BCC702 - Programação de Computadores II

Recursividade

Prof
Profª Valéria de Carvalho Santos

Recursividade

- Método para resolver problemas que são definidos a partir deles mesmos.

Exemplo

- Fatorial

```
0! = 1
1! = 1
2! = 2*1
3! = 3*2*1
4! = 4*3*2*1
5! = 5*4*3*2*1
…
```

```
0! = 1
1! = 1*0!
2! = 2*1!
3! = 3*2!
4! = 4*3!
5! = 5*4!
…
n! = n*(n-1)!, se n>0
```

Recursividade

- Fatorial de um número natural *n*

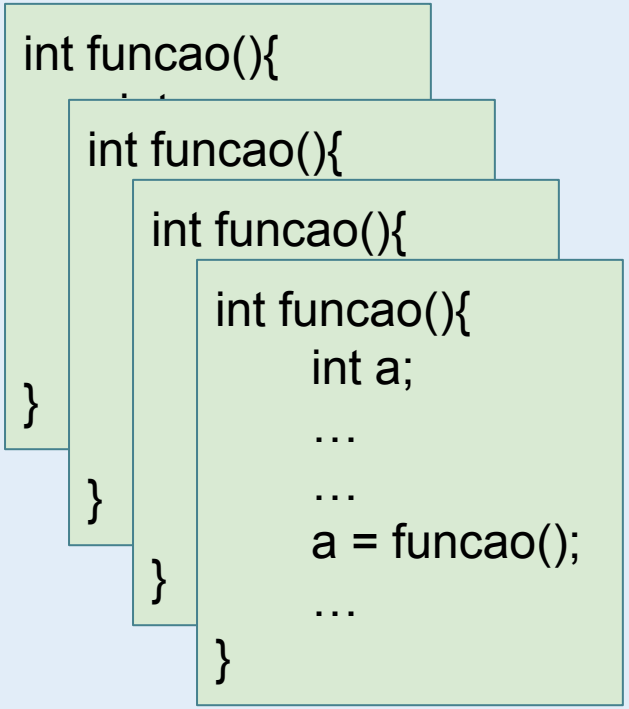
$$n! = \begin{cases} 1, & \text{se } n = 0 \\ n*(n-1)!, & \text{se } n>0 \end{cases}$$

- Isto é, para *n* maior do que zero, a função n! é definida com base nela mesma: definição recursiva

Recursividade

- Em programação, recursividade é implementada como uma função que chama a si mesma, direta ou indiretamente

Recursividade



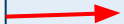
Recursividade

- Uma definição recursiva é constituída de duas partes:
  - *Parte 1*
    - Há um ou mais **casos base** que dizem o que fazer em situações simples; a resposta pode ser dada de imediato;
    - Garante que a recursão eventualmente possa parar.
  - *Parte 2*
    - Há um ou mais **casos recursivos** que são mais gerais e definem a função em termos de uma chamada mais simples a si mesma.

Versão recursiva

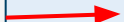
```
#include <iostream>
using namespace std;

int fat(int n){
	if(n==0)
		return 1;

	return n*fat(n-1);
}

int main(){
	int num=5;
	cout << “Fatorial de ” << num << “=” << fat(num) << endl;
	return 0;
}
```

```
int main(){
	cout << fat(5);
}
```



```
int fat(5){
	if(5==0){
		return 1;
	}
	return 5*fat(4);
}
```

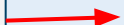
```
int main(){
      cout << fat(5);
}
```



```
int fat(5){
```

