Recursividade em Linguagem C - Estudo Dirigido

Explicação sobre Recursividade

Recursividade é uma técnica onde uma função chama a si mesma para resolver problemas menores da mesma natureza. Cada chamada recursiva deve trabalhar em um problema menor que o anterior, aproximando-se cada vez mais de um **caso base** que pode ser resolvido diretamente.

Estrutura Básica de uma Função Recursiva:

```c
tipo_retorno nome_funcao(parametros) {
    // 1. Caso base (condição de parada)
    if (condição_simples) {
        return valor_base;
    }
    // 2. Chamada recursiva (problema menor)
    else {
        return operacao(nome_funcao(parametro_reduzido));
    }
}
```

Fragmentos de Exemplos

Exemplo 1: Fatorial

```c
int fatorial(int n) {
    // Caso base: fatorial de 0 ou 1 é 1
    if (n <= 1) {
        return 1;
    }
    // Chamada recursiva: n! = n * (n-1)!
    else {
        return n * fatorial(n - 1);
```

```
    }
}
```

Exemplo 2: Fibonacci

c

```c
int fibonacci(int n) {
    // Casos base: fib(0) = 0, fib(1) = 1
    if (n == 0) return 0;
    if (n == 1) return 1;

    // Chamada recursiva: fib(n) = fib(n-1) + fib(n-2)
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

Exemplo 3: Soma de Array

c

```c
int soma_array(int arr[], int tamanho) {
    // Caso base: array vazio
    if (tamanho == 0) return 0;

    // Chamada recursiva: soma do primeiro elemento + soma do restante
    return arr[0] + soma_array(arr + 1, tamanho - 1);
}
```

Exercícios de Recursividade

Exercício 1: Potência Recursiva

Escreva uma função recursiva `potencia(int base, int expoente)` que calcule base^expoente.

Dica: Lembre-se que:

- $base^0 = 1$ (caso base)
- $base^n = base * base^{n-1}$ (chamada recursiva)
- Considere expoentes negativos se quiser um desafio extra

Exercício 2: Contagem Regressiva

Crie uma função `contagem_regressiva(int n)` que imprima números de n até 1 e depois "Fogo!".

Dica:

•Imprima o número atual antes da chamada recursiva para contar regressivamente
•Após a recursão, imprima "Fogo!" quando n for 0

Exercício 3: Soma de Dígitos

Implemente `soma_digitos(int n)` que retorne a soma dos dígitos de um número inteiro positivo.

Dica:

•Caso base: número com um único dígito
•Use n % 10 para pegar o último dígito
•Use n / 10 para remover o último dígito

Exercício 4: Máximo Divisor Comum (MDC)

Escreva uma função recursiva `mdc(int a, int b)` usando o algoritmo de Euclides.

Dica: Algoritmo de Euclides:

•mdc(a, 0) = a
•mdc(a, b) = mdc(b, a % b)

Exercício 5: Inversão de String

Crie `inverter_string(char *str, int inicio, int fim)` que inverta uma string recursivamente.

Dica:

•Caso base: quando início >= fim
•Troque o caractere na posição 'inicio' com o na posição 'fim'
•Chame a função recursivamente com inicio+1 e fim-1

Exercício 6: Busca Binária Recursiva

Implemente `busca_binaria(int arr[], int inicio, int fim, int alvo)` que retorne o índice do elemento procurado.

Dica:

•Caso base: array vazio (inicio > fim)
•Calcule o meio: (inicio + fim) / 2
•Compare o elemento do meio com o alvo e ajuste os índices

Exercício 7: Torres de Hanói

Resolva o problema das Torres de Hanói com uma função `hanoi(int discos, char origem, char destino, char auxiliar)`.

Dica:

•Caso base: 1 disco - mova diretamente de origem para destino
•Para n discos:

1.Mova n-1 discos de origem para auxiliar
2.Mova o disco maior de origem para destino
3.Mova n-1 discos de auxiliar para destino

Exercício 8: Palíndromo Recursivo

Escreva `eh_palindromo(char *str, int inicio, int fim)` que verifique se uma string é palíndromo.

Dica:

•Casos base: string vazia ou com 1 caractere é palíndromo
•Compare o primeiro e último caractere
•Se forem iguais, verifique recursivamente o substring interno

Exercício 9: Conversão de Base

Implemente `converter_base(int n, int base)` que converta um número decimal para outra base (2-16) recursivamente.

Dica:

•Caso base: n < base (converta diretamente)
•Converta n/base recursivamente
•Adicione o dígito correspondente a n % base
•Para bases > 10, use letras para dígitos > 9

Exercício 10: Soma de Elementos Pares

Crie `soma_pares(int arr[], int tamanho)` que retorne a soma apenas dos elementos pares de um array.

Dica:

•Caso base: array vazio (tamanho 0)
•Verifique se o primeiro elemento é par

•Some-o (se for par) com a soma recursiva do restante do array

Dicas para Resolução

1. **Identifique e defina o caso base**: Qual é a situação mais simples que pode ser resolvida diretamente? Além de evitar o loop infinito.
2. **Reduza o problema**: Como transformar o problema atual em uma versão menor do mesmo problema?
3. **Combine resultados**: Como combinar a solução do problema menor com o passo atual?
4. **Teste com valores pequenos**: Sempre teste com casos simples primeiro (0, 1, 2).
5. **Visualize a pilha de chamadas** para entender o fluxo
6. **Use parâmetros que reduzam o problema** a cada chamada
7. **Documente sua função** com comentários sobre o caso base e a lógica recursiva

Boa sorte com os exercícios! Lembre-se que a recursividade é um conceito que fica mais natural com a prática.