Material de Apoio - Semana 4

Estruturas de Repetição

Introdução a Técnicas de Programação (2025.2)

Objetivos de Aprendizagem

Ao final desta semana, você será capaz de:

- Compreender a necessidade e importância das estruturas de repetição
- Utilizar as estruturas `while`, `do/while` e `for` para repetições
- Escolher a estrutura de repetição mais adequada para cada situação
- Resolver problemas complexos usando estruturas de repetição

---

1. Introdução às Estruturas de Repetição

1.1 Por que Precisamos de Repetições?

Imagine que você precise escrever um programa para exibir os números de 1 a 100. Sem estruturas de repetição, você teria que escrever 100 linhas de código:

```c
#include <stdio.h>

int main() {
    printf("1\n");
    printf("2\n");
    printf("3\n");
    printf("4\n");
    // ... mais 96 linhas!
    printf("100\n");

    return 0;
}
```

Problemas desta abordagem:

- Código extenso e repetitivo
- Difícil manutenção
- Propenso a erros
- Inflexibilidade (e se quisermos mostrar até 1000?)

1.2 O Conceito de Iteração

Uma **iteração** é cada execução de um bloco de código dentro de uma estrutura de repetição. As estruturas de repetição nos permitem:

- Executar o mesmo bloco de código múltiplas vezes
- Controlar quantas vezes o código será executado
- Processar conjuntos de dados de forma eficiente
- Criar algoritmos mais elegantes e concisos

1.3 Tipos de Estruturas de Repetição em C

Em C, temos três principais estruturas de repetição:

1. `while` - Repetição condicional (testa antes de executar)
2. `do/while` - Repetição condicional (testa depois de executar)
3. `for` - Repetição controlada por contador

---

2. Estrutura de Repetição `while`

2.1 Sintaxe e Funcionamento

A estrutura `while` executa um bloco de código **enquanto** uma condição for verdadeira:

```
while (condição) {
    // bloco de código a ser repetido
    // modificação da condição (importante!)
}
```

Características importantes:

- A condição é avaliada **antes** de cada execução
- Se a condição for falsa inicialmente, o bloco nunca é executado
- É crucial modificar a condição dentro do bloco para evitar loops infinitos

2.2 Exemplo Básico: Contando de 1 a 100

```c
#include <stdio.h>

int main() {
    int i;

    i = 1;                    // Inicialização
    while (i <= 100) {        // Condição
        printf("%d\n", i);    // Bloco de código
        i = i + 1;            // Incremento (modificação da condição)
    }

    return 0;
}
```

Análise do fluxo:

1. `i` é inicializado com 1
2. Verifica se `i <= 100` (verdadeiro)
3. Executa o bloco (imprime 1)
4. Incrementa `i` para 2
5. Volta ao passo 2
6. Continua até `i` ser 101

2.3 Exemplos Práticos

Exemplo 1: Encontrar divisores de um número

```c
#include <stdio.h>

int main() {
    int n, i;

    printf("Digite um número: ");
    scanf("%d", &n);

    printf("Divisores de %d: ", n);
    i = 1;
    while (i <= n) {
        if (n % i == 0) {
            printf("%d ", i);
        }
        i++;
    }
    printf("\n");

    return 0;
}
```

Exemplo 2: Soma de números digitados pelo usuário

```c
#include <stdio.h>

int main() {
    int numero, soma = 0;

    printf("Digite números (0 para parar):\n");
    scanf("%d", &numero);

    while (numero != 0) {
        soma += numero;
        printf("Soma atual: %d\n", soma);
        printf("Digite outro número: ");
        scanf("%d", &numero);
    }

    printf("Soma final: %d\n", soma);
```

```
    return 0;
}
```

2.4 Cuidados com Loops Infinitos

Um **loop infinito** ocorre quando a condição nunca se torna falsa:

```
// ❌ ERRO: Loop infinito
int i = 1;
while (i <= 10) {
    printf("%d\n", i);
    // Esqueceu de incrementar i!
}

// ✅ CORRETO: Loop finito
int i = 1;
while (i <= 10) {
    printf("%d\n", i);
    i++;  // Incrementa i
}
```

