Capítulo 3 – Variáveis, Dados e Estruturação Algorítmica Avançada

Carga horária: 4h (2h teoria + 2h prática)
Etapa: Construção Estruturada de Programas

Objetivo: Este capítulo aprofunda o entendimento sobre como um programa realmente funciona por dentro.

Até aqui, você aprendeu a organizar ideias e transformar passos em instruções.

A partir de agora, você passará a compreender **como o computador guarda informações, como elas são manipuladas e como os algoritmos passam a ter comportamento inteligente**.

3.1. Variáveis, Tipos de Dados e Entrada/Saída

3.1.1. O que realmente são dados? (entendimento profundo)

Em computação, tudo começa com dados, mas este conceito costuma ser mal interpretado pelos iniciantes.

Dados não representam conhecimento.

Eles são apenas *fragmentos*.

Exemplos de dados soltos:

- 18
- “Maria”
- 9.5
- VERDADEIRO

Por si só, eles **não dizem nada**.

É o programador quem decide *o que eles significam* e *como serão usados*.

✓ **Quando organizados corretamente, tornam-se informação:**

- 18 → “Maria tem 18 anos.”
- 9.5 → “Nota final do aluno.”
- VERDADEIRO → “Sensor detectou presença.”

✓ **Quando interpretados, tornam-se conhecimento:**

- “Maria tem 18 anos. Logo, pode fazer a prova de habilitação.”

Este encadeamento: dado → informação → conhecimento, é o núcleo da computação.

E para que isso aconteça, o programa precisa **guardar**, **alterar** e **ler** esses valores.

É aí que entram as variáveis.

3.1.2. Por que variáveis são essenciais? (explicação aprofundada)

Sem variáveis, um algoritmo seria apenas um texto estático que repete sempre o mesmo resultado.

Exemplo de programa **sem variáveis**:

```
Escreva("A soma é 10")
```

Isso não é um programa... é apenas uma frase.

Um programa de verdade precisa **lidar com dados variáveis**, isto é, informações que mudam conforme o usuário ou o ambiente.

As variáveis permitem exatamente isso. Elas fazem o algoritmo **guardar um valor hoje e outro valor amanhã**, sem precisar reescrever o código.

✓ **Variáveis permitem:**

- armazenar dados digitados pelo usuário
- guardar resultados de cálculos
- criar condições (“se... então...”)
- controlar repetições (“enquanto... faça...”)
- responder a eventos
- registrar estados (como sensores em um Arduino)

Sem variáveis, **não existe lógica**, não existe decisão, não existe repetição. A variável é o “átomo” da programação.

3.1.3. Como funciona a memória do computador, uma abordagem simples e poderosa

Para entender variáveis, é fundamental visualizar a memória como uma sequência enorme de pequenas gavetas numeradas.

Cada gaveta pode guardar um valor.

Quando o algoritmo declara uma variável, o computador separa uma dessas gavetas e coloca uma “etiqueta” nela.

Por exemplo:

idade : inteiro

O computador reserva uma gaveta e coloca a etiqueta **idade**. A partir de agora, sempre que o programa precisar daquela gaveta, basta usar esse nome.

✓ **A variável não é o valor**

A variável é **o local onde o valor fica guardado**.

Assim como uma gaveta não é a camiseta...
Um nome de variável não é o dado, é onde ele fica armazenado.

3.1.4. Tipos de Dados

Por que existem tipos diferentes?

Porque a informação não é toda igual.

Um nome não é manipulado como um número; uma nota não é entendida como verdadeiro/falso.

Cada tipo expressa **um formato de dado** e **um conjunto de ações possíveis**.

1. **inteiro -** Representa números sem vírgula.

Permite:

- somas
- subtrações
- contagens
- comparações (maior, menor, igual)

2. **real -** Representa números com casas decimais.

Permite:

- cálculos mais precisos
- médias
- medições

- porcentagens

3. **caractere -** Usado para textos, mesmo que curtos.

É fundamental porque:

- nomes são textos
- mensagens são textos
- códigos, números de série, placas de carros, também são textos

4. **lógico -**Usado para expressar decisões.

Dois valores possíveis:

- VERDADEIRO
- FALSO

Ele é a base de toda a tomada de decisão da programação.

Sem esse tipo, seria impossível responder perguntas como:

- “o aluno foi aprovado?”
- “o botão está pressionado?”
- “o valor digitado está correto?”

3.1.5. Declaração de variáveis

Declarar uma variável significa dizer ao computador:

“Reserve um espaço na memória e identifique-o com este nome.”

Isso organiza o programa, evita erros e permite que o Visualg valide o tipo antes de processar.

✓ **Três objetivos da declaração:**

1. **Reservar espaço físico na memória**
2. **Garantir que o tipo de dado está correto**
3. **Permitir que o algoritmo entenda o significado daquela variável**

Sem a declaração, o programa não sabe:

- quanto espaço reservar
- que tipo de dado esperar
- como manipular o valor

3.1.6. Atribuição de valores

O operador <- não é uma "setinha decorativa".

