolica <= 100) OU
             (pressaoSistolica >= 160 E pressaoSistolica <= 180) OU
             (nivelDor > 7) ENTÃO
        classificacao ← "Muito Urgente"
        cor ← "laranja"
    // Verificar Urgente (amarelo)
    SENÃO SE (temperatura >= 37.5 E temperatura < 38) OU
             (nivelDor >= 4 E nivelDor <= 7) OU
             (idade > 65) ENTÃO
        classificacao ← "Urgente"
        cor ← "amarelo"
    // Pouco Urgente (verde) - demais casos
    SENÃO
        classificacao ← "Pouco Urgente"
        cor ← "verde"
    FIM SE

    ESCREVER "Classificação: ", classificacao, " (", cor, ")"

    RETORNAR classificacao, cor
FIM
```

O professor pode sugerir os seguintes casos para validação do algoritmo: emergência por temperatura alta (40°C temperatura, 120 pressão, 80 frequência cardíaca, dor 3, 45 anos), muito urgente por dor severa (37°C temperatura, 130 pressão, 70 frequência cardíaca, dor 8, 30 anos), urgente por idade avançada (36.5°C temperatura, 130 pressão, 70 frequência cardíaca, dor 2, 70 anos), e pouco urgente para caso normal (36.5°C temperatura, 120 pressão, 70 frequência cardíaca, dor 2, 30 anos).

Table 6.21: Teste 1: Emergência por temperatura alta (40°C): resultado: Emergência (vermelho)

Linha	Condição/Ação	Resultado	classificacao	cor	Próximo Passo
ENTRADA	temp ← 40, pressao ← 120, freq ← 80, dor ← 3, idade ← 45	-	indefinido	indefinido	1 linha SE

Linha	Condição/Ação	Resultado	classificacao	cor	Próximo Passo
SE	(temp > 39) OU …	(40 > 39) OU (120 < 90) OU (120 > 180) OU (80 < 50) OU (80 > 120) → VERDADEIRO OU FALSO OU FALSO OU FALSO OU FALSO → VERDADEIRO	“Emergência”	“vermelho”	EM SE
ESCREVER	Classificação: Emergência (vermelho)”	-	“Emergência”	“vermelho”	RETORNAR
RETORNAR	classificacao, cor	-	“Emergência”	“vermelho”	FIM

Table 6.22: Teste 2: Muito urgente por dor severa (dor 8): resultado: Muito Urgente (laranja)

Linha	Condição/Ação	Resultado	classificacao	cor	Próximo Passo
ENTRADA	temp ← 37, pressao ← 130, freq ← 70, dor ← 8, idade ← 30	-	indefinido	indefinido	Linha SE
SE	(temp > 39) OU …	(37 > 39) OU (130 < 90) OU (130 > 180) OU (70 < 50) OU (70 > 120) → FALSO OU FALSO OU FALSO OU FALSO OU FALSO → FALSO	indefinido	indefinido	SENÃO SE

Linha	Condição/Ação	Resultado	classificacao	cor	Próximo Passo
SENÃO SE	(temp >= 38 E temp <= 39) OU … OU (dor > 7)	(37 >= 38 E 37 <= 39) OU … OU (8 > 7) → FALSO OU … OU VERDADEIRO → VERDADEIRO	“Muito Urgente”	“laranja”	FIM SE
ESCREVER	“Classificação: Muito Urgente (laranja)”	-	“Muito Urgente”	“laranja”	RETORNAR
RETORNAR	classificacao, cor	-	“Muito Urgente”	“laranja”	FIM

Table 6.23: Teste 3: Urgente por idade avançada (70 anos): resultado: Urgente (amarelo)

Linha	Condição/Ação	Resultado	classificacao	cor	Próximo Passo
ENTRADA	temp ← 36.5, pressao ← 130, freq ← 70, dor ← 2, idade ← 70	-	indefinido	indefinido	Linha SE
SE	(temp > 39) OU …	(36.5 > 39) OU … → FALSO	indefinido	indefinido	SENÃO SE
SENÃO SE	(temp >= 38 E temp <= 39) OU …	(36.5 >= 38 E 36.5 <= 39) OU … OU (2 > 7) → FALSO OU … OU FALSO → FALSO	indefinido	indefinido	SENÃO SE
SENÃO SE	(temp >= 37.5 E temp < 38) OU … OU (idade > 65)	(36.5 >= 37.5 E 36.5 < 38) OU (2 >= 4 E 2 <= 7) OU (70 > 65) → FALSO OU FALSO OU VERDADEIRO → VERDADEIRO	“Urgente”	“amarelo”	FIM SE
ESCREVER	“Classificação: Urgente (amarelo)”	-	“Urgente”	“amarelo”	RETORNAR
RETORNAR	classificacao, cor	-	“Urgente”	“amarelo”	FIM

Table 6.24: Teste 4: Pouco urgente - caso normal: resultado: Pouco Urgente (verde)

Linha	Condição/Ação	Resultado	classificacao	cor	Próximo Passo
ENTRADA	temp ← 36.5, pressao ← 120, freq ← 70, dor ← 2, idade ← 30	-	indefinido	indefinido	Linha SE
SE	(temp > 39) OU …	FALSO	indefinido	indefinido	SENÃO SE
SENÃO SE	(temp >= 38 E temp <= 39) OU …	FALSO	indefinido	indefinido	SENÃO SE
SENÃO SE	(temp >= 37.5 E temp < 38) OU …	(36.5 >= 37.5 E 36.5 < 38) OU (2 >= 4 E 2 <= 7) OU (30 > 65) → FALSO OU FALSO OU FALSO → FALSO	indefinido	indefinido	SENÃO
SENÃO	classificacao ← “Pouco Urgente”, cor ← “verde”	-	“Pouco Urgente”	“verde”	FIM SE
ESCREVER	“Classificação: Pouco Urgente (verde)”	-	“Pouco Urgente”	“verde”	RETORNAR
RETORNAR	classificacao, cor	-	“Pouco Urgente”	“verde”	FIM

Em resumo:

Table 6.25: Resumo: algoritmo classifica corretamente conforme protocolo de Manchester, priorizando casos mais graves

Caso de Teste	Temp (°C)	Pressão	Freq. Card.	Dor	Idade	Critério Ativado	Classificação
Emergência	40	120	80	3	45	Temperatura > 39°C	Emergência (vermelho)
Muito Urgente	37	130	70	8	30	Dor > 7	Muito Urgente (laranja)
Urgente	36.5	130	70	2	70	Idade > 65 anos	Urgente (amarelo)
Pouco Urgente	36.5	120	70	2	30	Nenhum critério	Pouco Urgente (verde)

A versão mais próxima das linguagens de programação poderia ser:

```
ALGORITMO ClassificacaoEmergenciaConcisa

ENTRADA:
    temperatura, pressaoSistolica, frequenciaCardiaca, nivelDor, idade: numerico

SAÍDA:
    classificacao: texto

INÍCIO
    LER temperatura, pressaoSistolica, frequenciaCardiaca, nivelDor, idade

    SE (temperatura > 39) OU (pressaoSistolica < 90 OU pressaoSistolica > 180) OU
       (frequenciaCardiaca < 50 OU frequenciaCardiaca > 120) ENTÃO
        classificacao ← "Emergência (vermelho)"
    SENÃO SE (temperatura >= 38 E temperatura <= 39) OU
             (pressaoSistolica >= 90 E pressaoSistolica <= 100) OU
             (pressaoSistolica >= 160 E pressaoSistolica <= 180) OU (nivelDor > 7) ENTÃO
        classificacao ← "Muito Urgente (laranja)"
    SENÃO SE (temperatura >= 37.5 E temperatura < 38) OU (nivelDor >= 4 E nivelDor <= 7)
             (idade > 65) ENTÃO
        classificacao ← "Urgente (amarelo)"
    SENÃO
        classificacao ← "Pouco Urgente (verde)"

    RETORNAR classificacao
FIM
```

K. Sistema de Precificação Dinâmica: a empresa de transporte "MoveFast" precisa calcular tarifas dinâmicas. O sistema recebe `distancia_km`, `horario` (0-23), `dia_semana` (1-7), `condicoes_clima` ("normal", "chuva", "tempestade") e `demanda_regiao` ("baixa", "media", "alta"). A tarifa base é distancia_km × 2.50. Aplicar multiplicadores: horários de pico (7-9h, 17-19h): ×1.5; fins de semana (6-7): ×1.2; chuva: ×1.3; tempestade: ×1.8; demanda alta: ×1.4; demanda media: ×1.1. Se for horário de pico E fim de semana E tempestade E demanda alta, aplicar desconto de 10% no valor final (para evitar preços abusivos). Crie o fluxograma deste sistema de precificação.

Análise: o pseudocódigo:

```
ALGORITMO PrecificacaoDinamica

ENTRADA:
    distanciaKm: real
    horario: inteiro
    diaSemana: inteiro
    condicoesClima: texto
    demandaRegiao: texto

SAÍDA:
    tarifaFinal: real

INÍCIO
    ESCREVER "Digite a distância (km):"
    LER distanciaKm
```

```
    ESCREVER "Digite o horário (0-23):"
    LER horario
    ESCREVER "Digite o dia da semana (1-7):"
    LER diaSemana
    ESCREVER "Digite as condições do clima (normal/chuva/tempestade):"
    LER condicoesClima
    ESCREVER "Digite a demanda da região (baixa/media/alta):"
    LER demandaRegiao

    // Calcular tarifa base
    tarifaBase ← distanciaKm * 2.50
    tarifaFinal ← tarifaBase

    // Verificar horário de pico (7-9h ou 17-19h)
    horarioPico ← FALSO
    SE (horario >= 7 E horario <= 9) OU (horario >= 17 E horario <= 19) ENTÃO
        tarifaFinal ← tarifaFinal * 1.5
        horarioPico ← VERDADEIRO
    FIM SE

    // Verificar fim de semana (6=sábado, 7=domingo)
    fimSemana ← FALSO
    SE (diaSemana == 6) OU (diaSemana == 7) ENTÃO
        tarifaFinal ← tarifaFinal * 1.2
        fimSemana ← VERDADEIRO
    FIM SE

    // Verificar condições climáticas
    tempestade ← FALSO
    SE (condicoesClima == "chuva") ENTÃO
        tarifaFinal ← tarifaFinal * 1.3
    SENÃO SE (condicoesClima == "tempestade") ENTÃO
        tarifaFinal ← tarifaFinal * 1.8
        tempestade ← VERDADEIRO
    FIM SE

    // Verificar demanda da região
    demandaAlta ← FALSO
    SE (demandaRegiao == "media") ENTÃO
        tarifaFinal ← tarifaFinal * 1.1
    SENÃO SE (demandaRegiao == "alta") ENTÃO
        tarifaFinal ← tarifaFinal * 1.4
        demandaAlta ← VERDADEIRO
    FIM SE

    // Condição especial de desconto (10% off)
    SE (horarioPico == VERDADEIRO) E (fimSemana == VERDADEIRO) E
       (tempestade == VERDADEIRO) E (demandaAlta == VERDADEIRO) ENTÃO
        tarifaFinal ← tarifaFinal * 0.9
        ESCREVER "Desconto de 10% aplicado para evitar preços abusivos!"
    FIM SE
```

```
    ESCREVER "Tarifa base: R$ ", tarifaBase
    ESCREVER "Tarifa final: R$ ", tarifaFinal

    RETORNAR tarifaFinal
FIM
```

Finalmente, esta é uma oportunidade de discutir tanto a legibilidade do pseudocódigo quanto seu uso para testes e depuração. Neste caso, o professor pode sugerir os seguintes casos para validação do algoritmo: tarifa normal sem multiplicadores (10km, 14h quarta-feira, clima normal, demanda baixa), tarifa com múltiplos multiplicadores (5km, 8h sábado, chuva, demanda média), e o caso especial de desconto para evitar preços abusivos (8km, 18h domingo, tempestade, demanda alta).

Table 6.26: Teste 1: Tarifa normal (10km, dia útil, clima normal, demanda baixa): resultado: R$ 25.00

Linha	Condição/Ação	Resultado	tarifaFinal	Variáveis de Controle	Próximo Passo
ENTRADA	dist ← 10, hora ← 14, dia ← 3, clima ← “normal”, demanda ← “baixa”	-	indefinido	-	Cálculo base
tarifaBase	tarifaBase ← 10 * 2.50 = 25.00	25.00	25.00	-	Verificar pico
SE horário pico	(14 >= 7 E 14 <= 9) OU (14 >= 17 E 14 <= 19) → FALSO OU FALSO → FALSO	-	25.00	horarioPico ← FALSO	Verificar fim semana
SE fim semana	(3 == 6) OU (3 == 7) → FALSO OU FALSO → FALSO	-	25.00	fimSemana ← FALSO	Verificar clima
SE clima	clima == “chuva” → FALSO, clima == “tempestade” → FALSO	-	25.00	tempestade ← FALSO	Verificar demanda

Linha	Condição/Ação	Resultado	tarifaFinal	Variáveis de Controle	Próximo Passo
SE demanda	demanda == “media” → FALSO, demanda == “alta” → FALSO	-	25.00	demandaAlta ← FALSO	Verificar desconto
SE desconto	FALSO E FALSO E FALSO E FALSO → FALSO	-	25.00	-	ESCREVER
RETORNAR	tarifaFinal	-	25.00	-	FIM

Table 6.27: Teste 2: Múltiplos multiplicadores (5km, sábado pico, chuva, demanda média): resultado: R$ 32.18

Linha	Condição/Ação	Resultado	tarifaFinal	Variáveis de Controle	Próximo Passo
ENTRADA	distancia ← 5, hora ← 8, dia ← 6, clima ← “chuva”, demanda ← “media”	-	indefinido	-	Cálculo base
tarifaBase	tarifaBase ← 5 * 2.50 = 12.50	12.50	12.50	-	Verificar pico
SE horário pico	(8 >= 7 E 8 <= 9) OU (8 >= 17 E 8 <= 19) → VERDADEIRO OU FALSO → VERDADEIRO	12.50 * 1.5 = 18.75	18.75	horarioPico ← VERDADEIRO	Verificar fim semana
SE fim semana	(6 == 6) OU (6 == 7) → VERDADEIRO OU FALSO → VERDADEIRO	18.75 * 1.2 = 22.50	22.50	fimSemana ← VERDADEIRO	Verificar clima
SE clima	clima == “chuva” → VERDADEIRO	22.50 * 1.3 = 29.25	29.25	tempestade ← FALSO	Verificar demanda

Linha	Condição/Ação	Resultado	tarifaFinal	Variáveis de Controle	Próximo Passo
SE demanda	demanda == “media” → VERDADEIRO	29.25 * 1.1 = 32.18	32.18	demandaAlta ← FALSO	Verificar desconto
SE desconto	VERDADEIRO E VERDADEIRO E FALSO E FALSO → FALSO	-	32.18	-	ESCREVER
RETORNAR	tarifaFinal	-	32.18	-	FIM

Table 6.28: Teste 3: Caso especial com desconto (8km, domingo pico, tempestade, demanda alta): resultado: R$ 81.65 com desconto

Linha	Condição/Ação	Resultado	tarifaFinal	Variáveis de Controle	Próximo Passo
ENTRADA	dist ← 8, hora ← 18, dia ← 7, clima ← “tempestade”, demanda ← “alta”	-	indefinido	-	Cálculo base
tarifaBase	tarifaBase ← 8 * 2.50 = 20.00	20.00	20.00	-	Verificar pico
SE horário pico	(18 >= 7 E 18 <= 9) OU (18 >= 17 E 18 <= 19) → FALSO OU VERDADEIRO → VERDADEIRO	20.00 * 1.5 = 30.00	30.00	horarioPico ← VERDADEIRO	Verificar fim semana
SE fim semana	(7 == 6) OU (7 == 7) → FALSO OU VERDADEIRO → VERDADEIRO	30.00 * 1.2 = 36.00	36.00	fimSemana ← VERDADEIRO	Verificar clima
SE clima	clima == “tempestade” → VERDADEIRO	36.00 * 1.8 = 64.80	64.80	tempestade ← VERDADEIRO	Verificar demanda

Linha	Condição/Ação	Resultado	tarifaFinal	Variáveis de Controle	Próximo Passo
SE demanda	demanda == "alta" → VERDADEIRO	64.80 * 1.4 = 90.72	90.72	demandaAlta ← VERDADEIRO	Verificar desconto
SE desconto	VERDADEIRO E VERDADEIRO E VERDADEIRO E VERDADEIRO → VERDADEIRO	90.72 * 0.9 = 81.65	81.65	-	ESCREVER
RETORNAR	tarifaFinal	-	81.65	-	FIM

Em resumo:

Table 6.29: Resumo: algoritmo aplica corretamente todos os multiplicadores e o desconto especial para evitar preços abusivos

Caso de Teste	Dist (km)	Horário	Dia	Clima	Demanda	Multiplicadores Aplicados	Tarifa Final	Desconto
Normal	10	14	3 (qua)	normal	baixa	Nenhum	R$ 25.00	Não
Múltiplos	5	8	6 (sáb)	chuva	media	Pico (1.5) + Fim semana (1.2) + Chuva (1.3) + Média (1.1)	R$ 32.18	Não
Especial	8	18	7 (dom)	tempestade	alta	Pico (1.5) + Fim semana (1.2) + Tempestade (1.8) + Alta (1.4) - Desconto (0.9)	R$ 81.65	Sim (10%)

Para ver a transição o professor poderia usar:

```
ALGORITMO PrecificacaoDinamicaConcisa

ENTRADA:
    distanciaKm, horario, diaSemana: numerico
    condicoesClima, demandaRegiao: texto

SAÍDA:
    tarifaFinal: real

INÍCIO
    LER distanciaKm, horario, diaSemana, condicoesClima, demandaRegiao

    tarifaFinal ← distanciaKm * 2.50
```

```
    horarioPico ← ((horario >= 7 E horario <= 9) OU (horario >= 17 E horario <= 19))
    fimSemana ← (diaSemana == 6 OU diaSemana == 7)
    tempestade ← (condicoesClima == "tempestade")
    demandaAlta ← (demandaRegiao == "alta")

    SE horarioPico ENTÃO tarifaFinal ← tarifaFinal * 1.5
    SE fimSemana ENTÃO tarifaFinal ← tarifaFinal * 1.2
    SE (condicoesClima == "chuva") ENTÃO tarifaFinal ← tarifaFinal * 1.3
    SE tempestade ENTÃO tarifaFinal ← tarifaFinal * 1.8
    SE (demandaRegiao == "media") ENTÃO tarifaFinal ← tarifaFinal * 1.1
    SE demandaAlta ENTÃO tarifaFinal ← tarifaFinal * 1.4
    SE (horarioPico E fimSemana E tempestade E demandaAlta) ENTÃO tarifaFinal ← tarifaFin

    RETORNAR tarifaFinal
FIM
```

L. Avaliação de Risco de Investimento: a corretora "InvestSmart", gerenciada pela analista Dra. Patricia, desenvolveu um sistema para classificar perfis de investidor. O sistema recebe `idade`, `renda_mensal`, `patrimonio_liquido`, `experiencia_investimentos` (anos), `tolerancia_perda` (percentual) e `objetivo_prazo` ("curto", "medio", "longo"). O perfil é determinado por pontuação: idade < 30 (+2 pontos), 30-50 (+1 ponto), >50 (0 pontos); renda > 10000 (+2), 5000-10000 (+1), <5000 (0); patrimônio > 100000 (+2), 50000-100000 (+1), <50000 (0); experiência > 5 anos (+2), 2-5 anos (+1), <2 anos (0); tolerância $> 20\%$ (+2), 10-20% (+1), <10% (0); prazo longo (+2), médio (+1), curto (0). Classificação: 0-4 pontos: "Conservador"; 5-8: "Moderado"; 9-12: "Arrojado". Adicionalmente, se idade $>$ 60 E tolerância $< 15\%$, forçar "Conservador" independente da pontuação. Desenvolva este algoritmo de classificação.

Análise: o pseudocódigo poderia ser:

```
ALGORITMO AvaliacaoPerfilInvestidor

ENTRADA:
    idade: inteiro
    rendaMensal: real
    patrimonioLiquido: real
    experienciaInvestimentos: inteiro
    toleranciaPerda: real
    objetivoPrazo: texto

SAÍDA:
    pontuacaoTotal: inteiro
    perfilInvestidor: texto

INÍCIO
    ESCREVER "Digite a idade:"
    LER idade
    ESCREVER "Digite a renda mensal:"
    LER rendaMensal
    ESCREVER "Digite o patrimônio líquido:"
    LER patrimonioLiquido
    ESCREVER "Digite a experiência em investimentos (anos):"
```

```
    LER experienciaInvestimentos
    ESCREVER "Digite a tolerância à perda (%):"
    LER toleranciaPerda
    ESCREVER "Digite o objetivo de prazo (curto/medio/longo):"
    LER objetivoPrazo

    pontuacaoTotal ← 0

    // Pontuação por idade
    SE (idade < 30) ENTÃO
        pontuacaoTotal ← pontuacaoTotal + 2
    SENÃO SE (idade >= 30 E idade <= 50) ENTÃO
        pontuacaoTotal ← pontuacaoTotal + 1
    // idade > 50: +0 pontos
    FIM SE

    // Pontuação por renda
    SE (rendaMensal > 10000) ENTÃO
        pontuacaoTotal ← pontuacaoTotal + 2
    SENÃO SE (rendaMensal >= 5000 E rendaMensal <= 10000) ENTÃO
        pontuacaoTotal ← pontuacaoTotal + 1
    // renda < 5000: +0 pontos
    FIM SE

    // Pontuação por patrimônio
    SE (patrimonioLiquido > 100000) ENTÃO
        pontuacaoTotal ← pontuacaoTotal + 2
    SENÃO SE (patrimonioLiquido >= 50000 E patrimonioLiquido <= 100000) ENTÃO
        pontuacaoTotal ← pontuacaoTotal + 1
    // patrimônio < 50000: +0 pontos
    FIM SE

    // Pontuação por experiência
    SE (experienciaInvestimentos > 5) ENTÃO
        pontuacaoTotal ← pontuacaoTotal + 2
    SENÃO SE (experienciaInvestimentos >= 2 E experienciaInvestimentos <= 5) ENTÃO
        pontuacaoTotal ← pontuacaoTotal + 1
    // experiência < 2: +0 pontos
    FIM SE

    // Pontuação por tolerância
    SE (toleranciaPerda > 20) ENTÃO
        pontuacaoTotal ← pontuacaoTotal + 2
    SENÃO SE (toleranciaPerda >= 10 E toleranciaPerda <= 20) ENTÃO
        pontuacaoTotal ← pontuacaoTotal + 1
    // tolerância < 10: +0 pontos
    FIM SE

    // Pontuação por prazo
    SE (objetivoPrazo == "longo") ENTÃO
        pontuacaoTotal ← pontuacaoTotal + 2
```

```
    SENÃO SE (objetivoPrazo == "medio") ENTÃO
        pontuacaoTotal ← pontuacaoTotal + 1
    // prazo "curto": +0 pontos
    FIM SE

    // Classificação baseada na pontuação
    SE (pontuacaoTotal >= 0 E pontuacaoTotal <= 4) ENTÃO
        perfilInvestidor ← "Conservador"
    SENÃO SE (pontuacaoTotal >= 5 E pontuacaoTotal <= 8) ENTÃO
        perfilInvestidor ← "Moderado"
    SENÃO SE (pontuacaoTotal >= 9 E pontuacaoTotal <= 12) ENTÃO
        perfilInvestidor ← "Arrojado"
    FIM SE

    // Condição especial: forçar conservador para idosos com baixa tolerância
    SE (idade > 60) E (toleranciaPerda < 15) ENTÃO
        perfilInvestidor ← "Conservador"
        ESCREVER "Perfil ajustado para Conservador devido à idade e baixa tolerância ao r
    FIM SE

    ESCREVER "Pontuação total: ", pontuacaoTotal
    ESCREVER "Perfil do investidor: ", perfilInvestidor

    RETORNAR pontuacaoTotal, perfilInvestidor
FIM
```

O professor pode sugerir os seguintes casos para validação do algoritmo: jovem investidor arrojado (25 anos, R$ 12000 renda, R$ 150000 patrimônio, 3 anos experiência, 25% tolerância, prazo longo), investidor de perfil moderado (40 anos, R$ 7000 renda, R$ 70000 patrimônio, 4 anos experiência, 15% tolerância, prazo médio), e o caso especial de idoso forçado a conservador (65 anos, R$ 15000 renda, R$ 200000 patrimônio, 8 anos experiência, 12% tolerância, prazo longo).

Table 6.30: Teste 1: Jovem arrojado (25 anos, alta renda e patrimônio): resultado: 11 pontos - Arrojado

Linha	Condição/Ação	Resultado	pontuacaoTotal	perfilInvestidor	Próximo Passo
ENTRADA	idade ← 25, renda ← 12000, patrimonio ← 150000, exp ← 3, tolerancia ← 25, prazo ← “longo”	-	0	indefinido	Pontuação idade
SE idade	idade < 30 → 25 < 30 → VERDADEIRO	pontuacaoTotal + 2 = 2	2	indefinido	Pontuação renda

Linha	Condição/Ação	Resultado	pontuacaoTotal	perfilInvestidor	Próximo Passo
SE renda	renda > 10000 → 12000 > 10000 → VERDADEIRO	pontuacaoTotal + 2 = 4	4	indefinido	Pontuação patrimônio
SE patrimônio	patrimonio > 100000 → 150000 > 100000 → VERDADEIRO	pontuacaoTotal + 2 = 6	6	indefinido	Pontuação experiência
SE experiência	exp >= 2 E exp <= 5 → 3 >= 2 E 3 <= 5 → VERDADEIRO	pontuacaoTotal + 1 = 7	7	indefinido	Pontuação tolerância
SE tolerância	tolerancia > 20 → 25 > 20 → VERDADEIRO	pontuacaoTotal + 2 = 9	9	indefinido	Pontuação prazo
SE prazo	prazo == “longo” → VERDADEIRO	pontuacaoTotal + 2 = 11	11	indefinido	Classificação
SE classificação	pontuacao >= 9 E pontuacao <= 12 → 11 >= 9 E 11 <= 12 → VERDADEIRO	-	11	“Arrojado”	Condição especial
SE especial	(idade > 60) E (tolerancia < 15) → (25 > 60) E (25 < 15) → FALSO E FALSO → FALSO	-	11	“Arrojado”	ESCREVER
RETORNAR	pontuacaoTotal, perfilInvestidor	-	11	“Arrojado”	FIM

Table 6.31: Teste 2: Investidor moderado (40 anos, renda e patrimônio médios): resultado: 6 pontos - Moderado

Linha	Condição/Ação	Resultado	pontuacaoTotal	perfilInvestidor	Próximo Passo
ENTRADA	idade ← 40, renda ← 7000, patrimonio ← 70000, exp ← 4, tolerancia ← 15, prazo ← “medio”	-	0	indefinido	Pontuação idade
SE idade	idade >= 30 E idade <= 50 → 40 >= 30 E 40 <= 50 → VERDADEIRO	pontuacaoTotal + 1 = 1	1	indefinido	Pontuação renda
SE renda	renda >= 5000 E renda <= 10000 → 7000 >= 5000 E 7000 <= 10000 → VERDADEIRO	pontuacaoTotal + 1 = 2	2	indefinido	Pontuação patrimônio
SE patrimônio	patrimonio >= 50000 E patrimonio <= 100000 → 70000 >= 50000 E 70000 <= 100000 → VERDADEIRO	pontuacaoTotal + 1 = 3	3	indefinido	Pontuação experiência
SE experiência	exp >= 2 E exp <= 5 → 4 >= 2 E 4 <= 5 → VERDADEIRO	pontuacaoTotal + 1 = 4	4	indefinido	Pontuação tolerância
SE tolerância	tolerancia >= 10 E tolerancia <= 20 → 15 >= 10 E 15 <= 20 → VERDADEIRO	pontuacaoTotal + 1 = 5	5	indefinido	Pontuação prazo
SE prazo	prazo == “medio” → VERDADEIRO	pontuacaoTotal + 1 = 6	6	indefinido	Classificação

Linha	Condição/Ação	Resultado	pontuacaoTotal	perfilInvestidor	Próximo Passo
SE classificação	pontuacao >= 5 E pontuacao <= 8 → 6 >= 5 E 6 <= 8 → VERDADEIRO	-	6	“Moderado”	Condição especial
SE especial	(idade > 60) E (tolerancia < 15) → (40 > 60) E (15 < 15) → FALSO E FALSO → FALSO	-	6	“Moderado”	ESCREVER
RETORNAR	pontuacaoTotal, perfilInvestidor	-	6	“Moderado”	FIM

Table 6.32: Teste 3: Caso especial - idoso conservador (65 anos, boa situação mas baixa tolerância): resultado: 9 pontos mas forçado para Conservador

Linha	Condição/Ação	Resultado	pontuacaoTotal	perfilInvestidor	Próximo Passo
ENTRADA	idade ← 65, renda ← 15000, patrimonio ← 200000, exp ← 8, tolerancia ← 12, prazo ← “longo”	-	0	indefinido	Pontuação idade
SE idade	idade > 50 → 65 > 50 → VERDADEIRO (mas sem pontos)	-	0	indefinido	Pontuação renda
SE renda	renda > 10000 → 15000 > 10000 → VERDADEIRO	pontuacaoTotal + 2 = 2	2	indefinido	Pontuação patrimônio
SE patrimônio	patrimonio > 100000 → 200000 > 100000 → VERDADEIRO	pontuacaoTotal + 2 = 4	4	indefinido	Pontuação experiência

Linha	Condição/Ação	Resultado	pontuacaoTotal	perfilInvestidor	Próximo Passo
SE experiência	exp > 5 → 8 > 5 → VERDADEIRO	pontuacaoTotal + 2 = 6	6	indefinido	Pontuação tolerância
SE tolerância	tolerancia >= 10 E tolerancia <= 20 → 12 >= 10 E 12 <= 20 → VERDADEIRO	pontuacaoTotal + 1 = 7	7	indefinido	Pontuação prazo
SE prazo	prazo == “longo” → VERDADEIRO	pontuacaoTotal + 2 = 9	9	indefinido	Classificação
SE classificação	pontuacao >= 9 E pontuacao <= 12 → 9 >= 9 E 9 <= 12 → VERDADEIRO	-	9	“Arrojado”	Condição especial
SE especial	(idade > 60) E (tolerancia < 15) → (65 > 60) E (12 < 15) → VERDADEIRO E VERDADEIRO → VERDADEIRO	-	9	“Conservador”	ESCREVER
RETORNAR	pontuacaoTotal, perfilInvestidor	-	9	“Conservador”	FIM

Em resumo:

Table 6.33: Resumo: algoritmo classifica corretamente perfis de investidor e aplica proteção especial para idosos com baixa tolerância ao risco

Caso de Teste	Idade	Renda	Patrimônio	Exp.	Tolerância	Prazo	Pontos	Perfil Original	Perfil Final	Regra Especial
Jovem arrojado	25	12000	150000	3	25%	longo	11	Arrojado	Arrojado	Não
Moderado	40	7000	70000	4	15%	medio	6	Moderado	Moderado	Não

Caso de Teste	Idade	Renda	Patrimônio	Exp.	Tolerância	Prazo	Pontos	Perfil Original	Perfil Final	Regra Especial
Idoso especial	65	15000	200000	8	12%	longo	9	Arrojado	Conservador	Sim (idade>60 E tolerância<15%)

A versão mais próxima seria:

```
ALGORITMO AvaliacaoPerfilInvestidorConciso

ENTRADA:
    idade, rendaMensal, patrimonioLiquido, experienciaInvestimentos, toleranciaPerda: num
    objetivoPrazo: texto

SAÍDA:
    pontuacaoTotal: inteiro
    perfilInvestidor: texto

INÍCIO
    LER idade, rendaMensal, patrimonioLiquido, experienciaInvestimentos, toleranciaPerda,

    pontuacaoTotal ← 0
    SE (idade < 30) ENTÃO pontuacaoTotal ← pontuacaoTotal + 2
    SENÃO SE (idade <= 50) ENTÃO pontuacaoTotal ← pontuacaoTotal + 1

    SE (rendaMensal > 10000) ENTÃO pontuacaoTotal ← pontuacaoTotal + 2
    SENÃO SE (rendaMensal >= 5000) ENTÃO pontuacaoTotal ← pontuacaoTotal + 1

    SE (patrimonioLiquido > 100000) ENTÃO pontuacaoTotal ← pontuacaoTotal + 2
    SENÃO SE (patrimonioLiquido >= 50000) ENTÃO pontuacaoTotal ← pontuacaoTotal + 1

    SE (experienciaInvestimentos > 5) ENTÃO pontuacaoTotal ← pontuacaoTotal + 2
    SENÃO SE (experienciaInvestimentos >= 2) ENTÃO pontuacaoTotal ← pontuacaoTotal + 1

    SE (toleranciaPerda > 20) ENTÃO pontuacaoTotal ← pontuacaoTotal + 2
    SENÃO SE (toleranciaPerda >= 10) ENTÃO pontuacaoTotal ← pontuacaoTotal + 1

    SE (objetivoPrazo == "longo") ENTÃO pontuacaoTotal ← pontuacaoTotal + 2
    SENÃO SE (objetivoPrazo == "medio") ENTÃO pontuacaoTotal ← pontuacaoTotal + 1

    SE (pontuacaoTotal <= 4) ENTÃO perfilInvestidor ← "Conservador"
    SENÃO SE (pontuacaoTotal <= 8) ENTÃO perfilInvestidor ← "Moderado"
    SENÃO perfilInvestidor ← "Arrojado"

    SE (idade > 60) E (toleranciaPerda < 15) ENTÃO perfilInvestidor ← "Conservador"

    RETORNAR pontuacaoTotal, perfilInvestidor
FIM
```

M. Sistema de Controle de Qualidade Industrial: na fábrica “Precisão Total”, o

engenheiro Carlos implementa controle de qualidade para peças metálicas. O sistema mede `diametro_mm`, `peso_gramas`, `dureza_hrc`, `temperatura_teste` e `lote_producao`. Especificações: diâmetro 50mm ±0.5mm; peso 200g ±10g; dureza 45-55 HRC; temperatura teste entre 20-25°C. Classificação de defeitos: **Classe A** (aprovado): todas especificações atendidas; **Classe B** (aprovado com ressalvas): máximo 1 parâmetro fora da especificação E desvio não superior a 10% do limite; **Classe C** (retrabalho): 2 parâmetros fora OU 1 parâmetro com desvio > 10% mas 20%; **Classe D** (refugo): 3+ parâmetros fora OU qualquer desvio > 20%. Adicionalmente, lotes de produção pares têm tolerância aumentada em 5% para todos os parâmetros. Crie o fluxograma para este sistema de controle de qualidade.

Análise: o pseudocódigo:

```
ALGORITMO ControleQualidadeIndustrial

ENTRADA:
    diametroMm: real
    pesoGramas: real
    durezaHrc: real
    temperaturaTeste: real
    loteProducao: inteiro

SAÍDA:
    classeQualidade: texto
    statusAprovacao: texto

INÍCIO
    ESCREVER "Digite o diâmetro (mm):"
    LER diametroMm
    ESCREVER "Digite o peso (gramas):"
    LER pesoGramas
    ESCREVER "Digite a dureza (HRC):"
    LER durezaHrc
    ESCREVER "Digite a temperatura de teste (°C):"
    LER temperaturaTeste
    ESCREVER "Digite o lote de produção:"
    LER loteProducao

    // Especificações base
    diametroMin ← 49.5
    diametroMax ← 50.5
    pesoMin ← 190
    pesoMax ← 210
    durezaMin ← 45
    durezaMax ← 55
    tempMin ← 20
    tempMax ← 25

    // Verificar se lote é par (tolerância aumentada em 5%)
    SE (loteProducao % 2 == 0) ENTÃO
        diametroMin ← diametroMin - (0.5 * 0.05)
        diametroMax ← diametroMax + (0.5 * 0.05)
        pesoMin ← pesoMin - (10 * 0.05)
```

```
        pesoMax ← pesoMax + (10 * 0.05)
        durezaMin ← durezaMin - (5 * 0.05)
        durezaMax ← durezaMax + (5 * 0.05)
        tempMin ← tempMin - (2.5 * 0.05)
        tempMax ← tempMax + (2.5 * 0.05)
        ESCREVER "Lote par: tolerâncias aumentadas em 5%"
    FIM SE

    parametrosForaEspec ← 0
    maxDesvioPercentual ← 0

    // Verificar diâmetro
    SE (diametroMm < diametroMin) OU (diametroMm > diametroMax) ENTÃO
        parametrosForaEspec ← parametrosForaEspec + 1
        desvioMin ← ABS((diametroMm - diametroMin) / 50) * 100
        desvioMax ← ABS((diametroMm - diametroMax) / 50) * 100
        desvio ← MAIOR(desvioMin, desvioMax)
        SE (desvio > maxDesvioPercentual) ENTÃO maxDesvioPercentual ← desvio
    FIM SE

    // Verificar peso
    SE (pesoGramas < pesoMin) OU (pesoGramas > pesoMax) ENTÃO
        parametrosForaEspec ← parametrosForaEspec + 1
        desvioMin ← ABS((pesoGramas - pesoMin) / 200) * 100
        desvioMax ← ABS((pesoGramas - pesoMax) / 200) * 100
        desvio ← MAIOR(desvioMin, desvioMax)
        SE (desvio > maxDesvioPercentual) ENTÃO maxDesvioPercentual ← desvio
    FIM SE

    // Verificar dureza
    SE (durezaHrc < durezaMin) OU (durezaHrc > durezaMax) ENTÃO
        parametrosForaEspec ← parametrosForaEspec + 1
        desvioMin ← ABS((durezaHrc - durezaMin) / 50) * 100
        desvioMax ← ABS((durezaHrc - durezaMax) / 50) * 100
        desvio ← MAIOR(desvioMin, desvioMax)
        SE (desvio > maxDesvioPercentual) ENTÃO maxDesvioPercentual ← desvio
    FIM SE

    // Verificar temperatura
    SE (temperaturaTeste < tempMin) OU (temperaturaTeste > tempMax) ENTÃO
        parametrosForaEspec ← parametrosForaEspec + 1
        desvioMin ← ABS((temperaturaTeste - tempMin) / 22.5) * 100
        desvioMax ← ABS((temperaturaTeste - tempMax) / 22.5) * 100
        desvio ← MAIOR(desvioMin, desvioMax)
        SE (desvio > maxDesvioPercentual) ENTÃO maxDesvioPercentual ← desvio
    FIM SE

    // Classificação baseada nos critérios
    SE (parametrosForaEspec == 0) ENTÃO
        classeQualidade ← "Classe A"
        statusAprovacao ← "Aprovado"
```

```
    SENÃO SE (parametrosForaEspec == 1) E (maxDesvioPercentual <= 10) ENTÃO
        classeQualidade ← "Classe B"
        statusAprovacao ← "Aprovado com ressalvas"
    SENÃO SE (parametrosForaEspec == 2) OU
             ((parametrosForaEspec == 1) E (maxDesvioPercentual > 10 E maxDesvioPercentua
        classeQualidade ← "Classe C"
        statusAprovacao ← "Retrabalho"
    SENÃO SE (parametrosForaEspec >= 3) OU (maxDesvioPercentual > 20) ENTÃO
        classeQualidade ← "Classe D"
        statusAprovacao ← "Refugo"
    FIM SE

    ESCREVER "Parâmetros fora de especificação: ", parametrosForaEspec
    ESCREVER "Maior desvio percentual: ", maxDesvioPercentual, "%"
    ESCREVER "Classificação: ", classeQualidade
    ESCREVER "Status: ", statusAprovacao

    RETORNAR classeQualidade, statusAprovacao
FIM
```

O professor pode sugerir os seguintes casos para validação do algoritmo: peça perfeita dentro de todas as especificações (50.0mm diâmetro, 200g peso, 50 HRC dureza, 22°C temperatura, lote 101 ímpar), lote par com pequeno desvio no diâmetro (51.0mm diâmetro, 200g peso, 50 HRC dureza, 22°C temperatura, lote 102 par), e peça defeituosa com múltiplos parâmetros fora de especificação (48.0mm diâmetro, 170g peso, 40 HRC dureza, 30°C temperatura, lote 103 ímpar). O professor pode pedir que os alunos verifiquem se o pseudocódigo funciona corretamente para esses casos usando as Tabelas de Rastreio.

Table 6.34: Teste 1: Peça perfeita (todas especificações atendidas): resultado: Classe A - Aprovado

Linha	Condição/Ação	Resultado	parametrosForaEspec	maxDesvioPercentual	classeQualidade	Próximo Passo
ENTRADA	diam ← 50.0, peso ← 200, dureza ← 50, temp ← 22, lote ← 101	-	0	0	indefinido	Verificar lote
SE lote par	101 % 2 == 0 → FALSO	Tolerâncias base mantidas	0	0	indefinido	Verificar diâmetro

Linha	Condição/Ação	Resultado	parametrosForaEspec	desvioPercentual	classeQualidade	Próximo Passo
SE diâmetro	(50.0 < 49.5) OU (50.0 > 50.5) → FALSO OU FALSO → FALSO	Diâmetro OK	0	0	indefinido	Verificar peso
SE peso	(200 < 190) OU (200 > 210) → FALSO OU FALSO → FALSO	Peso OK	0	0	indefinido	Verificar dureza
SE dureza	(50 < 45) OU (50 > 55) → FALSO OU FALSO → FALSO	Dureza OK	0	0	indefinido	Verificar temperatura
SE temperatura	(22 < 20) OU (22 > 25) → FALSO OU FALSO → FALSO	Temperatura OK	0	0	indefinido	Classificação
SE classificação	parametrosForaEspec == 0 → 0 == 0 → VERDADEIRO	–	0	0	“Classe A”	ESCREVER
RETORNAR	classeQualidade, statusAprovacao	–	0	0	“Classe A”	FIM

Table 6.35: Teste 2: Lote par com pequeno desvio no diâmetro: resultado: Classe B - Aprovado com ressalvas

Linha	Condição/Ação	Resultado	parametrosForaEspec	maxDesvioPercentual	classeQualidade	Próximo Passo
ENTRADA	diam ← 51.0, peso ← 200, dureza ← 50, temp ← 22, lote ← 102	-	0	0	indefinido	Verificar lote
SE lote par	102 % 2 == 0 → VERDADEIRO	Tolerâncias aumentadas: diam 49.475-50.525, etc.	0	0	indefinido	Verificar diâmetro
SE diâmetro	(51.0 < 49.475) OU (51.0 > 50.525) → FALSO OU VERDADEIRO → VERDADEIRO	parametrosForaEspec = 1, desvio = (51.0 − 50.525)/50	1	*100 = 0.95%	1	0.95
SE outros	Peso, dureza, temp OK	-	1	0.95	indefinido	Classificação
SE classificação	(parametrosForaEspec == 1) E (maxDesvioPercentual <= 10) → (1 == 1) E (0.95 <= 10) → VERDADEIRO	-	1	0.95	“Classe B”	ESCREVER
RETORNAR	classeQualidade, statusAprovacao	-	1	0.95	“Classe B”	FIM

Table 6.36: Teste 3: Múltiplos parâmetros fora de especificação: resultado: Classe D - Refugo

Linha	Condição/Ação	Resultado	parametrosForaEspec	maxDesvioPercentual	classeQualidade	Próximo Passo
ENTRADA	diam ← 48.0, peso ← 170, dureza ← 40, temp ← 30, lote ← 103	-	0	0	indefinido	Verificar lote
SE lote par	103 % 2 == 0 → FALSO	Tolerâncias base	0	0	indefinido	Verificar parâmetros
Verificações	Diâmetro fora (48.0 < 49.5), Peso fora (170 < 190), Dureza fora (40 < 45), Temp fora (30 > 25)	parametrosForaEspec = 4, maxDesvioPercentual > 20%	4	>20	indefinido	Classificação
SE classificação	parametrosForaEspec >= 3 → 4 >= 3 → VERDADEIRO	-	4	>20	“Classe D”	ESCREVER
RETORNAR	classeQualidade, statusAprovacao	-	4	>20	“Classe D”	FIM

Em resumo:

Table 6.37: Resumo: algoritmo classifica corretamente peças conforme especificações industriais e aplica tolerâncias especiais para lotes pares

Caso de Teste	Diâmetro	Peso	Dureza	Temp	Lote	Parâm. Fora	Maior Desvio	Classe	Status
Perfeita	50.0	200	50	22	101 (ímpar)	0	0%	A	Aprovado
Lote par	51.0	200	50	22	102 (par)	1	0.95%	B	Aprovado c/ ressalvas

Caso de Teste	Diâmetro	Peso	Dureza	Temp	Lote	Parâm. Fora	Maior Desvio	Classe	Status
Defeituosa	48.0	170	40	30	103 (ímpar)	4	>20%	D	Refugo

O pseudocódigo mais próximo:

```
ALGORITMO ControleQualidadeIndustrialConciso

ENTRADA:
    diametroMm, pesoGramas, durezaHrc, temperaturaTeste: real
    loteProducao: inteiro

SAÍDA:
    classeQualidade: texto

INÍCIO
    LER diametroMm, pesoGramas, durezaHrc, temperaturaTeste, loteProducao

    toleranciaExtra ← SE (loteProducao % 2 == 0) ENTÃO 1.05 SENÃO 1.0

    diamMin ← 49.5 * toleranciaExtra
    diamMax ← 50.5 * toleranciaExtra
    pesoMin ← 190 * toleranciaExtra
    pesoMax ← 210 * toleranciaExtra
    durezaMin ← 45 * toleranciaExtra
    durezaMax ← 55 * toleranciaExtra
    tempMin ← 20 * toleranciaExtra
    tempMax ← 25 * toleranciaExtra

    parametrosForaEspec ← 0
    maxDesvio ← 0

    SE (diametroMm < diamMin) OU (diametroMm > diamMax) ENTÃO
        parametrosForaEspec ← parametrosForaEspec + 1
        desvio ← SE (diametroMm < diamMin) ENTÃO (diamMin - diametroMm) / 50 * 100 SENÃO
        SE (desvio > maxDesvio) ENTÃO maxDesvio ← desvio

    SE (pesoGramas < pesoMin) OU (pesoGramas > pesoMax) ENTÃO
        parametrosForaEspec ← parametrosForaEspec + 1
        desvio ← SE (pesoGramas < pesoMin) ENTÃO (pesoMin - pesoGramas) / 200 * 100 SENÃO
        SE (desvio > maxDesvio) ENTÃO maxDesvio ← desvio

    SE (durezaHrc < durezaMin) OU (durezaHrc > durezaMax) ENTÃO
        parametrosForaEspec ← parametrosForaEspec + 1
        desvio ← SE (durezaHrc < durezaMin) ENTÃO (durezaMin - durezaHrc) / 50 * 100 SENÃ
        SE (desvio > maxDesvio) ENTÃO maxDesvio ← desvio

    SE (temperaturaTeste < tempMin) OU (temperaturaTeste > tempMax) ENTÃO
        parametrosForaEspec ← parametrosForaEspec + 1
```

```
        desvio ← SE (temperaturaTeste < tempMin) ENTÃO (tempMin - temperaturaTeste) / 22.
        SE (desvio > maxDesvio) ENTÃO maxDesvio ← desvio

    SE (parametrosForaEspec == 0) ENTÃO classeQualidade ← "Classe A"
    SENÃO SE (parametrosForaEspec == 1) E (maxDesvio <= 10) ENTÃO classeQualidade ← "Clas
    SENÃO SE (parametrosForaEspec == 2) OU (parametrosForaEspec == 1 E maxDesvio <= 20) E
    SENÃO classeQualidade ← "Classe D"

    RETORNAR classeQualidade
FIM
```

N. Sistema de Avaliação para Promoção Corporativa: na empresa "Crescer Juntos", a gestora de RH Dra. Ana Carolina precisa automatizar o processo de avaliação para promoções. O sistema deve analisar múltiplos critérios dos funcionários: `tempo_empresa` (anos), `avaliacao_desempenho` (1-10), `nivel_formacao` ("medio", "superior", "pos"), `certificacoes_profissionais` (quantidade), `idade` e `salario_atual`. As regras de classificação são: **Promoção Imediata**: tempo 3 anos E avaliação 8 E (formação superior OU pós) E certificações 2; **Promoção Condicional**: tempo 2 anos E avaliação 7 E pelo menos um dos critérios extras (formação superior/pós OU certificações 1); **Desenvolvimento Necessário**: tempo 1 ano E avaliação 6; **Não Elegível**: demais casos. Adicionalmente, se idade $>$ 55 anos E salário $>$ R$ 15000, aplicar regra especial "Plano Sucessão" independente dos outros critérios. Desenvolva este algoritmo de classificação.

Análise: o pseudocódigo poderia ser:

```
ALGORITMO AvaliacaoPromocao

ENTRADA:
    tempoEmpresa: real
    avaliacaoDesempenho: real
    nivelFormacao: texto
    certificacoesProfissionais: inteiro
    idade: inteiro
    salarioAtual: real

SAÍDA:
    classificacao: texto
    statusPromocao: texto

INÍCIO
    ESCREVER "Digite o tempo na empresa (anos):"
    LER tempoEmpresa
    ESCREVER "Digite a avaliação de desempenho (1-10):"
    LER avaliacaoDesempenho
    ESCREVER "Digite o nível de formação (medio/superior/pos):"
    LER nivelFormacao
    ESCREVER "Digite o número de certificações profissionais:"
    LER certificacoesProfissionais
    ESCREVER "Digite a idade:"
    LER idade
    ESCREVER "Digite o salário atual:"
    LER salarioAtual
```

```
    // Verificar regra especial primeiro
    SE (idade > 55) E (salarioAtual > 15000) ENTÃO
        classificacao ← "Plano Sucessão"
        statusPromocao ← "Preparação para transição de conhecimento"
        ESCREVER "Regra especial aplicada: Plano Sucessão ativado."
    SENÃO
        // Verificar formação superior ou pós
        formacaoAvancada ← FALSO
        SE (nivelFormacao == "superior") OU (nivelFormacao == "pos") ENTÃO
            formacaoAvancada ← VERDADEIRO
        FIM SE

        // Verificar Promoção Imediata
        SE (tempoEmpresa >= 3) E (avaliacaoDesempenho >= 8) E
           (formacaoAvancada == VERDADEIRO) E (certificacoesProfissionais >= 2) ENTÃO
            classificacao ← "Promoção Imediata"
            statusPromocao ← "Aprovado para promoção no próximo ciclo"

        // Verificar Promoção Condicional
        SENÃO SE (tempoEmpresa >= 2) E (avaliacaoDesempenho >= 7) E
                 ((formacaoAvancada == VERDADEIRO) OU (certificacoesProfissionais >= 1))
            classificacao ← "Promoção Condicional"
            statusPromocao ← "Aprovado mediante cumprimento de requisitos adicionais"

        // Verificar Desenvolvimento Necessário
        SENÃO SE (tempoEmpresa >= 1) E (avaliacaoDesempenho >= 6) ENTÃO
            classificacao ← "Desenvolvimento Necessário"
            statusPromocao ← "Participar de programa de desenvolvimento antes da promoção

        // Não Elegível
        SENÃO
            classificacao ← "Não Elegível"
            statusPromocao ← "Não atende aos critérios mínimos para promoção"
        FIM SE
    FIM SE

    ESCREVER "Classificação: ", classificacao
    ESCREVER "Status: ", statusPromocao

    RETORNAR classificacao, statusPromocao
FIM
```

Casos de teste: funcionário para promoção imediata (4 anos empresa, avaliação 9, formação superior, 3 certificações, 30 anos, R$ 8000), promoção condicional (2.5 anos empresa, avaliação 7.5, formação média, 1 certificação, 28 anos, R$ 5000), e caso especial do plano sucessão (10 anos empresa, avaliação 8, formação pós, 5 certificações, 58 anos, R$ 18000). O professor pode pedir que os alunos verifiquem se o pseudocódigo funciona corretamente para esses casos usando as Tabelas de Rastreio.

Table 6.38: Teste 1: Promoção imediata (4 anos, avaliação 9, superior, 3 certificações): resultado: Promoção Imediata

Linha	Condição/Ação	Resultado	classificacao	formacaoAvancada	statusPromocao	Próximo Passo
ENTRADA	tempo ← 4, aval ← 9, form ← “superior”, cert ← 3, idade ← 30, sal ← 8000	-	indefinido	indefinido	indefinido	Regra especial
SE especial	(idade > 55) E (salario > 15000) → (30 > 55) E (8000 > 15000) → FALSO E FALSO → FALSO	-	indefinido	indefinido	indefinido	SENÃO
formacaoAvancada	(form == “superior”) OU (form == “pos”) → (“superior” == “superior”) OU (“superior” == “pos”) → VERDADEIRO OU FALSO → VERDADEIRO	-	indefinido	VERDADEIRO	indefinido	Promoção Imediata

Linha	Condição/Ação	Resultado	classificacao	formacaoAvancada	statusPromocao	Próximo Passo
SE Imediata	(tempo >= 3) E (aval >= 8) E (formAvancada == VERDADEIRO) E (cert >= 2) → (4 >= 3) E (9 >= 8) E (VERDADEIRO) E (3 >= 2) → VERDADEIRO	-	“Promoção Imediata”	VERDADEIRO	“Aprovado para promoção no próximo ciclo”	FIM SE
ESCREVER	Classificação: Promoção Imediata”	-	“Promoção Imediata”	VERDADEIRO	“Aprovado para promoção no próximo ciclo”	ESCREVER
RETORNAR	classificacao, statusPromocao	-	“Promoção Imediata”	VERDADEIRO	“Aprovado para promoção no próximo ciclo”	FIM

Table 6.39: Teste 2: Promoção condicional (2.5 anos, avaliação 7.5, ensino médio, 1 certificação): resultado: Promoção Condicional

Linha	Condição/Ação	Resultado	classificacao	formacaoAvancada	statusPromocao	Próximo Passo
ENTRADA	tempo ← 2.5, aval ← 7.5, form ← “medio”, cert ← 1, idade ← 28, sal ← 5000	-	indefinido	indefinido	indefinido	Regra especial
SE especial	(28 > 55) E (5000 > 15000) → FALSO E FALSO → FALSO	-	indefinido	indefinido	indefinido	SENÃO

Linha	Condição/Ação	Resultado	classificacao	formacaoAvancada	statusPromocao	Próximo Passo
formacaoAvancada	(formAvancada == “superior”) OU (form == “pos”) → (“medio” == “superior”) OU (“medio” == “pos”) → FALSO OU FALSO → FALSO	-	indefinido	FALSO	indefinido	Promoção Imediata
SE Imediata	(2.5 >= 3) E (7.5 >= 8) E (FALSO) E (1 >= 2) → FALSO E FALSO E FALSO E FALSO → FALSO	-	indefinido	FALSO	indefinido	SENÃO SE Condicional
SE Condicional	(tempo >= 2) E (aval >= 7) E ((formAvancada == VERDADEIRO) OU (cert >= 1)) → (2.5 >= 2) E (7.5 >= 7) E ((FALSO) OU (1 >= 1)) → VERDADEIRO E VERDADEIRO E (FALSO OU VERDADEIRO) → VERDADEIRO	-	“Promoção Condicional”	FALSO	“Aprovado mediante cumprimento de requisitos adicionais”	FIM SE

Linha	Condição/Ação	Resultado	classificacao	formacaoAvancada	statusPromocao	Próximo Passo
RETORNAR	NAR classificacao, statusPromocao	-	“Promoção Condicional”	FALSO	“Aprovado mediante cumprimento de requisitos adicionais”	FIM

Table 6.40: Teste 3: Caso especial - plano sucessão (58 anos, R$ 18000): resultado: Plano Sucessão

Linha	Condição/Ação	Resultado	classificacao	formacaoAvancada	statusPromocao	Próximo Passo
ENTRADA	tempo ← 10, aval ← 8, form ← “pos”, cert ← 5, idade ← 58, sal ← 18000	-	indefinido	indefinido	indefinido	Regra especial
SE especial	(idade > 55) E (salario > 15000) → (58 > 55) E (18000 > 15000) → VERDADEIRO E VERDADEIRO → VERDADEIRO	-	“Plano Sucessão”	indefinido	“Preparação para transição de conhecimento”	ESCREVER
ESCREVER	Regra especial aplicada: Plano Sucessão ativado.”	-	“Plano Sucessão”	indefinido	“Preparação para transição de conhecimento”	ESCREVER
ESCREVER	Classificação: Plano Sucessão”	-	“Plano Sucessão”	indefinido	“Preparação para transição de conhecimento”	ESCREVER
RETORNAR	NAR classificacao, statusPromocao	-	“Plano Sucessão”	indefinido	“Preparação para transição de conhecimento”	FIM

Table 6.41: Teste 4: Desenvolvimento necessário (1.5 anos, avaliação 6.5): resultado: Desenvolvimento Necessário

Linha	Condição/Ação	Resultado	classificacao	formacaoAvancada	statusPromocao	Próximo Passo
ENTRADA	tempo ← 1.5, aval ← 6.5, form ← “superior”, cert ← 0, idade ← 25, sal ← 4000	-	indefinido	indefinido	indefinido	Regra especial
SE especial	(25 > 55) E (4000 > 15000) → FALSO E FALSO → FALSO	-	indefinido	indefinido	indefinido	SENÃO
formacaoAvancada	(“superior” == “superior”) OU (“superior” == “pos”) → VERDADEIRO OU FALSO → VERDADEIRO	-	indefinido	VERDADEIRO	indefinido	Verificações
SE Imediata	(1.5 >= 3) E (6.5 >= 8) E (VERDADEIRO) E (0 >= 2) → FALSO E FALSO E VERDADEIRO E FALSO → FALSO	-	indefinido	VERDADEIRO	indefinido	SENÃO SE Condicional

Linha	Condição/Ação	Resultado	classificacao	formacaoAvancada	statusPromocao	Próximo Passo
SE Condicional	(1.5 >= 2) E (6.5 >= 7) E ((VERDADEIRO) OU (0 >= 1)) → FALSO E FALSO E (VERDADEIRO OU FALSO) → FALSO	-	indefinido	VERDADEIRO	indefinido	SENÃO SE Desenvolvimento
SE Desenvolvimento	(tempo >= 1) E (aval >= 6) → (1.5 >= 1) E (6.5 >= 6) → VERDADEIRO E VERDADEIRO → VERDADEIRO	-	“Desenvolvimento Necessário”	VERDADEIRO	“Participar de programa de desenvolvimento antes da promoção”	FIM SE
RETORNAR	Classificacao, statusPromocao	-	“Desenvolvimento Necessário”	VERDADEIRO	“Participar de programa de desenvolvimento antes da promoção”	FIM

Em resumo:

Table 6.42: Resumo: algoritmo classifica corretamente funcionários para promoção e aplica regra especial do plano sucessão

Caso de Teste	Tempo	Avaliação	Formação	Certificações	Idade	Salário	Critérios Atendidos	Classificação
Promoção imediata	4 anos	9.0	Superior	3	30	R$ 8000	Todos para imediata	Promoção Imediata
Promoção condicional	2.5 anos	7.5	Médio	1	28	R$ 5000	Tempo + avaliação + certificação	Promoção Condicional
Plano sucessão	10 anos	8.0	Pós	5	58	R$ 18000	Regra especial (idade>55 + salário>15k)	Plano Sucessão

Caso de Teste	Tempo	Avaliação	Formação	Certificações	Idade	Salário	Critérios Atendidos	Classificação
Desenvolvimento	1.5 anos	6.5	Superior	0	25	R$ 4000	Apenas tempo mínimo + avaliação básica	Desenvolvimento Necessário

Em seguida, o professor pode apresentar uma versão mais próxima das linguagens de programação, como C++ ou Python, para que os alunos vejam a transição do pseudocódigo para o código real.

```
ALGORITMO AvaliacaoPromocaoConciso

ENTRADA:
    tempoEmpresa, avaliacaoDesempenho, certificacoesProfissionais, idade, salarioAtual: n
    nivelFormacao: texto

SAÍDA:
    classificacao: texto

INÍCIO
    LER tempoEmpresa, avaliacaoDesempenho, nivelFormacao, certificacoesProfissionais, ida

    SE (idade > 55) E (salarioAtual > 15000) ENTÃO
        classificacao ← "Plano Sucessão"
    SENÃO
        formacaoAvancada ← (nivelFormacao == "superior") OU (nivelFormacao == "pos")

        SE (tempoEmpresa >= 3) E (avaliacaoDesempenho >= 8) E formacaoAvancada E (certifi
            classificacao ← "Promoção Imediata"
        SENÃO SE (tempoEmpresa >= 2) E (avaliacaoDesempenho >= 7) E (formacaoAvancada OU
            classificacao ← "Promoção Condicional"
        SENÃO SE (tempoEmpresa >= 1) E (avaliacaoDesempenho >= 6) ENTÃO
            classificacao ← "Desenvolvimento Necessário"
        SENÃO
            classificacao ← "Não Elegível"

    RETORNAR classificacao
FIM
```

6.1.3 Atividades *Plugged*: Programação C++23

A linguagem C++23 (106) é uma linguagem de programação de propósito geral, que suporta programação procedural, orientada a objetos e genérica. Esta linguagem é adequada à metodologia **DAA** graças ao artefato sintático `goto`, que permitirá que o aluno construa a cognição necessária ao entendimento dos laços de repetição antes de utilizar artefatos complexos como `for` e `while`.

Ainda estamos no primeiro módulo e o professor precisa ter cuidado com o nível de stress cognitivo ao qual sujeita os alunos. Dessa forma, o uso de ambientes de desenvolvimento

online parece ser mais adequado já que remove a necessidade de instalação de aplicativos específicos. Neste caso, estão disponíveis online:

1. Replit
2. OnlineGDB
3. C++ OnLine Compiler)
4. OnLine CPP

Cabe ao professor, testar e escolher entre estas opções, ou qualquer outra disponível online o ambiente mais adequado às suas características e necessidades. Uma vez definido o ambiente, as atividades devem ser baseadas nas atividades anteriores.

Objetivo: transformar em programas os algoritmos desenvolvidos nas tarefas anteriores.

Material: ambiente de desenvolvimento online, enunciados e pseudocódigo dos exercícios **A** até **N**.

O professor deve começar com um exercício simples para explicar o contexto do ambiente de desenvolvimento online e o básico da linguagem C++. Escolha entre os exercícios da Section 6.1.2 e repetir o ciclo de exercícios baseado na Técnica da Sequência de Fibonacci. A seguir estão os exemplos de uma sequência possível.

Atenção: os códigos a seguir utiliza o C++23 com a liberdade permitida pelos compiladores modernos. Assim, foi possível fazer todos os exemplos iniciais sem usar as bibliotecas de manipulação de *strings* além disso, o código irá parecer excessivamente complexo para que já conhece as abstrações de alto nível, mas é necessário para que os alunos possam criar as estruturas cognitivas necessárias ao **Raciocínio Algorítmico**. Por isso, não usamos as funções para cálculo de seno, cosseno, raiz quadrada, etc..

G. **Sistema de Acesso a Recursos**: a biblioteca universitária "Conhecimento Aberto" está testando um novo sistema de acesso online aos seus tomos. Para a fase de testes, existem um `nome_Usuário` ("admin") e uma `senha` ("123") pré-definidos. O objetivo é que os usuários insiram suas credenciais e o sistema verifique se o `login` foi bem-sucedido ou falhou. Usando apenas o `nome_Usuário` ("admin") e a `senha` ("123"), desenvolva um algoritmo, na forma de fluxograma, que simule este processo de login.

O pseudocódigo poderia ser:

```
ALGORITMO SistemaLoginConciso

CONSTANTES:
    USUARIO_VALIDO ← "admin"
    SENHA_VALIDA ← "123"

ENTRADA:
    nomeUsuário: texto
    senha: texto

SAÍDA:
    loginSucesso: booleano

INÍCIO
    LER nomeUsuário, senha
    loginSucesso ← (nomeUsuário == USUARIO_VALIDO) E (senha == SENHA_VALIDA)
    RETORNAR loginSucesso
FIM
```

Este algoritmo pode ser representado com o Listing 6.1.

Listing 6.1

```cpp
#include <iostream>

int main() {
    // CONSTANTES
    const std::string USUARIO_VALIDO = "admin";
    const std::string SENHA_VALIDA = "123";

    // ENTRADA
    std::string nomeUsuário, senha;

    std::cout << "Nome de Usuário: ";
    std::cin >> nomeUsuário;

    std::cout << "Senha: ";
    std::cin >> senha;

    // PROCESSAMENTO
    bool loginSucesso = (nomeUsuário == USUARIO_VALIDO) && (senha == SENHA_VALIDA);

    // SAÍDA
    if (loginSucesso) {
        std::cout << "Login bem-sucedido!" << std::endl;
    } else {
        std::cout << "Login falhou." << std::endl;
    }

    return 0;
}
```

O Listing 6.1 pode ser executado em qualquer ambiente de desenvolvimento C++ online, e está disponível no OnlineGDB. O professor deve refazer as tabelas de rastreio para este algoritmo, e sugerir casos de teste como: credenciais corretas (admin/123), nome correto com senha errada (admin/456), nome errado com senha correta (user/123), e ambas credenciais erradas (user/456). O professor pode pedir que os alunos verifiquem se o pseudocódigo funciona corretamente para esses casos usando as Tabelas de Rastreio.

H. Avaliação de Saúde Ocupacional: em uma clínica de saúde ocupacional, a enfermeira Joana Mãoleve realiza avaliações de rotina, incluindo o cálculo do Índice de Massa Corporal (IMC) dos funcionários. O IMC é calculado pela fórmula IMC = frac{peso}{altura * altura}, na qual o `peso` é em quilogramas e a `altura` em metros. Após o cálculo, o IMC é classificado para indicar o estado nutricional. Ajude a enfermeira Joana Mãoleve a criar um algoritmo, na forma de fluxograma, que, dados o `peso` e a `altura` de um funcionário, calcule o IMC e o classifique como:

- Menor que 18.5: “Abaixo do peso”
- Entre 18.5 e 24.9: “Peso normal”
- Entre 25 e 29.9: “Sobrepeso”
- 30 ou mais: “Obesidade”

Análise: o pseudocódigo poderia ser:

```
ALGORITMO AvaliacaoIMCConciso

ENTRADA:
    peso: real
    altura: real

SAÍDA:
    imc: real
    classificacao: texto

INÍCIO
    LER peso, altura
    imc ← peso / (altura * altura)

    SE (imc < 18.5) ENTÃO classificacao ← "Abaixo do peso"
    SENÃO SE (imc <= 24.9) ENTÃO classificacao ← "Peso normal"
    SENÃO SE (imc <= 29.9) ENTÃO classificacao ← "Sobrepeso"
    SENÃO classificacao ← "Obesidade"

    RETORNAR imc, classificacao
FIM
```

Este pseudocódigo pode ser representado com o Listing 6.2.

O código Listing 6.1 pode ser executado em qualquer ambiente de desenvolvimento C++ online, e está disponível no OnlineGDB.

O professor deve refazer as tabelas de rastreio para este algoritmo, e sugerir casos de teste como: IMC abaixo do peso (45kg, 1.70m), IMC normal (70kg, 1.75m), IMC sobrepeso (80kg, 1.70m), e IMC obesidade (100kg, 1.70m). O professor pode pedir que os alunos verifiquem se o pseudocódigo funciona corretamente para esses casos usando as Tabelas de Rastreio.

I. Sistema de Avaliação de Crédito Bancário: no banco "Futuro Seguro", o analista Roberto precisa automatizar a aprovação de empréstimos pessoais. O sistema deve considerar múltiplos critérios: `renda_mensal`, `idade`, `score_credito` e `tempo_emprego` (em anos). As regras são: (1) Renda mínima de 2000 reais; (2) Idade entre 21 e 65 anos; (3) Score de crédito de pelo menos 600; (4) Tempo de emprego de pelo menos 2 anos. Se todos os critérios forem atendidos, calcular o `valor_maximo_emprestimo` como renda_mensal $\times$ 10. Se o score for maior que 750, aplicar bônus de 20% no valor máximo. Se algum critério não for atendido, negar o empréstimo. Crie um fluxograma que implemente este sistema de avaliação.

Análise: o pseudocódigo poderia ser:

```
ALGORITMO AvaliacaoCreditoBancarioConciso

ENTRADA:
    rendaMensal, idade, scoreCredito, tempoEmprego: numerico

SAÍDA:
    aprovado: booleano
    valorMaximoEmprestimo: real

INÍCIO
    LER rendaMensal, idade, scoreCredito, tempoEmprego
```

Listing 6.2

```cpp
#include <iostream>

int main() {
    // ENTRADA
    double peso, altura;

    std::cout << "Peso (kg): ";
    std::cin >> peso;

    std::cout << "Altura (m): ";
    std::cin >> altura;

    // PROCESSAMENTO
    double imc = peso / (altura * altura);
    std::string classificacao;

    if (imc < 18.5) {
        classificacao = "Abaixo do peso";
    } else if (imc <= 24.9) {
        classificacao = "Peso normal";
    } else if (imc <= 29.9) {
        classificacao = "Sobrepeso";
    } else {
        classificacao = "Obesidade";
    }

    // SAÍDA
    std::cout << "IMC: " << imc << std::endl;
    std::cout << "Classificação: " << classificacao << std::endl;

    return 0;
}
```

```
    aprovado ← (rendaMensal >= 2000) E (idade >= 21 E idade <= 65) E
               (scoreCredito >= 600) E (tempoEmprego >= 2)

    SE aprovado ENTÃO
        valorMaximoEmprestimo ← rendaMensal * 10
        SE (scoreCredito > 750) ENTÃO
            valorMaximoEmprestimo ← valorMaximoEmprestimo * 1.2
        valorMaximoEmprestimo ← valorMaximoEmprestimo
    SENÃO
        valorMaximoEmprestimo ← 0

    RETORNAR aprovado, valorMaximoEmprestimo
FIM
```

Este pseudocódigo pode ser representado com o Listing 6.3.

O código Listing 6.3 pode ser executado em qualquer ambiente de desenvolvimento C++ online, e está disponível no OnlineGDB.

O professor deve validar o código usando: cliente aprovado sem bônus (R$ 5000 renda, 35 anos, score 700, 5 anos emprego), cliente aprovado com bônus (R$ 8000 renda, 28 anos, score 800, 3 anos emprego), negação por renda baixa (R$ 1500 renda, 30 anos, score 650, 3 anos emprego), e negação por múltiplos critérios (R$ 1800 renda, 19 anos, score 550, 1 ano emprego). O professor pode pedir que os alunos verifiquem se o pseudocódigo funciona corretamente para esses casos usando as Tabelas de Rastreio.

J. Classificação de Emergências Hospitalares: no Hospital "Santa Esperança", a enfermeira Carla utiliza o protocolo de Manchester para classificar a urgência dos pacientes. O sistema recebe `temperatura`, `pressao_sistolica`, `frequencia_cardiaca`, `nivel_dor` (escala 0-10) e `idade`. As classificações são: **Emergência** (vermelho): temperatura > 39°C OU pressão < 90 OU > 180 OU frequência < 50 OU > 120; **Muito Urgente** (laranja): temperatura entre 38-39°C OU pressão entre 90-100 OU 160-180 OU dor > 7; **Urgente** (amarelo): temperatura entre 37.5-38°C OU dor entre 4-7 OU idade > 65 anos; **Pouco Urgente** (verde): demais casos. Se múltiplas condições se aplicarem, usar a classificação mais alta. Desenvolva o algoritmo para esta triagem.

Análise: o pseudocódigo poderia ser:

```
ALGORITMO ClassificacaoEmergenciaConcisa

ENTRADA:
    temperatura, pressaoSistolica, frequenciaCardiaca, nivelDor, idade: numerico

SAÍDA:
    classificacao: texto

INÍCIO
    LER temperatura, pressaoSistolica, frequenciaCardiaca, nivelDor, idade

    SE (temperatura > 39) OU (pressaoSistolica < 90 OU pressaoSistolica > 180) OU
       (frequenciaCardiaca < 50 OU frequenciaCardiaca > 120) ENTÃO
        classificacao ← "Emergência (vermelho)"
    SENÃO SE (temperatura >= 38 E temperatura <= 39) OU
             (pressaoSistolica >= 90 E pressaoSistolica <= 100) OU
             (pressaoSistolica >= 160 E pressaoSistolica <= 180) OU (nivelDor > 7) ENTÃO
        classificacao ← "Muito Urgente (laranja)"
    SENÃO SE (temperatura >= 37.5 E temperatura < 38) OU (nivelDor >= 4 E nivelDor <= 7)
             (idade > 65) ENTÃO
        classificacao ← "Urgente (amarelo)"
    SENÃO
        classificacao ← "Pouco Urgente (verde)"

    RETORNAR classificacao
FIM
```

O algoritmo pode ser representado com o Listing 6.4.

O código Listing 6.4 pode ser executado em qualquer ambiente de desenvolvimento C++ online, e está disponível no OnlineGDB.

O professor pode verificar os seguintes casos para validação do algoritmo: emergência por temperatura alta (40°C temperatura, 120 pressão, 80 frequência cardíaca, dor 3, 45 anos), muito urgente por dor severa (37°C temperatura, 130 pressão, 70 frequência cardíaca, dor 8, 30

anos), urgente por idade avançada (36.5°C temperatura, 130 pressão, 70 frequência cardíaca, dor 2, 70 anos), e pouco urgente para caso normal (36.5°C temperatura, 120 pressão, 70 frequência cardíaca, dor 2, 30 anos).

K. Sistema de Precificação Dinâmica: a empresa de transporte "MoveFast" precisa calcular tarifas dinâmicas. O sistema recebe `distancia_km`, `horario` (0-23), `dia_semana` (1-7), `condicoes_clima` ("normal", "chuva", "tempestade") e `demanda_regiao` ("baixa", "media", "alta"). A tarifa base é distancia_km × 2.50. Aplicar multiplicadores: horários de pico (7-9h, 17-19h): ×1.5; fins de semana (6-7): ×1.2; chuva: ×1.3; tempestade: ×1.8; demanda alta: ×1.4; demanda media: ×1.1. Se for horário de pico E fim de semana E tempestade E demanda alta, aplicar desconto de 10% no valor final (para evitar preços abusivos). Crie o fluxograma deste sistema de precificação.

Análise: o pseudocódigo:

```
ALGORITMO PrecificacaoDinamicaConcisa

ENTRADA:
    distanciaKm, horario, diaSemana: numerico
    condicoesClima, demandaRegiao: texto

SAÍDA:
    tarifaFinal: real

INÍCIO
    LER distanciaKm, horario, diaSemana, condicoesClima, demandaRegiao

    tarifaFinal ← distanciaKm * 2.50
    horarioPico ← ((horario >= 7 E horario <= 9) OU (horario >= 17 E horario <= 19))
    fimSemana ← (diaSemana == 6 OU diaSemana == 7)
    tempestade ← (condicoesClima == "tempestade")
    demandaAlta ← (demandaRegiao == "alta")

    SE horarioPico ENTÃO tarifaFinal ← tarifaFinal * 1.5
    SE fimSemana ENTÃO tarifaFinal ← tarifaFinal * 1.2
    SE (condicoesClima == "chuva") ENTÃO tarifaFinal ← tarifaFinal * 1.3
    SE tempestade ENTÃO tarifaFinal ← tarifaFinal * 1.8
    SE (demandaRegiao == "media") ENTÃO tarifaFinal ← tarifaFinal * 1.1
    SE demandaAlta ENTÃO tarifaFinal ← tarifaFinal * 1.4
    SE (horarioPico E fimSemana E tempestade E demandaAlta) ENTÃO tarifaFinal ← tarifaFin

