

RECURSIVIDADE



Prof. Cristiane

Recursividade

Uma sub-rotina que chama a si mesma.

Exemplo:

```
int fpotencia(int numero, int potencia)
{
    potencia--;
    printf("\n-> funcao fpotencia");
    return numero*fpotencia(numero,potencia);
}
```

Condição de Paragem

Como uma função recursiva pode chamar-se a si mesma infinitamente, é essencial que exista uma **condição** que determine o fim do processo recursivo.

Condição -> determina o momento em que a função deverá **parar de chamar a si mesma.**

Condição de Paragem

```
int fpotencia(int numero, int potencia)
{
    if (potencia<=1)
        return numero;
    else
    {
        potencia--;
        printf("\n-> funcao fpotencia");
        return  numero*fpotencia(numero,potencia);
    }
}
```

Implementação

Duas partes fundamentais:

- **O caso elementar ou condição de Paragem:** não há necessidade de chamar uma nova instância da função
- **Regra Geral:** repetição do processo até atingir o ponto de paragem

Como funciona a recursividade

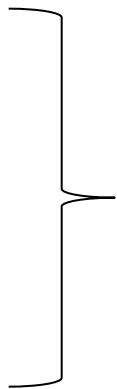
- Implementação através de pilha

Armazenam-se os dados usados na chamada de um procedimento que ainda não terminou de processar.

Como funciona a recursividade

```
if (potencia<=1)
    return numero;
else
{
    potencia--;
    return  numero*fpotencia(numero,potencia);
}
```

3
3*
3*
3*



pilha

Como funciona a recursividade

```
void escreveasterisco(int num)
{

    if (num<=1)
        printf("*");
    else
    {
        printf("*");
        num--;
        escreveasterisco(num);
    }
}
```


Como funciona a recursividade

Na recursividade, a cada chamada, uma nova cópia da função é executada, o que implica que as suas variáveis locais são independentes das variáveis locais da primeira cópia, e não podem relacionar-se diretamente.

Exemplo

Escrever uma função recursiva que calcule a soma dos números entre 1 e n.

```
int Soma (int n)
{
    if (n > 1)
        return n + Soma(n-1);
    else
        return 1;
}
```

1

2 + Soma(1)

3 + Soma(2)

4 + Soma(3)

Soma(4)

Vantagens

- Simplificação de alguns problemas;
Pode-se obter um código mais conciso e eficaz nessas situações;
- Uma solução recursiva pode, por outro lado, eliminar a necessidade de manter o **controle manual sobre uma série de variáveis** normalmente associadas aos métodos alternativos à recursividade.

Desvantagens

- As funções recursivas são geralmente mais lentas e ocupam mais memória do que as funções iterativas equivalentes.
- Um erro de implementação pode levar ao esgotamento dos recursos associados à pilha que gere a chamada a funções.

Quando usar

- Recursividade: definição mais clara e precisa dos algoritmos quando o problema é por natureza recursivo ou utiliza estruturas recursivas como as árvores.

Quando não usar recursividade

- Nem sempre usar um caminho recursivo é a melhor maneira de resolver um problema.
- Deve-se considerar o recurso que será consumido através das inúmeras chamadas à função.

Quando não usar recursividade

- Alguns problemas que são resolvidos por recursividade podem ser implementados usando laços de repetição (while ou for).

Bibliografia

Faculdade de Ciências e Tecnologia da
Universidade de Coimbra,
ESTRUTURAS DE DADOS E
ALGORITMOS, Ficha 3, 2004.

Ziviane, Nivio. Projeto de Algoritmos
com implementações em PASCAL e C.
São Paulo: Editora Thomson, 2007. 2ª
edição.