---

3. Estrutura de Repetição `do/while`

3.1 Sintaxe e Diferenças do `while`

```
do {
    // bloco de código
} while (condição);
```

Principais diferenças do `while`:

- Executa o bloco **pelo menos uma vez**
- A condição é testada **após** a execução do bloco
- Útil quando você precisa executar o código antes de testar a condição

3.2 Quando Usar `do/while`

O `do/while` é ideal para situações onde você precisa:

- Validar entrada do usuário
- Exibir menus que devem aparecer pelo menos uma vez
- Garantir que uma ação seja executada antes da verificação

3.3 Exemplo: Validação de Entrada

```c
#include <stdio.h>

int main() {
    int a, b;

    printf("Digite o numerador: ");
    scanf("%d", &a);

    do {
        printf("Digite o denominador (não pode ser 0): ");
        scanf("%d", &b);

        if (b == 0) {
            printf("Erro! O denominador não pode ser zero.\n");
        }
    } while (b == 0);

    printf("Resultado da divisão: %.2f\n", (float)a / b);
    return 0;
}
```

3.4 Exemplo: Menu de Opções

```c
#include <stdio.h>

int main() {
    int opcao;

    do {
        printf("\n=== MENU ===\n");
        printf("1. Somar dois números\n");
        printf("2. Multiplicar dois números\n");
        printf("3. Sair\n");
        printf("Escolha uma opção: ");
        scanf("%d", &opcao);

        switch (opcao) {
            case 1: {
                int a, b;
                printf("Digite dois números: ");
                scanf("%d %d", &a, &b);
                printf("Soma: %d\n", a + b);
                break;
            }
            case 2: {
                int a, b;
                printf("Digite dois números: ");
                scanf("%d %d", &a, &b);
                printf("Produto: %d\n", a * b);
                break;
```

```
            }
            case 3:
                printf("Saindo...\n");
                break;
            default:
                printf("Opção inválida!\n");
        }
    } while (opcao != 3);

    return 0;
}
```

---

4. Estrutura de Repetição `for`

4.1 Sintaxe e Componentes

A estrutura `for` é mais compacta e organizada para loops com contador:

```
for (inicialização; condição; incremento) {
    // bloco de código
}
```

Componentes:

- **Inicialização**: Executada uma vez, antes do loop começar
- **Condição**: Testada antes de cada iteração
- **Incremento**: Executado após cada iteração

4.2 Equivalência entre `for` e `while`

Estes códigos são equivalentes:

```
// Usando for
for (int i = 1; i <= 10; i++) {
    printf("%d\n", i);
}

// Usando while
int i = 1;          // Inicialização
while (i <= 10) {   // Condição
    printf("%d\n", i);
    i++;            // Incremento
}
```

4.3 Vantagens do `for`

- **Organização**: Todos os elementos do loop estão em uma linha

- **Legibilidade**: Fácil de ver inicialização, condição e incremento
- **Menos erros**: Dificulta esquecer o incremento
- **Padrão**: Amplamente usado para loops com contador

4.4 Exemplos Práticos

Exemplo 1: Tabuada

```c
#include <stdio.h>

int main() {
    int numero;

    printf("Digite um número para ver sua tabuada: ");
    scanf("%d", &numero);

    printf("\nTabuada do %d:\n", numero);
    for (int i = 1; i <= 10; i++) {
        printf("%d x %d = %d\n", numero, i, numero * i);
    }

    return 0;
}
```

Exemplo 2: Verificação de Número Primo

```c
#include <stdio.h>

int main() {
    int numero, divisores = 0;

    printf("Digite um número: ");
    scanf("%d", &numero);

    for (int i = 1; i <= numero; i++) {
        if (numero % i == 0) {
            divisores++;
        }
    }

    if (divisores == 2) {
        printf("%d é primo.\n", numero);
    } else {
        printf("%d não é primo.\n", numero);
    }

    return 0;
}
```