    RETORNAR tarifaFinal
FIM
```

O pseucódigo pode ser representado com o Listing 6.5.

O código Listing 6.5 pode ser executado em qualquer ambiente de desenvolvimento C++ online, e está disponível no OnlineGDB.

Neste caso, o professor pode sugerir os seguintes casos para validação do algoritmo: tarifa normal sem multiplicadores (10km, 14h quarta-feira, clima normal, demanda baixa), tarifa com múltiplos multiplicadores (5km, 8h sábado, chuva, demanda média), e o caso especial de desconto para evitar preços abusivos (8km, 18h domingo, tempestade, demanda alta).

L. Avaliação de Risco de Investimento: a corretora "InvestSmart", gerenciada pela analista Dra. Patricia, desenvolveu um sistema para classificar perfis de investidor. O sistema recebe `idade`, `renda_mensal`, `patrimonio_liquido`, `experiencia_investimentos` (anos),

`tolerancia_perda` (percentual) e `objetivo_prazo` (“curto”, “medio”, “longo”). O perfil é determinado por pontuação: idade < 30 (+2 pontos), 30-50 (+1 ponto), >50 (0 pontos); renda > 10000 (+2), 5000-10000 (+1), <5000 (0); patrimônio > 100000 (+2), 50000-100000 (+1), <50000 (0); experiência > 5 anos (+2), 2-5 anos (+1), <2 anos (0); tolerância $> 20\%$ (+2), 10-20% (+1), <10% (0); prazo longo (+2), médio (+1), curto (0). Classificação: 0-4 pontos: “Conservador”; 5-8: “Moderado”; 9-12: “Arrojado”. Adicionalmente, se idade $>$ 60 E tolerância $< 15\%$, forçar “Conservador” independente da pontuação. Desenvolva este algoritmo de classificação.

Análise: o pseudocódigo poderia ser:

```
ALGORITMO AvaliacaoPerfilInvestidorConciso

ENTRADA:
    idade, rendaMensal, patrimonioLiquido, experienciaInvestimentos, toleranciaPerda: num
    objetivoPrazo: texto

SAÍDA:
    pontuacaoTotal: inteiro
    perfilInvestidor: texto

INÍCIO
    LER idade, rendaMensal, patrimonioLiquido, experienciaInvestimentos, toleranciaPerda,

    pontuacaoTotal ← 0
    SE (idade < 30) ENTÃO pontuacaoTotal ← pontuacaoTotal + 2
    SENÃO SE (idade <= 50) ENTÃO pontuacaoTotal ← pontuacaoTotal + 1

    SE (rendaMensal > 10000) ENTÃO pontuacaoTotal ← pontuacaoTotal + 2
    SENÃO SE (rendaMensal >= 5000) ENTÃO pontuacaoTotal ← pontuacaoTotal + 1

    SE (patrimonioLiquido > 100000) ENTÃO pontuacaoTotal ← pontuacaoTotal + 2
    SENÃO SE (patrimonioLiquido >= 50000) ENTÃO pontuacaoTotal ← pontuacaoTotal + 1

    SE (experienciaInvestimentos > 5) ENTÃO pontuacaoTotal ← pontuacaoTotal + 2
    SENÃO SE (experienciaInvestimentos >= 2) ENTÃO pontuacaoTotal ← pontuacaoTotal + 1

    SE (toleranciaPerda > 20) ENTÃO pontuacaoTotal ← pontuacaoTotal + 2
    SENÃO SE (toleranciaPerda >= 10) ENTÃO pontuacaoTotal ← pontuacaoTotal + 1

    SE (objetivoPrazo == "longo") ENTÃO pontuacaoTotal ← pontuacaoTotal + 2
    SENÃO SE (objetivoPrazo == "medio") ENTÃO pontuacaoTotal ← pontuacaoTotal + 1

    SE (pontuacaoTotal <= 4) ENTÃO perfilInvestidor ← "Conservador"
    SENÃO SE (pontuacaoTotal <= 8) ENTÃO perfilInvestidor ← "Moderado"
    SENÃO perfilInvestidor ← "Arrojado"

    SE (idade > 60) E (toleranciaPerda < 15) ENTÃO perfilInvestidor ← "Conservador"

    RETORNAR pontuacaoTotal, perfilInvestidor
FIM
```

O pseudocódigo pode ser representado com o Listing 6.6.

O código Listing 6.6 pode ser executado em qualquer ambiente de desenvolvimento C++ online, e está disponível no OnlineGDB.

O professor pode sugerir os seguintes casos para validação do algoritmo: jovem investidor arrojado (25 anos, R$ 12000 renda, R$ 150000 patrimônio, 3 anos experiência, 25% tolerância, prazo longo), investidor de perfil moderado (40 anos, R$ 7000 renda, R$ 70000 patrimônio, 4 anos experiência, 15% tolerância, prazo médio), e o caso especial de idoso forçado a conservador (65 anos, R$ 15000 renda, R$ 200000 patrimônio, 8 anos experiência, 12% tolerância, prazo longo).

M. Sistema de Controle de Qualidade Industrial: na fábrica "Precisão Total", o engenheiro Carlos implementa controle de qualidade para peças metálicas. O sistema mede `diametro_mm`, `peso_gramas`, `dureza_hrc`, `temperatura_teste` e `lote_producao`. Especificações: diâmetro 50mm ±0.5mm; peso 200g ±10g; dureza 45-55 HRC; temperatura teste entre 20-25°C. Classificação de defeitos: **Classe A** (aprovado): todas especificações atendidas; **Classe B** (aprovado com ressalvas): máximo 1 parâmetro fora da especificação E desvio não superior a 10% do limite; **Classe C** (retrabalho): 2 parâmetros fora OU 1 parâmetro com desvio $> 10\%$ mas 20%; **Classe D** (refugo): 3+ parâmetros fora OU qualquer desvio $> 20\%$. Adicionalmente, lotes de produção pares têm tolerância aumentada em 5% para todos os parâmetros. Crie o fluxograma para este sistema de controle de qualidade.

Análise: o pseudocódigo:

```
ALGORITMO ControleQualidadeIndustrialConciso

ENTRADA:
    diametroMm, pesoGramas, durezaHrc, temperaturaTeste: real
    loteProducao: inteiro

SAÍDA:
    classeQualidade: texto

INÍCIO
    LER diametroMm, pesoGramas, durezaHrc, temperaturaTeste, loteProducao

    toleranciaExtra ← SE (loteProducao % 2 == 0) ENTÃO 1.05 SENÃO 1.0

    diamMin ← 49.5 * toleranciaExtra
    diamMax ← 50.5 * toleranciaExtra
    pesoMin ← 190 * toleranciaExtra
    pesoMax ← 210 * toleranciaExtra
    durezaMin ← 45 * toleranciaExtra
    durezaMax ← 55 * toleranciaExtra
    tempMin ← 20 * toleranciaExtra
    tempMax ← 25 * toleranciaExtra

    parametrosForaEspec ← 0
    maxDesvio ← 0

    SE (diametroMm < diamMin) OU (diametroMm > diamMax) ENTÃO
        parametrosForaEspec ← parametrosForaEspec + 1
        desvio ← SE (diametroMm < diamMin) ENTÃO (diamMin - diametroMm) / 50 * 100 SENÃO
        SE (desvio > maxDesvio) ENTÃO maxDesvio ← desvio
```

```
    SE (pesoGramas < pesoMin) OU (pesoGramas > pesoMax) ENTÃO
        parametrosForaEspec ← parametrosForaEspec + 1
        desvio ← SE (pesoGramas < pesoMin) ENTÃO (pesoMin - pesoGramas) / 200 * 100 SENÃO
        SE (desvio > maxDesvio) ENTÃO maxDesvio ← desvio

    SE (durezaHrc < durezaMin) OU (durezaHrc > durezaMax) ENTÃO
        parametrosForaEspec ← parametrosForaEspec + 1
        desvio ← SE (durezaHrc < durezaMin) ENTÃO (durezaMin - durezaHrc) / 50 * 100 SENÃ
        SE (desvio > maxDesvio) ENTÃO maxDesvio ← desvio

    SE (temperaturaTeste < tempMin) OU (temperaturaTeste > tempMax) ENTÃO
        parametrosForaEspec ← parametrosForaEspec + 1
        desvio ← SE (temperaturaTeste < tempMin) ENTÃO (tempMin - temperaturaTeste) / 22.
        SE (desvio > maxDesvio) ENTÃO maxDesvio ← desvio

    SE (parametrosForaEspec == 0) ENTÃO classeQualidade ← "Classe A"
    SENÃO SE (parametrosForaEspec == 1) E (maxDesvio <= 10) ENTÃO classeQualidade ← "Clas
    SENÃO SE (parametrosForaEspec == 2) OU (parametrosForaEspec == 1 E maxDesvio <= 20) E
    SENÃO classeQualidade ← "Classe D"

    RETORNAR classeQualidade
FIM
```

O pseudocódigo pode ser representado com o Listing 6.7.

O código Listing 6.7 pode ser executado em qualquer ambiente de desenvolvimento C++ online, e está disponível no OnlineGDB.

O professor pode sugerir os seguintes casos para validação do algoritmo: peça perfeita dentro de todas as especificações (50.0mm diâmetro, 200g peso, 50 HRC dureza, 22°C temperatura, lote 101 ímpar), lote par com pequeno desvio no diâmetro (51.0mm diâmetro, 200g peso, 50 HRC dureza, 22°C temperatura, lote 102 par), e peça defeituosa com múltiplos parâmetros fora de especificação (48.0mm diâmetro, 170g peso, 40 HRC dureza, 30°C temperatura, lote 103 ímpar). O professor pode pedir que os alunos verifiquem se o pseudocódigo funciona corretamente para esses casos usando as Tabelas de Rastreio.

N. Sistema de Avaliação para Promoção Corporativa: na empresa "Crescer Juntos", a gestora de RH Dra. Ana Carolina precisa automatizar o processo de avaliação para promoções. O sistema deve analisar múltiplos critérios dos funcionários: `tempo_empresa` (anos), `avaliacao_desempenho` (1-10), `nivel_formacao` ("medio", "superior", "pos"), `certificacoes_profissionais` (quantidade), `idade` e `salario_atual`. As regras de classificação são: **Promoção Imediata**: tempo 3 anos E avaliação 8 E (formação superior OU pós) E certificações 2; **Promoção Condicional**: tempo 2 anos E avaliação 7 E pelo menos um dos critérios extras (formação superior/pós OU certificações 1); **Desenvolvimento Necessário**: tempo 1 ano E avaliação 6; **Não Elegível**: demais casos. Adicionalmente, se idade $>$ 55 anos E salário $>$ R$ 15000, aplicar regra especial "Plano Sucessão" independente dos outros critérios. Desenvolva este algoritmo de classificação.

Análise: o pseudocódigo:

```
ALGORITMO AvaliacaoPromocaoConciso

ENTRADA:
    tempoEmpresa, avaliacaoDesempenho, certificacoesProfissionais, idade, salarioAtual: n
    nivelFormacao: texto
```

```
SAÍDA:
    classificacao: texto

INÍCIO
    LER tempoEmpresa, avaliacaoDesempenho, nivelFormacao, certificacoesProfissionais, ida

    SE (idade > 55) E (salarioAtual > 15000) ENTÃO
        classificacao ← "Plano Sucessão"
    SENÃO
        formacaoAvancada ← (nivelFormacao == "superior") OU (nivelFormacao == "pos")

        SE (tempoEmpresa >= 3) E (avaliacaoDesempenho >= 8) E formacaoAvancada E (certifi
            classificacao ← "Promoção Imediata"
        SENÃO SE (tempoEmpresa >= 2) E (avaliacaoDesempenho >= 7) E (formacaoAvancada OU
            classificacao ← "Promoção Condicional"
        SENÃO SE (tempoEmpresa >= 1) E (avaliacaoDesempenho >= 6) ENTÃO
            classificacao ← "Desenvolvimento Necessário"
        SENÃO
            classificacao ← "Não Elegível"

    RETORNAR classificacao
FIM
```

O pseudocódigo pode ser representado com o Listing 6.8.

O código Listing 6.8 pode ser executado em qualquer ambiente de desenvolvimento C++ online, e está disponível no OnlineGDB.

Casos de teste: funcionário para promoção imediata (4 anos empresa, avaliação 9, formação superior, 3 certificações, 30 anos, R$ 8000), promoção condicional (2.5 anos empresa, avaliação 7.5, formação média, 1 certificação, 28 anos, R$ 5000), e caso especial do plano sucessão (10 anos empresa, avaliação 8, formação pós, 5 certificações, 58 anos, R$ 18000). O professor pode pedir que os alunos verifiquem se o pseudocódigo funciona corretamente para esses casos usando as Tabelas de Rastreio.

6.1.4 Atividade *Unplugged*: Estruturas de Repetição

Nas primeiras três seções exploramos à Metodologia **DAAD** e a construção de algoritmos com estruturas condicionais. Nesta seção vamos introduzir os conceitos relacionados a laços de repetição completando as três estruturas básicas da programação imperativa: atribuição, decisão e repetição. Abaixo estão sete problemas que envolvem laços de repetição e que podem ser resolvidos com fluxogramas e pseudocódigo.

O primeiro exercício deve ser completamente resolvido pelo professor, no quadro.

A1. **Geração de Relatórios Financeiros**: a contabilidade da empresa "Números Certos" precisa de um sistema que gere rapidamente relatórios de multiplicação para auditorias internas. O Sr. Costa, o contador, frequentemente precisa visualizar a "tabuada" de um determinado `número` (multiplicado de 1 a 10). Projete um algoritmo que, dado um `número` inteiro, imprima sua tabuada completa.

Análise: o fluxograma pode ser representado pelo fluxo da Figure 6.8.

Figure 6.8: Fluxograma para representar o problema Geração de Relatórios Financeiros.

O pseudocódigo pode ser representado como:

```
ALGORITMO GeracaoTabuadaConciso

ENTRADA:
    numero: inteiro

SAÍDA:
    resultado: inteiro

INÍCIO
    LER numero
    contador ← 1

ETIQUETA LOOP:
    resultado ← numero * contador
    ESCREVER numero, " x ", contador, " = ", resultado
    contador ← contador + 1
    SE contador <= 10 ENTÃO IR PARA LOOP

FIM
```

O pseudocódigo pode ser representado com o Listing 6.9.

O código Listing 6.9 pode ser executado em qualquer ambiente de desenvolvimento C++ online, e está disponível no OnlineGDB.

Para testar adequadamente o algoritmo de geração de tabuada, o professor deverá executar pelo menos três casos de teste distintos com suas respectivas tabelas de rastreio.

1. **Caso 1 (Normal)**: numero = 5, este caso deve produzir a tabuada completa do 5 ($5 \times 1 = 5, 5 \times 2 = 10, ..., 5 \times 10 = 50$) e verificar se o laço de repetição executa exatamente 10 iterações.

Table 6.43: Tabela de rastreio para o Caso 1 demonstra que o algoritmo lida corretamente com números inteiros, produzindo uma tabuada de 5.{#tbl-rastreio-exa11}

Passo	numero	contador	resultado	condição (contador 10)	ação executada
1	5	1	5	verdadeiro	imprimir “5 x 1 = 5”, incrementar contador
2	5	2	10	verdadeiro	imprimir “5 x 2 = 10”, incrementar contador
3	5	3	15	verdadeiro	imprimir “5 x 3 = 15”, incrementar contador
4	5	4	20	verdadeiro	imprimir “5 x 4 = 20”, incrementar contador

Passo	numero	contador	resultado	condição (contador 10)	ação executada
5	5	5	25	verdadeiro	imprimir “5 x 5 = 25”, incrementar contador
6	5	6	30	verdadeiro	imprimir “5 x 6 = 30”, incrementar contador
7	5	7	35	verdadeiro	imprimir “5 x 7 = 35”, incrementar contador
8	5	8	40	verdadeiro	imprimir “5 x 8 = 40”, incrementar contador
9	5	9	45	verdadeiro	imprimir “5 x 9 = 45”, incrementar contador
10	5	10	50	verdadeiro	imprimir “5 x 10 = 50”, incrementar contador
11	5	11	-	falso	encerrar algoritmo

2. **Caso 2 (Limite)**: numero = 0, embora tecnicamente válido, produzirá uma tabuada de zeros, útil para verificar se o algoritmo funciona com valores extremos.

Table 6.44: Tabela de rastreio para o Caso 2 demonstra que o algoritmo lida corretamente com o zero, produzindo uma tabuada de zeros.{#tbl-rastreio-exa12}

Passo	numero	contador	resultado	condição (contador 10)	ação executada
1	0	1	0	verdadeiro	imprimir “0 x 1 = 0”, incrementar contador
2	0	2	0	verdadeiro	imprimir “0 x 2 = 0”, incrementar contador
3	0	3	0	verdadeiro	imprimir “0 x 3 = 0”, incrementar contador

Passo	numero	contador	resultado	condição (contador 10)	ação executada
4	0	4	0	verdadeiro	imprimir “0 x 4 = 0”, incrementar contador
5	0	5	0	verdadeiro	imprimir “0 x 5 = 0”, incrementar contador
6	0	6	0	verdadeiro	imprimir “0 x 6 = 0”, incrementar contador
7	0	7	0	verdadeiro	imprimir “0 x 7 = 0”, incrementar contador
8	0	8	0	verdadeiro	imprimir “0 x 8 = 0”, incrementar contador
9	0	9	0	verdadeiro	imprimir “0 x 9 = 0”, incrementar contador
10	0	10	0	verdadeiro	imprimir “0 x 10 = 0”, incrementar contador
11	0	11	-	falso	encerrar algoritmo

Caso 3 (Problemático): numero = -3, este caso revela uma limitação do algoritmo, pois embora execute corretamente do ponto de vista técnico (-3x1=-3, -3x2=-6, etc.), gera resultados negativos que podem não ser apropriados para “relatórios financeiros” em um contexto empresarial real.

Table 6.45: Tabela de rastreio para o Caso 3 demonstra que o algoritmo funciona matematicamente, mas gera valores negativos inadequados para um contexto de relatórios financeiros empresariais.{#tbl-rastreio-exa13}

Passo	numero	contador	resultado	condição (contador 10)	ação executada
1	-3	1	-3	verdadeiro	imprimir “-3 x 1 = -3”, incrementar contador
2	-3	2	-6	verdadeiro	imprimir “-3 x 2 = -6”, incrementar contador

Passo	numero	contador	resultado	condição (contador 10)	ação executada
3	-3	3	-9	verdadeiro	imprimir “-3 x 3 = -9”, incrementar contador
4	-3	4	-12	verdadeiro	imprimir “-3 x 4 = -12”, incrementar contador
5	-3	5	-15	verdadeiro	imprimir “-3 x 5 = -15”, incrementar contador
6	-3	6	-18	verdadeiro	imprimir “-3 x 6 = -18”, incrementar contador
7	-3	7	-21	verdadeiro	imprimir “-3 x 7 = -21”, incrementar contador
8	-3	8	-24	verdadeiro	imprimir “-3 x 8 = -24”, incrementar contador
9	-3	9	-27	verdadeiro	imprimir “-3 x 9 = -27”, incrementar contador
10	-3	10	-30	verdadeiro	imprimir “-3 x 10 = -30”, incrementar contador
11	-3	11	-	falso	encerrar algoritmo

Seguindo a técnica da Sequência de Fibonacci (1, 1, 2), o professor deve solicitar que os quatro próximos exercícios sejam resolvidos com fluxogramas, os fluxogramas sejam transformados em pseudocódigo, os pseudocódigos sejam transformados em código C++ e, finalmente, os códigos C++ sejam validados com tabelas de rastreio. Além disso, o professor deve dimensionar o tempo de cada exercício lembrando que o tempo para execução dos dois últimos deve ser aproximadamente igual ao tempo disponibilizado para o primeiro exercício.

B1. **Sequência de Lançamento de Foguetes**: na agência espacial “Horizonte Infinito”, o protocolo de lançamento de foguetes exige uma contagem regressiva precisa. O sistema deve exibir os números de 10 até 0, e no momento zero, a mensagem “Lançar!” deve ser exibida. O engenheiro Rui Gambia precisa de um algoritmo que automatize esta sequência de contagem regressiva para os lançamentos. Crie um fluxograma para este algoritmo.

Análise: neste caso, o fluxograma pode ser representado pelo fluxo da Figure 6.9.

Fluxograma para representar o problema Sequência de Lançamento de Foguetes.

Figure 6.9

O pseudocódigo pode ser representado como:

```
ALGORITMO SequenciaLancamentoFoguetesConciso

SAÍDA:
    contador: inteiro

INÍCIO
    contador ← 10

ETIQUETA LOOP:
    ESCREVER contador
    SE contador = 0 ENTÃO
        ESCREVER "Lançar!"
        IR PARA FIM
    SENÃO
        contador ← contador - 1
        IR PARA LOOP

ETIQUETA FIM:
FIM
```

O pseudocódigo pode ser representado com o Listing 6.10.

O código Listing 6.10 pode ser executado em qualquer ambiente de desenvolvimento C++ online, e está disponível no OnlineGDB.

C1. **Análise de Desempenho de Vendas**: o gerente de vendas da "Vendas Top" precisa calcular a `media_de_vendas` por vendedor. Ele insere os valores de `venda` um por um. No entanto, apenas vendas entre 0 e 1000 Reais são consideradas válidas. O processo de inserção de vendas termina quando um valor negativo é digitado (este valor negativo não deve ser incluído na média). Ajude o gerente criando um algoritmo que calcule a média das vendas válidas.

Análise: o fluxograma pode ser representado por:

Fluxograma para representar o problema Análise de Desempenho de Vendas.

Figure 6.10

O pseudocódigo pode ser representado como:

```
ALGORITMO AnaliseDesempenhoVendasConciso

ENTRADA:
    venda: real

SAÍDA:
    media: real

INÍCIO
    soma ← 0
```

```
    contador ← 0

ETIQUETA LOOP:
    LER venda
    SE venda < 0 ENTÃO IR PARA CALCULAR_MEDIA
    SE (venda >= 0) E (venda <= 1000) ENTÃO
        soma ← soma + venda
        contador ← contador + 1
    IR PARA LOOP

ETIQUETA CALCULAR_MEDIA:
    SE contador > 0 ENTÃO
        media ← soma / contador
        ESCREVER "Média:", media
    SENÃO
        ESCREVER "Nenhuma venda válida"
FIM
```

O pseudocódigo pode ser representado com o Listing 6.11.

O código Listing 6.11 pode ser executado em qualquer ambiente de desenvolvimento C++ online, e está disponível no OnlineGDB.

D1. **Jogo Interativo de Lógica**: a empresa de jogos "Mente Brilhante" está desenvolvendo um jogo de adivinhação. O computador "escolhe" um `numero_secreto` (por exemplo, 42) entre 1 e 100. O jogador tem 5 `tentativas` para adivinhar. A cada tentativa, o jogo deve informar se o `palpite` do jogador foi "Muito alto!", "Muito baixo!" ou "Correto! Você adivinhou o número!". Se o jogador acertar, o jogo termina com uma mensagem de vitória. Se as tentativas acabarem, o jogo termina com uma mensagem de derrota, revelando o `numero_secreto`. Considerando que o número secreto é fixo e você não vai mudá-lo nesta tarefa, crie um fluxograma para um algoritmo que represente o processo de adivinhação.

Análise: o pseudocódigo para este exercício pode ser representado como:

```
ALGORITMO JogoAdivinhacao

CONSTANTES:
    numero_secreto ← 42
    max_tentativas ← 5

ENTRADA:
    palpite: inteiro

SAÍDA:
    mensagem: texto

INÍCIO
    tentativas_feitas ← 0

ETIQUETA LOOP_TENTATIVAS:
    SE tentativas_feitas >= max_tentativas ENTÃO
        ESCREVER "Você perdeu! O número secreto era ", numero_secreto
        IR PARA FIM_JOGO
    FIM SE
```

```
    ESCREVER "Tentativa ", tentativas_feitas + 1, ". Digite seu palpite (1-100):"
    LER palpite
    tentativas_feitas ← tentativas_feitas + 1

    SE palpite = numero_secreto ENTÃO
        ESCREVER "Correto! Você adivinhou o número!"
        IR PARA FIM_JOGO
    SENÃO SE palpite > numero_secreto ENTÃO
        ESCREVER "Muito alto!"
    SENÃO
        ESCREVER "Muito baixo!"
    FIM SE

    IR PARA LOOP_TENTATIVAS

ETIQUETA FIM_JOGO:
FIM
```

E1. **Cálculo de Probabilidades**: no departamento de estatística da Universidade, a aluna Clara está estudando probabilidades e frequentemente precisa calcular o fatorial de números inteiros não negativos. O fatorial de um número n (representado por n!) é o produto de todos os inteiros positivos menores ou iguais a n. Por definição, $0! = 1$. Ajude a Clara a desenvolver um algoritmo que, dado um `numero`, calcule e exiba o seu fatorial. (Ex: $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$).

Análise: o pseudocódigo para este exercício pode ser representado como:

```
ALGORITMO CalculoFatorialConciso

ENTRADA:
    numero: inteiro

SAÍDA:
    fatorial: inteiro

INÍCIO
    LER numero
    fatorial ← 1
    contador ← numero

    SE numero = 0 ENTÃO IR PARA RESULTADO

ETIQUETA LOOP:
    fatorial ← fatorial * contador
    contador ← contador - 1
    SE contador > 0 ENTÃO IR PARA LOOP

ETIQUETA RESULTADO:
    ESCREVER numero, "! = ", fatorial
FIM
```

Que pode ser implementado em C++ 23 por:

```cpp
#include <iostream>

int main() {
    // ENTRADA
    int numero;

    std::cout << "Digite um número para calcular o fatorial: ";
    std::cin >> numero;

    // PROCESSAMENTO
    long long fatorial = 1;
    int contador = numero;

    if (numero == 0) goto RESULTADO;

LOOP:
    fatorial = fatorial * contador;
    contador = contador - 1;
    if (contador > 0) goto LOOP;

RESULTADO:
    std::cout << numero << "! = " << fatorial << std::endl;

    return 0;
}
```

F1. Sistema de Análise de Desempenho Acadêmico por Turmas: na escola "Excelência Educacional", a coordenadora pedagógica Profa. Marina precisa analisar o desempenho de múltiplas turmas simultaneamente. O sistema deve processar `numero_turmas` turmas, e para cada turma, deve ler `numero_alunos` da turma e as `notas` de cada aluno (0-100). Para cada turma, calcular e exibir: média da turma, quantidade de alunos aprovados (nota 60), maior nota e menor nota. Adicionalmente, o sistema deve identificar qual turma teve a maior média geral e quantos alunos no total ficaram abaixo da média de sua respectiva turma. Notas inválidas (fora do range 0-100) devem ser rejeitadas e solicitadas novamente. Se uma turma tiver menos de 3 alunos válidos, ela deve ser marcada como "Turma Inválida" e não incluída nas estatísticas gerais. No final, exibir um relatório consolidado com: média geral de todas as turmas válidas, total de alunos aprovados em toda a escola, e o número da turma com melhor desempenho.

Análise: o pseudocódigo para este exercício pode ser representado como:

```
ALGORITMO AnaliseDesempenhoAcademicoConciso

ENTRADA:
    numeroTurmas, numeroAlunos, nota: inteiro

SAÍDA:
    mediaTurma, alunosAprovados, maiorNota, menorNota: real
    melhorTurma, totalAbaixoMedia: inteiro

INÍCIO
    LER numeroTurmas
```

```
    turmaAtual ← 1
    melhorMedia ← 0
    melhorTurma ← 0
    totalTurmasValidas ← 0
    somaMediasGeral ← 0
    totalAlunosAprovadosGeral ← 0
    totalAbaixoMedia ← 0

ETIQUETA LOOP_TURMAS:
    ESCREVER "=== TURMA ", turmaAtual, " ==="
    LER numeroAlunos
    alunoAtual ← 1
    somaTurma ← 0
    alunosValidos ← 0
    alunosAprovados ← 0
    maiorNota ← 0
    menorNota ← 100

ETIQUETA LOOP_ALUNOS:
    ESCREVER "Nota do aluno ", alunoAtual, ": "

ETIQUETA LER_NOTA:
    LER nota
    SE (nota < 0) OU (nota > 100) ENTÃO
        ESCREVER "Nota inválida! Digite novamente: "
        IR PARA LER_NOTA

    somaTurma ← somaTurma + nota
    alunosValidos ← alunosValidos + 1
    SE nota >= 60 ENTÃO alunosAprovados ← alunosAprovados + 1
    SE nota > maiorNota ENTÃO maiorNota ← nota
    SE nota < menorNota ENTÃO menorNota ← nota

    alunoAtual ← alunoAtual + 1
    SE alunoAtual <= numeroAlunos ENTÃO IR PARA LOOP_ALUNOS

    SE alunosValidos < 3 ENTÃO
        ESCREVER "Turma Inválida (menos de 3 alunos válidos)"
        IR PARA PROXIMA_TURMA

    mediaTurma ← somaTurma / alunosValidos
    ESCREVER "Média da turma: ", mediaTurma
    ESCREVER "Alunos aprovados: ", alunosAprovados
    ESCREVER "Maior nota: ", maiorNota
    ESCREVER "Menor nota: ", menorNota

    totalTurmasValidas ← totalTurmasValidas + 1
    somaMediasGeral ← somaMediasGeral + mediaTurma
    totalAlunosAprovadosGeral ← totalAlunosAprovadosGeral + alunosAprovados

    SE mediaTurma > melhorMedia ENTÃO
```

```
        melhorMedia ← mediaTurma
        melhorTurma ← turmaAtual

    alunoContador ← 1
ETIQUETA CONTAR_ABAIXO_MEDIA:
    SE alunoContador <= alunosValidos ENTÃO
        SE notaAluno[alunoContador] < mediaTurma ENTÃO
            totalAbaixoMedia ← totalAbaixoMedia + 1
        alunoContador ← alunoContador + 1
        IR PARA CONTAR_ABAIXO_MEDIA

ETIQUETA PROXIMA_TURMA:
    turmaAtual ← turmaAtual + 1
    SE turmaAtual <= numeroTurmas ENTÃO IR PARA LOOP_TURMAS

    ESCREVER "=== RELATÓRIO CONSOLIDADO ==="
    SE totalTurmasValidas > 0 ENTÃO
        mediaGeral ← somaMediasGeral / totalTurmasValidas
        ESCREVER "Média geral: ", mediaGeral
        ESCREVER "Total de alunos aprovados: ", totalAlunosAprovadosGeral
        ESCREVER "Melhor turma: ", melhorTurma
        ESCREVER "Alunos abaixo da média: ", totalAbaixoMedia
    SENÃO
        ESCREVER "Nenhuma turma válida encontrada"

FIM
```

Que pode ser representado em C++23 por:

```cpp
#include <iostream>

int main() {
    // ENTRADA
    int numeroTurmas, numeroAlunos, nota;

    std::cout << "Número de turmas: ";
    std::cin >> numeroTurmas;

    // PROCESSAMENTO
    int turmaAtual = 1;
    double melhorMedia = 0;
    int melhorTurma = 0;
    int totalTurmasValidas = 0;
    double somaMediasGeral = 0;
    int totalAlunosAprovadosGeral = 0;
    int totalAbaixoMedia = 0;

    int notasTurma[1000]; // Array para armazenar notas da turma atual

LOOP_TURMAS:
    std::cout << "n=== TURMA " << turmaAtual << " ===" << std::endl;
    std::cout << "Número de alunos: ";
```

```
    std::cin >> numeroAlunos;

    int alunoAtual = 1;
    double somaTurma = 0;
    int alunosValidos = 0;
    int alunosAprovados = 0;
    double maiorNota = 0;
    double menorNota = 100;

LOOP_ALUNOS:
    std::cout << "Nota do aluno " << alunoAtual << ": ";

LER_NOTA:
    std::cin >> nota;
    if ((nota < 0) || (nota > 100)) {
        std::cout << "Nota inválida! Digite novamente: ";
        goto LER_NOTA;
    }

    notasTurma[alunosValidos] = nota;
    somaTurma = somaTurma + nota;
    alunosValidos = alunosValidos + 1;
    if (nota >= 60) alunosAprovados = alunosAprovados + 1;
    if (nota > maiorNota) maiorNota = nota;
    if (nota < menorNota) menorNota = nota;

    alunoAtual = alunoAtual + 1;
    if (alunoAtual <= numeroAlunos) goto LOOP_ALUNOS;

    if (alunosValidos < 3) {
        std::cout << "Turma Inválida (menos de 3 alunos válidos)" << std::endl;
        goto PROXIMA_TURMA;
    }

    double mediaTurma = somaTurma / alunosValidos;
    std::cout << "Média da turma: " << mediaTurma << std::endl;
    std::cout << "Alunos aprovados: " << alunosAprovados << std::endl;
    std::cout << "Maior nota: " << maiorNota << std::endl;
    std::cout << "Menor nota: " << menorNota << std::endl;

    totalTurmasValidas = totalTurmasValidas + 1;
    somaMediasGeral = somaMediasGeral + mediaTurma;
    totalAlunosAprovadosGeral = totalAlunosAprovadosGeral + alunosAprovados;

    if (mediaTurma > melhorMedia) {
        melhorMedia = mediaTurma;
        melhorTurma = turmaAtual;
    }

    int alunoContador = 0;
CONTAR_ABAIXO_MEDIA:
```

```
    if (alunoContador < alunosValidos) {
        if (notasTurma[alunoContador] < mediaTurma) {
            totalAbaixoMedia = totalAbaixoMedia + 1;
        }
        alunoContador = alunoContador + 1;
        goto CONTAR_ABAIXO_MEDIA;
    }

PROXIMA_TURMA:
    turmaAtual = turmaAtual + 1;
    if (turmaAtual <= numeroTurmas) goto LOOP_TURMAS;

    std::cout << "n=== RELATÓRIO CONSOLIDADO ===" << std::endl;
    if (totalTurmasValidas > 0) {
        double mediaGeral = somaMediasGeral / totalTurmasValidas;
        std::cout << "Média geral: " << mediaGeral << std::endl;
        std::cout << "Total de alunos aprovados: " << totalAlunosAprovadosGeral << std::e
        std::cout << "Melhor turma: " << melhorTurma << std::endl;
        std::cout << "Alunos abaixo da média: " << totalAbaixoMedia << std::endl;
    } else {
        std::cout << "Nenhuma turma válida encontrada" << std::endl;
    }

    return 0;
}
```

G1. **Simulação de Sistema de Caixa Eletrônico Bancário**: no banco "Futuro Digital", o analista Pedro precisa simular operações de caixa eletrônico para teste de capacidade. O sistema deve processar uma sequência de `operacoes` até que seja digitado 0 (encerrar). Os tipos de operação são: 1-Saque, 2-Depósito, 3-Consulta. Para cada operação, ler o `valor` correspondente. O sistema inicia com saldo R$ 1000,00. Regras: saques só são permitidos se houver saldo suficiente e se o valor for múltiplo de R$ 10,00 (limitação física das cédulas); depósitos devem ser valores positivos; a cada 5 operações bem-sucedidas consecutivas, aplicar uma taxa de manutenção de R$ 2,00; se o saldo ficar negativo ou se houver 3 operações inválidas consecutivas, bloquear o cartão e encerrar. Durante a simulação, manter contadores de: total de saques realizados, valor total depositado, número de consultas, operações inválidas consecutivas e operações válidas consecutivas. No encerramento, exibir relatório completo: saldo final, estatísticas de operações, total de taxas cobradas, e motivo do encerramento (usuário, bloqueio por saldo, ou bloqueio por operações inválidas).

Análise: o pseudocódigo para este exercício pode ser representado como:

```
ALGORITMO SimulacaoCaixaEletronicoConciso

ENTRADA:
    operacao, valor: real

SAÍDA:
    saldo, totalSaques, valorTotalDepositado, numeroConsultas: real
    operacoesInvalidasConsecutivas, operacoesValidasConsecutivas: inteiro

INÍCIO
```

```
    saldo ← 1000.00
    totalSaques ← 0
    valorTotalDepositado ← 0
    numeroConsultas ← 0
    operacoesInvalidasConsecutivas ← 0
    operacoesValidasConsecutivas ← 0
    totalTaxas ← 0
    motivoEncerramento ← "usuário"

ETIQUETA LOOP_OPERACOES:
    ESCREVER "Operação (1-Saque, 2-Depósito, 3-Consulta, 0-Encerrar): "
    LER operacao

    SE operacao = 0 ENTÃO IR PARA RELATORIO_FINAL

    SE operacao = 3 ENTÃO
        ESCREVER "Saldo atual: R$ ", saldo
        numeroConsultas ← numeroConsultas + 1
        operacoesValidasConsecutivas ← operacoesValidasConsecutivas + 1
        operacoesInvalidasConsecutivas ← 0
        IR PARA VERIFICAR_TAXA

    ESCREVER "Valor: R$ "
    LER valor

    SE operacao = 1 ENTÃO
        SE (valor % 10   0) OU (valor > saldo) OU (valor <= 0) ENTÃO
            ESCREVER "Operação inválida!"
            operacoesInvalidasConsecutivas ← operacoesInvalidasConsecutivas + 1
            operacoesValidasConsecutivas ← 0
        SENÃO
            saldo ← saldo - valor
            totalSaques ← totalSaques + 1
            ESCREVER "Saque realizado. Saldo: R$ ", saldo
            operacoesValidasConsecutivas ← operacoesValidasConsecutivas + 1
            operacoesInvalidasConsecutivas ← 0
    SENÃO SE operacao = 2 ENTÃO
        SE valor <= 0 ENTÃO
            ESCREVER "Operação inválida!"
            operacoesInvalidasConsecutivas ← operacoesInvalidasConsecutivas + 1
            operacoesValidasConsecutivas ← 0
        SENÃO
            saldo ← saldo + valor
            valorTotalDepositado ← valorTotalDepositado + valor
            ESCREVER "Depósito realizado. Saldo: R$ ", saldo
            operacoesValidasConsecutivas ← operacoesValidasConsecutivas + 1
            operacoesInvalidasConsecutivas ← 0
    SENÃO
        ESCREVER "Operação inválida!"
        operacoesInvalidasConsecutivas ← operacoesInvalidasConsecutivas + 1
        operacoesValidasConsecutivas ← 0
```

```
ETIQUETA VERIFICAR_TAXA:
    SE operacoesValidasConsecutivas = 5 ENTÃO
        saldo ← saldo - 2.00
        totalTaxas ← totalTaxas + 2.00
        operacoesValidasConsecutivas ← 0
        ESCREVER "Taxa de manutenção aplicada: R$ 2,00"

ETIQUETA VERIFICAR_BLOQUEIOS:
    SE saldo < 0 ENTÃO
        motivoEncerramento ← "bloqueio por saldo negativo"
        IR PARA RELATORIO_FINAL

    SE operacoesInvalidasConsecutivas >= 3 ENTÃO
        motivoEncerramento ← "bloqueio por operações inválidas"
        IR PARA RELATORIO_FINAL

    IR PARA LOOP_OPERACOES

ETIQUETA RELATORIO_FINAL:
    ESCREVER "=== RELATÓRIO FINAL ==="
    ESCREVER "Saldo final: R$ ", saldo
    ESCREVER "Total de saques: ", totalSaques
    ESCREVER "Valor total depositado: R$ ", valorTotalDepositado
    ESCREVER "Número de consultas: ", numeroConsultas
    ESCREVER "Total de taxas: R$ ", totalTaxas
    ESCREVER "Motivo do encerramento: ", motivoEncerramento

FIM
```

Que pode ser representado em C++23 por:

```cpp
#include <iostream>

int main() {
    // ENTRADA
    int operacao;
    double valor;

    // PROCESSAMENTO
    double saldo = 1000.00;
    int totalSaques = 0;
    double valorTotalDepositado = 0;
    int numeroConsultas = 0;
    int operacoesInvalidasConsecutivas = 0;
    int operacoesValidasConsecutivas = 0;
    double totalTaxas = 0;
    std::string motivoEncerramento = "usuário";

LOOP_OPERACOES:
    std::cout << "nOperação (1-Saque, 2-Depósito, 3-Consulta, 0-Encerrar): ";
    std::cin >> operacao;
```

```
    if (operacao == 0) goto RELATORIO_FINAL;

    if (operacao == 3) {
        std::cout << "Saldo atual: R$ " << saldo << std::endl;
        numeroConsultas = numeroConsultas + 1;
        operacoesValidasConsecutivas = operacoesValidasConsecutivas + 1;
        operacoesInvalidasConsecutivas = 0;
        goto VERIFICAR_TAXA;
    }

    std::cout << "Valor: R$ ";
    std::cin >> valor;

    if (operacao == 1) {
        if ((static_cast<int>(valor) % 10 != 0) || (valor > saldo) || (valor <= 0)) {
            std::cout << "Operação inválida!" << std::endl;
            operacoesInvalidasConsecutivas = operacoesInvalidasConsecutivas + 1;
            operacoesValidasConsecutivas = 0;
        } else {
            saldo = saldo - valor;
            totalSaques = totalSaques + 1;
            std::cout << "Saque realizado. Saldo: R$ " << saldo << std::endl;
            operacoesValidasConsecutivas = operacoesValidasConsecutivas + 1;
            operacoesInvalidasConsecutivas = 0;
        }
    } else if (operacao == 2) {
        if (valor <= 0) {
            std::cout << "Operação inválida!" << std::endl;
            operacoesInvalidasConsecutivas = operacoesInvalidasConsecutivas + 1;
            operacoesValidasConsecutivas = 0;
        } else {
            saldo = saldo + valor;
            valorTotalDepositado = valorTotalDepositado + valor;
            std::cout << "Depósito realizado. Saldo: R$ " << saldo << std::endl;
            operacoesValidasConsecutivas = operacoesValidasConsecutivas + 1;
            operacoesInvalidasConsecutivas = 0;
        }
    } else {
        std::cout << "Operação inválida!" << std::endl;
        operacoesInvalidasConsecutivas = operacoesInvalidasConsecutivas + 1;
        operacoesValidasConsecutivas = 0;
    }

VERIFICAR_TAXA:
    if (operacoesValidasConsecutivas == 5) {
        saldo = saldo - 2.00;
        totalTaxas = totalTaxas + 2.00;
        operacoesValidasConsecutivas = 0;
        std::cout << "Taxa de manutenção aplicada: R$ 2,00" << std::endl;
    }
```

```
VERIFICAR_BLOQUEIOS:
    if (saldo < 0) {
        motivoEncerramento = "bloqueio por saldo negativo";
        goto RELATORIO_FINAL;
    }

    if (operacoesInvalidasConsecutivas >= 3) {
        motivoEncerramento = "bloqueio por operações inválidas";
        goto RELATORIO_FINAL;
    }

    goto LOOP_OPERACOES;

RELATORIO_FINAL:
    std::cout << "n=== RELATÓRIO FINAL ===" << std::endl;
    std::cout << "Saldo final: R$ " << saldo << std::endl;
    std::cout << "Total de saques: " << totalSaques << std::endl;
    std::cout << "Valor total depositado: R$ " << valorTotalDepositado << std::endl;
    std::cout << "Número de consultas: " << numeroConsultas << std::endl;
    std::cout << "Total de taxas: R$ " << totalTaxas << std::endl;
    std::cout << "Motivo do encerramento: " << motivoEncerramento << std::endl;

    return 0;
}
```

I1. O biólogo Dr. Miguel está modelando o crescimento de uma população de bactérias que segue uma progressão aritmética. Ele precisa de um algoritmo que, dado o `primeiro_termo` da população inicial e a `razao` de crescimento diário, consiga exibir os `N` primeiros dias de crescimento dessa população.

J1. Para aumentar a segurança dos dados dos usuários, a empresa "CiberSeguro" exige que, ao criar uma nova conta, o usuário digite a sua `nova_senha` e, em seguida, a confirme digitando-a novamente. O sistema só deve aceitar o registro quando a `confirmação_senha` for exatamente igual à `nova_senha`. Desenvolva um algoritmo que implemente esta validação, continuando a pedir a confirmação até que as senhas coincidam.

K1. A Professora Lúcia, da disciplina de "Matemática Discreta", está ensinando a sequência de Fibonacci, que aparece em diversos fenômenos naturais (espirais de conchas, arranjo de folhas). A sequência começa com 0 e 1, e cada termo subsequente é a soma dos dois anteriores (0, 1, 1, 2, 3, 5, 8...). Ela precisa de um algoritmo que, dado um número inteiro `N`, gere e exiba os `N` primeiros termos desta sequência.

L1. No departamento de análise de dados, a estatística Marta está processando grandes conjuntos de números. Para uma análise específica, ela precisa de um algoritmo que, dado um `numero_inteiro_positivo`, determine quantos dos seus dígitos são pares e quantos são ímpares. (Ex: o número 12345 tem 2 dígitos pares (2, 4) e 3 dígitos ímpares (1, 3, 5)).

M1. O Banco "Futuro Digital" está desenvolvendo um protótipo de caixa eletrônico. O sistema inicia com um `saldo_inicial` de 1000 euros. O usuário pode escolher entre "Depositar", "Sacar" ou "Sair" do sistema.

- **Depositar**: O usuário informa o `valor_deposito`, que é adicionado ao `saldo`.
- **Sacar**: O usuário informa o `valor_saque`. Se o `saldo` for suficiente, o valor é debitado. Caso contrário, uma mensagem de "Saldo insuficiente" é exibida.

O sistema deve continuar a apresentar o menu e a executar as operações até que o usuário escolha a opção “Sair”.

6.2 For, While, Do While

Nesta seção, o professor deve escolher se irá continuar com C++23, ou já fará a transição para o Python. O Objetivo é criar o entendimento dos artefatos sintáticos que foram criados para representar laços de repetição: `for`, `while` e `do while`.

O professor deverá escolher três problemas, entre aqueles que foram anteriormente resolvidos, e reescrevê-los utilizando os laços de repetição `for`, `while` e `do while`. A seguir, são apresentados exemplos de como cada laço pode ser utilizado.

6.2.1 Laço `for`

O laço `for` deve ser utilizado quando se sabe o número exato de iterações que serão realizadas. Por exemplo, para imprimir os números de 1 a 10:

```
for (int i = 1; i <= 10; i++) {
    std::cout << i << std::endl;
}
```

O problema a seguir pode ser resolvido utilizando o laço `for`:

A1. **Geração de Relatórios Financeiros**: a contabilidade da empresa “Números Certos” precisa de um sistema que gere rapidamente relatórios de multiplicação para auditorias internas. O Sr. Costa, o contador, frequentemente precisa visualizar a “tabuada” de um determinado `número` (multiplicado de 1 a 10). Projete um algoritmo que, dado um `número` inteiro, imprima sua tabuada completa.

Análise: Este problema foi resolvido com o fluxograma apresentado na Figure 6.11.

Figure 6.11: Fluxograma para representar o problema Geração de Relatórios Financeiros, sem e com o laço for.

Existe um módulo no padrão ISO, (106), que pode ser utilizado para representar o laço `for`, um hexágono. Contudo, este diagrama não contém uma simbologia clara agregando um nível extra de abstração. Na Figure 6.11, o laço `for` é representado por um bloco de processamento, com a definição do laço, indicando o que deve ser contado, até que valor, e, dentro deste bloco, os comandos que devem ser executados dentro do laço estão representados com fundo branco. Esta representação é exclusiva deste trabalho em uma tentativa de facilitar a abstração e o entendimento.

A tabela Table 6.46 apresenta o pseudocódigo para o algorítmo que resolve o problema **A1** usando `goto` para desvio de fluxo e usando o laço `for`.

Table 6.46: Comparação entre o pseudocódigo sem e com o laço `for`.

Pseudocódigo (sem laço for)	Pseudocódigo (com laço for)
shell ALGORITMO GeracaoTabuadaConciso ENTRADA: numero: inteiro SAÍDA: resultado: inteiro INÍCIO LER numero contador ← 1 ETIQUETA LOOP: resultado ← numero * contador ESCREVER numero, " x ", contador, " = ", resultado contador ← contador + 1 SE contador <= 10 ENTÃO IR PARA LOOP FIM 	shell ALGORITMO GeracaoTabuadaComFor ENTRADA: numero: inteiro SAÍDA: resultado: inteiro INÍCIO LER numero PARA contador DE 1 ATÉ 10 PASSO 1 FAÇA resultado ← numero * contador ESCREVER numero, " x ", contador, " = ", resultado FIM PARA FIM

O código Listing 6.12 exemplifica o uso do laço `for` em C++23.

Em Python, este algorítmo seria representado por:

```
numero = int(input("Digite um número: "))
for contador in range(1, 11):
    resultado = numero * contador
    print(f"{numero} x {contador} = {resultado}")
```

6.2.2 Laço `while`

O problema a seguir, que o aluno deve ter resolvido anteriormente, pode ser resolvido utilizando o laço `while`.

C1. **Análise de Desempenho de Vendas**: o gerente de vendas da "Vendas Top" precisa calcular a `media_de_vendas` por vendedor. Ele insere os valores de `venda` um por um. No entanto, apenas vendas entre 0 e 1000 Reais são consideradas válidas. O processo de inserção de vendas termina quando um valor negativo é digitado (este valor negativo não deve ser incluído na média). Ajude o gerente criando um algoritmo que calcule a média das vendas válidas.

Análise: os pseudocódigos do algorítmo para resolver o problema **C1** são apresentados na Table 6.47.

Table 6.47: Comparação entre o pseudocódigo com `goto` e com `while`.

Pseudocódigo (com goto)	Pseudocódigo (com while)

ALGORITMO AnaliseDesempenhoVendasConciso
 	
ENTRADA:
venda: real
 	
SAÍDA:
media: real
 	
INÍCIO
soma ← 0
  contador ← 0
 	
ETIQUETA LOOP:
  LER venda
  SE venda < 0 ENTÃO IR PARA CALCULAR_MEDIA
  SE (venda >= 0) E (venda <= 1000) ENTÃO
soma ← soma + venda
contador ← contador + 1
  IR PARA LOOP
 	
ETIQUETA CALCULAR_MEDIA:
  SE contador > 0 ENTÃO
  media ← soma / contador
  ESCREVER "Média:", media
  SENÃO
ESCREVER "Nenhuma venda válida"
FIM
 	
	ALGORITMO AnaliseDesempenhoVendasComWhile
	
	ENTRADA:
	venda: real
	
	SAÍDA:
	media: real
	
	INÍCIO
	  soma ← 0
	  contador ← 0
	
	ENQUANTO VERDADEIRO FAÇA
	LER venda
	  SE venda < 0 ENTÃO SAIR DO LOOP
	  SE (venda >= 0) E (venda <= 1000) ENTÃO
	  soma ← soma + venda
	  contador ← contador + 1
	  FIM ENQUANTO
	
	  SE contador > 0 ENTÃO
	media ← soma / contador
	ESCREVER "Média:", media
	SENÃO
	  ESCREVER "Nenhuma venda válida"
	FIM

O exercício **C1** pode ser resolvido em C++23 usando o código Listing 6.13.

Em Python, o mesmo algorítmo seria representado por:

```python
soma = 0
contador = 0
while True:
    venda = float(input("Digite o valor da venda (negativo para encerrar): "))
    if venda < 0:
        break
    if 0 <= venda <= 1000:
        soma += venda
        contador += 1
if contador > 0:
    media = soma / contador
    print(f"Média de vendas: {media}")
else:
    print("Nenhuma venda válida foi inserida.")
```

6.2.3 Laço `do while`

O problema a seguir, que o aluno deve ter resolvido anteriormente, pode ser resolvido utilizando o laço `do while`.

F1. **Sistema de Análise de Desempenho Acadêmico por Turmas**: na escola "Excelência Educacional", a coordenadora pedagógica Profa. Marina precisa analisar o desempenho de múltiplas turmas simultaneamente. O sistema deve processar `numero_turmas` turmas, e para cada turma, deve ler `numero_alunos` da turma e as `notas` de cada aluno (0-100). Para cada turma, calcular e exibir: média da turma, quantidade de alunos aprovados (nota 60), maior nota e menor nota. Adicionalmente, o sistema deve identificar qual turma teve a maior média geral e quantos alunos no total ficaram abaixo da média de sua respectiva

turma. Notas inválidas (fora do range 0-100) devem ser rejeitadas e solicitadas novamente. Se uma turma tiver menos de 3 alunos válidos, ela deve ser marcada como "Turma Inválida" e não incluída nas estatísticas gerais. No final, exibir um relatório consolidado com: média geral de todas as turmas válidas, total de alunos aprovados em toda a escola, e o número da turma com melhor desempenho.

Análise: o pseudocódigo para este exercício, usando `goto` pode ser representado como:

```
ALGORITMO AnaliseDesempenhoAcademicoConciso

ENTRADA:
    numeroTurmas, numeroAlunos, nota: inteiro

SAÍDA:
    mediaTurma, alunosAprovados, maiorNota, menorNota: real
    melhorTurma, totalAbaixoMedia: inteiro

INÍCIO
    LER numeroTurmas
    turmaAtual ← 1
    melhorMedia ← 0
    melhorTurma ← 0
    totalTurmasValidas ← 0
    somaMediasGeral ← 0
    totalAlunosAprovadosGeral ← 0
    totalAbaixoMedia ← 0

ETIQUETA LOOP_TURMAS:
    ESCREVER "=== TURMA ", turmaAtual, " ==="
    LER numeroAlunos
    alunoAtual ← 1
    somaTurma ← 0
    alunosValidos ← 0
    alunosAprovados ← 0
    maiorNota ← 0
    menorNota ← 100

ETIQUETA LOOP_ALUNOS:
    ESCREVER "Nota do aluno ", alunoAtual, ": "

ETIQUETA LER_NOTA:
    LER nota
    SE (nota < 0) OU (nota > 100) ENTÃO
        ESCREVER "Nota inválida! Digite novamente: "
        IR PARA LER_NOTA

    somaTurma ← somaTurma + nota
    alunosValidos ← alunosValidos + 1
    SE nota >= 60 ENTÃO alunosAprovados ← alunosAprovados + 1
    SE nota > maiorNota ENTÃO maiorNota ← nota
    SE nota < menorNota ENTÃO menorNota ← nota
```

```
    alunoAtual ← alunoAtual + 1
    SE alunoAtual <= numeroAlunos ENTÃO IR PARA LOOP_ALUNOS

    SE alunosValidos < 3 ENTÃO
        ESCREVER "Turma Inválida (menos de 3 alunos válidos)"
        IR PARA PROXIMA_TURMA

    mediaTurma ← somaTurma / alunosValidos
    ESCREVER "Média da turma: ", mediaTurma
    ESCREVER "Alunos aprovados: ", alunosAprovados
    ESCREVER "Maior nota: ", maiorNota
    ESCREVER "Menor nota: ", menorNota

    totalTurmasValidas ← totalTurmasValidas + 1
    somaMediasGeral ← somaMediasGeral + mediaTurma
    totalAlunosAprovadosGeral ← totalAlunosAprovadosGeral + alunosAprovados

    SE mediaTurma > melhorMedia ENTÃO
        melhorMedia ← mediaTurma
        melhorTurma ← turmaAtual

    alunoContador ← 1
ETIQUETA CONTAR_ABAIXO_MEDIA:
    SE alunoContador <= alunosValidos ENTÃO
        SE notaAluno[alunoContador] < mediaTurma ENTÃO
            totalAbaixoMedia ← totalAbaixoMedia + 1
        alunoContador ← alunoContador + 1
        IR PARA CONTAR_ABAIXO_MEDIA

ETIQUETA PROXIMA_TURMA:
    turmaAtual ← turmaAtual + 1
    SE turmaAtual <= numeroTurmas ENTÃO IR PARA LOOP_TURMAS

    ESCREVER "=== RELATÓRIO CONSOLIDADO ==="
    SE totalTurmasValidas > 0 ENTÃO
        mediaGeral ← somaMediasGeral / totalTurmasValidas
        ESCREVER "Média geral: ", mediaGeral
        ESCREVER "Total de alunos aprovados: ", totalAlunosAprovadosGeral
        ESCREVER "Melhor turma: ", melhorTurma
        ESCREVER "Alunos abaixo da média: ", totalAbaixoMedia
    SENÃO
        ESCREVER "Nenhuma turma válida encontrada"

FIM
```

Por outro lado, usando o laço `do while`, o pseudocódigo para este exercício pode ser representado por:

```
// Versão com goto (atual)
LER_NOTA:
    std::cin >> nota;
    if ((nota < 0) || (nota > 100)) {
```

```
        std::cout << "Nota inválida! Digite novamente: ";
        goto LER_NOTA;
    }

// Equivalente com DO-WHILE
do {
    std::cin >> nota;
    if ((nota < 0) || (nota > 100)) {
        std::cout << "Nota inválida! Digite novamente: ";
    }
} while ((nota < 0) || (nota > 100));
```

O exercício **F1** pode ser resolvido em C++23 usando o código Listing 6.14.
Em Python, o mesmo algorítmo seria representado por:

```
numero_alunos = int(input("Número de alunos: "))

soma = 0
alunos_validos = 0
alunos_aprovados = 0
maior_nota = 0
menor_nota = 100

for i in range(1, numero_alunos + 1):
    print(f"Aluno {i}")

    # SIMULAÇÃO DO DO-WHILE
    while True:
        nota = int(input("Digite a nota (0-100): "))

        if 0 <= nota <= 100:
            break  # Sai do loop se válida
        else:
            print("Nota inválida! Deve estar entre 0 e 100.")

    # PROCESSAMENTO DA NOTA VÁLIDA
    soma += nota
    alunos_validos += 1

    if nota >= 60:
        alunos_aprovados += 1

    if nota > maior_nota:
        maior_nota = nota

    if nota < menor_nota:
        menor_nota = nota

# SAÍDA
if alunos_validos > 0:
    media = soma / alunos_validos
```

```
    print("n=== RELATÓRIO DA TURMA ===")
    print(f"Média da turma: {media}")
    print(f"Alunos aprovados: {alunos_aprovados}")
    print(f"Maior nota: {maior_nota}")
    print(f"Menor nota: {menor_nota}")
else:
    print("Nenhum aluno válido.")
```

6.3 Problemas para Prática

Uma vez que o aluno tenha compreendido os laços de repetição, o professor pode optar por solicitar que os alunos refaçam os exercícios que fizeram, em um esforço de abstração e compreensão dos laços de repetição, seja por repetição ou pela apresentação de um conjunto novo de problemas. A seguir está uma lista com 12 problemas para aplicação usando a Técnica da Sequência de Fibonacci (1, 1, 2, 3, 5).

1f. Na empresa de consultoria matemática "Números Dourados", o analista precisa gerar os primeiros `N` termos da sequência de Fibonacci para análise de padrões. A sequência começa com 0 e 1, e cada termo seguinte é a soma dos dois anteriores (0, 1, 1, 2, 3, 5, 8, 13...). Crie um algoritmo que receba `N` e exiba os primeiros N termos da sequência.

2w. No supermercado "Compra Certa", o gerente precisa registrar vendas de produtos até que o estoque se esgote. O sistema inicia com `estoque_inicial` unidades de um produto. A cada venda, deve ler a `quantidade_vendida` e subtrair do estoque. O processo continua enquanto houver estoque disponível e a quantidade vendida for válida (positiva e não maior que o estoque atual). Ao final, exibir o total de vendas realizadas e a quantidade total vendida.

3d. Na calculadora científica "MathPro", o usuário deve escolher operações matemáticas através de um menu: 1-Adição, 2-Subtração, 3-Multiplicação, 4-Divisão, 5-Sair. Após escolher uma operação válida (1-4), ler dois números e exibir o resultado. Se escolher 5, encerrar o programa. Para qualquer opção inválida, exibir mensagem de erro e mostrar o menu novamente. O menu deve aparecer pelo menos uma vez.

4f. No laboratório meteorológico "Clima Exato", a pesquisadora precisa gerar uma tabela de conversão de Celsius para Fahrenheit e Kelvin. O algoritmo deve receber `temperatura_inicial`, `temperatura_final` e `incremento`, e exibir uma tabela com todas as temperaturas no intervalo especificado. Fórmulas: F = C × 9/5 + 32 e K = C + 273.15.

5w. Na empresa de jogos "Mente Ágil", o desenvolvedor criou um jogo onde o computador "pensa" em um número entre 1 e 50 (fixo: 27). O jogador faz tentativas até acertar. A cada tentativa errada, o jogo informa se o número é "maior" ou "menor" que o palpite. Contar quantas tentativas foram necessárias e exibir no final junto com uma mensagem de parabéns.

6d. No sistema de cadastro da rede social "ConectaMais", o usuário deve criar uma senha que atenda critérios específicos: ter pelo menos 8 caracteres, conter pelo menos um número e começar com letra maiúscula. O sistema deve solicitar a senha e validar todos os critérios. Se algum critério não for atendido, explicar qual falhou e solicitar nova senha. O processo continua até uma senha válida ser inserida.

7f. No banco "Investimento Seguro", o consultor financeiro precisa mostrar a evolução de um investimento ao longo dos anos. Dados o `capital_inicial`, `taxa_juros` anual e `numero_anos`, calcular e exibir o montante a cada ano usando a fórmula: Montante = Capital × (1 + taxa)^ano. Exibir uma tabela mostrando ano a ano a evolução do investimento.

8w. Na escola "Futuro Brilhante", o professor precisa calcular a média de uma turma. O sistema lê notas de alunos (valores entre 0 e 10) até que seja digitada uma nota inválida (negativa ou maior que 10), que indica o fim da entrada. Calcular e exibir: média da turma,

maior nota, menor nota, quantidade de alunos aprovados (nota 7) e percentual de aprovação.

9d. No banco digital "FinTech Nova", antes de executar uma transferência, o sistema deve confirmar os dados com o usuário. Exibir: valor da transferência, conta de destino e taxa. Perguntar "Confirma a transação? (S/N)". Se digitar 'S' ou 's', executar a transferência. Se digitar 'N' ou 'n', cancelar. Para qualquer outra resposta, mostrar "Opção inválida" e perguntar novamente. O sistema deve fazer a pergunta pelo menos uma vez.

10f. Na gráfica "Arte Digital", o designer precisa gerar padrões visuais usando asteriscos. Dado um número `N`, criar um padrão triangular onde a primeira linha tem 1 asterisco, a segunda tem 2, e assim por diante até N asteriscos na última linha. Exemplo para N=4:

```
*
**
***
****
```

11w. No sistema de segurança "Proteção Total", o usuário tem no máximo 3 tentativas para inserir a senha correta (fixo: "seguro123"). A cada tentativa errada, decrementar o contador e informar quantas tentativas restam. Se acertar, exibir "Acesso liberado". Se esgotar as tentativas, exibir "Acesso bloqueado - sistema travado". O processo para quando a senha for correta ou as tentativas acabarem.

12d. No hospital "Cuidar Bem", após cada atendimento, o paciente deve avaliar o serviço numa escala de 1 a 5 (1-Péssimo, 2-Ruim, 3-Regular, 4-Bom, 5-Excelente). O sistema deve solicitar a avaliação e só aceitar valores entre 1 e 5. Para valores inválidos, exibir "Avaliação inválida. Use valores de 1 a 5" e solicitar novamente. Após uma avaliação válida, exibir uma mensagem de agradecimento correspondente à nota dada.

6.4 Avaliação: Problemas Ad-hoc

Um problema *ad-hoc*, no domínio da Metodologia **DAAD**, é um quebra-cabeça que depende de uma observação específica, de uma regra matemática ou de uma simulação cuidadosa, utilizando apenas as ferramentas mais fundamentais da programação.

A dificuldade na solução destes problemas não será encontrada no processo de escrever o código, que muitas vezes é curto e direto. A dificuldade está em descobrir a chave que destrava a solução. Os problemas são resolvidos com lógica e com as técnicas de solução de problemas que estamos discutindo neste estudo. Contudo, como esta disciplina está proposta para os primeiros dois períodos dos cursos de Ciência e Engenharia da Computação, o enunciado dos problemas foi simplificado, transformando-os em problemas de **Raciocínio Algorítmico** simples. Ainda assim, eles exigem uma abordagem estruturada e lógica para serem resolvidos.

1. **Decomposição**: compreender e Testar com Casos Pequenos. A primeira etapa é garantir um entendimento absoluto do problema. Isso envolve ler o enunciado múltiplas vezes e, principalmente, testar o processo manualmente com exemplos simples. Para o problema "Número Perfeito", por exemplo, testar com N=6, N=12 e N=28 revelará o padrão de funcionamento. **Identificar o Desafio Central**: Qual é a restrição ou a dificuldade principal? No problema "Divisível por 11", o desafio não é a divisibilidade em si, mas como verificá-la processando o número dígito a dígito, sem armazená-los.

2. **Abstração**: quase todo problema ad-hoc tem um momento Eureka! Pode ser uma propriedade matemática, como a regra de divisibilidade por 11, uma otimização, como usar a raiz quadrada para encontrar divisores, ou a redução de um sistema complexo a uma regra simples. Aqui, existe a espiral da **Depuração**: testar a solução com números pequenos, como 1, 2, 3, 4, 5, até encontrar um padrão ou uma regra que funcione. A

abstração é o processo de identificar essa regra geral que pode ser aplicada a todos os casos do problema. A depuração permite validar a regra.

3. **Algoritmização**: apenas com a lógica da solução bem definida, a implementação pode começar. O objetivo é traduzir a ideia em um algoritmo claro, passo a passo, antes de escrever qualquer código. Este processo transforma um raciocínio abstrato em um conjunto de instruções concretas. Um fluxograma, um pseudocódigo ou o próprio código são formas de representar essa lógica. A implementação deve ser feita com clareza, seguindo a lógica definida na etapa anterior.

4. **Depuração**: testar a solução com casos de teste variados, incluindo os limites do problema. A depuração é essencial para garantir que a solução funcione corretamente em todos os cenários possíveis. Isso envolve verificar se a implementação atende às condições do problema e se produz os resultados esperados.

A Table 6.48 apresenta um resumo dos problemas selecionados para avaliação deste módulo, apresentando uma visão geral dos cinco problemas analisados, destacando os conceitos fundamentais que cada um exercita.

Table 6.48: Resumo dos problemas selecionados para avaliação deste módulo.

Nome do Problema	Conceito Chave	Estruturas Fundamentais
Divisibilidade por 11	Regra de Divisibilidade Matemática (Posições Ímpares vs Pares)	Laço de Repetição, Decisão, Manipulação de Dígitos
Verificação de Número Palíndromo	Reversão e Comparação de Sequências Numéricas	Laço de Repetição, Decisão, Construção de Número Invertido
Raiz Digital de um Número	Iteração Convergente (Soma Repetida até Dígito Único)	Laços Aninhados, Decisão, Soma Acumulativa
Número Feliz	Detecção de Ciclos com Limite Superior	Laços Aninhados, Decisão, Aritmética de Quadrados
Maior e Menor Dígito	Busca de Extremos em Sequência Linear	Laço de Repetição, Decisão, Comparação de Valores

As seções a seguir detalham cada um dos cinco problemas, com enunciado estendido, para diminuir a necessidade de pesquisa de matemática teórica, seguidos de uma análise detalhada do problema frente à Metodologia **DAAD**.

6.4.1 Problema 1: Divisibilidade por 11

Dado um número inteiro positivo N ($1 \leq N \leq 10^9$), determine se N é divisível por 11. Você só pode usar operações aritméticas básicas e laços de repetição, e não pode armazenar os dígitos do número em estruturas de dados compostas como arrays ou estruturas similares.

Divisibilidade: um número a é divisível por um número b se a divisão a/b resultar em um número inteiro, sem resto. A chave para resolver o problema é a conhecida regra de divisibilidade por 11: um número é divisível por 11 se a diferença absoluta entre a soma dos dígitos em posições ímpares e a soma dos dígitos em posições pares for um número divisível por 11. Por exemplo, para $N = 132$, os dígitos são 1, 3 e 2. Posições (da direita para a esquerda): 2 (pos 1, ímpar), 3 (pos 2, par), 1 (pos 3, ímpar).

- Soma ímpar: $2 + 1 = 3$

- Soma par: 3
- Diferença: $|3 - 3| = 0$. Como 0 é divisível por 11, 132 também é.

6.4.1.1 Decomposição e Abstração

A primeira abordagem seria simplesmente calcular $N\%11$. No entanto, o problema foi desenhado para treinar a manipulação de um número dígito a dígito. Esta restrição induz a necessidade de entender a divisibilidade em um nível mais fundamental. O aluno, ou grupo, terá que descobrir como processar o número sem armazenar seus dígitos, o que é um desafio interessante.

O desafio é implementar esta regra sem usar arrays. Para isso, será necessário processar o número N da direita para a esquerda usando operações de módulo e divisão. O laço while $(N > 0)$ é a ferramenta perfeita para isso. Em cada iteração, $N\%10$ nos dá o último dígito, e $N = N/10$ remove esse dígito. Para alternar entre as somas par e ímpar, é possível usar uma variável de controle (*posicao*) que alterna entre 0 e 1 a cada passo.

6.4.1.2 Algoritmização

```
ALGORITMO VerificaDivisibilidadePor11Conciso

ENTRADA:
    N: inteiro

SAÍDA:
    divisivel: booleano

INÍCIO
    LER N
    numeroOriginal ← N
    somaImpar ← 0
    somaPar ← 0
    posicao ← 1

ETIQUETA EXTRAIR_DIGITOS:
    SE N = 0 ENTÃO IR PARA CALCULAR_DIFERENCA

    digito ← N % 10
    N ← N / 10

    SE (posicao % 2 = 1) ENTÃO
        somaImpar ← somaImpar + digito
    SENÃO
        somaPar ← somaPar + digito

    posicao ← posicao + 1
    IR PARA EXTRAIR_DIGITOS

ETIQUETA CALCULAR_DIFERENCA:
    SE somaImpar >= somaPar ENTÃO
        diferenca ← somaImpar - somaPar
```

```
    SENÃO
        diferenca ← somaPar - somaImpar

    SE (diferenca % 11 = 0) ENTÃO
        divisivel ← VERDADEIRO
        ESCREVER numeroOriginal, " é divisível por 11"
    SENÃO
        divisivel ← FALSO
        ESCREVER numeroOriginal, " não é divisível por 11"

    ESCREVER "Soma posições ímpares:", somaImpar
    ESCREVER "Soma posições pares:", somaPar
    ESCREVER "Diferença absoluta:", diferenca

    RETORNAR divisivel
FIM
```

6.4.1.3 Depuração

O erro mais comum é confundir a ordem das posições ou não alternar a variável de controle corretamente. É útil testar com caneta e papel usando um número como 98765 para garantir que as somas estão sendo calculadas corretamente. A beleza desta solução está em como ela decompõe um número grande em uma série de operações simples, usando apenas algumas variáveis para manter o estado.

6.4.2 Problema 2: Verificação de Número Palíndromo

Dado um número inteiro positivo N ($1 \leq N \leq 10^9$), determine se N é um palíndromo. Um número palíndromo é aquele que permanece o mesmo quando seus dígitos são invertidos. Você só pode usar operações aritméticas básicas e laços de repetição, e não pode armazenar os dígitos do número em estruturas de dados compostas como arrays ou strings.

Número Palíndromo: um número que se lê da mesma forma da esquerda para a direita e da direita para a esquerda. Por exemplo, 12321 é palíndromo porque invertido continua sendo 12321. Já 12345 não é palíndromo porque invertido vira 54321. Para resolver este problema, precisamos construir o número invertido e comparar com o original.

6.4.2.1 Decomposição e Abstração

O desafio está em construir o número invertido sem usar arrays para armazenar os dígitos. A estratégia é processar o número original da direita para a esquerda, extraindo cada dígito com N mod 10 e construindo simultaneamente o número invertido. Para construir o invertido, multiplicamos o valor atual por 10 e somamos o novo dígito. O algoritmo termina quando o número original se torna 0, momento em que comparamos o número invertido com o valor original salvo.

6.4.2.2 Algoritmização

```
ALGORITMO VerificaPalindromoConciso

ENTRADA:
    N: inteiro
```

```
SAÍDA:
    palindromo: booleano

INÍCIO
    LER N
    numeroOriginal ← N
    numeroInvertido ← 0

ETIQUETA INVERTER_NUMERO:
    SE N = 0 ENTÃO IR PARA COMPARAR

    digito ← N % 10
    numeroInvertido ← numeroInvertido * 10 + digito
    N ← N / 10
    IR PARA INVERTER_NUMERO

ETIQUETA COMPARAR:
    SE numeroOriginal = numeroInvertido ENTÃO
        palindromo ← VERDADEIRO
        ESCREVER numeroOriginal, " é palíndromo"
    SENÃO
        palindromo ← FALSO
        ESCREVER numeroOriginal, " não é palíndromo"

    ESCREVER "Número original:", numeroOriginal
    ESCREVER "Número invertido:", numeroInvertido

    RETORNAR palindromo
FIM
```

6.4.2.3 Depuração

O erro mais frequente é esquecer de salvar o número original antes de modificá-lo, resultando em comparação incorreta. Outro erro comum é a construção incorreta do número invertido. É importante ter em mente que para adicionar um dígito à direita, multiplicamos por 10 e somamos o dígito. Tabelas de Rastreio podem ser criadas com números simples como 121 e 12321 para verificar a lógica.

6.4.3 Problema 3: Raiz Digital de um Número

Dado um número inteiro positivo N ($1 \leq N \leq 10^9$), calcule sua raiz digital. A raiz digital é obtida somando repetidamente os dígitos de um número até obter um único dígito. Você só pode usar operações aritméticas básicas e laços de repetição, e não pode armazenar os dígitos do número em estruturas de dados compostas.

Raiz Digital (RD): resultado da soma iterativa dos dígitos até sobrar apenas um dígito. Por exemplo, para $N = 9875$: soma dos dígitos $RD = 9 + 8 + 7 + 5 = 29$. Como 29 tem dois dígitos, repetimos: $2 + 9 = 11$. Como 11 tem dois dígitos: $1 + 1 = 2$. A raiz digital de 9875 é 2. Existe uma fórmula matemática para isso que não pode ser usada para resolver este problema. O objetivo é implementar o processo iterativo para treinar manipulação de dígitos.

6.4.3.1 Decomposição e Abstração

O algoritmo precisa de dois laços de repetição aninhados: um externo que continua enquanto o número tiver mais de um dígito, e um interno que soma todos os dígitos do número atual. A cada iteração do loop externo, o número é substituído pela soma de seus dígitos. O processo para quando o número se torna menor que 10, um único dígito. Para verificar se um número tem mais de um dígito, basta testar se é maior ou igual a 10.

6.4.3.2 Algoritmização

```
ALGORITMO CalculaRaizDigitalConciso

ENTRADA:
    N: inteiro

SAÍDA:
    raizDigital: inteiro

INÍCIO
    LER N
    numeroOriginal ← N

ETIQUETA LOOP_PRINCIPAL:
    SE N < 10 ENTÃO IR PARA RESULTADO

    soma ← 0
    numeroTemp ← N

ETIQUETA SOMAR_DIGITOS:
    SE numeroTemp = 0 ENTÃO
        N ← soma
        IR PARA LOOP_PRINCIPAL

    digito ← numeroTemp % 10
    soma ← soma + digito
    numeroTemp ← numeroTemp / 10
    IR PARA SOMAR_DIGITOS

ETIQUETA RESULTADO:
    raizDigital ← N
    ESCREVER "Número original:", numeroOriginal
    ESCREVER "Raiz digital:", raizDigital

