Funções recursivas

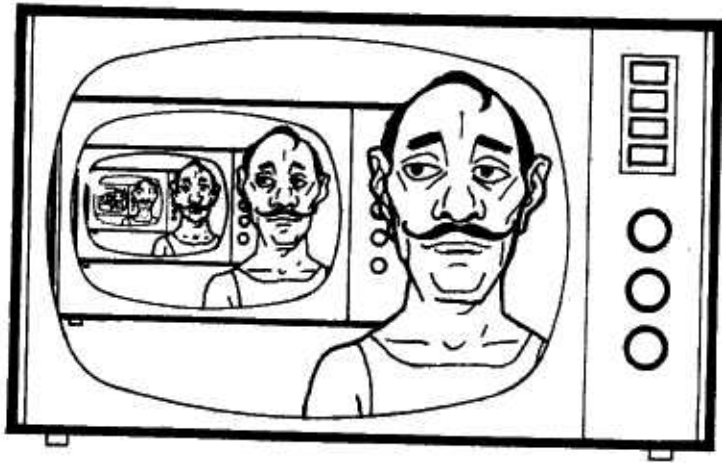
Disciplina: PROGRAMAÇÃO II

Prof. Jean Eduardo Glazar
Curso de Sistemas de Informação
Campus Colatina-ES

INSTITUTO FEDERAL
Espírito Santo

Definição

- Um objeto é dito recursivo se ele consistir parcialmente ou for definido em termos de si próprio. Recursividade é encontrada não apenas em matemática mas também no dia a dia. Quem nunca viu uma figura que contenha a si própria?



Definição

- Uma função recursiva é aquela que "chama" ela mesma.
- Muitos problemas têm a seguinte propriedade: cada instância — ou seja, cada exemplo concreto — do problema contém uma instância menor do mesmo problema. Dizemos que esses problemas têm estrutura recursiva.
- Para resolver um tal problema podemos aplicar o seguinte método:
  - se o problema é pequeno:
    - resolva-o diretamente (use força bruta se necessário);
  - se o problema é grande:
    - reduza-o a uma versão menor do mesmo problema;
    - aplique o método ao problema menor.

Exemplo clássico: Fatorial

- Aplicando a técnica para a função fatorial: ***n! (n >= 0)***
  - se o problema é pequeno:
    - resolva-o diretamente (use força bruta se necessário);
      - ***0! = 1***
  - se o problema é grande:
    - reduza-o a uma versão menor do mesmo problema;
    - aplique o método ao problema menor.
      - ***n! = n . (n-1)!***

Exemplo clássico: Fatorial

- De acordo com a definição, o fatorial do número 5, expressado por **5!**, seria 5 x 4 x 3 x 2 x 1 = 120. Vamos ver como a definição recursiva da função fatorial pode ser aplicada neste caso.
- Pela nossa definição recursiva (slide anterior), **5!** = 5 x **4!**.
- Então antes de avaliar **5!**, devemos primeiro avaliar **4!**. Usando a definição mais uma vez, vemos que **4!** = 4 x **3!**. Então devemos avaliar **3!**. Repetindo o processo nos temos que:

Exemplo clássico: Fatorial

1. **5!** = 5 x **4!**
2. **4!** = 4 x **3!**
3. **3!** = 3 x **2!**
4. **2!** = 2 x **1!**
5. **1!** = 1 x **0!**
6. **0! = 1**

Exemplo clássico: Fatorial

1. **5!** = 5 x **4!** = 5 x **24** **= 120**
2. **4!** = 4 x **3!** = 4 x **6** = **24**
3. **3!** = 3 x **2!** = 3 x **2** = **6**
4. **2!** = 2 x **1!** = 2 x **1** = **2**
5. **1!** = 1 x **0!** = 1 x **1** = **1**
6. **0! = 1**

