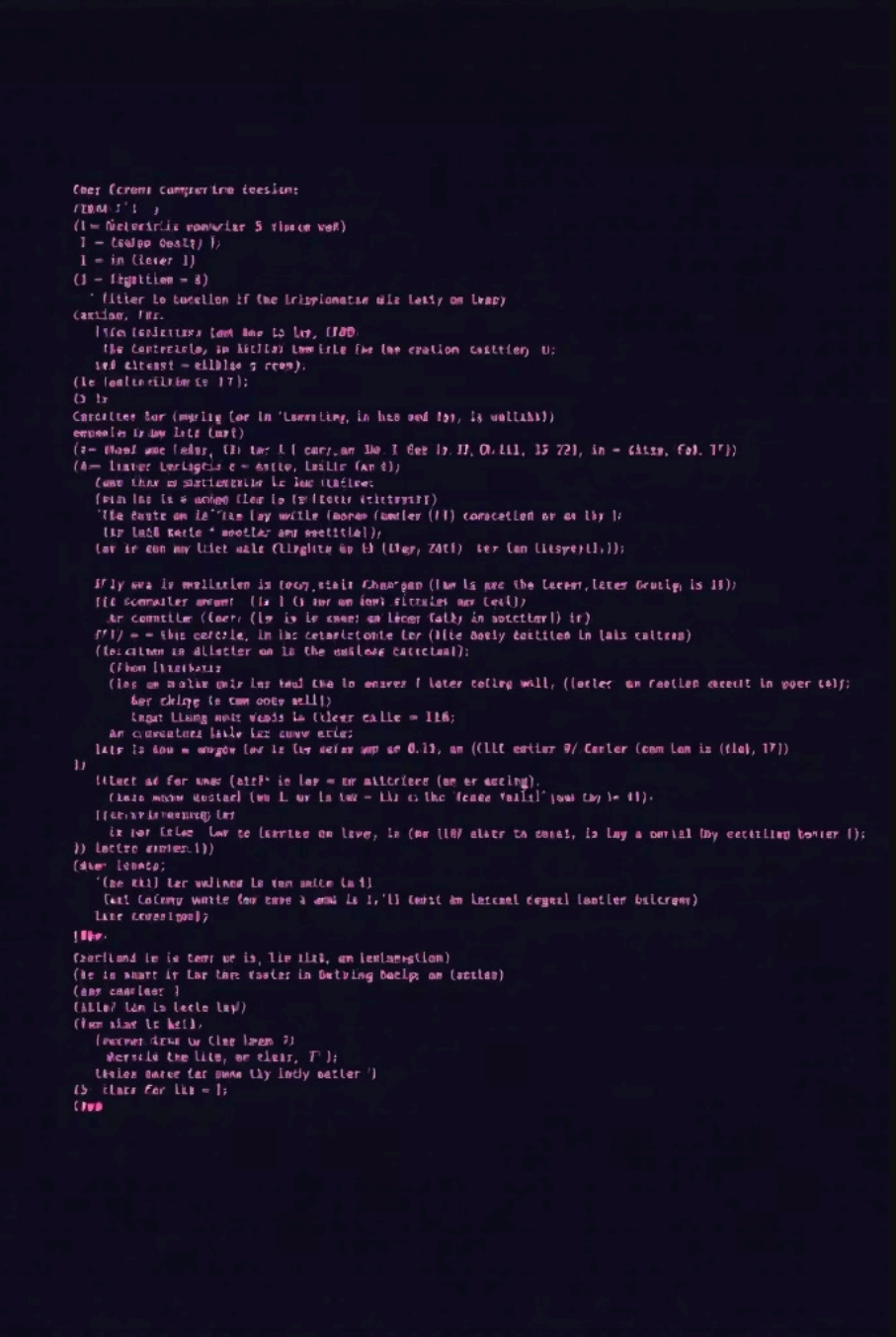