    RETORNAR raizDigital
FIM
```

6.4.3.3 Depuração

Um erro comum é não reinicializar corretamente as variáveis a cada iteração. Certifique-se de que `soma` é zerada antes de cada cálculo e que `numeroTemp` recebe o valor atual de N. Outro erro é confundir as condições de parada dos laços de repetição. Uma forma interessante de

depurar é a criação de tabela de rastreio com números como 38 ($3+8=11$, $1+1=2$) para verificar múltiplas iterações.

6.4.4 Problema 4: Número Feliz

Dado um número inteiro positivo N ($1 \leq N \leq 10^9$), determine se N é um "número feliz". Um número é feliz se, ao repetir o processo de substituir o número pela soma dos quadrados de seus dígitos, eventualmente chegamos ao número 1. Se o processo entrar em um ciclo que não inclui 1, o número é "infeliz". Você só pode usar operações aritméticas básicas e laços de repetição, limitando-se a 100 iterações para detectar ciclos.

Número Feliz: conceito matemático onde aplicamos repetidamente a operação "soma dos quadrados dos dígitos". Por exemplo, $N = 23$: $2^2+3^2 = 4+9 = 13$. Depois $1^2+3^2 = 1+9 = 10$. Depois $1^2+0^2 = 1+0 = 1$. Como chegamos a 1, o número 23 é feliz. Alguns números entram em ciclos infinitos, como $4 \to 16 \to 37 \to 58 \to 89 \to 145 \to 42 \to 20 \to 4$, sendo portanto infelizes.

6.4.4.1 Decomposição e Abstração

O algoritmo precisa repetir o processo de calcular a soma dos quadrados dos dígitos até chegar a 1 (feliz) ou detectar um possível ciclo. Como a identificação automática de ciclos requer estruturas de dados mais complexas, como arrays e conjunto, as iterações estão limitadas a 100. Se após 100 iterações o valor 1 não for alcançado, o algoritmo deve assumir que o número é infeliz. Para cada iteração, os dígitos são extraídos, seus quadrados são calculados e somados, substituindo o número original por esta soma.

6.4.4.2 Algoritmização

```
ALGORITMO VerificaNumeroFelizConciso

ENTRADA:
    N: inteiro

SAÍDA:
    feliz: booleano

INÍCIO
    LER N
    numeroOriginal ← N
    iteracoes ← 0

ETIQUETA LOOP_PRINCIPAL:
    SE N = 1 ENTÃO IR PARA NUMERO_FELIZ
    SE iteracoes >= 100 ENTÃO IR PARA NUMERO_INFELIZ

    soma ← 0
    numeroTemp ← N

ETIQUETA CALCULAR_SOMA_QUADRADOS:
    SE numeroTemp = 0 ENTÃO
        N ← soma
        iteracoes ← iteracoes + 1
```

```
        IR PARA LOOP_PRINCIPAL

    digito ← numeroTemp % 10
    soma ← soma + (digito * digito)
    numeroTemp ← numeroTemp / 10
    IR PARA CALCULAR_SOMA_QUADRADOS

ETIQUETA NUMERO_FELIZ:
    feliz ← VERDADEIRO
    ESCREVER numeroOriginal, " é um número feliz"
    ESCREVER "Chegou a 1 em", iteracoes, "iterações"
    IR PARA FIM_ALGORITMO

ETIQUETA NUMERO_INFELIZ:
    feliz ← FALSO
    ESCREVER numeroOriginal, " é um número infeliz"
    ESCREVER "Não chegou a 1 em 100 iterações"

ETIQUETA FIM_ALGORITMO:
    RETORNAR feliz
FIM
```

6.4.4.3 Depuração

O erro mais comum é calcular incorretamente o quadrado dos dígitos ou somar incorretamente. O aluno deve verificar se `digito * digito` está sendo calculado corretamente. Outro erro é não reinicializar `soma` a cada iteração. Números conhecidos: 1 (já é feliz), 7 (feliz), e 4 (infeliz) são interessantes para a criação de tabelas de rastreio.

6.4.5 Problema 5: Maior e Menor Dígito

Dado um número inteiro positivo N ($1 \leq N \leq 10^9$), encontre o maior e o menor dígito que compõem este número. Você só pode usar operações aritméticas básicas e laços de repetição, e não pode armazenar os dígitos do número em estruturas de dados compostas como arrays.

Maior e Menor Dígito: em qualquer número, cada dígito está entre 0 e 9. O objetivo é percorrer todos os dígitos do número e manter registro do maior e menor encontrados. Por exemplo, para $N = 49327$, os dígitos são 4, 9, 3, 2, 7. O maior dígito é 9 e o menor é 2. Este problema treina a capacidade de manter estado (maior e menor) durante a iteração pelos dígitos.

6.4.5.1 Decomposição e Abstração

A estratégia é inicializar o maior dígito com 0 (menor valor possível) e o menor dígito com 9, maior valor possível. Conforme extraímos cada dígito usando N mod 10, comparamos com os valores atuais de maior e menor, atualizando quando necessário. O processo continua até que todos os dígitos sejam processados ($N = 0$). A beleza desta solução está na simplicidade: apenas duas comparações por dígito são suficientes para manter o estado desejado.

6.4.5.2 Algoritmização

```
ALGORITMO EncontraMaiorMenorDigitoConciso

ENTRADA:
    N: inteiro

SAÍDA:
    maiorDigito, menorDigito: inteiro

INÍCIO
    LER N
    numeroOriginal ← N
    maiorDigito ← 0
    menorDigito ← 9

ETIQUETA PROCESSAR_DIGITOS:
    SE N = 0 ENTÃO IR PARA RESULTADO

    digito ← N % 10
    N ← N / 10

    SE digito > maiorDigito ENTÃO
        maiorDigito ← digito

    SE digito < menorDigito ENTÃO
        menorDigito ← digito

    IR PARA PROCESSAR_DIGITOS

ETIQUETA RESULTADO:
    ESCREVER "Número original:", numeroOriginal
    ESCREVER "Maior dígito:", maiorDigito
    ESCREVER "Menor dígito:", menorDigito
    ESCREVER "Diferença:", (maiorDigito - menorDigito)

    RETORNAR maiorDigito, menorDigito
FIM
```

6.4.5.3 Depuração

Um erro inicial comum é não inicializar corretamente as variáveis: maior deve começar em 0 (menor valor possível para um dígito) e menor deve começar em 9. Outro erro é esquecer de processar números com dígitos 0, certifique-se de que o algoritmo funciona corretamente com números como 1029. Para a tabela de rastreio, considere casos extremos como 1111 e 9000.

Listing 6.3

```cpp
#include <iostream>

int main() {
    // ENTRADA
    double rendaMensal, idade, scoreCredito, tempoEmprego;

    std::cout << "Renda mensal (R$): ";
    std::cin >> rendaMensal;

    std::cout << "Idade: ";
    std::cin >> idade;

    std::cout << "Score de crédito: ";
    std::cin >> scoreCredito;

    std::cout << "Tempo de emprego (anos): ";
    std::cin >> tempoEmprego;

    // PROCESSAMENTO
    bool aprovado = (rendaMensal >= 2000) && (idade >= 21 && idade <= 65) &&
                    (scoreCredito >= 600) && (tempoEmprego >= 2);

    double valorMaximoEmprestimo;

    if (aprovado) {
        valorMaximoEmprestimo = rendaMensal * 10;
        if (scoreCredito > 750) {
            valorMaximoEmprestimo = valorMaximoEmprestimo * 1.2;
        }
        valorMaximoEmprestimo = valorMaximoEmprestimo;
    } else {
        valorMaximoEmprestimo = 0;
    }

    // SAÍDA
    if (aprovado) {
        std::cout << "Empréstimo APROVADO" << std::endl;
        std::cout << "Valor máximo: R$ " << valorMaximoEmprestimo << std::endl;
    } else {
        std::cout << "Empréstimo NEGADO" << std::endl;
        std::cout << "Critérios não atendidos" << std::endl;
    }

    return 0;
}
```

Listing 6.4

```cpp
#include <iostream>

int main() {
    // ENTRADA
    double temperatura, pressaoSistolica, frequenciaCardiaca, nivelDor, idade;

    std::cout << "Temperatura (°C): ";
    std::cin >> temperatura;

    std::cout << "Pressão sistólica (mmHg): ";
    std::cin >> pressaoSistolica;

    std::cout << "Frequência cardíaca (bpm): ";
    std::cin >> frequenciaCardiaca;

    std::cout << "Nível de dor (0-10): ";
    std::cin >> nivelDor;

    std::cout << "Idade (anos): ";
    std::cin >> idade;

    // PROCESSAMENTO
    std::string classificacao;

    if ((temperatura > 39) || (pressaoSistolica < 90 || pressaoSistolica > 180) ||
        (frequenciaCardiaca < 50 || frequenciaCardiaca > 120)) {
        classificacao = "Emergência (vermelho)";
    } else if ((temperatura >= 38 && temperatura <= 39) ||
               (pressaoSistolica >= 90 && pressaoSistolica <= 100) ||
               (pressaoSistolica >= 160 && pressaoSistolica <= 180) || (nivelDor > 7)) {
        classificacao = "Muito Urgente (laranja)";
    } else if ((temperatura >= 37.5 && temperatura < 38) || (nivelDor >= 4 && nivelDor <=
               (idade > 65)) {
        classificacao = "Urgente (amarelo)";
    } else {
        classificacao = "Pouco Urgente (verde)";
    }

    // SAÍDA
    std::cout << "Classificação: " << classificacao << std::endl;

    return 0;
}
```

Listing 6.5

```cpp
#include <iostream>

int main() {
    // ENTRADA
    double distanciaKm, horario, diaSemana;
    std::string condicoesClima, demandaRegiao;

    std::cout << "Distância (km): ";
    std::cin >> distanciaKm;

    std::cout << "Horário (0-23): ";
    std::cin >> horario;

    std::cout << "Dia da semana (1-7): ";
    std::cin >> diaSemana;

    std::cout << "Condições do clima (normal/chuva/tempestade): ";
    std::cin >> condicoesClima;

    std::cout << "Demanda da região (baixa/media/alta): ";
    std::cin >> demandaRegiao;

    // PROCESSAMENTO
    double tarifaFinal = distanciaKm * 2.50;

    bool horarioPico = ((horario >= 7 && horario <= 9) || (horario >= 17 && horario <= 19
    bool fimSemana = (diaSemana == 6 || diaSemana == 7);
    bool tempestade = (condicoesClima == "tempestade");
    bool demandaAlta = (demandaRegiao == "alta");

    if (horarioPico) {
        tarifaFinal = tarifaFinal * 1.5;
    }

    if (fimSemana) {
        tarifaFinal = tarifaFinal * 1.2;
    }

    if (condicoesClima == "chuva") {
        tarifaFinal = tarifaFinal * 1.3;
    }

    if (tempestade) {
        tarifaFinal = tarifaFinal * 1.8;
    }

    if (demandaRegiao == "media") {
        tarifaFinal = tarifaFinal * 1.1;
    }

    if (demandaAlta) {
        tarifaFinal = tarifaFinal * 1.4;
    }

    if (horarioPico && fimSemana && tempestade && demandaAlta) {
```

Listing 6.6

```
#include <iostream>

int main() {
    // ENTRADA
    double idade, rendaMensal, patrimonioLiquido, experienciaInvestimentos, toleranciaPer
    std::string objetivoPrazo;

    std::cout << "Idade: ";
    std::cin >> idade;

    std::cout << "Renda mensal (R$): ";
    std::cin >> rendaMensal;

    std::cout << "Patrimônio líquido (R$): ";
    std::cin >> patrimonioLiquido;

    std::cout << "Experiência em investimentos (anos): ";
    std::cin >> experienciaInvestimentos;

    std::cout << "Tolerância à perda (%): ";
    std::cin >> toleranciaPerda;

    std::cout << "Objetivo de prazo (curto/medio/longo): ";
    std::cin >> objetivoPrazo;

    // PROCESSAMENTO
    int pontuacaoTotal = 0;

    if (idade < 30) {
        pontuacaoTotal = pontuacaoTotal + 2;
    } else if (idade <= 50) {
        pontuacaoTotal = pontuacaoTotal + 1;
    }

    if (rendaMensal > 10000) {
        pontuacaoTotal = pontuacaoTotal + 2;
    } else if (rendaMensal >= 5000) {
        pontuacaoTotal = pontuacaoTotal + 1;
    }

    if (patrimonioLiquido > 100000) {
        pontuacaoTotal = pontuacaoTotal + 2;
    } else if (patrimonioLiquido >= 50000) {
        pontuacaoTotal = pontuacaoTotal + 1;
    }

    if (experienciaInvestimentos > 5) {
        pontuacaoTotal = pontuacaoTotal + 2;
    } else if (experienciaInvestimentos >= 2) {
        pontuacaoTotal = pontuacaoTotal + 1;
    }

    if (toleranciaPerda > 20) {
        pontuacaoTotal = pontuacaoTotal + 2;
    } else if (toleranciaPerda >= 10) {
```

Listing 6.7

```
#include <iostream>

int main() {
    // ENTRADA
    double diametroMm, pesoGramas, durezaHrc, temperaturaTeste;
    int loteProducao;

    std::cout << "Diâmetro (mm): ";
    std::cin >> diametroMm;

    std::cout << "Peso (gramas): ";
    std::cin >> pesoGramas;

    std::cout << "Dureza (HRC): ";
    std::cin >> durezaHrc;

    std::cout << "Temperatura de teste (°C): ";
    std::cin >> temperaturaTeste;

    std::cout << "Lote de produção: ";
    std::cin >> loteProducao;

    // PROCESSAMENTO
    double toleranciaExtra;
    if (loteProducao % 2 == 0) {
        toleranciaExtra = 1.05;
    } else {
        toleranciaExtra = 1.0;
    }

    double diamMin = 49.5 * toleranciaExtra;
    double diamMax = 50.5 * toleranciaExtra;
    double pesoMin = 190 * toleranciaExtra;
    double pesoMax = 210 * toleranciaExtra;
    double durezaMin = 45 * toleranciaExtra;
    double durezaMax = 55 * toleranciaExtra;
    double tempMin = 20 * toleranciaExtra;
    double tempMax = 25 * toleranciaExtra;

    int parametrosForaEspec = 0;
    double maxDesvio = 0;

    if ((diametroMm < diamMin) || (diametroMm > diamMax)) {
        parametrosForaEspec = parametrosForaEspec + 1;
        double desvio;
        if (diametroMm < diamMin) {
            desvio = (diamMin - diametroMm) / 50 * 100;
        } else {
            desvio = (diametroMm - diamMax) / 50 * 100;
        }
        if (desvio > maxDesvio) {
            maxDesvio = desvio;
        }
    }
```

Listing 6.8

```cpp
#include <iostream>

int main() {
    // ENTRADA
    double tempoEmpresa, avaliacaoDesempenho, certificacoesProfissionais, idade, salarioA
    std::string nivelFormacao;

    std::cout << "Tempo na empresa (anos): ";
    std::cin >> tempoEmpresa;

    std::cout << "Avaliação de desempenho (1-10): ";
    std::cin >> avaliacaoDesempenho;

    std::cout << "Nível de formação (medio/superior/pos): ";
    std::cin >> nivelFormacao;

    std::cout << "Certificações profissionais (quantidade): ";
    std::cin >> certificacoesProfissionais;

    std::cout << "Idade: ";
    std::cin >> idade;

    std::cout << "Salário atual (R$): ";
    std::cin >> salarioAtual;

    // PROCESSAMENTO
    std::string classificacao;

    if ((idade > 55) && (salarioAtual > 15000)) {
        classificacao = "Plano Sucessão";
    } else {
        bool formacaoAvancada = (nivelFormacao == "superior") || (nivelFormacao == "pos")

        if ((tempoEmpresa >= 3) && (avaliacaoDesempenho >= 8) && formacaoAvancada && (cer
            classificacao = "Promoção Imediata";
        } else if ((tempoEmpresa >= 2) && (avaliacaoDesempenho >= 7) && (formacaoAvancada
            classificacao = "Promoção Condicional";
        } else if ((tempoEmpresa >= 1) && (avaliacaoDesempenho >= 6)) {
            classificacao = "Desenvolvimento Necessário";
        } else {
            classificacao = "Não Elegível";
        }
    }

    // SAÍDA
    std::cout << "Classificação: " << classificacao << std::endl;

    return 0;
}
```

Listing 6.9

```cpp
#include <iostream>

int main() {
    // ENTRADA
    int numero, contador;

    std::cout << "Digite um número para a tabuada: ";
    std::cin >> numero;

    // PROCESSAMENTO
    contador = 1;

LOOP:
    int resultado = numero * contador;
    std::cout << numero << " x " << contador << " = " << resultado << std::endl;
    contador = contador + 1;
    if (contador <= 10) goto LOOP;

    return 0;
}
```

Listing 6.10

```cpp
#include <iostream>

int main() {
    // PROCESSAMENTO
    int contador = 10;

LOOP:
    std::cout << contador << std::endl;
    if (contador == 0) {
        std::cout << "Lançar!" << std::endl;
        goto FIM;
    } else {
        contador = contador - 1;
        goto LOOP;
    }

FIM:
    return 0;
}
```

Listing 6.11

```cpp
#include <iostream>

int main() {
    // ENTRADA
    double venda;

    // PROCESSAMENTO
    double soma = 0;
    int contador = 0;

LOOP:
    std::cout << "Digite o valor da venda (negativo para encerrar): ";
    std::cin >> venda;
    if (venda < 0) goto CALCULAR_MEDIA;
    if ((venda >= 0) && (venda <= 1000)) {
        soma = soma + venda;
        contador = contador + 1;
    }
    goto LOOP;

CALCULAR_MEDIA:
    if (contador > 0) {
        double media = soma / contador;
        std::cout << "Média de vendas: " << media << std::endl;
    } else {
        std::cout << "Nenhuma venda válida foi inserida." << std::endl;
    }

    return 0;
}
```

Listing 6.12

```cpp
#include <iostream>

int main() {
    int numero;
    std::cout << "Digite um número: ";
    std::cin >> numero;

    for (int contador = 1; contador <= 10; contador++) {
        int resultado = numero * contador;
        std::cout << numero << " x " << contador << " = " << resultado << std::endl;
    }

    return 0;
}
```

Listing 6.13

```cpp
#include <iostream>

int main() {
    // ENTRADA
    double venda;

    // PROCESSAMENTO
    double soma = 0;
    int contador = 0;

    while (true) {
        std::cout << "Digite o valor da venda (negativo para encerrar): ";
        std::cin >> venda;

        if (venda < 0) break;

        if ((venda >= 0) && (venda <= 1000)) {
            soma = soma + venda;
            contador = contador + 1;
        }
    }

    // SAÍDA
    if (contador > 0) {
        double media = soma / contador;
        std::cout << "Média de vendas: " << media << std::endl;
    } else {
        std::cout << "Nenhuma venda válida foi inserida." << std::endl;
    }

    return 0;
}
```

Listing 6.14

```cpp
#include <iostream>

int main() {
    // ENTRADA
    int numeroAlunos;
    int nota;

    std::cout << "Número de alunos: ";
    std::cin >> numeroAlunos;

    // PROCESSAMENTO
    double soma = 0;
    int alunosValidos = 0;
    int alunosAprovados = 0;
    int maiorNota = 0;
    int menorNota = 100;

    for (int i = 1; i <= numeroAlunos; i++) {
        std::cout << "Aluno " << i << std::endl;

        // VALIDAÇÃO COM DO-WHILE
        do {
            std::cout << "Digite a nota (0-100): ";
            std::cin >> nota;

            if ((nota < 0) || (nota > 100)) {
                std::cout << "Nota inválida! Deve estar entre 0 e 100." << std::endl;
            }
        } while ((nota < 0) || (nota > 100));

        // PROCESSAMENTO DA NOTA VÁLIDA
        soma = soma + nota;
        alunosValidos = alunosValidos + 1;

        if (nota >= 60) {
            alunosAprovados = alunosAprovados + 1;
        }

        if (nota > maiorNota) {
            maiorNota = nota;
        }

        if (nota < menorNota) {
            menorNota = nota;
        }
    }

    // SAÍDA
    if (alunosValidos > 0) {
        double media = soma / alunosValidos;
        std::cout << "n=== RELATÓRIO DA TURMA ===" << std::endl;
        std::cout << "Média da turma: " << media << std::endl;
        std::cout << "Alunos aprovados: " << alunosAprovados << std::endl;
        std::cout << "Maior nota: " << maiorNota << std::endl;
        std::cout << "Menor nota: " << menorNota << std::endl;
```

7 Módulo 2: Semanas 4-8 (20 Horas-Aula): Abstração e Reconhecimento de Padrões

No módulo 2, os alunos aprofundam-se na **Abstração** e no **Reconhecimento de Padrões**, dois pilares fundamentais do **Raciocínio Algoritmico**. Este módulo é estruturado em torno da Metodologia **DAAD**, que inclui as etapas de **D**ecomposição, **A**bstração, **A**lgoritmização e **D**epuração. Neste módulo devem ser abordados os seguintes tópicos:

- Princípios de abstração: identificação de informações essenciais e criação de modelos simplificados;
- Reconhecimento de padrões: identificação de similaridades, tendências e generalizações;
- Abstração na programação: componentes reutilizáveis, funções e estruturas de controle;
- Estruturas de dados simples: vectors, arrays e structs;
- Exercícios avançados de interpretação, visualização de dados e simulação de sistemas;
- Identificação de padrões de código recorrentes para refatoração.

Além disso, a partir do exercício **H2**, os alunos devem ser incentivados a dividir o trabalho em partes para facilitar a implementação e a depuração. A divisão do trabalho é uma habilidade essencial para o desenvolvimento de software, pois permite que os alunos se concentrem em partes específicas do problema, facilitando a identificação de erros e a implementação de soluções. Neste módulo, não usaremos nenhum sistema de controle de versão, propositalmente, para que os alunos possam aprender a importância de dividir o trabalho em partes e como isso pode facilitar a implementação e a depuração. Em cada um destes exercícios existe uma proposta de divisão que serve de orientação para o professor.

Embora o reconhecimento de padrões não esteja explicitado no acrônimo **DAAD**, esta competência é um elemento essencial do **Raciocínio Algorítmico** e da resolução de problemas. Exercícios de **Algoritmização** mais complexos serão propostos para induzir a criatividade, além de fomentar a capacidade de reconhecimento de padrões. Neste contexto, o reconhecimento de padrões é visto como uma habilidade que permite aos alunos identificar e aplicar soluções previamente aprendidas a novos problemas, facilitando a abstração e a decomposição de problemas complexos.

Neste módulo, o professor deve decidir qual linguagem de programação será utilizada durantes os Módulos 2 e 3. A linguagem escolhida deve ser adequada para o nível dos alunos e a familiaridade do professor, e permitir a implementação dos conceitos de abstração e reconhecimento de padrões. A Python (71) é uma escolha comum, mas outras linguagens como C++ (106) podem ser utilizadas. Neste documento será utilizada a linguagem C++ como exemplo, mas os conceitos são aplicáveis a qualquer linguagem de programação. Se o professor optar por continuar com o C++ 23, é importante observar que a partir deste módulo, os exemplos usarão artefatos modernos desta linguagem, como o uso de `auto` e `ranges` para dedução de tipos e a biblioteca padrão C++ (STL) para manipulação de dados.

7.1 Abstração: Reutilização

Aplicação da abstração na programação: criação de componentes reutilizáveis, funções, laços de repetição e estruturas de dados. Utilizando a Linguagem C++ com as abstrações de função, `for`, `while`, `do while`.

Antes de ofertar os novos exercícios, o professor pode revisar os conceitos de abstração que foram introduzidos no Módulo 1. Como por exemplo, o professor pode considerar usar os exercícios 4 exercícios a seguir para revisar os conceitos de abstração e controle de fluxo. Observe que os exercícios são complexos e demandam tempo de implementação, por isso o professor deve escolher os exercícios de acordo com o tamanho e o resultado da turma, mas deve se manter na Técnica da Sequência de Fibonacci. Vimos a Técnica da Sequência de Fibonacci na Section 6.1.1 nesta técnica além de tempos e quantidades, o professor precisa se preocupar que a devolutiva deve ser feita no quadro, com o professor explicando cada passo do algoritmo, enfatizando a importância da decomposição e controle de fluxo. A seguir está uma lista de problemas que podem ser usados para esta atividade:

Revisão 1: dado um número inteiro positivo N ($1 \leq N \leq 10^9$), encontre o maior e o menor dígito que compõem este número. Você só pode usar operações aritméticas básicas e laços de repetição, e não pode armazenar os dígitos do número em estruturas de dados compostas como arrays.

Em qualquer número, cada dígito está entre 0 e 9. O objetivo é percorrer todos os dígitos do número e manter registro do maior e menor encontrados. Por exemplo, para $N = 49327$, os dígitos são 4, 9, 3, 2, 7. O maior dígito é 9 e o menor é 2. Este problema treina a capacidade de manter estado (maior e menor) durante a iteração pelos dígitos.

Algoritmização:

```
ALGORITMO EncontraMaiorMenorDigitoConciso

ENTRADA:
    N: inteiro

SAÍDA:
    maiorDigito, menorDigito: inteiro

INÍCIO
    LER N
    numeroOriginal ← N
    maiorDigito ← 0
    menorDigito ← 9

ETIQUETA PROCESSAR_DIGITOS:
    SE N = 0 ENTÃO IR PARA RESULTADO

    digito ← N % 10
    N ← N / 10

    SE digito > maiorDigito ENTÃO
        maiorDigito ← digito

    SE digito < menorDigito ENTÃO
        menorDigito ← digito

    IR PARA PROCESSAR_DIGITOS

ETIQUETA RESULTADO:
    ESCREVER "Número original:", numeroOriginal
    ESCREVER "Maior dígito:", maiorDigito
```

```
    ESCREVER "Menor dígito:", menorDigito
    ESCREVER "Diferença:", (maiorDigito - menorDigito)

    RETORNAR maiorDigito, menorDigito
FIM
```

O código Listing 7.1 apresenta o código em C++23 que implementa o algoritmo acima.

Listing 7.1

```cpp
#include <iostream>

int main() {
    int N;
    std::cout << "Digite um número: ";
    std::cin >> N;

    int numeroOriginal = N;
    int maiorDigito = 0;
    int menorDigito = 9;

    PROCESSAR_DIGITOS:
    if (N == 0) goto RESULTADO;

    int digito = N % 10;
    N = N / 10;

    if (digito > maiorDigito) {
        maiorDigito = digito;
    }

    if (digito < menorDigito) {
        menorDigito = digito;
    }

    goto PROCESSAR_DIGITOS;

    RESULTADO:
    std::cout << "Número original: " << numeroOriginal << std::endl;
    std::cout << "Maior dígito: " << maiorDigito << std::endl;
    std::cout << "Menor dígito: " << menorDigito << std::endl;
    std::cout << "Diferença: " << (maiorDigito - menorDigito) << std::endl;

    return 0;
}
```

Mas, como os alunos já devem estar familiarizados com as estruturas de repetição, o professor pode propor uma versão mais concisa do algoritmo, utilizando a abstração de função. O código Listing 7.2, apresenta o código em C++23 que implementa o algoritmo acima, mas com a abstração de função.

Revisão 2: no Laboratório de Engenharia, o professor Carlos precisa implementar uma

Listing 7.2

```cpp
#include <iostream>

int main() {
    auto N = 0;
    std::cout << "Digite um número: ";
    std::cin >> N;

    auto numeroOriginal = N;
    auto maiorDigito = 0;
    auto menorDigito = 9;

    while (N != 0) {
        auto digito = N % 10;
        N /= 10;

        if (digito > maiorDigito) maiorDigito = digito;
        if (digito < menorDigito) menorDigito = digito;
    }

    std::cout << "Número original: " << numeroOriginal << std::endl;
    std::cout << "Maior dígito: " << maiorDigito << std::endl;
    std::cout << "Menor dígito: " << menorDigito << std::endl;
    std::cout << "Diferença: " << (maiorDigito - menorDigito) << std::endl;

    return 0;
}
```

calculadora científica que calcule o seno de ângulos com alta precisão. Como as funções trigonométricas padrão podem ter limitações de precisão, ele decide usar a série de Taylor para sen(x). A série é definida como:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2n+1}}{(2n+1)!}$$

O sistema deve receber um `angulo` em radianos e a `precisao` desejada, número de termos da série, calcular o seno usando a série de Taylor e comparar com o resultado obtido na sua calculadora. Desenvolva um algoritmo que implemente este cálculo, mostrando cada termo da série e o erro absoluto em relação ao valor da biblioteca padrão.

Solução:

Neste caso, o pseudocódigo pode ser representado como:

Pseudocódigo:

```
ALGORITMO CalculadoraSeno

ENTRADA:
    precisao: inteiro

SAÍDA:
    senoCalculado: real
```

```
    senoReferencia: real
    erroAbsoluto: real

INÍCIO
    // Valor de teste pré-definido:  /6 (30°)
    angulo ← 0.523599
    senoReferencia ← 0.5

    ESCREVER "Calculando sen( /6) = sen(30°) = sen(0.523599 rad)"
    ESCREVER "Valor de referência: 0.5"
    ESCREVER "Digite o número de termos da série:"
    LER precisao

    senoCalculado ← 0
    termo ← angulo

    PARA i DE 0 ATÉ (precisao - 1) FAÇA
        SE (i % 2 == 0) ENTÃO
            senoCalculado ← senoCalculado + termo
            ESCREVER "Termo ", i+1, ": +", termo, " | Soma parcial: ", senoCalculado
        SENÃO
            senoCalculado ← senoCalculado - termo
            ESCREVER "Termo ", i+1, ": -", termo, " | Soma parcial: ", senoCalculado
        FIM SE

        // Calcular próximo termo: x^(2n+3) / (2n+3)!
        termo ← termo * angulo * angulo / ((2*i + 2) * (2*i + 3))
    FIM PARA

    erroAbsoluto ← ABS(senoCalculado - senoReferencia)

    ESCREVER "Ângulo: ", angulo, " radianos ( /6 ou 30°)"
    ESCREVER "Seno calculado: ", senoCalculado
    ESCREVER "Seno de referência: ", senoReferencia
    ESCREVER "Erro absoluto: ", erroAbsoluto

    RETORNAR senoCalculado, senoReferencia, erroAbsoluto
FIM
```

O código em C++23 que implementa o algoritmo acima, utilizando a biblioteca padrão C++ (STL) e ranges, pode ser representado visto na Listing 7.3:

O código da Listing 7.3 implementa a série de Taylor para calcular o seno de um ângulo em radianos, mostrando cada termo da série e o erro absoluto em relação ao valor de referência. O professor pode propor diferentes valores de `angulo` e `senoReferencia` para testar a precisão do cálculo a Table 7.1 contém algumas sugestões.

Table 7.1: Valores de teste para diferentes ângulos na série de Taylor do seno

Teste	Ângulo	Graus	Radianos	Seno de Referência	Código
1	/4	45°	0.785398	0.707107	`auto angulo = 0.785398; auto senoReferencia = 0.707107;`
2	/3	60°	1.047198	0.866025	`auto angulo = 1.047198; auto senoReferencia = 0.866025;`
3	/2	90°	1.570796	1.0	`auto angulo = 1.570796; auto senoReferencia = 1.0;`

Revisão 3. na empresa de software "Algoritmos Precisos", a matemática Dra. Marina está desenvolvendo um módulo para calcular o valor de pi com diferentes níveis de precisão. Ela escolheu usar a série de Leibniz, também conhecida como série de Gregory-Leibniz e dada por:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

O sistema deve receber o `numero_termos` desejado, calcular pi usando a série, mostrar a convergência a cada 1000 iterações e comparar com o valor de pi digitado manualmente. Além disso, deve calcular quantos termos são necessários para atingir uma precisão de pelo menos 6 casas decimais. Crie um algoritmo que implemente este cálculo de pi.

Solução:

Este exercício envolve a aproximação do fatorial usando a fórmula de Stirling e uma série de correção. O pseudocódigo para este exercício pode ser representado como:

```
ALGORITMO CalculadoraPi

ENTRADA:
    numeroTermos: inteiro

SAÍDA:
    piCalculado: real
    piReferencia: real
    termosParaPrecisao: inteiro

INÍCIO
```

```
    ESCREVER "Digite o número de termos:"
    LER numeroTermos

    somaLeibniz ← 0
    precisaoAtingida ← FALSO
    termosParaPrecisao ← 0
    piReferencia ← 3.141592653589793

    PARA i DE 0 ATÉ (numeroTermos - 1) FAÇA
        SE (i % 2 == 0) ENTÃO
            somaLeibniz ← somaLeibniz + (1.0 / (2*i + 1))
        SENÃO
            somaLeibniz ← somaLeibniz - (1.0 / (2*i + 1))
        FIM SE

        piCalculado ← 4 * somaLeibniz

        // Mostrar progresso a cada 1000 termos
        SE ((i + 1) % 1000 == 0) ENTÃO
            ESCREVER "Termos: ", i+1, " | Pi aproximado: ", piCalculado
        FIM SE

        // Verificar precisão de 6 casas decimais
        SE (ABS(piCalculado - piReferencia) < 0.000001) E (precisaoAtingida == FALSO) ENT
            termosParaPrecisao ← i + 1
            precisaoAtingida ← VERDADEIRO
        FIM SE
    FIM PARA

    ESCREVER "Pi calculado: ", piCalculado
    ESCREVER "Pi de referência: ", piReferencia
    ESCREVER "Erro absoluto: ", ABS(piCalculado - piReferencia)
    SE (precisaoAtingida) ENTÃO
        ESCREVER "Precisão de 6 casas atingida com ", termosParaPrecisao, " termos"
    SENÃO
        ESCREVER "Precisão de 6 casas não atingida com ", numeroTermos, " termos"
    FIM SE

    RETORNAR piCalculado, termosParaPrecisao
FIM
```

Neste caso, o código Listing 7.4, em C++23 mostra como pseudocódigo pode ser representado com C++23, utilizando a biblioteca padrão C++ (STL) e ranges. O uso de `std::views::iota` permite iterar facilmente sobre uma sequência de números, e o uso de `auto` simplifica a dedução de tipos, fazendo com que o código da Listing 7.4 seja mais conciso e legível.

Revisão 4: no Centro de Pesquisa em Matemática Computacional, o Dr. Roberto está estudando diferentes métodos para aproximar fatoriais de números grandes. Ele decidiu implementar uma versão simplificada da fórmula de Stirling combinada com uma série de correção. A fórmula de Stirling é uma aproximação do fatorial de um número grande n e é dada por:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \cdot \left(1 + \frac{1}{12n} + \frac{1}{288n^2}\right)$$

Que pode ser expressa usando somatório e produtório como:

$$\text{Aproximação} = \sqrt{2\pi n} \cdot \prod_{i=1}^{n} \frac{i}{e} \cdot \sum_{k=0}^{2} \frac{c_k}{n^k}$$

Na qual $c_0 = 1$, $c_1 = \frac{1}{12}$, $c_2 = \frac{1}{288}$.

O sistema deve receber um número `n`, calcular o fatorial exato (para $n \leq 20$), calcular a aproximação usando a fórmula acima, mostrar os valores do produtório e somatório separadamente, e calcular o erro percentual entre o valor exato e a aproximação. Como a fórmula de Stirling requer o cálculo de $\sqrt{2\pi n}$, o algoritmo deve implementar o método babilônico para calcular raiz quadrada sem depender de bibliotecas matemáticas externas. O método babilônico utiliza a fórmula iterativa $x_{novo} = \frac{1}{2}(x + \frac{numero}{x})$, repetindo até atingir a precisão desejada. Desenvolva um algoritmo que implemente este sistema de aproximação fatorial, incluindo uma função para calcular a raiz quadrada usando o método babilônico, uma função para calcular o valor absoluto sem usar a biblioteca `cmath`, e utilizando os valores pré-definidos $\pi = 3.141592653589793$ e $e = 2.718281828459045$.

Solução:

```
ALGORITMO AproximacaoFatorial

ENTRADA:
    n: inteiro

SAÍDA:
    fatorialExato: inteiro
    aproximacaoStirling: real
    erroPercentual: real

INÍCIO
    ESCREVER "Digite o valor de n (  20):"
    LER n

    // Calcular fatorial exato
    fatorialExato ← 1
    PARA i DE 1 ATÉ n FAÇA
        fatorialExato ← fatorialExato * i
    FIM PARA

    // Calcular produtório:  (i/e) de i=1 até n
    produtorio ← 1.0
    e ← 2.718281828459045
    PARA i DE 1 ATÉ n FAÇA
        produtorio ← produtorio * (i / e)
    FIM PARA
    ESCREVER "Produtório  (i/e): ", produtorio

    // Calcular somatório: Σ(ck/n^k) de k=0 até 2
    c0 ← 1.0
    c1 ← 1.0 / 12.0
```

```
    c2 ← 1.0 / 288.0

    somatorio ← c0 + (c1 / n) + (c2 / (n * n))
    ESCREVER "Somatório (1 + 1/12n + 1/288n²): ", somatorio

    // Calcular raiz quadrada usando método babilônico
    valor ← 2 * pi * n
    raizParte ← RAIZ_BABILONICA(valor)

    // Calcular aproximação completa
    pi ← 3.141592653589793
    aproximacaoStirling ← raizParte * produtorio * somatorio

    // Calcular erro percentual
    erroPercentual ← VALOR_ABSOLUTO(aproximacaoStirling - fatorialExato) / fatorialExato

    ESCREVER "Fatorial exato de ", n, "!: ", fatorialExato
    ESCREVER "Aproximação de Stirling: ", aproximacaoStirling
    ESCREVER "√(2 n): ", raizParte
    ESCREVER "Erro percentual: ", erroPercentual, "%"

    RETORNAR fatorialExato, aproximacaoStirling, erroPercentual
FIM

FUNÇÃO RAIZ_BABILONICA(numero: real) → real
    SE numero <= 0 ENTÃO RETORNAR 0
    x ← numero
    REPETIR
        raiz ← 0.5 * (x + numero / x)
        SE VALOR_ABSOLUTO(raiz - x) < 0.000001 ENTÃO
            RETORNAR raiz
        x ← raiz
    ATÉ FALSO
FIM FUNÇÃO

FUNÇÃO VALOR_ABSOLUTO(valor: real) → real
    SE valor < 0 ENTÃO
        RETORNAR -valor
    SENÃO
        RETORNAR valor
    FIM SE
FIM FUNÇÃO
```

Considerando o pseudo código acima, o código Listing 7.5 permite a implementação do algoritmo em C++23:

Para testar o código Listing 7.5, o professor pode usar os seguintes valores de entrada para $n = 5$:

- Fatorial exato: $5! = 120$
- Aproximação de Stirling esperada: 118.02
- Erro percentual esperado: 1.65%

7.1.1 Reconhecimento de Padrões: Função

O reconhecimento de padrões é uma habilidade essencial no raciocínio algorítmico, permitindo identificar soluções comuns para problemas recorrentes. Foi o reconhecimento de padrões que levou à criação de funções, que são blocos reutilizáveis de código que encapsulam uma lógica específica. Neste módulo, os alunos devem aprender a identificar padrões em problemas antes de criar funções para resolver esses problemas. O objetivo é que a metáfora de "função" seja vista como uma abstração que permite resolver problemas complexos de forma modular e reutilizável. Além disso, é importante que os alunos entendam que a função é um desvio de fluxo que permite encapsular uma lógica específica, tornando o código mais legível e organizado. Mas que implica em custo computacional extra.

Os exercícios a seguir podem ser aplicados segundo a Técnica da Sequência de Fibonacci, na qual o professor deve escolher o número de problemas que serão resolvidos em cada etapa de acordo com o tamanho e o resultado da turma, mas deve se manter na Técnica da Sequência de Fibonacci. A Técnica da Sequência de Fibonacci foi abordada na Section 6.1.1 nesta técnica além de tempos e quantidades, o professor precisa se preocupar que a devolutiva deve ser feita no quadro, com o professor explicando cada passo do algoritmo, enfatizando a importância da decomposição e abstração. A seguir está uma lista de problemas que podem ser usados para esta atividade, começando com a repetição de um exercício anterior, apenas revisão:

A. Um canhão dispara projéteis seguindo a equação de altura $h(t) = -4.9t^2 + v_0 \sin(\theta) \cdot t + h_0$, na qual t é o tempo em segundos, v_0 é a velocidade inicial, θ é o ângulo de lançamento e h_0 é a altura inicial. Você deverá determinar quando o projétil atinge o solo a Atendendo as seguintes especificações.

Entrada: v_0 (velocidade inicial em m/s), θ (ângulo em graus), h_0 (altura inicial em metros)

Tarefa: Calcule quando o projétil atinge o solo (h = 0). Se existirem duas soluções positivas, mostre a menor (primeiro impacto). Se não houver soluções reais positivas, informe que o projétil não atinge o solo.

Saída: Tempo de impacto ou mensagem de erro.

Além desta especificações, você deve considerar que o seno de um ângulo pode ser aproximado usando a série de Taylor.

A série de Taylor para calcular $\sin(x)$ é dada por:

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2n+1}}{(2n+1)!}$$

Expandindo os primeiros termos:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \ldots$$

Nesta expressão:

- x deve estar em radianos;
- $n!$ representa o fatorial de n;
- A série alterna entre sinais positivos e negativos;
- Para precisão razoável, geralmente 7 termos são suficientes.

Análise: o pseudocódigo para este exercício pode ser representado como:

```
ALGORITMO TrajetoriaProjetil

ENTRADA:
    v0, theta, h0: real
```

```
SAÍDA:
    tempo: real ou mensagem

INÍCIO
    LER v0, theta, h0

    // Converter ângulo para radianos
    pi ← 3.14159265
    radianos ← theta * pi
    radianos ← radianos / 180

    // Calcular seno usando aproximação por série de Taylor (5 termos)
    seno ← radianos
    termo ← radianos
    i ← 1
    ENQUANTO i <= 4 FAÇA
        termo ← termo * radianos
        termo ← termo * radianos
        temp ← 2 * i
        temp ← temp + 1
        temp2 ← temp - 1
        temp3 ← temp2 * temp
        termo ← termo / temp3
        SE i % 2 = 0 ENTÃO
            seno ← seno + termo
        SENÃO
            seno ← seno - termo
        FIM SE
        i ← i + 1
    FIM ENQUANTO

    // Coeficientes da equação quadrática: at² + bt + c = 0
    a ← -4.9
    b ← v0 * seno
    c ← h0

    // Calcular discriminante
    temp ← b * b
    temp2 ← 4 * a
    temp3 ← temp2 * c
    discriminante ← temp - temp3

    SE discriminante < 0 ENTÃO
        ESCREVER "O projétil não atinge o solo"
    SENÃO
        // Calcular raiz quadrada do discriminante usando método de Newton
        raizQuadrada ← discriminante / 2
        ENQUANTO abs(raizQuadrada * raizQuadrada - discriminante) > 0.0001 FAÇA
            temp ← raizQuadrada * raizQuadrada
            temp ← temp + discriminante
```

```
            raizQuadrada ← temp / (2 * raizQuadrada)
        FIM ENQUANTO

        // Calcular as duas raízes
        temp ← 0 - b
        temp2 ← 2 * a
        raiz1 ← temp + raizQuadrada
        raiz1 ← raiz1 / temp2
        raiz2 ← temp - raizQuadrada
        raiz2 ← raiz2 / temp2

        // Verificar soluções positivas
        SE raiz1 > 0 E raiz2 > 0 ENTÃO
            SE raiz1 < raiz2 ENTÃO
                tempo ← raiz1
            SENÃO
                tempo ← raiz2
            FIM SE
            ESCREVER "Tempo de impacto:", tempo, "segundos"
        SENÃO SE raiz1 > 0 ENTÃO
            ESCREVER "Tempo de impacto:", raiz1, "segundos"
        SENÃO SE raiz2 > 0 ENTÃO
            ESCREVER "Tempo de impacto:", raiz2, "segundos"
        SENÃO
            ESCREVER "O projétil não atinge o solo"
        FIM SE
    FIM SE
FIM
```

Nós estamos tentando criar a cognição necessária para entender o uso de funções. Assim, o professor pode propor o uso de um fluxograma para melhorar a visualização do algoritmo apresentado no pseudocódigo acima. Para tanto, o professor pode usar o fluxograma apresentado na Figure 7.1.

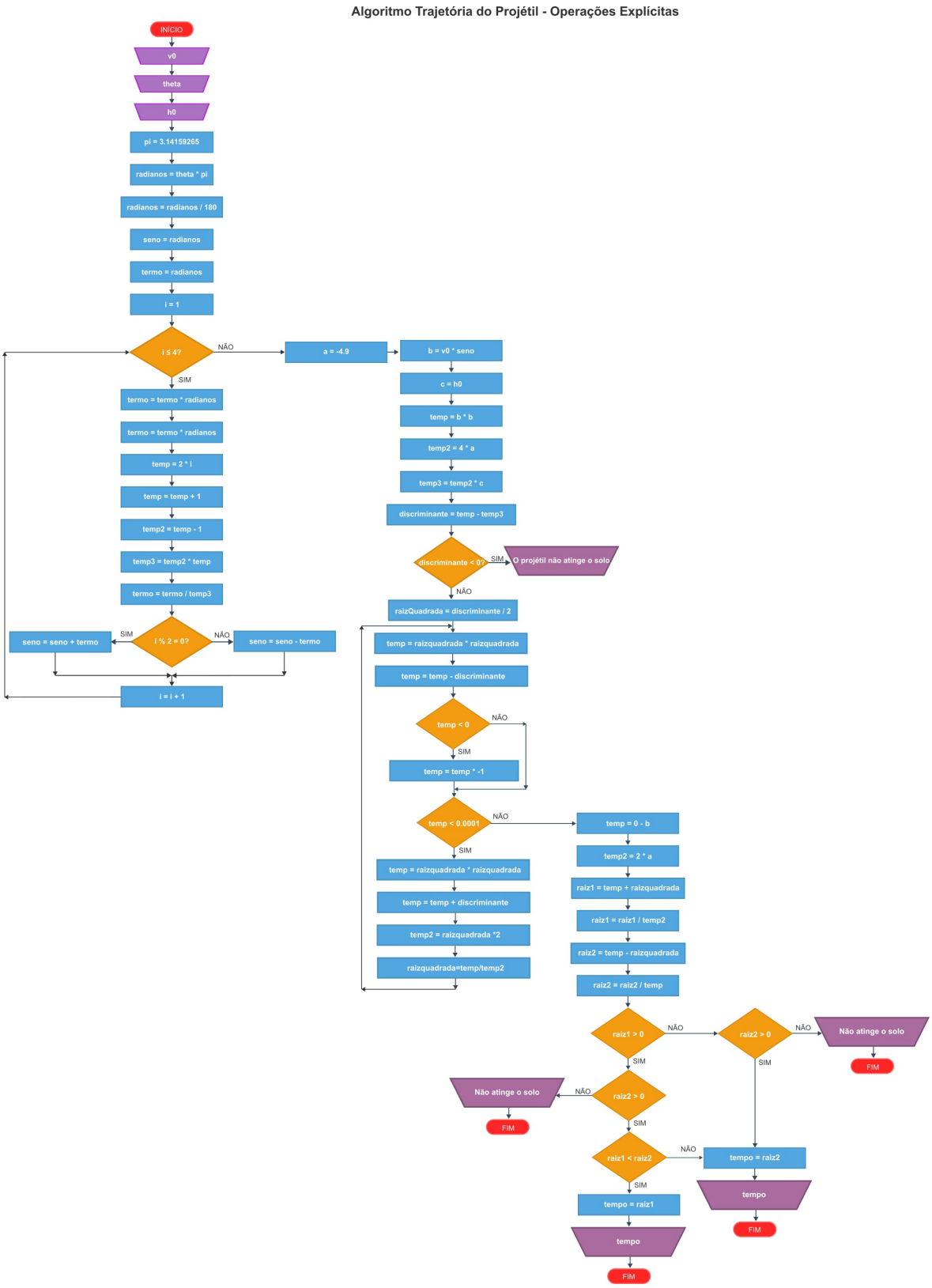


Figure 7.1: Fluxograma do Algoritmo de Trajetória de Projétil usando apenas operações binárias e unárias.

O fluxograma da Figure 7.1 mostra as operações binárias e unárias necessárias para resolver o problema da trajetória do projétil detalhando cada processo necessário. Esta estrutura pode criar dúvidas e confusão. Contudo, é preciso lembrar que neste momento, estamos

trabalhando para criar a abstração necessária ao entendimento do conceito de função. E não usando funções.

O professor pode usar este fluxograma para destacar a necessidade do uso de abstrações mais eficientes. Por exemplo, podemos criar um pseudocódigo, ou conjunto de módulos, só para calcular o quadrado de um valor e dividir este quadrado por outro valor, em vez de um bloco para multiplicar um valor por si mesmo e então dividir este valor por outro, como é feito na Figure 7.1.

Usando a Figure 7.1, será importante destacar as áreas que podem ser transformadas em funções, como o cálculo do seno usando a série de Taylor, que pode ser encapsulado em uma função separada.

Finalmente, neste ponto, fica claro que o fluxograma não atende todas as abstrações que precisamos construir para usar linguagens imperativas de forma estruturada. Deste ponto, em diante, vamos dar prioridade ao pseudocódigo.

B. Um fazendeiro possui 120 metros de cerca para delimitar um terreno retangular. Uma das dimensões será x metros, resultando numa área $A = x(60 - x)$ metros quadrados. Determine as dimensões possíveis do terreno para atingir uma determinada área. Sabendo que a equação que representa este fenômeno é $x^2 - 60x + \text{área} = 0$. Considere as seguintes especificações:

Entrada: Área desejada em metros quadrados

Saída: As duas dimensões possíveis ou mensagem informando que a área é impossível de atingir.

Solução:

Já usando funções no pseudocódigo, podemos representar o problema da seguinte forma:

```
ALGORITMO DimensoesTerreno

ENTRADA:
    area_desejada: real

SAÍDA:
    dimensoes: real, real ou mensagem

INÍCIO
    LER area_desejada

    // CONFIGURAR COEFICIENTES DA EQUAÇÃO QUADRÁTICA
    // Equação: x² - 60x + area_desejada = 0
    a ← 1
    b ← -60
    c ← area_desejada

    // CALCULAR DISCRIMINANTE
    temp1 ← b * b
    temp2 ← 4 * a
    temp3 ← temp2 * c
    discriminante ← temp1 - temp3

    // VERIFICAR VIABILIDADE DA SOLUÇÃO
    SE discriminante < 0 ENTÃO
        ESCREVER "Área impossível de atingir com 120 metros de cerca"
    SENÃO
```

```
        // CALCULAR RAIZ QUADRADA (MÉTODO DE NEWTON)
        raizQuadrada ← calcularRaizQuadrada(discriminante)

        // CALCULAR AS RAÍZES DA EQUAÇÃO QUADRÁTICA
        raiz1, raiz2 ← calcularRaizes(a, b, raizQuadrada)

        // CALCULAR AS DIMENSÕES DO TERRENO
        // Primeira dimensão: x = raiz1
        dimensao1_x ← raiz1
        temp_sub1 ← 60 - raiz1
        dimensao1_y ← temp_sub1

        // Segunda dimensão: x = raiz2
        dimensao2_x ← raiz2
        temp_sub2 ← 60 - raiz2
        dimensao2_y ← temp_sub2

        // VALIDAR DIMENSÕES FÍSICAS
        valida1 ← FALSO
        valida2 ← FALSO

        // Verificar primeira solução
        SE dimensao1_x > 0 E dimensao1_x < 60 ENTÃO
            SE dimensao1_y > 0 E dimensao1_y < 60 ENTÃO
                valida1 ← VERDADEIRO
            FIM SE
        FIM SE

        // Verificar segunda solução
        SE dimensao2_x > 0 E dimensao2_x < 60 ENTÃO
            SE dimensao2_y > 0 E dimensao2_y < 60 ENTÃO
                valida2 ← VERDADEIRO
            FIM SE
        FIM SE

        // APRESENTAR RESULTADOS
        SE valida1 = VERDADEIRO E valida2 = VERDADEIRO ENTÃO
            ESCREVER "Primeira solução:"
            ESCREVER "Dimensões:", dimensao1_x, "metros x", dimensao1_y, "metros"
            ESCREVER "Segunda solução:"
            ESCREVER "Dimensões:", dimensao2_x, "metros x", dimensao2_y, "metros"
        SENÃO SE valida1 = VERDADEIRO ENTÃO
            ESCREVER "Dimensões possíveis:"
            ESCREVER dimensao1_x, "metros x", dimensao1_y, "metros"
        SENÃO SE valida2 = VERDADEIRO ENTÃO
            ESCREVER "Dimensões possíveis:"
            ESCREVER dimensao2_x, "metros x", dimensao2_y, "metros"
        SENÃO
            ESCREVER "Área impossível de atingir com as limitações físicas"
        FIM SE
    FIM SE
```

```
FIM

// FUNÇÃO AUXILIAR: CALCULAR RAIZ QUADRADA (MÉTODO DE NEWTON)
FUNÇÃO calcularRaizQuadrada(discriminante: real): real
INÍCIO
    raizQuadrada ← discriminante / 2

    ENQUANTO abs(raizQuadrada * raizQuadrada - discriminante) > 0.0001 FAÇA
        // Calcular raizQuadrada²
        temp_quad ← raizQuadrada * raizQuadrada

        // Calcular numerador: raizQuadrada² + discriminante
        numerador ← temp_quad + discriminante

        // Calcular denominador: 2 * raizQuadrada
        denominador ← 2 * raizQuadrada

        // Nova aproximação
        raizQuadrada ← numerador / denominador
    FIM ENQUANTO

    RETORNAR raizQuadrada
FIM

// FUNÇÃO AUXILIAR: CALCULAR RAÍZES DA EQUAÇÃO QUADRÁTICA
FUNÇÃO calcularRaizes(a: real, b: real, raizQuadrada: real): real, real
INÍCIO
    // Calcular -b
    menos_b ← 0 - b

    // Calcular denominador comum: 2a
    denominador_comum ← 2 * a

    // Calcular raiz1: (-b + √Δ) / 2a
    numerador1 ← menos_b + raizQuadrada
    raiz1 ← numerador1 / denominador_comum

    // Calcular raiz2: (-b - √Δ) / 2a
    numerador2 ← menos_b - raizQuadrada
    raiz2 ← numerador2 / denominador_comum

    RETORNAR raiz1, raiz2
FIM

// FUNÇÃO AUXILIAR: VALOR ABSOLUTO
FUNÇÃO abs(valor: real): real
INÍCIO
    SE valor < 0 ENTÃO
        resultado ← 0 - valor
    SENÃO
        resultado ← valor
```

```
    FIM SE
    RETORNAR resultado
FIM
```

Este pseudocódigo serve como ferramenta de abstração para o professor, que pode usar este algoritmo para explicar a decomposição do problema em funções menores. O professor pode destacar como cada função tem uma responsabilidade específica, como calcular a raiz quadrada, calcular as raízes da equação quadrática e validar as dimensões físicas. Além disso, o professor pode enfatizar a importância de validar as entradas e saídas de cada função, garantindo que o código seja robusto e fácil de entender. Por fim, as funções auxiliares foram colocadas no final do pseudocódigo para destacar que elas são abstrações que podem ser reutilizadas em outros contextos, reforçando a ideia de modularidade, reutilização de código e de controle de fluxo. Em C++23, o pseudocódigo está implementado no site GDB Online no link https://onlinegdb.com/Si-IeJICB.

C. Um veículo em movimento tem sua posição descrita por $s(t) = s_0 + v_0 t + \frac{1}{2}at^2$, na qual s_0 é a posição inicial, v_0 é a velocidade inicial e a é a aceleração. Sua tarefa é criar um algoritmo que calcule em que instante(s) o veículo passa por uma posição final s_f. Para tanto, considere as seguintes especificações:

Entrada: s_0 (posição inicial em metros), v_0 (velocidade inicial em m/s), a (aceleração em m/s^2), s_f (posição final desejada em metros)

Tarefa: Calcule em que instante(s) o veículo passa pela posição s_f. A equação é $\frac{1}{2}at^2 + v_0 t + (s_0 - s_f) = 0$.

Saída: Tempo(s) em que o veículo passa pela posição ou mensagem de que nunca passa.

Solução:

Neste caso, reusando as abstrações criadas para funções do exercício anterior, teremos:

```
ALGORITMO PosicaoVeiculo

ENTRADA:
    s0, v0, a, sf: real

SAÍDA:
    tempo(s): real ou mensagem

INÍCIO
    LER s0, v0, a, sf

    // VERIFICAR TIPO DE MOVIMENTO

    SE a = 0 ENTÃO
        // Movimento uniforme: s = s0 + v0*t
        // sf = s0 + v0*t  =>  t = (sf - s0)/v0

        SE v0 = 0 ENTÃO
            SE s0 = sf ENTÃO
                ESCREVER "O veículo está sempre na posição desejada"
            SENÃO
                ESCREVER "O veículo nunca passa pela posição desejada"
            FIM SE
        SENÃO
            // Calcular t = (sf - s0) / v0
```

```
            diferenca ← sf - s0
            tempo ← diferenca / v0
            ESCREVER "O veículo passa pela posição no tempo:", tempo, "segundos"
        FIM SE
    SENÃO

        // CONFIGURAR COEFICIENTES DA EQUAÇÃO QUADRÁTICA
        // Equação: (1/2)at² + v0t + (s0 - sf) = 0
        // Multiplicando por 2: at² + 2v0t + 2(s0 - sf) = 0
        coef_a ← a
        temp_2v0 ← 2 * v0
        coef_b ← temp_2v0
        temp_diff ← s0 - sf
        temp_2diff ← 2 * temp_diff
        coef_c ← temp_2diff


        // CALCULAR DISCRIMINANTE
        temp1 ← coef_b * coef_b
        temp2 ← 4 * coef_a
        temp3 ← temp2 * coef_c
        discriminante ← temp1 - temp3


        // VERIFICAR VIABILIDADE DA SOLUÇÃO
        SE discriminante < 0 ENTÃO
            ESCREVER "O veículo nunca passa pela posição desejada"
        SENÃO

            // CALCULAR RAIZ QUADRADA (MÉTODO DE NEWTON)
            raizQuadrada ← calcularRaizQuadrada(discriminante)

            // CALCULAR AS RAÍZES DA EQUAÇÃO QUADRÁTICA
           tempo1, tempo2 ← calcularRaizes(coef_a, coef_b, raizQuadrada)

            // VALIDAR E APRESENTAR TEMPOS
            tempos_validos ← 0

            // Verificar primeiro tempo
            SE tempo1 >= 0 ENTÃO
                tempos_validos ← tempos_validos + 1
                primeiro_tempo_valido ← tempo1
            FIM SE

            // Verificar segundo tempo
            SE tempo2 >= 0 ENTÃO
                tempos_validos ← tempos_validos + 1
                SE tempos_validos = 1 ENTÃO
                    primeiro_tempo_valido ← tempo2
                SENÃO
                    segundo_tempo_valido ← tempo2
```

```
            FIM SE
        FIM SE


        // APRESENTAR RESULTADOS
        SE tempos_validos = 0 ENTÃO
            ESCREVER "O veículo passou pela posição apenas no passado:"
            ESCREVER "Tempo 1:", tempo1, "segundos"
            ESCREVER "Tempo 2:", tempo2, "segundos"
        SENÃO SE tempos_validos = 1 ENTÃO
            ESCREVER "O veículo passa pela posição no tempo:"
            ESCREVER primeiro_tempo_valido, "segundos"

            // Mostrar também tempo negativo se existir
            SE tempo1 < 0 ENTÃO
                ESCREVER "Também passou no passado em:", tempo1, "segundos"
            FIM SE
            SE tempo2 < 0 ENTÃO
                ESCREVER "Também passou no passado em:", tempo2, "segundos"
            FIM SE
        SENÃO
            // Ordenar os tempos (menor primeiro)
            SE primeiro_tempo_valido > segundo_tempo_valido ENTÃO
                temp_swap ← primeiro_tempo_valido
                primeiro_tempo_valido ← segundo_tempo_valido
                segundo_tempo_valido ← temp_swap
            FIM SE

            ESCREVER "O veículo passa pela posição em dois momentos:"
            ESCREVER "Primeiro tempo:", primeiro_tempo_valido, "segundos"
            ESCREVER "Segundo tempo:", segundo_tempo_valido, "segundos"
        FIM SE
    FIM SE
FIM SE
FIM


// FUNÇÃO AUXILIAR: CALCULAR RAIZ QUADRADA (MÉTODO DE NEWTON)
FUNÇÃO calcularRaizQuadrada(discriminante: real): real
INÍCIO
    raizQuadrada ← discriminante / 2

    ENQUANTO abs(raizQuadrada * raizQuadrada - discriminante) > 0.0001 FAÇA
        // Calcular raizQuadrada²
        temp_quad ← raizQuadrada * raizQuadrada

        // Calcular numerador: raizQuadrada² + discriminante
        numerador ← temp_quad + discriminante

        // Calcular denominador: 2 * raizQuadrada
        denominador ← 2 * raizQuadrada
```

```
        // Nova aproximação
        raizQuadrada ← numerador / denominador
    FIM ENQUANTO

    RETORNAR raizQuadrada
FIM


// FUNÇÃO AUXILIAR: CALCULAR RAÍZES DA EQUAÇÃO QUADRÁTICA
FUNÇÃO calcularRaizes(a: real, b: real, raizQuadrada: real): real, real
INÍCIO
    // Calcular -b
    menos_b ← 0 - b

    // Calcular denominador comum: 2a
    denominador_comum ← 2 * a

    // Calcular raiz1: (-b + √Δ) / 2a
    numerador1 ← menos_b + raizQuadrada
    raiz1 ← numerador1 / denominador_comum

    // Calcular raiz2: (-b - √Δ) / 2a
    numerador2 ← menos_b - raizQuadrada
    raiz2 ← numerador2 / denominador_comum

    RETORNAR raiz1, raiz2
FIM

// FUNÇÃO AUXILIAR: VALOR ABSOLUTO
FUNÇÃO abs(valor: real): real
INÍCIO
    SE valor < 0 ENTÃO
        resultado ← 0 - valor
    SENÃO
        resultado ← valor
    FIM SE
    RETORNAR resultado
FIM
```

O código em C++23 para o pseudocódigo acima pode ser encontrado no site GDB Online no link https://onlinegdb.com/2ZaBx62x_.

D. Uma empresa possui função de lucro $L(q) = -2q^2 + 80q - 300$, onde q é a quantidade produzida e L é o lucro em reais. Você deverá criar um programa que determine a quantidade de produção necessária para atingir o lucro desejado. A equação é $-2q^2 + 80q - (300 +$ lucro desejado$) = 0$. Considerando as seguintes especificações:

Entrada: Lucro desejado em reais

Saída: Quantidade(s) de produção ou mensagem de que o lucro é inatingível.

Solução:

Neste caso, teremos:

```
ALGORITMO LucroEmpresa

ENTRADA:
    lucro_desejado: real

SAÍDA:
    quantidade(s): real ou mensagem

INÍCIO
    LER lucro_desejado

    // CONFIGURAR COEFICIENTES DA EQUAÇÃO QUADRÁTICA
    // Equação: -2q² + 80q - (300 + lucro_desejado) = 0
    a ← -2
    b ← 80
    temp_soma ← 300 + lucro_desejado
    temp_negativo ← 0 - temp_soma
    c ← temp_negativo

    // CALCULAR DISCRIMINANTE
    temp1 ← b * b
    temp2 ← 4 * a
    temp3 ← temp2 * c
    discriminante ← temp1 - temp3

    // VERIFICAR VIABILIDADE DA SOLUÇÃO
    SE discriminante < 0 ENTÃO
        ESCREVER "Lucro inatingível com a função de produção atual"
    SENÃO

        // CALCULAR RAIZ QUADRADA (MÉTODO DE NEWTON)
        raizQuadrada ← calcularRaizQuadrada(discriminante)

        // CALCULAR AS RAÍZES DA EQUAÇÃO QUADRÁTICA
        quantidade1, quantidade2 ← calcularRaizes(a, b, raizQuadrada)

        // VALIDAR QUANTIDADES FÍSICAS
        valida1 ← FALSO
        valida2 ← FALSO

        // Verificar primeira quantidade
        SE quantidade1 > 0 ENTÃO
            valida1 ← VERDADEIRO
        FIM SE

        // Verificar segunda quantidade
        SE quantidade2 > 0 ENTÃO
            valida2 ← VERDADEIRO
        FIM SE

        // CALCULAR LUCROS PARA VERIFICAÇÃO
```

```
SE valida1 = VERDADEIRO ENTÃO
    // Calcular L(q1) = -2q1² + 80q1 - 300
    temp_q1_quadrado ← quantidade1 * quantidade1
    temp_menos2q1_quadrado ← -2 * temp_q1_quadrado
    temp_80q1 ← 80 * quantidade1
    temp_soma_termos ← temp_menos2q1_quadrado + temp_80q1
    lucro_verificacao1 ← temp_soma_termos - 300
FIM SE

SE valida2 = VERDADEIRO ENTÃO
    // Calcular L(q2) = -2q2² + 80q2 - 300
    temp_q2_quadrado ← quantidade2 * quantidade2
    temp_menos2q2_quadrado ← -2 * temp_q2_quadrado
    temp_80q2 ← 80 * quantidade2
    temp_soma_termos2 ← temp_menos2q2_quadrado + temp_80q2
    lucro_verificacao2 ← temp_soma_termos2 - 300
FIM SE

// APRESENTAR RESULTADOS
SE valida1 = VERDADEIRO E valida2 = VERDADEIRO ENTÃO
    // Ordenar quantidades (menor primeiro)
    SE quantidade1 > quantidade2 ENTÃO
        temp_swap ← quantidade1
        quantidade1 ← quantidade2
        quantidade2 ← temp_swap
    FIM SE

    ESCREVER "Duas quantidades de produção possíveis:"
    ESCREVER "Primeira quantidade:", quantidade1, "unidades"
    ESCREVER "Segunda quantidade:", quantidade2, "unidades"
    ESCREVER "Ambas geram o lucro de:", lucro_desejado, "reais"

SENÃO SE valida1 = VERDADEIRO ENTÃO
    ESCREVER "Quantidade de produção necessária:"
    ESCREVER quantidade1, "unidades"
    ESCREVER "Lucro gerado:", lucro_verificacao1, "reais"

    // Mostrar também quantidade negativa se existir
    SE quantidade2 < 0 ENTÃO
        ESCREVER "Quantidade negativa descartada:", quantidade2
    FIM SE

SENÃO SE valida2 = VERDADEIRO ENTÃO
    ESCREVER "Quantidade de produção necessária:"
    ESCREVER quantidade2, "unidades"
    ESCREVER "Lucro gerado:", lucro_verificacao2, "reais"

    // Mostrar também quantidade negativa se existir
    SE quantidade1 < 0 ENTÃO
        ESCREVER "Quantidade negativa descartada:", quantidade1
    FIM SE
```

```
        SENÃO
            ESCREVER "Todas as soluções resultam em quantidades negativas:"
            ESCREVER "Quantidade 1:", quantidade1, "unidades"
            ESCREVER "Quantidade 2:", quantidade2, "unidades"
            ESCREVER "Não é possível produzir quantidades negativas"
        FIM SE
    FIM SE

    // INFORMAÇÕES ADICIONAIS SOBRE A EMPRESA
    ESCREVER "Informações da função de lucro L(q) = -2q² + 80q - 300:"

    // Calcular lucro máximo (vértice da parábola)
    // q_max = -b/(2a) = -80/(2*(-2)) = 80/4 = 20
    temp_2a ← 2 * a
    temp_menos_b ← 0 - b
    q_maximo ← temp_menos_b / temp_2a

    // Calcular lucro máximo L(20)
    temp_qmax_quadrado ← q_maximo * q_maximo
    temp_menos2qmax_quadrado ← -2 * temp_qmax_quadrado
    temp_80qmax ← 80 * q_maximo
    temp_soma_max ← temp_menos2qmax_quadrado + temp_80qmax
    lucro_maximo ← temp_soma_max - 300

    ESCREVER "Quantidade para lucro máximo:", q_maximo, "unidades"
    ESCREVER "Lucro máximo possível:", lucro_maximo, "reais"
FIM

// FUNÇÃO AUXILIAR: CALCULAR RAIZ QUADRADA (MÉTODO DE NEWTON)
FUNÇÃO calcularRaizQuadrada(discriminante: real): real
INÍCIO
    raizQuadrada ← discriminante / 2

    ENQUANTO abs(raizQuadrada * raizQuadrada - discriminante) > 0.0001 FAÇA
        // Calcular raizQuadrada²
        temp_quad ← raizQuadrada * raizQuadrada

        // Calcular numerador: raizQuadrada² + discriminante
        numerador ← temp_quad + discriminante

        // Calcular denominador: 2 * raizQuadrada
        denominador ← 2 * raizQuadrada

        // Nova aproximação
        raizQuadrada ← numerador / denominador
    FIM ENQUANTO

    RETORNAR raizQuadrada
FIM
```

```
// FUNÇÃO AUXILIAR: CALCULAR RAÍZES DA EQUAÇÃO QUADRÁTICA
FUNÇÃO calcularRaizes(a: real, b: real, raizQuadrada: real): real, real
INÍCIO
    // Calcular -b
    menos_b ← 0 - b

    // Calcular denominador comum: 2a
    denominador_comum ← 2 * a

    // Calcular raiz1: (-b + √Δ) / 2a
    numerador1 ← menos_b + raizQuadrada
    raiz1 ← numerador1 / denominador_comum

    // Calcular raiz2: (-b - √Δ) / 2a
    numerador2 ← menos_b - raizQuadrada
    raiz2 ← numerador2 / denominador_comum

    RETORNAR raiz1, raiz2
FIM

// FUNÇÃO AUXILIAR: VALOR ABSOLUTO
FUNÇÃO abs(valor: real): real
INÍCIO
    SE valor < 0 ENTÃO
        resultado ← 0 - valor
    SENÃO
        resultado ← valor
    FIM SE
    RETORNAR resultado
FIM
```

O código em C++23 para o pseudocódigo acima pode ser encontrado no site GDB Online no link https://onlinegdb.com/flCt2mss0.

E. O ganho de uma antena parabólica em função do diâmetro segue a relação $G = -0.1d^2 + 3d + 5$, onde d é o diâmetro em metros e G é o ganho em dB. Você precisa criar um algoritmo que calcule o diâmetro de uma antena que forneça o ganho desejado. A equação é $-0.1d^2 + 3d + (5 - \text{ganho desejado}) = 0$. Considere as seguintes especificações:

Entrada: Ganho mínimo desejado em dB

Saída: Diâmetro(s) da antena ou mensagem de que o ganho é impossível.

Solução:

Para este exercício o pseudocódigo será semelhante ao anterior, mas com ajustes para a função de ganho da antena parabólica:

```
ALGORITMO GanhoAntenaParabolica

ENTRADA:
    ganho_desejado: real

SAÍDA:
    diametro(s): real ou mensagem
```

```
INÍCIO
    LER ganho_desejado

    // CONFIGURAR COEFICIENTES DA EQUAÇÃO QUADRÁTICA
    // Equação: -0.1d² + 3d + (5 - ganho_desejado) = 0
    a ← -0.1
    b ← 3
    temp_subtracao ← 5 - ganho_desejado
    c ← temp_subtracao

    // CALCULAR DISCRIMINANTE
    temp1 ← b * b
    temp2 ← 4 * a
    temp3 ← temp2 * c
    discriminante ← temp1 - temp3

    // VERIFICAR VIABILIDADE DA SOLUÇÃO
    SE discriminante < 0 ENTÃO
        ESCREVER "Ganho impossível de atingir com antena parabólica"
    SENÃO

        // CALCULAR RAIZ QUADRADA (MÉTODO DE NEWTON)
        raizQuadrada ← calcularRaizQuadrada(discriminante)


        // CALCULAR AS RAÍZES DA EQUAÇÃO QUADRÁTICA
        diametro1, diametro2 ← calcularRaizes(a, b, raizQuadrada)

        // VALIDAR DIÂMETROS FÍSICOS
        valida1 ← FALSO
        valida2 ← FALSO

        // Verificar primeiro diâmetro
        SE diametro1 > 0 ENTÃO
            valida1 ← VERDADEIRO
        FIM SE

        // Verificar segundo diâmetro
        SE diametro2 > 0 ENTÃO
            valida2 ← VERDADEIRO
        FIM SE


        // CALCULAR GANHOS PARA VERIFICAÇÃO
        SE valida1 = VERDADEIRO ENTÃO
            // Calcular G(d1) = -0.1d1² + 3d1 + 5
            temp_d1_quadrado ← diametro1 * diametro1
            temp_menos01d1_quadrado ← -0.1 * temp_d1_quadrado
            temp_3d1 ← 3 * diametro1
            temp_soma_termos ← temp_menos01d1_quadrado + temp_3d1
            ganho_verificacao1 ← temp_soma_termos + 5
```

```
FIM SE

SE valida2 = VERDADEIRO ENTÃO
    // Calcular G(d2) = -0.1d2² + 3d2 + 5
    temp_d2_quadrado ← diametro2 * diametro2
    temp_menos01d2_quadrado ← -0.1 * temp_d2_quadrado
    temp_3d2 ← 3 * diametro2
    temp_soma_termos2 ← temp_menos01d2_quadrado + temp_3d2
    ganho_verificacao2 ← temp_soma_termos2 + 5
FIM SE


// APRESENTAR RESULTADOS
SE valida1 = VERDADEIRO E valida2 = VERDADEIRO ENTÃO
    // Ordenar diâmetros (menor primeiro)
    SE diametro1 > diametro2 ENTÃO
        temp_swap ← diametro1
        diametro1 ← diametro2
        diametro2 ← temp_swap
    FIM SE

    ESCREVER "Dois diâmetros de antena possíveis:"
    ESCREVER "Primeiro diâmetro:", diametro1, "metros"
    ESCREVER "Segundo diâmetro:", diametro2, "metros"
    ESCREVER "Ambos geram o ganho de:", ganho_desejado, "dB"
    ESCREVER "Recomendação: usar o menor diâmetro por economia de material"

SENÃO SE valida1 = VERDADEIRO ENTÃO
    ESCREVER "Diâmetro da antena necessário:"
    ESCREVER diametro1, "metros"
    ESCREVER "Ganho gerado:", ganho_verificacao1, "dB"

    // Mostrar também diâmetro negativo se existir
    SE diametro2 < 0 ENTÃO
        ESCREVER "Diâmetro negativo descartado:", diametro2
    FIM SE

SENÃO SE valida2 = VERDADEIRO ENTÃO
    ESCREVER "Diâmetro da antena necessário:"
    ESCREVER diametro2, "metros"
    ESCREVER "Ganho gerado:", ganho_verificacao2, "dB"

    // Mostrar também diâmetro negativo se existir
    SE diametro1 < 0 ENTÃO
        ESCREVER "Diâmetro negativo descartado:", diametro1
    FIM SE

SENÃO
    ESCREVER "Todas as soluções resultam em diâmetros negativos:"
    ESCREVER "Diâmetro 1:", diametro1, "metros"
    ESCREVER "Diâmetro 2:", diametro2, "metros"
```

```
            ESCREVER "Não é possível construir antenas com diâmetros negativos"
        FIM SE
    FIM SE


    // INFORMAÇÕES TÉCNICAS DA ANTENA
    ESCREVER "Informações da função de ganho G(d) = -0.1d² + 3d + 5:"