Exemplo 3: Cálculo de Fatorial

```c
#include <stdio.h>

int main() {
    int numero;
    long long fatorial = 1;

    printf("Digite um número: ");
    scanf("%d", &numero);

    if (numero < 0) {
        printf("Fatorial não definido para números negativos.\n");
    } else {
        for (int i = 1; i <= numero; i++) {
            fatorial *= i;
        }
        printf("Fatorial de %d = %lld\n", numero, fatorial);
    }

    return 0;
}
```

4.5 Variações do `for`

Loop decrescente:

```c
// Contando de 10 até 1
for (int i = 10; i >= 1; i--) {
    printf("%d\n", i);
}
```

Incremento personalizado:

```c
// Números pares de 0 a 20
for (int i = 0; i <= 20; i += 2) {
    printf("%d\n", i);
}
```

Múltiplas variáveis:

```c
// Duas variáveis no mesmo loop
for (int i = 0, j = 10; i < j; i++, j--) {
    printf("i = %d, j = %d\n", i, j);
}
```

5. Escolhendo a Estrutura Correta

5.1 Quando Usar Cada Estrutura

Situação	Estrutura Recomendada	Justificativa
Número exato de repetições conhecido	`for`	Mais organizado e claro
Repetir até uma condição ser atendida	`while`	Flexibilidade na condição
Executar pelo menos uma vez	`do/while`	Garante execução mínima
Percorrer intervalos numéricos	`for`	Controle natural do contador
Validação de entrada	`do/while`	Executa antes de testar
Processamento de dados indefinidos	`while`	Para quando não sabe quantos dados virão

5.2 Exemplos Comparativos

Situação: Ler números até digitar 0

```c
// Melhor opção: while
int numero;
scanf("%d", &numero);
while (numero != 0) {
    printf("Você digitou: %d\n", numero);
    scanf("%d", &numero);
}

// Opção menos elegante: for
int numero;
for (scanf("%d", &numero); numero != 0; scanf("%d", &numero)) {
    printf("Você digitou: %d\n", numero);
}
```

Situação: Menu que deve aparecer pelo menos uma vez

```c
// Melhor opção: do/while
int opcao;
do {
    printf("1. Opção A\n2. Opção B\n3. Sair\nEscolha: ");
    scanf("%d", &opcao);
    // processar opção
} while (opcao != 3);
```

```c
// Opção menos elegante: while
int opcao;
printf("1. Opção A\n2. Opção B\n3. Sair\nEscolha: ");
scanf("%d", &opcao);
while (opcao != 3) {
    // processar opção
    printf("1. Opção A\n2. Opção B\n3. Sair\nEscolha: ");
    scanf("%d", &opcao);
}
```

---

6. Problemas Clássicos com Estruturas de Repetição

6.1 Problema 3n + 1 (Conjectura de Collatz)

```c
#include <stdio.h>

int main() {
    int n, passos = 0;

    printf("Digite um número inteiro positivo: ");
    scanf("%d", &n);

    printf("Sequência: %d", n);

    while (n != 1) {
        if (n % 2 == 0) {
            n = n / 2;
        } else {
            n = 3 * n + 1;
        }
        printf(" -> %d", n);
        passos++;
    }

    printf("\nNúmero de passos: %d\n", passos);
    return 0;
}
```

6.2 Série de Fibonacci

```c
#include <stdio.h>

int main() {
    int n, primeiro = 0, segundo = 1, proximo;

    printf("Quantos termos da sequência de Fibonacci? ");
    scanf("%d", &n);
```

```
    if (n >= 1) printf("Fibonacci: %d", primeiro);
    if (n >= 2) printf(" %d", segundo);

    for (int i = 3; i <= n; i++) {
        proximo = primeiro + segundo;
        printf(" %d", proximo);
        primeiro = segundo;
        segundo = proximo;
    }
    printf("\n");

    return 0;
}
```

