Etapa 10 — Depuração e Tratamento de Erros (Debugging and Error Handling)

Trilha: Lógica de Programação

Introdução

Mesmo os melhores programadores cometem erros.
Esses erros são chamados de **bugs**, e o processo de encontrá-los e corrigi-los é conhecido como **depuração (debugging)**.

Além disso, é importante que o programa saiba **lidar com erros inesperados** — como entradas inválidas ou falhas de execução.
Esse processo é chamado de **tratamento de erros (error handling)**.

> "Erros não são falhas; são oportunidades de entender melhor o sistema."

Tipos de Erros em Programação

Durante o desenvolvimento de um algoritmo, diferentes tipos de erros podem ocorrer. Eles são geralmente classificados em três categorias:

1. Erros de Sintaxe (Syntax Errors)

São erros de escrita, quando o código não segue as regras da linguagem.
O programa nem chega a ser executado.

Exemplo:

```
se x > 10

  escreva("Maior que 10")
```

Falta o comando **"então" (then)**.

Dica:
Verifique sempre a **estrutura e os delimitadores** (como parênteses e chaves).

2. Erros de Execução (Runtime Errors)

O programa é compilado, mas **para de funcionar durante a execução**.
Exemplo clássico: divisão por zero.

Exemplo:

a ← 10

b ← 0

c ← a / b // erro de execução

Dica:
Esses erros podem ser **prevenidos** validando entradas e verificando condições antes de executar.

3. Erros Lógicos (Logic Errors)

São os mais difíceis de encontrar, pois **não geram mensagens de erro**.
O programa roda, mas o resultado está incorreto.

Exemplo:

media ← soma / total + 1 // lógica incorreta

Dica:
Faça **testes de mesa (dry run)** para verificar passo a passo os cálculos e resultados.

O que é Depuração (Debugging)

Depuração é o processo sistemático de **identificar, entender e corrigir erros** em um programa.
Ela envolve analisar o comportamento do código, examinar variáveis e acompanhar o fluxo de execução.

Objetivo:
Garantir que o programa se comporte conforme o esperado.

Exemplo:

escreva("Início da função")

resultado ← calcular_media(notas)

escreva("Resultado:", resultado)

Essas mensagens ajudam a **rastrear a execução** e entender onde o erro ocorre.

Técnicas de Depuração

- **Leitura de código (code review)** — ler cuidadosamente o código e buscar inconsistências.
- **Mensagens de log (logging)** — imprimir informações no console.
- **Execução passo a passo (step-by-step)** — usar o depurador da IDE para verificar linha por linha.
- **Pontos de interrupção (breakpoints)** — pausar o programa em locais estratégicos.
- **Análise de variáveis (watch/inspect)** — observar o valor das variáveis durante a execução.

“Depurar é pensar como o computador pensa.”

Uso de Logs (Logging)

Logs são mensagens que ajudam a **acompanhar o comportamento interno** do programa. Eles registram o que aconteceu e onde.

Exemplo de log genérico:

```
escreva("Iniciando processamento")

resultado ← calcular_total()

escreva("Resultado obtido:", resultado)
```

Boas práticas:

- Use níveis de log (info, warning, error).
- Não exiba dados sensíveis.
- Remova logs desnecessários antes da entrega do sistema.

Tratamento de Erros (Error Handling)

O tratamento de erros é o conjunto de técnicas usadas para **prevenir que o programa falhe completamente** diante de situações inesperadas.

Exemplo de estrutura genérica:

```
tente

  resultado ← dividir(a, b)
```

```
capture erro
  escreva("Ocorreu um erro: ", erro)
fimtente
```

“Tratar erros é antecipar o inesperado.”

Objetivos principais:

- Evitar que o programa trave.
- Exibir mensagens amigáveis.
- Registrar o erro para análise posterior.

Erros Comuns em Programas

- Divisão por zero
- Variável não inicializada
- Índice fora dos limites do vetor
- Conversão de tipo inválida
- Falhas de entrada de dados (usuário digita texto em vez de número)

Exemplo:

```
funcao dividir(a, b)
  se b = 0 entao
    escreva("Erro: divisão por zero.")
    retorna 0
  senao
    retorna a / b
  fimse
fimfuncao
```

Estratégias de Tratamento

- **Valide as entradas antes de processar**
- **Use valores padrão (default)** em caso de falha
- **Crie mensagens claras** para o usuário
- **Utilize blocos try/catch (tente/capture)**
- **Documente e registre os erros**

Exemplo com validação:

```
entrada ← leia()

se nao_e_numero(entrada) entao

  escreva("Entrada inválida. Digite um número.")

senao

  resultado ← entrada * 2

  escreva("Resultado:", resultado)

fimse
```

Ferramentas de Depuração

- **Visual Studio Code**
- **PyCharm**
- **Eclipse**
- **Dev-C++**
- **Logcat (Android)**
- **Console (JavaScript/Python)**

Esses ambientes permitem **pausar, inspecionar e analisar o comportamento do programa** durante a execução.

Boas Práticas

- Antecipe possíveis falhas.
- Use mensagens descritivas.
- Evite mensagens genéricas como “Erro desconhecido”.
- Teste entradas extremas (ex: zero, vazio, muito grandes).
- Documente o erro e a solução.

“Erros bem tratados não assustam o usuário — aumentam a confiança.”

Exercícios de Múltipla Escolha

1. O que significa depurar (debugging)?
a) Compilar o programa
b) Executar o código
c) Encontrar e corrigir erros
d) Apagar variáveis

2. Qual tipo de erro impede o programa de ser executado?
a) Erro de execução
b) Erro lógico
c) Erro de sintaxe
d) Erro de entrada

3. Qual é um exemplo de erro de execução?
a) Esquecer um parêntese
b) Resultado incorreto de uma fórmula
c) Dividir um número por zero
d) Variável com nome incorreto

4. O tratamento de erros serve para:
a) Deixar o programa mais rápido
b) Evitar a execução de funções
c) Fazer o programa reagir bem a falhas
d) Exibir o resultado final

5. Qual estrutura é usada para capturar erros durante a execução?
a) repita/até
b) tente/capture
c) para/enquanto
d) inicie/fim

Gabarito

1. c) Encontrar e corrigir erros
2. c) Erro de sintaxe
3. c) Dividir um número por zero
4. c) Fazer o programa reagir bem a falhas
5. b) tente/capture