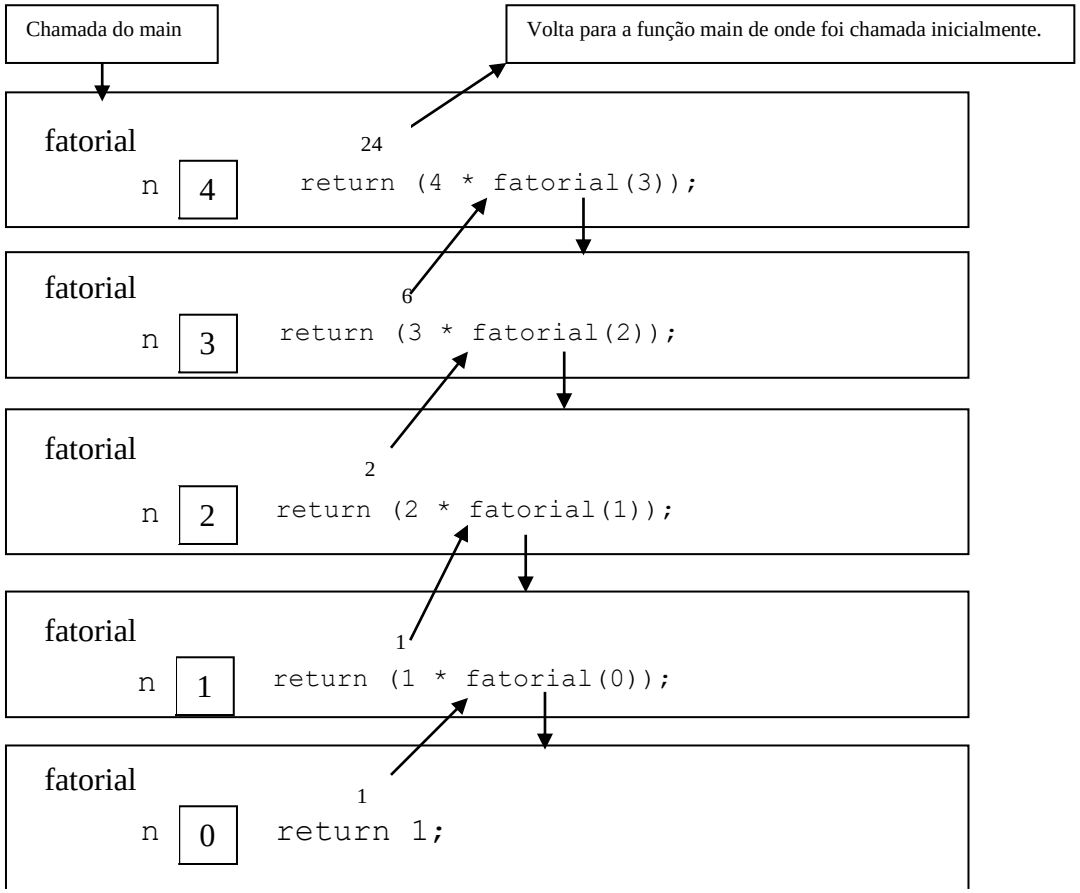
Fatorial: função iterativa

```
int fatorial (int n) {
        int i, fat = 1;
        for (i=1; i <= n; i++) {
            fat = fat * i;
        }
        return fat;
}
```

Fatorial: função recursiva

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

Exemplo: fatorial(4);



Pontos importantes

- A recursão é um recurso interessante que várias linguagens possuem, mas deve ser utilizado com cuidado.
- A primeira e mais importante coisa a se providenciar é um **critério de parada**. Este vai determinar quando a função deverá parar de chamar a si mesma. Isto impede que a função se chame infinitas vezes.
- É importante salientar, que a versão recursiva de uma determinada função de um modo geral é, computacionalmente, mais "pesada" que a versão iterativa (sem recursividade), pois a forma recursiva realiza várias chamadas de função.

Pontos importantes

- No entanto, em algumas implementações como a de árvores binárias, que são estruturas de dados que serão vistas em conceitos mais avançados de programação, as versões recursivas de suas operações, além de mais naturais, são mais simples de se implementar.
- Portanto, o programador deve ter sempre em mente esse balanço entre **facilidade de implementar** e **esforço computacional**, para fazer a opção do uso ou não de recursividade

Exemplo: Multiplicação usando somas

- A multiplicação entre dois **números naturais** pode ser resolvida utilizando várias somas.
- A multiplicação **a x b**, onde **a** e **b** são inteiros positivos, pode ser definido como **a** adicionado a ele mesmo **b** vezes.

A versão recursiva desta definição seria:

Se **b == 0**: **a x b = 0**

Se **b == 1**: **a x b = a**

Se **b > 1**: **a x b = a + a x (b - 1)**

Exemplo: Multiplicação usando somas

```
int multiplicacao (int a, int b) {
        if ( a == 0 || b == 0 ) {
            return 0;
        }
        else if ( b == 1) {
            return a;
        }
        else {
            return (a + multiplicacao (a, b-1) );
        }
}
```

Exercícios

1) Implemente a função recursiva para determinar o ***n-ésimo*** termo da sequência de Fibonnacci, onde os dois primeiros termos são 1 e 1, e os demais são a soma dos dois termos anteriores.
2) Implemente a função de Fibonacci do exercício 1 de forma iterativa.
3) Compare o tempo de execução dessas duas funções quando n = 40.
4) Implemente uma função recursiva para encontrar o maior elemento de um vetor de números inteiros.

INSTITUTO
FEDERAL
Espírito Santo