

2º Semestre

Técnicas de Programação

Luiz Fernando Carvalho

luizfcarvalhoo@gmail.com

Introdução

- Uma função é dita **recursiva** quando dentro do seu código existe uma **chamada para si mesma**.

```
1  int fatorial(int n){
2      int fat;
3      if(n <= 1)
4          return 1;
5      else{
6          fat = n * fatorial(n-1);
7          return fat;
8      }
9  }
10
11 int main(){
12     int n=3, resultado;
13     resultado = fatorial(n);
14     printf("%d", resultado);
15
16     return 0;
17 }
```

Recursão

- A recursão é uma técnica que define um problema em termos de um ou mais versões menores deste mesmo problema;
- Portanto, essa ferramenta pode ser utilizada sempre que for possível expressar a solução de um problema em função do próprio problema;

Homem capota de Uno ao tentar ajudar irmão que capotou de Uno

25 julho, 2018

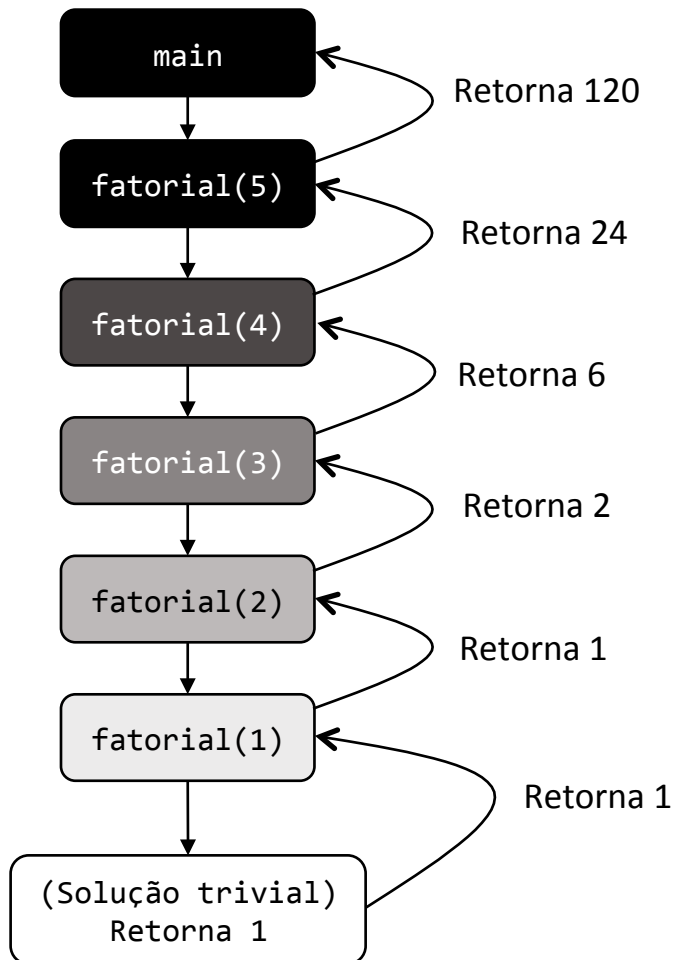


Recursão

```
1  int fatorial(int n){
2      if(n == 0)
3          return 1;
4      else if(n < 0){
5          exit(0);
6      }
7      return n * fatorial(n-1);
8  }
9
10 int main(){
11     int n=3, resultado;
12     resultado = fatorial(n);
13     printf("%d", resultado);
14
15     return 0;
16 }
```

```
fatorial(5)
=> return 5 * fatorial(4)
=> return 4 * fatorial(3)
=> return 3 * fatorial(2)
=> return 2 * fatorial(1)
=> return 1 * fatorial(0)
=> 0 == 0
<= return 1
<= return 1 * 1 → (1)
<= return 2 * 1 → (2)
<= return 3 * 2 → (6)
<= return 4 * 6 → (24)
<= return 5 * 24 → (120)
```

Recursão



```
fatorial(5)
=> return 5 * fatorial(4)
=> return 4 * fatorial(3)
=> return 3 * fatorial(2)
=> return 2 * fatorial(1)
=> return 1 * fatorial(0)
=> 0 == 0
<= return 1
<= return 1 * 1 → (1)
<= return 2 * 1 → (2)
<= return 3 * 2 → (6)
<= return 4 * 6 → (24)
<= return 5 * 24 → (120)
```

Recursão

- Em uma função recursiva pode ocorrer um problema de terminação do programa, como um *loop* infinito;
- Para determinar a terminação das repetições, deve-se:
 - Definir uma função que implica em uma condição de terminação (solução trivial);
 - Provar que a função decresce a cada iteração, permitindo que, eventualmente, esta solução trivial seja atingida;

Um exemplo de recursividade. Dada uma função que recebe um parâmetro **par**:

funcao(**par**)

- Teste de término de recursão utilizando **par**
 - Se teste for verdadeiro, retornar a solução final
- Processamento
 - Aqui a função processa as informações baseado em **par**
- Chamada recursiva utilizando **par**
 - **par** deve ser modificado para fazer a recursão chegar a um término