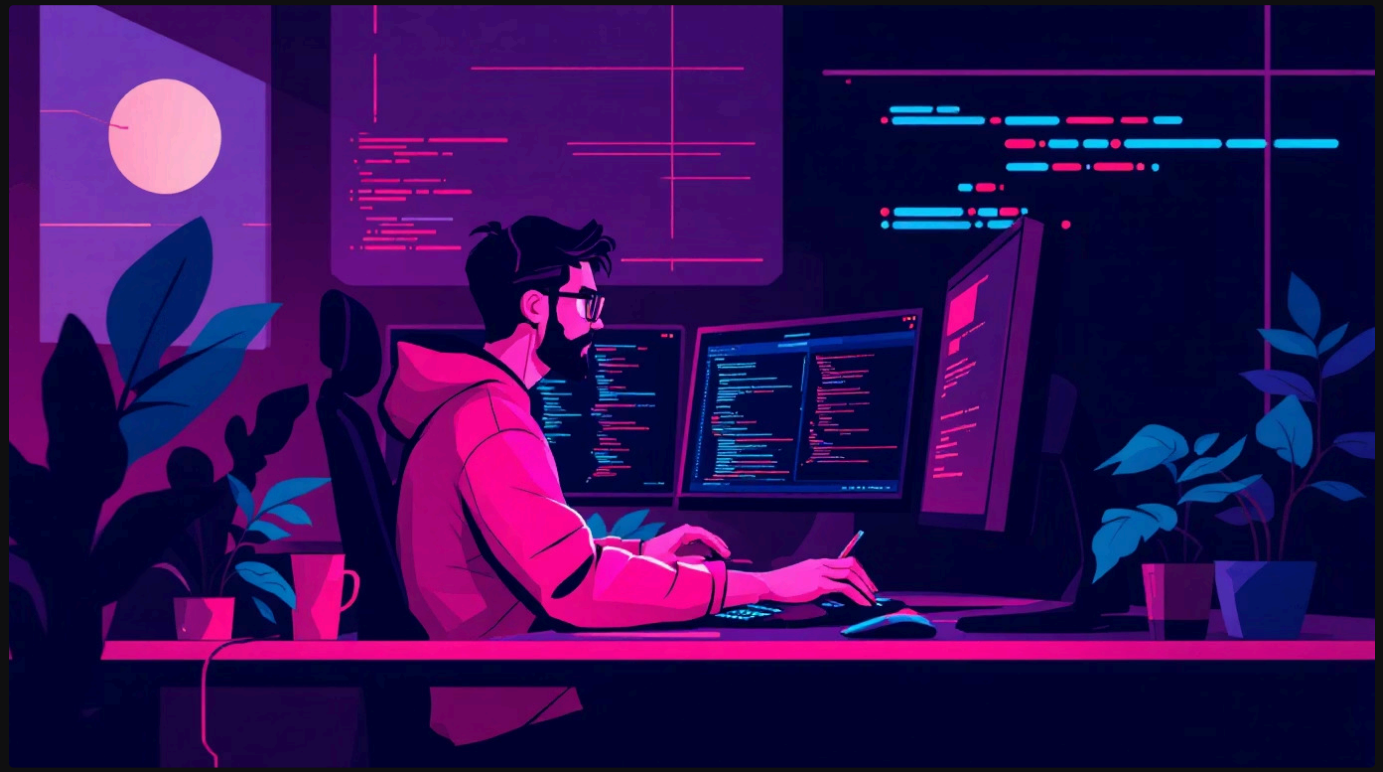


