Introdução a Algoritmos

Raoni F. S. Teixeira

Aula 6 - Mais sobre Comandos de Repetição

1 Relembrando a aula anterior

Na aula passada, estudamos os comandos `while` e `do-while`, compreendendo como eles permitem repetir ações até que uma condição deixe de ser verdadeira. Aprendemos também a identificar três perguntas fundamentais que norteiam qualquer repetição:

1. O que deve ser repetido?
2. Quantas vezes deve se repetir?
3. Qual condição representa essa repetição?

Essas perguntas continuam a nos guiar. Elas revelam o padrão por trás de cada laço de repetição, ajudando a enxergar a lógica que transforma uma sequência manual de comandos em um processo automatizado.

Hoje, ampliaremos essa ideia. Vamos descobrir outras formas de estruturar repetições, torná-las mais concisas e compreender como encerrá-las quando necessário.

2 Do `while` ao `for`: quando a repetição ganha forma

O comando `while` é versátil, mas em muitas situações há um padrão fixo que se repete: um valor inicial, uma condição e um passo de incremento. Quando esses três elementos aparecem juntos, podemos reorganizá-los em uma forma mais compacta — o comando `for`.

2.1 Estrutura do `for`

```
for (inicio; condicao; passo) {
    comandos;
}
```

```
int i;
for (i = 1; i <= 100; i++) {
    printf("%d␣", i);
}
```

O mesmo código, escrito com `while`, exigiria linhas adicionais. O `for` agrupa, em uma única estrutura, o início, a condição e o passo. Esse agrupamento melhora a legibilidade — um valor que William Zinsser sempre destacou: "*Good writing is clear thinking made visible.*"

2.2 Generalizando o padrão

Assim como antes, perguntamos:

- O que se repete? A impressão do número.
- Quantas vezes? Até o contador alcançar 100.
- Qual condição? Enquanto `i <= 100`.

O `for` é, portanto, uma forma natural de expressar repetições que possuem uma contagem definida.

3 Repetições dentro de repetições: o poder do aninhamento

O próximo passo é entender que laços também podem conter outros laços. Essa técnica — chamada **aninhamento** — permite percorrer estruturas bidimensionais, como tabelas, matrizes ou padrões numéricos.

```
int i, j;
for (i = 1; i <= 5; i++) {
    printf("Linha␣%d:␣", i);
    for (j = 0; j < 5; j++) {
        printf("%d␣", j);
    }
    printf("\n");
}
```

Cada linha contém um laço interno que imprime as colunas. Esse padrão é comum em algoritmos de processamento de dados, gráficos ou tabulações.

Resultado:

```
Linha 1: 0 1 2 3 4
Linha 2: 0 1 2 3 4
...
```

4 Encerrando repetições: quando parar é essencial

Assim como é importante iniciar um laço, também é fundamental saber encerrá-lo. Um programa que nunca termina perde seu propósito. Em `while` e `for`, o controle é dado pela condição, mas existem situações em que queremos interromper um laço antecipadamente.

4.1 O comando `break`

O comando `break` encerra imediatamente a execução de um laço, mesmo que a condição ainda seja verdadeira.

```
int i;
for (i = 0; i < 100; i++) {
    printf("O␣valor␣de␣i␣  ␣%d\n", i);
    if (i == 10)
        break;
}
```

O programa imprime apenas de 0 a 10. Apesar de útil, o `break` deve ser usado com cuidado: ele torna o fluxo de execução menos previsível e pode reduzir a clareza do código.

4.2 O comando `continue`

O `continue` não encerra o laço, mas faz com que a execução volte imediatamente ao teste da condição, saltando o restante do bloco atual.

```
int i, aux;
for (i = 0; i < 10; i++) {
    scanf("%d", &aux);
    if (aux <= 0)
        continue;
    printf("Numero␣positivo:␣%d\n", aux);
}
```

Essa técnica é útil para "pular" casos especiais, mantendo o controle da repetição sem interrompê-la.

4.3 Uma observação sobre estilo e clareza

O uso excessivo de `break` e `continue` pode fragmentar o raciocínio do leitor. Como lembra Zinsser, "*a clear sentence is no accident*": um bom código, como um bom texto, conduz o leitor com previsibilidade e ritmo. Quando a estrutura de repetição é bem planejada, o comportamento é fácil de seguir — sem surpresas escondidas.