    // Calcular diâmetro para ganho máximo (vértice da parábola)
    // d_max = -b/(2a) = -3/(2*(-0.1)) = 3/0.2 = 15
    temp_2a ← 2 * a
    temp_menos_b ← 0 - b
    d_maximo ← temp_menos_b / temp_2a

    // Calcular ganho máximo G(15)
    temp_dmax_quadrado ← d_maximo * d_maximo
    temp_menos01dmax_quadrado ← -0.1 * temp_dmax_quadrado
    temp_3dmax ← 3 * d_maximo
    temp_soma_max ← temp_menos01dmax_quadrado + temp_3dmax
    ganho_maximo ← temp_soma_max + 5

    ESCREVER "Diâmetro para ganho máximo:", d_maximo, "metros"
    ESCREVER "Ganho máximo possível:", ganho_maximo, "dB"

    // Calcular diâmetro mínimo para ganho positivo (G = 0)
    // -0.1d² + 3d + 5 = 0
    a_min ← -0.1
    b_min ← 3
    c_min ← 5

    temp1_min ← b_min * b_min
    temp2_min ← 4 * a_min
    temp3_min ← temp2_min * c_min
    discriminante_min ← temp1_min - temp3_min

    SE discriminante_min >= 0 ENTÃO
        raizQuadrada_min ← calcularRaizQuadrada(discriminante_min)
        diametro_min1, diametro_min2 ← calcularRaizes(a_min, b_min, raizQuadrada_min)

        // Escolher a maior raiz positiva (ponto onde ganho volta a zero)
        SE diametro_min1 > diametro_min2 ENTÃO
            d_limite ← diametro_min1
        SENÃO
            d_limite ← diametro_min2
        FIM SE

        ESCREVER "Diâmetro limite (ganho = 0dB):", d_limite, "metros"
    FIM SE
FIM

// FUNÇÃO AUXILIAR: CALCULAR RAIZ QUADRADA (MÉTODO DE NEWTON)
```

```
FUNÇÃO calcularRaizQuadrada(discriminante: real): real
INÍCIO
    raizQuadrada ← discriminante / 2

    ENQUANTO abs(raizQuadrada * raizQuadrada - discriminante) > 0.0001 FAÇA
        // Calcular raizQuadrada²
        temp_quad ← raizQuadrada * raizQuadrada

        // Calcular numerador: raizQuadrada² + discriminante
        numerador ← temp_quad + discriminante

        // Calcular denominador: 2 * raizQuadrada
        denominador ← 2 * raizQuadrada

        // Nova aproximação
        raizQuadrada ← numerador / denominador
    FIM ENQUANTO

    RETORNAR raizQuadrada
FIM

// FUNÇÃO AUXILIAR: CALCULAR RAÍZES DA EQUAÇÃO QUADRÁTICA
FUNÇÃO calcularRaizes(a: real, b: real, raizQuadrada: real): real, real
INÍCIO
    // Calcular -b
    menos_b ← 0 - b

    // Calcular denominador comum: 2a
    denominador_comum ← 2 * a

    // Calcular raiz1: (-b + √Δ) / 2a
    numerador1 ← menos_b + raizQuadrada
    raiz1 ← numerador1 / denominador_comum

    // Calcular raiz2: (-b - √Δ) / 2a
    numerador2 ← menos_b - raizQuadrada
    raiz2 ← numerador2 / denominador_comum

    RETORNAR raiz1, raiz2
FIM

// FUNÇÃO AUXILIAR: VALOR ABSOLUTO
FUNÇÃO abs(valor: real): real
INÍCIO
    SE valor < 0 ENTÃO
        resultado ← 0 - valor
    SENÃO
        resultado ← valor
    FIM SE
    RETORNAR resultado
FIM
```

O código em C++23 para o pseudocódigo acima pode ser encontrado no site GDB Online no link https://onlinegdb.com/ZGj9kp4Jp.

7.1.2 Usando funções da biblioteca padrão

A maioria das linguagens de programação modernas possui um conjunto de funções pré-definidas para as ações comuns em uma grande variedade de aplicações. Essas funções são otimizadas e testadas, oferecendo uma maneira eficiente de realizar tarefas comuns sem a necessidade de reescrever o código. Os exercícios a seguir exploram o uso destas funções. O professor, deve adequar estes exercícios de acordo com a linguagem de programação escolhida para a disciplina. Aqui, os exemplos serão realizados em C++23.

Para o conjunto e exercícios a seguir, considere que o aluno já tenha sido apresentado às funções de manipulação de *strings*, manipulação de arquivos, e funções matemáticas básicas. Além das funções, métodos e classes da biblioteca padrão que estamos usando desde o primeiro código que fizemos, como `std::cout`, `std::cin`, `std::string`, etc.

Os exercícios a seguir foram dimensionados para o uso da Técnica da Sequência de Fibonacci (1, 1, 2, 3,), como foi proposto para as seções anteriores.

Finalmente, todos os exercícios são para a criação de sistemas especialistas. O professor pode aproveitar este conjunto de exercícios para discutir com os alunos a importância de se criar sistemas especialistas.

A2: Uma casa lotérica precisa de um sistema para calcular probabilidades e combinações para seus jogos. O sistema deve calcular quantas combinações existem para a Mega-Sena (6 números entre 1 e 60), probabilidades de acertos parciais e outras operações matemáticas relacionadas. Desenvolva um programa que ofereça um menu com opções para calcular fatorial de um número, combinações $C(n, r)$, permutações $P(n, r)$, as probabilidades relacionadas com a Mega-Sena, e a chance de ganhar o prêmio principal de outras modalidades de loteria (Lotofácil, Quina, etc.). Conforme as especificações a seguir:

Funções a implementar:

- **Funções próprias**: `calcularFatorial()`, `calcularCombinacao()`, `calcularPermutacao()`
- **Funções da biblioteca padrão**: As funções que sejam necessárias para resolver o problema

Menu do sistema:

1. Calcular fatorial de um número;
2. Calcular combinação $C(n, r)$;
3. Calcular permutação $P(n, r)$;
4. Mega-Sena: probabilidades de acertos (sena, quina, quadra);
5. Outras loterias: Lotofácil (15 de 25), Quina (5 de 80), Dupla Sena (6 de 50);
6. Sair.

Entrada: Números para os cálculos conforme a opção escolhida
Saída: Menu interativo com resultados dos cálculos matemáticos
Operações específicas:

- Mega-Sena: $C(60, 6)$ combinações totais, probabilidades de 4, 5 e 6 acertos;
- Lotofácil: $C(25, 15)$ para 15 acertos;
- Quina: $C(80, 5)$ para 5 acertos;
- Dupla Sena: $C(50, 6)$ para 6 acertos em cada sorteio.

Solução:
Começamos com o pseudocódigo para o exercício:

```
ALGORITMO SistemaCasaLoterica

ENTRADA:
    opcao_menu, numeros: inteiro

SAÍDA:
    resultados_calculos: real

INÍCIO
    opcao ← 0

    ENQUANTO opcao  7 FAÇA

        // APRESENTAR MENU PRINCIPAL

        ESCREVER "=== SISTEMA CASA LOTÉRICA ==="
        ESCREVER "1. Calcular Fatorial"
        ESCREVER "2. Calcular Combinação C(n,r)"
        ESCREVER "3. Calcular Permutação P(n,r)"
        ESCREVER "4. Mega-Sena: Probabilidades Completas"
        ESCREVER "5. Outros Jogos de Loteria"
        ESCREVER "6. Simulação de Apostas"
        ESCREVER "7. Sair"
        ESCREVER "Escolha uma opção:"
        LER opcao

        // PROCESSAR OPÇÃO ESCOLHIDA
        SE opcao = 1 ENTÃO
            ESCREVER "Digite um número para calcular o fatorial:"
            LER n
            SE n >= 0 ENTÃO
                resultado ← calcularFatorial(n)
                ESCREVER "Fatorial de", n, "=", resultado
            SENÃO
                ESCREVER "Erro: Número deve ser não-negativo"
            FIM SE

        SENÃO SE opcao = 2 ENTÃO
            ESCREVER "Digite n (total de elementos):"
            LER n
            ESCREVER "Digite r (elementos escolhidos):"
            LER r
            SE n >= 0 E r >= 0 E r <= n ENTÃO
                resultado ← calcularCombinacao(n, r)
                ESCREVER "C(", n, ",", r, ") =", resultado
            SENÃO
                ESCREVER "Erro: Parâmetros inválidos (n 0, r 0, r n)"
            FIM SE

        SENÃO SE opcao = 3 ENTÃO
            ESCREVER "Digite n (total de elementos):"
```

```
        LER n
        ESCREVER "Digite r (elementos escolhidos):"
        LER r
        SE n >= 0 E r >= 0 E r <= n ENTÃO
            resultado ← calcularPermutacao(n, r)
            ESCREVER "P(", n, ",", r, ") =", resultado
        SENÃO
            ESCREVER "Erro: Parâmetros inválidos (n 0, r 0, r n)"
        FIM SE

    SENÃO SE opcao = 4 ENTÃO
        ESCREVER "=== MEGA-SENA: ANÁLISE COMPLETA ==="

        // Combinações totais possíveis
        total_combinacoes ← calcularCombinacao(60, 6)
        ESCREVER "Total de combinações possíveis:", total_combinacoes

        // Probabilidade de acertar 6 números (sena)
        prob_sena ← 1.0 / total_combinacoes
        ESCREVER "Sena (6 acertos):"
        ESCREVER "  Probabilidade:", prob_sena
        ESCREVER "  Chance: 1 em", total_combinacoes

        // Probabilidade de acertar 5 números (quina)
        comb_5_acertos ← calcularCombinacao(6, 5)
        comb_1_erro ← calcularCombinacao(54, 1)
        casos_quina ← comb_5_acertos * comb_1_erro
        prob_quina ← casos_quina / total_combinacoes
        chance_quina ← total_combinacoes / casos_quina
        ESCREVER "Quina (5 acertos):"
        ESCREVER "  Probabilidade:", prob_quina
        ESCREVER "  Chance: 1 em", chance_quina

        // Probabilidade de acertar 4 números (quadra)
        comb_4_acertos ← calcularCombinacao(6, 4)
        comb_2_erros ← calcularCombinacao(54, 2)
        casos_quadra ← comb_4_acertos * comb_2_erros
        prob_quadra ← casos_quadra / total_combinacoes
        chance_quadra ← total_combinacoes / casos_quadra
        ESCREVER "Quadra (4 acertos):"
        ESCREVER "  Probabilidade:", prob_quadra
        ESCREVER "  Chance: 1 em", chance_quadra

    SENÃO SE opcao = 5 ENTÃO
        ESCREVER "=== OUTROS JOGOS DE LOTERIA ==="
        ESCREVER "Escolha o jogo:"
        ESCREVER "1. Lotofácil (15 de 25)"
        ESCREVER "2. Quina (5 de 80)"
        ESCREVER "3. Dupla Sena (6 de 50, dois sorteios)"
        ESCREVER "4. Personalizado"
        LER jogo
```

```
        SE jogo = 1 ENTÃO
            // Lotofácil: 15 números de 25
            total_lotofacil ← calcularCombinacao(25, 15)
            prob_lotofacil ← 1.0 / total_lotofacil
            ESCREVER "Lotofácil - Combinações possíveis:", total_lotofacil
            ESCREVER "Probabilidade de 15 acertos:", prob_lotofacil

        SENÃO SE jogo = 2 ENTÃO
            // Quina: 5 números de 80
            total_quina ← calcularCombinacao(80, 5)
            prob_quina_jogo ← 1.0 / total_quina
            ESCREVER "Quina - Combinações possíveis:", total_quina
            ESCREVER "Probabilidade de 5 acertos:", prob_quina_jogo

        SENÃO SE jogo = 3 ENTÃO
            // Dupla Sena: 6 números de 50, dois sorteios
            total_dupla ← calcularCombinacao(50, 6)
            prob_um_sorteio ← 1.0 / total_dupla
            prob_qualquer_sorteio ← 2.0 * prob_um_sorteio
            ESCREVER "Dupla Sena - Combinações por sorteio:", total_dupla
            ESCREVER "Probabilidade em um sorteio:", prob_um_sorteio
            ESCREVER "Probabilidade em qualquer sorteio:", prob_qualquer_sorteio

        SENÃO SE jogo = 4 ENTÃO
            ESCREVER "Digite o total de números disponíveis:"
            LER total_numeros
            ESCREVER "Digite quantos números são sorteados:"
            LER numeros_sorteados
            SE total_numeros >= numeros_sorteados E numeros_sorteados > 0 ENTÃO
                total_personalizado ← calcularCombinacao(total_numeros, numeros_sorte
                prob_personalizado ← 1.0 / total_personalizado
                ESCREVER "Combinações possíveis:", total_personalizado
                ESCREVER "Probabilidade de acerto:", prob_personalizado
            SENÃO
                ESCREVER "Parâmetros inválidos"
            FIM SE
        SENÃO
            ESCREVER "Opção inválida"
        FIM SE

    SENÃO SE opcao = 6 ENTÃO
        ESCREVER "=== SIMULAÇÃO DE APOSTAS ==="
        ESCREVER "Digite o valor da aposta individual (R$):"
        LER valor_aposta
        ESCREVER "Digite o número de apostas:"
        LER num_apostas

        // Custo total
        custo_total ← valor_aposta * num_apostas
        ESCREVER "Custo total: R$", custo_total
```

```
            // Probabilidade de ganhar pelo menos uma vez usando pow()
            total_comb_mega ← calcularCombinacao(60, 6)
            prob_perder_uma ← (total_comb_mega - 1.0) / total_comb_mega
            prob_perder_todas ← pow(prob_perder_uma, num_apostas)
            prob_ganhar_pelo_menos_uma ← 1.0 - prob_perder_todas

            ESCREVER "Probabilidade de não ganhar:", prob_perder_todas
            ESCREVER "Probabilidade de ganhar pelo menos uma vez:", prob_ganhar_pelo_meno

            // Estatísticas
            apostas_necessarias ← total_comb_mega
            custo_garantia ← apostas_necessarias * valor_aposta
            ESCREVER "Apostas necessárias para garantir vitória:", apostas_necessarias
            ESCREVER "Custo para garantir vitória: R$", custo_garantia

        SENÃO SE opcao = 7 ENTÃO
            ESCREVER "Encerrando sistema da casa lotérica..."

        SENÃO
            ESCREVER "Opção inválida! Tente novamente."
        FIM SE

        // Pausa para visualização
        SE opcao   7 ENTÃO
            ESCREVER "Pressione Enter para continuar..."
            LER pausa
        FIM SE
    FIM ENQUANTO
FIM

// FUNÇÃO: CALCULAR FATORIAL
FUNÇÃO calcularFatorial(n: inteiro): real
INÍCIO
    SE n < 0 ENTÃO
        RETORNAR -1
    FIM SE

    SE n = 0 OU n = 1 ENTÃO
        RETORNAR 1
    FIM SE

    resultado ← 1.0
    PARA i DE 2 ATÉ n FAÇA
        resultado ← resultado * i
    FIM PARA

    RETORNAR resultado
FIM

// FUNÇÃO: CALCULAR COMBINAÇÃO C(n,r)
```

```
FUNÇÃO calcularCombinacao(n: inteiro, r: inteiro): real
INÍCIO
    SE n < 0 OU r < 0 OU r > n ENTÃO
        RETORNAR -1
    FIM SE

    SE r = 0 OU r = n ENTÃO
        RETORNAR 1
    FIM SE

    // Otimização: C(n,r) = C(n,n-r)
    SE r > n - r ENTÃO
        r ← n - r
    FIM SE

    // Cálculo iterativo para evitar overflow
    resultado ← 1.0
    PARA i DE 0 ATÉ r-1 FAÇA
        resultado ← resultado * (n - i)
        resultado ← resultado / (i + 1)
    FIM PARA

    RETORNAR resultado
FIM

// FUNÇÃO: CALCULAR PERMUTAÇÃO P(n,r)
FUNÇÃO calcularPermutacao(n: inteiro, r: inteiro): real
INÍCIO
    SE n < 0 OU r < 0 OU r > n ENTÃO
        RETORNAR -1
    FIM SE

    SE r = 0 ENTÃO
        RETORNAR 1
    FIM SE

    // P(n,r) = n! / (n-r)! = n * (n-1) * ... * (n-r+1)
    resultado ← 1.0
    PARA i DE 0 ATÉ r-1 FAÇA
        resultado ← resultado * (n - i)
    FIM PARA

    RETORNAR resultado
FIM
```

O código em C++23 para implementar o pseudocódigo acima pode ser encontrado no site GDB Online no link https://onlinegdb.com/ugIolKSuu.

Finalmente, este exercício **A2** pode ser usado para introduzir outra abstração, `switch-case`. Sendo assim, o pseudocódigo será:

```
ALGORITMO SistemaCasaLoterica

ENTRADA:
    opcao_menu, numeros: inteiro

SAÍDA:
    resultados_calculos: real

INÍCIO
    opcao ← 0

    ENQUANTO opcao  6 FAÇA

        // APRESENTAR MENU PRINCIPAL
        ESCREVER "=== SISTEMA CASA LOTÉRICA ==="
        ESCREVER "1. Calcular fatorial de um número"
        ESCREVER "2. Calcular combinação C(n,r)"
        ESCREVER "3. Calcular permutação P(n,r)"
        ESCREVER "4. Mega-Sena: probabilidades de acertos"
        ESCREVER "5. Outras loterias"
        ESCREVER "6. Sair"
        ESCREVER "Escolha uma opção:"
        LER opcao

        // PROCESSAR OPÇÃO COM SWITCH-CASE
        ESCOLHA opcao FAÇA
            CASO 1:
                ESCREVER "Digite um número para calcular o fatorial:"
                LER n
                SE n >= 0 ENTÃO
                    resultado ← calcularFatorial(n)
                    ESCREVER "Fatorial de", n, "=", resultado
                SENÃO
                    ESCREVER "Erro: Número deve ser não-negativo"
                FIM SE
                PARE

            CASO 2:
                ESCREVER "Digite n (total de elementos):"
                LER n
                ESCREVER "Digite r (elementos escolhidos):"
                LER r
                SE n >= 0 E r >= 0 E r <= n ENTÃO
                    resultado ← calcularCombinacao(n, r)
                    ESCREVER "C(", n, ",", r, ") =", resultado
                SENÃO
                    ESCREVER "Erro: Parâmetros inválidos"
                FIM SE
                PARE

            CASO 3:
```

```
            ESCREVER "Digite n (total de elementos):"
            LER n
            ESCREVER "Digite r (elementos escolhidos):"
            LER r
            SE n >= 0 E r >= 0 E r <= n ENTÃO
                resultado ← calcularPermutacao(n, r)
                ESCREVER "P(", n, ",", r, ") =", resultado
            SENÃO
                ESCREVER "Erro: Parâmetros inválidos"
            FIM SE
            PARE

        CASO 4:
            ESCREVER "=== MEGA-SENA: PROBABILIDADES ==="

            // Combinações totais C(60,6)
            total_combinacoes ← calcularCombinacao(60, 6)
            ESCREVER "Total de combinações:", total_combinacoes

            // Sena (6 acertos)
            prob_sena ← 1.0 / total_combinacoes
            ESCREVER "Sena (6 acertos): 1 em", total_combinacoes

            // Quina (5 acertos): C(6,5) * C(54,1) / C(60,6)
            casos_quina ← calcularCombinacao(6, 5) * calcularCombinacao(54, 1)
            prob_quina ← casos_quina / total_combinacoes
            chance_quina ← total_combinacoes / casos_quina
            ESCREVER "Quina (5 acertos): 1 em", chance_quina

            // Quadra (4 acertos): C(6,4) * C(54,2) / C(60,6)
            casos_quadra ← calcularCombinacao(6, 4) * calcularCombinacao(54, 2)
            prob_quadra ← casos_quadra / total_combinacoes
            chance_quadra ← total_combinacoes / casos_quadra
            ESCREVER "Quadra (4 acertos): 1 em", chance_quadra
            PARE

        CASO 5:
            ESCREVER "=== OUTRAS LOTERIAS ==="
            ESCREVER "1. Lotofácil (15 de 25)"
            ESCREVER "2. Quina (5 de 80)"
            ESCREVER "3. Dupla Sena (6 de 50)"
            ESCREVER "Escolha:"
            LER subloteria

            ESCOLHA subloteria FAÇA
                CASO 1:
                    total_lotofacil ← calcularCombinacao(25, 15)
                    ESCREVER "Lotofácil - Combinações:", total_lotofacil
                    ESCREVER "Chance de ganhar: 1 em", total_lotofacil
                    PARE
```

```
                    CASO 2:
                        total_quina ← calcularCombinacao(80, 5)
                        ESCREVER "Quina - Combinações:", total_quina
                        ESCREVER "Chance de ganhar: 1 em", total_quina
                        PARE

                    CASO 3:
                        total_dupla ← calcularCombinacao(50, 6)
                        ESCREVER "Dupla Sena - Combinações por sorteio:", total_dupla
                        ESCREVER "Chance por sorteio: 1 em", total_dupla
                        PARE

                    PADRÃO:
                        ESCREVER "Opção inválida"
                        PARE
                FIM ESCOLHA
                PARE

            CASO 6:
                ESCREVER "Encerrando sistema..."
                PARE

            PADRÃO:
                ESCREVER "Opção inválida! Tente novamente."
                PARE
        FIM ESCOLHA

        // Pausa para visualização
        SE opcao  6 ENTÃO
            ESCREVER "Pressione Enter para continuar..."
            LER pausa
        FIM SE
    FIM ENQUANTO
FIM

// FUNÇÃO: CALCULAR FATORIAL
FUNÇÃO calcularFatorial(n: inteiro): real
INÍCIO
    SE n < 0 ENTÃO
        RETORNAR -1
    FIM SE

    SE n = 0 OU n = 1 ENTÃO
        RETORNAR 1
    FIM SE

    resultado ← 1.0
    PARA i DE 2 ATÉ n FAÇA
        resultado ← resultado * i
    FIM PARA
```

```
    RETORNAR resultado
FIM

// FUNÇÃO: CALCULAR COMBINAÇÃO C(n,r)
FUNÇÃO calcularCombinacao(n: inteiro, r: inteiro): real
INÍCIO
    SE n < 0 OU r < 0 OU r > n ENTÃO
        RETORNAR -1
    FIM SE

    SE r = 0 OU r = n ENTÃO
        RETORNAR 1
    FIM SE

    // Otimização: C(n,r) = C(n,n-r)
    SE r > n - r ENTÃO
        r ← n - r
    FIM SE

    // Cálculo iterativo para evitar overflow
    resultado ← 1.0
    PARA i DE 0 ATÉ r-1 FAÇA
        resultado ← resultado * (n - i)
        resultado ← resultado / (i + 1)
    FIM PARA

    RETORNAR resultado
FIM

// FUNÇÃO: CALCULAR PERMUTAÇÃO P(n,r)
FUNÇÃO calcularPermutacao(n: inteiro, r: inteiro): real
INÍCIO
    SE n < 0 OU r < 0 OU r > n ENTÃO
        RETORNAR -1
    FIM SE

    SE r = 0 ENTÃO
        RETORNAR 1
    FIM SE

    // P(n,r) = n! / (n-r)!
    resultado ← 1.0
    PARA i DE 0 ATÉ r-1 FAÇA
        resultado ← resultado * (n - i)
    FIM PARA

    RETORNAR resultado
FIM
```

Neste caso, #### Código C++23 O código para implementar o pseudocódigo com `switch-case` está disponível no site GDB Online no link https://onlinegdb.com/WED1nNGeV.

B2: um banco precisa de um sistema para orientar clientes sobre investimentos. O sistema deve calcular rendimentos de aplicações em renda fixa, determinar quanto tempo leva para duplicar um investimento e validar se os dados inseridos pelo cliente estão corretos. Crie um programa que calcule o montante final de um investimento com juros compostos, determine em quantos anos o capital dobra de valor e valide se as entradas do usuário são positivas e realistas.

Funções a implementar:

- **Funções próprias**: `calcularJuroComposto()`, `tempoParaDobrar()`, `validarEntrada()`
- **Funções padrão**: `pow()`, `log()`

Entrada: Capital inicial, taxa de juros anual, período de investimento **Saída**: Montante final, tempo para dobrar o capital, alertas de validação

Solução:

Neste caso, podemos representar o probleam como:

```
ALGORITMO SistemaBancarioInvestimentos

ENTRADA:
    capital_inicial, taxa_juros, periodo: real

SAÍDA:
    montante_final, tempo_dobrar, alertas: real

INÍCIO
    opcao ← 0

    ENQUANTO opcao  5 FAÇA

        // APRESENTAR MENU PRINCIPAL

        ESCREVER "=== SISTEMA BANCÁRIO DE INVESTIMENTOS ==="
        ESCREVER "1. Calcular montante final (juros compostos)"
        ESCREVER "2. Calcular tempo para dobrar o capital"
        ESCREVER "3. Validar dados de entrada"
        ESCREVER "4. Sistema completo de análise"
        ESCREVER "5. Sair"
        ESCREVER "Escolha uma opção:"
        LER opcao


        // PROCESSAR OPÇÃO ESCOLHIDA

        SE opcao = 1 ENTÃO
            ESCREVER "=== CÁLCULO DE MONTANTE FINAL ==="
            ESCREVER "Digite o capital inicial (R$):"
            LER capital_inicial
            ESCREVER "Digite a taxa de juros anual (%):"
            LER taxa_juros
            ESCREVER "Digite o período em anos:"
            LER periodo
```

```
            // Validar entradas
            validacao ← validarEntrada(capital_inicial, taxa_juros, periodo)

            SE validacao = VERDADEIRO ENTÃO
                montante ← calcularJuroComposto(capital_inicial, taxa_juros, periodo)
                rendimento ← montante - capital_inicial
                ESCREVER "Capital inicial: R$", capital_inicial
                ESCREVER "Montante final: R$", montante
                ESCREVER "Rendimento: R$", rendimento
                porcentagem_ganho ← (rendimento / capital_inicial) * 100
                ESCREVER "Ganho percentual:", porcentagem_ganho, "%"
            SENÃO
                ESCREVER "Dados inválidos! Verifique os valores inseridos."
            FIM SE

        SENÃO SE opcao = 2 ENTÃO
            ESCREVER "=== TEMPO PARA DOBRAR O CAPITAL ==="
            ESCREVER "Digite a taxa de juros anual (%):"
            LER taxa_juros

            SE taxa_juros > 0 E taxa_juros <= 50 ENTÃO
                tempo ← tempoParaDobrar(taxa_juros)
                ESCREVER "Com taxa de", taxa_juros, "% ao ano:"
                ESCREVER "Tempo para dobrar o capital:", tempo, "anos"

                // Regra prática dos 72
                regra_72 ← 72 / taxa_juros
                ESCREVER "Regra dos 72 (aproximação):", regra_72, "anos"
                diferenca ← abs(tempo - regra_72)
                ESCREVER "Diferença entre cálculo exato e regra dos 72:", diferenca, "ano
            SENÃO
                ESCREVER "Taxa de juros inválida! Use valores entre 0.1% e 50%."
            FIM SE

        SENÃO SE opcao = 3 ENTÃO
            ESCREVER "=== VALIDAÇÃO DE DADOS ==="
            ESCREVER "Digite o capital inicial (R$):"
            LER capital_inicial
            ESCREVER "Digite a taxa de juros anual (%):"
            LER taxa_juros
            ESCREVER "Digite o período em anos:"
            LER periodo

            validacao ← validarEntrada(capital_inicial, taxa_juros, periodo)

            SE validacao = VERDADEIRO ENTÃO
                ESCREVER "  Todos os dados estão válidos!"
                ESCREVER "  Capital inicial: R$", capital_inicial, "(válido)"
                ESCREVER "  Taxa de juros:", taxa_juros, "% ao ano (válida)"
                ESCREVER "  Período:", periodo, "anos (válido)"
            SENÃO
```

```
                ESCREVER "  Dados inválidos detectados!"

                // Validações específicas
                SE capital_inicial <= 0 ENTÃO
                    ESCREVER "  Capital inicial deve ser positivo"
                FIM SE

                SE taxa_juros <= 0 OU taxa_juros > 50 ENTÃO
                    ESCREVER "  Taxa de juros deve estar entre 0.1% e 50%"
                FIM SE

                SE periodo <= 0 OU periodo > 100 ENTÃO
                    ESCREVER "  Período deve estar entre 1 e 100 anos"
                FIM SE
            FIM SE

        SENÃO SE opcao = 4 ENTÃO
            ESCREVER "=== SISTEMA COMPLETO DE ANÁLISE ==="
            ESCREVER "Digite o capital inicial (R$):"
            LER capital_inicial
            ESCREVER "Digite a taxa de juros anual (%):"
            LER taxa_juros
            ESCREVER "Digite o período em anos:"
            LER periodo

            // Validar todas as entradas
            validacao ← validarEntrada(capital_inicial, taxa_juros, periodo)

            SE validacao = VERDADEIRO ENTÃO
                ESCREVER "\n--- ANÁLISE COMPLETA DO INVESTIMENTO ---"

                // Cálculo do montante final
                montante ← calcularJuroComposto(capital_inicial, taxa_juros, periodo)
                rendimento ← montante - capital_inicial

                ESCREVER "  RESULTADOS FINANCEIROS:"
                ESCREVER "Capital inicial: R$", capital_inicial
                ESCREVER "Montante final: R$", montante
                ESCREVER "Rendimento total: R$", rendimento
                porcentagem_ganho ← (rendimento / capital_inicial) * 100
                ESCREVER "Ganho percentual total:", porcentagem_ganho, "%"

                // Tempo para dobrar
                tempo_dobrar ← tempoParaDobrar(taxa_juros)
                ESCREVER "\n  TEMPO PARA DOBRAR:"
                ESCREVER "Tempo para dobrar o capital:", tempo_dobrar, "anos"

                // Análise comparativa
                SE periodo >= tempo_dobrar ENTÃO
                    multiplicador ← periodo / tempo_dobrar
                    ESCREVER "Em", periodo, "anos, o capital dobrará", multiplicador, "ve
```

```
                SENÃO
                    faltam_anos ← tempo_dobrar - periodo
                    ESCREVER "Faltam", faltam_anos, "anos para o capital dobrar"
                FIM SE

                // Recomendações
                ESCREVER "\n  RECOMENDAÇÕES:"
                SE taxa_juros < 5 ENTÃO
                    ESCREVER "   Taxa baixa - considere outras opções de investimento"
                SENÃO SE taxa_juros > 20 ENTÃO
                    ESCREVER "   Taxa alta - verifique os riscos envolvidos"
                SENÃO
                    ESCREVER "  Taxa dentro de padrões normais do mercado"
                FIM SE

                SE periodo < 2 ENTÃO
                    ESCREVER "  Investimento de curto prazo - juros compostos têm menor im
                SENÃO SE periodo > 10 ENTÃO
                    ESCREVER "  Investimento de longo prazo - excelente para juros compost
                FIM SE

            SENÃO
                ESCREVER "  Não é possível realizar a análise com dados inválidos."
                ESCREVER "Por favor, corrija os valores e tente novamente."
            FIM SE

        SENÃO SE opcao = 5 ENTÃO
            ESCREVER "Encerrando sistema bancário..."

        SENÃO
            ESCREVER "Opção inválida! Tente novamente."
        FIM SE

        // Pausa para visualização
        SE opcao   5 ENTÃO
            ESCREVER "\nPressione Enter para continuar..."
            LER pausa
        FIM SE
    FIM ENQUANTO
FIM


// FUNÇÃO: CALCULAR JUROS COMPOSTOS
// M = C * (1 + i)^t

FUNÇÃO calcularJuroComposto(capital: real, taxa: real, tempo: real): real
INÍCIO
    // Converter taxa percentual para decimal
    taxa_decimal ← taxa / 100

    // Calcular (1 + taxa)
```

```
    base ← 1 + taxa_decimal

    // Calcular (1 + taxa)^tempo usando pow()
    fator_multiplicacao ← pow(base, tempo)

    // Calcular montante final
    montante ← capital * fator_multiplicacao

    RETORNAR montante
FIM


// FUNÇÃO: TEMPO PARA DOBRAR O CAPITAL
// 2 = (1 + i)^t  =>  t = log(2) / log(1 + i)

FUNÇÃO tempoParaDobrar(taxa: real): real
INÍCIO
    // Converter taxa percentual para decimal
    taxa_decimal ← taxa / 100

    // Calcular (1 + taxa)
    base ← 1 + taxa_decimal

    // Calcular log(2) / log(1 + taxa) usando log()
    numerador ← log(2)
    denominador ← log(base)
    tempo ← numerador / denominador

    RETORNAR tempo
FIM

// FUNÇÃO: VALIDAR ENTRADA
FUNÇÃO validarEntrada(capital: real, taxa: real, periodo: real): logico
INÍCIO
    // Validar capital inicial
    SE capital <= 0 ENTÃO
        RETORNAR FALSO
    FIM SE

    // Validar taxa de juros (entre 0.1% e 50%)
    SE taxa <= 0 OU taxa > 50 ENTÃO
        RETORNAR FALSO
    FIM SE

    // Validar período (entre 1 e 100 anos)
    SE periodo <= 0 OU periodo > 100 ENTÃO
        RETORNAR FALSO
    FIM SE

    // Se chegou até aqui, todos os dados são válidos
    RETORNAR VERDADEIRO
```

```
FIM
```

Resultando no código em C++23 que pode ser encontrado no GDB Online no link https://onlinegdb.com/Ha2Sfflfd.

C2: Um escritório de engenharia civil precisa de um sistema para calcular parâmetros de rampas de acessibilidade e escadas. O sistema deve determinar ângulos de inclinação, comprimentos necessários e validar se os projetos atendem às normas de acessibilidade. Crie um programa que calcule ângulos de rampas, comprimentos de escadas e valide conformidade com normas brasileiras de acessibilidade.

Funções a implementar: - **Funções próprias**: `calcularAnguloRampa()`, `calcularComprimentoEscad` `validarAcessibilidade()` - **Funções padrão**: `sin()`, `cos()`, `tan()`, `atan()`, `asin()`

Entrada: Altura, comprimento, ângulo (conforme tipo de cálculo) **Saída**: Ângulos em graus, comprimentos em metros, status de conformidade com normas

Cálculos específicos: - Rampa de acessibilidade: máximo 8.33% de inclinação (NBR 9050) - Escadas: ângulo ideal entre 25° e 35° - Conversão radianos/graus automática

Solução:

Começando pelo pseudocódigo, teremos:

```
ALGORITMO SistemaEngenhariaCivil

ENTRADA:
    altura, comprimento, angulo: real

SAÍDA:
    angulos, comprimentos, status_conformidade: real

INÍCIO
    opcao ← 0
    PI ← 3.14159265359

    ENQUANTO opcao   6 FAÇA

        // APRESENTAR MENU PRINCIPAL
        ESCREVER "=== SISTEMA DE ENGENHARIA CIVIL ==="
        ESCREVER "1. Calcular ângulo de rampa de acessibilidade"
        ESCREVER "2. Calcular comprimento de escada"
        ESCREVER "3. Validar conformidade com normas"
        ESCREVER "4. Análise completa de projeto"
        ESCREVER "5. Conversões trigonométricas"
        ESCREVER "6. Sair"
        ESCREVER "Escolha uma opção:"
        LER opcao


        // PROCESSAR OPÇÃO ESCOLHIDA
        SE opcao = 1 ENTÃO
            ESCREVER "=== CÁLCULO DE ÂNGULO DE RAMPA ==="
            ESCREVER "Digite a altura da rampa (metros):"
            LER altura
            ESCREVER "Digite o comprimento da rampa (metros):"
            LER comprimento
```

```
        SE altura > 0 E comprimento > 0 ENTÃO
            angulo ← calcularAnguloRampa(altura, comprimento)
            inclinacao_percentual ← (altura / comprimento) * 100

            ESCREVER "Altura:", altura, "m"
            ESCREVER "Comprimento:", comprimento, "m"
            ESCREVER "Ângulo de inclinação:", angulo, "graus"
            ESCREVER "Inclinação percentual:", inclinacao_percentual, "%"

            // Verificar conformidade NBR 9050
            SE inclinacao_percentual <= 8.33 ENTÃO
                ESCREVER "  Rampa CONFORME com NBR 9050 (máx. 8.33%)"
            SENÃO
                ESCREVER "  Rampa NÃO CONFORME com NBR 9050"
                comprimento_minimo ← altura / 0.0833
                ESCREVER "Comprimento mínimo necessário:", comprimento_minimo, "m"
            FIM SE
        SENÃO
            ESCREVER "Erro: Altura e comprimento devem ser positivos"
        FIM SE

    SENÃO SE opcao = 2 ENTÃO
        ESCREVER "=== CÁLCULO DE COMPRIMENTO DE ESCADA ==="
        ESCREVER "Digite a altura total da escada (metros):"
        LER altura
        ESCREVER "Digite o ângulo desejado (graus):"
        LER angulo

        SE altura > 0 E angulo > 0 E angulo < 90 ENTÃO
            comprimento ← calcularComprimentoEscada(altura, angulo)
            projecao_horizontal ← altura / tan(angulo * PI / 180)

            ESCREVER "Altura total:", altura, "m"
            ESCREVER "Ângulo:", angulo, "graus"
            ESCREVER "Comprimento da escada:", comprimento, "m"
            ESCREVER "Projeção horizontal:", projecao_horizontal, "m"

            // Verificar ângulo ideal para escadas
            SE angulo >= 25 E angulo <= 35 ENTÃO
                ESCREVER "  Ângulo IDEAL para escadas (25° a 35°)"
            SENÃO SE angulo < 25 ENTÃO
                ESCREVER "  Ângulo muito baixo - escada pode ser desconfortável"
            SENÃO
                ESCREVER "  Ângulo muito alto - escada pode ser perigosa"
            FIM SE
        SENÃO
            ESCREVER "Erro: Altura deve ser positiva e ângulo entre 0° e 90°"
        FIM SE

    SENÃO SE opcao = 3 ENTÃO
```

```
        ESCREVER "=== VALIDAÇÃO DE CONFORMIDADE ==="
        ESCREVER "Escolha o tipo de estrutura:"
        ESCREVER "1. Rampa de acessibilidade"
        ESCREVER "2. Escada"
        LER tipo_estrutura

        SE tipo_estrutura = 1 ENTÃO
            ESCREVER "Digite a altura da rampa (metros):"
            LER altura
            ESCREVER "Digite o comprimento da rampa (metros):"
            LER comprimento

            SE altura > 0 E comprimento > 0 ENTÃO
                conformidade ← validarAcessibilidade(altura, comprimento, 1)

                SE conformidade = VERDADEIRO ENTÃO
                    ESCREVER "  Rampa APROVADA - Conforme NBR 9050"
                SENÃO
                    ESCREVER "  Rampa REPROVADA - Não conforme NBR 9050"
                FIM SE
            SENÃO
                ESCREVER "Dados inválidos"
            FIM SE

        SENÃO SE tipo_estrutura = 2 ENTÃO
            ESCREVER "Digite o ângulo da escada (graus):"
            LER angulo

            SE angulo > 0 E angulo < 90 ENTÃO
                conformidade ← validarAcessibilidade(0, 0, angulo)

                SE conformidade = VERDADEIRO ENTÃO
                    ESCREVER "  Escada APROVADA - Ângulo adequado"
                SENÃO
                    ESCREVER "  Escada com ângulo inadequado"
                FIM SE
            SENÃO
                ESCREVER "Ângulo inválido"
            FIM SE
        SENÃO
            ESCREVER "Opção inválida"
        FIM SE

    SENÃO SE opcao = 4 ENTÃO
        ESCREVER "=== ANÁLISE COMPLETA DE PROJETO ==="
        ESCREVER "Digite a altura (metros):"
        LER altura
        ESCREVER "Digite o comprimento disponível (metros):"
        LER comprimento_disponivel

        SE altura > 0 E comprimento_disponivel > 0 ENTÃO
```

```
            ESCREVER "\n--- ANÁLISE PARA RAMPA ---"

            // Análise de rampa
            angulo_rampa ← calcularAnguloRampa(altura, comprimento_disponivel)
            inclinacao ← (altura / comprimento_disponivel) * 100

            ESCREVER "Ângulo atual:", angulo_rampa, "graus"
            ESCREVER "Inclinação:", inclinacao, "%"

            comprimento_ideal_rampa ← altura / 0.0833
            ESCREVER "Comprimento ideal para rampa:", comprimento_ideal_rampa, "m"

            SE comprimento_disponivel >= comprimento_ideal_rampa ENTÃO
                ESCREVER "  Espaço SUFICIENTE para rampa conforme"
            SENÃO
                ESCREVER "  Espaço INSUFICIENTE para rampa conforme"
            FIM SE

            ESCREVER "\n--- ANÁLISE PARA ESCADA ---"

            // Análise de escada com ângulos ideais
            angulo_30 ← 30.0
            comprimento_escada_30 ← calcularComprimentoEscada(altura, angulo_30)
            projecao_30 ← altura / tan(angulo_30 * PI / 180)

            ESCREVER "Com ângulo de 30° (ideal):"
            ESCREVER "Comprimento da escada:", comprimento_escada_30, "m"
            ESCREVER "Projeção horizontal:", projecao_30, "m"

            SE projecao_30 <= comprimento_disponivel ENTÃO
                ESCREVER "  Espaço SUFICIENTE para escada ideal"
            SENÃO
                ESCREVER "  Espaço INSUFICIENTE para escada ideal"

                // Calcular ângulo máximo possível
                angulo_maximo ← atan(altura / comprimento_disponivel) * 180 / PI
                ESCREVER "Ângulo máximo possível:", angulo_maximo, "graus"
            FIM SE

            ESCREVER "\n--- RECOMENDAÇÃO FINAL ---"
            SE comprimento_disponivel >= comprimento_ideal_rampa ENTÃO
                ESCREVER "  RECOMENDAÇÃO: Construir rampa de acessibilidade"
            SENÃO SE projecao_30 <= comprimento_disponivel ENTÃO
                ESCREVER "  RECOMENDAÇÃO: Construir escada com ângulo ideal"
            SENÃO
                ESCREVER "  RECOMENDAÇÃO: Reavaliar projeto - espaço limitado"
            FIM SE
        SENÃO
            ESCREVER "Dados inválidos"
        FIM SE
```

```
        SENÃO SE opcao = 5 ENTÃO
            ESCREVER "=== CONVERSÕES TRIGONOMÉTRICAS ==="
            ESCREVER "1. Graus para radianos"
            ESCREVER "2. Radianos para graus"
            ESCREVER "3. Calcular seno/cosseno/tangente"
            LER tipo_conversao

            SE tipo_conversao = 1 ENTÃO
                ESCREVER "Digite o ângulo em graus:"
                LER angulo_graus
                radianos ← angulo_graus * PI / 180
                ESCREVER angulo_graus, "° =", radianos, "radianos"

            SENÃO SE tipo_conversao = 2 ENTÃO
                ESCREVER "Digite o ângulo em radianos:"
                LER angulo_radianos
                graus ← angulo_radianos * 180 / PI
                ESCREVER angulo_radianos, "rad =", graus, "°"

            SENÃO SE tipo_conversao = 3 ENTÃO
                ESCREVER "Digite o ângulo em graus:"
                LER angulo_graus
                radianos ← angulo_graus * PI / 180

                seno ← sin(radianos)
                cosseno ← cos(radianos)
                tangente ← tan(radianos)

                ESCREVER "Para ângulo de", angulo_graus, "°:"
                ESCREVER "Seno:", seno
                ESCREVER "Cosseno:", cosseno
                ESCREVER "Tangente:", tangente
            SENÃO
                ESCREVER "Opção inválida"
            FIM SE

        SENÃO SE opcao = 6 ENTÃO
            ESCREVER "Encerrando sistema de engenharia..."

        SENÃO
            ESCREVER "Opção inválida! Tente novamente."
        FIM SE

        // Pausa para visualização
        SE opcao  6 ENTÃO
            ESCREVER "\nPressione Enter para continuar..."
            LER pausa
        FIM SE
    FIM ENQUANTO
FIM
```

```
// FUNÇÃO: CALCULAR ÂNGULO DE RAMPA
FUNÇÃO calcularAnguloRampa(altura: real, comprimento: real): real
INÍCIO
    PI ← 3.14159265359

    // Calcular ângulo usando arcotangente
    angulo_radianos ← atan(altura / comprimento)

    // Converter para graus
    angulo_graus ← angulo_radianos * 180 / PI

    RETORNAR angulo_graus
FIM


// FUNÇÃO: CALCULAR COMPRIMENTO DE ESCADA
FUNÇÃO calcularComprimentoEscada(altura: real, angulo_graus: real): real
INÍCIO
    PI ← 3.14159265359

    // Converter ângulo para radianos
    angulo_radianos ← angulo_graus * PI / 180

    // Calcular comprimento usando seno: altura = comprimento * sen(ângulo)
    comprimento ← altura / sin(angulo_radianos)

    RETORNAR comprimento
FIM


// FUNÇÃO: VALIDAR ACESSIBILIDADE
FUNÇÃO validarAcessibilidade(altura: real, comprimento: real, angulo: real): logico
INÍCIO
    // Se ângulo foi fornecido, é validação de escada
    SE angulo > 0 ENTÃO
        // Validar escada: ângulo ideal entre 25° e 35°
        SE angulo >= 25 E angulo <= 35 ENTÃO
            RETORNAR VERDADEIRO
        SENÃO
            RETORNAR FALSO
        FIM SE
    SENÃO
        // Validar rampa: inclinação máxima 8.33% (NBR 9050)
        inclinacao_percentual ← (altura / comprimento) * 100

        SE inclinacao_percentual <= 8.33 ENTÃO
            RETORNAR VERDADEIRO
        SENÃO
            RETORNAR FALSO
        FIM SE
    FIM SE
```

```
FIM
```

O que código para implementação do pseudocódigo acima está disponível no GDB OnLine no endereço https://onlinegdb.com/_2rc38V9b.

D2: Uma empresa de engenharia acústica precisa de um sistema para calcular níveis de ruído e intensidade sonora em projetos urbanos. O sistema deve converter intensidades para decibéis, calcular atenuação sonora com a distância e somar níveis de ruído de múltiplas fontes. Crie um programa que processe medições acústicas para controle de poluição sonora.

Funções a implementar: - **Funções próprias**: `calcularDecibeis()`, `calcularAtenuacao()`, `somarRuidos()` - **Funções padrão**: `log10()`, `exp()`, `pow()`, `log()`

Entrada: Intensidade sonora, distância, múltiplos níveis de ruído **Saída**: Níveis em dB, atenuação por distância, soma logarítmica de ruídos

Cálculos específicos: - Conversão I→dB: dB = 10×log (I/I) - Atenuação: dB_dist = dB_orig - 20×log (d /d) - Soma de ruídos: dB_total = 10×log (10^(dB /10))

Solução:

Começando pelo pseudocódigo, teremos:

```
ALGORITMO SistemaEngenhariaAcustica

ENTRADA:
    intensidade, distancia, niveis_db: real

SAÍDA:
    decibeis, atenuacao, soma_ruidos: real

INÍCIO
    opcao ← 0
    I0 ← 1e-12  // Intensidade de referência: 10 ¹ ² W/m²

    ENQUANTO opcao   6 FAÇA

        // APRESENTAR MENU PRINCIPAL

        ESCREVER "=== SISTEMA DE ENGENHARIA ACÚSTICA ==="
        ESCREVER "1. Converter intensidade para decibéis"
        ESCREVER "2. Calcular atenuação sonora com distância"
        ESCREVER "3. Somar níveis de ruído (múltiplas fontes)"
        ESCREVER "4. Análise completa de projeto acústico"
        ESCREVER "5. Conversões logarítmicas"
        ESCREVER "6. Sair"
        ESCREVER "Escolha uma opção:"
        LER opcao


        // PROCESSAR OPÇÃO ESCOLHIDA

        SE opcao = 1 ENTÃO
            ESCREVER "=== CONVERSÃO INTENSIDADE → DECIBÉIS ==="
            ESCREVER "Digite a intensidade sonora (W/m²):"
            LER intensidade
```

```
            SE intensidade > 0 ENTÃO
                decibeis ← calcularDecibeis(intensidade)

                ESCREVER "Intensidade:", intensidade, "W/m²"
                ESCREVER "Nível sonoro:", decibeis, "dB"

                // Classificação do ruído
                SE decibeis <= 30 ENTÃO
                    ESCREVER "Classificação: Muito silencioso"
                SENÃO SE decibeis <= 50 ENTÃO
                    ESCREVER "Classificação: Silencioso"
                SENÃO SE decibeis <= 70 ENTÃO
                    ESCREVER "Classificação: Moderado"
                SENÃO SE decibeis <= 85 ENTÃO
                    ESCREVER "Classificação: Alto"
                SENÃO SE decibeis <= 100 ENTÃO
                    ESCREVER "Classificação: Muito alto"
                SENÃO
                    ESCREVER "Classificação: Perigoso - usar proteção auditiva"
                FIM SE

                // Limites legais
                SE decibeis > 55 ENTÃO
                    ESCREVER "  Acima do limite diurno residencial (55 dB)"
                FIM SE

                SE decibeis > 70 ENTÃO
                    ESCREVER "  Acima do limite comercial (70 dB)"
                FIM SE

            SENÃO
                ESCREVER "Erro: Intensidade deve ser positiva"
            FIM SE

        SENÃO SE opcao = 2 ENTÃO
            ESCREVER "=== CÁLCULO DE ATENUAÇÃO SONORA ==="
            ESCREVER "Digite o nível inicial (dB):"
            LER nivel_inicial
            ESCREVER "Digite a distância inicial (metros):"
            LER distancia_inicial
            ESCREVER "Digite a nova distância (metros):"
            LER distancia_final

            SE nivel_inicial >= 0 E distancia_inicial > 0 E distancia_final > 0 ENTÃO
                nivel_final ← calcularAtenuacao(nivel_inicial, distancia_inicial, distanc
                perda_sonora ← nivel_inicial - nivel_final

                ESCREVER "Nível inicial:", nivel_inicial, "dB a", distancia_inicial, "m"
                ESCREVER "Nível final:", nivel_final, "dB a", distancia_final, "m"
                ESCREVER "Perda sonora:", perda_sonora, "dB"
```

```
            // Análise da atenuação
            SE distancia_final > distancia_inicial ENTÃO
                fator_distancia ← distancia_final / distancia_inicial
                ESCREVER "Distância aumentou", fator_distancia, "vezes"
                ESCREVER "Redução esperada: ~", 20 * log10(fator_distancia), "dB"
            SENÃO
                ESCREVER "  Som ficou mais alto - distância diminuiu"
            FIM SE

        SENÃO
            ESCREVER "Erro: Dados devem ser positivos e válidos"
        FIM SE

    SENÃO SE opcao = 3 ENTÃO
        ESCREVER "=== SOMA DE NÍVEIS DE RUÍDO ==="
        ESCREVER "Quantas fontes de ruído você quer somar (2-5):"
        LER num_fontes

        SE num_fontes >= 2 E num_fontes <= 5 ENTÃO
            nivel1 ← 0
            nivel2 ← 0
            nivel3 ← 0
            nivel4 ← 0
            nivel5 ← 0

            ESCREVER "Digite o nível da fonte 1 (dB):"
            LER nivel1
            ESCREVER "Digite o nível da fonte 2 (dB):"
            LER nivel2

            SE num_fontes >= 3 ENTÃO
                ESCREVER "Digite o nível da fonte 3 (dB):"
                LER nivel3
            FIM SE

            SE num_fontes >= 4 ENTÃO
                ESCREVER "Digite o nível da fonte 4 (dB):"
                LER nivel4
            FIM SE

            SE num_fontes = 5 ENTÃO
                ESCREVER "Digite o nível da fonte 5 (dB):"
                LER nivel5
            FIM SE

            // Calcular soma logarítmica
            nivel_total ← somarRuidos(nivel1, nivel2, nivel3, nivel4, nivel5, num_fon

            ESCREVER "Fontes individuais:"
            ESCREVER "Fonte 1:", nivel1, "dB"
            ESCREVER "Fonte 2:", nivel2, "dB"
```

```
            SE num_fontes >= 3 ENTÃO
                ESCREVER "Fonte 3:", nivel3, "dB"
            FIM SE
            SE num_fontes >= 4 ENTÃO
                ESCREVER "Fonte 4:", nivel4, "dB"
            FIM SE
            SE num_fontes = 5 ENTÃO
                ESCREVER "Fonte 5:", nivel5, "dB"
            FIM SE

            ESCREVER "Nível total combinado:", nivel_total, "dB"

            // Análise da soma
            maior_individual ← nivel1
            SE nivel2 > maior_individual ENTÃO maior_individual ← nivel2 FIM SE
            SE nivel3 > maior_individual ENTÃO maior_individual ← nivel3 FIM SE
            SE nivel4 > maior_individual ENTÃO maior_individual ← nivel4 FIM SE
            SE nivel5 > maior_individual ENTÃO maior_individual ← nivel5 FIM SE

            incremento ← nivel_total - maior_individual
            ESCREVER "Incremento sobre a fonte mais alta:", incremento, "dB"

        SENÃO
            ESCREVER "Número de fontes deve estar entre 2 e 5"
        FIM SE

    SENÃO SE opcao = 4 ENTÃO
        ESCREVER "=== ANÁLISE COMPLETA DE PROJETO ACÚSTICO ==="
        ESCREVER "Digite a intensidade da fonte principal (W/m²):"
        LER intensidade_principal
        ESCREVER "Digite a distância da medição (metros):"
        LER distancia_medicao
        ESCREVER "Digite o nível de ruído de fundo (dB):"
        LER ruido_fundo

        SE intensidade_principal > 0 E distancia_medicao > 0 E ruido_fundo >= 0 ENTÃO
            ESCREVER "\n--- ANÁLISE DA FONTE PRINCIPAL ---"

            // Converter intensidade para dB
            nivel_fonte ← calcularDecibeis(intensidade_principal)
            ESCREVER "Nível da fonte:", nivel_fonte, "dB"

            // Calcular níveis em diferentes distâncias
            distancia_1m ← 1.0
            nivel_1m ← calcularAtenuacao(nivel_fonte, distancia_medicao, distancia_1m

            distancia_10m ← 10.0
            nivel_10m ← calcularAtenuacao(nivel_fonte, distancia_medicao, distancia_1

            distancia_100m ← 100.0
```

```
            nivel_100m ← calcularAtenuacao(nivel_fonte, distancia_medicao, distancia_

            ESCREVER "Nível a 1m:", nivel_1m, "dB"
            ESCREVER "Nível a 10m:", nivel_10m, "dB"
            ESCREVER "Nível a 100m:", nivel_100m, "dB"

            ESCREVER "\n--- ANÁLISE COM RUÍDO DE FUNDO ---"

            // Combinar fonte principal com ruído de fundo
            nivel_combinado ← somarRuidos(nivel_fonte, ruido_fundo, 0, 0, 0, 2)
            ESCREVER "Nível combinado (fonte + fundo):", nivel_combinado, "dB"

            mascaramento ← nivel_combinado - ruido_fundo
            ESCREVER "Efeito de mascaramento:", mascaramento, "dB"

            ESCREVER "\n--- CONFORMIDADE LEGAL ---"

            // Verificar limites legais (CONAMA 001/90)
            SE nivel_combinado <= 55 ENTÃO
                ESCREVER "  Conforme para área residencial ( 55 dB diurno)"
            SENÃO
                ESCREVER "  Não conforme para área residencial"
                reducao_necessaria ← nivel_combinado - 55
                ESCREVER "Redução necessária:", reducao_necessaria, "dB"
            FIM SE

            SE nivel_combinado <= 70 ENTÃO
                ESCREVER "  Conforme para área comercial ( 70 dB)"
            SENÃO
                ESCREVER "  Não conforme para área comercial"
            FIM SE

            ESCREVER "\n--- RECOMENDAÇÕES ---"

            SE nivel_combinado > 85 ENTÃO
                ESCREVER "  Recomenda-se proteção auditiva"
            FIM SE

            SE nivel_combinado > 55 ENTÃO
                distancia_segura ← distancia_medicao * pow(10, (nivel_combinado - 55)
                ESCREVER "  Distância mínima para 55dB:", distancia_segura, "m"
            FIM SE

        SENÃO
            ESCREVER "Dados inválidos"
        FIM SE

    SENÃO SE opcao = 5 ENTÃO
        ESCREVER "=== CONVERSÕES LOGARÍTMICAS ==="
        ESCREVER "1. dB para intensidade (W/m²)"
        ESCREVER "2. Intensidade para dB"
```

```
        ESCREVER "3. Calcular log  e logaritmo natural"
        LER tipo_conversao

        SE tipo_conversao = 1 ENTÃO
            ESCREVER "Digite o nível em dB:"
            LER decibeis

            // Converter dB para intensidade: I = I  × 10^(dB/10)
            intensidade ← I0 * pow(10, decibeis / 10)
            ESCREVER decibeis, "dB =", intensidade, "W/m²"

        SENÃO SE tipo_conversao = 2 ENTÃO
            ESCREVER "Digite a intensidade (W/m²):"
            LER intensidade

            SE intensidade > 0 ENTÃO
                decibeis ← calcularDecibeis(intensidade)
                ESCREVER intensidade, "W/m² =", decibeis, "dB"
            SENÃO
                ESCREVER "Intensidade deve ser positiva"
            FIM SE

        SENÃO SE tipo_conversao = 3 ENTÃO
            ESCREVER "Digite um número positivo:"
            LER numero

            SE numero > 0 ENTÃO
                log_10 ← log10(numero)
                log_natural ← log(numero)

                ESCREVER "Número:", numero
                ESCREVER "log (", numero, ") =", log_10
                ESCREVER "ln(", numero, ") =", log_natural
            SENÃO
                ESCREVER "Número deve ser positivo"
            FIM SE
        SENÃO
            ESCREVER "Opção inválida"
        FIM SE

    SENÃO SE opcao = 6 ENTÃO
        ESCREVER "Encerrando sistema de engenharia acústica..."

    SENÃO
        ESCREVER "Opção inválida! Tente novamente."
    FIM SE

    // Pausa para visualização
    SE opcao  6 ENTÃO
        ESCREVER "\nPressione Enter para continuar..."
        LER pausa
```

```
        FIM SE
    FIM ENQUANTO
FIM

// FUNÇÃO: CALCULAR DECIBÉIS
// dB = 10 × log (I/I )
FUNÇÃO calcularDecibeis(intensidade: real): real
INÍCIO
    I0 ← 1e-12  // Intensidade de referência: 10 ¹ ² W/m²

    // Calcular razão I/I
    razao ← intensidade / I0

    // Calcular dB = 10 × log (I/I )
    decibeis ← 10 * log10(razao)

    RETORNAR decibeis
FIM


// FUNÇÃO: CALCULAR ATENUAÇÃO
// dB_dist = dB_orig - 20 × log (d /d )
FUNÇÃO calcularAtenuacao(nivel_inicial: real, dist_inicial: real, dist_final: real): real
INÍCIO
    // Calcular razão das distâncias
    razao_distancias ← dist_final / dist_inicial

    // Calcular perda por distância: 20 × log (d /d )
    perda ← 20 * log10(razao_distancias)

    // Nível final = nível inicial - perda
    nivel_final ← nivel_inicial - perda

    RETORNAR nivel_final
FIM

// FUNÇÃO: SOMAR RUÍDOS
// dB_total = 10 × log ( 10^(dB /10))
FUNÇÃO somarRuidos(db1: real, db2: real, db3: real, db4: real, db5: real, num_fontes: int
INÍCIO
    // Converter cada dB para intensidade relativa
    intensidade1 ← pow(10, db1 / 10)
    intensidade2 ← pow(10, db2 / 10)

    soma_intensidades ← intensidade1 + intensidade2

    SE num_fontes >= 3 ENTÃO
        intensidade3 ← pow(10, db3 / 10)
        soma_intensidades ← soma_intensidades + intensidade3
    FIM SE
```

```
    SE num_fontes >= 4 ENTÃO
        intensidade4 ← pow(10, db4 / 10)
        soma_intensidades ← soma_intensidades + intensidade4
    FIM SE

    SE num_fontes = 5 ENTÃO
        intensidade5 ← pow(10, db5 / 10)
        soma_intensidades ← soma_intensidades + intensidade5
    FIM SE

    // Converter soma de volta para dB
    db_total ← 10 * log10(soma_intensidades)

    RETORNAR db_total
FIM
```

Cujo código C++ correspondente está disponivel no GDB OnLine no link https://onlinegdb.com/-ACks3zVu.

E2: Uma empresa de engenharia elétrica precisa de um sistema para calcular parâmetros de circuitos trifásicos, eficiência energética e análise de harmônicos. O sistema deve determinar as potências ativa (P), reativa (Q) e aparente (S), calcular o fator de potência (FP) e analisar a distorção harmônica total (THD) em instalações industriais.

Funções a implementar: - **Funções próprias**: `calcularPotenciaTrifasica()`, `calcularFatorPotencia()`, `analisarHarmonicos()` - **Funções padrão**: `sqrt()`, `pow()`, `sin()`, `cos()`, `atan()`, `fabs()`

Entrada: Tensão (V), corrente (I), ângulo de fase (ϕ), e as amplitudes das frequências harmônicas (H_h). **Saída**: Potências (P em kW, Q em kVAr, S em kVA), fator de potência (FP), $THD\%$, e status de conformidade com normas.

Cálculos específicos: - Potência ativa (P):

$$P = \sqrt{3} \cdot V \cdot I \cdot \cos(\phi)$$

- Potência reativa (Q):

$$Q = \sqrt{3} \cdot V \cdot I \cdot \sin(\phi)$$

- Fator de Potência (FP):

$$FP = \cos(\phi)$$

- Distorção Harmônica Total (THD):

$$THD = \frac{\sqrt{\sum_{h=2}^{n} H_h^2}}{H_1} \times 100\%$$

Solução:

Começando pelo pseudocódigo.

```
ALGORITMO SistemaEngenhariaEletrica

ENTRADA:
    tensao, corrente, angulo_fase, harmonicos: real

SAÍDA:
```

```
    potencias, fator_potencia, thd, conformidade: real

INÍCIO
    opcao ← 0
    PI ← 3.14159265359

    ENQUANTO opcao  6 FAÇA

        // BLOCO: APRESENTAR MENU PRINCIPAL
        ESCREVER "=== SISTEMA DE ENGENHARIA ELÉTRICA ==="
        ESCREVER "1. Calcular potências trifásicas"
        ESCREVER "2. Calcular fator de potência"
        ESCREVER "3. Analisar distorção harmônica (THD)"
        ESCREVER "4. Análise completa do sistema elétrico"
        ESCREVER "5. Conversões e utilidades elétricas"
        ESCREVER "6. Sair"
        ESCREVER "Escolha uma opção:"
        LER opcao


        // BLOCO: PROCESSAR OPÇÃO ESCOLHIDA
        SE opcao = 1 ENTÃO
            ESCREVER "=== CÁLCULO DE POTÊNCIAS TRIFÁSICAS ==="
            ESCREVER "Digite a tensão de linha (V):"
            LER tensao
            ESCREVER "Digite a corrente de linha (A):"
            LER corrente
            ESCREVER "Digite o ângulo de fase (graus):"
            LER angulo_graus

            SE tensao > 0 E corrente > 0 ENTÃO
                potencia_ativa ← calcularPotenciaTrifasica(tensao, corrente, angulo_graus
                potencia_reativa ← calcularPotenciaTrifasica(tensao, corrente, angulo_gra
                potencia_aparente ← calcularPotenciaTrifasica(tensao, corrente, angulo_gr

                ESCREVER "Tensão de linha:", tensao, "V"
                ESCREVER "Corrente de linha:", corrente, "A"
                ESCREVER "Ângulo de fase:", angulo_graus, "°"
                ESCREVER "Potência ativa (P):", potencia_ativa, "kW"
                ESCREVER "Potência reativa (Q):", potencia_reativa, "kVAr"
                ESCREVER "Potência aparente (S):", potencia_aparente, "kVA"

                // Classificação do sistema
                SE potencia_ativa < 100 ENTÃO
                    ESCREVER "Classificação: Sistema de baixa potência"
                SENÃO SE potencia_ativa < 1000 ENTÃO
                    ESCREVER "Classificação: Sistema de média potência"
                SENÃO
                    ESCREVER "Classificação: Sistema de alta potência"
                FIM SE
```

```
        SENÃO
            ESCREVER "Erro: Tensão e corrente devem ser positivas"
        FIM SE

    SENÃO SE opcao = 2 ENTÃO
        ESCREVER "=== CÁLCULO DE FATOR DE POTÊNCIA ==="
        ESCREVER "Digite a potência ativa (kW):"
        LER potencia_ativa
        ESCREVER "Digite a potência aparente (kVA):"
        LER potencia_aparente

        SE potencia_ativa > 0 E potencia_aparente > 0 E potencia_ativa <= potencia_ap
            fator_potencia ← calcularFatorPotencia(potencia_ativa, potencia_aparente)
            angulo_phi ← acos(fator_potencia) * 180 / PI

            ESCREVER "Potência ativa:", potencia_ativa, "kW"
            ESCREVER "Potência aparente:", potencia_aparente, "kVA"
            ESCREVER "Fator de potência:", fator_potencia
            ESCREVER "Ângulo  :", angulo_phi, "°"

            // Análise da qualidade
            SE fator_potencia >= 0.95 ENTÃO
                ESCREVER "  Excelente - Sem necessidade de correção"
            SENÃO SE fator_potencia >= 0.85 ENTÃO
                ESCREVER "  Bom - Considerar correção para grandes cargas"
            SENÃO SE fator_potencia >= 0.75 ENTÃO
                ESCREVER "  Regular - Recomenda-se correção"
            SENÃO
                ESCREVER "  Ruim - Correção urgente necessária"
            FIM SE

            // Cálculo de capacitores para correção
            SE fator_potencia < 0.95 ENTÃO
                potencia_reativa_atual ← potencia_ativa * tan(acos(fator_potencia))
                potencia_reativa_desejada ← potencia_ativa * tan(acos(0.95))
                capacitores_necessarios ← potencia_reativa_atual - potencia_reativa_d
                ESCREVER "Capacitores necessários para FP=0.95:", fabs(capacitores_ne
            FIM SE

        SENÃO
            ESCREVER "Erro: Dados inválidos (P deve ser   S)"
        FIM SE

    SENÃO SE opcao = 3 ENTÃO
        ESCREVER "=== ANÁLISE DE DISTORÇÃO HARMÔNICA ==="
        ESCREVER "Digite a amplitude da fundamental (A ou V):"
        LER fundamental
        ESCREVER "Quantos harmônicos analisar (2-10):"
        LER num_harmonicos

        SE fundamental > 0 E num_harmonicos >= 2 E num_harmonicos <= 10 ENTÃO
```

```
            thd ← analisarHarmonicos(fundamental, num_harmonicos)

            ESCREVER "Fundamental (H1):", fundamental
            ESCREVER "THD total:", thd, "%"

            // Classificação da qualidade
            SE thd <= 5 ENTÃO
                ESCREVER "  Excelente qualidade - THD  5%"
            SENÃO SE thd <= 8 ENTÃO
                ESCREVER "  Boa qualidade - THD  8%"
            SENÃO SE thd <= 15 ENTÃO
                ESCREVER "  Qualidade aceitável - THD  15%"
            SENÃO
                ESCREVER "  Má qualidade - THD > 15%"
            FIM SE

            // Normas brasileiras (PRODIST)
            SE thd <= 10 ENTÃO
                ESCREVER "  Conforme PRODIST - Módulo 8"
            SENÃO
                ESCREVER "  Não conforme PRODIST - Módulo 8"
            FIM SE

        SENÃO
            ESCREVER "Erro: Dados inválidos"
        FIM SE

    SENÃO SE opcao = 4 ENTÃO
        ESCREVER "=== ANÁLISE COMPLETA DO SISTEMA ELÉTRICO ==="
        ESCREVER "Digite a tensão de linha (V):"
        LER tensao
        ESCREVER "Digite a corrente de linha (A):"
        LER corrente
        ESCREVER "Digite o ângulo de fase (graus):"
        LER angulo_graus
        ESCREVER "Digite a amplitude da fundamental para THD:"
        LER fundamental

        SE tensao > 0 E corrente > 0 E fundamental > 0 ENTÃO
            ESCREVER "\n--- ANÁLISE DE POTÊNCIAS ---"

            // Cálculos de potência
            potencia_ativa ← calcularPotenciaTrifasica(tensao, corrente, angulo_graus
            potencia_reativa ← calcularPotenciaTrifasica(tensao, corrente, angulo_gra
            potencia_aparente ← calcularPotenciaTrifasica(tensao, corrente, angulo_gr

            ESCREVER "Potência ativa:", potencia_ativa, "kW"
            ESCREVER "Potência reativa:", potencia_reativa, "kVAr"
            ESCREVER "Potência aparente:", potencia_aparente, "kVA"

            // Fator de potência
```

```
            fator_potencia ← calcularFatorPotencia(potencia_ativa, potencia_aparente)
            ESCREVER "Fator de potência:", fator_potencia

            ESCREVER "\n--- ANÁLISE DE QUALIDADE ---"

            // THD com 5 harmônicos por padrão
            thd ← analisarHarmonicos(fundamental, 5)
            ESCREVER "THD (5 harmônicos):", thd, "%"

            ESCREVER "\n--- ANÁLISE ECONÔMICA ---"

            // Custo estimado por perdas
            perda_percentual ← (1 - fator_potencia) * 100
            ESCREVER "Perdas por baixo FP:", perda_percentual, "%"

            // Estimativa de custo mensal (R$ 0,50/kWh)
            custo_kwh ← 0.50
            horas_mes ← 720  // 30 dias × 24 horas
            custo_mensal ← potencia_ativa * horas_mes * custo_kwh
            ESCREVER "Custo mensal estimado: R$", custo_mensal

            ESCREVER "\n--- CONFORMIDADE GERAL ---"

            problemas ← 0

            SE fator_potencia < 0.85 ENTÃO
                problemas ← problemas + 1
                ESCREVER "  Fator de potência baixo"
            FIM SE

            SE thd > 10 ENTÃO
                problemas ← problemas + 1
                ESCREVER "  THD acima do limite PRODIST"
            FIM SE

            SE problemas = 0 ENTÃO
                ESCREVER "  Sistema em conformidade total"
            SENÃO
                ESCREVER "  Sistema com", problemas, "não conformidade(s)"
            FIM SE

        SENÃO
            ESCREVER "Dados inválidos"
        FIM SE

    SENÃO SE opcao = 5 ENTÃO
        ESCREVER "=== CONVERSÕES E UTILIDADES ELÉTRICAS ==="
        ESCREVER "1. Converter potência (W  HP  CV)"
        ESCREVER "2. Calcular resistência por temperatura"
        ESCREVER "3. Converter ângulos (graus  radianos)"
        LER tipo_conversao
```

```
SE tipo_conversao = 1 ENTÃO
    ESCREVER "Digite a potência em Watts:"
    LER potencia_watts

    SE potencia_watts > 0 ENTÃO
        hp ← potencia_watts / 745.7
        cv ← potencia_watts / 735.5

        ESCREVER potencia_watts, "W ="
        ESCREVER "  ", hp, "HP"
        ESCREVER "  ", cv, "CV"
    SENÃO
        ESCREVER "Potência deve ser positiva"
    FIM SE

SENÃO SE tipo_conversao = 2 ENTÃO
    ESCREVER "Digite a resistência a 20°C (Ω):"
    LER r20
    ESCREVER "Digite a temperatura atual (°C):"
    LER temperatura

    // Coeficiente de temperatura do cobre
    alfa ← 0.00393
    r_temp ← r20 * (1 + alfa * (temperatura - 20))

    ESCREVER "Resistência a", temperatura, "°C:", r_temp, "Ω"
    variacao ← ((r_temp - r20) / r20) * 100
    ESCREVER "Variação:", variacao, "%"

SENÃO SE tipo_conversao = 3 ENTÃO
    ESCREVER "1. Graus para radianos"
    ESCREVER "2. Radianos para graus"
    LER direcao

    SE direcao = 1 ENTÃO
        ESCREVER "Digite o ângulo em graus:"
        LER graus
        radianos ← graus * PI / 180
        ESCREVER graus, "° =", radianos, "rad"
    SENÃO SE direcao = 2 ENTÃO
        ESCREVER "Digite o ângulo em radianos:"
        LER radianos
        graus ← radianos * 180 / PI
        ESCREVER radianos, "rad =", graus, "°"
    SENÃO
        ESCREVER "Opção inválida"
    FIM SE
SENÃO
    ESCREVER "Opção inválida"
FIM SE
```

```
        SENÃO SE opcao = 6 ENTÃO
            ESCREVER "Encerrando sistema de engenharia elétrica..."