6.3 Jogo de Adivinhação

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    int numero_secreto, palpite, tentativas = 0;

    srand(time(NULL));  // Inicializa gerador de números aleatórios
    numero_secreto = rand() % 100 + 1;  // Número entre 1 e 100

    printf("=== JOGO DE ADIVINHAÇÃO ===\n");
    printf("Adivinhe o número entre 1 e 100!\n\n");

    do {
        printf("Digite seu palpite: ");
        scanf("%d", &palpite);
        tentativas++;

        if (palpite > numero_secreto) {
            printf("Muito alto! Tente um número menor.\n");
        } else if (palpite < numero_secreto) {
            printf("Muito baixo! Tente um número maior.\n");
        } else {
            printf("🎉 Parabéns! Você acertou em %d tentativas!\n",
tentativas);
        }

    } while (palpite != numero_secreto);

    return 0;
}
```

---

7. Boas Práticas com Estruturas de Repetição

7.1 Indentação e Legibilidade

✅ **Boa indentação:**

```c
for (int i = 1; i <= 10; i++) {
    printf("Número: %d\n", i);
    if (i % 2 == 0) {
        printf("  É par!\n");
    }
}
```

❌ **Má indentação:**

```c
for (int i = 1; i <= 10; i++) {
printf("Número: %d\n", i);
if (i % 2 == 0) {
printf("  É par!\n");
}
}
```

7.2 Nomenclatura de Variáveis de Controle

✅ **Bons nomes:**

```c
for (int contador = 0; contador < 10; contador++) { ... }
for (int linha = 0; linha < altura; linha++) { ... }
for (int indice = 0; indice < tamanho; indice++) { ... }
```

❌ **Nomes pouco descritivos:**

```c
for (int x = 0; x < 10; x++) { ... }
for (int a = 0; a < altura; a++) { ... }
```

7.3 Evitando Loops Infinitos

Sempre certifique-se de que:

- A condição pode se tornar falsa
- A variável de controle é modificada dentro do loop
- A lógica de parada está correta

```c
// ✅ Correto: condição pode se tornar falsa
int i = 0;
```

```c
while (i < 10) {
    printf("%d\n", i);
    i++;  // Modifica a variável de controle
}

// ❌ Perigoso: pode gerar loop infinito
int i = 0;
while (i != 10) {
    printf("%d\n", i);
    i += 2;  // Se i passar de 10, nunca será igual a 10
}
```

7.4 Inicialização de Variáveis

```c
// ✅ Inicialize variáveis importantes
int soma = 0;
int contador = 0;
for (int i = 1; i <= 10; i++) {
    soma += i;
    contador++;
}

// ❌ Variáveis não inicializadas podem ter valores aleatórios
int soma;  // Valor indefinido!
int contador;  // Valor indefinido!
```

---

8. Exercícios Resolvidos

8.1 Números Pares e Ímpares

```c
#include <stdio.h>

int main() {
    int n;

    printf("Digite n: ");
    scanf("%d", &n);

    printf("Pares: ");
    for (int i = 1; i <= n; i++) {
        if (i % 2 == 0) {
            printf("%d ", i);
        }
    }

    printf("\nÍmpares: ");
    for (int i = 1; i <= n; i++) {
        if (i % 2 != 0) {
```

```
            printf("%d ", i);
        }
    }
    printf("\n");

    return 0;
}
```

8.2 Acerte a Senha

```
#include <stdio.h>

int main() {
    int senha_correta = 1234;
    int tentativa;
    int numero_tentativas = 0;

    do {
        printf("Digite a senha: ");
        scanf("%d", &tentativa);
        numero_tentativas++;

        if (tentativa != senha_correta) {
            printf("Senha incorreta! Tente novamente.\n");
        }
    } while (tentativa != senha_correta);

    printf("Senha correta! Você acertou em %d tentativas.\n",
numero_tentativas);
    return 0;
}
```

8.3 Calculadora com Menu