Writing Correct Programs

Column 4 — Programming Pearls

Como escrever código correto desde o início usando invariantes e pensamento lógico. Uma jornada pelo capítulo que mudou a forma como programadores pensam sobre correção.

Programação Corretiva ≠ Apenas Testar



O Problema

Binary Search — um algoritmo aparentemente simples que gerou erros por décadas.

A maioria dos programadores profissionais falha ao implementá-lo corretamente.

O capítulo de Bentley mostra: programar bem exige **pensamento lógico estruturado**, não apenas testes.

O Desafio da Busca Binária

O objetivo é encontrar um elemento **t** em um vetor ordenado ou retornar **-1** se não existir.

1

Vetor Ordenado

[1, 3, 5, 7, 9, 11, 15]

2

Reduzir Intervalo

Eliminar metade a cada passo

3

Casos Extremos

Aqui estão os perigos

O perigo está nos detalhes: limites errados, loops infinitos, array out of bounds.

O Conceito-Chave: Invariante

O que é um Invariante?

> "Se **t** existir, ele está dentro do intervalo **[l, u]**."

Um invariante é **algo que permanece verdadeiro** durante cada iteração do loop.

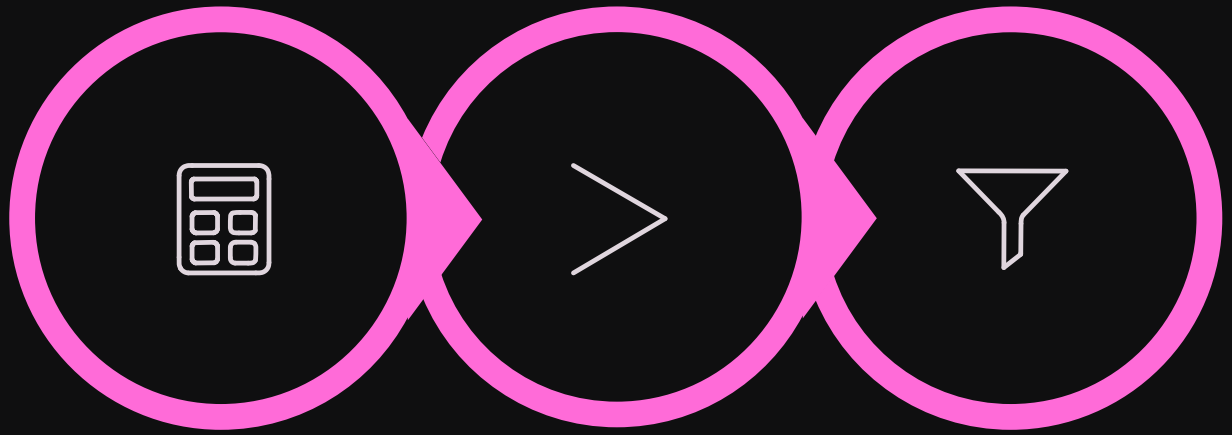
Analogia: como um elevador que *nunca* sai do prédio — ele pode subir ou descer, mas permanece dentro do edifício.

A lógica do código é construída para preservar esse invariante.

Visual: Intervalo Reduzindo

Cada iteração **reduz** o intervalo, mas mantém a **propriedade fundamental**.