        SENÃO
            ESCREVER "Opção inválida! Tente novamente."
        FIM SE

        // Pausa para visualização
        SE opcao  6 ENTÃO
            ESCREVER "\nPressione Enter para continuar..."
            LER pausa
        FIM SE
    FIM ENQUANTO
FIM

// FUNÇÃO: CALCULAR POTÊNCIA TRIFÁSICA
// P = √3 × V × I × cos( )  [tipo 1]
// Q = √3 × V × I × sin( )  [tipo 2]
// S = √3 × V × I           [tipo 3]

FUNÇÃO calcularPotenciaTrifasica(tensao: real, corrente: real, angulo_graus: real, tipo:
INÍCIO
    PI ← 3.14159265359

    // Converter ângulo para radianos
    angulo_rad ← angulo_graus * PI / 180

    // Fator √3
    raiz_tres ← sqrt(3)

    // Potência base em VA
    potencia_base ← raiz_tres * tensao * corrente

    SE tipo = 1 ENTÃO
        // Potência ativa (kW)
        potencia_ativa ← potencia_base * cos(angulo_rad)
        RETORNAR potencia_ativa / 1000
    SENÃO SE tipo = 2 ENTÃO
        // Potência reativa (kVAr)
        potencia_reativa ← potencia_base * sin(angulo_rad)
        RETORNAR potencia_reativa / 1000
    SENÃO SE tipo = 3 ENTÃO
        // Potência aparente (kVA)
        RETORNAR potencia_base / 1000
    SENÃO
        RETORNAR 0
    FIM SE
FIM

// FUNÇÃO: CALCULAR FATOR DE POTÊNCIA
```

```
// FP = P / S = cos( )
FUNÇÃO calcularFatorPotencia(potencia_ativa: real, potencia_aparente: real): real
INÍCIO
    SE potencia_aparente = 0 ENTÃO
        RETORNAR 0
    FIM SE

    fator_potencia ← potencia_ativa / potencia_aparente

    // Garantir que está entre 0 e 1
    SE fator_potencia > 1 ENTÃO
        fator_potencia ← 1
    FIM SE

    SE fator_potencia < 0 ENTÃO
        fator_potencia ← 0
    FIM SE

    RETORNAR fator_potencia
FIM

// FUNÇÃO: ANALISAR HARMÔNICOS
// THD = √( H²  )/H  × 100%
FUNÇÃO analisarHarmonicos(fundamental: real, num_harmonicos: inteiro): real
INÍCIO
    soma_quadrados ← 0

    // Processar cada harmônico sequencialmente
    PARA i DE 2 ATÉ num_harmonicos FAÇA
        ESCREVER "Digite a amplitude do", i, "º harmônico:"
        LER harmonica

        SE harmonica >= 0 ENTÃO
            quadrado ← pow(harmonica, 2)
            soma_quadrados ← soma_quadrados + quadrado

            // Mostrar porcentagem individual
            percentual_individual ← (harmonica / fundamental) * 100
            ESCREVER "H", i, ":", percentual_individual, "% da fundamental"
        FIM SE
    FIM PARA

    // Calcular THD
    SE fundamental > 0 ENTÃO
        raiz_soma ← sqrt(soma_quadrados)
        thd ← (raiz_soma / fundamental) * 100
        RETORNAR thd
    SENÃO
        RETORNAR 0
    FIM SE
FIM
```

O código C++ correspondente está disponível no GDB OnLine no link https://onlinegdb.com/hWZ7dfvok.

F2: Um observatório astronômico precisa de um sistema para calcular posições de corpos celestes, distâncias interplanetárias e janelas de lançamento de missões espaciais. O sistema deve converter coordenadas celestes (e.g., de (α, δ) para (x, y, z)), calcular órbitas elípticas e determinar tempos de trânsito (t) usando mecânica orbital.

Funções a implementar: - **Funções próprias**: `calcularPosicaoOrbital()`, `determinarJanelaLancamento()`, `converterCoordenadas()` - **Funções padrão**: `sin()`, `cos()`, `tan()`, `asin()`, `acos()`, `sqrt()`, `pow()`, `fmod()`

Entrada: Elementos orbitais (e.g., $a, e, i, \Omega, \omega, M$), coordenadas, tempo (t), velocidade inicial (v_0). **Saída**: Posições celestes, distâncias (d) em UA, velocidade de escape (v_e), delta-V (Δv).

Cálculos específicos: - Equação de Kepler (relaciona anomalia média M com anomalia excêntrica E e excentricidade e):

$$M = E - e\sin(E)$$

- Velocidade orbital (Equação *vis-viva*), onde v é a velocidade, μ o parâmetro gravitacional, r a distância radial e a o semi-eixo maior:

$$v = \sqrt{\mu\left(\frac{2}{r} - \frac{1}{a}\right)}$$

Solução:
Começando com o pseudocódigo:

```
ALGORITMO SistemaObservatorioAstronomico

ENTRADA:
    elementos_orbitais, coordenadas, tempo, velocidades: real

SAÍDA:
    posicoes_celestes, distancias_ua, velocidades_escape, delta_v: real

INÍCIO
    opcao ← 0
    PI ← 3.14159265359
    MU_SOL ← 1.32712440018e20  //  solar em m³/s²
    UA ← 149597870700.0        // Unidade Astronômica em metros

    ENQUANTO opcao  6 FAÇA

        // BLOCO: APRESENTAR MENU PRINCIPAL
        ESCREVER "=== SISTEMA DE OBSERVATÓRIO ASTRONÔMICO ==="
        ESCREVER "1. Calcular posição orbital de corpo celeste"
        ESCREVER "2. Determinar janela de lançamento"
        ESCREVER "3. Converter coordenadas celestes"
        ESCREVER "4. Análise completa de missão espacial"
        ESCREVER "5. Conversões astronômicas"
        ESCREVER "6. Sair"
        ESCREVER "Escolha uma opção:"
        LER opcao
```

```
// BLOCO: PROCESSAR OPÇÃO ESCOLHIDA
SE opcao = 1 ENTÃO
    ESCREVER "=== CÁLCULO DE POSIÇÃO ORBITAL ==="
    ESCREVER "Digite o semi-eixo maior (UA):"
    LER semi_eixo_maior
    ESCREVER "Digite a excentricidade (0-1):"
    LER excentricidade
    ESCREVER "Digite a anomalia média (graus):"
    LER anomalia_media_graus
    ESCREVER "Digite a inclinação (graus):"
    LER inclinacao_graus

    SE semi_eixo_maior > 0 E excentricidade >= 0 E excentricidade < 1 ENTÃO
        posicao_orbital ← calcularPosicaoOrbital(semi_eixo_maior, excentricidade,

        ESCREVER "Semi-eixo maior:", semi_eixo_maior, "UA"
        ESCREVER "Excentricidade:", excentricidade
        ESCREVER "Anomalia média:", anomalia_media_graus, "graus"
        ESCREVER "Distância heliocêntrica:", posicao_orbital, "UA"

        // Calcular período orbital (3ª Lei de Kepler)
        periodo_anos ← sqrt(pow(semi_eixo_maior, 3))
        ESCREVER "Período orbital:", periodo_anos, "anos"

        // Velocidade no periélio e afélio
        r_perielio ← semi_eixo_maior * (1 - excentricidade)
        r_afelio ← semi_eixo_maior * (1 + excentricidade)

        // Converter UA para metros para cálculos de velocidade
        a_metros ← semi_eixo_maior * UA
        r_per_metros ← r_perielio * UA
        r_af_metros ← r_afelio * UA

        v_perielio ← sqrt(MU_SOL * (2 / r_per_metros - 1 / a_metros))
        v_afelio ← sqrt(MU_SOL * (2 / r_af_metros - 1 / a_metros))

        ESCREVER "Velocidade no periélio:", v_perielio / 1000, "km/s"
        ESCREVER "Velocidade no afélio:", v_afelio / 1000, "km/s"

    SENÃO
        ESCREVER "Erro: Parâmetros orbitais inválidos"
    FIM SE

SENÃO SE opcao = 2 ENTÃO
    ESCREVER "=== DETERMINAÇÃO DE JANELA DE LANÇAMENTO ==="
    ESCREVER "Digite a distância Terra-destino atual (UA):"
    LER distancia_atual
    ESCREVER "Digite a velocidade relativa (km/s):"
    LER velocidade_relativa
    ESCREVER "Digite o período sinódico do destino (dias):"
```

```
            LER periodo_sinodico

            SE distancia_atual > 0 E velocidade_relativa > 0 E periodo_sinodico > 0 ENTÃO
                janela_otima ← determinarJanelaLancamento(distancia_atual, velocidade_rel

                ESCREVER "Distância atual:", distancia_atual, "UA"
                ESCREVER "Velocidade relativa:", velocidade_relativa, "km/s"
                ESCREVER "Tempo de viagem ótimo:", janela_otima, "dias"

                // Calcular delta-V necessário
                distancia_metros ← distancia_atual * UA
                delta_v_ms ← sqrt(2 * MU_SOL / distancia_metros)
                delta_v_escape_terra ← 11200  // m/s
                delta_v_total ← delta_v_ms + delta_v_escape_terra

                ESCREVER "Delta-V de escape:", delta_v_escape_terra / 1000, "km/s"
                ESCREVER "Delta-V interplanetário:", delta_v_ms / 1000, "km/s"
                ESCREVER "Delta-V total:", delta_v_total / 1000, "km/s"

                // Próxima janela
                proxima_janela ← janela_otima + periodo_sinodico
                ESCREVER "Próxima janela em:", proxima_janela, "dias"

            SENÃO
                ESCREVER "Erro: Parâmetros de missão inválidos"
            FIM SE

        SENÃO SE opcao = 3 ENTÃO
            ESCREVER "=== CONVERSÃO DE COORDENADAS CELESTES ==="
            ESCREVER "1. Equatoriais ( , ) para cartesianas (x,y,z)"
            ESCREVER "2. Cartesianas (x,y,z) para equatoriais ( , )"
            ESCREVER "3. Eclípticas para equatoriais"
            LER tipo_conversao

            SE tipo_conversao = 1 ENTÃO
                ESCREVER "Digite a ascensão reta   (horas):"
                LER ascensao_reta_horas
                ESCREVER "Digite a declinação   (graus):"
                LER declinacao_graus
                ESCREVER "Digite a distância (UA):"
                LER distancia

                SE distancia > 0 ENTÃO
                    x_coord, y_coord, z_coord ← converterCoordenadas(ascensao_reta_horas,

                    ESCREVER "Coordenadas equatoriais:"
                    ESCREVER "  =", ascensao_reta_horas, "h"
                    ESCREVER "  =", declinacao_graus, "°"
                    ESCREVER "Coordenadas cartesianas:"
                    ESCREVER "x =", x_coord, "UA"
                    ESCREVER "y =", y_coord, "UA"
```

```
            ESCREVER "z =", z_coord, "UA"
        SENÃO
            ESCREVER "Distância deve ser positiva"
        FIM SE

    SENÃO SE tipo_conversao = 2 ENTÃO
        ESCREVER "Digite a coordenada x (UA):"
        LER x_coord
        ESCREVER "Digite a coordenada y (UA):"
        LER y_coord
        ESCREVER "Digite a coordenada z (UA):"
        LER z_coord

        ascensao_reta, declinacao, distancia ← converterCoordenadas(x_coord, y_co

        ESCREVER "Coordenadas cartesianas:"
        ESCREVER "x =", x_coord, "UA"
        ESCREVER "y =", y_coord, "UA"
        ESCREVER "z =", z_coord, "UA"
        ESCREVER "Coordenadas equatoriais:"
        ESCREVER "  =", ascensao_reta, "h"
        ESCREVER "  =", declinacao, "°"
        ESCREVER "Distância =", distancia, "UA"

    SENÃO SE tipo_conversao = 3 ENTÃO
        ESCREVER "Digite a longitude eclíptica (graus):"
        LER longitude_ecl
        ESCREVER "Digite a latitude eclíptica (graus):"
        LER latitude_ecl

        obliquidade ← 23.43929  // Obliquidade da eclíptica

        // Conversão eclíptica para equatorial
        long_rad ← longitude_ecl * PI / 180
        lat_rad ← latitude_ecl * PI / 180
        obl_rad ← obliquidade * PI / 180

        ascensao_reta_rad ← atan2(sin(long_rad) * cos(obl_rad) - tan(lat_rad) * s
        declinacao_rad ← asin(sin(lat_rad) * cos(obl_rad) + cos(lat_rad) * sin(ob

        ascensao_reta_final ← ascensao_reta_rad * 180 / PI / 15  // Converter par
        declinacao_final ← declinacao_rad * 180 / PI

        SE ascensao_reta_final < 0 ENTÃO
            ascensao_reta_final ← ascensao_reta_final + 24
        FIM SE

        ESCREVER "Coordenadas eclípticas:"
        ESCREVER "Longitude =", longitude_ecl, "°"
        ESCREVER "Latitude =", latitude_ecl, "°"
        ESCREVER "Coordenadas equatoriais:"
```

```
            ESCREVER "  =", ascensao_reta_final, "h"
            ESCREVER "  =", declinacao_final, "°"
        SENÃO
            ESCREVER "Opção inválida"
        FIM SE

    SENÃO SE opcao = 4 ENTÃO
        ESCREVER "=== ANÁLISE COMPLETA DE MISSÃO ESPACIAL ==="
        ESCREVER "Digite o planeta de destino (1=Marte, 2=Júpiter, 3=Saturno):"
        LER planeta_destino
        ESCREVER "Digite a massa da espaçonave (kg):"
        LER massa_nave

        SE planeta_destino >= 1 E planeta_destino <= 3 E massa_nave > 0 ENTÃO
            ESCREVER "\n--- PARÂMETROS DO DESTINO ---"

            // Definir parâmetros planetários
            SE planeta_destino = 1 ENTÃO
                // Marte
                nome_planeta ← "Marte"
                semi_eixo_planeta ← 1.52
                excentricidade_planeta ← 0.0934
                periodo_planeta ← 687
                massa_planeta ← 6.39e23
                raio_planeta ← 3389.5e3

            SENÃO SE planeta_destino = 2 ENTÃO
                // Júpiter
                nome_planeta ← "Júpiter"
                semi_eixo_planeta ← 5.20
                excentricidade_planeta ← 0.0489
                periodo_planeta ← 4333
                massa_planeta ← 1.898e27
                raio_planeta ← 69911e3

            SENÃO
                // Saturno
                nome_planeta ← "Saturno"
                semi_eixo_planeta ← 9.54
                excentricidade_planeta ← 0.0565
                periodo_planeta ← 10759
                massa_planeta ← 5.683e26
                raio_planeta ← 58232e3
            FIM SE

            ESCREVER "Destino:", nome_planeta
            ESCREVER "Semi-eixo maior:", semi_eixo_planeta, "UA"
            ESCREVER "Período orbital:", periodo_planeta, "dias"

            ESCREVER "\n--- CÁLCULOS DE TRAJETÓRIA ---"
```

```
            // Calcular transferência de Hohmann
            semi_eixo_transferencia ← (1.0 + semi_eixo_planeta) / 2
            periodo_transferencia ← sqrt(pow(semi_eixo_transferencia, 3)) * 365.25
            tempo_viagem ← periodo_transferencia / 2

            ESCREVER "Semi-eixo da órbita de transferência:", semi_eixo_transferencia
            ESCREVER "Tempo de viagem:", tempo_viagem, "dias"

            // Velocidades características
            v_terra_orbital ← 29.78  // km/s
            v_escape_terra ← 11.2    // km/s

            // Velocidade no afélio da transferência
            a_transf_metros ← semi_eixo_transferencia * UA
            r_destino_metros ← semi_eixo_planeta * UA
            v_chegada ← sqrt(MU_SOL * (2 / r_destino_metros - 1 / a_transf_metros))
            v_chegada_kms ← v_chegada / 1000

            ESCREVER "Velocidade de chegada:", v_chegada_kms, "km/s"

            // Delta-V total
            delta_v_partida ← v_escape_terra + 3.6  // 3.6 km/s para escape da órbita
            delta_v_chegada ← fabs(v_chegada_kms - semi_eixo_planeta * 6.28)  // Apro
            delta_v_missao ← delta_v_partida + delta_v_chegada

            ESCREVER "Delta-V de partida:", delta_v_partida, "km/s"
            ESCREVER "Delta-V de chegada:", delta_v_chegada, "km/s"
            ESCREVER "Delta-V total da missão:", delta_v_missao, "km/s"

            ESCREVER "\n--- ANÁLISE ENERGÉTICA ---"

            // Energia específica
            energia_especifica ← pow(delta_v_missao * 1000, 2) / 2  // J/kg
            energia_total ← energia_especifica * massa_nave / 1e9   // GJ

            ESCREVER "Energia específica:", energia_especifica / 1e6, "MJ/kg"
            ESCREVER "Energia total necessária:", energia_total, "GJ"

            // Período sinódico para próxima oportunidade
            periodo_sinodico_calc ← fabs(365.25 * periodo_planeta / (periodo_planeta
            ESCREVER "Próxima janela em:", periodo_sinodico_calc, "dias"

        SENÃO
            ESCREVER "Parâmetros inválidos"
        FIM SE

    SENÃO SE opcao = 5 ENTÃO
        ESCREVER "=== CONVERSÕES ASTRONÔMICAS ==="
        ESCREVER "1. Anos-luz para UA e parsecs"
        ESCREVER "2. Magnitudes estelares"
        ESCREVER "3. Tempos astronômicos"
```

```
LER tipo_conversao_astro

SE tipo_conversao_astro = 1 ENTÃO
    ESCREVER "Digite a distância em anos-luz:"
    LER anos_luz

    SE anos_luz > 0 ENTÃO
        ua_equiv ← anos_luz * 63241.1  // 1 ano-luz = 63241.1 UA
        parsecs ← anos_luz / 3.26156   // 1 parsec = 3.26156 anos-luz

        ESCREVER anos_luz, "anos-luz ="
        ESCREVER "  ", ua_equiv, "UA"
        ESCREVER "  ", parsecs, "parsecs"
    SENÃO
        ESCREVER "Distância deve ser positiva"
    FIM SE

SENÃO SE tipo_conversao_astro = 2 ENTÃO
    ESCREVER "Digite a magnitude aparente:"
    LER mag_aparente
    ESCREVER "Digite a distância em parsecs:"
    LER distancia_pc

    SE distancia_pc > 0 ENTÃO
        modulo_distancia ← 5 * log10(distancia_pc) - 5
        magnitude_absoluta ← mag_aparente - modulo_distancia

        ESCREVER "Magnitude aparente:", mag_aparente
        ESCREVER "Módulo de distância:", modulo_distancia
        ESCREVER "Magnitude absoluta:", magnitude_absoluta
    SENÃO
        ESCREVER "Distância deve ser positiva"
    FIM SE

SENÃO SE tipo_conversao_astro = 3 ENTÃO
    ESCREVER "Digite o tempo em dias julianos:"
    LER dias_julianos

    // Converter para anos desde J2000
    anos_j2000 ← (dias_julianos - 2451545.0) / 365.25

    ESCREVER "Dias julianos:", dias_julianos
    ESCREVER "Anos desde J2000.0:", anos_j2000

    // Equinócio vernal aproximado
    equinocio ← fmod(anos_j2000, 1.0) * 365.25
    SE equinocio < 0 ENTÃO equinocio ← equinocio + 365.25 FIM SE

    ESCREVER "Dia do ano aproximado:", equinocio
SENÃO
    ESCREVER "Opção inválida"
```

```
            FIM SE

        SENÃO SE opcao = 6 ENTÃO
            ESCREVER "Encerrando sistema de observatório astronômico..."

        SENÃO
            ESCREVER "Opção inválida! Tente novamente."
        FIM SE

        // Pausa para visualização
        SE opcao  6 ENTÃO
            ESCREVER "\nPressione Enter para continuar..."
            LER pausa
        FIM SE
    FIM ENQUANTO
FIM

// FUNÇÃO: CALCULAR POSIÇÃO ORBITAL
// Resolve equação de Kepler: M = E - e*sin(E)
FUNÇÃO calcularPosicaoOrbital(semi_eixo: real, excentricidade: real, anomalia_media_graus
INÍCIO
    PI ← 3.14159265359

    // Converter anomalia média para radianos
    anomalia_media_rad ← anomalia_media_graus * PI / 180

    // Resolver equação de Kepler iterativamente
    anomalia_excentrica ← anomalia_media_rad  // Estimativa inicial

    PARA i DE 1 ATÉ 10 FAÇA
        anomalia_excentrica_nova ← anomalia_media_rad + excentricidade * sin(anomalia_exc

        // Verificar convergência
        diferenca ← fabs(anomalia_excentrica_nova - anomalia_excentrica)
        SE diferenca < 1e-8 ENTÃO
            PARE
        FIM SE

        anomalia_excentrica ← anomalia_excentrica_nova
    FIM PARA

    // Calcular distância heliocêntrica
    distancia_heliocentrica ← semi_eixo * (1 - excentricidade * cos(anomalia_excentrica))

    RETORNAR distancia_heliocentrica
FIM

// FUNÇÃO: DETERMINAR JANELA DE LANÇAMENTO
// Calcula tempo ótimo considerando mecânica orbital
FUNÇÃO determinarJanelaLancamento(distancia_ua: real, velocidade_kms: real, periodo_sinod
INÍCIO
```

```
    // Converter distância para metros e velocidade para m/s
    UA ← 149597870700.0
    distancia_metros ← distancia_ua * UA
    velocidade_ms ← velocidade_kms * 1000

    // Tempo de viagem básico
    tempo_viagem_segundos ← distancia_metros / velocidade_ms
    tempo_viagem_dias ← tempo_viagem_segundos / 86400  // segundos por dia

    // Ajuste para órbita elíptica (transferência de Hohmann)
    fator_correcao ← sqrt(2) * 0.5  // Aproximação para transferência elíptica
    tempo_otimizado ← tempo_viagem_dias * fator_correcao

    // Considerar período sinódico para janela
    SE tempo_otimizado > periodo_sinodico / 2 ENTÃO
        tempo_otimizado ← periodo_sinodico / 2
    FIM SE

    RETORNAR tempo_otimizado
FIM

// FUNÇÃO: CONVERTER COORDENADAS
// Tipo 1: ( , ,r) -> (x,y,z)
// Tipo 2: (x,y,z) -> ( , ,r)
FUNÇÃO converterCoordenadas(param1: real, param2: real, param3: real, tipo: inteiro): rea
INÍCIO
    PI ← 3.14159265359

    SE tipo = 1 ENTÃO
        // Equatoriais para cartesianas
        ascensao_reta_horas ← param1
        declinacao_graus ← param2
        distancia ← param3

        // Converter para radianos
        alpha_rad ← ascensao_reta_horas * 15 * PI / 180  // 15°/hora
        delta_rad ← declinacao_graus * PI / 180

        // Coordenadas cartesianas
        x ← distancia * cos(delta_rad) * cos(alpha_rad)
        y ← distancia * cos(delta_rad) * sin(alpha_rad)
        z ← distancia * sin(delta_rad)

        RETORNAR x, y, z

    SENÃO SE tipo = 2 ENTÃO
        // Cartesianas para equatoriais
        x ← param1
        y ← param2
        z ← param3
```

```
        // Distância
        distancia ← sqrt(x * x + y * y + z * z)

        // Ascensão reta
        alpha_rad ← atan2(y, x)
        SE alpha_rad < 0 ENTÃO
            alpha_rad ← alpha_rad + 2 * PI
        FIM SE
        ascensao_reta_horas ← alpha_rad * 180 / PI / 15

        // Declinação
        delta_rad ← asin(z / distancia)
        declinacao_graus ← delta_rad * 180 / PI

        RETORNAR ascensao_reta_horas, declinacao_graus, distancia

    SENÃO
        RETORNAR 0, 0, 0
    FIM SE
FIM
```

A partir desse pseudocódigo, podemos implementar o código em C++ que está disponível em https://onlinegdb.com/acME7DYnz.

G2: Um laboratório farmacêutico precisa de um sistema para análise estatística de dados experimentais, cálculo de concentrações moleculares e modelagem farmacocinética. O sistema deve processar dados sequencialmente, calcular estatísticas descritivas (como média, μ, e desvio padrão, σ) e determinar parâmetros de absorção/eliminação de medicamentos.

Funções a implementar: - **Funções próprias**: `calcularEstatisticas()`, `modelarFarmacocinetica()`, `determinarBioequivalencia()` - **Funções padrão**: `exp()`, `log()`, `pow()`, `sqrt()`, `fabs()`, `floor()`, `ceil()`

Entrada: Concentrações sanguíneas sequenciais (C), tempos de coleta (t), doses administradas ($Dose$). **Saída**: Média (μ), desvio padrão (σ), meia-vida ($t_{1/2}$), clearance (CL), área sob a curva (AUC), resultado de bioequivalência.

Cálculos específicos: - Modelo monocompartimental, onde $C(t)$ é a concentração no tempo t, C_0 é a concentração inicial e k é a constante de eliminação:

$$C(t) = C_0 \cdot e^{-k \cdot t}$$

- Área Sob a Curva (AUC), calculada numericamente (e.g., pela regra dos trapézios):

$$AUC = \int_0^{\infty} C(t)\, dt$$

- Clearance (CL), ou depuração, que relaciona a dose com a área sob a curva:

$$CL = \frac{Dose}{AUC}$$

- Estatísticas calculadas sequencialmente (sem o uso de arrays para armazenamento completo dos dados).

Solução:
Primeiro o pseudocódigo para a função de cálculo de estatísticas e farmacocinética:

```
ALGORITMO SistemaLaboratorioFarmaceutico

ENTRADA:
    concentracoes, tempos, doses: real

SAÍDA:
    media, desvio_padrao, meia_vida, clearance, auc, bioequivalencia: real

INÍCIO
    opcao ← 0

    ENQUANTO opcao  6 FAÇA

        // BLOCO: APRESENTAR MENU PRINCIPAL

        ESCREVER "=== SISTEMA DE LABORATÓRIO FARMACÊUTICO ==="
        ESCREVER "1. Calcular estatísticas descritivas"
        ESCREVER "2. Modelar farmacocinética"
        ESCREVER "3. Determinar bioequivalência"
        ESCREVER "4. Análise completa de medicamento"
        ESCREVER "5. Cálculos auxiliares"
        ESCREVER "6. Sair"
        ESCREVER "Escolha uma opção:"
        LER opcao


        // BLOCO: PROCESSAR OPÇÃO ESCOLHIDA

        SE opcao = 1 ENTÃO
            ESCREVER "=== CÁLCULO DE ESTATÍSTICAS DESCRITIVAS ==="
            ESCREVER "Quantos dados você tem (2-20):"
            LER num_dados

            SE num_dados >= 2 E num_dados <= 20 ENTÃO
                media, desvio_padrao ← calcularEstatisticas(num_dados)

                ESCREVER "Número de amostras:", num_dados
                ESCREVER "Média ( ):", media
                ESCREVER "Desvio padrão ( ):", desvio_padrao
                ESCREVER "Coeficiente de variação:", (desvio_padrao / media) * 100, "%"

                // Classificação da variabilidade
                cv ← (desvio_padrao / media) * 100
                SE cv <= 5 ENTÃO
                    ESCREVER "Variabilidade: Muito baixa (excelente)"
                SENÃO SE cv <= 10 ENTÃO
                    ESCREVER "Variabilidade: Baixa (boa)"
                SENÃO SE cv <= 20 ENTÃO
                    ESCREVER "Variabilidade: Moderada (aceitável)"
                SENÃO
                    ESCREVER "Variabilidade: Alta (revisar protocolo)"
```

```
            FIM SE

        SENÃO
            ESCREVER "Número de dados deve estar entre 2 e 20"
        FIM SE

    SENÃO SE opcao = 2 ENTÃO
        ESCREVER "=== MODELAGEM FARMACOCINÉTICA ==="
        ESCREVER "Digite a concentração inicial C0 (mg/L):"
        LER c0
        ESCREVER "Digite a constante de eliminação k (/h):"
        LER k_eliminacao
        ESCREVER "Digite o tempo para análise (h):"
        LER tempo_analise
        ESCREVER "Digite a dose administrada (mg):"
        LER dose

        SE c0 > 0 E k_eliminacao > 0 E tempo_analise > 0 E dose > 0 ENTÃO
            concentracao_tempo, meia_vida, auc, clearance ← modelarFarmacocinetica(c0

            ESCREVER "Concentração inicial C0:", c0, "mg/L"
            ESCREVER "Constante de eliminação k:", k_eliminacao, "/h"
            ESCREVER "Concentração em t =", tempo_analise, "h:", concentracao_tempo,
            ESCREVER "Meia-vida (t½):", meia_vida, "h"
            ESCREVER "AUC (0→∞):", auc, "mg h/L"
            ESCREVER "Clearance (CL):", clearance, "L/h"

            // Análise da eliminação
            percentual_eliminado ← (1 - concentracao_tempo / c0) * 100
            ESCREVER "Percentual eliminado em", tempo_analise, "h:", percentual_elimi

            // Tempo para concentração específica
            ESCREVER "Para atingir 10% da concentração inicial:"
            tempo_10_porcento ← -log(0.1) / k_eliminacao
            ESCREVER "Tempo necessário:", tempo_10_porcento, "h"

        SENÃO
            ESCREVER "Todos os valores devem ser positivos"
        FIM SE

    SENÃO SE opcao = 3 ENTÃO
        ESCREVER "=== DETERMINAÇÃO DE BIOEQUIVALÊNCIA ==="
        ESCREVER "Dados do medicamento de referência:"
        ESCREVER "Digite a AUC de referência (mg h/L):"
        LER auc_referencia
        ESCREVER "Digite a Cmax de referência (mg/L):"
        LER cmax_referencia

        ESCREVER "Dados do medicamento teste:"
        ESCREVER "Digite a AUC do teste (mg h/L):"
        LER auc_teste
```

```
            ESCREVER "Digite a Cmax do teste (mg/L):"
            LER cmax_teste

            SE auc_referencia > 0 E cmax_referencia > 0 E auc_teste > 0 E cmax_teste > 0 E
                bioequivalencia ← determinarBioequivalencia(auc_referencia, cmax_referenc

                razao_auc ← auc_teste / auc_referencia
                razao_cmax ← cmax_teste / cmax_referencia

                ESCREVER "Medicamento de referência:"
                ESCREVER "  AUC:", auc_referencia, "mg h/L"
                ESCREVER "  Cmax:", cmax_referencia, "mg/L"

                ESCREVER "Medicamento teste:"
                ESCREVER "  AUC:", auc_teste, "mg h/L"
                ESCREVER "  Cmax:", cmax_teste, "mg/L"

                ESCREVER "Razão AUC (teste/referência):", razao_auc
                ESCREVER "Razão Cmax (teste/referência):", razao_cmax

                SE bioequivalencia = 1 ENTÃO
                    ESCREVER "Resultado: BIOEQUIVALENTES"
                    ESCREVER "Ambas as razões estão entre 0.80 e 1.25"
                SENÃO
                    ESCREVER "Resultado: NÃO BIOEQUIVALENTES"
                    SE razao_auc < 0.8 OU razao_auc > 1.25 ENTÃO
                        ESCREVER "AUC fora do intervalo 0.80-1.25"
                    FIM SE
                    SE razao_cmax < 0.8 OU razao_cmax > 1.25 ENTÃO
                        ESCREVER "Cmax fora do intervalo 0.80-1.25"
                    FIM SE
                FIM SE

            SENÃO
                ESCREVER "Todos os valores devem ser positivos"
            FIM SE

        SENÃO SE opcao = 4 ENTÃO
            ESCREVER "=== ANÁLISE COMPLETA DE MEDICAMENTO ==="
            ESCREVER "Quantas amostras de concentração foram coletadas (3-10):"
            LER num_amostras

            SE num_amostras >= 3 E num_amostras <= 10 ENTÃO
                ESCREVER "Digite a dose administrada (mg):"
                LER dose_total

                SE dose_total > 0 ENTÃO
                    // Calcular estatísticas das concentrações
                    media_concentracao, dp_concentracao ← calcularEstatisticas(num_amostr

                    ESCREVER "Estimativa dos parâmetros farmacocinéticos:"
```

```
ESCREVER "Digite a concentração inicial estimada C0 (mg/L):"
LER c0_estimado
ESCREVER "Digite a constante de eliminação estimada k (/h):"
LER k_estimado

SE c0_estimado > 0 E k_estimado > 0 ENTÃO
    // Modelagem completa
    concentracao_24h, meia_vida_calc, auc_calc, clearance_calc ← mode

    ESCREVER "\n--- ESTATÍSTICAS DESCRITIVAS ---"
    ESCREVER "Média das concentrações:", media_concentracao, "mg/L"
    ESCREVER "Desvio padrão:", dp_concentracao, "mg/L"
    cv_total ← (dp_concentracao / media_concentracao) * 100
    ESCREVER "Coeficiente de variação:", cv_total, "%"

    ESCREVER "\n--- PARÂMETROS FARMACOCINÉTICOS ---"
    ESCREVER "Concentração inicial (C0):", c0_estimado, "mg/L"
    ESCREVER "Meia-vida (t½):", meia_vida_calc, "h"
    ESCREVER "AUC (0→∞):", auc_calc, "mg h/L"
    ESCREVER "Clearance total:", clearance_calc, "L/h"
    ESCREVER "Volume de distribuição:", dose_total / c0_estimado, "L"

    ESCREVER "\n--- PREDIÇÕES ---"
    ESCREVER "Concentração em 24h:", concentracao_24h, "mg/L"

    // Tempo para diferentes frações
    tempo_50_porcento ← meia_vida_calc
    tempo_25_porcento ← 2 * meia_vida_calc
    tempo_12_5_porcento ← 3 * meia_vida_calc

    ESCREVER "Tempo para 50% eliminação:", tempo_50_porcento, "h"
    ESCREVER "Tempo para 75% eliminação:", tempo_25_porcento, "h"
    ESCREVER "Tempo para 87.5% eliminação:", tempo_12_5_porcento, "h"

    ESCREVER "\n--- CLASSIFICAÇÃO FARMACOCINÉTICA ---"

    SE meia_vida_calc < 6 ENTÃO
        ESCREVER "Eliminação: Rápida (t½ < 6h)"
    SENÃO SE meia_vida_calc < 24 ENTÃO
        ESCREVER "Eliminação: Moderada (6h  t½ < 24h)"
    SENÃO
        ESCREVER "Eliminação: Lenta (t½  24h)"
    FIM SE

    SE clearance_calc < 1 ENTÃO
        ESCREVER "Clearance: Baixo (< 1 L/h)"
    SENÃO SE clearance_calc < 10 ENTÃO
        ESCREVER "Clearance: Moderado (1-10 L/h)"
    SENÃO
        ESCREVER "Clearance: Alto (> 10 L/h)"
    FIM SE
```

```
                SENÃO
                    ESCREVER "Parâmetros estimados devem ser positivos"
                FIM SE
            SENÃO
                ESCREVER "Dose deve ser positiva"
            FIM SE
        SENÃO
            ESCREVER "Número de amostras deve estar entre 3 e 10"
        FIM SE

    SENÃO SE opcao = 5 ENTÃO
        ESCREVER "=== CÁLCULOS AUXILIARES ==="
        ESCREVER "1. Converter unidades de concentração"
        ESCREVER "2. Calcular dose por peso corporal"
        ESCREVER "3. Funções matemáticas (exp, log, etc.)"
        LER tipo_calculo

        SE tipo_calculo = 1 ENTÃO
            ESCREVER "Digite a concentração em mg/L:"
            LER conc_mg_l
            ESCREVER "Digite o peso molecular (g/mol):"
            LER peso_molecular

            SE conc_mg_l >= 0 E peso_molecular > 0 ENTÃO
                // Converter mg/L para  M
                conc_micromolar ← (conc_mg_l * 1000) / peso_molecular
                ESCREVER conc_mg_l, "mg/L =", conc_micromolar, " M"

                // Converter para outras unidades
                conc_ng_ml ← conc_mg_l * 1000
                conc_g_dl ← conc_mg_l / 10

                ESCREVER "Equivalências:"
                ESCREVER "  ", conc_ng_ml, "ng/mL"
                ESCREVER "  ", conc_g_dl, " g/dL"
            SENÃO
                ESCREVER "Valores devem ser não negativos"
            FIM SE

        SENÃO SE tipo_calculo = 2 ENTÃO
            ESCREVER "Digite a dose total (mg):"
            LER dose_mg
            ESCREVER "Digite o peso corporal (kg):"
            LER peso_kg

            SE dose_mg > 0 E peso_kg > 0 ENTÃO
                dose_por_kg ← dose_mg / peso_kg
                ESCREVER "Dose por peso corporal:", dose_por_kg, "mg/kg"

                // Classificação pediátrica vs adulto
```

```
                SE peso_kg < 20 ENTÃO
                    ESCREVER "Classificação: Dose pediátrica"
                SENÃO SE peso_kg < 80 ENTÃO
                    ESCREVER "Classificação: Dose adulto padrão"
                SENÃO
                    ESCREVER "Classificação: Dose para peso elevado"
                FIM SE
            SENÃO
                ESCREVER "Valores devem ser positivos"
            FIM SE

        SENÃO SE tipo_calculo = 3 ENTÃO
            ESCREVER "Digite um valor para testar funções matemáticas:"
            LER valor

            SE valor > 0 ENTÃO
                resultado_exp ← exp(valor)
                resultado_log ← log(valor)
                resultado_sqrt ← sqrt(valor)
                resultado_floor ← floor(valor)
                resultado_ceil ← ceil(valor)

                ESCREVER "Para x =", valor, ":"
                ESCREVER "exp(x) =", resultado_exp
                ESCREVER "ln(x) =", resultado_log
                ESCREVER "sqrt(x) =", resultado_sqrt
                ESCREVER "floor(x) =", resultado_floor
                ESCREVER "ceil(x) =", resultado_ceil

                SE valor <= 10 ENTÃO
                    potencia_2 ← pow(valor, 2)
                    potencia_3 ← pow(valor, 3)
                    ESCREVER "x² =", potencia_2
                    ESCREVER "x³ =", potencia_3
                FIM SE
            SENÃO
                ESCREVER "Valor deve ser positivo para logaritmo"
            FIM SE
        SENÃO
            ESCREVER "Opção inválida"
        FIM SE

    SENÃO SE opcao = 6 ENTÃO
        ESCREVER "Encerrando sistema de laboratório farmacêutico..."

    SENÃO
        ESCREVER "Opção inválida! Tente novamente."
    FIM SE

    // Pausa para visualização
    SE opcao  6 ENTÃO
```

```
            ESCREVER "\nPressione Enter para continuar..."
            LER pausa
        FIM SE
    FIM ENQUANTO
FIM

// FUNÇÃO: CALCULAR ESTATÍSTICAS
// Processamento sequencial sem arrays
FUNÇÃO calcularEstatisticas(n: inteiro): real, real
INÍCIO
    soma ← 0
    soma_quadrados ← 0

    // Primeira passagem: calcular média
    PARA i DE 1 ATÉ n FAÇA
        ESCREVER "Digite o valor", i, ":"
        LER valor
        soma ← soma + valor
    FIM PARA

    media ← soma / n

    // Segunda passagem: calcular desvio padrão
    ESCREVER "Agora digite os valores novamente para calcular o desvio padrão:"
    PARA i DE 1 ATÉ n FAÇA
        ESCREVER "Digite novamente o valor", i, ":"
        LER valor
        diferenca ← valor - media
        quadrado_diferenca ← pow(diferenca, 2)
        soma_quadrados ← soma_quadrados + quadrado_diferenca
    FIM PARA

    variancia ← soma_quadrados / (n - 1)
    desvio_padrao ← sqrt(variancia)

    RETORNAR media, desvio_padrao
FIM

// FUNÇÃO: MODELAR FARMACOCINÉTICA
// C(t) = C0 × e^(-kt)
FUNÇÃO modelarFarmacocinetica(c0: real, k: real, tempo: real, dose: real): real, real, re
INÍCIO
    // Concentração no tempo t: C(t) = C0 × e^(-kt)
    concentracao_t ← c0 * exp(-k * tempo)

    // Meia-vida: t½ = ln(2) / k
    meia_vida ← log(2) / k

    // AUC (0→∞): AUC = C0 / k
    auc ← c0 / k
```

```
    // Clearance: CL = Dose / AUC
    clearance ← dose / auc

    RETORNAR concentracao_t, meia_vida, auc, clearance
FIM

// FUNÇÃO: DETERMINAR BIOEQUIVALÊNCIA
// Critérios: 0.80   razão   1.25 para AUC e Cmax
FUNÇÃO determinarBioequivalencia(auc_ref: real, cmax_ref: real, auc_teste: real, cmax_tes
INÍCIO
    // Calcular razões
    razao_auc ← auc_teste / auc_ref
    razao_cmax ← cmax_teste / cmax_ref

    // Verificar critérios de bioequivalência
    auc_ok ← (razao_auc >= 0.80) E (razao_auc <= 1.25)
    cmax_ok ← (razao_cmax >= 0.80) E (razao_cmax <= 1.25)

    // Retornar 1 se bioequivalente, 0 se não
    SE auc_ok E cmax_ok ENTÃO
        RETORNAR 1
    SENÃO
        RETORNAR 0
    FIM SE
FIM
```

O pseudocódigo proposto pode ser representado pelo código disponível em https://onlinegdb.com/ayWgb_nfH.

Nota: O professor pode adaptar os exercícios para incluir mais funções ou módulos, dependendo do nível de conhecimento dos alunos e do tempo disponível. O Exercício **G2** é uma boa oportunidade para o professor introduzir estruturas de dados complexas, pares e vetores por exemplo.

7.2 Decomposição e Abstração: Estruturas de Dados Compostas

Ao completar o exercício **G2**, os alunos devem estar prontos para desafios mais complexos que envolvem estruturas de dados compostas, como arrays, vetores, strings, tuplas e registros. Que completam o conhecimento básico de manipulação de dados. As abstrações representadas por estas estruturas tem impacto além dos cursos de Ciência e Engenharia da Computação.

Nesta seção, serão usadas as funções, métodos e módulos que são relacinados a leitura e gravação de arquivos. Novamente, o professor deve escolhar ente usar a Técnica da Sequência de Fibonacci, ou a sua própria metodologia. Considerando que a Técnica da Sequência de Fibonacci é uma possibilidade, a seguir estão explicitados 7 exercícios (1, 1, 2, 3).

Finalmente, neste ponto da disciplina, o aluno já será capaz de ler código e entender o algoritmo que ele representa. Deste ponto em diante, os exercícios serão mais desafiadores. Porém, os eventuas fluxograms, ou pseudocódigos, serão deixados para o professor e seus alunos, de acordo com a capacidade e evolução de cada turma.

Os exercícios são mais desafiadores e, consequentemente mais demorados, o professor pode sugerir que eles sejam realizados em grupos e, neste caso, cabe ao professor dividir os exerícios em módulos que possam ser implementados separadamente. A divisão em módulos é uma boa prática de programação, que ajuda a organizar o código e facilita a manutenção. Além

disso, serve como prática de **D**epuração e engenharia de software. Nos exercícios a seguir serão considerados grupos de 4 alunos.

Estratégia de Apresentação: O professor pode dividir a turma em grupos de 4 alunos, apresentar o problema e solicitar que seja criado um fluxograma para a solução. O problema **H2** deve ser usado para apresentar uma estratégia de **D**ecomposição. Todos os outros problemas devem ser apresentados, sem incluir qualquer estratégia de decomposição. É importante que os alunos mudem de personagem a cada exercício, para que todos tenham a oportunidade de praticar a leitura e escrita de código. Ou seja, a pessoa que ocupa a posição Aluno 1 no exercício **H2** deve ser a Aluno 2 no exercício **I2**, e assim por diante.

7.2.1 H2

Uma pequena biblioteca precisa gerenciar os empréstimos de livros, registrando o nome do usuário e a quantidade de dias do empréstimo. O programa deve usar um array para armazenar até 10 empréstimos, salvar os dados em disco e permitir consultas. Desenvolva um sistema que registre empréstimos, mantenha um histórico em um array e permita consultar todos os empréstimos ou os de um usuário específico.

Funções a implementar:

- **Estruturas de dados**: Array para armazenar nomes e dias de empréstimo;
- **Funções próprias**: `registrarEmprestimo()`, `salvarEmprestimos()`, `lerEmprestimos()`, `buscarEmprestimo()`;
- **Funções padrão**: `getline()`, `stoi()`, `to_string()`;

Entrada: Nome do usuário (string) e dias de empréstimo (inteiro, 1 a 30) **Saída**: Lista de todos os empréstimos ou empréstimos de um usuário específico, com média dos dias de empréstimo

7.2.1.1 Algoritmização: Pseudocódigo e Código

O pseudocódigo para esta tarefa pode ser:

```
ALGORITMO GerenciadorEmprestimos

    // Constantes
    MAX_EMPRESTIMOS ← 10
    ARQUIVO ← "empréstimos.csv"

    // Declaração de arrays
    nomes[MAX_EMPRESTIMOS]: string
    dias[MAX_EMPRESTIMOS]: inteiro
    quantidade: inteiro

INÍCIO
    // Inicializar sistema
    ESCREVER "=== Sistema de Empréstimos de Livros ==="
    ESCREVER "Inicializando sistema..."
    inicializarArquivo()

    // Carregar dados existentes
    lerEmprestimos(nomes, dias, quantidade)
```

```
    FAÇA
        exibirMenu()
        ESCREVER "Escolha uma opção: "
        LER opcao

        SE opcao = 1 ENTÃO
            registrarEmprestimo(nomes, dias, quantidade)
            salvarEmprestimos(nomes, dias, quantidade)
            ESCREVER quantidade, " empréstimos registrados."
        SENÃO SE opcao = 2 ENTÃO
            ESCREVER "Digite o nome do usuário para busca: "
            LER nome
            buscarEmprestimo(nome)
        SENÃO SE opcao = 3 ENTÃO
            nomesTemp[MAX_EMPRESTIMOS]: string
            diasTemp[MAX_EMPRESTIMOS]: inteiro
            totalRegistros: inteiro
            SE lerEmprestimos(nomesTemp, diasTemp, totalRegistros) = FALSO ENTÃO
                ESCREVER "Erro ao ler dados ou arquivo não encontrado!"
            SENÃO
                SE totalRegistros = 0 ENTÃO
                    ESCREVER "Nenhum registro encontrado!"
                SENÃO
                    ESCREVER "=== RELATÓRIO COMPLETO ==="
                    soma ← 0.0
                    PARA i DE 0 ATÉ totalRegistros - 1 FAÇA
                        ESCREVER "Usuário: ", nomesTemp[i], ", Dias: ", diasTemp[i]
                        soma ← soma + diasTemp[i]
                    FIM PARA
                    ESCREVER "Total de empréstimos: ", totalRegistros
                    ESCREVER "Média dos dias: ", soma / totalRegistros
                FIM SE
            FIM SE
        SENÃO SE opcao = 0 ENTÃO
            ESCREVER "Encerrando sistema..."
        SENÃO
            ESCREVER "Opção inválida! Tente novamente."
        FIM SE

        SE opcao   0 ENTÃO
            ESCREVER "Pressione Enter para continuar..."
            LER pausa
        FIM SE
    ENQUANTO opcao   0

FIM

FUNÇÃO registrarEmprestimo(nomes: string[], dias: inteiro[], quantidade: inteiro): void
INÍCIO
    SE quantidade   MAX_EMPRESTIMOS ENTÃO
        ESCREVER "Limite de empréstimos atingido!"
```

```
        RETORNAR
    FIM SE

    FAÇA
        ESCREVER "Digite o nome do usuário (ou 'sair' para encerrar): "
        LER nome
        SE nome = "sair" ENTÃO
            RETORNAR
        FIM SE

        SE nome = "" OU nome CONTER "," ENTÃO
            ESCREVER "Nome inválido! Deve ser não vazio e sem vírgulas."
            CONTINUAR
        FIM SE

        ESCREVER "Digite a quantidade de dias (1-30): "
        LER entrada
        TENTAR
            dia ← CONVERTER_PARA_INTEIRO(entrada)
            SE dia   1 E dia   30 ENTÃO
                nomes[quantidade] ← nome
                dias[quantidade] ← dia
                quantidade ← quantidade + 1
                SE quantidade = MAX_EMPRESTIMOS ENTÃO
                    ESCREVER "Limite de empréstimos atingido!"
                    RETORNAR
                FIM SE
            SENÃO
                ESCREVER "Dias fora do intervalo (1-30). Ignorado."
            FIM SE
        SENÃO
            ESCREVER "Valor inválido para dias. Ignorado."
        FIM TENTAR
    ENQUANTO VERDADEIRO
FIM

FUNÇÃO salvarEmprestimos(nomes: string[], dias: inteiro[], quantidade: inteiro): void
INÍCIO
    ABRIR arquivo(ARQUIVO, SOBRESCREVER)
    SE arquivo NÃO ABERTO ENTÃO
        ESCREVER "Erro ao abrir arquivo para gravação!"
        RETORNAR
    FIM SE

    ESCREVER_ARQUIVO(arquivo, "Nome,Dias")
    PARA i DE 0 ATÉ quantidade - 1 FAÇA
        ESCREVER_ARQUIVO(arquivo, nomes[i], ",", dias[i])
    FIM PARA
    FECHAR arquivo
FIM
```

```
FUNÇÃO lerEmprestimos(nomes: string[], dias: inteiro[], quantidade: inteiro, nomeFiltro:
INÍCIO
    ABRIR arquivo(ARQUIVO, LEITURA)
    SE arquivo NÃO ABERTO ENTÃO
        RETORNAR FALSO
    FIM SE

    quantidade ← 0
    LER_LINHA(arquivo, linha)
    SE linha CONTER "Nome" ENTÃO
        // Cabeçalho detectado
    SENÃO
        REPOSICIONAR_ARQUIVO(arquivo, INÍCIO)
    FIM SE

    ENQUANTO LER_LINHA(arquivo, linha) E quantidade < MAX_EMPRESTIMOS FAÇA
        SEPARAR(linha, ",", nome, diasStr)
        SE nomeFiltro = "" OU nome = nomeFiltro ENTÃO
            TENTAR
                dias[quantidade] ← CONVERTER_PARA_INTEIRO(diasStr)
                nomes[quantidade] ← nome
                quantidade ← quantidade + 1
            SENÃO
                // Ignorar linhas inválidas
            FIM TENTAR
        FIM SE
    FIM ENQUANTO

    SE quantidade = MAX_EMPRESTIMOS E MAIS_LINHAS(arquivo) ENTÃO
        ESCREVER "Aviso: Mais registros no arquivo do que o limite de ", MAX_EMPRESTIMOS
    FIM SE

    FECHAR arquivo
    RETORNAR VERDADEIRO
FIM

FUNÇÃO buscarEmprestimo(nome: string): void
INÍCIO
    nomes[MAX_EMPRESTIMOS]: string
    dias[MAX_EMPRESTIMOS]: inteiro
    totalRegistros: inteiro

    SE lerEmprestimos(nomes, dias, totalRegistros, nome) = FALSO ENTÃO
        ESCREVER "Erro ao ler dados!"
        RETORNAR
    FIM SE

    SE totalRegistros = 0 ENTÃO
        ESCREVER "Nenhum empréstimo encontrado para ", nome, "!"
        RETORNAR
    FIM SE
```

```
    ESCREVER "=== EMPRÉSTIMOS DE ", nome, " ==="
    soma ← 0.0
    PARA i DE 0 ATÉ totalRegistros - 1 FAÇA
        ESCREVER "Usuário: ", nomes[i], ", Dias: ", dias[i]
        soma ← soma + dias[i]
    FIM PARA
    ESCREVER "Total de empréstimos: ", totalRegistros
    ESCREVER "Média dos dias: ", soma / totalRegistros
FIM

FUNÇÃO inicializarArquivo(): void
INÍCIO
    SE ARQUIVO NÃO EXISTIR ENTÃO
        ABRIR arquivo(ARQUIVO, ESCRITA)
        SE arquivo ABERTO ENTÃO
            ESCREVER_ARQUIVO(arquivo, "Nome,Dias")
            FECHAR arquivo
            ESCREVER "Arquivo de dados criado: ", ARQUIVO
        SENÃO
            ESCREVER "Erro ao criar arquivo de dados!"
        FIM SE
    SENÃO
        ESCREVER "Arquivo de dados encontrado: ", ARQUIVO
    FIM SE
FIM

FUNÇÃO exibirMenu(): void
INÍCIO
    ESCREVER "=== MENU PRINCIPAL ==="
    ESCREVER "1. Registrar empréstimos"
    ESCREVER "2. Buscar empréstimos por usuário"
    ESCREVER "3. Relatório completo"
    ESCREVER "0. Sair"
FIM
```

7.2.1.2 Decomposição: Divisão de Tarefas para o Problema H2

Para resolver o problema **H2** de gerenciamento de empréstimos de livros, o trabalho pode ser dividido entre quatro alunos que trabalharão independentemente na mesma sala, sem sistema de controle de versão, durante o mesmo período de tempo. Cada aluno será responsável por uma parte específica do sistema, com interfaces claras para facilitar a integração final. Abaixo está a divisão das tarefas, considerando as funções solicitadas (`registrarEmprestimo`, `salvarEmprestimos`, `lerEmprestimos`, `buscarEmprestimo`).

Aluno 1: Função `registrarEmprestimo` e Validação de Entrada

Responsabilidades: implementar `registrarEmprestimo(std::string nomes[], int dias[], int& quantidade)` para coletar nome do usuário e dias de empréstimo (1 a 30). Validar:

- Nome não vazio e sem vírgulas (para evitar corromper o CSV);
- Dias como inteiro entre 1 e 30, usando stoi com tratamento de exceções;
- Não zerar quantidade para preservar empréstimos anteriores no array.

Permitir múltiplos empréstimos até `MAX_EMPRESTIMOS` (10) ou até o usuário digitar "sair"; Corrigir uso de `std::cin.ignore` para evitar problemas com o `buffer`. Ou o equivalente em Python.

Tarefas específicas:

- Escrever a função com entrada via `getline` para nome e dias;
- Validar nome (Ex.: `if (nome.empty() || nome.find(',') != std::string::npos)`);
- Adicionar novos empréstimos a partir do índice quantidade, sem zerar o array;
- Testar isoladamente com entradas válidas e inválidas (Ex.: dias fora do intervalo, nomes com vírgulas).

Interface:

Recebe `nomes[]`, `dias[]` e `quantidade` como parâmetros; Fornece dados válidos para `salvarEmprestimos` e consultas.

Aluno 2: Função `salvarEmprestimos` e Inicialização do Arquivo

Responsabilidades:

- Implementar `salvarEmprestimos(const std::string nomes[], const int dias[], int quantidade)` para gravar empréstimos no arquivo `emprestimos.csv`;
- Sobrescrever o arquivo (usando `std::ios::trunc`) para evitar duplicatas;
- Implementar `inicializarArquivo()` para criar o arquivo CSV com cabeçalho `Nome,Dias` se não existir;
- Verificar erros ao abrir/escrever no arquivo.

Tarefas específicas:

- Gravar cada par nome,dias em uma linha do CSV, sobrescrevendo o arquivo;
- Preservar o cabeçalho ao sobrescrever;
- Verificar abertura do arquivo e exibir mensagens de erro;
- Testar isoladamente com diferentes tamanhos de arrays e verificar o arquivo gerado.

Interface:

Recebe `nomes[], dias[]` e quantidade de `registrarEmprestimo`; Produz um arquivo CSV lido por `lerEmprestimos`.

Aluno 3: Função `lerEmprestimos` e Carregamento Inicial

Responsabilidades:

- Implementar `lerEmprestimos(std::string nomes[], int dias[], int& quantidade, const std::string& nomeFiltro = "")` para ler o arquivo CSV; Suportar leitura de todos os registros ou apenas os de um usuário (com nomeFiltro); Alertar se o arquivo tiver mais registros que `MAX_EMPRESTIMOS`; Carregar dados do arquivo no início do programa (no `main`) para sincronizar o array local.

Tarefas específicas:

- Ler o CSV, ignorar o cabeçalho e preencher nomes e dias;
- Implementar filtragem por nomeFiltro quando não vazio;
- Contar registros e alertar se exceder `MAX_EMPRESTIMOS`;
- Modificar o `main` para chamar `lerEmprestimos` no início;
- Testar com arquivos CSV vazios, com menos de 10 registros e com mais de 10.

Interface:

- Lê o arquivo CSV de `salvarEmprestimos`;

- Fornece dados para buscarEmprestimo e relatório completo.

Aluno 4: Função buscarEmprestimo, Relatório e Interface do Usuário
Responsabilidades:

- Implementar `buscarEmprestimo(const std::string& nome)` para exibir empréstimos de um usuário e calcular a média dos dias;
- Implementar o relatório completo (opção 3 do menu) para listar todos os empréstimos e a média dos dias;
- Escrever `exibirMenu()` e gerenciar a interface no `main`, corrigindo `std::cin.ignore`;
- Garantir saídas formatadas (média com uma casa decimal, mensagens claras).

Tarefas específicas:

- Chamar `lerEmprestimos` com filtro de nome e exibir resultados com total e média;
- Implementar relatório completo no `main`, usando `lerEmprestimos` sem filtro;
- Corrigir `std::cin.ignore` no `main` (e.g., usar `std::cin.ignore(std::numeric_limits<std::strea` `'\n')` após `std::cin >> opcao`);
- Testar a interface com busca de usuário existente/inexistente e relatórios.

Interface:
Usa dados de `lerEmprestimos` para exibir resultados; Interage com o usuário, fornecendo nomes para `registrarEmprestimo` e `buscarEmprestimo`.

7.2.1.3 Considerações para Integração do Problema H2

Interfaces: os alunos devem concordar com assinaturas das funções e formato do CSV (nome,dias com cabeçalho); **Depuração**: os alunos devem fazer a **D**epuração de cada parte isoladamente, simulando entradas/saídas;
Passos importantes da integração:

- Copiar main e exibirMenu do Aluno 4;
- Inserir `inicializarArquivo` e `salvarEmprestimos` do Aluno 2;
- Adicionar `lerEmprestimos` do Aluno 3, ajustando o `main` para carregar dados iniciais;
- Incluir `registrarEmprestimo` do Aluno 1 e `buscarEmprestimo` do Aluno 4.

Resolução de conflitos: discutir problemas imediatamente na sala. **Depuração**: testar o programa com cenários como registrar empréstimos, buscar por usuário e exibir relatórios.

7.2.1.4 Código C++23

Um dos códigos em C++23 que implementa o pseudocódigo acima pode ser encontrado em https://onlinegdb.com/KzjXw8xZm.

7.2.2 I2

Um estudante quer organizar suas tarefas diárias em uma lista dinâmica, com descrições curtas. O programa deve usar um vector para armazenar as tarefas, salvar em disco e permitir consultas. Desenvolva um sistema que gerencie tarefas, adicione ou remova tarefas de um vector e permita buscar tarefas por palavra-chave.
Funções a implementar:

- **Funções próprias**: `adicionarTarefa()`, `removerTarefa()`, `salvarTarefas()`, `lerTarefas()`, `buscarTarefa()`
- **Funções padrão**: `getline()`, `to_string()`

Entrada: Descrição da tarefa (string) ou palavra-chave para busca **Saída**: Lista completa de tarefas ou tarefas encontradas, com contagem total de tarefas

7.2.2.1 Algoritmização: Pseudocódigo e Código

```
ALGORITMO GerenciadorTarefas

    // Constantes
    ARQUIVO ← "tarefas.txt"

    // Declaração do vetor dinâmico
    tarefas: vetor dinâmico de string

INÍCIO
    // Inicializar sistema
    ESCREVER "=== Sistema de Gerenciamento de Tarefas ==="
    ESCREVER "Inicializando sistema..."
    inicializarArquivo()

    // Carregar dados existentes
    lerTarefas(tarefas)

    FAÇA
        exibirMenu()
        ESCREVER "Escolha uma opção: "
        LER opcao

        SE opcao = 1 ENTÃO
            adicionarTarefa(tarefas)
            salvarTarefas(tarefas)
            ESCREVER "Tarefa(s) registrada(s)."
        SENÃO SE opcao = 2 ENTÃO
            ESCREVER "Digite o índice da tarefa a remover (1-based): "
            LER indice
            removerTarefa(tarefas, indice)
            salvarTarefas(tarefas)
            ESCREVER "Tarefa removida."
        SENÃO SE opcao = 3 ENTÃO
            ESCREVER "Digite a palavra-chave para busca (ou vazio para todas): "
            LER palavraChave
            buscarTarefa(tarefas, palavraChave)
        SENÃO SE opcao = 4 ENTÃO
            tarefasTemp: vetor dinâmico de string
            SE lerTarefas(tarefasTemp) = FALSO ENTÃO
                ESCREVER "Erro ao ler dados ou arquivo não encontrado!"
            SENÃO
                SE tamanho(tarefasTemp) = 0 ENTÃO
                    ESCREVER "Nenhuma tarefa encontrada!"
                SENÃO
                    ESCREVER "=== RELATÓRIO COMPLETO ==="
                    PARA i DE 0 ATÉ tamanho(tarefasTemp) - 1 FAÇA
                        ESCREVER "Tarefa ", i + 1, ": ", tarefasTemp[i]
                    FIM PARA
                    ESCREVER "Total de tarefas: ", tamanho(tarefasTemp)
```

```
                FIM SE
            FIM SE
        SENÃO SE opcao = 0 ENTÃO
            ESCREVER "Encerrando sistema..."
        SENÃO
            ESCREVER "Opção inválida! Tente novamente."
        FIM SE

        SE opcao   0 ENTÃO
            ESCREVER "Pressione Enter para continuar..."
            LER pausa
        FIM SE
    ENQUANTO opcao   0

FIM

FUNÇÃO adicionarTarefa(tarefas: vetor dinâmico de string): void
INÍCIO
    FAÇA
        ESCREVER "Digite a descrição da tarefa (ou 'sair' para encerrar): "
        LER descricao
        SE descricao = "sair" ENTÃO
            RETORNAR
        FIM SE

        SE descricao = "" OU descricao CONTER "," ENTÃO
            ESCREVER "Descrição inválida! Deve ser não vazia e sem vírgulas."
            CONTINUAR
        FIM SE

        tarefas.ADICIONAR(descricao)
    ENQUANTO VERDADEIRO
FIM

FUNÇÃO salvarTarefas(tarefas: vetor dinâmico de string): void
INÍCIO
    ABRIR arquivo(ARQUIVO, SOBRESCREVER)
    SE arquivo NÃO ABERTO ENTÃO
        ESCREVER "Erro ao abrir arquivo para gravação!"
        RETORNAR
    FIM SE

    ESCREVER_ARQUIVO(arquivo, "Tarefas")
    PARA i DE 0 ATÉ tamanho(tarefas) - 1 FAÇA
        ESCREVER_ARQUIVO(arquivo, tarefas[i])
    FIM PARA
    FECHAR arquivo
FIM

FUNÇÃO lerTarefas(tarefas: vetor dinâmico de string): booleano
INÍCIO
```

```
    ABRIR arquivo(ARQUIVO, LEITURA)
    SE arquivo NÃO ABERTO ENTÃO
        RETORNAR FALSO
    FIM SE

    tarefas.LIMPAR()
    LER_LINHA(arquivo, linha)
    SE linha CONTER "Tarefas" ENTÃO
        // Cabeçalho detectado
    SENÃO
        REPOSICIONAR_ARQUIVO(arquivo, INÍCIO)
    FIM SE

    ENQUANTO LER_LINHA(arquivo, linha) FAÇA
        SE linha   "" ENTÃO
            tarefas.ADICIONAR(linha)
        FIM SE
    FIM ENQUANTO

    FECHAR arquivo
    RETORNAR VERDADEIRO
FIM

FUNÇÃO removerTarefa(tarefas: vetor dinâmico de string, indice: inteiro): void
INÍCIO
    SE indice < 1 OU indice > tamanho(tarefas) ENTÃO
        ESCREVER "Índice inválido!"
        RETORNAR
    FIM SE

    tarefas.REMOVER(indice - 1)
FIM

FUNÇÃO buscarTarefa(tarefas: vetor dinâmico de string, palavraChave: string): void
INÍCIO
    totalEncontradas ← 0
    SE palavraChave = "" ENTÃO
        ESCREVER "=== TODAS AS TAREFAS ==="
        PARA i DE 0 ATÉ tamanho(tarefas) - 1 FAÇA
            ESCREVER "Tarefa ", i + 1, ": ", tarefas[i]
            totalEncontradas ← totalEncontradas + 1
        FIM PARA
    SENÃO
        ESCREVER "=== TAREFAS COM PALAVRA-CHAVE '", palavraChave, "' ==="
        PARA i DE 0 ATÉ tamanho(tarefas) - 1 FAÇA
            SE tarefas[i] CONTER palavraChave ENTÃO
                ESCREVER "Tarefa ", i + 1, ": ", tarefas[i]
                totalEncontradas ← totalEncontradas + 1
            FIM SE
        FIM PARA
    FIM SE
```

```
    SE totalEncontradas = 0 ENTÃO
        ESCREVER "Nenhuma tarefa encontrada!"
    SENÃO
        ESCREVER "Total de tarefas encontradas: ", totalEncontradas
    FIM SE
FIM

FUNÇÃO inicializarArquivo(): void
INÍCIO
    SE ARQUIVO NÃO EXISTIR ENTÃO
        ABRIR arquivo(ARQUIVO, ESCRITA)
        SE arquivo ABERTO ENTÃO
            ESCREVER_ARQUIVO(arquivo, "Tarefas")
            FECHAR arquivo
            ESCREVER "Arquivo de dados criado: ", ARQUIVO
        SENÃO
            ESCREVER "Erro ao criar arquivo de dados!"
        FIM SE
    SENÃO
        ESCREVER "Arquivo de dados encontrado: ", ARQUIVO
    FIM SE
FIM

FUNÇÃO exibirMenu(): void
INÍCIO
    ESCREVER "=== MENU PRINCIPAL ==="
    ESCREVER "1. Adicionar tarefa"
    ESCREVER "2. Remover tarefa"
    ESCREVER "3. Buscar tarefas por palavra-chave"
    ESCREVER "4. Relatório completo"
    ESCREVER "0. Sair"
FIM
```

7.2.2.2 Decomposição: Divisão de Tarefas para o Problema I2

Aluno 1: Função `adicionarTarefa` e Validação de Entrada
Responsabilidades:

- Implementar `adicionarTarefa(vector<string>& tarefas)` para coletar descrições de tarefas (strings).
- Validar:
  - Descrição não vazia e sem vírgulas (para evitar corromper o arquivo)]
  - Permitir múltiplas adições até o usuário digitar "sair";
- Usar `getline` para entrada da descrição;
- Adicionar tarefas ao vetor dinâmico sem limite fixo.

Tarefas específicas:

- Escrever a função para ler descrição via `getline` e adicionar ao vetor;
- Validar descrição (ex.: `if (descricao.empty() || descricao.find(',') != string::npos)` em C++);

- Testar isoladamente com entradas válidas (ex.: “Estudar C++”) e inválidas (ex.: ““,”Comprar leite, pão”);
- Tratar buffer de entrada adequadamente (ex.: evitar problemas com `std::cin`).

Interface:

- Recebe `vector<string>& tarefas` como parâmetro;
- Modifica o vetor diretamente, fornecendo dados para `salvarTarefas` e consultas.

Aluno 2: Função `salvarTarefas` e Inicialização do Arquivo
Responsabilidades:

- Implementar `salvarTarefas(const vector<string>& tarefas)` para gravar tarefas em `tarefas.txt` (uma tarefa por linha);
- Sobrescrever o arquivo (ex.: `std::ios::trunc` em C++) para evitar duplicatas;
- Implementar `inicializarArquivo()` para criar o arquivo com cabeçalho “Tarefas” se não existir;
- Verificar erros ao abrir/escrever no arquivo.

Tarefas específicas:

- Gravar cada tarefa em uma linha, sobrescrevendo o arquivo;
- Preservar o cabeçalho ao sobrescrever;
- Verificar abertura do arquivo e exibir mensagens de erro;
- Testar isoladamente com vetores de diferentes tamanhos e verificar o arquivo gerado.

Interface:

- Recebe `vector<string>& tarefas` de `adicionarTarefa` ou `removerTarefa`.
- Produz um arquivo lido por `lerTarefas`.

Aluno 3: Função `lerTarefas` e Carregamento Inicial
Responsabilidades:

- Implementar `lerTarefas(vector<string>& tarefas)` para ler o arquivo e preencher o vetor;
- Ignorar o cabeçalho do arquivo;
- Carregar dados do arquivo no início do programa (no `main`) para sincronizar o vetor local;
- Tratar erros como arquivo inexistente ou vazio.

Tarefas específicas:

- Ler o arquivo linha por linha, ignorar o cabeçalho e adicionar tarefas ao vetor;
- Limpar o vetor antes de carregar para evitar duplicatas;
- Modificar o `main` para chamar `lerTarefas` no início;
- Testar com arquivos vazios, com uma tarefa e com múltiplas tarefas.

Interface:

- Lê o arquivo gerado por `salvarTarefas`;
- Fornece dados para `buscarTarefa` e relatórios.

Aluno 4: Funções `removerTarefa`, `buscarTarefa`, Relatório e Interface do Usuário
Responsabilidades:

- Implementar `removerTarefa(vector<string>& tarefas, int indice)` para remover uma tarefa pelo índice (1-based para o usuário);
- Implementar `buscarTarefa(const vector<string>& tarefas, const string& palavraChave)` para exibir tarefas que contenham a palavra-chave (case-insensitive, se possível);
- Implementar o relatório completo (opção 3 do menu) para listar todas as tarefas e contar o total;
- Escrever `exibirMenu()` e gerenciar a interface no `main`, corrigindo problemas de buffer (ex.: `std::cin.ignore` em C++);
- Garantir saídas formatadas (ex.: mensagens claras, contagem de tarefas).

Tarefas específicas:

- Implementar remoção por índice com validação (ex.: índice válido);
- Implementar busca por palavra-chave, verificando substrings na descrição;
- Implementar relatório completo no `main`, usando `lerTarefas` para listar todas as tarefas;
- Corrigir `std::cin.ignore` no `main` (ex.: `std::cin.ignore(std::numeric_limits<std::streamsize '\n')` após leitura de opção);
- Testar a interface com busca de palavras existentes/inexistentes, remoção válida/inválida e relatórios.

Interface:

- Usa dados de `lerTarefas` para exibir resultados;
- Interage com o usuário, fornecendo descrições para `adicionarTarefa`, índices para `removerTarefa` e palavras-chave para `buscarTarefa`.

7.2.2.3 Considerações para Integração

Interfaces:

- Os alunos devem concordar com a assinatura das funções e o formato do arquivo (`tarefas.txt`, uma tarefa por linha, com cabeçalho);
- O vetor dinâmico (`vector<string>`) é passado por referência para modificações e como const para consultas.

Depuração:

- Cada aluno deve depurar sua parte isoladamente, simulando entradas/saídas (ex.: arquivos de teste, vetores pré-preenchidos).

Passos importantes da integração:

1. Copiar `main` e `exibirMenu` do Aluno 4;
2. Inserir `inicializarArquivo` e `salvarTarefas` do Aluno 2;
3. Adicionar `lerTarefas` do Aluno 3, ajustando o `main` para carregar dados iniciais;
4. Incluir `adicionarTarefa` do Aluno 1, `removerTarefa` e `buscarTarefa` do Aluno 4.

Resolução de conflitos:

- Discutir problemas imediatamente na sala (ex.: formato do arquivo, tratamento de buffer);

- Testar o programa completo com cenários como adicionar tarefas, remover, buscar por palavra-chave e exibir relatórios.

Depuração:

- Verificar adição de múltiplas tarefas, remoção por índice, busca por palavra-chave e relatório completo;
- Testar casos extremos (arquivo vazio, palavra-chave não encontrada, índice inválido).

7.2.2.4 Código C++23

A solução em C++23 para o problema I2 pode ser encontrada em https://onlinegdb.com/SHylYeSy7.‘

7.2.3 J2

Uma loja de conveniência precisa registrar os produtos vendidos, armazenando nome e preço como uma string formatada. O programa deve usar um array de strings para até 15 produtos, salvar em disco e permitir consultas. Desenvolva um sistema que registre vendas, mantenha um array de strings com os produtos e permita buscar produtos por nome.

Funções a implementar:

- **Funções próprias**: `registrarProduto()`, `salvarProdutos()`, `lerProdutos()`, `buscarProduto()`
- **Funções padrão**: `getline()`, `stod()`, `to_string()`

Entrada: Nome do produto e preço (combinados em uma string no formato “nome,preco”)
Saída: Lista de todos os produtos ou produtos encontrados por busca, com soma dos preços

7.2.3.1 Algoritmização: Pseudocódigo

```
ALGORITMO GerenciadorProdutos

    // Constantes
    MAX_PRODUTOS ← 15
    ARQUIVO ← "produtos.csv"

    // Declaração de arrays
    produtos[MAX_PRODUTOS]: string
    quantidade: inteiro

INÍCIO
    // Inicializar sistema
    ESCREVER "=== Sistema de Gerenciamento de Produtos ==="
    ESCREVER "Inicializando sistema..."
    inicializarArquivo()

    // Carregar dados existentes
    lerProdutos(produtos, quantidade)

    FAÇA
        exibirMenu()
```

```
        ESCREVER "Escolha uma opção: "
        LER opcao

        SE opcao = 1 ENTÃO
            registrarProduto(produtos, quantidade)
            salvarProdutos(produtos, quantidade)
            ESCREVER quantidade, " produto(s) registrado(s)."
        SENÃO SE opcao = 2 ENTÃO
            ESCREVER "Digite o nome do produto para busca: "
            LER nome
            buscarProduto(nome)
        SENÃO SE opcao = 3 ENTÃO
            produtosTemp[MAX_PRODUTOS]: string
            totalRegistros: inteiro
            SE lerProdutos(produtosTemp, totalRegistros) = FALSO ENTÃO
                ESCREVER "Erro ao ler dados ou arquivo não encontrado!"
            SENÃO
                SE totalRegistros = 0 ENTÃO
                    ESCREVER "Nenhum produto encontrado!"
                SENÃO
                    ESCREVER "=== RELATÓRIO COMPLETO ==="
                    soma ← 0.0
                    PARA i DE 0 ATÉ totalRegistros - 1 FAÇA
                        SEPARAR(produtosTemp[i], ",", nome, precoStr)
                        TENTAR
                            preco ← CONVERTER_PARA_DOUBLE(precoStr)
                            ESCREVER "Produto: ", nome, ", Preço: R$", preco
                            soma ← soma + preco
                        SENÃO
                            ESCREVER "Erro ao processar preço em: ", produtosTemp[i]
                        FIM TENTAR
                    FIM PARA
                    ESCREVER "Total de produtos: ", totalRegistros
                    ESCREVER "Soma dos preços: R$", soma
                FIM SE
            FIM SE
        SENÃO SE opcao = 0 ENTÃO
            ESCREVER "Encerrando sistema..."
        SENÃO
            ESCREVER "Opção inválida! Tente novamente."
        FIM SE

        SE opcao  0 ENTÃO
            ESCREVER "Pressione Enter para continuar..."
            LER pausa
        FIM SE
    ENQUANTO opcao  0

FIM

FUNÇÃO registrarProduto(produtos: string[], quantidade: inteiro): void
```

```
INÍCIO
    SE quantidade  MAX_PRODUTOS ENTÃO
        ESCREVER "Limite de produtos atingido!"
        RETORNAR
    FIM SE

    FAÇA
        ESCREVER "Digite o nome do produto (ou 'sair' para encerrar): "
        LER nome
        SE nome = "sair" ENTÃO
            RETORNAR
        FIM SE

        SE nome = "" OU nome CONTER "," ENTÃO
            ESCREVER "Nome inválido! Deve ser não vazio e sem vírgulas."
            CONTINUAR
        FIM SE

        ESCREVER "Digite o preço do produto (positivo): "
        LER entrada
        TENTAR
            preco ← CONVERTER_PARA_DOUBLE(entrada)
            SE preco > 0 ENTÃO
                produtos[quantidade] ← nome + "," + to_string(preco)
                quantidade ← quantidade + 1
                SE quantidade = MAX_PRODUTOS ENTÃO
                    ESCREVER "Limite de produtos atingido!"
                    RETORNAR
                FIM SE
            SENÃO
                ESCREVER "Preço inválido! Deve ser positivo."
            FIM SE
        SENÃO
            ESCREVER "Valor inválido para preço. Ignorado."
        FIM TENTAR
    ENQUANTO VERDADEIRO
FIM

FUNÇÃO salvarProdutos(produtos: string[], quantidade: inteiro): void
INÍCIO
    ABRIR arquivo(ARQUIVO, SOBRESCREVER)
    SE arquivo NÃO ABERTO ENTÃO
        ESCREVER "Erro ao abrir arquivo para gravação!"
        RETORNAR
    FIM SE

    ESCREVER_ARQUIVO(arquivo, "Nome,Preco")
    PARA i DE 0 ATÉ quantidade - 1 FAÇA
        ESCREVER_ARQUIVO(arquivo, produtos[i])
    FIM PARA
    FECHAR arquivo
```

```
FIM

FUNÇÃO lerProdutos(produtos: string[], quantidade: inteiro, nomeFiltro: string = ""): boo
INÍCIO
    ABRIR arquivo(ARQUIVO, LEITURA)
    SE arquivo NÃO ABERTO ENTÃO
        RETORNAR FALSO
    FIM SE

    quantidade ← 0
    LER_LINHA(arquivo, linha)
    SE linha CONTER "Nome" ENTÃO
        // Cabeçalho detectado
    SENÃO
        REPOSICIONAR_ARQUIVO(arquivo, INÍCIO)
    FIM SE

    ENQUANTO LER_LINHA(arquivo, linha) E quantidade < MAX_PRODUTOS FAÇA
        SEPARAR(linha, ",", nome, precoStr)
        SE nomeFiltro = "" OU nome CONTER nomeFiltro ENTÃO
            TENTAR
                preco ← CONVERTER_PARA_DOUBLE(precoStr)
                produtos[quantidade] ← linha
                quantidade ← quantidade + 1
            SENÃO
                // Ignorar linhas inválidas
            FIM TENTAR
        FIM SE
    FIM ENQUANTO