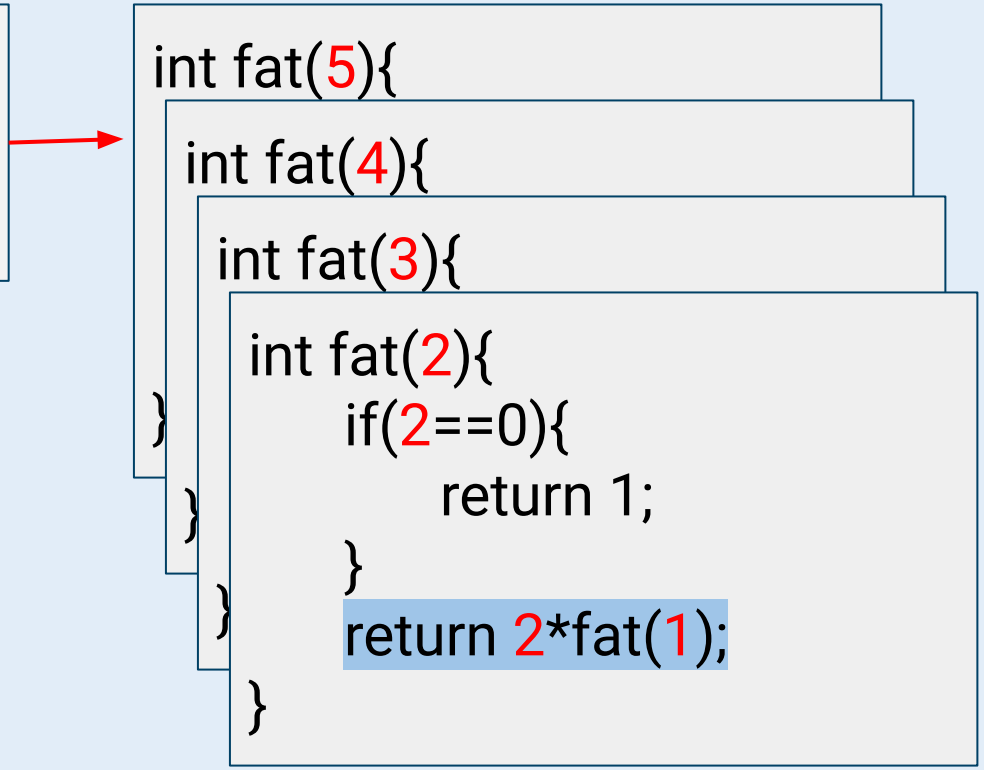
```
int fat(4){
      if(4==0){
            return 1;
      }
      return 4*fat(3);
}
```

```
int main(){
      cout << fat(5);
}
```

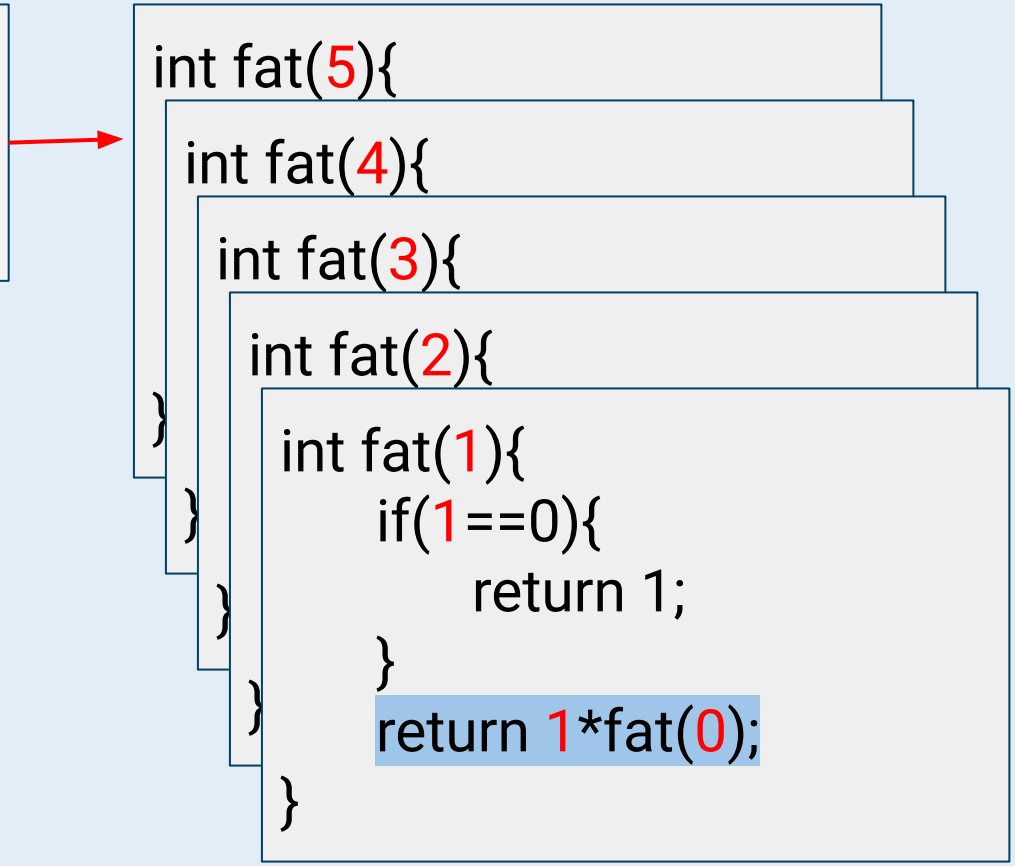


```
int fat(5){
    int fat(4){
        int fat(3){
            if(3==0){
                return 1;
            }
            return 3*fat(2);
        }
    }
}
```