Recursão - Exemplo

- Exemplo: soma N primeiros números inteiros

$$S(N) = \begin{cases} 1, & \text{se } N = 1 \text{ (solução trivial)} \\ S(N-1) + N, & \text{se } N > 1 \text{ (chamada recursiva)} \end{cases}$$

```
1  int soma(int n){
2      if(n == 1)
3          return 1;
4      else
5          return n+soma(n-1);
6  }
7
8  int main(){
9      int n;
10
11     printf("Forneca o n: ");
12     scanf("%d", &n);
13     printf("Resultado: %d", soma(n));
14     return 0;
15 }
```

Solução trivial

Chamada recursiva

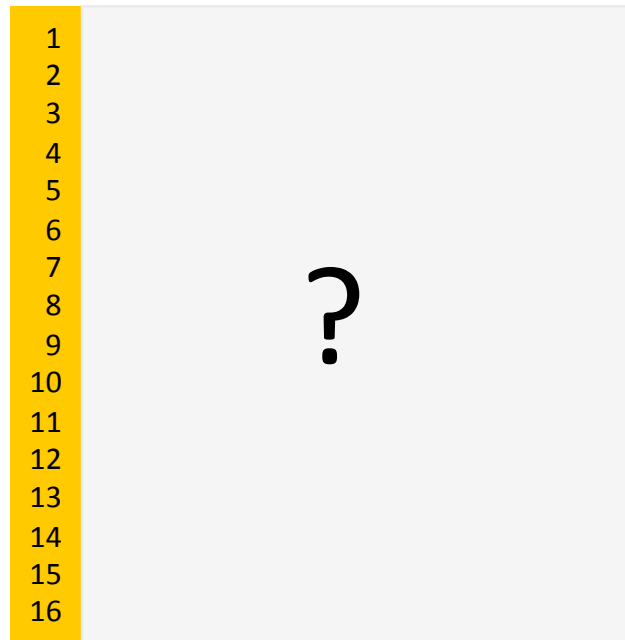
Recursão - Exemplo

- Exemplo: Fibonacci

$$S(N) = \begin{cases} 1, \text{ se } N = 1 \\ 1, \text{ se } N = 2 \\ \text{fib}(N - 1) + \text{fib}(N - 2), \text{ se } N > 2 \end{cases}$$

\rightarrow Soluções triviais

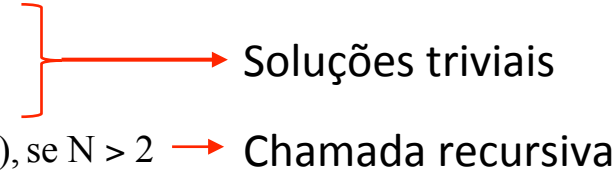
\rightarrow Chamada recursiva



Recursão - Exemplo

- Exemplo: Fibonacci

$$S(N) = \begin{cases} 1, \text{ se } N = 1 \\ 1, \text{ se } N = 2 \\ \text{fib}(N-1) + \text{fib}(N-2), \text{ se } N > 2 \end{cases}$$



Soluções triviais

Chamada recursiva

```
1 int fib(int n){
2     if(n == 1)
3         return 1;
4     if(n == 2)
5         return 1;
6     return fib(n-1)+fib(n-2);
7 }
8
9 int main(){
10     int n;
11
12     printf("Forneca o n: ");
13     scanf("%d", &n);
14     printf("Resultado: %d", fib(n));
15     return 0;
16 }
```

Recursão

- Um algoritmo recursivo é mais elegante e menor que a sua versão iterativa;
- Exibe maior clareza no processo desempenhado pelo algoritmo;
- Elimina a necessidade de manter o controle manual sobre uma série de variáveis normalmente associadas aos métodos iterativos como, por exemplo, contadores e acumuladores;
- Por outro lado, um programa recursivo tende a exigir mais espaço de memória;
- Pode ser mais lento do que a versão iterativa, pois vai preenchendo a pilha com chamadas para a própria função;
- Pode se tornar mais difícil para depurar;

Exercícios

1. Escreva uma função recursiva para calcular o valor de uma base **x** elevada a um expoente **y**;

Solução trivial: $x^0=1$

Passo da recursão: $x^n = x * x^{n-1}$

2. Usando recursividade, calcule a soma de todos os valores de um *array* de inteiros;

Solução trivial: Tamanho do array = 0. Soma é 0.

Passo da recursão: $v[n-1] + \text{soma do restante do array}$

3. Dado um vetor de inteiros e o seu número de elementos, inverta a posição dos seus elementos

Solução trivial: Tamanho do array menor ou igual a 1

Passo da recursão: troca o 1º e o último elementos e inverte o resto do array

Exercícios

```
void inverte(int v[], int esq, int dir){  
    int t;  
    if (esq >= dir)  
        return;  
    t = v[esq];  
    v[esq] = v[dir];  
    v[dir] = t;  
    inverte(v, esq+1, dir-1);  
}
```

Exercícios

4. Um problema típico em ciência da computação consiste em converter um número da sua forma decimal para binária. Crie um algoritmo recursivo para resolver esse problema.

Solução trivial: **x=0** quando o número inteiro já foi convertido para binário

Passo da recursão: **saber como $x/2$ é convertido. Depois, imprimir um dígito (0 ou 1) dado o sucesso da divisão.**

5. Considere a função

```
int X(int a){  
    if(a <= 0) return 0;  
    else return a + X(a-1);  
}
```

O que essa função faz? Escreva uma função não-recursiva que resolve o mesmo problema

Referências

1. Notas de aula profa. Silvana M. A. de Lara. Universidade de São Paulo – São Carlos. ICMC.