    SE quantidade = MAX_PRODUTOS E MAIS_LINHAS(arquivo) ENTÃO
        ESCREVER "Aviso: Mais registros no arquivo do que o limite de ", MAX_PRODUTOS
    FIM SE

    FECHAR arquivo
    RETORNAR VERDADEIRO
FIM

FUNÇÃO buscarProduto(nome: string): void
INÍCIO
    produtosTemp[MAX_PRODUTOS]: string
    totalRegistros: inteiro

    SE lerProdutos(produtosTemp, totalRegistros, nome) = FALSO ENTÃO
        ESCREVER "Erro ao ler dados!"
        RETORNAR
    FIM SE

    SE totalRegistros = 0 ENTÃO
        ESCREVER "Nenhum produto encontrado para '", nome, "'!"
        RETORNAR
```

```
    FIM SE

    ESCREVER "=== PRODUTOS COM NOME CONTENDO '", nome, "' ==="
    soma ← 0.0
    PARA i DE 0 ATÉ totalRegistros - 1 FAÇA
        SEPARAR(produtosTemp[i], ",", nomeProduto, precoStr)
        TENTAR
            preco ← CONVERTER_PARA_DOUBLE(precoStr)
            ESCREVER "Produto: ", nomeProduto, ", Preço: R$", preco
            soma ← soma + preco
        SENÃO
            ESCREVER "Erro ao processar preço em: ", produtosTemp[i]
        FIM TENTAR
    FIM PARA
    ESCREVER "Total de produtos: ", totalRegistros
    ESCREVER "Soma dos preços: R$", soma
FIM

FUNÇÃO inicializarArquivo(): void
INÍCIO
    SE ARQUIVO NÃO EXISTIR ENTÃO
        ABRIR arquivo(ARQUIVO, ESCRITA)
        SE arquivo ABERTO ENTÃO
            ESCREVER_ARQUIVO(arquivo, "Nome,Preco")
            FECHAR arquivo
            ESCREVER "Arquivo de dados criado: ", ARQUIVO
        SENÃO
            ESCREVER "Erro ao criar arquivo de dados!"
        FIM SE
    SENÃO
        ESCREVER "Arquivo de dados encontrado: ", ARQUIVO
    FIM SE
FIM

FUNÇÃO exibirMenu(): void
INÍCIO
    ESCREVER "=== MENU PRINCIPAL ==="
    ESCREVER "1. Registrar produto"
    ESCREVER "2. Buscar produtos por nome"
    ESCREVER "3. Relatório completo"
    ESCREVER "0. Sair"
FIM
```

7.2.3.2 Decomposição: Divisão de Tarefas para o Problema J2

O problema **J2** propõe um sistema para gerenciar vendas de produtos em uma loja de conveniência, usando um array fixo de até 15 strings formatadas como "nome,preco", com persistência em arquivo e funcionalidades de registro, salvamento, leitura e busca por nome. Abaixo está a divisão de tarefas entre quatro alunos, trabalhando independentemente na mesma sala, sem controle de versão, com interfaces claras para facilitar a integração. O pseudocódigo completo é apresentado em seguida.

Aluno 1: Função `registrarProduto` e Validação de Entrada
Responsabilidades:

- Implementar `registrarProduto(string produtos[], int& quantidade)` para coletar nome do produto (string) e preço (double, positivo), formatando como "nome,preco".
- Validar:
  - Nome não vazio e sem vírgulas (para evitar corromper o formato);
  - Preço como double positivo, usando `stod` com tratamento de exceções;
  - Não zerar `quantidade` para preservar produtos anteriores no array;
- Permitir múltiplos registros até `MAX_PRODUTOS` (15) ou até o usuário digitar "sair";
- Tratar buffer de entrada adequadamente (ex.: corrigir uso de `std::cin.ignore` em C++).

Tarefas específicas:

- Escrever a função com entrada via `getline` para nome e preço;
- Validar nome (ex.: `if (nome.empty() || nome.find(',') != string::npos)` em C++);
- Validar preço (ex.: converter com `stod`, verificar se positivo);
- Adicionar string formatada ("nome,preco") ao array a partir do índice `quantidade`;
- Testar isoladamente com entradas válidas (ex.: "Leite,3.50") e inválidas (ex.: "","Pão,leite", "-5").

Interface:

- Recebe `produtos[]` e `quantidade` como parâmetros;
- Fornece strings formatadas para `salvarProdutos` e consultas.

Aluno 2: Função `salvarProdutos` e Inicialização do Arquivo
Responsabilidades:

- Implementar `salvarProdutos(const string produtos[], int quantidade)` para gravar produtos em `produtos.csv` (formato: "Nome,Preco" por linha);
- Sobrescrever o arquivo (ex.: `std::ios::trunc` em C++) para evitar duplicatas;
- Implementar `inicializarArquivo()` para criar o arquivo CSV com cabeçalho "Nome,Preco" se não existir;
- Verificar erros ao abrir/escrever no arquivo.

Tarefas específicas:

- Gravar cada string formatada ("nome,preco") em uma linha, sobrescrevendo o arquivo;
- Preservar o cabeçalho ao sobrescrever;
- Verificar abertura do arquivo e exibir mensagens de erro;
- Testar isoladamente com diferentes tamanhos de arrays e verificar o arquivo gerado.

Interface:

- Recebe `produtos[]` e `quantidade` de `registrarProduto`;
- Produz um arquivo CSV lido por `lerProdutos`.

Aluno 3: Função `lerProdutos` e Carregamento Inicial
Responsabilidades:

- Implementar `lerProdutos(string produtos[], int& quantidade, const string& nomeFiltro = "")` para ler o arquivo CSV e preencher o array;
- Suportar leitura de todos os produtos ou filtragem por nome (substring no campo nome);
- Alertar se o arquivo tiver mais registros que `MAX_PRODUTOS` (15);
- Carregar dados do arquivo no início do programa (no `main`) para sincronizar o array local.

Tarefas específicas:

- Ler o CSV, ignorar o cabeçalho e preencher o array com strings formatadas;
- Implementar filtragem por `nomeFiltro` quando não vazio (verificar substring no campo nome);
- Contar registros e alertar se exceder `MAX_PRODUTOS`;
- Modificar o `main` para chamar `lerProdutos` no início;
- Testar com arquivos CSV vazios, com menos de 15 registros e com mais de 15.

Interface:

- Lê o arquivo CSV de `salvarProdutos`;
- Fornece dados para `buscarProduto` e relatório completo.

Aluno 4: Função `buscarProduto`, Relatório e Interface do Usuário
Responsabilidades:

- Implementar `buscarProduto(const string& nome)` para exibir produtos cujo nome contenha a substring fornecida, com soma dos preços;
- Implementar o relatório completo (opção 3 do menu) para listar todos os produtos e a soma dos preços;
- Escrever `exibirMenu()` e gerenciar a interface no `main`, corrigindo problemas de buffer (ex.: `std::cin.ignore` em C++);
- Garantir saídas formatadas (ex.: preços com duas casas decimais, mensagens claras);

Tarefas específicas:

- Chamar `lerProdutos` com filtro de nome e exibir resultados com total e soma dos preços;
- Implementar relatório completo no `main`, usando `lerProdutos` sem filtro;
- Corrigir `std::cin.ignore` no `main` (ex.: `std::cin.ignore(std::numeric_limits<std::streamsize '\n')` após leitura de opção);
- Testar a interface com busca de nomes existentes/inexistentes e relatórios.

Interface:

- Usa dados de `lerProdutos` para exibir resultados;
- Interage com o usuário, fornecendo nomes para `registrarProduto` e `buscarProduto`.

7.2.3.3 Considerações para Integração

Interfaces:

- Os alunos devem concordar com a assinatura das funções e o formato do CSV (uma linha por produto, "nome,preco" com cabeçalho);

- O array `produtos[]` armazena strings no formato "nome,preco", e `quantidade` rastreia o número de registros.

Depuração:

- Cada aluno deve depurar sua parte isoladamente, simulando entradas/saídas (ex.: arquivos de teste, arrays pré-preenchidos).

Passos importantes da integração:

1. Copiar `main` e `exibirMenu` do Aluno 4;
2. Inserir `inicializarArquivo` e `salvarProdutos` do Aluno 2;
3. Adicionar `lerProdutos` do Aluno 3, ajustando o `main` para carregar dados iniciais;
4. Incluir `registrarProduto` do Aluno 1 e `buscarProduto` do Aluno 4.

Resolução de conflitos:

- Discutir problemas imediatamente na sala (ex.: formato do CSV, tratamento de buffer);
- Testar o programa completo com cenários como registrar produtos, buscar por nome e exibir relatórios.

Depuração:

- Verificar registro de múltiplos produtos, busca por nome (substring), e relatório completo com soma de preços;
- Testar casos extremos (arquivo vazio, nome não encontrado, mais de 15 produtos).

7.2.3.4 Código C++23

O código correspondente ao problema J2 pode ser encontrado em https://onlinegdb.com/_Ng8nLlV1.

7.2.4 K2

Um clube de eventos precisa gerenciar registros de participantes em atividades culturais. Cada participante é registrado com nome, quantidade de eventos que participou e total pago pelas inscrições. O programa deve usar structs para armazenar os dados, salvar em disco e permitir consultas. Desenvolva um sistema que registre participantes, mantenha os dados em structs, salve em um arquivo CSV e permita consultar todos os registros ou os de um participante específico.

Funções a implementar:

- **Funções próprias**: registrarParticipante(), salvarParticipantes(), lerParticipantes(), buscarParticipante();
- **Funções padrão**: getline(), stoi(), stod(), to_string().
- **Entrada**: nome do participante (string), quantidade de eventos (inteiro, 1 a 50), total pago (decimal).
- **Saída**: lista de todos os participantes ou registros de um participante específico, com média de eventos e soma dos valores pagos.

7.2.4.1 Algoritmização: Pseudocódigo

```
ALGORITMO GerenciadorParticipantes

    // Estrutura de dados
    ESTRUTURA Participante
        nome: string
        eventos: inteiro
        totalPago: decimal
    FIM ESTRUTURA

    // Constantes
    MAX_PARTICIPANTES ← 20
    ARQUIVO ← "participantes.csv"

    // Declaração de arrays
    participantes[MAX_PARTICIPANTES]: Participante
    quantidade: inteiro

INÍCIO
    // Inicializar sistema
    ESCREVER "=== Sistema de Gerenciamento de Participantes ==="
    ESCREVER "Inicializando sistema..."
    inicializarArquivo()

    // Carregar dados existentes
    lerParticipantes(participantes, quantidade)

    FAÇA
        exibirMenu()
        ESCREVER "Escolha uma opção: "
        LER opcao

        SE opcao = 1 ENTÃO
            registrarParticipante(participantes, quantidade)
            salvarParticipantes(participantes, quantidade)
            ESCREVER quantidade, " participante(s) registrado(s)."
        SENÃO SE opcao = 2 ENTÃO
            ESCREVER "Digite o nome do participante para busca: "
            LER nome
            buscarParticipante(nome)
        SENÃO SE opcao = 3 ENTÃO
            participantesTemp[MAX_PARTICIPANTES]: Participante
            totalRegistros: inteiro
            SE lerParticipantes(participantesTemp, totalRegistros) = FALSO ENTÃO
                ESCREVER "Erro ao ler dados ou arquivo não encontrado!"
            SENÃO
                SE totalRegistros = 0 ENTÃO
                    ESCREVER "Nenhum participante encontrado!"
                SENÃO
                    ESCREVER "=== RELATÓRIO COMPLETO ==="
                    somaEventos ← 0.0
                    somaPagos ← 0.0
```

```
                    PARA i DE 0 ATÉ totalRegistros - 1 FAÇA
                        ESCREVER "Participante: ", participantesTemp[i].nome, ", Eventos:
                        somaEventos ← somaEventos + participantesTemp[i].eventos
                        somaPagos ← somaPagos + participantesTemp[i].totalPago
                    FIM PARA
                    ESCREVER "Total de participantes: ", totalRegistros
                    ESCREVER "Média de eventos: ", somaEventos / totalRegistros
                    ESCREVER "Soma dos valores pagos: R$", somaPagos
                FIM SE
            FIM SE
        SENÃO SE opcao = 0 ENTÃO
            ESCREVER "Encerrando sistema..."
        SENÃO
            ESCREVER "Opção inválida! Tente novamente."
        FIM SE

        SE opcao   0 ENTÃO
            ESCREVER "Pressione Enter para continuar..."
            LER pausa
        FIM SE
    ENQUANTO opcao   0

FIM

FUNÇÃO registrarParticipante(participantes: Participante[], quantidade: inteiro): void
INÍCIO
    SE quantidade   MAX_PARTICIPANTES ENTÃO
        ESCREVER "Limite de participantes atingido!"
        RETORNAR
    FIM SE

    FAÇA
        ESCREVER "Digite o nome do participante (ou 'sair' para encerrar): "
        LER nome
        SE nome = "sair" ENTÃO
            RETORNAR
        FIM SE

        SE nome = "" OU nome CONTER "," ENTÃO
            ESCREVER "Nome inválido! Deve ser não vazio e sem vírgulas."
            CONTINUAR
        FIM SE

        ESCREVER "Digite a quantidade de eventos (1-50): "
        LER entradaEventos
        TENTAR
            eventos ← CONVERTER_PARA_INTEIRO(entradaEventos)
            SE eventos   1 E eventos   50 ENTÃO
                ESCREVER "Digite o total pago (positivo): "
                LER entradaTotal
                TENTAR
```

```
                    total ← CONVERTER_PARA_DOUBLE(entradaTotal)
                    SE total > 0 ENTÃO
                        participantes[quantidade].nome ← nome
                        participantes[quantidade].eventos ← eventos
                        participantes[quantidade].totalPago ← total
                        quantidade ← quantidade + 1
                        SE quantidade = MAX_PARTICIPANTES ENTÃO
                            ESCREVER "Limite de participantes atingido!"
                            RETORNAR
                        FIM SE
                    SENÃO
                        ESCREVER "Total pago inválido! Deve ser positivo."
                    FIM SE
                SENÃO
                    ESCREVER "Valor inválido para total pago. Ignorado."
                FIM TENTAR
            SENÃO
                ESCREVER "Quantidade de eventos fora do intervalo (1-50). Ignorado."
            FIM SE
        SENÃO
            ESCREVER "Valor inválido para eventos. Ignorado."
        FIM TENTAR
    ENQUANTO VERDADEIRO
FIM

FUNÇÃO salvarParticipantes(participantes: Participante[], quantidade: inteiro): void
INÍCIO
    ABRIR arquivo(ARQUIVO, SOBRESCREVER)
    SE arquivo NÃO ABERTO ENTÃO
        ESCREVER "Erro ao abrir arquivo para gravação!"
        RETORNAR
    FIM SE

    ESCREVER_ARQUIVO(arquivo, "Nome,Eventos,TotalPago")
    PARA i DE 0 ATÉ quantidade - 1 FAÇA
        ESCREVER_ARQUIVO(arquivo, participantes[i].nome, ",", participantes[i].eventos, "
    FIM PARA
    FECHAR arquivo
FIM

FUNÇÃO lerParticipantes(participantes: Participante[], quantidade: inteiro, nomeFiltro: s
INÍCIO
    ABRIR arquivo(ARQUIVO, LEITURA)
    SE arquivo NÃO ABERTO ENTÃO
        RETORNAR FALSO
    FIM SE

    quantidade ← 0
    LER_LINHA(arquivo, linha)
    SE linha CONTER "Nome" ENTÃO
        // Cabeçalho detectado
```

```
    SENÃO
        REPOSICIONAR_ARQUIVO(arquivo, INÍCIO)
    FIM SE

    ENQUANTO LER_LINHA(arquivo, linha) E quantidade < MAX_PARTICIPANTES FAÇA
        SEPARAR(linha, ",", nome, eventosStr, totalStr)
        SE nomeFiltro = "" OU nome = nomeFiltro ENTÃO
            TENTAR
                eventos ← CONVERTER_PARA_INTEIRO(eventosStr)
                total ← CONVERTER_PARA_DOUBLE(totalStr)
                participantes[quantidade].nome ← nome
                participantes[quantidade].eventos ← eventos
                participantes[quantidade].totalPago ← total
                quantidade ← quantidade + 1
            SENÃO
                // Ignorar linhas inválidas
            FIM TENTAR
        FIM SE
    FIM ENQUANTO

    SE quantidade = MAX_PARTICIPANTES E MAIS_LINHAS(arquivo) ENTÃO
        ESCREVER "Aviso: Mais registros no arquivo do que o limite de ", MAX_PARTICIPANTE
    FIM SE

    FECHAR arquivo
    RETORNAR VERDADEIRO
FIM

FUNÇÃO buscarParticipante(nome: string): void
INÍCIO
    participantesTemp[MAX_PARTICIPANTES]: Participante
    totalRegistros: inteiro

    SE lerParticipantes(participantesTemp, totalRegistros, nome) = FALSO ENTÃO
        ESCREVER "Erro ao ler dados!"
        RETORNAR
    FIM SE

    SE totalRegistros = 0 ENTÃO
        ESCREVER "Nenhum participante encontrado para '", nome, "'!"
        RETORNAR
    FIM SE

    ESCREVER "=== REGISTROS DE ", nome, " ==="
    somaEventos ← 0.0
    somaPagos ← 0.0
    PARA i DE 0 ATÉ totalRegistros - 1 FAÇA
        ESCREVER "Participante: ", participantesTemp[i].nome, ", Eventos: ", participante
        somaEventos ← somaEventos + participantesTemp[i].eventos
        somaPagos ← somaPagos + participantesTemp[i].totalPago
    FIM PARA
```

```
    ESCREVER "Total de registros: ", totalRegistros
    ESCREVER "Média de eventos: ", somaEventos / totalRegistros
    ESCREVER "Soma dos valores pagos: R$", somaPagos
FIM

FUNÇÃO inicializarArquivo(): void
INÍCIO
    SE ARQUIVO NÃO EXISTIR ENTÃO
        ABRIR arquivo(ARQUIVO, ESCRITA)
        SE arquivo ABERTO ENTÃO
            ESCREVER_ARQUIVO(arquivo, "Nome,Eventos,TotalPago")
            FECHAR arquivo
            ESCREVER "Arquivo de dados criado: ", ARQUIVO
        SENÃO
            ESCREVER "Erro ao criar arquivo de dados!"
        FIM SE
    SENÃO
        ESCREVER "Arquivo de dados encontrado: ", ARQUIVO
    FIM SE
FIM

FUNÇÃO exibirMenu(): void
INÍCIO
    ESCREVER "=== MENU PRINCIPAL ==="
    ESCREVER "1. Registrar participante"
    ESCREVER "2. Buscar participante por nome"
    ESCREVER "3. Relatório completo"
    ESCREVER "0. Sair"
FIM
```

7.2.4.2 Decomposição: Divisão de Tarefas para o Problema K2

O problema **K2** propõe um sistema para gerenciar registros de participantes em atividades culturais, usando structs para armazenar nome, quantidade de eventos e total pago, com persistência em arquivo CSV e funcionalidades de registro, salvamento, leitura e busca por nome. Abaixo está a divisão de tarefas entre quatro alunos, trabalhando independentemente na mesma sala, sem controle de versão, com interfaces claras para facilitar a integração. O pseudocódigo completo é apresentado em seguida.

Aluno 1: Função `registrarParticipante` e Validação de Entrada
Responsabilidades:

- Implementar `registrarParticipante(Participante participantes[], int& quantidade)` para coletar nome (string), quantidade de eventos (inteiro, 1 a 50) e total pago (double, positivo);
- Validar:
  - Nome não vazio e sem vírgulas (para evitar corromper o CSV);
  - Quantidade de eventos entre 1 e 50, usando `stoi` com tratamento de exceções;
  - Total pago como double positivo, usando `stod` com tratamento de exceções;
  - Não zerar `quantidade` para preservar registros anteriores;

- Permitir múltiplos registros até `MAX_PARTICIPANTES` (definido como 20, por exemplo) ou até o usuário digitar "sair";
- Tratar buffer de entrada adequadamente (ex.: corrigir uso de `std::cin.ignore` em C++).

Tarefas específicas:

- Escrever a função com entrada via `getline` para nome, eventos e total pago;
- Validar nome (ex.: `if (nome.empty() || nome.find(',') != string::npos)` em C++);
- Validar eventos (ex.: converter com `stoi`, verificar intervalo 1-50);
- Validar total pago (ex.: converter com `stod`, verificar se positivo);
- Adicionar dados à struct `Participante` no array a partir do índice `quantidade`;
- Testar isoladamente com entradas válidas (ex.: "Ana,5,100.50") e inválidas (ex.: "","João,pão", "0", "-10").

Interface:

- Recebe `participantes[]` e `quantidade` como parâmetros;
- Fornece dados válidos para `salvarParticipantes` e consultas.

Aluno 2: Função `salvarParticipantes` e Inicialização do Arquivo
Responsabilidades:

- Implementar `salvarParticipantes(const Participante participantes[], int quantidade)` para gravar registros em `participantes.csv` (formato: "Nome,Eventos,TotalPago");
- Sobrescrever o arquivo (ex.: `std::ios::trunc` em C++) para evitar duplicatas;
- Implementar `inicializarArquivo()` para criar o arquivo CSV com cabeçalho "Nome,Eventos,TotalPago" se não existir;
- Verificar erros ao abrir/escrever no arquivo.

Tarefas específicas:

- Gravar cada registro (nome, eventos, total pago) em uma linha do CSV, sobrescrevendo o arquivo;
- Preservar o cabeçalho ao sobrescrever;
- Verificar abertura do arquivo e exibir mensagens de erro;
- Testar isoladamente com diferentes tamanhos de arrays e verificar o arquivo gerado.

Interface:

- Recebe `participantes[]` e `quantidade` de `registrarParticipante`;
- Produz um arquivo CSV lido por `lerParticipantes`.

Aluno 3: Função `lerParticipantes` e Carregamento Inicial
Responsabilidades:

- Implementar `lerParticipantes(Participante participantes[], int& quantidade, const string& nomeFiltro = "")` para ler o arquivo CSV e preencher o array de structs;
- Suportar leitura de todos os registros ou filtragem por nome (exato ou substring);
- Alertar se o arquivo tiver mais registros que `MAX_PARTICIPANTES`;

- Carregar dados do arquivo no início do programa (no `main`) para sincronizar o array local.

Tarefas específicas:

- Ler o CSV, ignorar o cabeçalho e preencher o array de structs;
- Implementar filtragem por `nomeFiltro` quando não vazio (ex.: verificar igualdade ou substring no campo nome);
- Contar registros e alertar se exceder `MAX_PARTICIPANTES`;
- Modificar o `main` para chamar `lerParticipantes` no início;
- Testar com arquivos CSV vazios, com menos de 20 registros e com mais de 20.

Interface:

- Lê o arquivo CSV de `salvarParticipantes`;
- Fornece dados para `buscarParticipante` e relatório completo.

Aluno 4: Função `buscarParticipante`, Relatório e Interface do Usuário
Responsabilidades:

- Implementar `buscarParticipante(const string& nome)` para exibir registros de um participante, com média de eventos e soma dos valores pagos;
- Implementar o relatório completo (opção 3 do menu) para listar todos os participantes, com média de eventos e soma total dos valores pagos;
- Escrever `exibirMenu()` e gerenciar a interface no `main`, corrigindo problemas de buffer (ex.: `std::cin.ignore` em C++);
- Garantir saídas formatadas (ex.: preços com duas casas decimais, média com uma casa decimal, mensagens claras).

Tarefas específicas:

- Chamar `lerParticipantes` com filtro de nome e exibir resultados com total, média de eventos e soma dos valores pagos;
- Implementar relatório completo no `main`, usando `lerParticipantes` sem filtro;
- Corrigir `std::cin.ignore` no `main` (ex.: `std::cin.ignore(std::numeric_limits<std::streamsize '\n')` após leitura de opção);
- Testar a interface com busca de nomes existentes/inexistentes e relatórios.

Interface:

- Usa dados de `lerParticipantes` para exibir resultados;
- Interage com o usuário, fornecendo nomes para `registrarParticipante` e `buscarParticipante`.

7.2.4.3 Considerações para Integração

Interfaces:

- Os alunos devem concordar com a estrutura da `struct Participante` (campos: nome, eventos, totalPago) e o formato do CSV ("Nome,Eventos,TotalPago" com cabeçalho);
- O array `participantes[]` armazena `structs`, e `quantidade` rastreia o número de registros.

Depuração:

- Cada aluno deve depurar sua parte isoladamente, simulando entradas/saídas (ex.: arquivos de teste, arrays pré-preenchidos).

Passos importantes da integração:

1. Copiar `main` e `exibirMenu` do Aluno 4;
2. Inserir `inicializarArquivo` e `salvarParticipantes` do Aluno 2;
3. Adicionar `lerParticipantes` do Aluno 3, ajustando o `main` para carregar dados iniciais;
4. Incluir `registrarParticipante` do Aluno 1 e `buscarParticipante` do Aluno 4.

Resolução de conflitos:

- Discutir problemas imediatamente na sala (ex.: formato do CSV, tratamento de buffer);
- Testar o programa completo com cenários como registrar participantes, buscar por nome e exibir relatórios.

Depuração:

- Verificar registro de múltiplos participantes, busca por nome, e relatório completo com média de eventos e soma de valores;
- Testar casos extremos (arquivo vazio, nome não encontrado, mais de 20 participantes).

7.2.4.4 Código C++23

O código correspondente ao problema K2 pode ser encontrado em https://onlinegdb.com/HPGcXEHrR.

7.2.5 L2

A prefeitura instalou sensores de temperatura pela cidade e precisa processar os dados coletados durante um mês. O sistema deve armazenar todas as temperaturas, identificar padrões climáticos, calcular estatísticas detalhadas e gerar relatórios para o departamento de meio ambiente. Desenvolva um programa que armazene até 30 temperaturas diárias em um array, calcule a temperatura média do dia e do mês, identifique as temperaturas máxima e mínima, no dia e no mês, calcule desvio padrão e identifique dias com temperaturas extremas. Para isso o programa deve manter em disco um arquivo, csv, com data, hora e valor de cada temperatura registrada.

Funções a implementar:

- **Funções próprias**: `lerTemperaturas()`, `gravarTemperaturas()`, `lerDadosTemperaturas()`, `calcularEstatisticas()`, `encontrarExtremos()`, `calcularDesvio()`
- **Funções padrão**: `abs()`, `round()`, `sqrt()`

Entrada: um valor de temperatura por hora, ou até 30 valores de uma vez na linha de comando, separados por vírgulas. Os valores de temperatura estão em graus Celsius, com uma casa decimal.

Saída: Relatório completo com médias, extremos, desvio padrão e análise de tendências. O usuário pode escolher um dia específico para ver as temperaturas registradas, ou o relatório completo de um mês específico.

7.2.5.1 Algoritmização: Pseudocódigo

```
ALGORITMO GerenciadorTemperaturas

    // Constantes
    MAX_TEMPERATURAS ← 30
    ARQUIVO ← "temperaturas.csv"

    // Declaração de arrays
    temperaturas[MAX_TEMPERATURAS]: real

INÍCIO
    // Inicializar sistema
    ESCREVER "=== Sistema de Gerenciamento de Temperaturas ==="
    ESCREVER "Inicializando sistema..."
    inicializarArquivo()

    FAÇA
        exibirMenu()
        ESCREVER "Escolha uma opção: "
        LER opcao

        SE opcao = 1 ENTÃO
            ESCREVER "Digite a data (DD/MM/AAAA): "
            LER data
            quantidade ← 0
            SE VALIDAR_DATA(data) ENTÃO
                lerTemperaturas(temperaturas, quantidade, data)
                gravarTemperaturas(temperaturas, quantidade, data)
                ESCREVER quantidade, " temperatura(s) registrada(s)."
            SENÃO
                ESCREVER "Data inválida! Use o formato DD/MM/AAAA."
            FIM SE
        SENÃO SE opcao = 2 ENTÃO
            ESCREVER "Digite a data para busca (DD/MM/AAAA, ou vazio para todas): "
            LER dataFiltro
            SE dataFiltro = "" OU VALIDAR_DATA(dataFiltro) ENTÃO
                quantidade ← 0
                SE lerDadosTemperaturas(temperaturas, quantidade, dataFiltro) = FALSO ENT
                    ESCREVER "Erro ao ler dados ou arquivo não encontrado!"
                SENÃO SE quantidade = 0 ENTÃO
                    ESCREVER "Nenhuma temperatura encontrada para a data!"
                SENÃO
                    media ← 0.0
                    minima ← 0.0
                    maxima ← 0.0
                    calcularEstatisticas(temperaturas, quantidade, media)
                    encontrarExtremos(temperaturas, quantidade, minima, maxima)
                    desvio ← calcularDesvio(temperaturas, quantidade, media)
                    ESCREVER "=== RELATÓRIO PARA ", dataFiltro, " ==="
                    PARA i DE 0 ATÉ quantidade - 1 FAÇA
                        ESCREVER "Hora: ", FORMATAR_HORA(i), ", Temperatura: ", temperatu
                    FIM PARA
```

```
                ESCREVER "Total de temperaturas: ", quantidade
                ESCREVER "Média: ", ARREDONDAR(media, 1), "°C"
                ESCREVER "Mínima: ", ARREDONDAR(minima, 1), "°C"
                ESCREVER "Máxima: ", ARREDONDAR(maxima, 1), "°C"
                ESCREVER "Desvio padrão: ", ARREDONDAR(desvio, 2), "°C"
            FIM SE
        SENÃO
            ESCREVER "Data inválida! Use o formato DD/MM/AAAA."
        FIM SE
    SENÃO SE opcao = 3 ENTÃO
        // Relatório mensal (supõe que o usuário escolhe um mês/ano)
        ESCREVER "Digite o mês e ano (MM/AAAA): "
        LER mesAno
        SE VALIDAR_MES_ANO(mesAno) ENTÃO
            temperaturasTemp[MAX_TEMPERATURAS]: real
            totalRegistros ← 0
            diasExtremos: vetor dinâmico de string
            mediaMensal ← 0.0
            minimaMensal ← 0.0
            maximaMensal ← 0.0
            SE lerDadosTemperaturas(temperaturasTemp, totalRegistros, "") = FALSO ENT
                ESCREVER "Erro ao ler dados ou arquivo não encontrado!"
            SENÃO SE totalRegistros = 0 ENTÃO
                ESCREVER "Nenhuma temperatura encontrada!"
            SENÃO
                ESCREVER "=== RELATÓRIO MENSAL PARA ", mesAno, " ==="
                soma ← 0.0
                contagemTotal ← 0
                PARA CADA data NO ARQUIVO QUE CORRESPONDE A mesAno FAÇA
                    quantidade ← 0
                    lerDadosTemperaturas(temperaturasTemp, quantidade, data)
                    SE quantidade > 0 ENTÃO
                        mediaDia ← 0.0
                        minimaDia ← 0.0
                        maximaDia ← 0.0
                        calcularEstatisticas(temperaturasTemp, quantidade, mediaDia)
                        encontrarExtremos(temperaturasTemp, quantidade, minimaDia, ma
                        desvioDia ← calcularDesvio(temperaturasTemp, quantidade, medi
                        soma ← soma + mediaDia * quantidade
                        contagemTotal ← contagemTotal + quantidade
                        SE minimaDia < minimaMensal OU minimaMensal = 0 ENTÃO
                            minimaMensal ← minimaDia
                        FIM SE
                        SE maximaDia > maximaMensal ENTÃO
                            maximaMensal ← maximaDia
                        FIM SE
                        SE ABS(mediaDia - mediaMensal) > 2 * desvioDia ENTÃO
                            diasExtremos.ADICIONAR(data)
                        FIM SE
                    FIM SE
                FIM PARA
```

```
                    mediaMensal ← soma / contagemTotal
                    desvioMensal ← calcularDesvio(temperaturasTemp, totalRegistros, media
                    ESCREVER "Média mensal: ", ARREDONDAR(mediaMensal, 1), "°C"
                    ESCREVER "Mínima mensal: ", ARREDONDAR(minimaMensal, 1), "°C"
                    ESCREVER "Máxima mensal: ", ARREDONDAR(maximaMensal, 1), "°C"
                    ESCREVER "Desvio padrão mensal: ", ARREDONDAR(desvioMensal, 2), "°C"
                    ESCREVER "Dias com temperaturas extremas: "
                    SE tamanho(diasExtremos) = 0 ENTÃO
                        ESCREVER "Nenhum dia extremo identificado."
                    SENÃO
                        PARA CADA dia EM diasExtremos FAÇA
                            ESCREVER dia
                        FIM PARA
                    FIM SE
                FIM SE
            SENÃO
                ESCREVER "Mês/ano inválido! Use o formato MM/AAAA."
            FIM SE
        SENÃO SE opcao = 0 ENTÃO
            ESCREVER "Encerrando sistema..."
        SENÃO
            ESCREVER "Opção inválida! Tente novamente."
        FIM SE

        SE opcao   0 ENTÃO
            ESCREVER "Pressione Enter para continuar..."
            LER pausa
        FIM SE
    ENQUANTO opcao   0

FIM

FUNÇÃO lerTemperaturas(temperaturas: real[], quantidade: inteiro, data: string): void
INÍCIO
    SE quantidade   MAX_TEMPERATURAS ENTÃO
        ESCREVER "Limite de temperaturas atingido!"
        RETORNAR
    FIM SE

    ESCREVER "Digite as temperaturas (separadas por vírgulas, ou 'sair'): "
    LER entrada
    SE entrada = "sair" ENTÃO
        RETORNAR
    FIM SE

    SEPARAR(entrada, ",", valores)
    PARA CADA valor EM valores FAÇA
        SE quantidade   MAX_TEMPERATURAS ENTÃO
            ESCREVER "Limite de temperaturas atingido!"
            RETORNAR
        FIM SE
```

```
        TENTAR
            temp ← CONVERTER_PARA_REAL(valor)
            temp ← ARREDONDAR(temp, 1)
            temperaturas[quantidade] ← temp
            quantidade ← quantidade + 1
        SENÃO
            ESCREVER "Valor inválido: ", valor, ". Ignorado."
        FIM TENTAR
    FIM PARA
FIM

FUNÇÃO gravarTemperaturas(temperaturas: real[], quantidade: inteiro, data: string): void
INÍCIO
    ABRIR arquivo(ARQUIVO, APPEND)
    SE arquivo NÃO ABERTO ENTÃO
        ESCREVER "Erro ao abrir arquivo para gravação!"
        RETORNAR
    FIM SE

    PARA i DE 0 ATÉ quantidade - 1 FAÇA
        ESCREVER_ARQUIVO(arquivo, data, ",", FORMATAR_HORA(i), ",", ARREDONDAR(temperatur
    FIM PARA
    FECHAR arquivo
FIM

FUNÇÃO lerDadosTemperaturas(temperaturas: real[], quantidade: inteiro, dataFiltro: string
INÍCIO
    ABRIR arquivo(ARQUIVO, LEITURA)
    SE arquivo NÃO ABERTO ENTÃO
        RETORNAR FALSO
    FIM SE

    quantidade ← 0
    LER_LINHA(arquivo, linha)
    SE linha CONTER "Data" ENTÃO
        // Cabeçalho detectado
    SENÃO
        REPOSICIONAR_ARQUIVO(arquivo, INÍCIO)
    FIM SE

    ENQUANTO LER_LINHA(arquivo, linha) E quantidade < MAX_TEMPERATURAS FAÇA
        SEPARAR(linha, ",", data, hora, tempStr)
        SE dataFiltro = "" OU data = dataFiltro ENTÃO
            TENTAR
                temp ← CONVERTER_PARA_REAL(tempStr)
                temperaturas[quantidade] ← ARREDONDAR(temp, 1)
                quantidade ← quantidade + 1
            SENÃO
                // Ignorar linhas inválidas
            FIM TENTAR
        FIM SE
```

```
    FIM ENQUANTO

    SE quantidade = MAX_TEMPERATURAS E MAIS_LINHAS(arquivo) ENTÃO
        ESCREVER "Aviso: Mais registros para a data do que o limite de ", MAX_TEMPERATURA
    FIM SE

    FECHAR arquivo
    RETORNAR VERDADEIRO
FIM

FUNÇÃO calcularEstatisticas(temperaturas: real[], quantidade: inteiro, media: real): void
INÍCIO
    SE quantidade = 0 ENTÃO
        media ← 0.0
        RETORNAR
    FIM SE
    soma ← 0.0
    PARA i DE 0 ATÉ quantidade - 1 FAÇA
        soma ← soma + temperaturas[i]
    FIM PARA
    media ← soma / quantidade
FIM

FUNÇÃO encontrarExtremos(temperaturas: real[], quantidade: inteiro, minima: real, maxima:
INÍCIO
    SE quantidade = 0 ENTÃO
        minima ← 0.0
        maxima ← 0.0
        RETORNAR
    FIM SE
    minima ← temperaturas[0]
    maxima ← temperaturas[0]
    PARA i DE 1 ATÉ quantidade - 1 FAÇA
        SE temperaturas[i] < minima ENTÃO
            minima ← temperaturas[i]
        FIM SE
        SE temperaturas[i] > maxima ENTÃO
            maxima ← temperaturas[i]
        FIM SE
    FIM PARA
FIM

FUNÇÃO calcularDesvio(temperaturas: real[], quantidade: inteiro, media: real): real
INÍCIO
    SE quantidade = 0 ENTÃO
        RETORNAR 0.0
    FIM SE
    soma ← 0.0
    PARA i DE 0 ATÉ quantidade - 1 FAÇA
        soma ← soma + (temperaturas[i] - media) * (temperaturas[i] - media)
    FIM PARA
```

```
    RETORNAR RAIZ_QUADRADA(soma / quantidade)
FIM

FUNÇÃO exibirMenu(): void
INÍCIO
    ESCREVER "=== MENU PRINCIPAL ==="
    ESCREVER "1. Registrar temperaturas"
    ESCREVER "2. Relatório por dia"
    ESCREVER "3. Relatório mensal"
    ESCREVER "0. Sair"
FIM

FUNÇÃO VALIDAR_DATA(data: string): booleano
INÍCIO
    SE tamanho(data)  10 OU data[2]  '/' OU data[5]  '/' ENTÃO
        RETORNAR FALSO
    FIM SE
    TENTAR
        dia ← CONVERTER_PARA_INTEIRO(data[0:2])
        mes ← CONVERTER_PARA_INTEIRO(data[3:5])
        ano ← CONVERTER_PARA_INTEIRO(data[6:10])
        SE dia  1 E dia  31 E mes  1 E mes  12 E ano  2000 E ano  2100 ENTÃO
            RETORNAR VERDADEIRO
        FIM SE
    SENÃO
        RETORNAR FALSO
    FIM TENTAR
    RETORNAR FALSO
FIM

FUNÇÃO VALIDAR_MES_ANO(mesAno: string): booleano
INÍCIO
    SE tamanho(mesAno)  7 OU mesAno[2]  '/' ENTÃO
        RETORNAR FALSO
    FIM SE
    TENTAR
        mes ← CONVERTER_PARA_INTEIRO(mesAno[0:2])
        ano ← CONVERTER_PARA_INTEIRO(mesAno[3:7])
        SE mes  1 E mes  12 E ano  2000 E ano  2100 ENTÃO
            RETORNAR VERDADEIRO
        FIM SE
    SENÃO
        RETORNAR FALSO
    FIM TENTAR
    RETORNAR FALSO
FIM

FUNÇÃO FORMATAR_HORA(indice: inteiro): string
INÍCIO
    hora ← indice
    RETORNAR to_string(hora) + ":00"
```

```
FIM
```

7.2.5.2 Decomposição: Divisão de Tarefas para o Problema L2

O problema **L2** propõe um sistema para gerenciar dados de temperatura coletados por sensores durante um mês, armazenando até 30 temperaturas diárias em um array, salvando em um arquivo CSV e gerando estatísticas (média, extremos, desvio padrão) e relatórios. Abaixo está a divisão de tarefas entre quatro alunos, trabalhando independentemente na mesma sala, sem controle de versão, com interfaces claras para facilitar a integração. O pseudocódigo completo é apresentado em seguida.

Aluno 1: Função `lerTemperaturas` e Validação de Entrada
Responsabilidades:

- Implementar `lerTemperaturas(float temperaturas[], int& quantidade, string data)` para coletar até 30 temperaturas diárias (em °C, com uma casa decimal) via linha de comando, separadas por vírgulas, associadas a uma data específica.
- Validar:
  - Temperaturas como valores `float`, com uma casa decimal (ex.: 23.5);
  - Quantidade de temperaturas não exceder 30;
  - Data no formato "DD/MM/AAAA" (ex.: 01/07/2025);
  - Não zerar `quantidade` para preservar registros anteriores;
- Permitir entrada de múltiplas temperaturas em uma única linha (ex.: "23.5,24.0,22.8") ou uma por vez até o usuário indicar "sair";
- Tratar buffer de entrada adequadamente (ex.: corrigir uso de `std::cin.ignore` em C++).

Tarefas específicas:

- Escrever a função com entrada via `getline` para data e temperaturas;
- Validar data (ex.: verificar formato "DD/MM/AAAA" com 10 caracteres e barras corretas);
- Validar temperaturas (ex.: converter com `stod`, verificar uma casa decimal usando `round`);
- Adicionar temperaturas ao array a partir do índice `quantidade`;
- Testar isoladamente com entradas válidas (ex.: "01/07/2025,23.5,24.0") e inválidas (ex.: "01-07-2025", "23.55", " ").

Interface:

- Recebe `temperaturas[]`, `quantidade` e `data` como parâmetros;
- Fornece dados válidos para `gravarTemperaturas` e análises.

Aluno 2: Função `gravarTemperaturas` e Inicialização do Arquivo
Responsabilidades:

- Implementar `gravarTemperaturas(const float temperaturas[], int quantidade, const string& data)` para gravar temperaturas em `temperaturas.csv` (formato: "Data,Hora,Temperatura");
- Adicionar registros ao arquivo (ex.: `std::ios::app` em C++) sem sobrescrever dados anteriores;

- Implementar `inicializarArquivo()` para criar o arquivo CSV com cabeçalho "Data,Hora,Temperatura" se não existir;
- Verificar erros ao abrir/escrever no arquivo.

Tarefas específicas:

- Gravar cada temperatura com data, hora (ex.: "00:00" a "23:00") e valor em uma linha do CSV;
- Verificar abertura do arquivo e exibir mensagens de erro;
- Testar isoladamente com diferentes quantidades de temperaturas e verificar o arquivo gerado.

Interface:

- Recebe `temperaturas[]`, `quantidade` e `data` de `lerTemperaturas`;
- Produz um arquivo CSV lido por `lerDadosTemperaturas`.

Aluno 3: Função `lerDadosTemperaturas` e Carregamento Inicial
Responsabilidades:

- Implementar `lerDadosTemperaturas(float temperaturas[], int& quantidade, const string& dataFiltro = "")` para ler o arquivo CSV e preencher o array com temperaturas de um dia específico ou todos os dados;
- Suportar filtragem por `dataFiltro` (ex.: "01/07/2025") ou leitura completa se vazio;
- Alertar se o número de temperaturas exceder 30 para um dia;
- Carregar dados do arquivo no início do programa (no `main`) para sincronizar o array local;

Tarefas específicas:

- Ler o CSV, ignorar o cabeçalho e preencher o array com temperaturas;
- Implementar filtragem por `dataFiltro` quando não vazio;
- Contar registros e alertar se exceder 30 por dia;
- Modificar o `main` para chamar `lerDadosTemperaturas` no início;
- Testar com arquivos CSV vazios, com menos de 30 temperaturas e com múltiplos dias.

Interface:

- Lê o arquivo CSV de `gravarTemperaturas`;
- Fornece dados para `calcularEstatisticas` e `encontrarExtremos`.

Aluno 4: Funções `calcularEstatisticas`, `encontrarExtremos`, `calcularDesvio`, Relatório e Interface do Usuário
Responsabilidades:

- Implementar `calcularEstatisticas(const float temperaturas[], int quantidade, float& media)` para calcular a média das temperaturas;
- Implementar `encontrarExtremos(const float temperaturas[], int quantidade, float& minima, float& maxima)` para identificar temperaturas mínima e máxima;
- Implementar `calcularDesvio(const float temperaturas[], int quantidade, float media)` para calcular o desvio padrão;
- Implementar o relatório completo (opção 3 do menu) para listar estatísticas do mês (média, extremos, desvio padrão, dias com temperaturas extremas);

- Escrever `exibirMenu()` e gerenciar a interface no `main`, corrigindo problemas de buffer (ex.: `std::cin.ignore` em C++);
- Garantir saídas formatadas (ex.: uma casa decimal para temperaturas e média, duas casas para desvio padrão).

Tarefas específicas:

- Calcular média, extremos e desvio padrão para um dia ou mês;
- Identificar dias com temperaturas extremas (ex.: fora de 2 desvios padrão da média mensal);
- Chamar `lerDadosTemperaturas` com filtro de data para relatórios diários;
- Implementar relatório completo no `main`, usando `lerDadosTemperaturas` sem filtro;
- Corrigir `std::cin.ignore` no `main` (ex.: `std::cin.ignore(std::numeric_limits<std::streamsize '\n')` após leitura de opção);
- Testar com busca de dias existentes/inexistentes, relatórios diários e mensais.

Interface:

- Usa dados de `lerDadosTemperaturas` para exibir resultados;
- Interage com o usuário, fornecendo datas para `lerTemperaturas` e relatórios.

7.2.5.3 Considerações para Integração

Interfaces:

- Os alunos devem concordar com o formato do CSV ("Data,Hora,Temperatura") e a estrutura do array `temperaturas[]`;
- O array armazena até 30 temperaturas por dia, e `quantidade` rastreia o número de registros.

Depuração:

- Cada aluno deve depurar sua parte isoladamente, simulando entradas/saídas (ex.: arquivos de teste, arrays pré-preenchidos);

Passos importantes da integração:

1. Copiar `main` e `exibirMenu` do Aluno 4;
2. Inserir `inicializarArquivo` e `gravarTemperaturas` do Aluno 2;
3. Adicionar `lerDadosTemperaturas` do Aluno 3, ajustando o `main` para carregar dados iniciais;
4. Incluir `lerTemperaturas` do Aluno 1, `calcularEstatisticas`, `encontrarExtremos` e `calcularDesvio` do Aluno 4.

Resolução de conflitos:

- Discutir problemas imediatamente na sala (ex.: formato do CSV, validação de data);
- Testar o programa completo com cenários como registrar temperaturas, buscar por dia e exibir relatórios.

Depuração:

- Verificar registro de temperaturas, busca por dia, e relatório mensal com estatísticas;
- Testar casos extremos (arquivo vazio, data inválida, mais de 30 temperaturas).

7.2.5.4 Código C++23

O código correspondente ao problema L2 pode ser encontrado em https://onlinegdb.com/zAYiGjouh.

7.2.6 M2

Um laboratório de pesquisas precisa de uma calculadora que mantenha histórico completo das operações para auditoria e análise posterior. A calculadora deve realizar operações complexas e permitir análise estatística do histórico. Desenvolva uma calculadora científica que execute operações matemáticas avançadas, mantenha um array com histórico das operações, permita buscar operações anteriores e calcule estatísticas sobre o uso.

Funções a implementar:

- **Funções próprias**: `adicionarHistorico()`, `mostrarHistorico()`, `buscarOperacao()`, `analisarUso()`
- **Funções padrão**: `sin()`, `cos()`, `tan()`, `exp()`, `log10()`

Entrada: Expressões matemáticas e comandos de histórico **Saída**: Resultados, histórico completo e análises estatísticas de uso

7.2.6.1 Algoritmização: Pseudocódigo

```
ALGORITMO CalculadoraCientifica

    // Constantes
    MAX_OPERACOES ← 100
    ARQUIVO ← "historico.csv"

    // Declaração de arrays
    historico[MAX_OPERACOES]: string
    quantidade: inteiro

INÍCIO
    // Inicializar sistema
    ESCREVER "=== Calculadora Científica com Histórico ==="
    ESCREVER "Inicializando sistema..."
    inicializarArquivo()

    // Carregar dados existentes
    mostrarHistorico(historico, quantidade)

    FAÇA
        exibirMenu()
        ESCREVER "Escolha uma opção: "
        LER opcao

        SE opcao = 1 ENTÃO
            ESCREVER "Digite a expressão matemática (ou 'sair'): "
            LER expressao
            SE expressao   "sair" ENTÃO
                SE VALIDAR_EXPRESSAO(expressao) ENTÃO
```

```
                    resultado ← CALCULAR_EXPRESSAO(expressao)
                    adicionarHistorico(historico, quantidade, expressao, resultado)
                    gravarHistorico(historico, quantidade)
                    ESCREVER "Resultado: ", ARREDONDAR(resultado, 2)
                SENÃO
                    ESCREVER "Expressão inválida! Use sin(), cos(), tan(), exp(), log10()
                FIM SE
            FIM SE
        SENÃO SE opcao = 2 ENTÃO
            ESCREVER "Digite a expressão para busca (ex.: 'sin', ou vazio para todas): "
            LER expressaoFiltro
            buscarOperacao(expressaoFiltro)
        SENÃO SE opcao = 3 ENTÃO
            historicoTemp[MAX_OPERACOES]: string
            totalRegistros: inteiro
            SE mostrarHistorico(historicoTemp, totalRegistros) = FALSO ENTÃO
                ESCREVER "Erro ao ler dados ou arquivo não encontrado!"
            SENÃO SE totalRegistros = 0 ENTÃO
                ESCREVER "Nenhuma operação encontrada!"
            SENÃO
                ESCREVER "=== RELATÓRIO COMPLETO ==="
                PARA i DE 0 ATÉ totalRegistros - 1 FAÇA
                    SEPARAR(historicoTemp[i], ",", expressao, resultadoStr)
                    TENTAR
                        resultado ← CONVERTER_PARA_REAL(resultadoStr)
                        ESCREVER "Operação ", i + 1, ": ", expressao, " = ", ARREDONDAR(r
                    SENÃO
                        ESCREVER "Erro ao processar resultado em: ", historicoTemp[i]
                    FIM TENTAR
                FIM PARA
                ESCREVER "Total de operações: ", totalRegistros
                analisarUso(historicoTemp, totalRegistros)
            FIM SE
        SENÃO SE opcao = 0 ENTÃO
            ESCREVER "Encerrando sistema..."
        SENÃO
            ESCREVER "Opção inválida! Tente novamente."
        FIM SE

        SE opcao   0 ENTÃO
            ESCREVER "Pressione Enter para continuar..."
            LER pausa
        FIM SE
    ENQUANTO opcao   0

FIM

FUNÇÃO adicionarHistorico(historico: string[], quantidade: inteiro, expressao: string, re
INÍCIO
    SE quantidade   MAX_OPERACOES ENTÃO
        ESCREVER "Limite de operações atingido!"
```

```
        RETORNAR
    FIM SE

    SE expressao = "" OU expressao CONTER "," ENTÃO
        ESCREVER "Expressão inválida! Deve ser não vazia e sem vírgulas."
        RETORNAR
    FIM SE

    historico[quantidade] ← expressao + "," + to_string(ARREDONDAR(resultado, 2))
    quantidade ← quantidade + 1
FIM

FUNÇÃO gravarHistorico(historico: string[], quantidade: inteiro): void
INÍCIO
    ABRIR arquivo(ARQUIVO, APPEND)
    SE arquivo NÃO ABERTO ENTÃO
        ESCREVER "Erro ao abrir arquivo para gravação!"
        RETORNAR
    FIM SE

    PARA i DE quantidade - 1 ATÉ quantidade - 1 FAÇA
        ESCREVER_ARQUIVO(arquivo, historico[i])
    FIM PARA
    FECHAR arquivo
FIM

FUNÇÃO mostrarHistorico(historico: string[], quantidade: inteiro, expressaoFiltro: string
INÍCIO
    ABRIR arquivo(ARQUIVO, LEITURA)
    SE arquivo NÃO ABERTO ENTÃO
        RETORNAR FALSO
    FIM SE

    quantidade ← 0
    LER_LINHA(arquivo, linha)
    SE linha CONTER "Expressao" ENTÃO
        // Cabeçalho detectado
    SENÃO
        REPOSICIONAR_ARQUIVO(arquivo, INÍCIO)
    FIM SE

    ENQUANTO LER_LINHA(arquivo, linha) E quantidade < MAX_OPERACOES FAÇA
        SEPARAR(linha, ",", expressao, resultadoStr)
        SE expressaoFiltro = "" OU expressao CONTER expressaoFiltro ENTÃO
            TENTAR
                resultado ← CONVERTER_PARA_REAL(resultadoStr)
                historico[quantidade] ← linha
                quantidade ← quantidade + 1
            SENÃO
                // Ignorar linhas inválidas
            FIM TENTAR
```

```
        FIM SE
    FIM ENQUANTO

    SE quantidade = MAX_OPERACOES E MAIS_LINHAS(arquivo) ENTÃO
        ESCREVER "Aviso: Mais registros no arquivo do que o limite de ", MAX_OPERACOES
    FIM SE

    FECHAR arquivo
    RETORNAR VERDADEIRO
FIM

FUNÇÃO buscarOperacao(expressao: string): void
INÍCIO
    historicoTemp[MAX_OPERACOES]: string
    totalRegistros: inteiro

    SE mostrarHistorico(historicoTemp, totalRegistros, expressao) = FALSO ENTÃO
        ESCREVER "Erro ao ler dados!"
        RETORNAR
    FIM SE

    SE totalRegistros = 0 ENTÃO
        ESCREVER "Nenhuma operação encontrada para '", expressao, "'!"
        RETORNAR
    FIM SE

    ESCREVER "=== OPERAÇÕES COM '", expressao, "' ==="
    PARA i DE 0 ATÉ totalRegistros - 1 FAÇA
        SEPARAR(historicoTemp[i], ",", expressaoEncontrada, resultadoStr)
        TENTAR
            resultado ← CONVERTER_PARA_REAL(resultadoStr)
            ESCREVER "Operação ", i + 1, ": ", expressaoEncontrada, " = ", ARREDONDAR(res
        SENÃO
            ESCREVER "Erro ao processar resultado em: ", historicoTemp[i]
        FIM TENTAR
    FIM PARA
    ESCREVER "Total de operações encontradas: ", totalRegistros
FIM

FUNÇÃO analisarUso(historico: string[], quantidade: inteiro): void
INÍCIO
    contSin ← 0
    contCos ← 0
    contTan ← 0
    contExp ← 0
    contLog ← 0
    contOutros ← 0

    PARA i DE 0 ATÉ quantidade - 1 FAÇA
        SEPARAR(historico[i], ",", expressao, resultadoStr)
        SE expressao CONTER "sin(" ENTÃO
```

```
            contSin ← contSin + 1
        SENÃO SE expressao CONTER "cos(" ENTÃO
            contCos ← contCos + 1
        SENÃO SE expressao CONTER "tan(" ENTÃO
            contTan ← contTan + 1
        SENÃO SE expressao CONTER "exp(" ENTÃO
            contExp ← contExp + 1
        SENÃO SE expressao CONTER "log10(" ENTÃO
            contLog ← contLog + 1
        SENÃO
            contOutros ← contOutros + 1
        FIM SE
    FIM PARA

    ESCREVER "=== ANÁLISE DE USO ==="
    ESCREVER "Operações com sin: ", contSin
    ESCREVER "Operações com cos: ", contCos
    ESCREVER "Operações com tan: ", contTan
    ESCREVER "Operações com exp: ", contExp
    ESCREVER "Operações com log10: ", contLog
    ESCREVER "Outras operações: ", contOutros
FIM

FUNÇÃO inicializarArquivo(): void
INÍCIO
    SE ARQUIVO NÃO EXISTIR ENTÃO
        ABRIR arquivo(ARQUIVO, ESCRITA)
        SE arquivo ABERTO ENTÃO
            ESCREVER_ARQUIVO(arquivo, "Expressao,Resultado")
            FECHAR arquivo
            ESCREVER "Arquivo de dados criado: ", ARQUIVO
        SENÃO
            ESCREVER "Erro ao criar arquivo de dados!"
        FIM SE
    SENÃO
        ESCREVER "Arquivo de dados encontrado: ", ARQUIVO
    FIM SE
FIM

FUNÇÃO exibirMenu(): void
INÍCIO
    ESCREVER "=== MENU PRINCIPAL ==="
    ESCREVER "1. Realizar operação"
    ESCREVER "2. Buscar operações por expressão"
    ESCREVER "3. Relatório completo"
    ESCREVER "0. Sair"
FIM

FUNÇÃO VALIDAR_EXPRESSAO(expressao: string): booleano
INÍCIO
    SE expressao = "" ENTÃO
```

```
        RETORNAR FALSO
    FIM SE
    // Verifica se a expressão contém funções válidas ou operações básicas
    SE expressao CONTER "sin(" OU expressao CONTER "cos(" OU expressao CONTER "tan(" OU
       expressao CONTER "exp(" OU expressao CONTER "log10(" OU
       expressao CONTER "+" OU expressao CONTER "-" OU expressao CONTER "*" OU expressao
        RETORNAR VERDADEIRO
    FIM SE
    RETORNAR FALSO
FIM

FUNÇÃO CALCULAR_EXPRESSAO(expressao: string): real
INÍCIO
    // Implementação simplificada: supõe que a expressão é válida e usa funções padrão
    SE expressao CONTER "sin(" ENTÃO
        valor ← EXTRAIR_NUMERO(expressao)
        RETORNAR SIN(valor)
    SENÃO SE expressao CONTER "cos(" ENTÃO
        valor ← EXTRAIR_NUMERO(expressao)
        RETORNAR COS(valor)
    SENÃO SE expressao CONTER "tan(" ENTÃO
        valor ← EXTRAIR_NUMERO(expressao)
        RETORNAR TAN(valor)
    SENÃO SE expressao CONTER "exp(" ENTÃO
        valor ← EXTRAIR_NUMERO(expressao)
        RETORNAR EXP(valor)
    SENÃO SE expressao CONTER "log10(" ENTÃO
        valor ← EXTRAIR_NUMERO(expressao)
        RETORNAR LOG10(valor)
    SENÃO
        // Supõe operação básica (ex.: "2 + 3")
        TENTAR
            resultado ← AVALIAR_OPERACAO_BASICA(expressao)
            RETORNAR resultado
        SENÃO
            RETORNAR 0.0
        FIM TENTAR
    FIM SE
FIM
```

7.2.6.2 Decomposição: Divisão de Tarefas para o Problema M2

O problema **M2** propõe uma calculadora científica que executa operações matemáticas avançadas, armazena um histórico de operações em um array, permite consultas ao histórico e realiza análises estatísticas do uso. O histórico é salvo em um arquivo CSV para auditoria. Abaixo está a divisão de tarefas entre quatro alunos, trabalhando independentemente na mesma sala, sem controle de versão, com interfaces claras para facilitar a integração. O pseudocódigo completo é apresentado em seguida.

Aluno 1: Função `adicionarHistorico` e Validação de Entrada

Responsabilidades:

- Implementar `adicionarHistorico(string historico[], int& quantidade,`

`string expressao, real resultado)` para coletar e armazenar expressões matemáticas (string) e seus resultados (real) no array de histórico;
- Validar:
  - Expressão não vazia e sem vírgulas (para evitar corromper o CSV);
  - Resultado como número real válido;
  - Quantidade não exceder `MAX_OPERACOES` (definido como 100, por exemplo);
- Interpretar expressões matemáticas (ex.: "sin(3.14)", "2 + 3 * 4", "log10(100)") e calcular resultados usando `sin`, `cos`, `tan`, `exp`, `log10`;
- Tratar buffer de entrada adequadamente (ex.: corrigir uso de `std::cin.ignore` em C++).

Tarefas específicas:

- Escrever a função para ler expressões via `getline` e calcular resultados;
- Validar expressão (ex.: `if (expressao.empty() || expressao.find(',') != string::npos)` em C++);
- Adicionar par "expressao,resultado" ao array a partir do índice `quantidade`;
- Testar isoladamente com expressões válidas (ex.: "sin(0)", "2 + 2") e inválidas (ex.: "","sin,a").

Interface:

- Recebe `historico[]`, `quantidade`, `expressao` e `resultado` como parâmetros;
- Fornece dados válidos para `gravarHistorico` e análises.

Aluno 2: Função `gravarHistorico` e Inicialização do Arquivo
Responsabilidades:

- Implementar `gravarHistorico(const string historico[], int quantidade)` para gravar o histórico em `historico.csv` (formato: "Expressao,Resultado");
- Adicionar registros ao arquivo (ex.: `std::ios::app` em C++) para preservar o histórico anterior;
- Implementar `inicializarArquivo()` para criar o arquivo CSV com cabeçalho "Expressao,Resultado" se não existir;
- Verificar erros ao abrir/escrever no arquivo.

Tarefas específicas:

- Gravar cada entrada do histórico (expressão e resultado) em uma linha do CSV;
- Verificar abertura do arquivo e exibir mensagens de erro;
- Testar isoladamente com diferentes tamanhos de arrays e verificar o arquivo gerado.

Interface:

- Recebe `historico[]` e `quantidade` de `adicionarHistorico`;
- Produz um arquivo CSV lido por `mostrarHistorico`.

Aluno 3: Função `mostrarHistorico` e Carregamento Inicial
Responsabilidades:

- Implementar `mostrarHistorico(string historico[], int& quantidade, const string& expressaoFiltro = "")` para ler o arquivo CSV e preencher o array com o histórico, filtrando por expressão (*substring*) se especificado;

- Suportar leitura de todo o histórico ou filtragem por `expressaoFiltro`;
- Alertar se o número de operações exceder `MAX_OPERACOES`;
- Carregar dados do arquivo no início do programa (no `main`) para sincronizar o array local.

Tarefas específicas:

- Ler o CSV, ignorar o cabeçalho e preencher o array com strings no formato "expressao,resultado";
- Implementar filtragem por `expressaoFiltro` quando não vazio;
- Contar registros e alertar se exceder `MAX_OPERACOES`;
- Modificar o `main` para chamar `mostrarHistorico` no início;
- Testar com arquivos CSV vazios, com poucas operações e com mais de 100.

Interface:

- Lê o arquivo CSV de `gravarHistorico`;
- Fornece dados para `buscarOperacao` e análises.

Aluno 4: Função `buscarOperacao`, `analisarUso`, Relatório e Interface do Usuário
Responsabilidades:

- Implementar `buscarOperacao(const string& expressao)` para exibir operações cujo nome (ex.: "sin", "cos") ou expressão completa contenha a substring fornecida;
- Implementar `analisarUso(const string historico[], int quantidade)` para calcular estatísticas de uso (ex.: número de operações por tipo, frequência de cada função);
- Implementar o relatório completo (opção 3 do menu) para listar todo o histórico e estatísticas de uso;
- Escrever `exibirMenu()` e gerenciar a interface no `main`, corrigindo problemas de buffer (ex.: `std::cin.ignore` em C++);
- Garantir saídas formatadas (ex.: resultados com duas casas decimais, mensagens claras).

Tarefas específicas:

- Chamar `mostrarHistorico` com filtro de expressão para buscas;
- Implementar análise estatística (ex.: contar uso de `sin`, `cos`, etc.);
- Implementar relatório completo no `main`, usando `mostrarHistorico` sem filtro;
- Corrigir `std::cin.ignore` no `main` (ex.: `std::cin.ignore(std::numeric_limits<std::streamsize '\n')` após leitura de opção);
- Testar com busca de expressões existentes/inexistentes e relatórios.

Interface:

- Usa dados de `mostrarHistorico` para exibir resultados;
- Interage com o usuário, fornecendo expressões para `adicionarHistorico` e `buscarOperacao`.

7.2.6.3 Considerações para Integração

Interfaces:

- Os alunos devem concordar com o formato do CSV ("Expressao,Resultado") e a estrutura do array `historico[]` (strings no formato "expressao,resultado");

- O array armazena até `MAX_OPERACOES` entradas, e `quantidade` rastreia o número de registros.

Depuração:

- Cada aluno deve depurar sua parte isoladamente, simulando entradas/saídas (ex.: arquivos de teste, arrays pré-preenchidos).

Passos importantes da integração:

1. Copiar `main` e `exibirMenu` do Aluno 4;
2. Inserir `inicializarArquivo` e `gravarHistorico` do Aluno 2;
3. Adicionar `mostrarHistorico` do Aluno 3, ajustando o `main` para carregar dados iniciais;
4. Incluir `adicionarHistorico`, `buscarOperacao` e `analisarUso` do Aluno 4.

Resolução de conflitos:

- Discutir problemas imediatamente na sala (ex.: formato do CSV, validação de expressões);
- Testar o programa completo com cenários como adicionar operações, buscar por expressão e exibir relatórios.

Depuração:

- Verificar registro de operações, busca por expressão, e relatório com estatísticas.
- Testar casos extremos (arquivo vazio, expressão inválida, mais de 100 operações).

7.2.6.4 Código C++23

O código correspondente ao problema M2 pode ser encontrado em https://onlinegdb.com/UHaBupDCz.

7.2.7 N2

Uma loja de departamentos precisa organizar as vendas diárias por categoria de produtos. Cada categoria (ex.: "Roupas", "Eletrônicos") contém uma lista de valores de vendas (decimais). O programa deve usar vetores de vetores para armazenar as vendas, salvar em disco e permitir consultas. Desenvolva um sistema que registre vendas por categoria, mantenha os dados em vetores de vetores, salve em um arquivo CSV e permita consultar todas as vendas ou as de uma categoria específica.

Funções a implementar:

- **Funções próprias:** `registrarVendas()`, `salvarVendas()`, `lerVendas()`, `buscarVendasCategoria(`
- **Funções padrão:** `getline()`, `stod()`, `to_string()`

Entrada: Nome da categoria (string) e valores de vendas (decimais, positivos) **Saída:** Lista de todas as vendas por categoria ou vendas de uma categoria específica, com soma total das vendas

7.2.7.1 Algoritmização: Pseudocódigo

```
ALGORITMO GerenciadorVendas

    // Constantes
    MAX_CATEGORIAS ← 50
    ARQUIVO ← "vendas.csv"

    // Declaração de vetores
    vendas: vetor de vetores de real
    categorias: vetor de string
    numCategorias: inteiro

INÍCIO
    // Inicializar sistema
    ESCREVER "=== Sistema de Gerenciamento de Vendas ==="
    ESCREVER "Inicializando sistema..."
    inicializarArquivo()

    // Carregar dados existentes
    lerVendas(vendas, categorias, numCategorias)

    FAÇA
        exibirMenu()
        ESCREVER "Escolha uma opção: "
        LER opcao

        SE opcao = 1 ENTÃO
            registrarVendas(vendas, categorias, numCategorias)
            salvarVendas(vendas, categorias, numCategorias)
            ESCREVER "Venda(s) registrada(s)."
        SENÃO SE opcao = 2 ENTÃO
            ESCREVER "Digite a categoria para busca: "
            LER categoria
            buscarVendasCategoria(categoria)
        SENÃO SE opcao = 3 ENTÃO
            vendasTemp: vetor de vetores de real
            categoriasTemp: vetor de string
            totalRegistros: inteiro
            SE lerVendas(vendasTemp, categoriasTemp, totalRegistros) = FALSO ENTÃO
                ESCREVER "Erro ao ler dados ou arquivo não encontrado!"
            SENÃO SE totalRegistros = 0 ENTÃO
                ESCREVER "Nenhuma venda encontrada!"
            SENÃO
                ESCREVER "=== RELATÓRIO COMPLETO ==="
                somaTotal ← 0.0
                PARA i DE 0 ATÉ totalRegistros - 1 FAÇA
                    ESCREVER "Categoria: ", categoriasTemp[i]
                    somaCategoria ← 0.0
                    PARA CADA valor EM vendasTemp[i] FAÇA
                        ESCREVER "  Venda: R$", ARREDONDAR(valor, 2)
                        somaCategoria ← somaCategoria + valor
                    FIM PARA
```

```
                    ESCREVER "  Soma da categoria: R$", ARREDONDAR(somaCategoria, 2)
                    somaTotal ← somaTotal + somaCategoria
                FIM PARA
                ESCREVER "Total de categorias: ", totalRegistros
                ESCREVER "Soma total das vendas: R$", ARREDONDAR(somaTotal, 2)
            FIM SE
        SENÃO SE opcao = 0 ENTÃO
            ESCREVER "Encerrando sistema..."
        SENÃO
            ESCREVER "Opção inválida! Tente novamente."
        FIM SE

        SE opcao   0 ENTÃO
            ESCREVER "Pressione Enter para continuar..."
            LER pausa
        FIM SE
    ENQUANTO opcao   0

FIM

FUNÇÃO registrarVendas(vendas: vetor de vetores de real, categorias: vetor de string, num
INÍCIO
    FAÇA
        ESCREVER "Digite a categoria (ou 'sair' para encerrar): "
        LER categoria
        SE categoria = "sair" ENTÃO
            RETORNAR
        FIM SE

        SE categoria = "" OU categoria CONTER "," ENTÃO
            ESCREVER "Categoria inválida! Deve ser não vazia e sem vírgulas."
            CONTINUAR
        FIM SE

        indiceCategoria ← -1
        PARA i DE 0 ATÉ numCategorias - 1 FAÇA
            SE categorias[i] = categoria ENTÃO
                indiceCategoria ← i
                SAIR
            FIM SE
        FIM PARA

        SE indiceCategoria = -1 ENTÃO
            SE numCategorias   MAX_CATEGORIAS ENTÃO
                ESCREVER "Limite de categorias atingido!"
                RETORNAR
            FIM SE
            categorias[numCategorias] ← categoria
            vendas[numCategorias]: novo vetor de real
            indiceCategoria ← numCategorias
            numCategorias ← numCategorias + 1
```

```
        FIM SE

        ESCREVER "Digite o valor da venda (positivo, ou 'sair'): "
        LER entrada
        SE entrada = "sair" ENTÃO
            CONTINUAR
        FIM SE

        TENTAR
            valor ← CONVERTER_PARA_REAL(entrada)
            SE valor > 0 ENTÃO
                vendas[indiceCategoria].ADICIONAR(valor)
            SENÃO
                ESCREVER "Valor inválido! Deve ser positivo."
            FIM SE
        SENÃO
            ESCREVER "Valor inválido para venda. Ignorado."
        FIM TENTAR
    ENQUANTO VERDADEIRO
FIM

FUNÇÃO salvarVendas(vendas: vetor de vetores de real, categorias: vetor de string, numCat
INÍCIO
    ABRIR arquivo(ARQUIVO, SOBRESCREVER)
    SE arquivo NÃO ABERTO ENTÃO
        ESCREVER "Erro ao abrir arquivo para gravação!"
        RETORNAR
    FIM SE

    ESCREVER_ARQUIVO(arquivo, "Categoria,Valor")
    PARA i DE 0 ATÉ numCategorias - 1 FAÇA
        PARA CADA valor EM vendas[i] FAÇA
            ESCREVER_ARQUIVO(arquivo, categorias[i], ",", ARREDONDAR(valor, 2))
        FIM PARA
    FIM PARA
    FECHAR arquivo
FIM

FUNÇÃO lerVendas(vendas: vetor de vetores de real, categorias: vetor de string, numCatego
INÍCIO
    ABRIR arquivo(ARQUIVO, LEITURA)
    SE arquivo NÃO ABERTO ENTÃO
        RETORNAR FALSO
    FIM SE

    numCategorias ← 0
    vendas.LIMPAR()
    categorias.LIMPAR()

    LER_LINHA(arquivo, linha)
    SE linha CONTER "Categoria" ENTÃO
```

```
        // Cabeçalho detectado
    SENÃO
        REPOSICIONAR_ARQUIVO(arquivo, INÍCIO)
    FIM SE

    ENQUANTO LER_LINHA(arquivo, linha) FAÇA
        SEPARAR(linha, ",", categoria, valorStr)
        SE categoriaFiltro = "" OU categoria = categoriaFiltro ENTÃO
            TENTAR
                valor ← CONVERTER_PARA_REAL(valorStr)
                indiceCategoria ← -1
                PARA i DE 0 ATÉ numCategorias - 1 FAÇA
                    SE categorias[i] = categoria ENTÃO
                        indiceCategoria ← i
                        SAIR
                    FIM SE
                FIM PARA
                SE indiceCategoria = -1 ENTÃO
                    categorias[numCategorias] ← categoria
                    vendas[numCategorias]: novo vetor de real
                    indiceCategoria ← numCategorias
                    numCategorias ← numCategorias + 1
                FIM SE
                vendas[indiceCategoria].ADICIONAR(valor)
            SENÃO
                // Ignorar linhas inválidas
            FIM TENTAR
        FIM SE
    FIM ENQUANTO

    FECHAR arquivo
    RETORNAR VERDADEIRO
FIM

FUNÇÃO buscarVendasCategoria(categoria: string): void
INÍCIO
    vendasTemp: vetor de vetores de real
    categoriasTemp: vetor de string
    totalRegistros: inteiro

    SE lerVendas(vendasTemp, categoriasTemp, totalRegistros, categoria) = FALSO ENTÃO
        ESCREVER "Erro ao ler dados!"
        RETORNAR
    FIM SE

    SE totalRegistros = 0 ENTÃO
        ESCREVER "Nenhuma venda encontrada para '", categoria, "'!"
        RETORNAR
    FIM SE

    ESCREVER "=== VENDAS DA CATEGORIA ", categoria, " ==="
```

```
    somaCategoria ← 0.0
    PARA CADA valor EM vendasTemp[0] FAÇA
        ESCREVER "Venda: R$", ARREDONDAR(valor, 2)
        somaCategoria ← somaCategoria + valor
    FIM PARA
    ESCREVER "Soma da categoria: R$", ARREDONDAR(somaCategoria, 2)
    ESCREVER "Total de vendas: ", tamanho(vendasTemp[0])
FIM

FUNÇÃO inicializarArquivo(): void
INÍCIO
    SE ARQUIVO NÃO EXISTIR ENTÃO
        ABRIR arquivo(ARQUIVO, ESCRITA)
        SE arquivo ABERTO ENTÃO
            ESCREVER_ARQUIVO(arquivo, "Categoria,Valor")
            FECHAR arquivo
            ESCREVER "Arquivo de dados criado: ", ARQUIVO
        SENÃO
            ESCREVER "Erro ao criar arquivo de dados!"
        FIM SE
    SENÃO
        ESCREVER "Arquivo de dados encontrado: ", ARQUIVO
    FIM SE
FIM