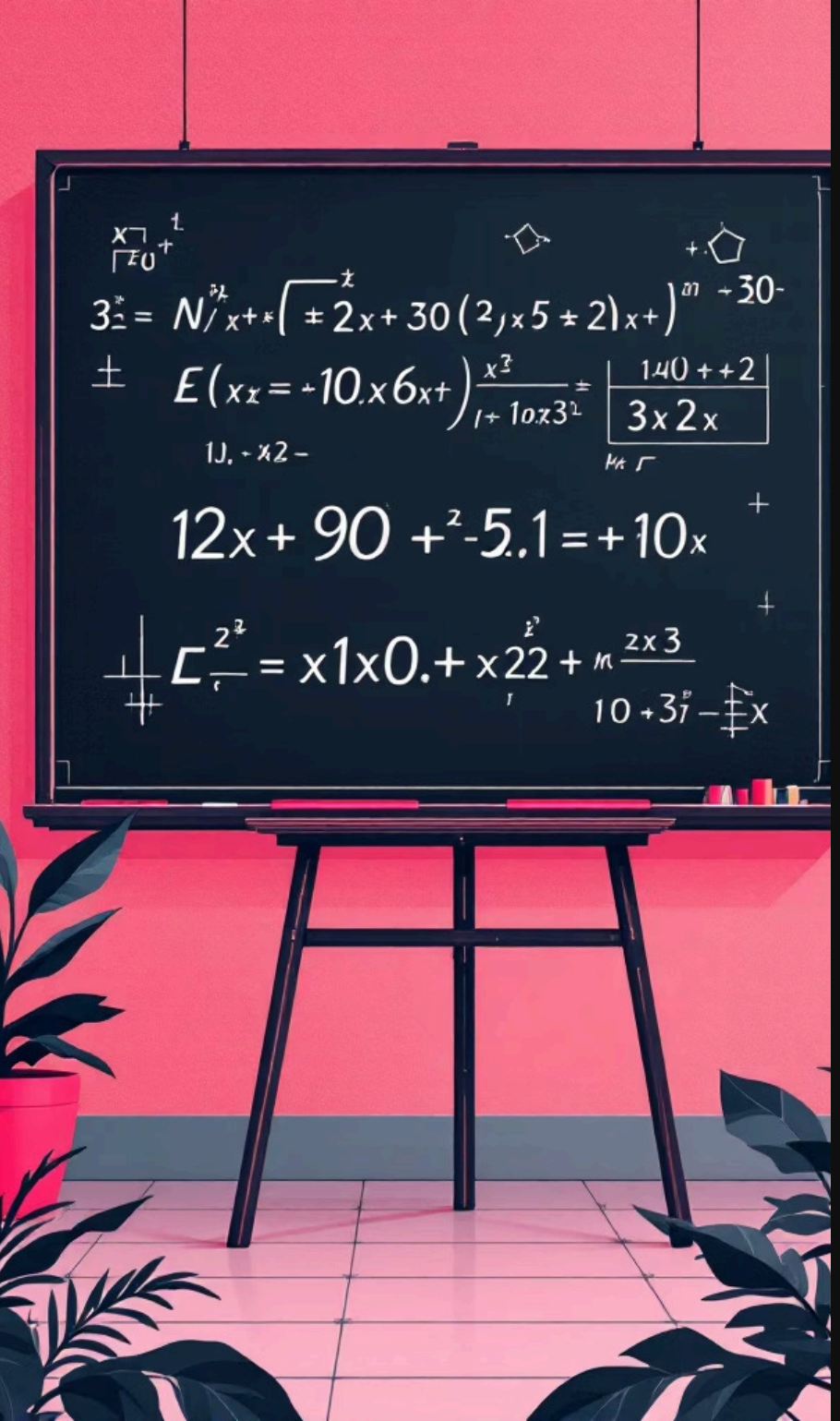
Construindo o Algoritmo Corretamente



Calcular m

Comparar

Reduzir intervalo



3 Etapas para Provar Correção

Initialization

O invariante começa verdadeiro: no início, **l = 0** e **u = n-1**, então o elemento (se existir) está no intervalo completo.

Preservation

Se o invariante vale antes da iteração, ele vale **também depois**. Cada comparação preserva a propriedade.

Termination

O intervalo **sempre diminui** (l aumenta ou u diminui). Eventualmente, **l > u** e o loop termina.

O código está correto se e somente se as 3 etapas são satisfeitas.

Principais Lições do Capítulo

Invariantes guiam design

Binary Search é só um exemplo — a ideia aplica-se a **qualquer** algoritmo que usa loops.

Pensar antes de codar

Programação correta nasce do raciocínio lógico estruturado, não de tentativa e erro.

Testes não provam correção

Testes mostram exemplos; invariantes explicam o **porquê**. Corretude não é sorte.

> "Código correto nasce do raciocínio, não da sorte."