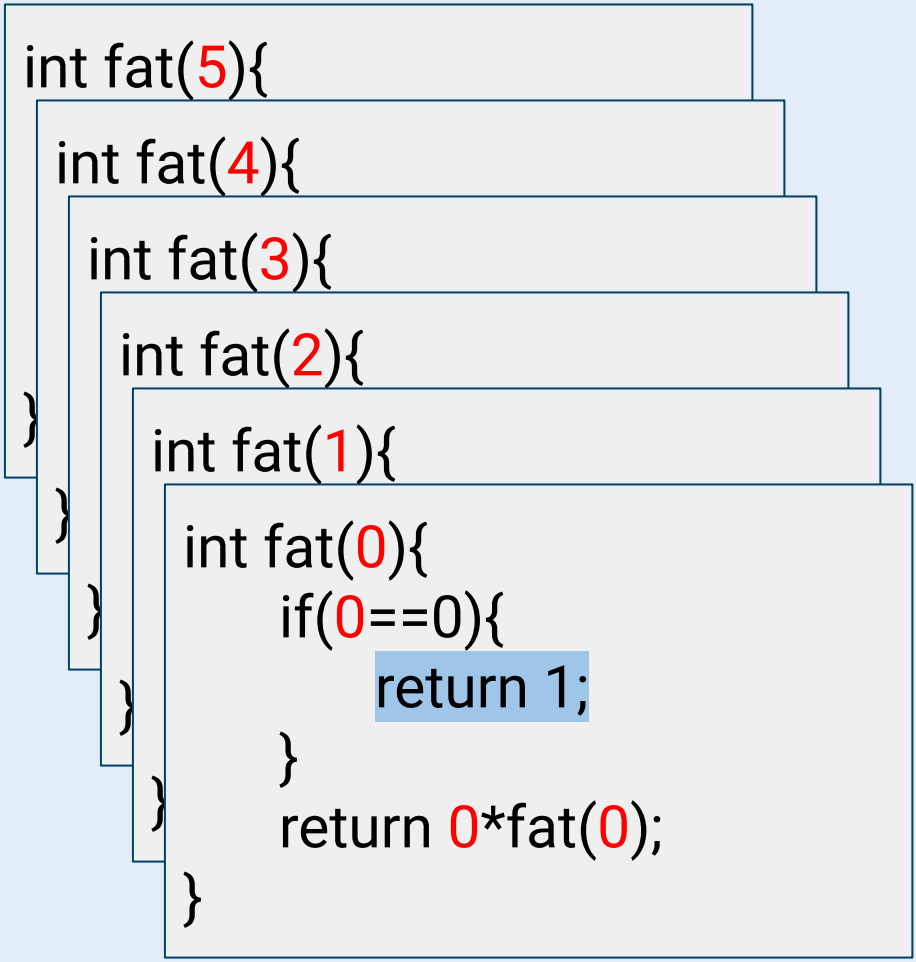
```
int main(){
      cout << fat(5);
}
```



