Explorando Recursividade: Transformando um `for` em Recursão

Monitor Luis Phillip

March 16, 2024

1 Introdução

Vamos mergulhar na arte da recursividade, transformando uma estrutura de repetição `for` em uma função recursiva. Para ilustrar esse processo, consideremos o cálculo do fatorial de um número como exemplo.

2 Estrutura de Repetição `for`

```
int n = 5; // N mero do fatorial
int resultado = 1;
for (int i = 2; i <= n; i++) {
    resultado *= i; // Realiza a multiplica  o 1x2x3x4x..n
}
```

Listing 1: Estrutura de Repetição `for`

2.1 Análise das Variáveis

- `i`: Usado para controle do incremento.
- `n`: Determina o máximo do fatorial.
- `resultado`: Armazena valores intermediários para continuar a multiplicação do fatorial.

3 Transformando em Recursão

Podemos adaptar as variáveis não constantes para parâmetros de uma função recursiva. Vamos ver como fazer isso:

```
int fatorial(int i, int n, int resultado) {
    if (i <= n) {
        resultado *= i;
        i++;
        return fatorial(i, n, resultado);
    } else {
        return resultado;
    }
}
// Chamada na main: fatorial(2, n, resultado)
```

Listing 2: Função Recursiva para o Fatorial

3.1 Passo 1: Seleção de Variáveis

Selecionamos variáveis que mudam a cada repetição e as transformamos em parâmetros da função recursiva. No exemplo, isso inclui `i` e `resultado`.

3.2 Passo 2: Condição de Parada

Estabelecemos uma condição de parada na função recursiva, semelhante à condição de parada no `for`. No caso, quando `i` não for mais menor ou igual a `n`.

3.3 Passo 3: Retorno de Valores

Uma função recursiva possui escopo limitado diferente das estruturas de repetição, então devemos lidar com cautela no retorno para que o resultado certo seja retornado. Como fundamento da recursão, devemos sempre lembrar que a função sempre irá retornar de trás para frente, (pois ela chama todas as instâncias até a condição de parada) como em uma pilha.

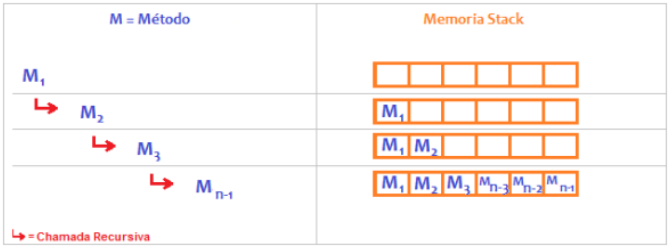

Figure 1: Pilha de chamadas em uma função recursiva

Como na imagem acima, podemos observar a pilha de chamadas até o retorno. Podemos verificar essa cautela em outra forma de fazer um fatorial de forma recursiva.

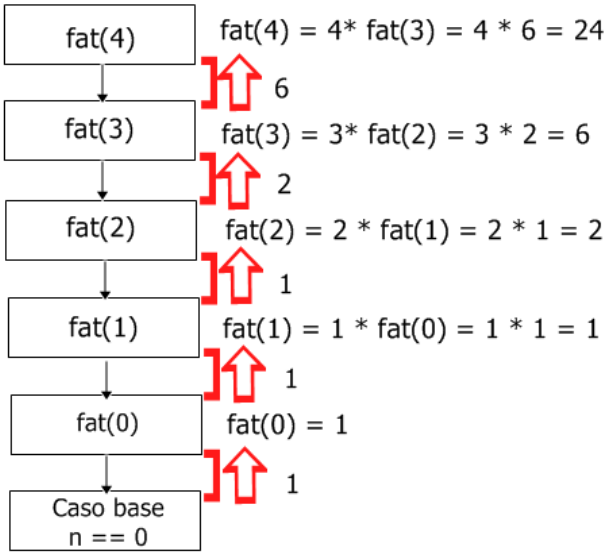

Figure 2: Pilha de chamadas em uma função Fatorial

```c
int fatorial(int n) {
    if (n == 0) {
        return 1;
    }
    return n * fatorial(n - 1);
}
```

Listing 3: Recursão Direta para o Fatorial

Nesta versão, a função espera o retorno da chamada recursiva, começando do número fornecido e diminuindo até alcançar a condição de parada (`n == 0`), retornando 1. Essa abordagem calcula o fatorial de forma eficiente, reduzindo a quantidade de parâmetros na função.

4 Exemplos Adicionais e Erros Comuns

Além do cálculo do fatorial, a recursão pode ser aplicada em uma variedade de problemas. Abaixo, apresentamos mais alguns exemplos e destacamos erros comuns que podem ocorrer ao trabalhar com funções recursivas.

4.1 Exemplo: Soma dos Números

A soma dos números de 1 a `n` pode ser calculada de forma recursiva da seguinte maneira:

```
int soma(int n) {
    if (n == 0) {
        return 0;
    }
    return n + soma(n - 1);
}
```

Listing 4: Função Recursiva para a Soma dos Números

4.2 Exemplo: Fibonacci

A sequência de Fibonacci é um exemplo clássico de aplicação de recursão.

```
int fibonacci(int n) {
    if (n <= 1) {
        return n;
    }
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

Listing 5: Função Recursiva para a Sequência de Fibonacci

4.3 Erro Comum: Falta de Condição de Parada

Esquecer de incluir uma condição de parada pode resultar em um loop infinito, consumindo toda a memória disponível.

4.4 Erro Comum: Chamada Recursiva Incorreta

Uma chamada recursiva incorreta pode levar a resultados inesperados ou a falhas no programa. Certifique-se sempre de ajustar os parâmetros corretamente em cada chamada.

4.5 Principais Dificuldades

A recursão pode parecer uma hidra de sete cabeças desorganizada e confusa, mas acredite, há algoritmos, principalmente em AEDs 2, que são muito mais fáceis se utilizarmos recursão. A dica para tornar a recursão mais fácil é praticar. Quanto mais você praticar e notar o comportamento da recursividade, mais natural o pensamento recursivo se torna e mais fácil você irá compreender e desenvolver códigos recursivos.

4.6 Dicas para Parâmetros

Quando temos uma função recursiva com muitos parâmetros de suporte, como no caso abaixo:

```
int fatorial(int i, int n, int resultado) {
    if (i <= n) {
        resultado *= i;
        i++;
        return fatorial(i, n, resultado);
    } else {
        return resultado;
    }
}
```

Listing 6: Exemplo de Função Recursiva com Muitos Parâmetros

Podemos criar duas funções (uma que o desenvolvedor pode chamar na `main` com os parâmetros necessários) e outra que a função 1 irá chamar para de fato resolver. Por exemplo:

```
int fatorial(int i, int n, int resultado) {
    if (i <= n) {
        resultado *= i;
        i++;
        return fatorial(i, n, resultado);
    } else {
        return resultado;
    }}
int fatorial(int n) { //func chamada na main
    fatorial(1, n, 1);}
```

Listing 7: Exemplo de Função Recursiva e sua Chamada na `main`

Essa prática é muito bem vista e aplicada para facilitar a vida do programador em saber quais parâmetros mandar para a função.No caso em algumas linguagens nos permite privar a função, e então fica claro qual devemos chamar