```
#include <stdio.h>

int main() {
    int opcao;
    float num1, num2;

    do {
        printf("\n=== CALCULADORA ===\n");
        printf("1. Somar\n");
        printf("2. Subtrair\n");
        printf("3. Multiplicar\n");
        printf("4. Dividir\n");
        printf("5. Sair\n");
        printf("Escolha uma opção: ");
        scanf("%d", &opcao);
```

```
        if (opcao >= 1 && opcao <= 4) {
            printf("Digite dois números: ");
            scanf("%f %f", &num1, &num2);
        }

        switch (opcao) {
            case 1:
                printf("Resultado: %.2f\n", num1 + num2);
                break;
            case 2:
                printf("Resultado: %.2f\n", num1 - num2);
                break;
            case 3:
                printf("Resultado: %.2f\n", num1 * num2);
                break;
            case 4:
                if (num2 != 0) {
                    printf("Resultado: %.2f\n", num1 / num2);
                } else {
                    printf("Erro: Divisão por zero!\n");
                }
                break;
            case 5:
                printf("Saindo da calculadora...\n");
                break;
            default:
                printf("Opção inválida!\n");
        }
    } while (opcao != 5);

    return 0;
}
```

---

9. Dicas para a Lista de Exercícios

9.1 Estratégias de Resolução

1. **Identifique o tipo de repetição necessária:**
    - Quantidade conhecida → `for`
    - Condição de parada → `while`
    - Executar pelo menos uma vez → `do/while`
2. **Planeje antes de codificar:**
    - Defina a condição de parada
    - Identifique o que muda a cada iteração
    - Determine valores iniciais
3. **Teste com casos simples primeiro:**

- Comece com exemplos pequenos
- Verifique se a lógica está correta
- Depois teste com casos mais complexos

9.2 Checklist de Verificação

Antes de finalizar seu código, verifique:

- [ ] A condição de parada está correta?
- [ ] As variáveis foram inicializadas?
- [ ] A variável de controle é modificada dentro do loop?
- [ ] O código trata casos especiais (números negativos, zero, etc.)?
- [ ] A indentação está clara?
- [ ] Os nomes das variáveis são descritivos?

9.3 Depuração de Loops

Para debugar loops problemáticos:

```
// Adicione prints de depuração
int i = 0;
while (i < 10) {
    printf("DEBUG: i = %d\n", i);  // Print de depuração
    // seu código aqui
    i++;
    printf("DEBUG: i após incremento = %d\n", i);
}
```

---

10. Preparação para a Próxima Semana

Na **Semana 5**, estudaremos:

- Conceitos fundamentais de funções
- Declaração e chamada de funções
- Passagem de parâmetros
- Valor de retorno
- Escopo de variáveis
- Funções recursivas básicas

Exercício preparatório: Pense em como você poderia transformar alguns dos códigos desta semana em funções reutilizáveis. Por exemplo, como criar uma função para verificar se um número é primo?

---

11. Recursos Adicionais

11.1 Comandos Úteis para Compilação e Depuração

```
# Compilar com flags de depuração
gcc -g -Wall -o programa programa.c

# Executar com GDB
gdb ./programa

# Comandos básicos do GDB
(gdb) break main
(gdb) run
(gdb) step
(gdb) print variavel
(gdb) continue
```

11.2 Ferramentas Online para Prática

- **Online C Compiler**: onlinegdb.com
- **Replit**: replit.com

11.3 Exercícios Extras para Prática

1. Crie um programa que calcule o MDC de dois números usando o algoritmo de Euclides
2. Implemente um conversor de decimal para binário
3. Desenvolva um programa que desenhe padrões com asteriscos usando loops
4. Crie um validador de CPF simples (apenas verificação de dígitos)

Lembre-se: A prática constante é fundamental para dominar as estruturas de repetição. Experimente modificar os exemplos e criar suas próprias variações!