Ele tem um significado profundo:

"Recebe o valor"

Ele não compara, não soma, não calcula.
Ele apenas **coloca um dado dentro da variável**.

Exemplo:

idade <- 30

O valor **30** passa a ocupar a gaveta chamada **idade**.

Se depois fizermos:

idade <- 50

O valor **30 desaparece** e é substituído por **50**.

Isso ensina a você uma ideia central da computação:

Os dados são mutáveis.

3.1.7. Entrada de dados

O comando leia() permite que o programa receba valores externos, digitados pelo usuário.

Essa é uma das etapas mais importantes da lógica:

- o programa deixa de ser estático
- passa a reagir
- passa a se adaptar
- deixa de ser previsível

Tudo isso por causa da entrada de dados.

3.1.8. Saída de dados

Saída é a forma do programa comunicar seus resultados.

Você precisa entender que saída não é apenas “mostrar algo”. É **a última etapa do ciclo lógico**, onde a informação se revela.

Um programa sem saída é como uma conversa onde só um fala.

3.1.9. Ciclo Entrada → Processamento → Saída

Este ciclo é tão fundamental que aparece em:

- robótica
- automação industrial
- microcontroladores (como Arduino)
- software comercial
- jogos digitais
- sistemas embarcados
- redes neurais

Não importa a tecnologia, **todo sistema segue o mesmo fluxo**.

✓ Entrada

Recebe dados do usuário ou do ambiente.

✓ Processamento

Transforma dados em resultados.
É aqui que ocorrem cálculos, decisões, validações.

✓ Saída

Mostra o resultado ao usuário ou envia para outra parte do sistema.

Ensinar esse ciclo desenvolve nem Você o “pensar estruturado”.

3.2. Pseudocódigo Avançado e Estruturação Algorítmica

O pseudocódigo deixa de ser apenas um texto com comandos e passa a ser uma **linguagem de organização do pensamento**.

3.2.1. A Profundidade do Pseudocódigo

Um dos grandes erros na aprendizagem inicial é acreditar que pseudocódigo serve apenas “para treinar”.

Na verdade, ele serve para:

- documentar raciocínios
- estruturar soluções
- prever erros
- visualizar a lógica independentemente da linguagem
- planejar sistemas complexos

Profissionais experientes usam pseudocódigo diariamente para:

- desenhar rotinas
- explicar funções para a equipe
- validar ideias antes de codificar
- treinar novos programadores

3.2.2. A Estrutura Interna de um Algoritmo

O Visualg usa uma estrutura simples:

```
algoritmo "nome"
var
   ...
inicio
   ...
fimalgoritmo
```

Mas, pedagogicamente, essa estrutura representa algo muito maior.

Cada parte cumpre uma função lógica:

✓ Cabeçalho

Define a identidade do algoritmo.
Em sistemas reais, cada rotina deve ter um propósito claro.

✓ Declaração de variáveis

Define os recursos que o programa precisa para funcionar.
É como preparar as ferramentas antes da obra.

✓ Corpo do algoritmo

Contém as ações.
Cada linha representa uma instrução clara, objetiva e sem ambiguidade.

✓ Encerramento

Fecha a execução, permitindo controle total do fluxo.

3.2.3. Estratégias profissionais para escrever pseudocódigos

1. **Trabalhar em camadas**
   Primeiro o rascunho, depois a versão detalhada.

2. **Descrever a intenção antes da ação**
   Ex.: // calcular média antes de escrever o cálculo

3. **Evitar instruções desnecessárias**
   Pseudocódigo não é texto, é lógica.

4. **Organizar blocos por função**
   Entrada / Processamento / Saída

5. **Sempre revisar a ordem lógica**
   Um erro de ordem altera toda a execução.

3.2.4. Modelos de Estruturas Avançadas

✓ Modelo 1 — Algoritmo completo e bem-organizado:

```
algoritmo "processo"
var
  dado1, dado2, resultado : inteiro

inicio
  // Entrada
  escreva("Digite o primeiro valor: ")
  leia(dado1)

  escreva("Digite o segundo valor: ")
  leia(dado2)

  // Processamento
```

```
  resultado <- dado1 + dado2

  // Saída
  escreval("Resultado: ", resultado)

fimalgoritmo
```

3.2.5. Preparação para Condicionais e Repetições

Ao dominar pseudocódigo estruturado, você estará pronto para:

- criar decisões (SE / ENTÃO / SENÃO)
- criar laços (ENQUANTO / PARA)
- escrever algoritmos de controle (LEDs, sensores, motores no Arduino)
- resolver problemas de lógica clássicos
- manipular dados de forma inteligente

Este capítulo é a base para tudo isso.

3.2.6. Conclusão

Ao final deste capítulo, o você deverá compreender profundamente que:

- Variáveis são a memória viva do programa.
- Tipos de dados dão forma e significado à informação.
- Entrada e saída transformam programas em sistemas interativos.
- O ciclo Entrada → Processamento → Saída é universal.
- Pseudocódigo é uma ferramenta de pensamento, não apenas de treino.
- Estruturar algoritmos é o primeiro passo para criar programas reais.