FUNÇÃO exibirMenu(): void
INÍCIO
    ESCREVER "=== MENU PRINCIPAL ==="
    ESCREVER "1. Registrar vendas"
    ESCREVER "2. Buscar vendas por categoria"
    ESCREVER "3. Relatório completo"
    ESCREVER "0. Sair"
FIM
```

7.2.7.2 Decomposição: Divisão de Tarefas para o Problema N2

O problema **N2** propõe um sistema para gerenciar vendas diárias de uma loja de departamentos, organizadas por categorias de produtos, usando vetores de vetores para armazenar valores de vendas (decimais), com persistência em arquivo CSV e funcionalidades de registro, salvamento, leitura e consulta por categoria. Abaixo está a divisão de tarefas entre quatro alunos, trabalhando independentemente na mesma sala, sem controle de versão, com interfaces claras para facilitar a integração. O pseudocódigo completo é apresentado em seguida.

Aluno 1: Função `registrarVendas` e Validação de Entrada
Responsabilidades:

- Implementar `registrarVendas(vetor de vetores de real vendas, vetor de string categorias, inteiro& numCategorias)` para coletar nome da categoria (string) e valores de vendas (decimais, positivos).
- Validar:
  - Nome da categoria não vazio e sem vírgulas (para evitar corromper o CSV);

  - Valores de vendas como decimais positivos, usando `stod` com tratamento de exceções;
  - Permitir múltiplas vendas por categoria até o usuário indicar "sair";
- Adicionar nova categoria ao vetor `categorias` se não existir, ou atualizar o vetor de vendas da categoria existente.

Tarefas específicas:

- Escrever a função com entrada via `getline` para categoria e valores de vendas;
- Validar categoria (ex.: `if (categoria.empty() || categoria.find(',') != string::npos)` em C++);
- Validar vendas (ex.: converter com `stod`, verificar se positivo);
- Adicionar vendas ao vetor de vetores, criando nova entrada em `categorias` se necessário;
- Testar isoladamente com entradas válidas (ex.: "Roupas,29.99,49.99") e inválidas (ex.: "","Eletrônicos,pão", "-10.0").

Interface:

- Recebe `vendas` (vetor de vetores de real), `categorias` (vetor de string) e `numCategorias` como parâmetros;
- Fornece dados válidos para `salvarVendas` e consultas.

Aluno 2: Função `salvarVendas` e Inicialização do Arquivo
Responsabilidades:

- Implementar `salvarVendas(const vetor de vetores de real vendas, const vetor de string categorias, inteiro numCategorias)` para gravar vendas em `vendas.csv` (formato: "Categoria,Valor");
- Sobrescrever o arquivo (ex.: `std::ios::trunc` em C++) para evitar duplicatas;
- Implementar `inicializarArquivo()` para criar o arquivo CSV com cabeçalho "Categoria,Valor" se não existir.
- Verificar erros ao abrir/escrever no arquivo;

Tarefas específicas:

- Gravar cada venda com sua categoria em uma linha do CSV, sobrescrevendo o arquivo;
- Preservar o cabeçalho ao sobrescrever;
- Verificar abertura do arquivo e exibir mensagens de erro;
- Testar isoladamente com diferentes tamanhos de vetores e verificar o arquivo gerado.

Interface:

- Recebe `vendas`, `categorias` e `numCategorias` de `registrarVendas`;
- Produz um arquivo CSV lido por `lerVendas`.

Aluno 3: Função `lerVendas` e Carregamento Inicial
Responsabilidades:

- Implementar `lerVendas(vetor de vetores de real vendas, vetor de string categorias, inteiro& numCategorias, const string& categoriaFiltro = "")` para ler o arquivo CSV e preencher os vetores;
- Suportar leitura de todas as vendas ou filtragem por categoria (exata);

- Carregar dados do arquivo no início do programa (no `main`) para sincronizar os vetores locais.

Tarefas específicas:

- Ler o CSV, ignorar o cabeçalho e preencher `vendas` e `categorias`;
- Implementar filtragem por `categoriaFiltro` quando não vazio;
- Modificar o `main` para chamar `lerVendas` no início;
- Testar com arquivos CSV vazios, com poucas vendas e com múltiplas categorias.

Interface:

- Lê o arquivo CSV de `salvarVendas`;
- Fornece dados para `buscarVendasCategoria` e relatório completo.

Aluno 4: Função `buscarVendasCategoria`, Relatório e Interface do Usuário
Responsabilidades:

- Implementar `buscarVendasCategoria(const string& categoria)` para exibir vendas de uma categoria específica, com soma dos valores;
- Implementar o relatório completo (opção 3 do menu) para listar todas as vendas por categoria e a soma total;
- Escrever `exibirMenu()` e gerenciar a interface no `main`, corrigindo problemas de buffer (ex.: `std::cin.ignore` em C++);
- Garantir saídas formatadas (ex.: valores com duas casas decimais, mensagens claras).

Tarefas específicas:

- Chamar `lerVendas` com filtro de categoria e exibir resultados com soma;
- Implementar relatório completo no `main`, usando `lerVendas` sem filtro;
- Corrigir `std::cin.ignore` no `main` (ex.: `std::cin.ignore(std::numeric_limits<std::streamsize '\n')` após leitura de opção);
- Testar com busca de categorias existentes/inexistentes e relatórios.

Interface:

- Usa dados de `lerVendas` para exibir resultados;
- Interage com o usuário, fornecendo categorias para `registrarVendas` e `buscarVendasCategoria`.

7.2.7.3 Considerações para Integração

Interfaces:

- Os alunos devem concordar com o formato do CSV ("Categoria,Valor") e a estrutura dos vetores (`vendas` como vetor de vetores de real, `categorias` como vetor de string);
- O número de categorias é rastreado por `numCategorias`.

Testes unitários:

- Cada aluno deve depurar sua parte isoladamente, simulando entradas/saídas (ex.: arquivos de teste, vetores pré-preenchidos).

Passos importantes da integração:

1. Copiar `main` e `exibirMenu` do Aluno 4;

2. Inserir `inicializarArquivo` e `salvarVendas` do Aluno 2;
3. Adicionar `lerVendas` do Aluno 3, ajustando o `main` para carregar dados iniciais;
4. Incluir `registrarVendas` do Aluno 1 e `buscarVendasCategoria` do Aluno 4.

Resolução de conflitos:

- Discutir problemas imediatamente na sala (ex.: formato do CSV, tratamento de buffer);
- Testar o programa completo com cenários como registrar vendas, buscar por categoria e exibir relatórios.

Teste conjunto:

- Verificar registro de vendas, busca por categoria, e relatório completo com soma total;
- Testar casos extremos (arquivo vazio, categoria não encontrada, múltiplas vendas);

7.2.7.4 Código C++23

O código correspondente ao problema N2 pode ser encontrado em https://onlinegdb.com/t5M4rGaOm3.

Listing 7.3

```cpp
#include <iostream>
#include <ranges>

// Função para valor absoluto sem usar cmath
auto abs_manual(auto valor) {
    return (valor < 0) ? -valor : valor;
}

int main() {
    // Valor de teste pré-definido:  /6 (30°)
    auto angulo = 0.523599;       //  /6 radianos
    auto senoReferencia = 0.5;    // sen( /6) = 0.5

    auto precisao = 0;

    std::cout << "Calculando sen( /6) = sen(30°) = sen(0.523599 rad)" << std::endl;
    std::cout << "Valor de referência: 0.5" << std::endl;
    std::cout << "Digite o número de termos da série: ";
    std::cin >> precisao;

    auto senoCalculado = 0.0;
    auto termo = angulo;

    for (auto i : std::views::iota(0, precisao)) {
        if (i % 2 == 0) {
            senoCalculado += termo;
            std::cout << "Termo " << i+1 << ": +" << termo
                 << " | Soma parcial: " << senoCalculado << std::endl;
        } else {
            senoCalculado -= termo;
            std::cout << "Termo " << i+1 << ": -" << termo
                 << " | Soma parcial: " << senoCalculado << std::endl;
        }

        // Próximo termo: x^(2n+3) / (2n+3)!
        termo = termo * angulo * angulo / ((2*i + 2) * (2*i + 3));
    }

    auto erroAbsoluto = abs_manual(senoCalculado - senoReferencia);

    std::cout << std::endl;
    std::cout << "Ângulo: " << angulo << " radianos ( /6 ou 30°)" << std::endl;
    std::cout << "Seno calculado: " << senoCalculado << std::endl;
    std::cout << "Seno de referência: " << senoReferencia << std::endl;
    std::cout << "Erro absoluto: " << erroAbsoluto << std::endl;

    return 0;
}
```

Listing 7.4

```cpp
#include <iostream>
#include <ranges>
#include <cmath>

int main() {
    auto numeroTermos = 0;
    auto somaLeibniz = 0.0;
    auto piCalculado = 0.0;
    auto precisaoAtingida = false;
    auto termosParaPrecisao = 0;

    auto piReferencia = 3.1415926535897932;

    std::cout << "Digite o número de termos: ";
    std::cin >> numeroTermos;

    for (auto i : std::views::iota(0, numeroTermos)) {
        if (i % 2 == 0) {
            somaLeibniz += 1.0 / (2*i + 1);
        } else {
            somaLeibniz -= 1.0 / (2*i + 1);
        }

        piCalculado = 4 * somaLeibniz;

        // Mostrar progresso a cada 1000 termos
        if ((i + 1) % 1000 == 0) {
            std::cout << "Termos: " << i+1 << " | Pi aproximado: "
                 << piCalculado << std::endl;
        }

        // Verificar precisão de 6 casas decimais
        if (std::abs(piCalculado - piReferencia) < 0.000001 and not precisaoAtingida) {
            termosParaPrecisao = i + 1;
            precisaoAtingida = true;
        }
    }

    std::cout << "Pi calculado: " << piCalculado << std::endl;
    std::cout << "Pi de referência: " << piReferencia << std::endl;
    std::cout << "Erro absoluto: " << std::abs(piCalculado - piReferencia) << std::endl;

    if (precisaoAtingida) {
        std::cout << "Precisão de 6 casas atingida com " << termosParaPrecisao << " termo
    } else {
        std::cout << "Precisão de 6 casas não atingida com " << numeroTermos << " termos"
    }

    return 0;
}
```

Listing 7.5

```
#include <iostream>
#include <ranges>

// Constantes matemáticas definidas manualmente
auto PI = 3.141592653589793;
auto E = 2.718281828459045;

// Função para valor absoluto sem usar cmath
auto abs_manual(auto valor) {
    return (valor < 0) ? -valor : valor;
}

// Função para raiz quadrada usando método babilônico
auto sqrt_babilonico(auto numero) {
    if (numero <= 0) return 0.0;

    auto x = numero;
    while (true) {
        auto raiz = 0.5 * (x + numero / x);
        if (abs_manual(raiz - x) < 0.000001) {
            return raiz;
        }
        x = raiz;
    }
}

auto calcularFatorial(auto n) {
    auto resultado = 1LL;
    for (auto i : std::views::iota(1, n+1)) {
        resultado *= i;
    }
    return resultado;
}

int main() {
    auto n = 0;
    std::cout << "Digite o valor de n (  20): ";
    std::cin >> n;

    if (n > 20) {
        std::cout << "Valor muito grande. Use n   20." << std::endl;
        return 1;
    }

    // Calcular fatorial exato
    auto fatorialExato = calcularFatorial(n);

    // Calcular produtório:  (i/e) de i=1 até n
    auto produtorio = 1.0;
    for (auto i : std::views::iota(1, n+1)) {
        produtorio *= (static_cast<double>(i) / E);
    }
    std::cout << "Produtório  (i/e): " << produtorio << std::endl;

    // Calcular somatório: Σ(ck/n^k) de k=0 até 2
```

8 Módulo 3: Semanas 9-13 (20 Horas-Aula): Algoritmização e Estruturas de Dados

O aluno começou esta disciplina com exercícios lúdicos para a criação de abstrações, passou por um conjunto de exercícios de decomposição, abstração gráfica, algoritmização e depuração. Depois enfrentou um período de aplicação destes conceitos, enquanto aprendia as estruturas sintáticas básicas para a criação de algoritmos na forma de programas. Agora, ele detém as ferramentas necessárias para a criação dos seus próprios algoritmos. No entanto, para a construção de qualquer algoritmo será necessário que ele, o aluno, solidifique os seus constructos cognitivos recém-adquiridos, para transformá-los em habilidades práticas. A habilidade que está sendo desenvolvida é a competência de resolver problemas com **Raciocínio Algorítmico**.

Nesta fase, o foco será no Design de Algoritmos, priorizando o desenvolvimento de procedimentos passo a passo, análise de eficiência e complexidade algorítmica. Este módulo inclui:

- Design de algoritmos: procedimentos passo a passo e análise de eficiência;
- Estruturas de dados: pilhas;
- Algoritmos fundamentais: busca linear e binária, ordenação;

Para atender este conjunto de conteúdos os exercícios estão divididos

1. **Algoritmos Comuns**: Estudo de algoritmos comuns: busca, ordenação e algoritmos básicos de grafos (`transversalidade`). Sem a matemática envolvida. Porém, forçando a análise dos custos computacionais, complexidade relacionada a estes algoritmos. Apenas no campo da abstração. A análise de algoritmos será feita através de exercícios práticos, na qual os alunos deverão implementar e analisar a eficiência dos algoritmos propostos.

2. **Estruturas de Dados**: Estudo da pilha. A pilha é uma estrutura de dados fundamental que permite o armazenamento e recuperação de dados de forma eficiente. Os alunos aprenderão a implementar pilhas, entender suas operações básicas (push, pop, peek) e analisar sua eficiência. A pilha será utilizada em exercícios práticos para resolver problemas comuns, como validação de expressões matemáticas e conversão de notação.

3. **Tarefas de Programação e Análise**: Tarefas de programação que exigem a implementação e análise de algoritmos. Serão propostos exercícios que requeiram interpretação para encontrar as soluções de decomposição, abstração e algoritmização.

Existe um conjunto de algoritmos e estruturas de dados comuns que são fundamentais para a programação. Estes artefatos são frequentemente utilizados e são essenciais para o desenvolvimento de soluções eficientes. Os artefatos, algoritmos e estruturas de dados, abordados neste módulo da disciplina de **Raciocínio Algorítmico** formam a base de muitos problemas da vida real que podem ser resolvidos com a aplicação das técnicas de computação. O objetivo desta seção não é explicar estes artefatos nem na forma matemática nem na forma algorítmica. O objetivo é provocar a redescoberta dessas soluções pelos alunos. Para atingir este objetivo estão sendo propostas cinco atividades de aplicação da metodologia **DAAD**que

levarão os alunos a deduzir, formalizar e analisar algoritmos comuns, como ordenação e busca e estruturas de dados como pilha.

Cada atividade foi concebida para ser realizada em um período de 90 minutos em um dia de aula com 4 horas consecutivas. Esta distribuição de tempo foi realizada a partir de algumas teorias neuro-cognitivas na esperança de criar uma rotina de consolidação de conhecimento. As atividades foram divididas em quatro etapas, cada uma com um objetivo específico, e foram projetadas para serem realizadas em grupos pequenos, promovendo a interação e a colaboração entre os alunos.

A avaliação será feita através de autoavaliação e avaliação entre pares, com foco na criação de algoritmos, depuração, análise de eficiência e complexidade algorítmica. A avaliação será baseada em indicadores de sucesso, como a criação correta de algoritmos, a depuração eficaz, a aplicação adequada a contextos novos e a profundidade das reflexões metacognitivas. Para tanto, foram criadas rubricas disponibilizadas online para auxiliar na avaliação dos alunos pelos próprios alunos.

Cabe ao professor estar preparado para reconhecer os padrões cognitivos que levaram a criação dos artefatos estudados neste módulo para poder reconhecer, e fomentar, estes padrões nos alunos. Além disso, o professor deve estar preparado para discutir as vantagens e desvantagens dos algoritmos e estruturas de dados estudados, bem como suas aplicações práticas.

8.1 Atividade A3: O Desafio da Ordenação

O objetivo desta atividade é levar os alunos a deduzir, formalizar e analisar algoritmos de ordenação através da experimentação prática, aplicando princípios da neurociência cognitiva a fim de maximizar a retenção e transferência do conhecimento. O foco desta disciplina está na aplicação das teorias da descoberta ativa, prática de recuperação estruturada e desenvolvimento metacognitivo (19).

Materiais: para cada grupo prepare os seguintes materiais:

- Um conjunto de 7-9 cartas numeradas;
- Envelope A: contendo cartas em ordem aleatória, para cada grupo (ex: [3, 1, 6, 2, 5, 4]);
- Envelope B: contendo cartas representando o “melhor caso”, para cada grupo (ex: [1, 2, 3, 5, 4, 6], quase ordenado);
- Envelope C: contendo cartas representando o “pior caso”, para cada grupo (ex: [8, 7, 6, 5, 4, 3], ordem inversa);
- Folhas para rascunho de pseudocódigo/fluxograma;
- Folhas para a Tabela de Rastreio para depuração;
- Canetas azuis, vermelhas e verdes para marcação de comparações e trocas, para cada grupo;
- Folha de Reflexão Metacognitiva.
- Quadro branco e marcadores coloridos, para o professor apresentar os conceitos e guiar a discussão.

Duração Total: 90 minutos (distribuição baseada na teoria do *spacing effect* para melhor consolidação)

8.1.1 Etapa 0: Ativação de Esquemas e *Retrieval Practice* (10 minutos)

Fundamentação: A ativação de conhecimento prévio facilita a criação de conexões neurais mais robustas (16).

Material: Quadro branco, marcadores coloridos, alunos divididos em grupos.

1. ***Warm-up - Retrieval Practice*** (5 min):
   - Instrução: "Sem consultar qualquer material, online ou impresso, listem em 2 minutos tudo que sabem sobre:
     - Como identificar eficiência de algoritmos;
     - Estratégias que usam para resolver problemas de ordenação no dia a dia.
   - Objetivo: ativar estruturas cognitivas relacionadas ao conhecimento necessário, antes da nova aprendizagem.
2. **Ativação de Esquemas** (5 min):
   - Instrução: Antes de abrir os envelopes, ordenem estes cenários do dia a dia:
     - Fila de banco por chegada
     - Produtos por preço
     - Emails por prioridade
   - Que estratégias mentais vocês usaram? Anotem 2-3 padrões comuns."
   - Objetivo: estabelecer conexões entre conhecimento cotidiano e algoritmos formais

Nota: Informe aos alunos que os envelopes, cartas, canetas, serão devolvidos no final da tarefa juntamente com os pseudocódigos e fluxogramas que eles criarem. Este material será usado na próxima aula.

8.1.2 Etapa 1: Descoberta Estruturada com Decomposição (22 minutos)

Fundamentação: A decomposição com o agrupamento de tarefas, reduz sobrecarga cognitiva permitindo um processamento mais profundo (16).

8.1.2.1 Sub-etapa 1a: Descoberta Pura (8 min)

Fundamentação: A descoberta ativa promove maior engajamento e retenção (19). **Material**: Envelope A (cartas aleatórias); Folhas em branco para Tabela de Rastreio, rascunho e observações; canetas coloridas (azuis, vermelhas e verdes).

1. Formação: Organizar turma em grupos de 4-5 alunos
2. Desafio Inicial:
   - Entregar apenas Envelope A (Aleatório);
   - Instrução: "Anotem a posição inicial das cartas no envelope";
   - Instrução: "Ordenem estas cartas e durante a ordenação criem um conjunto de regras que qualquer pessoa possa seguir";
   - Foco: Apenas na lógica, sem formalização.

8.1.2.2 Sub-etapa 1b: Formalização (7 min)

- Instrução: "Agora convertam sua estratégia para pseudocódigo ou fluxograma"
- Perguntas guia:
  - Qual parte foi mais difícil de entender?
  - Vocês perceberam algum padrão nas trocas?
  - Como verificar se o algoritmo está correto?

8.1.2.3 Sub-etapa 1c: Depuração com Tabela de Rastreio (8 min)

- Entregar Tabela de Rastreio;
- Instrução: “Executem o algoritmo passo a passo na tabela para provar que funciona”;
- Usar canetas de cores diferentes para comparações e trocas;
- Objetivo: Metacognição acerca do processo algorítmico.

8.1.2.4 Sub-etapa 1d: Consolidação - *Retrieval Practice* (2 min)

Fundamentação: A recuperação ativa fortalece a consolidação de memória (18).

- Instrução: “Fechem as anotações. Expliquem para o colega ao lado a lógica do algoritmo sem consultar anotações”.

8.1.3 Etapa 2: Predição e Teste de Eficiência (18 minutos)

Fundamentação: A predição ativa o processamento elaborativo, melhorando compreensão (153).

8.1.3.1 Sub-etapa 2a: *Elaborative Prediction* (5 min)

- Antes de entregar Envelopes 2 e 3:
- Instrução: “Baseado na lógica do algoritmo, expliquem, no papel, em linguagem natural, como ele se comportaria em três casos:
  - Melhor caso: _____ comparações, _____ trocas;
  - Pior caso: _____ comparações, _____ trocas;
  - Justifiquem as suas predições”;

8.1.3.2 Sub-etapa 2b: Teste e Coleta de Dados (13 min)

- Entregar Envelopes 2 e 3;
- Instrução: “Usem exatamente o mesmo algoritmo para os três casos. Contem comparações e trocas”;
- Comparar predições com resultados reais;
- Criar tabela de resultados, conforme o modelo abaixo:

Table 8.1: Tabela modelo para resultados de predições e comparações.

Caso	Predição Comp.	Real Comp.	Predição Trocas	Real Trocas
Aleatório	(valor)	(valor)	(valor)	(valor)
Melhor	(valor)	(valor)	(valor)	(valor)
Pior	(valor)	(valor)	(valor)	(valor)

8.1.4 Etapa 3: Apresentação a partir do Questionamento (20 minutos)

Fundamentação: Perguntas elaborativas aprofundam compreensão e criam conexões causais (154).

8.1.4.1 Sub-etapa 3a: Recuperação Antes da Apresentação (3 min)

- Instrução: “Antes de apresentar, reconstituam mentalmente todo o processo sem consultar anotações”

8.1.4.2 Sub-etapa 3b: Apresentação Estruturada (17 min)

Cada grupo apresenta:

1. Fluxograma/pseudocódigo no quadro;
2. Tabela de resultados (predições vs realidade);
3. Gráfico visual de comparações/trocas por caso.

Perguntas do Professor (Elaborative Interrogation):

- "Por que esse algoritmo precisou de mais trocas neste caso?"
- "Como isso se relaciona com complexidade algorítmica?"
- "Que aconteceria com 1000 elementos?"
- "Em que situação prática isso seria problemático?"

Conexões pelo Professor:

- Bubble Sort: "Essa estratégia de vizinhos é o Bubble Sort. Observem como o pior caso explodiu. Por quê?"
- Selection Sort: "O Selection Sort sempre tem poucas trocas. Quando isso é vantajoso?"
- Insertion Sort: "O Insertion Sort foi eficiente no melhor caso. Quando usar?"

Nota: existe a possibilidade de que os alunos apliquem algoritmos como o quick sort, merge sort ou outros algoritmos de ordenação mais avançados. O professor deve estar preparado para discutir as vantagens e desvantagens desses algoritmos em relação aos apresentados.

8.1.5 Etapa 4: *Transfer Learning* e Metacognição (12 minutos)

Fundamentação: A transferência para novos contextos consolida aprendizagem profunda (ROHRER; TAYLOR, 2007).

8.1.5.1 Sub-etapa 4a: Transfer Tasks (7 min)

Instrução: "Respondam, em papel, as perguntas a seguir referentes a aplicação dos algoritmos a situações reais:

1. Como ordenariam playlist de 10.000 músicas em smartphone com pouca memória?
2. Como ordenariam lista de emergências médicas por gravidade?
3. Que modificações fariam nos algoritmos para cada caso?"

8.1.5.2 Sub-etapa 4b: Consolidação e Conexões (3 min)

Instrução: "Discutam em grupo e anotem rapidamente:

1. Que situação do dia a dia de vocês se beneficiaria mais de algoritmos de ordenação eficientes?
2. Entre os algoritmos que descobriram hoje, qual escolheriam para organizar uma biblioteca com 50.000 livros? Por quê?
3. Identifiquem um exemplo no qual a simplicidade do algoritmo seria mais importante que a eficiência."

Objetivo: Consolidar conexões entre os algoritmos estudados e aplicações práticas, e reforçar a importância do contexto na escolha de soluções algorítmicas.

8.1.6 Etapa 5: Reflexão Metacognitiva (10 minutos)

Fundamentação: Reflexão metacognitiva melhora aprendizagem futura (17).

8.1.6.1 Sub-etapa 5a: Reflexão Individual (5 min)

Instrução: "Cada um de vocês deve escrever uma reflexão individual sobre o processo de aprendizagem de hoje, respondendo às seguintes perguntas:

1. "Que momento foi mais difícil no processo? Por quê?"
2. "Quando vocês 'clicaram' com a solução? O que mudou?"
3. "Se fossem ensinar para um calouro, que passo fariam diferente?"
4. "Que conexões veem entre esta atividade e problemas reais de programação?"

8.1.6.2 Sub-etapa 5b: Discussão Final Guiada (5 min)

O professor deve conduzir uma discussão final, na qual os alunos compartilham suas reflexões e insights. Perguntas adicionais podem incluir:

1. "Observando todos os dados, qual algoritmo é mais eficiente para lista quase ordenada?"
2. "Qual sofre mais com ordem inversa? Por que isso acontece neurologicamente?"
3. "Se troca fosse custosa mas comparação barata, qual seria melhor?"
4. "Para 1000 itens, como cresceria o número de operações? Linear ou quadrático?"
5. "Existe um algoritmo 'melhor' universal? Por que não?"
6. "Em fita magnética sequencial, qual seria mais adequado?"
7. "Em lista encadeada, qual ficaria mais difícil?"

8.1.7 Sessão de Follow-up - Spacing Effect (Aula seguinte - 15 min)

Fundamentação: O spacing effect demonstra que distribuição temporal consolida memória de longo prazo (CEPEDA et al., 2006).

8.1.7.1 Retrieval Practice Diferido:

1. **Reconstrução** (3 min): "Sem material, desenhem o algoritmo que criaram";
2. **Predição Escalada** (2 min): "Para 100 elementos invertidos, quantas comparações aproximadamente?";
3. **Aplicação Contextual** (5 min): "Identifiquem situação real ideal para cada algoritmo";
4. **Mini-quiz Adaptativo** (5 min): "Dados 3 arrays, escolham algoritmo mais eficiente";

8.1.7.2 Avaliação do Aprendizado

A avaliação será baseada em *peers review* e autoavaliação, com foco nos seguintes indicadores de sucesso:

- **Imediato**: Criação correta de algoritmo + depuração eficaz;
- **Transferência**: Aplicação adequada a contextos novos;
- **Retenção**: Performance no follow-up sem consulta;
- **Metacognição**: Reflexões demonstrando awareness do processo.

A avaliação deve ser realizada de acordo com as seguintes métricas de resultados:

- Precisão nas predições vs resultados reais
- Qualidade das justificativas causais

- Adequação das aplicações de transfer
- Profundidade das reflexões metacognitivas

Para que a avaliação seja eficaz, o professor deve garantir que os alunos tenham acesso a um feedback construtivo e que as reflexões sejam discutidas em grupo, promovendo um ambiente de aprendizagem colaborativa. A devolutiva deve ser feita pelo grupo que avaliou. Para tornar este processo de avaliação uniforme, existe uma ferramenta online disponível para auxiliar na avaliação em https://frankalcantara.com/raciocinio/avaliacao/modulo3-1.html

8.2 Atividade B3: Algoritmos de Ordenação

Na Atividade Section 8.1 os alunos descobriram os algoritmos de ordenação mais comuns. Se o resultado desta atividade foi satisfatório, ou seja, os alunos realmente descobriram os algoritmos, então esta atividade se resume à implementação dos algoritmos descobertos, a partir dos fluxogramas ou pseudocódigos criados pelos alunos.

Material Necessário:: computadores com ambiente de programação configurado, acesso à internet para pesquisa e os fluxogramas ou pseudocódigos criados na atividade anterior.

Objetivo: Implementar os algoritmos de ordenação descobertos na atividade anterior, analisando a eficiência e complexidade de cada um.

1. Distribua entre os grupos os fluxogramas, ou pseudocódigos, criados na atividade anterior. Assegure-se de que cada grupo tenha um fluxograma, ou pseudocódigo, diferente daquele criado na aula anterior;

2. Cada grupo irá implementar cada um dos algoritmos de ordenação criados na atividade anterior, ou seja, todos os grupos irão implementar todos os algoritmos da atividade anterior. A implementação deve ser feita em um único arquivo de código, com uma função para cada algoritmo de ordenação. A função deve receber como parâmetro uma lista de números e retornar a lista ordenada;

3. Após a implementação, cada grupo deve analisar a eficiência do algoritmo, medindo o tempo de execução e o número de comparações e trocas realizadas. Para isso, cada algoritmo deverá ordenar três listas diferentes de números aleatórios, uma lista quase ordenada e uma lista ordenada em ordem inversa. O resultado deve ser documentado em uma tabela e entregue ao professor.

Nota: como é a primeira vez que os alunos implementam algoritmos usando as funções de medida de tempo pode ser necessário que o professor explique como usar essas funções. Os professores que optarem pelo Python podem usar a função `time.perf_counter()`, a função `time.process_time()` ou o módulo `timeit` para medições rigorosas do tempo de execução de cada algoritmo. Um código em C++ 23 capaz de carregar os textos gerados no formato criado na página de avaliação disponível em https://frankalcantara.com/raciocinio/avaliacao/modulo3-2.html está apresentado pode ser visto na Listing 8.1.

8.2.1 Análise Esperada:

Com base nas atividades realizadas e nos resultados obtidos, espera-se que os alunos identifiquem e compreendam os seguintes pontos, consolidando seu aprendizado sobre a implementação e aplicação de estruturas de dados e algoritmos:

8.2.1.1 Sobre Algoritmos de Ordenação Implementados:

- **Bubble Sort O(n²)**: algoritmo mais simples, mas menos eficiente para listas grandes;
- **Insertion Sort O(n²)**: muito eficiente para listas pequenas ou quase ordenadas;
- **Selection Sort O(n²)**: comportamento consistente, mas não adaptativo;
- **Merge Sort O(n log n)**: mantém eficiência mesmo com listas grandes.

Nota: o uso da notação Big O é apenas informativo e serve como métrica de comparação.

8.2.1.2 Comparações de Performance:

- **Lista de números aleatória**: Merge Sort demonstra superioridade clara;
- **Lista de números quase ordenada**: Insertion Sort pode superar algoritmos mais complexos;
- **Lista de números em ordem inversa**: representa o pior caso para a maioria dos algoritmos;
- **Escalabilidade**: diferença dramática entre algoritmos de complexidade diferentes.

8.2.1.3 Sobre Implementação Prática:

- **Simplicidade vs Eficiência**: algoritmos simples nem sempre são os melhores;
- **Adaptabilidade**: alguns algoritmos se beneficiam de dados parcialmente ordenados;
- **Estabilidade**: preservação da ordem relativa de elementos iguais;
- **Uso de memória**: algoritmos in-place vs algoritmos que requerem memória adicional.

8.2.1.4 Insights sobre Medição de Performance:

- **Tempo de execução**: varia significativamente entre algoritmos e tipos de dados;
- **Número de comparações**: métrica importante para entender complexidade;
- **Número de trocas**: impacta performance quando movimentação de dados é custosa;
- **Comportamento assintótico**: como o desempenho muda com o tamanho dos dados.

8.2.1.5 Conexões com Conceitos Fundamentais:

- **Análise de complexidade**: importância da notação Big O na prática;
- **Trade-offs**: balanceamento entre simplicidade, eficiência e uso de recursos;
- **Escolha de algoritmos**: contexto determina qual algoritmo é mais apropriado;
- **Otimização prematura**: nem sempre o algoritmo mais eficiente é necessário.

8.2.2 Perguntas para Reflexão:

1. **Por que o Insertion Sort pode ser mais rápido que o Merge Sort em listas pequenas?**
2. **Em que situações você escolheria Bubble Sort mesmo sabendo que é menos eficiente?**
3. **Como o custo de movimentação de dados afeta a escolha entre diferentes algoritmos?**
4. **Qual algoritmo você usaria para ordenar uma lista de 10 milhões de elementos? Por quê?**
5. **Por que é importante medir tanto o tempo quanto o número de operações?**
6. **Como os resultados mudariam se os dados fossem strings em vez de números?**
7. **Que outros fatores além da eficiência deveriam influenciar a escolha de um algoritmo?**

8. **Como estes conceitos se aplicam a problemas de ordenação no mundo real?**

8.2.3 Extensões para Alunos Avançados:

- Implementar Quick Sort e comparar com os algoritmos básicos;
- Analisar o comportamento com diferentes tamanhos de dados (100, 1000, 10000 elementos);
- Implementar versões otimizadas dos algoritmos (Bubble Sort com flag, Insertion Sort binário);
- Criar visualizações gráficas do processo de ordenação;
- Comparar performance com funções de ordenação das bibliotecas padrão;
- Investigar algoritmos híbridos (Timsort, Introsort);
- Implementar algoritmos de ordenação externa para dados que não cabem na memória.

8.3 Atividade C3: Onde Está ...?

O objetivo desta atividade é levar os alunos a deduzirem, formalizarem e analisarem algoritmos de busca através da experimentação prática, aplicando princípios da neurociência cognitiva para melhorar a retenção e transferência do conhecimento. O foco está na aplicação das teorias da descoberta ativa, prática de recuperação estruturada e desenvolvimento metacognitivo ((16), (18), (19)).

Materiais: para cada grupo:

- Um conjunto de 15 cartas numeradas (1 a 15);
- Envelope A: contendo cartas em ordem aleatória (ex: [7, 2, 11, 4, 15, 1, 9, 13, 6, 3, 12, 8, 10, 5, 14]);
- Envelope B: contendo cartas em ordem crescente (ex: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]);
- Envelope C: contendo cartas em ordem decrescente (ex: [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]);
- Cartões de “Missão” com números específicos para buscar (ex: “Encontre o 8”, “Encontre o 3”, “Encontre o 12”);
- Folhas para rascunho de pseudocódigo/fluxograma;
- Folhas para Tabela de Rastreio para depuração;
- Canetas coloridas para marcar verificações e estratégias;
- Folha de Reflexão Metacognitiva.
- Quadro branco e marcadores coloridos para o professor apresentar conceitos e guiar discussão.

Duração Total: 90 minutos (distribuição baseada na teoria de *spacing effect* para melhor consolidação)

8.3.1 Etapa 0: Ativação de Esquemas e *Retrieval Practice* (10 minutos)

Fundamentação: A ativação de conhecimento prévio facilita a criação de conexões neurais mais robustas (16).

Material: Quadro branco, marcadores coloridos, alunos divididos em grupos.

1. ***Warm-up - Retrieval Practice*** (5 min):
   - Instrução: "Sem consultar qualquer material, listem em 2 minutos tudo que sabem sobre:
     - Como procurar algo em um conjunto de dados;
     - Estratégias que usam para encontrar informações no dia a dia.
   - Objetivo: ativar estruturas cognitivas relacionadas ao conhecimento necessário, antes da nova aprendizagem.
2. **Ativação de Esquemas** (5 min):
   - Instrução: "Antes de abrir os envelopes, descrevam como encontrariam:
     - Um nome específico em uma agenda telefônica;
     - Um livro específico em uma biblioteca;
     - Uma música específica em uma playlist;
   - Que estratégias mentais vocês usam? Anotem 2-3 padrões comuns."
   - Objetivo: estabelecer conexões entre conhecimento cotidiano e algoritmos formais

Nota: Informar aos alunos que os envelopes, cartas, canetas serão devolvidos no final da tarefa juntamente com os pseudocódigos e fluxogramas criados. Este material será usado na próxima aula.

8.3.2 Etapa 1: Descoberta Estruturada com Decomposição (22 minutos)

Fundamentação: A decomposição com agrupamento de tarefas reduz sobrecarga cognitiva permitindo processamento mais profundo (16).

8.3.2.1 Sub-etapa 1a: Descoberta Pura (6 min)

Fundamentação: A descoberta ativa promove maior engajamento e retenção (19). **Material**: Envelope A (cartas aleatórias); Cartão de Missão "Encontre o X" (cada grupo um número diferente); Folhas em branco para Tabela de Rastreio, rascunho e observações; canetas coloridas.

1. Formação: Organizar turma em grupos de 4-5 alunos
2. Desafio Inicial:
   - Entregar apenas Envelope A (Aleatório) e primeira missão;
   - Instrução: "Anotar a posição inicial das cartas no envelope";
   - Instrução: "Encontrem o número especificado no seu cartão de missão nestas cartas e criem um conjunto de regras que qualquer pessoa possa seguir para encontrar qualquer número solicitado";
   - Foco: Apenas na lógica, sem formalização.
   - **Importante**: Contar quantas cartas verificaram até encontrar o número indicado na sua missão.

8.3.2.2 Sub-etapa 1b: Formalização (6 min)

- Instrução: "Agora convertam sua estratégia para pseudocódigo ou fluxograma"
- Perguntas guia:
  - Qual foi o número máximo de verificações necessárias?
  - Existe uma forma de garantir que sempre encontrarão o número?
  - Como verificar se o algoritmo está correto?

8.3.2.3 Sub-etapa 1c: Depuração com Tabela de Rastreio (6 min)

- Instrução: "Executem o algoritmo passo a passo na tabela para encontrar o número especificado. Registrem cada verificação e troca de posição";
- Usar canetas de cores diferentes para marcar verificações;
- Objetivo: Metacognição acerca do processo algorítmico.

8.3.2.4 Sub-etapa 1d: Consolidação - *Retrieval Practice* (4 min)

Fundamentação: A recuperação ativa fortalece a consolidação de memória (18).

- Instrução: "Fechem as anotações. Expliquem para o colega ao lado a lógica do algoritmo sem consultar anotações".

8.3.3 Etapa 2: Predição e Teste de Eficiência (20 minutos)

Fundamentação: A predição ativa o processamento elaborativo, melhorando compreensão (153).

8.3.3.1 Sub-etapa 2a: *Elaborative Prediction* (6 min)

- Antes de entregar Envelopes B e C:
- Instrução: "Baseado na lógica do algoritmo, expliquem, no papel, em linguagem natural, como ele se comportaria em três casos diferentes:
  - Cartas aleatórias: _____ verificações máximas;
  - Cartas ordenadas crescente: _____ verificações máximas;
  - Cartas ordenadas decrescente: _____ verificações máximas;
  - Justifiquem as suas predições";

8.3.3.2 Sub-etapa 2b: Teste e Descoberta de Otimização (14 min)

- Entregar Envelopes B e C;
- Instrução: "Usem exatamente o mesmo algoritmo para os três casos. Busquem o número que seja o número especificado +1 em cada conjunto";
- Primeira rodada (4 min): Aplicar algoritmo original;
- Pausa para reflexão (3 min): "Observando as cartas ordenadas, existe uma forma mais eficiente?";
- Segunda rodada (4 min): Testar estratégia otimizada se descobriram;
- Comparar predições com resultados reais (3 min);
- Criar tabela de resultados:

Table 8.2: Modelo de tabela para registrar resultados.{#tbl-resultados2}

Caso	Algoritmo Original	Algoritmo Otimizado	Diferença
Aleatório	(valor)	(valor)	(valor)
Ordenado Cresc.	(valor)	(valor)	(valor)
Ordenado Decresc.	(valor)	(valor)	(valor)

Nota: a busca binária não é natural. Ela foi, provavelmente, um avanço evolutivo então, pode ser que que o professor tenha que induzir esta descoberta. Reforçando a sugestão: "Observando as cartas ordenadas, existe uma forma mais eficiente?"

8.3.4 Etapa 3: Apresentação a partir do Questionamento (18 minutos)

Fundamentação: Perguntas elaborativas aprofundam compreensão e criam conexões causais (154).

8.3.4.1 Sub-etapa 3a: Recuperação Antes da Apresentação (3 min)

- Instrução: "Antes de apresentar, reconstituam mentalmente todo o processo sem consultar anotações". Todavia, apenas os grupos que descobriram a busca binária, ou algum outro algoritmo otimizado, devem participar da apresentação.

8.3.4.2 Sub-etapa 3b: Apresentação Estruturada (15 min)

Cada grupo apresenta:

1. Fluxograma/pseudocódigo do algoritmo original no quadro;
2. Fluxograma/pseudocódigo do algoritmo otimizado (se descobriram);
3. Tabela de resultados (predições vs realidade);
4. Gráfico visual de verificações por caso.

Perguntas do Professor (Elaborative Interrogation):

- "Por que o algoritmo otimizado foi mais eficiente com cartas ordenadas?"
- "O que acontece se buscarmos o primeiro número da lista ordenada?"
- "E se buscarmos o último número?"
- "Como isso se relaciona com eficiência algorítmica?"
- "Que aconteceria com 1000 cartas?"
- "Em que situação prática cada algoritmo seria melhor?"

Conexões pelo Professor:

- Busca Linear: "Essa estratégia sequencial é a Busca Linear. Sempre funciona, mas qual é o custo?"
- Busca Binária: "A estratégia de dividir ao meio é a Busca Binária. Por que só funciona com dados ordenados?"

Além das perguntas, o professor pode destacar a complexidade dos algoritmos, talvez com uma introdução fluída e superficial à notação Big O, para que os alunos entendam que a eficiência não é apenas sobre o número de verificações, mas também sobre como o algoritmo escala com o tamanho dos dados.

8.3.5 Etapa 4: *Transfer Learning* e Metacognição (10 minutos)

Fundamentação: A transferência para novos contextos consolida a aprendizagem profunda (18).

8.3.5.1 Sub-etapa 4a: Transfer Tasks (7 min)

Instrução: "Respondam, em papel, as perguntas a seguir referentes à aplicação dos algoritmos a situações reais:

1. Como buscariam um contato específico em uma agenda telefônica de 10.000 nomes?
2. Como encontrariam uma palavra específica em um dicionário?
3. Como buscariam um produto específico em um catálogo online não-ordenado?
4. Que modificações fariam nos algoritmos para cada caso?"

8.3.5.2 Sub-etapa 4b: Consolidação e Conexões (3 min)

Instrução: "Discutam em grupo e anotem rapidamente:

1. Que situação do dia a dia de vocês se beneficiaria mais de algoritmos de busca eficientes?
2. Entre os algoritmos que descobriram hoje, qual escolheriam para encontrar um contato específico em uma agenda de 100.000 pessoas? Por quê?
3. Identifiquem um exemplo no qual a busca linear seria mais apropriada que a busca binária, mesmo com dados ordenados."

Objetivo: Consolidar conexões entre os algoritmos estudados e aplicações práticas, reforçando a importância da estrutura dos dados na escolha de estratégias de busca.

8.3.6 Etapa 5: Reflexão Metacognitiva (10 minutos)

Fundamentação: Reflexão metacognitiva melhora aprendizagem futura (17).

8.3.6.1 Sub-etapa 5a: Reflexão Individual (5 min)

Instrução: "Cada um deve escrever uma reflexão individual sobre o processo de descoberta de hoje, respondendo às seguintes perguntas:

1. "Que momento foi mais confuso no processo? Por quê?"
2. "Quando vocês 'clicaram' com a solução da pilha? O que mudou?"
3. "Como a pilha resolve o problema de precedência de operadores?"
4. "Que conexões veem entre esta atividade e problemas reais de programação?"

8.3.6.2 Sub-etapa 5b: Discussão Final Guiada (5 min)

O professor deve conduzir uma discussão final na qual os alunos compartilham suas reflexões e insights. Perguntas adicionais podem incluir:

1. "Por que a pilha funciona melhor que tentar resolver mentalmente?"
2. "Qual é a vantagem de eliminar parênteses?"
3. "Como isso se relaciona com o modo que computadores 'pensam'?"
4. "Em que outras situações uma estrutura LIFO seria útil?"
5. "Como vocês aplicariam este conceito em desenvolvimento de software?"
6. "Que outros problemas matemáticos poderiam se beneficiar desta abordagem?" ### Sessão de Follow-up - Spacing Effect (Aula seguinte - 12 min)

Fundamentação: O spacing effect demonstra que distribuição temporal consolida memória de longo prazo (154).

8.3.6.3 Retrieval Practice Diferido:

1. **Reconstrução** (3 min): "Sem material, desenhem os dois algoritmos que descobriram";
2. **Predição Escalada** (2 min): "Para 1000 elementos ordenados, quantas verificações máximas para cada algoritmo?";
3. **Aplicação Contextual** (4 min): "Identifiquem situação real ideal para cada algoritmo";
4. **Mini-quiz Adaptativo** (3 min): "Dados 3 cenários, escolham algoritmo mais eficiente";

8.3.7 Avaliação do Aprendizado

A avaliação será baseada em *peer review* e autoavaliação, com foco em:

8.3.7.1 Indicadores de Sucesso:

- **Imediato**: Criação correta de algoritmo + depuração eficaz
- **Transferência**: Aplicação adequada a contextos novos

- **Retenção**: Performance no follow-up sem consulta
- **Metacognição**: Reflexões demonstrando awareness do processo

8.3.7.2 Métricas Quantitativas:

- Precisão nas predições vs resultados reais
- Qualidade das justificativas causais
- Adequação das aplicações de transfer
- Profundidade das reflexões metacognitivas

Ferramenta online de avaliação disponível em: https://frankalcantara.com/raciocinio/avaliacao/modulo3-3.html

8.4 Atividade D3: Algoritmos de Busca

Na Atividade **C3** os alunos descobriram os algoritmos de busca mais comuns. Se o resultado foi satisfatório, ou seja, os alunos realmente descobriram os algoritmos, então esta atividade se resume à implementação dos algoritmos descobertos, a partir dos fluxogramas ou pseudocódigos criados pelos alunos.

Material Necessário: computadores com ambiente de programação configurado, acesso à internet para pesquisa e os fluxogramas ou pseudocódigos criados na atividade anterior.

Objetivo: Implementar os algoritmos de busca descobertos na atividade anterior, analisando a eficiência e complexidade de cada um.

8.4.1 Procedimento:

1. Distribua entre os grupos os fluxogramas ou pseudocódigos, criados na atividade anterior.

2. Certifique-se de que cada grupo tenha um fluxograma ou pseudocódigo, diferente daquele criado na aula anterior;

3. Cada grupo irá implementar cada um dos algoritmos de busca criados na atividade anterior, ou seja, todos os grupos irão implementar todos os algoritmos da atividade anterior. A implementação deve ser feita em uma única folha de código, com uma

função para cada algoritmo de busca. A função deve receber como parâmetros uma lista de números e um valor a ser buscado, retornando a posição do elemento (ou -1 se não encontrado);

4. Após a implementação, cada grupo deve analisar a eficiência do algoritmo, medindo o tempo de execução e o número de comparações realizadas. Para isso, cada algoritmo deverá buscar elementos específicos em três listas diferentes: uma lista de números aleatórios, uma lista ordenada crescente e uma lista ordenada decrescente. O resultado deve ser documentado em uma tabela e entregue ao professor.

Nota: Como é a primeira vez que os alunos implementam algoritmos de busca com funções de medida de tempo, pode ser necessário que o professor explique como usar essas funções. O professor pode usar a função `time.perf_counter()`, a função `time.process_time()` ou o módulo `timeit` para medições rigorosas do tempo de execução de cada algoritmo.

8.4.2 Exemplo de código base para medição:

Os alunos podem usar este código base em C++ 23 para carregar dados e medir performance dos algoritmos de busca. O código é compatível com arquivos CSV gerados pela ferramenta disponível em https://frankalcantara.com/raciocinio/avaliacao/modulo3-4.html

Um exemplo de código em C++ 23 para medir o tempo de execução dos algoritmos de busca que os alunos irão implementar pode ser visto na Listing 8.2.

8.4.3 Análise Esperada:

Com base nas atividades realizadas e nos resultados obtidos, espera-se que os alunos identifiquem e compreendam os seguintes pontos, consolidando seu aprendizado sobre a implementação e aplicação de estruturas de dados e algoritmos:

8.4.3.1 Sobre Busca Linear O(n):

- **Funciona sempre**: em qualquer lista, ordenada ou não
- **Comportamento previsível**: verifica elementos sequencialmente
- **Pior caso**: precisa verificar todos os elementos
- **Melhor caso**: encontra o elemento na primeira posição
- **Caso médio**: encontra o elemento na metade da lista

8.4.3.2 Sobre Busca Binária O(log n):

- **Requer ordenação**: só funciona em listas ordenadas
- **Muito mais eficiente**: para listas grandes, diferença é dramática
- **Estratégia "dividir para conquistar"**: elimina metade dos elementos a cada comparação
- **Crescimento logarítmico**: para 1000 elementos, máximo 10 comparações

8.4.3.3 Comparações Práticas:

- Para **1000 elementos**: Linear = até 1000 comparações, Binária = até 10 comparações
- Para **1.000.000 elementos**: Linear = até 1.000.000 comparações, Binária = até 20 comparações
- **Trade-off importante**: custo de ordenação vs benefício da busca binária

8.4.3.4 Insights Pedagógicos:

- A importância da **estrutura dos dados** na escolha do algoritmo
- **Quando usar cada algoritmo**:
  - Busca linear: dados não ordenados, poucas buscas
  - Busca binária: dados ordenados, muitas buscas
- A diferença entre **complexidade teórica** e **performance prática**
- Como o **tamanho dos dados** afeta dramaticamente a performance

8.4.3.5 Conexões com o Mundo Real:

- **Agenda telefônica**: exemplo clássico de busca binária
- **Bases de dados**: índices funcionam como busca binária
- **Algoritmos de busca web**: combinam múltiplas estratégias
- **Jogos de adivinhação**: "maior/menor" é busca binária

8.4.4 Perguntas para Reflexão:

1. **Por que a busca binária só funciona em listas ordenadas?**
2. **Em que situações você usaria busca linear mesmo tendo dados ordenados?**
3. **Como o custo de ordenação afeta a escolha do algoritmo?**
4. **Que outros algoritmos do dia a dia seguem a estratégia "dividir para conquistar"?**

8.4.5 Extensões para Alunos Avançados:

- **Busca Ternária**: dividir array em três partes em vez de duas, analisar se melhora performance teórica e prática comparada à busca binária
- **Busca com Múltiplos Critérios**: implementar busca por ranges (encontrar todos elementos entre dois valores), busca aproximada (elemento mais próximo)
- **Benchmark Extensivo**: criar suite de testes com diferentes tamanhos (1K a 10M elementos), diferentes distribuições de dados (gaussiana, exponencial, uniforme)
- **Implementação de Índices**: criar estruturas de índice simples (como B-trees básicas) e comparar com busca direta

8.5 Atividade E3: Criando os Enemigos da HP

O objetivo desta atividade é levar os alunos a deduzir, formalizar e analisar o uso de **pilha (*stack*)** para avaliação de expressões matemáticas através da experimentação prática com notação polonesa reversa, aplicando princípios da neurociência cognitiva para maximizar a retenção e transferência do conhecimento. O foco está na aplicação das teorias da descoberta ativa, prática de recuperação estruturada e desenvolvimento metacognitivo ((16), (18), (19)).

Materiais: para cada grupo de alunos:

- **Cartas de números** (0-9, múltiplas cópias de cada);
- **Cartas de operadores** (+, -, ×, ÷);
- **Pilha física** (pequena caixa ou área delimitada na mesa);
- **Expressões matemáticas** em cartões (3 níveis de dificuldade);
- **Calculadora tradicional** para verificação;

- **Folhas para anotações** e cálculos intermediários;
- **Folha para a Tabela de rastreio** para depuração do processo;
- **Cronômetro** para medir tempo de resolução;
- **Folha para reflexão metacognitiva**.
- **Quadro branco** e **marcadores coloridos** para o professor apresentar conceitos e guiar discussão.

Duração Total: 90 minutos (distribuição baseada na teoria de *spacing effect* para melhor consolidação)

8.5.1 Etapa 0: Ativação de Esquemas e *Retrieval Practice* (10 minutos)

Fundamentação: A ativação do conhecimento prévio facilita a criação de conexões neurais mais robustas (16).

Material: Quadro branco, marcadores coloridos, alunos divididos em grupos. Arranjar mesas para induzir interação em grupos pequenos.

1. ***Warm-up - Retrieval Practice*** (5 min):
   - Instrução: "Sem consultar qualquer material, listem em 2 minutos tudo que sabem sobre:
     - Como resolver expressões matemáticas complexas;
     - Problemas que vocês tiveram com precedência de operadores;
     - Estratégias que usaram para não se confundir com parênteses."
   - Objetivo: ativar estruturas cognitivas relacionadas a cálculos e ordem de operações.
2. **Ativação de Esquemas** (5 min):
   - Instrução: "Tentem resolver mentalmente, sem qualquer anotação, as seguintes expressões:
     - $3 + 4 \times 2$;
     - $2 \times 3 + 4 \times 5$;
     - $((3 + 4) \times 2) + 1$
   - Discutam em grupo: os pontos nos quais sentiram dificuldade? Que estratégias usaram?";
   - Objetivo: estabelecer uma espécie frustração produtiva (16) com expressões complexas.

Nota: Informar aos alunos que as cartas, pilha e anotações serão reutilizadas ao longo da atividade para comparação de métodos.

8.5.2 Etapa 1: Descoberta Estruturada com Decomposição (22 minutos)

Fundamentação: A descoberta de limitações motiva a busca por soluções mais eficientes (16).

8.5.2.1 Sub-etapa 1a: Desafio das Expressões Complexas (10 min)

Fundamentação: Expor os alunos a problemas genuinamente difíceis cria necessidade cognitiva para ferramentas melhores (19). **Material**: Cartões com expressões matemáticas complexas; papel para cálculos; cronômetro.

1. Formação: Organizar turma em grupos de 4-5 alunos
2. Desafio Inicial:
    - Entregar cartões com expressões como:
        - $1535 + \times$;
        - $23 + 45 + \times$;
        - $82 \div 3 + 1-$;
    - **Importante**: o professor deve evitar explicar que se trata da notação polonesa reversa;
    - Instrução: "Estas são expressões matemáticas escritas de forma especial. Tentem descobrir como calcular o resultado.";
    - Cronometragem: é importante registrar quanto tempo os alunos gastam tentando descobrir o padrão e resolver as expressões.

8.5.2.2 Sub-etapa 1b: Reflexão sobre Dificuldades (6 min)

- Instrução: "Anotem as dificuldades que encontraram:";
- Perguntas guia:
    - Por que essas expressões são confusas?
    - Como vocês normalmente resolvem cálculos?
    - O que falta nessas expressões para ficarem claras?
    - Existe algum padrão que vocês conseguem identificar?

8.5.2.3 Sub-etapa 1c: Tentativa de Solução Colaborativa (6 min)

- Instrução: "Trabalhem em grupo para tentar resolver pelo menos uma expressão"
- Observar estratégias emergentes
- **Não dar dicas ainda** - deixar a descoberta acontecer naturalmente
- Objetivo: Documentar as abordagens intuitivas antes da intervenção

8.5.3 Etapa 2: Introdução da Ferramenta Pilha (20 minutos)

Fundamentação: Introduzir a solução após estabelecer a necessidade maximiza o impacto da descoberta (153).

8.5.3.1 Sub-etapa 2a: Demonstração Guiada (10 min)

Material: Pilha física, cartas de números e operadores.

- **Regras da Pilha Mágica:**
    1. **Número**: sempre colocar na pilha
    2. **Operador**: tirar dois números da pilha, fazer a operação, colocar resultado de volta
    3. **Resultado final**: único número que sobra na pilha
- **Demonstração com 15 3 5 + ×:**
    - **15** → pilha: [15]

- **3** → pilha: [15, 3]

- **5** → pilha: [15, 3, 5]
- **+** → tira 5 e 3, calcula 3+5=8, coloca 8 → pilha: [15, 8]
- **×** → tira 8 e 15, calcula 15×8=120, coloca 120 → pilha: [120]
- **Resultado: 120**

8.5.3.2 Sub-etapa 2b: Prática Guiada (5 min)

- Instrução: "Agora tentem resolver usando a pilha as outras expressões do desafio inicial"
- **2 3 + 4 5 + ×** (esperado: 35)
- **8 2 ÷ 3 + 1 -** (esperado: 6)
- Acompanhar grupo por grupo, corrigir uso da pilha
- Usar cartas físicas para manipulação tátil

8.5.3.3 Sub-etapa 2c: Comparação de Eficiência (5 min)

- Instrução: "Comparem o tempo que levaram agora versus na primeira tentativa"
- Criar tabela:

Table 8.3: Modelo de tabela para comparação de eficiência.{#tabela-eficiencia3}

Expressão	Tempo Sem Pilha	Tempo Com Pilha	Diferença
15 3 5 + ×	(valor) min	(valor) min	(valor) min
2 3 + 4 5 + ×	(valor) min	(valor) min	(valor) min

8.5.4 Etapa 3: Formalização e Algoritmo (18 minutos)

Fundamentação: A formalização após descoberta prática consolida o aprendizado (154).

8.5.4.1 Sub-etapa 3a: Criação do Algoritmo (10 min)

- Instrução: "Criem um algoritmo passo-a-passo para qualquer pessoa conseguir usar a 'pilha mágica' "

Os alunos devem criar um algoritmo na forma de fluxograma, ou pseudocódigo, que descreva o processo de avaliação de expressões em notação polonesa reversa usando pilha.

Nota: talvez seja necessário que o professor chame a atenção para as tarefas mínimas que devem ser incluídas no algoritmo, como "inicializar pilha", "empilhar número", "desempilhar dois números", "aplicar operador" e "empilhar resultado". Apenas para complementar a estrutura que os alunos criaram.

8.5.4.2 Sub-etapa 3b: Teste com Expressões Mais Complexas (8 min)

- **Novos desafios:**

  - $12 + 34 + \times$ (esperado: 21);
  - $62 \div 3 \times 4+$ (esperado: 13);
  - $512 + 4 \times +3-$ (esperado: 14);

- Usar tabela de rastreio para depuração:

Table 8.4: Modelo de tabela de rastreio para depuração.{#tabela-rastreio3}

Passo	Símbolo	Ação	Estado da Pilha
1	5	empilhar	[5]
2	1	empilhar	[5, 1]
3	2	empilhar	[5, 1, 2]
4	+	calcular 1+2=3	[5, 3]
...	...	...	...

8.5.5 Etapa 4: Transfer Learning e Conexões (10 minutos)

Fundamentação: A transferência para novos contextos consolida a aprendizagem profunda (18).

8.5.5.1 Sub-etapa 4a: Conversão de Notações (5 min)

Instrução: "Convertam estas expressões tradicionais para notação polonesa reversa e resolvam:"

1. $\mathbf{3 + 4 \times 2} \rightarrow \mathbf{3\ 4\ 2 \times +}$ (esperado: 11)
2. $\mathbf{(2 + 3) \times 4} \rightarrow \mathbf{2\ 3 + 4 \times}$ (esperado: 20)

3. $\mathbf{2 \times (3 + 4) + 1} \rightarrow \mathbf{2\ 3\ 4 + \times 1 +}$ (esperado: 15)

Insight esperado: A notação polonesa reversa **elimina a necessidade de parênteses** e **ambiguidade de precedência**.

8.5.5.2 Sub-etapa 4b: Conexões com Mundo Real (5 min)

Instrução: "Onde mais vocês acham que essa estratégia é usada?"
Discussão guiada:

- **Calculadoras HP**: ainda usam RPN hoje;
- **Compiladores**: convertem expressões para pilha antes de avaliar;
- **Máquinas virtuais**: Java, Python usam pilha para operações;
- **Undo/Redo**: mesma estrutura LIFO para desfazer ações.

8.5.6 Etapa 5: Reflexão Metacognitiva (10 minutos)

Fundamentação: Reflexão metacognitiva melhora aprendizagem futura (17).

8.5.6.1 Sub-etapa 5a: Reflexão Individual (5 min)

Instrução: "Cada um deve escrever uma reflexão individual sobre o processo de descoberta de hoje:"

1. "Que momento foi mais confuso no processo? Por quê?"
2. "Quando vocês 'clicaram' com a solução da pilha? O que mudou?"
3. "Como a pilha resolve o problema de precedência de operadores?"
4. "Que conexões veem entre esta atividade e programação de computadores?"

8.5.6.2 Sub-etapa 5b: Discussão Final Guiada (5 min)

O professor deve conduzir discussão na qual alunos compartilham reflexões:

1. "Por que a pilha funciona melhor que tentar resolver mentalmente?"
2. "Qual é a vantagem de eliminar parênteses?"
3. "Como isso se relaciona com o modo que computadores 'pensam'?"
4. "Em que outras situações uma estrutura LIFO seria útil?"

8.5.7 Sessão de Follow-up - Spacing Effect (Aula seguinte - 12 min)

Fundamentação: O spacing effect demonstra que distribuição temporal consolida memória de longo prazo (154).

8.5.7.1 Retrieval Practice Diferido:

1. **Reconstrução** (3 min): "Sem material, expliquem o algoritmo da pilha para RPN"
2. **Aplicação Prática** (4 min): "Resolvam **4 2 + 3 × 1 -** usando apenas papel"
3. **Conversão** (3 min): "Convertam **(2+3)×4-1** para RPN e resolvam"
4. **Conexão** (2 min): "Identifiquem 2 situações do dia a dia que usam LIFO"

8.5.8 Avaliação do Aprendizado

A avaliação será baseada em *peer review* e autoavaliação, com foco nos seguintes indicadores de sucesso:

- **Imediato**: Uso correto da pilha para avaliar expressões RPN
- **Transferência**: Conversão correta de notação infixa para RPN

- **Retenção**: Performance no follow-up sem consulta
- **Metacognição**: Reflexões demonstrando compreensão do conceito LIFO

Estes indicadores devem ser avaliados segundo as seguintes métricas:

- Tempo de resolução (antes vs depois da pilha)
- Precisão na avaliação de expressões
- Qualidade das conversões infixa→RPN Profundidade das reflexões metacognitivas

Ferramenta online de avaliação disponível em: https://frankalcantara.com/raciocinio/avaliacao/modulo3-5.html

8.5.9 Análise Esperada:

Com base nas atividades realizadas e nos resultados obtidos, espera-se que os alunos identifiquem e compreendam os seguintes pontos, consolidando seu aprendizado sobre a implementação e aplicação de estruturas de dados e algoritmos:

8.5.9.1 Sobre Pilha para Avaliação de Expressões:

- **Elimina ambiguidade**: não precisa de parênteses nem regras de precedência
- **Processo sistemático**: sempre funciona, sem exceções
- **Eficiência cognitiva**: reduz carga mental para expressões complexas
- **Ordem natural**: LIFO espelha como operações são aninhadas

8.5.9.2 Sobre Notação Polonesa Reversa (RPN):

- **Sem parênteses**: estrutura inerente da pilha resolve precedência
- **Avaliação linear**: processa da esquerda para direita, sem backtracking
- **Menos erros**: impossível criar expressões ambíguas
- **Computacionalmente eficiente**: compiladores usam este método

8.5.9.3 Conexões Fundamentais:

- **Estrutura de dados apropriada** resolve problemas naturalmente
- **LIFO** aparece em múltiplos contextos (undo, call stack, parsing)
- **Abstrações matemáticas** têm implementações práticas em computação
- **Ferramentas simples** (pilha) resolvem problemas complexos (precedência)

8.5.9.4 Insights Pedagógicos:

- A **frustração inicial** motiva busca por soluções melhores
- **Manipulação física** (cartas) reforça conceitos abstratos
- **Descoberta guiada** é mais eficaz que apresentação direta
- **Aplicação imediata** consolida aprendizado teórico

8.5.10 Extensões Possíveis:

Para alunos avançados, o professor pode propor:

- Implementar calculadora RPN em código;
- Converter expressões com múltiplos níveis de parênteses;
- Explorar notação polonesa normal (operador antes dos operandos);
- Analisar como compiladores fazem parsing de expressões;
- Criar visualizações do processo de avaliação.

8.6 Atividade F3: Implementação de Pilha para Avaliação de Expressões

Na Atividade E3 os alunos descobriram como usar a estrutura de dados pilha para avaliar expressões em notação polonesa reversa e possivelmente como converter expressões da notação infixa para a notação polonesa reversa. Se o resultado desta atividade foi satisfatório, ou seja, os alunos realmente compreenderam o conceito de pilha e sua aplicação, então a atividade **F3** se resume à implementação dos algoritmos descobertos, a partir dos fluxogramas ou pseudocódigos criados pelos alunos.

Material Necessário: computadores com ambiente de programação configurado, acesso à internet para pesquisa e os fluxogramas ou pseudocódigos criados na atividade anterior.

Objetivo: Implementar a estrutura de dados pilha e os algoritmos para avaliação de expressões em notação polonesa reversa descobertos na atividade anterior, analisando a eficiência e precisão de cada implementação.

8.6.1 Procedimento:

1. Distribua entre os grupos os fluxogramas, ou pseudocódigos, criados na atividade anterior. Certifique-se de que cada grupo tenha um fluxograma, ou pseudocódigo, diferente daquele criado por este mesmo grupo na aula anterior;

2. Cada grupo irá implementar cada um dos algoritmos criados na atividade anterior, ou seja, todos os grupos irão implementar todos os algoritmos da atividade anterior. A implementação deve ser feita em uma única folha de código, com as seguintes funções:

    - Uma estrutura para implementar a pilha;
    - Uma função para avaliar expressões em notação polonesa reversa;
    - Uma função para converter expressões da notação infixa para RPN (se foi descoberta);
    - Função principal para testar e medir performance;

3. Após a implementação, cada grupo deve analisar a eficiência dos algoritmos, medindo o tempo de execução e a precisão dos resultados. Para isso, cada algoritmo deverá processar três conjuntos diferentes de expressões: expressões simples, expressões complexas e expressões com múltiplos níveis de operações. O resultado deve ser documentado em uma tabela e entregue ao professor.

Nota: Como é a primeira vez que os alunos implementam estruturas de dados customizadas, pode ser necessário que o professor reveja, ou explique os conceitos básicos de arrays, vetores, listas e registros. O professor pode demonstrar como usar vetores (`vector`) em C++ ou listas em Python para implementar a pilha.

8.6.2 Exemplo de código base para medição:

Os alunos podem usar este código base em C++ 23 para implementar e testar sua estrutura de pilha.

Um exemplo de código em C++ 23 para implementar e medir o tempo de execução dos algoritmos de pilha que os alunos irão desenvolver pode ser visto no código Listing 8.3. Este código inclui a estrutura de pilha, funções básicas para manipulação da pilha, tokenização de expressões, avaliação de expressões em notação polonesa reversa e medição de tempo de execução. Contudo,, neste código foi utilizada uma pilha simples, para armazenar *strings*, na tentativa de criar um código parecido com os pseudocódigos que são criados por alunos que ainda não são proficients em estruturas de dados.

8.6.3 Análise Esperada:

Com base nas atividades realizadas e nos resultados obtidos, espera-se que os alunos identifiquem e compreendam os seguintes pontos, consolidando seu aprendizado sobre a implementação e aplicação de estruturas de dados e algoritmos:

8.6.3.1 Sobre a Estrutura de Dados Pilha:

- **Operações básicas**: empilhar, desempilhar, topo, vazia são todas $O(1)$;
- **Comportamento LIFO**: *Last In, First Out* - fundamental para muitas aplicações;
- **Simplicidade**: implementação direta usando array dinâmico (`vector`);
- **Robustez**: verificações de erro evitam operações inválidas.

8.6.3.2 Sobre Avaliação de Expressões RPN:

- **Algoritmo determinístico**: sempre funciona da mesma forma;
- **Sem ambiguidade**: não precisa de regras de precedência;
- **Eficiência**: processamento linear, $O(n)$ no qual n é o número de tokens;
- **Facilidade de implementação**: lógica clara e direta.

8.6.3.3 Sobre Conversão Infixa para RPN:

- **Complexidade adicional**: mais difícil que avaliação RPN;
- **Gerenciamento de parênteses**: requer cuidado especial;
- **Verificação de validade**: muitas oportunidades para erro de sintaxe.

8.6.3.4 Comparações de Performance:

- **Expressões simples**: tempo quase instantâneo, principalmente sobrecarga de função;
- **Expressões complexas**: crescimento linear com número de operações;
- **Conversão infixa**: tempo adicional para parsing, mas resultado equivalente;
- **Operações de pilha**: métrica interessante para complexidade algorítmica.

8.6.3.5 Insights Pedagógicos:

- **Estrutura adequada**: pilha resolve naturalmente o problema de avaliação;
- **Abstração poderosa**: oculta complexidade da precedência de operadores;
- **Implementação incremental**: construir do simples para o complexo;
- **Casos de erro**: importante considerar entradas inválidas.

8.6.4 Perguntas para Reflexão:

1. **Por que a pilha é a estrutura ideal para avaliação de expressões RPN?**
2. **Como o algoritmo de conversão infixa→RPN gerencia precedência de operadores?**
3. **Que aconteceria se usássemos uma fila em vez de pilha para RPN?**
4. **Como esta implementação se relaciona com compiladores reais?**
5. **Que otimizações poderiam ser feitas para expressões muito grandes?**

8.6.5 Extensões para Alunos Avançados:

- Implementar suporte a funções matemáticas (sin, cos, log);
- Adicionar variáveis e constantes às expressões;
- Criar interface gráfica para visualizar o funcionamento da pilha;
- Implementar otimizações para expressões com muitas constantes;
- Comparar performance com biblioteca padrão (std::stack).

Listing 8.1

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>
#include <chrono>
#include <iomanip>
#include <algorithm>
#include <limits>

using namespace std;
using namespace std::chrono;

// Estrutura para armazenar os dados do CSV
struct DataSet {
    vector<int> lista_aleatoria;
    vector<int> lista_80_ordenada;
    vector<int> lista_decrescente;
};

// Estrutura para armazenar resultados de tempo
struct ResultadoTempo {
    double tempo_aleatorio;
    double tempo_80_ordenado;
    double tempo_decrescente;
};

// ==================== PROTÓTIPOS DAS FUNÇÕES ====================
DataSet lerCSV(const string& nomeArquivo);
void bubbleSort(vector<int>& arr);
void insertionSort(vector<int>& arr);
void selectionSort(vector<int>& arr);
void mergeSort(vector<int>& arr);
void mergeSortRecursivo(vector<int>& arr, int esquerda, int direita);
void merge(vector<int>& arr, int esquerda, int meio, int direita);
template<typename Func> double medirTempo(vector<int> dados, Func algoritmo);
bool estaOrdenado(const vector<int>& arr);
void testarAlgoritmos(const DataSet& dados);
void exibirTabelaResultados(const vector<pair<string, ResultadoTempo>>& resultados);
void analisarResultados(const vector<pair<string, ResultadoTempo>>& resultados);

// Função para ler o arquivo CSV
DataSet lerCSV(const string& nomeArquivo) {
    DataSet dados;
    ifstream arquivo(nomeArquivo);
    string linha;

    if (!arquivo.is_open()) {
        cerr << "Erro: Não foi possível abrir o arquivo " << nomeArquivo << std::endl;
        return dados;
    }

    // Pular o cabeçalho
    getline(arquivo, linha);
```

Listing 8.2

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>
#include <chrono>
#include <iomanip>
#include <algorithm>
#include <limits>
#include <set>

using namespace std;
using namespace std::chrono;

// Estrutura para armazenar os dados do CSV
struct DataSet {
    vector<int> lista_aleatoria;
    vector<int> lista_ordenada_cresc;
    vector<int> lista_ordenada_decresc;
    vector<int> elementos_busca;
};

// Estrutura para armazenar resultados de busca
struct ResultadoBusca {
    bool encontrado;
    int posicao;
    int comparacoes;
    double tempo_ms;
    int elemento_buscado;
};

// Estrutura para armazenar estatísticas de um algoritmo
struct EstatisticasAlgoritmo {
    string nome;
    int total_comparacoes;
    double tempo_total_ms;
    int elementos_encontrados;
    int elementos_nao_encontrados;
    double tempo_medio_ms;
    double comparacoes_medias;
};

// ==================== PROTÓTIPOS DAS FUNÇÕES ====================
DataSet lerCSV(const string& nomeArquivo);
vector<int> extrairElementosBusca(const string& nomeArquivo);
ResultadoBusca buscaLinear(const vector<int>& arr, int alvo);
ResultadoBusca buscaBinaria(const vector<int>& arr, int alvo);
void testarAlgoritmos(const DataSet& dados);
void exibirResultadosDetalhados(const vector<ResultadoBusca>& resultados,
                                const string& nomeAlgoritmo,
                                const string& tipoLista);
void exibirEstatisticasComparativas(const vector<EstatisticasAlgoritmo>& stats);
EstatisticasAlgoritmo calcularEstatisticas(const vector<ResultadoBusca>& resultados,
                                           const string& nome);
void demonstrarComplexidade(size_t tamanho);
```

Listing 8.3

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>
#include <chrono>
#include <iomanip>
#include <cctype>
#include <cmath>

// ==================== ESTRUTURA SIMPLES DA PILHA ====================

struct Pilha {
    std::vector<std::string> elementos;
    int topo;

    Pilha() : topo(-1) {}
};

// Estruturas para dados de teste
struct ConjuntoExpressoes {
    std::vector<std::string> expressoes_simples;
    std::vector<std::string> expressoes_complexas;
    std::vector<std::string> expressoes_infixas;
    std::vector<double> resultados_esperados_simples;
    std::vector<double> resultados_esperados_complexas;
    std::vector<double> resultados_esperados_infixas;
};

struct ResultadoTeste {
    double resultado_calculado;
    double resultado_esperado;
    double tempo_ms;
    bool sucesso;
    std::string erro;
    int operacoes_realizadas;
};

struct EstatisticasTeste {
    std::string nome;
    int sucessos;
    int erros;
    double tempo_medio_ms;
    double operacoes_medias;
    double precisao_percentual;
};

// ==================== FUNÇÕES BÁSICAS DA PILHA ====================

void empilhar(Pilha& pilha, const std::string& elemento) {
    pilha.elementos.push_back(elemento);
    pilha.topo++;
}

std::string desempilhar(Pilha& pilha) {
```

9 Módulo 4: Semanas 14-16 (12 Horas-Aula): Depuração e Aplicação em Projetos

Este módulo final consolida o ciclo da metodologia **DAAD** ao focar na **Depuração Sistemática**. Os alunos farão a transição da verificação de programas com `std::cout` para a aplicação de técnicas formais de identificação de erros, rastreamento de execução e teste e versionamento de software. O objetivo é equipá-los com as ferramentas e os processos necessários para validar a funcionalidade de seus algoritmos e colaborar em projetos de maior escala.

A aplicação dos conceitos será realizada através do desenvolvimento de projetos que simulam desafios do mundo real, exigindo a integração de todas as habilidades desenvolvidas ao longo da disciplina.

- **Depuração Sistemática**: abordagem de técnicas para identificação de erros e métodos de rastreamento de execução de programas;
- **Princípios de Teste de Software**: Introdução aos conceitos de teste de unidade e teste de integração, com atividades práticas de depuração;
- **Sistemas de Controle de Versão**: Apresentação de ferramentas como **Git** e plataformas como **GitHub** para o desenvolvimento colaborativo;
- **Metodologias de Desenvolvimento**: Exploração de processos iterativos que integram a depuração contínua;
- **Aplicação Integrada**: Utilização de todos os elementos da metodologia **DAAD** em projetos práticos;
- **Desenvolvimento de Projetos**: Foco na resolução de problemas complexos, aplicando o raciocínio algorítmico a cenários concretos.

9.1 Depuração Sistemática

Esta etapa final do aprendizado se concentrará na **Depuração Sistemática**, uma competência fundamental para o desenvolvimento de software robusto. Serão exercitadas algumas das técnicas formais para a identificação de erros, métodos de teste e procedimentos para o rastreamento da execução de programas.

1. **Princípios de Teste de Software**: serão abordados os fundamentos do teste de software, incluindo teste de unidade e teste de integração. A abordagem será prática, por meio de um conjunto de exercícios que permitirão aos alunos aplicar os conceitos em cenários controlados.

9.2 Controle de Versão e Colaboração

1. **Sistemas de Controle de Versão**: introdução aos sistemas de controle de versão, com foco no projeto Git (155) e na plataforma GitHub (156). O objetivo é demonstrar como essas ferramentas facilitam o desenvolvimento colaborativo, o gerenciamento de alterações e o rastreamento de erros em projetos de software. Estes sistemas serão utilizados para versionar o código desenvolvido pelos alunos, permitindo que eles aprendam a colaborar efetivamente em projetos de programação mesmo que não estejam fisicamente juntos.