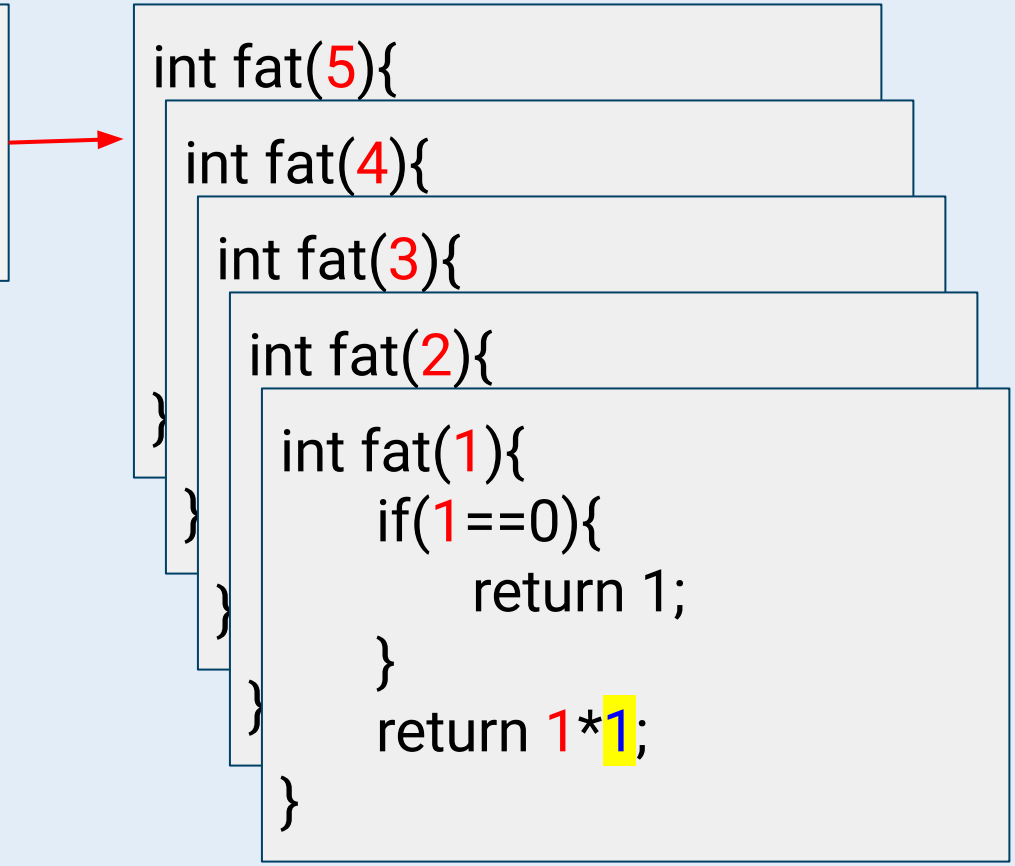
```
int main(){
      cout << fat(5);
}
```