9.3 Exercícios: Sistemas Complexos com Testes e Controle de Versão

Estes exercícios são os mais complexos e desafiadores da disciplina. Ainda assim, eles foram planejados para serem acessíveis, exigindo a aplicação de todos os conceitos aprendidos, de forma progressiva. Caberá ao professor apresentar a teoria de testes unitários, testes de integração e controle de versão, de forma lúdica, fluida e interativa. Os exercícios também foram projetados para serem aplicados com a Técnica da Sequência de Fibonacci. Todavia, existem 8 exercícios, em vez dos 7 (1, 1, 2, 3) que usamos antes. Este execício extra, o **A4** pode ser usado pelo professor para introduzir os conceitos necessários.

Todos os exercícios foram planejados considerando que os alunos poderão usar as ferramentas de busca e inquisição baseadas em Inteligência Artificial, para auxiliar na resolução dos problemas complementando conhecimentos que não estejam diretamente ligados a resolução do problema ou ao **Raciocínio Algorítmico**. Por exemplo, o professor deve incentivar o uso dessas ferramentas para desenvolver testes automatizados no Github. A ideia é que os alunos aprendam a usar essas ferramentas como ferramentas de suporte ao processo de aprendizado e resolução de problemas sem perder a autonomia e o controle sobre seu próprio aprendizado.

9.3.1 Par 1: Sistema de Biblioteca Digital

9.3.1.1 A4: O Bibliotecário Virtual Inteligente**

Uma biblioteca universitária precisa modernizar seu sistema de gestão, criando um programa que gerencie empréstimos, devoluções, multas e recomendações de livros. O sistema deve trabalhar com múltiplos arquivos CSV (livros.csv, usuarios.csv, emprestimos.csv) e oferecer funcionalidades como busca inteligente, cálculo automático de multas e sugestões personalizadas baseadas no histórico de leitura.

Funções a implementar:

- **Estruturas de dados**: arrays para livros, usuários e empréstimos; estruturas para representar cada entidade;
- **Funções de validação**: `validarLivro()`, `validarUsuario()`, `validarDataEmprestimo()`, `validarCPF()`;
- **Funções de cálculo**: `calcularMulta()`, `calcularPrazoVencimento()`, `calcularCompatibilidadeL`
- **Funções de busca**: `buscarLivroPorTitulo()`, `buscarLivroPorAutor()`, `buscarUsuarioPorCPF()`, `verificarDisponibilidade()`;
- **Funções de recomendação**: `sugerirLivrosSimilares()`, `analisarHistoricoUsuario()`, `gerarRecomendacaoPersonalizada()`;
- **Funções de relatório**: `gerarRelatorioEmprestimos()`, `gerarRelatorioMultas()`, `gerarEstatisticasGerais()`;
- **Funções de persistência**: `salvarLivros()`, `carregarLivros()`, `salvarEmprestimos()`, `carregarEmprestimos()`.

Entrada: dados de livros (ISBN, título, autor, categoria, ano), usuários (CPF, nome, curso, semestre) e empréstimos (data, prazo, status).

Saída: Sistema interativo com menus para todas as operações, relatórios formatados e recomendações personalizadas.

9.3.1.1.1 Decomposição: Divisão de Tarefas para o Problema A4

Aluno 1: Módulo de Validação + Testes de Persistência

- **Desenvolver**: `validarLivro()`, `validarUsuario()`, `validarDataEmprestimo()`, `validarCPF()`;
- **Testar**: Funções de persistência do Aluno 4 (`salvarLivros()`, `carregarLivros()`, `salvarEmprestimos()`, `carregarEmprestimos()`);
- **Testes a criar**: `testarSalvarLivros()`, `testarCarregarLivros()`, `testarSalvarEmprestimos()`, `testarCarregarEmprestimos()`;
- **Foco**: Garantir entrada de dados válida e verificar se persistência funciona corretamente.

Aluno 2: Módulo de Cálculos + Testes de Validação

- **Desenvolver**: `calcularMulta()`, `calcularPrazoVencimento()`, `calcularCompatibilidadeLeitura` `verificarDisponibilidade()`;
- **Testar**: Funções de validação do Aluno 1 (`validarLivro()`, `validarUsuario()`, `validarDataEmprestimo()`, `validarCPF()`);
- **Testes a criar**: `testarValidarLivro()`, `testarValidarUsuario()`, `testarValidarDataEmprestimo` `testarValidarCPF()`;
- **Foco**: Algoritmos de cálculo precisos e validação robusta de entrada.

Aluno 3: Módulo de Busca e Relatórios + Testes de Cálculos

- **Desenvolver**: `buscarLivroPorTitulo()`, `buscarLivroPorAutor()`, `buscarUsuarioPorCPF()`, `gerarRelatorioEmprestimos()`, `gerarRelatorioMultas()`, `gerarEstatisticasGerais()`;
- **Testar**: Funções de cálculo do Aluno 2 (`calcularMulta()`, `calcularPrazoVencimento()`, `calcularCompatibilidadeLeitura()`, `verificarDisponibilidade()`);
- **Testes a criar**: `testarCalcularMulta()`, `testarCalcularPrazoVencimento()`, `testarCalcularCompatibilidadeLeitura()`, `testarVerificarDisponibilidade()`;
- **Foco**: Busca eficiente e relatórios informativos com cálculos corretos.

Aluno 4: Módulo de Recomendação e Persistência + Testes de Busca

- **Desenvolver**: `sugerirLivrosSimilares()`, `analisarHistoricoUsuario()`, `gerarRecomendacaoPersonalizada()`, `salvarLivros()`, `carregarLivros()`, `salvarEmprestimos()`, `carregarEmprestimos()`;
- **Testar**: Funções de busca e relatórios do Aluno 3 (`buscarLivroPorTitulo()`, `buscarLivroPorAutor()`, `buscarUsuarioPorCPF()`, `gerarRelatorioEmprestimos()`, `gerarRelatorioMultas()`, `gerarEstatisticasGerais()`);
- **Testes a criar**: `testarBuscarLivroPorTitulo()`, `testarBuscarLivroPorAutor()`, `testarBuscarUsuarioPorCPF()`, `testarGerarRelatorioEmprestimos()`, `testarGerarRelatorioMul` `testarGerarEstatisticasGerais()`;
- **Foco**: Sistema de recomendação inteligente e verificação da precisão das buscas.

9.3.1.2 B4: Testando a Biblioteca dos Sonhos

Implementação completa do sistema da biblioteca com todas as funções de A4, agora com foco na integração entre módulos e refinamento dos testes. Cada aluno implementa suas funções de A4 e melhora os testes que já desenvolveu, criando cenários mais complexos e edge cases.

Funções a implementar:

- Todas as funções de A4 totalmente implementadas;
- Aprimoramento dos testes unitários com mais cenários;
- Interface principal integrando todos os módulos;
- Sistema de logs para rastrear operações.

Entrada: Datasets de teste mais complexos com casos extremos e dados corrompidos;
Saída: Sistema robusto com testes abrangentes e relatórios de qualidade

9.3.1.2.1 Decomposição: Divisão de Tarefas para o Problema B4

Aluno 1: Refinamento de Validação + Testes Avançados de Persistência

- **Implementar**: versões robustas das funções de validação de A4;
- **Expandir testes**: criar 3-4 cenários de teste para cada função de persistência;
- **Casos extremos**: arquivos corrompidos, dados inconsistentes, limites de memória;
- **Foco**: validação à prova de falhas e persistência confiável.

Aluno 2: Otimização de Cálculos + Testes Complexos de Validação

- **Implementar**: versões otimizadas das funções de cálculo de A4;
- **Expandir testes**: criar 3-4 cenários para cada função de validação;
- **Casos extremos**: datas inválidas, CPFs mal formados, dados vazios;
- **Foco**: cálculos precisos e validação abrangente.

Aluno 3: Interface Principal + Testes de Performance de Cálculos

- **Implementar**: interface que integra busca e relatórios de A4;
- **Expandir testes**: testar performance e precisão dos cálculos;
- **Casos extremos**: grandes volumes de dados, cálculos com overflow;
- **Foco**: interface intuitiva e verificação de performance.

Aluno 4: Sistema de Logs + Testes de Integração de Busca

- **Implementar**: sistema de recomendação e persistência de A4 + logs;
- **Expandir testes**: testar busca com grandes datasets;
- **Casos extremos**: buscas com critérios inexistentes, dados duplicados;
- **Foco**: rastreabilidade do sistema e busca robusta.

9.3.2 Par 2: Sistema de Delivery de Comida

9.3.2.1 C4: O Delivery que Nunca Falha**

Uma startup de delivery precisa de um sistema robusto que gerencie pedidos, calcule fretes, estime tempos de entrega e processe pagamentos. O sistema deve trabalhar com dados de restaurantes, pratos, clientes e entregadores, oferecendo funcionalidades como aplicação de descontos promocionais, rastreamento de pedidos e análise de satisfação do cliente.

Funções a implementar:

- **Estruturas de dados**: arrays para restaurantes, pratos, clientes, pedidos e entregadores;
- **Funções de validação**: `validarPedido()`, `validarEnderecoEntrega()`, `validarFormaPagamento()` `validarPromocao()`;
- **Funções de cálculo**: `calcularFrete()`, `calcularDesconto()`, `calcularTaxaServico()`, `calcularTempoEstimado()`;
- **Funções de processamento**: `processarPagamento()`, `alocarEntregador()`, `atualizarStatusPedido()`, `confirmarEntrega()`;
- **Funções de rastreamento**: `rastrearPedido()`, `notificarCliente()`, `atualizarLocalizacao()`, `calcularTempoRestante()`;
- **Funções de análise**: `avaliarRestaurante()`, `analisarSatisfacao()`, `gerarRelatorioVendas()`, `identificarProblemasEntrega()`;
- **Funções de persistência**: `salvarPedidos()`, `carregarRestaurantes()`, `salvarAvaliacoes()`, `carregarHistorico()`;

Entrada: Dados de restaurantes (nome, endereço, cardápio, avaliação), pedidos (itens, quantidade, observações) e informações de entrega.

Saída: Interface de pedidos com cálculos em tempo real, sistema de rastreamento e relatórios de performance.

9.3.2.1.1 Decomposição: Divisão de Tarefas para o Problema C4

Aluno 1: Módulo de Validação + Testes de Análise

- **Desenvolver**: `validarPedido()`, `validarEnderecoEntrega()`, `validarFormaPagamento()`, `validarPromocao()`;
- **Testar**: Funções de análise do Aluno 4 (`avaliarRestaurante()`, `analisarSatisfacao()`, `gerarRelatorioVendas()`, `identificarProblemasEntrega()`);
- **Testes a criar**: `testarAvaliarRestaurante()`, `testarAnalisarSatisfacao()`, `testarGerarRelatorioVendas()`, `testarIdentificarProblemasEntrega()`;
- **Foco**: Entrada de dados confiável e verificação da qualidade das análises.

Aluno 2: Módulo de Cálculos + Testes de Validação

- **Desenvolver**: `calcularFrete()`, `calcularDesconto()`, `calcularTaxaServico()`, `calcularTempoEstimado()`;
- **Testar**: Funções de validação do Aluno 1 (`validarPedido()`, `validarEnderecoEntrega()`, `validarFormaPagamento()`, `validarPromocao()`);
- **Testes a criar**: `testarValidarPedido()`, `testarValidarEnderecoEntrega()`, `testarValidarFormaPagamento()`, `testarValidarPromocao()`;
- **Foco**: Cálculos financeiros precisos e validação robusta.

Aluno 3: Módulo de Processamento + Testes de Cálculos

- **Desenvolver**: `processarPagamento()`, `alocarEntregador()`, `atualizarStatusPedido()`, `confirmarEntrega()`;
- **Testar**: Funções de cálculo do Aluno 2 (`calcularFrete()`, `calcularDesconto()`, `calcularTaxaServico()`, `calcularTempoEstimado()`);
- **Testes a criar**: `testarCalcularFrete()`, `testarCalcularDesconto()`, `testarCalcularTaxaServic` `testarCalcularTempoEstimado()`;
- **Foco**: Fluxo de processamento eficiente e cálculos corretos.

Aluno 4: Módulo de Rastreamento e Análise + Testes de Processamento

- **Desenvolver**: `rastrearPedido()`, `notificarCliente()`, `atualizarLocalizacao()`, `calcularTempoRestante()`, `avaliarRestaurante()`, `analisarSatisfacao()`, `gerarRelatorioVendas()`, `identificarProblemasEntrega()`;
- **Testar**: Funções de processamento do Aluno 3 (`processarPagamento()`, `alocarEntregador()`, `atualizarStatusPedido()`, `confirmarEntrega()`);
- **Testes a criar**: `testarProcessarPagamento()`, `testarAlocarEntregador()`, `testarAtualizarStatusPedido()`, `testarConfirmarEntrega()`;
- **Foco**: Sistema de rastreamento em tempo real e verificação do processamento

9.3.2.2 D4: Integrando o Restaurante Digital

Implementação completa do sistema de delivery com foco em fluxos integrados. O sistema deve demonstrar como validação, cálculos, processamento e rastreamento trabalham em conjunto, simulando o fluxo completo desde o pedido até a entrega.

Funções a implementar:

- Todas as funções de **C4** implementadas e integradas;
- Fluxos completos de negócio (pedido → pagamento → entrega);
- Tratamento de erros em cenários reais;
- Métricas de performance do sistema.

Entrada: Simulação de múltiplos pedidos simultâneos com cenários de stress.
Saída: Sistema integrado com relatórios de fluxo e métricas de performance.

9.3.2.2.1 Decomposição: Divisão de Tarefas para o Problema D4

Aluno 1: Coordenação de Fluxos + Testes de Stress de Análise

- **Implementar**: sistema que coordena validação com outros módulos;
- **Expandir testes**: testar análises com grandes volumes de dados;
- **Cenários complexos**: múltiplos pedidos simultâneos, análises em tempo real;
- **Foco**: coordenação entre módulos e robustez das análises.

Aluno 2: Otimização de Performance + Testes de Integração de Validação

- **Implementar**: versões otimizadas dos cálculos para alta demanda;
- **Expandir testes**: testar validação em cenários de stress;
- **Cenários complexos**: validação de centenas de pedidos por minuto;
- **Foco**: performance sob carga e validação eficiente.

Aluno 3: Tratamento de Erros + Testes de Fluxo de Cálculos

- **Implementar**: sistema robusto de processamento com recuperação de erros;
- **Expandir testes**: testar cálculos em cenários de falha;
- **Cenários complexos**: falhas de rede, timeouts, dados corrompidos;
- **Foco**: resiliência do sistema e confiabilidade dos cálculos.

Aluno 4: Monitoramento e Métricas + Testes End-to-End

- **Implementar**: sistema de monitoramento do rastreamento e análise;
- **Expandir testes**: testes completos do fluxo pedido → entrega;
- **Cenários complexos**: fluxos com falhas e recuperação automática;
- **Foco**: observabilidade do sistema e testes de ponta a ponta.

9.3.3 Par 3: Sistema de Rede Social Universitária

9.3.3.1 E4: A Rede Social da Turma**

Uma universidade quer criar uma rede social interna para conectar estudantes, professores e funcionários. O sistema deve gerenciar perfis, conexões, posts, eventos e grupos de estudo, oferecendo funcionalidades como feed personalizado, sistema de compatibilidade acadêmica e análise de sentimentos das postagens.

Funções a implementar:

- **Estruturas de dados**: arrays para usuários, posts, conexões, grupos e eventos;
- **Funções de validação**: `validarUsuario()`, `validarPost()`, `validarEvento()`, `validarGrupo()`;
- **Funções de compatibilidade**: `calcularCompatibilidadeAcademica()`, `sugerirAmigos()`, `analisarInteressesComuns()`, `identificarGruposRelevantes()`;
- **Funções de conteúdo**: `filtrarPosts()`, `ordenarFeed()`, `moderarConteudo()`, `analisarSentimento()`;
- **Funções sociais**: `criarConexao()`, `removerConexao()`, `bloquearUsuario()`, `reportarConteudo()`;
- **Funções de eventos**: `criarEvento()`, `confirmarPresenca()`, `notificarParticipantes()`, `gerarResumoEvento()`;
- **Funções de análise**: `gerarEstatisticasPerfil()`, `analisarEngajamento()`, `identificarTendencias()`, `gerarRelatorioAtividade()`.

Entrada: dados de usuários (matrícula, curso, interesses), posts (texto, imagens, tags) e eventos (data, local, descrição).

Saída: interface social com feed personalizado, sugestões de conexões e análise de atividade.

9.3.3.1.1 Decomposição: Divisão de Tarefas para o Problema E4

Aluno 1: Módulo de Validação + Testes de Eventos

- **Desenvolver**: `validarUsuario()`, `validarPost()`, `validarEvento()`, `validarGrupo()`;
- **Testar**: Funções de eventos do Aluno 4 (`criarEvento()`, `confirmarPresenca()`, `notificarParticipantes()`, `gerarResumoEvento()`);
- **Testes a criar**: `testarCriarEvento()`, `testarConfirmarPresenca()`, `testarNotificarParticipar` `testarGerarResumoEvento()`;
- **Foco**: Validação robusta de conteúdo social e funcionamento dos eventos.

Aluno 2: Módulo de Compatibilidade + Testes de Validação

- **Desenvolver**: `calcularCompatibilidadeAcademica()`, `sugerirAmigos()`, `analisarInteressesCom` `identificarGruposRelevantes()`;
- **Testar**: Funções de validação do Aluno 1 (`validarUsuario()`, `validarPost()`, `validarEvento()`, `validarGrupo()`);
- **Testes a criar**: `testarValidarUsuario()`, `testarValidarPost()`, `testarValidarEvento()`, `testarValidarGrupo()`;
- **Foco**: Algoritmos sociais inteligentes e validação de entrada.

Aluno 3: Módulo de Conteúdo e Social + Testes de Compatibilidade

- **Desenvolver**: `filtrarPosts()`, `ordenarFeed()`, `moderarConteudo()`, `analisarSentimento()`, `criarConexao()`, `removerConexao()`, `bloquearUsuario()`, `reportarConteudo()`;

- **Testar**: Funções de compatibilidade do Aluno 2 (`calcularCompatibilidadeAcademica()`, `sugerirAmigos()`, `analisarInteressesComuns()`, `identificarGruposRelevantes()`);
- **Testes a criar**: `testarCalcularCompatibilidadeAcademica()`, `testarSugerirAmigos()`, `testarAnalisarInteressesComuns()`, `testarIdentificarGruposRelevantes()`;
- **Foco**: curadoria de conteúdo e verificação dos algoritmos sociais.

Aluno 4: Módulo de Eventos e Análise + Testes de Conteúdo

- **Desenvolver**: `criarEvento()`, `confirmarPresenca()`, `notificarParticipantes()`, `gerarResumoEvento()`, `gerarEstatisticasPerfil()`, `analisarEngajamento()`, `identificarTendencias()`, `gerarRelatorioAtividade()`;
- **Testar**: Funções de conteúdo do Aluno 3 (`filtrarPosts()`, `ordenarFeed()`, `moderarConteudo()`, `analisarSentimento()`, `criarConexao()`, `removerConexao()`, `bloquearUsuario()`, `reportarConteudo()`);
- **Testes a criar**: `testarFiltrarPosts()`, `testarOrdenarFeed()`, `testarModerarConteudo()`, `testarAnalisarSentimento()`, `testarCriarConexao()`, `testarRemoverConexao()`, `testarBloquearUsuario()`, `testarReportarConteudo()`;
- **Foco**: gestão de eventos e verificação da curadoria de conteúdo.

9.3.3.2 F4: Conectando a Universidade**

Implementação completa da rede social com foco em desenvolvimento colaborativo usando Git. O sistema deve demonstrar fluxos de trabalho em equipe, resolução de conflitos de merge e implementação de features em paralelo usando controle de versão.

Funções a implementar:

- Todas as funções de E4 implementadas com controle de versão;
- Workflow de desenvolvimento usando Git (branches, pull requests);
- Resolução colaborativa de conflitos de merge;
- Documentação e versionamento do código.

Entrada: desenvolvimento colaborativo com múltiplas features em paralelo.
Saída: sistema integrado com histórico completo de desenvolvimento no Git.

9.3.3.2.1 Decomposição: Divisão de Tarefas para o Problema F4

Aluno 1: Feature Branch 'validation' + Gestão de Conflitos

- **Implementar**: módulo de validação de E4 em branch separada;
- **Git workflow**: criar branch `feature/validation`, commits regulares;
- **Resolução de conflitos**: coordenar merges com outras features;
- **Foco**: desenvolvimento isolado e integração colaborativa.

Aluno 2: Feature Branch 'social-algorithms' + Code Review

- **Implementar**: módulo de compatibilidade de E4 em branch separada;
- **Git workflow**: branch `feature/social-algorithms`, pull requests;
- **Code review**: revisar código dos outros alunos antes do merge;
- **Foco**: qualidade de código e revisão colaborativa.

Aluno 3: Feature Branch 'content-curation' + Documentação

- **Implementar**: módulo de conteúdo de E4 em branch separada;
- **Git workflow**: branch `feature/content-curation`, documentação no Git;

- **Documentação**: README, comentários, wiki do projeto;
- **Foco**: funcionalidades principais e documentação clara.

Aluno 4: Branch 'events-analytics' + Integração Final

- **Implementar**: módulo de eventos de E4 em branch separada;
- **Git workflow**: branch `feature/events-analytics`, coordenação de merges;
- **Integração**: merge final de todas as features na branch main;
- **Foco**: coordenação geral e integração do projeto.

9.3.4 Par 4: Sistema de E-commerce Estudantil (COM TESTES DE INTEGRAÇÃO)

9.3.4.1 G4: A Loja Virtual dos Universitários

Estudantes universitários querem criar um marketplace para vender e trocar itens entre colegas (livros usados, eletrônicos, roupas, materiais de estudo). O sistema deve gerenciar produtos, vendedores, compradores, transações e avaliações, oferecendo funcionalidades como busca inteligente, sistema de reputação e cálculo automático de preços baseado em condição do produto.

Funções a implementar:

- **Estruturas de dados**: Arrays para produtos, usuários, transações, avaliações e categorias;
- **Funções de validação**: `validarProduto()`, `validarTransacao()`, `validarAvaliacao()`, `validarPagamento()`;
- **Funções de precificação**: `calcularPrecoFinal()`, `aplicarDesconto()`, `calcularFrete()`, `estimarDepreciacao()`;
- **Funções de estoque**: `gerenciarEstoque()`, `verificarDisponibilidade()`, `reservarProduto()`, `atualizarQuantidade()`;
- **Funções de busca**: `buscarProdutos()`, `filtrarPorCategoria()`, `ordenarResultados()`, `buscarVendedor()`;
- **Funções de recomendação**: `recomendarProdutos()`, `analisarHistoricoCompras()`, `sugerirSimilares()`, `identificarTendencias()`;
- **Funções de avaliação**: `processarAvaliacao()`, `calcularReputacao()`, `gerarRelatorioVendedor()`, `moderarComentarios()`;
- **Funções financeiras**: `processarPagamento()`, `calcularComissao()`, `gerarFatura()`, `processarEstorno()`.

INTRODUÇÃO A TESTES DE INTEGRAÇÃO: Neste exercício, além dos testes unitários simples, cada aluno criará **testes de integração** que verificam como seus módulos interagem com os de outros alunos.

Entrada: dados de produtos (nome, categoria, condição, preço), usuários vendedores/compradores e transações.

Saída: marketplace funcional com sistema de busca, avaliações e relatórios de vendas.

9.3.4.1.1 Decomposição: Divisão de Tarefas para o Problema G4

Aluno 1: Módulo de Validação + Testes de Integração Financeiro

- **Desenvolver**: `validarProduto()`, `validarTransacao()`, `validarAvaliacao()`, `validarPagamento()`;
- **Testar unitário**: Funções financeiras do Aluno 4 (`processarPagamento()`, `calcularComissao()`, `gerarFatura()`, `processarEstorno()`);

- **Testes de integração**: `testarFluxoValidacaoParaPagamento()` - verificar se dados validados fluem corretamente para o módulo financeiro;
- **Foco**: Validação robusta e integração com sistema financeiro.

Aluno 2: Módulo de Precificação e Estoque + Testes de Integração Validação

- **Desenvolver**: `calcularPrecoFinal()`, `aplicarDesconto()`, `calcularFrete()`, `estimarDepreciacao()`, `gerenciarEstoque()`, `verificarDisponibilidade()`, `reservarProduto()`, `atualizarQuantidade()`;
- **Testar unitário**: Funções de validação do Aluno 1 (`validarProduto()`, `validarTransacao()`, `validarAvaliacao()`, `validarPagamento()`);
- **Testes de integração**: `testarFluxoProdutoValidadoParaEstoque()` - verificar se produtos validados são corretamente adicionados ao estoque;
- **Foco**: Gestão de preços/estoque e integração com validação.

Aluno 3: Módulo de Busca e Recomendação + Testes de Integração Estoque

- **Desenvolver**: `buscarProdutos()`, `filtrarPorCategoria()`, `ordenarResultados()`, `buscarVendedor()`, `recomendarProdutos()`, `analisarHistoricoCompras()`, `sugerirSimilares()`, `identificarTendencias()`;
- **Testar unitário**: Funções de estoque do Aluno 2 (`gerenciarEstoque()`, `verificarDisponibilidade()`, `reservarProduto()`, `atualizarQuantidade()`);
- **Testes de integração**: `testarBuscaComEstoqueAtualizado()` - verificar se busca reflete mudanças de estoque em tempo real;
- **Foco**: Sistema de busca inteligente e integração com estoque.

Aluno 4: Módulo Financeiro e Avaliação + Testes de Integração Busca

- **Desenvolver**: `processarPagamento()`, `calcularComissao()`, `gerarFatura()`, `processarEstorno()`, `processarAvaliacao()`, `calcularReputacao()`, `gerarRelatorioVendedor()`, `moderarComentarios()`;
- **Testar unitário**: Funções de busca do Aluno 3 (`buscarProdutos()`, `filtrarPorCategoria()`, `ordenarResultados()`, `buscarVendedor()`);
- **Testes de integração**: `testarCompraCompletaComBusca()` - verificar fluxo completo: busca → seleção → pagamento → avaliação;
- **Foco**: Sistema financeiro e integração com todo o fluxo de compra.

9.3.4.2 H4: Lançando o Marketplace da Turma

Projeto final integrando todas as competências do módulo: testes unitários, testes de integração avançados e controle de versão com CI/CD. O sistema deve demonstrar um pipeline completo de desenvolvimento com automated testing, code review e releases via GitHub.

FERRAMENTAS DE TESTE DE INTEGRAÇÃO:

- **Python**: pytest com fixtures e mocks;
- **C++**: doctest ou Catch2 com seções de integração.

Funções a implementar:

- implementação completa de todas as funções do **G4**;
- testes unitários e de integração para todas as funções;
- Pipeline CI/CD básico usando GitHub Actions;
- Testes automatizados com framework de teste;
- Sistema de releases e versionamento semântico.

Entrada: desenvolvimento colaborativo com múltiplas features, releases e *hotfixes*.
Saída: marketplace funcional com pipeline de desenvolvimento profissional completo.

9.3.4.2.1 Decomposição: Divisão de Tarefas para o Problema H4

Aluno 1: Implementação Core + Suite de Testes Unitários

- **Implementar**: módulo de validação de **G4** com melhorias;
- **Framework de teste**: Configurar pytest (Python) ou doctest (C++) no projeto;
- **Testes automatizados**: Criar 20+ testes unitários com cobertura $> 80\%$;
- **CI/CD**: Configurar GitHub Actions para executar testes automaticamente;
- **Foco**: Base sólida com testes automatizados.

Aluno 2: Features Avançadas + Testes de Integração

- **Implementar**: Módulo de precificação/estoque de **G4** com otimizações;
- **Testes de integração**: Usar framework para testar interação entre módulos;
- **Performance testing**: Testes de carga e stress usando ferramentas do framework;
- **Mocks e fixtures**: Criar dados de teste reutilizáveis;
- **Foco**: Testes sofisticados e análise de performance.

Aluno 3: Interface e UX + Testes End-to-End

- **Implementar**: Módulo de busca/recomendação de **G4** com interface melhorada;
- **Testes end-to-end**: Testar fluxos completos de usuário;
- **Documentação**: Criar documentação técnica e de usuário;
- **Code coverage**: Gerar relatórios de cobertura de código;
- **Foco**: Experiência do usuário e testes abrangentes.

Aluno 4: DevOps e Release Management + Integração Contínua

- **Implementar**: Módulo financeiro de **G4** com logging e monitoramento;
- **DevOps**: Configurar pipeline completo de CI/CD no GitHub;
- **Release management**: Implementar versionamento semântico e releases automatizadas;
- **Quality gates**: Configurar verificações automáticas de qualidade;
- **Monitoring**: Sistema básico de logs e métricas;
- **Foco**: Automação e gestão de releases profissional.

9.4 Metodologia de Testes Unitários Simples

9.4.1 Para Exercícios A4-F4 (Sem Frameworks)

Os testes devem ser funções simples que retornam `true` se o teste passou ou `false` se falhou:

Exemplo em C++:

```
bool testarValidarLivro() {
    // Teste caso válido
    if (!validarLivro("1234567890123", "Algoritmos", "Cormen", 2020)) {
        cout << "ERRO: Livro válido rejeitado" << endl;
        return false;
    }

    // Teste caso inválido - ISBN curto
    if (validarLivro("123", "Título", "Autor", 2020)) {
        cout << "ERRO: ISBN inválido aceito" << endl;
        return false;
```

```
    }

    // Teste caso inválido - ano futuro
    if (validarLivro("1234567890123", "Título", "Autor", 2030)) {
        cout << "ERRO: Ano futuro aceito" << endl;
        return false;
    }

    cout << "  testarValidarLivro passou em todos os casos" << endl;
    return true;
}
```

Exemplo em Python:

```
def testar_validar_livro():
    # Teste caso válido
    if not validar_livro("1234567890123", "Algoritmos", "Cormen", 2020):
        print("ERRO: Livro válido rejeitado")
        return False

    # Teste caso inválido - ISBN curto
    if validar_livro("123", "Título", "Autor", 2020):
        print("ERRO: ISBN inválido aceito")
        return False

    # Teste caso inválido - ano futuro
    if validar_livro("1234567890123", "Título", "Autor", 2030):
        print("ERRO: Ano futuro aceito")
        return False

    print("  testar_validar_livro passou em todos os casos")
    return True
```

9.4.2 Para Exercícios G4-H4 (Com Frameworks)

Python com pytest:

```
import pytest

def test_validar_produto():
    assert validar_produto("Notebook Dell", "Eletrônicos", "Usado", 1500.0)
    assert not validar_produto("", "Eletrônicos", "Usado", 1500.0)  # Nome vazio
    assert not validar_produto("Notebook", "Categoria_Inexistente", "Usado", 1500.0)

def test_integracao_validacao_estoque():
    # Teste de integração
    produto = {"nome": "iPhone 12", "categoria": "Eletrônicos", "preco": 2000.0}
    assert validar_produto(produto["nome"], produto["categoria"], "Novo", produto["preco"]
    assert adicionar_ao_estoque(produto)
    assert verificar_disponibilidade(produto["nome"]) > 0
```

C++ com doctest:

```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
#include "doctest.h"

TEST_CASE("Validação de produto") {
    CHECK(validarProduto("Notebook Dell", "Eletrônicos", "Usado", 1500.0));
    CHECK_FALSE(validarProduto("", "Eletrônicos", "Usado", 1500.0)); // Nome vazio
}

TEST_CASE("Integração validação-estoque") {
    Produto p = {"iPhone 12", "Eletrônicos", 2000.0, "Novo"};
    REQUIRE(validarProduto(p.nome, p.categoria, p.condicao, p.preco));
    REQUIRE(adicionarAoEstoque(p));
    CHECK(verificarDisponibilidade(p.nome) > 0);
}
```

9.5 Rubrica e Tecnologia de Avaliação

A avaliação dos exercicios pode, e deve ser completamente automatizada usando ferramentas de Inteligência Artificial integradas ao Github Classroom (157), para verificar a implementação e os testes. Os critérios de avaliação podem incluir:

- **Funcionalidade**: 25% - Sistema funciona conforme especificado
- **Qualidade dos Testes**: 30% - Cobertura, cenários e qualidade dos testes
- **Colaboração**: 25% - Divisão de tarefas e trabalho em equipe efetivo
- **Controle de Versão**: 20% - Uso adequado do Git (F4 e H4)

9.5.1 Visão Geral da Avaliação

Público-alvo: alunos do primeiro período de Computação;
Modalidade: avaliação automatizada via GitHub Classroom integrado a ferramentas de Inteligência Artificial;

A tabela Table 9.1 resume os critérios de avaliação e seus pesos:

Table 9.1: Tabela de Rubrica de Avaliação.

Critério	Peso	Pontos Máximos
Funcionalidade	25%	25
Qualidade dos Testes	30%	30
Colaboração	25%	25
Controle de Versão	20%	20
TOTAL	100%	**100**

9.5.2 Critério 1: Funcionalidade (25 pontos)

Descrição: sistema implementado funciona conforme especificação, com todas as funções operacionais e requisitos atendidos. A tabela Table 9.2 define os níveis de funcionalidade e seus indicadores quantitativos:

Table 9.2: Tabela de Níveis de Funcionalidade para Avaliação.

Nível	Pontos	Descrição	Indicadores Quantitativos
Avançado	22-25	Sistema funciona perfeitamente em todos os cenários testados	• 90-100% das funções implementadas • 0 erros de compilação • 0 crashes durante execução • Todos os requisitos funcionais atendidos
Proficiente	18-21	Sistema funciona bem na maioria dos cenários	• 75-89% das funções implementadas • 0 erros de compilação • Máximo 1 crash em cenários extremos • 80%+ dos requisitos atendidos
Desenvolvendo	12-17	Sistema funciona parcialmente com algumas limitações	• 50-74% das funções implementadas • Máximo 2 erros menores de compilação • Funciona para casos básicos • 60%+ dos requisitos atendidos
Iniciante	0-11	Sistema não funciona ou funciona de forma muito limitada	• Menos de 50% das funções implementadas • Múltiplos erros de compilação • Falhas frequentes de execução • Menos de 60% dos requisitos atendidos

9.5.2.1 Métricas Automatizadas

- Compilação bem-sucedida (Python: execução sem SyntaxError, C++: g++ sem erros);
- Execução de casos de teste básicos sem crashes;
- Cobertura de implementação das funções especificadas;
- Validação de entrada e tratamento de erros.

9.5.3 Critério 2: Qualidade dos Testes (30 pontos)

Descrição: Testes unitários e de integração demonstram compreensão da verificação de software e cobertura adequada dos cenários. A tabela Table 9.3 define os níveis de qualidade dos testes e seus indicadores quantitativos:

Table 9.3: Tabela de Níveis de Qualidade dos Testes para Avaliação.

Nível	Pontos	Descrição	Indicadores Quantitativos
Avançado	26-30	Testes abrangentes com cenários complexos e edge cases	• 15+ testes unitários implementados • 90%+ dos testes passando • Inclui casos extremos e erro • Testes claros e bem documentados • Divisão cruzada implementada corretamente
Proficiente	21-25	Testes adequados cobrindo cenários principais	• 10-14 testes unitários implementados • 80-89% dos testes passando • Inclui alguns casos de erro • Maioria dos testes compreensíveis • Divisão cruzada parcialmente implementada
Desenvolvendo	13-20	Testes básicos cobrindo funcionalidades principais	• 6-9 testes unitários implementados • 70-79% dos testes passando • Casos básicos cobertos • Testes simples mas funcionais • Tentativa de divisão cruzada

Nível	Pontos	Descrição	Indicadores Quantitativos
Iniciante	0-12	Testes insuficientes ou inadequados	• Menos de 6 testes implementados • Menos de 70% dos testes passando • Testes muito simples ou não funcionais • Sem divisão cruzada adequada

9.5.3.1 Critérios Específicos por Exercício

9.5.3.1.1 A4-F4: Testes Unitários Simples

- Funções `testarFuncaoX()` retornando `bool`;
- Casos válidos e inválidos testados;
- Mensagens informativas de erro;
- Cada aluno testa funções de outro aluno.

9.5.3.1.2 G4-H4: Testes de Integração

- Uso de framework (pytest/doctest);
- Testes de fluxo completo entre módulos;
- Mocks e fixtures apropriados;
- Testes de performance e stress.

9.5.3.1.3 Métricas Automatizadas

- Número de testes implementados vs especificados;
- Taxa de sucesso dos testes (% que passam);
- Cobertura de casos de uso (normal, erro, extremo);
- Qualidade da nomenclatura e documentação dos testes.

9.5.4 Critério 3: Colaboração (25 pontos)

Descrição: Trabalho em equipe efetivo com divisão clara de responsabilidades e integração colaborativa. A tabela Table 9.4 define os níveis de colaboração e seus indicadores quantitativos:

Table 9.4: Tabela de Níveis de Colaboração para Avaliação.

Nível	Pontos	Descrição	Indicadores Quantitativos
Avançado	22-25	Colaboração exemplar com divisão equilibrada e integração perfeita	• 4 alunos contribuindo ativamente • Commits equilibrados (15-35% cada) • Integração sem conflitos • Documentação colaborativa clara • Peer review efetivo
Proficiente	18-21	Boa colaboração com divisão adequada	• 3-4 alunos contribuindo • Distribuição razoável de commits • Poucos conflitos de integração • Documentação adequada • Algum peer review

Nível	Pontos	Descrição	Indicadores Quantitativos
Desenvolvendo	12-17	Colaboração básica com algumas falhas de coordenação	• 2-3 alunos contribuindo significativamente • Distribuição desigual de trabalho • Alguns problemas de integração • Documentação mínima
Iniciante	0-11	Colaboração inadequada ou trabalho individual disfarçado	• 1-2 alunos fazendo a maioria do trabalho • Commits concentrados • Falhas graves de integração • Pouca ou nenhuma documentação colaborativa

9.5.4.1 Métricas Automatizadas:

- Distribuição de commits por autor;
- Número de pull requests e code reviews;
- Frequência de commits ao longo do tempo;
- Resolução de conflitos de merge;
- Qualidade das mensagens de commit.

9.5.5 Critério 4: Controle de Versão (20 pontos)

Descrição: uso adequado do Git e GitHub para versionamento, colaboração e integração contínua.

Nota: Aplicável apenas aos exercícios F4 e H4. A tabela Table 9.5 define os níveis de controle de versão e seus indicadores quantitativos:

Table 9.5: Tabela de Níveis de Controle de Versão para Avaliação.

Nível	Pontos	Descrição	Indicadores Quantitativos
Avançado	18-20	Uso profissional do Git com workflow completo	• Branches organizadas (feature/bugfix) • Pull requests com code review • Mensagens de commit descritivas • CI/CD funcionando (H4) • Releases com versionamento semântico • README e documentação atualizados
Proficiente	14-17	Uso competente do Git com boas práticas	• Uso básico de branches • Alguns pull requests • Mensagens de commit adequadas • Tentativa de CI/CD • Documentação básica
Desenvolvendo	10-13	Uso básico do Git com tentativas de organização	• Commits regulares • Tentativa de usar branches • Mensagens de commit simples mas claras • Estrutura básica de projeto
Iniciante	0-9	Uso inadequado ou mínimo do Git	• Poucos commits • Trabalho apenas na branch main • Mensagens vagas ou inadequadas • Sem organização do projeto

9.5.5.1 Métricas Automatizadas

- Número e frequência de commits;
- Uso de branches (feature, develop, main);

- Pull requests criados e merged;
- Qualidade das mensagens de commit (comprimento, clareza);
- Configuração de CI/CD (GitHub Actions);
- Estrutura do repositório (README, .gitignore, estrutura de pastas).

9.6 Sistema de Avaliação Automática

Este sistema de avaliação foi especialmente desenvolvido considerando que os estudantes estão em seu primeiro período de contato com a Ciência e Engenharia de Computação, priorizando o processo de descoberta e aprendizado sobre a perfeição do produto final. Os critérios reconhecem as limitações naturais de alunos iniciantes, valorizando tentativas bem-intencionadas, colaboração efetiva e compreensão dos conceitos fundamentais. O feedback gerado pelos relatórios automáticos terão caráter construtivo, fornecendo orientações específicas para melhorias em vez de simplesmente atribuir pontuações, ajudando os estudantes a identificarem próximos passos concretos em seu desenvolvimento.

O sistema mantém flexibilidade para ajustes baseados na performance geral da turma, permitindo calibrações quando necessário para garantir que os critérios permaneçam desafiadores mas alcançáveis. Todos os critérios de avaliação, métricas e expectativas são compartilhados com os alunos antes do início das atividades, garantindo transparência total no processo avaliativo e permitindo que os estudantes compreendam exatamente o que é esperado deles, promovendo assim um ambiente de aprendizagem mais justo e direcionado.

Todo o texto desta seção pode ser usado como *prompt* para um sistema de Inteligência Artificial que esteja integrado ao Github Classroom. Esta integração será responsável por gerar os relatórios de avaliação, feedback e sugestões de melhoria para cada aluno, com base nos critérios e métricas definidos nas seções a seguir.

9.6.1 Fluxo de Avaliação:

1. **Análise Automática do Repositório**:
   - Clonagem e análise da estrutura do projeto;
   - Compilação/execução automática do código;
   - Execução da suíte de testes;
   - Análise de métricas Git.
2. **Geração de Relatório**:
   - Pontuação automática por critério;
   - Identificação de pontos fortes e fracos;
   - Sugestões de melhoria específicas;
   - Comparação com padrões da turma.
3. **Validação Manual** (quando necessário):
   - Revisão de casos ambíguos;
   - Verificação de funcionalidades complexas;
   - Avaliação de aspectos qualitativos.

9.6.2 Tolerâncias para Primeiro Período:

- **Compilação**: Avisos permitidos, mas sem erros fatais;
- **Testes**: Foco na implementação correta vs performance;
- **Git**: Ênfase no uso básico correto vs workflows avançados;
- **Documentação**: Clara e útil vs detalhada e profissional.

9.6.3 Feedback Personalizado

Para cada critério de avaliação, o sistema deve fornecer feedback específico e orientações para melhorias.

Funcionalidade:

- Lista de funções implementadas vs especificadas;
- Casos de teste que falharam com explicações;
- Sugestões de correção para erros comuns.

Qualidade dos Testes:

- Comparação de cobertura com expectativa;
- Identificação de cenários não testados;
- Exemplos de melhorias nos testes existentes.

Colaboração:

- Gráfico de contribuições por membro;
- Identificação de desequilíbrios no trabalho;
- Sugestões para melhor distribuição de tarefas.

Controle de Versão:

- Análise do histórico de commits;
- Sugestões de workflow Git;
- Boas práticas não implementadas.

9.6.4 Critérios de Aprovação

Para aprovação no módulo: o aluno deve atingir pelo menos:

- **Pontuação mínima**: 70 pontos (70%);
- **Critérios obrigatórios**:
  - Funcionalidade $\geq$ 15 pontos;
  - Qualidade dos Testes $\geq$ 18 pontos;
  - Pelo menos um critério em nível “Proficiente”;

Referências

[1] TEACHING LONDON COMPUTING. **Algorithmic Thinking**., 2023. Disponível em: <https://teachinglondoncomputing.org/resources/developing-computational-thinking/algorithmic-thinking/>

[2] YANOFSKY, N. S. Towards a Definition of an Algorithm. **Journal of Logic and Computation**, v. 21, n. 2, p. 253–286, 2011.

[3] TYLDUM, M. **O Jogo da Imitação**. Filme; Black Bear Pictures, 2014.

[4] ZARE, M. et al. A Survey of Imitation Learning: Algorithms, Recent Developments, and Challenges. **IEEE Transactions on Cybernetics**, v. 54, p. 7173–7186, 2023.

[5] HIEBERT, J.; LEFEVRE, P. Conceptual and procedural knowledge in mathematics: An introductory analysis. Em: HIEBERT, J. (Ed.). **Conceptual and procedural knowledge: The case of mathematics**. Hillsdale, NJ: Lawrence Erlbaum Associates, 1986. p. 1–27.

[6] LI, Y. et al. Computational Thinking Is More about Thinking than Computing. **Journal for STEM Education Research**, v. 3, n. 1, p. 1–18, 2020.

[7] HUANG, R. et al. Computational Thinking and the New Curriculum Standards of Information Technology for Senior High Schools in China. Em: ABELSON, H.; KONG, S.-C. (Eds.). **Computational Thinking Curricula in K–12: International Implementations**. Cambridge, Massachusetts; London, England: The MIT Press, 2020.

[8] OSA, T. et al. An Algorithmic Perspective on Imitation Learning. **Found. Trends Robotics**, v. 7, p. 1–179, 2018.

[9] BISHOP, J. M. Artificial Intelligence Is Stupid and Causal Reasoning Will Not Fix It. **Frontiers in Psychology**, v. 11, p. 513474, 2020.

[10] BISHOP, J. Artificial Intelligence Is Stupid and Causal Reasoning Will Not Fix It. **Frontiers in Psychology**, v. 11, 2020.

[11] LU, H. et al. **Brain Intelligence: Go Beyond Artificial Intelligence**., 2017. Disponível em: <https://arxiv.org/abs/1706.01040>

[12] KOSMYNA, N. et al. **Your Brain on ChatGPT: Accumulation of Cognitive Debt when Using an AI Assistant for Essay Writing Task**., 2025. Disponível em: <https://arxiv.org/abs/2506.08872>

[13] HSU, T.-C.; CHANG, Y.-S.; CHEN, S.-Y. Teaching AI with games: the impact of generative AI drawing on computational thinking skills. **Education and Information Technologies**, 2025.

[14] TEDRE, M.; DENNING, P. J. **The Long Quest for Computational Thinking**. Proceedings of the 16th Koli Calling Conference on Computing Education Research. **Anais**...Koli, Finland: nov. 2016.

[15] CHUI, M. et al. **The state of AI in 2022—and a half decade in review**. [s.l.] McKinsey & Company, 2022. Disponível em: <https://www.mckinsey.com/~/media/mckinsey/business%20functions/quantumblack/our%20insights/the%20state%20of%20ai%20in%202022%20and%20a%20half%20decade%20in%20review/the-state-of-ai-in-2022-and-a-half-decade-in-review.pdf>. Acesso em: 10 jul. 2025.

[16] SWELLER, J. Cognitive load theory. **Psychology of Learning and Motivation**, v. 55, p. 37–76, 2011.

[17] SCHRAW, G.; MOSHMAN, D. Metacognitive theories. **Educational Psychology Review**, v. 7, n. 4, p. 351–371, 1995.

[18] ROHRER, D.; TAYLOR, K. The shuffling of mathematics problems improves learning. **Instructional Science**, v. 35, n. 6, p. 481–498, 2007.

[19] ROEDIGER, H. L.; BUTLER, A. C. The critical role of retrieval practice in long-term retention. **Trends in Cognitive Sciences**, v. 15, n. 1, p. 20–27, 2011.

[20] CAMBRIDGE UNIVERSITY. **Cambridge Mathematics**., 2025.

[21] AI INDEX STEERING COMMITTEE. **2025 AI Index Report**. Stanford, CA: Stanford Institute for Human-Centered Artificial Intelligence, 2025. Disponível em: <https://hai.stanford.edu/ai-index/2025-ai-index-report>. Acesso em: 10 jul. 2025.

[22] LEINONEN, J.; HELLAS, A.; IHANTOLA, P. **How Can "Vibe-Coding" Transform Programming Education?** Communications of the ACM, maio 2024. Disponível em: <https://cacm.acm.org/blogcacm/how-can-vibe-coding-transform-programming-education/>

[23] BRASIL. **Síntese de área: Ciência da Computação (Bacharelado/Licenciatura)**. Brasília, DF: Ministério da Educação, Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira (Inep), Diretoria de Avaliação da Educação Superior (Daes), 2021. Disponível em: <https://download.inep.gov.br/educacao_superior/enade/relatorio_sintese/2021/Enade_2021_Relatorios_Sintese_Area_Ciencia_Computacao.pdf>. Acesso em: 10 jul. 2025.

[24] BRASIL. **Relatório Síntese de Área: Engenharia de Computação**. Brasília, DF: Ministério da Educação, Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira (Inep), Diretoria de Avaliação da Educação Superior (Daes), 2023. Disponível em: <https://download.inep.gov.br/educacao_superior/enade/relatorio_sintese/2023/engenharia_de_computacao.pdf>. Acesso em: 10 jul. 2025.

[25] KRAMER, J. Is Abstraction the Key to Computing? **Communications of the ACM**, v. 50, n. 4, p. 36–42, 2007.

[26] WINSLOW, L. E. Programming pedagogy – a psychological overview. **ACM SIGCSE Bulletin**, v. 28, n. 3, p. 17–22, 1996.

[27] MIROLO, C. et al. Abstraction in Computer Science Education: An Overview. **Informatics in Education**, v. 20, n. 4, p. 615–639, 2021.

[28] WING, J. M. Computational thinking. **Communications of the ACM**, v. 49, n. 3, p. 33–35, mar. 2006.

[29] LEHMANN, T. H. Using algorithmic thinking to design algorithms: The case of critical path analysis. **The Journal of Mathematical Behavior**, v. 71, p. 101079, 2023.

[30] LITHNER, J. A research framework for algorithmic and creative reasoning. **Educational Studies in Mathematics**, v. 67, n. 3, p. 255–276, 2008.

[31] HARISMAN, Y. et al. Exploring Students' Mathematical Reasoning Behavior. **Education Sciences**, v. 13, 2023.

[32] HURRELL, D. P. Conceptual knowledge OR Procedural knowledge OR Conceptual knowledge AND Procedural knowledge: why the conjunction is important for teachers. **Australian Journal of Teacher Education**, v. 46, n. 2, p. art. 4, 2021.

[33] NATIONAL COUNCIL OF TEACHERS OF MATHEMATICS – NCTM. **Principles and Standards for School Mathematics**. Reston, VA: NCTM, 2000.

[34] KALDEWAIJ, A. **Programming: The Derivation of Algorithms**. [s.l.] Prentice Hall, 1990.

[35] GIBBONS, J. **Algorithm Design with Haskell**. University of Oxford, 2020. Acesso em: 10 jul. 2025

[36] BIRD, R.; DE MOOR, O. **Algebra of Programming**. [s.l.] Prentice Hall, 1997.

[37] GÜNDOĞDU, F. et al. Exploring mathematical reasoning skills. **ScienceDirect**, 2023.

[38] SILVA QUINTO, W. A. et al. Explorando o impacto da Inteligência Artificial na formação do pensamento crítico entre acadêmicos de T.I. na Região Norte do Brasil. **Caderno Pedagógico**, v. 22, n. 7, 2025.

[39] TSAI, C.-Y.; YANG, Y.-F. The impact of unplugged activities on developing computational thinking skills in elementary school students. **Journal of Educational Technology & Society**, v. 22, n. 3, p. 77–89, 2019.

[40] POLAT, E.; YILMAZ, R. M. Unplugged versus plugged-in: examining basic programming achievement and computational thinking of 6th-grade students. **Education and Information Technologies**, v. 27, p. 9145–9179, 2022.

[41] BRACKMANN, C. P. et al. Unplugged versus plugged-in: examining basic programming achievement and computational thinking of 6th-grade students. **Journal of Computer Science Education**, 2022.

[42] GIBBONS, J. Functional Algorithm Design, Part 0. **SIGPLAN Blog**, 2020.

[43] ARA'UJO, A. L. S. DE O.; ANDRADE, W. L. DE; GUERRERO, D. D. S. **Um mapeamento sistem'atico sobre a avaliaç ao do pensamento computacional no Brasil**. Anais do V Congresso Brasileiro de Inform'atica na Educaç ao (CBIE). **Anais**...2016.

[44] PAPERT, S. **Logo: Computadores e Educação**. São Paulo: Brasiliense, 1985.

[45] SAIDIN, N. D. et al. Benefits and challenges of applying computational thinking in education. **International Journal of Information and Education Technology**, v. 11, n. 5, p. 248–254, 2021.

[46] COMPUTER SCIENCE TEACHERS ASSOCIATION. **Computational Thinking: A Definition for K-12**., 2011. Disponível em: <https://csteachers.org/teaching-computational-thinking-in-early-elementary/>. Acesso em: 7 jul. 2025

[47] CURZON, P. et al. Computational Thinking. Em: FINCHER, S. A.; ROBINS, A. V. (Eds.). **The Cambridge Handbook of Computing Education Research**. Cambridge: Cambridge University Press, 2019. p. 513–546.

[48] LEHMANN, T. H. How current perspectives on algorithmic thinking can be applied to students' engagement in algorithmatizing tasks. **Mathematics Education Research Journal**, v. 36, n. 3, p. 609–643, 2024.

[49] HIJÓN-NEIRA, R. et al. Computational Thinking Measurement of CS University Students in the AI Era. **Preprints.org**, maio 2024.

[50] RIBEIRO, L. et al. Entendendo o Pensamento Computacional: além da programação. **Revista Brasileira de Informática na Educação**, v. 25, n. 3, p. 45–62, 2017.

[51] KONG, S. et al. **Pensamento Computacional na Educação: perspectivas internacionais**. São Paulo: Penso, 2020.

[52] MEDEIROS, W. M. **Pensamento Computacional ou Programação? Uma análise de práticas pedagógicas com Scratch**. Dissertação (Mestrado em Educação)—Uberlândia: UFU, 2024.

[53] SCRATCH FOUNDATION. **Scratch.**, [s.d.][s.d.]. Disponível em: <https://scratch.mit.edu/>. Acesso em: 7 jul. 2025

[54] SBC (SOCIEDADE BRASILEIRA DE COMPUTAÇÃO). **Referenciais de Formação em Computação: Educação Básica**. Porto AlegreSBC, 2017.

[55] HORA, N. DA. **O ensino do pensamento computacional no Brasil na era digital**. Futura, 9 fev. 2022. Disponível em: <https://futura.frm.org.br/conteudo/professores/artigo/o-ensino-do-pensamento-computacional-no-brasil-na-era-digital>. Acesso em: 9 jul. 2025

[56] FELIPUSSI, A. L.; PADUA, C. C. S. **Relato de aulas com robô programável e Pensamento Computacional**. Anais do 12º Congresso Brasileiro de Informática na Educação. **Anais**...Recife: SBC, 2023.

[57] FUTURA, C. P. **O ensino do pensamento computacional no Brasil na era digital**. São Paulo: Fundação Roberto Marinho, 2023.

[58] KONG, S.-C. Learning Composite and Prime Numbers Through Developing an App: An Example of Computational Thinking Development Through Primary Mathematics Learning. Em: KONG, S.-C.; ABELSON, H. (Eds.). **Computational Thinking Education**. Singapore: Springer, 2019. p. 155–177.

[59] KIRWAN, C.; COSTELLO, E.; DONLON, E. ADAPTTER: Developing a Framework for Teaching Computational Thinking in Second-Level Schools by Design Research. **TechTrends**, v. 66, n. 4, p. 495–509, 2022.

[60] HOARE, C. A. R. An Axiomatic Basis for Computer Programming. **Communications of the ACM**, v. 12, n. 10, p. 576–580, 1969.

[61] LEVESON, N. G. **Engineering a Safer World: Systems Thinking Applied to Safety**. Cambridge, MA: MIT Press, 2012.

[62] CURZON, P.; MCOWAN, P.; BLACKWELL, A. Unplugged approaches to the teaching of computing to adults. **Journal of Computing Sciences in Colleges**, v. 29, n. 4, p. 90–97, 2014.

[63] WEINTROP, D.; WILENSKY, U. **Using Unplugged Activities in Adult Programming Education**. Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15). **Anais**...New York, NY, USA: ACM, 2015.

[64] BERGMANN, J.; SAMS, A. **Flip your classroom: reach every student in every class every day**. Eugene: ISTE, 2012.

[65] ETH ZURICH. **Flipped Classroom**. https://ethz.ch/en/the-eth-zurich/education/educational-development/planning-teaching/flipped-classroom.html, 2025. Acesso em: 15 jul. 2025

[66] LEIFER, L. A. **Project Based Learning in Design Education**. Proceedings of the 1970 Conference on Design Methods. **Anais**...1970.

[67] KIM, Y. Computational Thinking. Em: **Educational Technology: An Online Handbook for Pre-K-12**. [s.l.] EdTech Books, 2021.

[68] BELL, T. et al. Computer Science Unplugged: school students doing real computing without computers. **Article**, 2009.

[69] NEW YORK HALL OF SCIENCE. **Computational Thinking School Strategy Guide**., [s.d.][s.d.]. Disponível em: <https://nysci.org/pdf/NYSCI-Computational-Thinking-School-Strategy-Guide-04-09-2024.pdf>. Acesso em: 6 jul. 2025

[70] WONG, M. M. Y. et al. Self-development Through Service-Oriented Stress-Adaption-Growth (SOSAG) Process in the Engagement of Computational Thinking Co-teaching Education. Em: KONG, S.-C.; ABELSON, H. (Eds.). **Computational Thinking Education**. Singapore: Springer Nature Singapore Pte Ltd., 2019.

[71] PYTHON SOFTWARE FOUNDATION. **The Python Language Reference**. Disponível em: <https://docs.python.org/3/reference/index.html>. Acesso em: 13 jul. 2025.

[72] PENN STATE UNIVERSITY. **Computer Science (CMPSC) - University Bulletin - Penn State**., [s.d.]a[s.d.]a. Disponível em: <https://bulletins.psu.edu/university-course-descriptions/undergraduate/cmpsc/>. Acesso em: 6 jul. 2025

[73] LEROY, X. et al. **The OCaml system: Documentation and user's manual**. [s.l.] INRIA (Institut National de Recherche en Informatique et en Automatique), 2025.

[74] GOSLING, J. et al. **The Java® Language Specification, Java SE 22 Edition**. Redwood City, California: Oracle America, Inc., 2024.

[75] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **Information technology – Programming languages – C**. Geneva, Switzerland: International Organization for Standardization (ISO), 2018. Disponível em: <https://www.iso.org/standard/74528.html>.

[76] 21, I. J. 1/SC 22/WG. **ISO/IEC 14882:2024 - Programming languages — C++**. [s.l.] International Organization for Standardization, 2024. Disponível em: <https://www.iso.org/standard/82312.html>.

[77] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO/IEC 13211-1:1995 - Information technology – Programming languages – Prolog – Part 1: General core**. [s.l.] ISO/IEC, 1995.

[78] STANFORD UNIVERSITY. **CS106A Syllabus**., 2025. Disponível em: <https://web.stanford.edu/class/archive/cs/cs106a/cs106a.1258/syllabus>

[79] ŽIVKOVIĆ, M. **XLogo4Schools**. SourceForge, 2014. Disponível em: <http://sourceforge.net/projects/xlogo4schools>

[80] WATTENHOFER, R. **Computational Thinking**. [s.l.] ETH Zürich, 2020.

[81] AMENDUM, M. L. B. et al. **Computational Thinking Rubric for Problem Solving**. Newark, DE: The College School, University of Delaware, 2018. Disponível em: <https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/4/8672/files/2018/12/Computational-Thinking-Rubric-2ktkkgv.pdf>.

[82] ROMÁN-GONZÁLEZ, M.; MORENO-LEÓN, J.; ROBLES, G. Combining Assessment Tools for a Comprehensive Evaluation of Computational Thinking Interventions. Em: KONG, S.-C.; ABELSON, H. (Eds.). **Computational Thinking Education**. [s.l.] Springer, 2019. p. 85–100.

[83] SMITH, J. A. C. B.; GROVER, S.; SMITH, A. L. **Measuring Computational Thinking and Computer Science Outcomes: A Resource for CS for All Researchers**. [s.l.] U.S. Department of Education, mar. 2021. Disponível em: <https://www.ed.gov/sites/ed/files/2021/03/CSCTOutcomes_508.pdf>.

[84] RASPBERRY PI FOUNDATION. **The UK Bebras Challenge**. https://www.bebras.uk/, 2025.

[85] NATIONAL CENTER FOR EDUCATION STATISTICS (NCES). **Glossary: Credit Hour**. Washington, D.C.U.S. Department of Education, 2010. Disponível em: <https://www.naicu.edu/media/l3ckll0r/20101214_credithourbkgrnd12-13-10.pdf>. Acesso em: 7 jul. 2025

[86] EUROPEAN COMMISSION. **ECTS Users' Guide**. LuxembourgPublications Office of the European Union, 2015. Disponível em: <https://education.ec.europa.eu/sites/default/files/document-library-docs/ects-users-guide_en.pdf>. Acesso em: 7 jul. 2025

[87] XIAMEN UNIVERSITY. **Overseas Education College. Credits**. XiamenOEC, [s.d.][s.d.]. Disponível em: <https://oec.xmu.edu.cn/en/Study/Credits.htm>. Acesso em: 8 jul. 2025

[88] BRASIL. **Lei nº 9.394, de 20 de dezembro de 1996. Estabelece as diretrizes e bases da educação nacional**., 1996. Disponível em: <https://www.planalto.gov.br/ccivil_03/leis/l9394.htm>. Acesso em: 11 jul. 2025

[89] BRASIL. **Resolução nº 2, de 18 de junho de 2007. Dispõe sobre carga horária mínima e procedimentos relativos à integralização e duração dos cursos de graduação, bacharelados, na modalidade presencial**. **Diário Oficial da União**Brasília, DF, 19 jun. 2007. Disponível em: <http://portal.mec.gov.br/cne/arquivos/pdf/2007/rces002_07.pdf>. Acesso em: 11 jul. 2025

[90] BRASIL. **Parecer CNE/CES nº 261, de 9 de novembro de 2006. Solicita pronunciamento desta Câmara sobre a duração do curso de Bacharelado em Educação Física e sua carga horária**. **Diário Oficial da União**Brasília, DF, 18 jan. 2007. Disponível em: <http://portal.mec.gov.br/cne/arquivos/pdf/2006/pces261_06.pdf>. Acesso em: 11 jul. 2025

[91] FRAUCHES, C. **Educação Superior Comentada: Políticas, diretrizes, legislação e normas do ensino superior**. ABMES, Colunas, 4 jul. 2011. Disponível em: <https://www.abmes.org.br/colunas/detalhe/255/educacao-superior-comentada-politicas-diretrizes-legislacao-e-normas-do-ensino-superior>. Acesso em: 11 jul. 2025

[92] CARNEGIE MELLON CENTER OF COMPUTATIONAL THINKING. **What is Computational Thinking**., [s.d.][s.d.]. Disponível em: <https://www.cs.cmu.edu/~CompThink/>. Acesso em: 6 jul. 2025

[93] PROJECT OLYMPUS, CARNEGIE MELLON UNIVERSITY. **PROBE Projects**. https://www.cmu.edu/project-olympus/probe-projects/index.html, [s.d.].

[94] OPEN LEARNING INITIATIVE, CARNEGIE MELLON UNIVERSITY. **Principles of Computation with Python — Open & Free**. https://oli.cmu.edu/courses/principles-of-computation-with-python-open-free/, [s.d.].

[95] UNIVERSITY OF CAMBRIDGE. **Department of Computer Science and Technology. Algorithms 1**., [s.d.]a[s.d.]a. Disponível em: <https://www.cl.cam.ac.uk/teaching/2324/Algorithm1/>. Acesso em: 12 jul. 2025

[96] VEX ROBOTICS. **VEX Robotics Official Website**. https://www.vexrobotics.com/, 2025.

[97] CARNEGIE MELLON ROBOTICS ACADEMY. **Coding and Computational Thinking with VEX V5**., [s.d.][s.d.]. Disponível em: <https://www.cmu.edu/roboticsacademy/roboticscurriculum/VEX%20Curriculum/coding_v5.html>. Acesso em: 6 jul. 2025

[98] DENNING, P. J.; TEDRE, M. **Computational Thinking**. Cambridge, MA: The MIT Press, 2019.

[99] MASSACHUSETTS INSTITUTE OF TECHNOLOGY. **A computational thinking requirement for MIT undergraduates: Report of the working group on computational thinking**., 2017. Disponível em: <https://facultygovernance.mit.edu/sites/default/files/reports/2017-01_computational_thinking_requirement_FINAL_CLEAN.pdf>. Acesso em: 11 jul. 2025

[100] MASSACHUSETTS INSTITUTE OF TECHNOLOGY. **General Institute Requirements**. **MIT Course Catalog**, [s.d.]. Disponível em: <https://catalog.mit.edu/mit/undergraduate-education/general-institute-requirements/>. Acesso em: 11 jul. 2025

[101] HARVARD UNIVERSITY. **CS50: Introduction to Computer Science**., 2025. Disponível em: <https://pll.harvard.edu/course/cs50-introduction-computer-science>

[102] HARVARD UNIVERSITY. **Syllabus - CS50: Computer Science Courses and Programs from Harvard**., 2025. Disponível em: <https://cs50.harvard.edu/syllabus>

[103] STANFORD UNIVERSITY. **CS106B Syllabus**., 2025. Disponível em: <https://web.stanford.edu/class/cs106b/syllabus>

[104] FREDOVERFLOW. **Karel The Robot**. Disponível em: <https://github.com/fredoverflow/karel>. Acesso em: 13 jul. 2025.

[105] STANFORD UNIVERSITY. **Programming Methodology - Stanford Engineering Everywhere | CS106A**., 2025. Disponível em: <https://see.stanford.edu/course/cs106a>

[106] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **Information technology – Programming languages – C++**. Geneva, Switzerland: International Organization for Standardization (ISO), 2020. Disponível em: <https://isocpp.org/std/the-standard>.

[107] RYCROFT-SMITH, L.; CONNOLLY, C. Comparing conceptions of mathematical and computational thinking cycles. **Cambridge Espresso**, 2019.

[108] GOULD, T.; RYCROFT-SMITH, L. Establishing concepts of ratio. **Instant, Cambridge Mathematics**, 2022.

[109] DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY. **Course pages 2023-24**. University of Cambridge; https://www.cl.cam.ac.uk/teaching/2324/, 2024.

[110] DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY. **Algorithms 1**. University of Cambridge; https://www.cl.cam.ac.uk/teaching/2324/Algorithm1/, 2024.

[111] UNIVERSITY OF CAMBRIDGE. **Faculty of Education. Computational Thinking Challenge**., [s.d.]b[s.d.]b. Disponível em: <https://www.educ.cam.ac.uk/research/programmes/computationalthinking/>. Acesso em: 12 jul. 2025

[112] DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY. **Algorithms 2**. University of Cambridge; https://www.cl.cam.ac.uk/teaching/2324/Algorithm2/, 2024.

[113] DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF OXFORD. **Computer Science**. University of Oxford; https://www.cs.ox.ac.uk/teaching/bacompsci/, 2024.

[114] DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF OXFORD. **Mathematics and Computer Science**. University of Oxford; https://www.cs.ox.ac.uk/teaching/mcs/, 2024.

[115] UNIVERSITY OF OXFORD. **UK Bebras participants in the Oxford University Computing Challenge**., [s.d.]d[s.d.]d. Disponível em: <https://www.raspberrypi.org/blog/uk-bebras-oxford-university-computing-challenge-2022/>. Acesso em: 12 jul. 2025

[116] UNIVERSITY OF OXFORD. **Oxford University Computing Challenge and related programs**., [s.d.]b[s.d.]b. Disponível em: <https://www.raspberrypi.org/blog/uk-bebras-oxford-university-computing-challenge-2022/>. Acesso em: 12 jul. 2025

[117] UKCT CHALLENGES. **About Us**. UKCT Challenges; https://ukctchallenges.org/about/, 2024.

[118] IMPERIAL COLLEGE LONDON. **Programme Specification 2024-25: MEng Computing**. https://www.imperial.ac.uk/media/imperial-college/study/programme-specifications/computing/24x2f25/G401-MEng-Computing-2024-25.pdf, 2024.

[119] IMPERIAL COLLEGE LONDON. **I-X – Reimagining the university in an age of rapid innovation**. https://www.imperial.ac.uk/stories/ix-rapid-innovation/, 2020.

[120] CARDIFF SCHOOL OF COMPUTER SCIENCE AND INFORMATICS. **CM1101: Computational Thinking**. https://data.cardiff.ac.uk/legacy/grails/module/CM1101.html, 2025.

[121] CARDIFF SCHOOL OF COMPUTER SCIENCE AND INFORMATICS. **CM1103: Problem Solving with Python**. https://data.cardiff.ac.uk/legacy/grails/module/CM1103/24A.html, 2025.

[122] EUROPEAN COMMISSION. **Digital Education Action Plan 2021-2027**., 2021. Disponível em: <https://education.ec.europa.eu/focus-topics/digital-education/action-plan>. Acesso em: 6 jul. 2025

[123] ETH ZÜRICH. **Computational Thinking**., 2020. Disponível em: <https://disco.ethz.ch/courses/hs20/coti/lecturenotes/script.pdf>. Acesso em: 6 jul. 2025

[124] ETH ZURICH. **Programming for beginners**. https://ethz.ch/en/studies/bachelor/beginning-your-studies/subject-related-preparation/programming-beginners.html, 2025. Acesso em: 15 jul. 2025

[125] EHL HOSPITALITY BUSINESS SCHOOL. **Course Catalogue: Undergraduate Programs**. Lausanne, Chur-Passugg, Singapore: EHL Hospitality Business School, 2023-2024.

[126] UNIVERSITY OF YORK. **Computational Thinking: Foundations and Fundamentals**. YorkDepartment of Computer Science, 2020.

[127] SENTANCE, S. et al. **Creating cool stuff: Pupils' experience of the BBC micro:bit**. Proceedings of the ACM SIGCSE Technical Symposium. **Anais**...Seattle: ACM, 2017.

[128] KURKOVSKY, S. Mobile computing and robotics in one course: Teaching design patterns. **Journal of Computing Sciences in Colleges**, v. 28, n. 6, p. 67–73, 2013.

[129] BARAK, B. **Chapter 15: NP, NP completeness, and the Cook-Levin Theorem**. https://introtcs.org/public/ch15_nptime.html, 2023.

[130] MURRAY, R. M.; LI, Z.; SASTRY, S. S. **A Mathematical Introduction to Robotic Manipulation**. [s.l.] CRC Press, 1994.

[131] GÉRON, A. **Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems**. 2nd. ed. [s.l.] O'Reilly Media, Inc., 2019.

[132] HERLIHY, M.; SHAVIT, N. **The Art of Multiprocessor Programming**. Revised Reprint ed. Waltham, MA, USA: Morgan Kaufmann Publishers Inc., 2012.

[133] TANENBAUM, A. S.; BOS, H. **Modern Operating Systems**. 4th. ed. USA: Prentice Hall Press, 2015.

[134] NETFLIX, INC. **Chaos Engineering**., 2021. Disponível em: <https://netflix.github.io/chaosmonkey/>. Acesso em: 10 jul. 2025

[135] MILLER, G. A. The magical number seven, plus or minus two: Some limits on our capacity for processing information. **Psychological Review**, v. 63, n. 2, p. 81–97, 1956.

[136] FRIGG, R. **Models and representation: Why structures are not enough**. London: London School of Economics, 2002. Disponível em: <https://romanfrigg.org/wp-content/uploads/writings/Models_and_Representation.pdf>.

[137] MUGGLETON, S. Inductive logic programming: Theory and methods. **Journal of Logic Programming**, v. 19, n. 20, p. 629–679, 1991.

[138] GROVER, S.; PEA, R. Computational thinking in K–12: A review of the state of the field. **Educational Researcher**, v. 42, n. 1, p. 38–43, 2013.

[139] ALCANTARA, F. C. DE. **Fluxograma Interativo**. https://frankalcantara.com/fluxograma/index.html, [s.d.].

[140] BRENNAN, K.; RESNICK, M. **New frameworks for studying and assessing the development of computational thinking**. Proceedings of the 2012 Annual Meeting of the American Educational Research Association. **Anais**...Vancouver, Canada: 2012.

[141] PRITCHARD, T. A. **Using flowcharts, code and animation for improved comprehension and ability in novice programming**. tese de doutorado—[s.l.] University of South Wales, 2018.

[142] ALI, F. **Effect of Flowcharts on Code Comprehension of Novice Programmers**. mathesis—[s.l.] Chemnitz University of Technology, 2022.

[143] SOCIEDADE BRASILEIRA DE COMPUTAÇÃO. **Nota Técnica da Sociedade Brasileira de Computação sobre a BNCC-EF e a BNCC-EM**. Porto Alegre, RS, Brasil: Sociedade Brasileira de Computação (SBC); https://www.sbc.org.br/wp-content/uploads/2024/07/Nota-t-cnica-sobre-a-BNCC-Ensino-m-dio-e-fundamental-2018.pdf, 2018.

[144] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **Information processing – Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts**. [s.l.] International Organization for Standardization, 1985. Disponível em: <https://cdn.standards.iteh.ai/samples/11955/1b7dd254a2a54fd7a89d616dc0570e18/ISO-5807-1985.pdf>.

[145] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **Information processing – Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts**. Geneva, CHInternational Organization for Standardization, 1985.

[146] ANDRZEJEWSKA, M.; STOLIŃSKA, A. Do Structured Flowcharts Outperform Pseudocode? Evidence From Eye Movements. **IEEE Access**, v. 10, p. 132965–132975, 2022.

[147] SCANLAN, D. A. Structured Flowcharts Outperform Pseudocode: An Experimental Comparison. **IEEE Software**, v. 6, n. 5, p. 28–36, 1989.

[148] THREEKUNPRAPA, A.; YASRI, P. Patterns of Computational Thinking Development while Solving Unplugged Coding Activities Coupled with the 3S Approach for Self-Directed Learning. **European Journal of Educational Research**, v. 9, n. 3, p. 1025–1045, 2020.

[149] SIOZOU, S.; TSELIOS, N.; KOMIS, V. **Effect of algorithms' multiple representations in the context of programming education**. Proceedings of the 4th Pan-Hellenic Conference "Informatics and Education". **Anais**...Patras, Greece: 2008. Disponível em: <https://www.researchgate.net/publication/220373268_Effect_of_algorithms'_multiple_representations_in_the_context_of_programming_education>

[150] THE KNOWLEDGE ACADEMY. **Differences Between Flowchart and Pseudocode: A Detailed Comparison**., 2025. Disponível em: <https://www.theknowledgeacademy.com/blog/flowchart-vs-pseudocode/>

[151] KELLEHER, C. D.; PAUSCH, R. Exploring the role of visualization and engagement in computer science education. **ACM SIGCSE Bulletin**, v. 39, n. 3, p. 213–217, 2007.

[152] MEN, N. et al. **Hybrid Beamforming for RIS-Aided Multiuser Millimeter Wave Systems**. 2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring). **Anais**...2022.

[153] CHI, M. T. H. et al. Eliciting self-explanations improves understanding. **Cognitive Science**, v. 18, n. 3, p. 439–477, 1994.

[154] CEPEDA, N. J. et al. Distributed practice in verbal recall tasks: A review and quantitative synthesis. **Psychological Bulletin**, v. 132, n. 3, p. 354–380, 2006.

[155] THE GIT PROJECT. **Git**. https://git-scm.com/, 2025.

[156] GITHUB, INC. **GitHub: Where the world builds software**. urlhttps://github.com/, 2025.

[157] **GitHub Classroom**. https://classroom.github.com/, [s.d.].