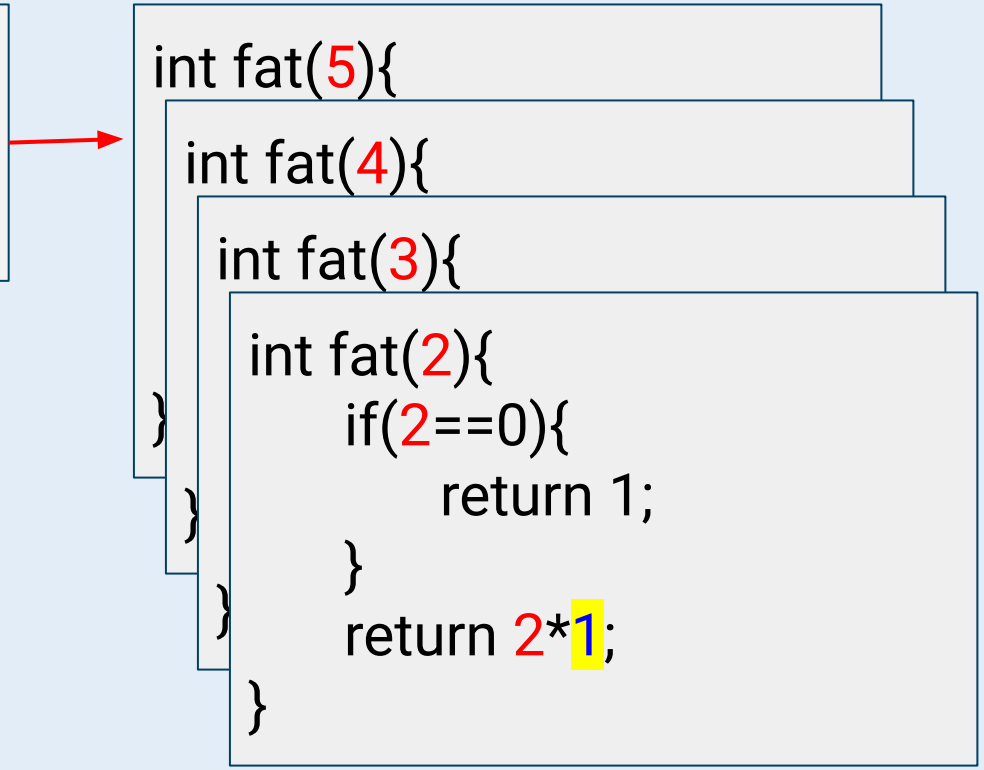
```
int main(){
      cout << fat(5);
}
```



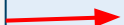
```
int main(){
      cout << fat(5);
}
```



```
int main(){
      cout << fat(5);
}
```



```
int main(){
      cout << fat(5);
}
```



```
int fat(5){
```

```
int fat(4){
```

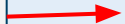
```
int fat(3){
      if(3==0){
            return 1;
      }
      return 3*2;
}
```

```
int main(){
      cout << fat(5);
}
```

```
int fat(5){
}
```

```
int fat(4){
      if(4==0){
            return 1;
      }
      return 4*6;
}
```

```
int main(){
      cout << fat(5);
}
```



```
int fat(5){
      if(5==0){
            return 1;
      }
      return 5*24;
}
```

```
int main(){
	cout << 120;
}
```

Versão iterativa

```
#include <iostream>
using namespace std;

int fat(int n){‘
      int f = 1;
      for(int i = 1; i<=n; i++)
            f = f*i;
      return f;
}

int main(){
      int num=5;
      cout << “Fatorial de ” << num << “=” << fat(num) << endl;
      return 0;
}
```

Recursão x Iteração

	Recursão	Iteração
Vantagens	- Problemas definidos em termos recursivos - Clareza de código	- Melhor desempenho
Desvantagens	- Pode consumir mais recurso computacional	- Menos legível

Exercício 1

- Faça um programa com uma *função não recursiva* que calcula a soma dos números de 1 até *n*, onde n é a entrada do programa.

- Faça um programa com uma *função recursiva* que calcula a soma dos números de 1 até *n*, onde n é a entrada do programa.

Exercício 2

- Faça um programa com uma *função potência,* que recebe como parâmetros a base e o expoente e retorna a potência. A base e o expoente são as entradas do programa.

Exercício 3

- Qual o valor de X(4), se X é dada pelo seguinte código?

https://www.ime.usp.br/~pf/algoritmos/aulas/solucoes/recu1.html

```
int X (int n) {

    if (n == 1 || n == 2)
        return n;

    else
        return X(n-1) + n * X(n-2);
}
```