Faculdade de Computação - FACOM

Bacharelado em Sistemas de Informação

FACOM32201 - Algoritmos e Programação II

Prof. Thiago Pirola Ribeiro

FUNÇÕES

Recursividade

Recursividade

- Um procedimento é dito **recursivo**, se contém, em sua descrição, **uma** ou **mais chamadas** a si **mesmo** (**chamadas recursivas**).

Recursividade

- Um procedimento é dito **recursivo**, se contém, em sua descrição, **uma** ou **mais chamadas** a si **mesmo** (**chamadas recursivas**).

- Naturalmente, todo procedimento recursivo ou não, deve possuir uma chamada de um local exterior a ele, ou seja, uma chamada exterior.

Recursividade

- Um procedimento é dito **recursivo**, se contém, em sua descrição, **uma** ou **mais chamadas** a si **mesmo** (**chamadas recursivas**).
- Naturalmente, todo procedimento recursivo ou não, deve possuir uma chamada de um local exterior a ele, ou seja, uma chamada exterior.
- Alguns problemas são solucionados com mais facilidade com o uso de recursão.

Recursividade

- Um procedimento é dito **recursivo**, se contém, em sua descrição, **uma** ou **mais chamadas** a si **mesmo** (**chamadas recursivas**).
- Naturalmente, todo procedimento recursivo ou não, deve possuir uma chamada de um local exterior a ele, ou seja, uma chamada exterior.
- Alguns problemas são solucionados com mais facilidade com o uso de recursão.
- Geralmente são problemas nos quais fazemos um cálculo com os dados, e depois fazemos novamente o mesmo cálculo com o resultado.

Recursividade

- A recursão pode ter um final feliz, quando a cadeia de recursão chega a um fim e temos um resultado.

Recursividade

- A recursão pode ter um final feliz, quando a cadeia de recursão chega a um fim e temos um resultado.
- Ou pode ter um final infeliz, quando a cadeia recursiva não tem fim e acaba travando o programa.

Recursividade

- A recursão pode ter um final feliz, quando a cadeia de recursão chega a um fim e temos um resultado.
- Ou pode ter um final infeliz, quando a cadeia recursiva não tem fim e acaba travando o programa.
- É importante entender que quando uma função chama a si mesma, uma nova cópia da função passa a ser executada.

Recursividade

- A recursão pode ter um final feliz, quando a cadeia de recursão chega a um fim e temos um resultado.
- Ou pode ter um final infeliz, quando a cadeia recursiva não tem fim e acaba travando o programa.
- É importante entender que quando uma função chama a si mesma, uma nova cópia da função passa a ser executada.
- As variáveis locais da segunda cópia são independentes das variáveis locais da primeira cópia, e não podem afetar umas às outras diretamente.

Recursividade

- Por exemplo, um algoritmo que faz o cálculo do fatorial de um número:

Recursividade

- Por exemplo, um algoritmo que faz o cálculo do fatorial de um número:

$$N! = 1 * 2 * ... * M * ... * N - 1 * N$$

Recursividade

- A função de fatorial é definida como:

  $0! = 1$

  $1! = 1$

  $2! = 1 * 2 = 2$ ou $2! = 2 * 1 = 2$

  $3! = 1 * 2 * 3 = 6$ ou $3! = 3 * 2 * 1 = 6 \ldots$

  $N! = 1 * 2 * 3 * \ldots * (N-1) * N$ ou $N! = N * (N-1)!$

- A função de fatorial pode ser implementada com recursividade ou sem recursividade.

Sem Recursividade

```
1   int fatorial(int N){
2       int i, fat;
3       fat = 1;
4       if (N == 0 || N == 1)
5           fat = 1;
6       else
7           for(i=2; i<=N; i++)
8               fat = fat * i;
9       return(fat);
10  }
```

Com Recursividade

```
1       int Fatorial(int N)
2       {
3           if (N == 0 || N == 1)
4               return(1);
5           else
6               return(N * Fatorial(N-1));
7   }
```

Com Recursividade

```
1  int Fatorial(int N)
2  {
3      if (N == 0 || N == 1)
4          return(1);
5      else
6          return(N * Fatorial(N-1));
7  }
```

Critério de Parada!

Chamada Recursiva!

Recursividade - Fatorial

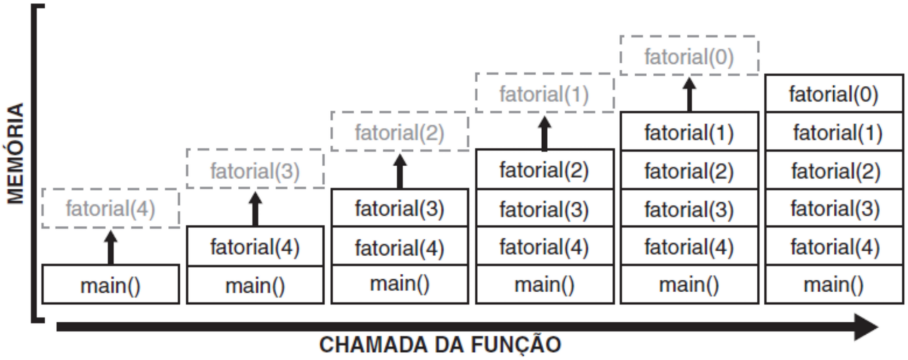
Sem Recursividade

```
int fatorial(int N){
    int i, fat;
    fat = 1;
    if (N == 0 || N == 1)
        fat = 1;
    else
        for(i=2; i<=N; i++)
            fat = fat * i;
    return(fat);
}
```

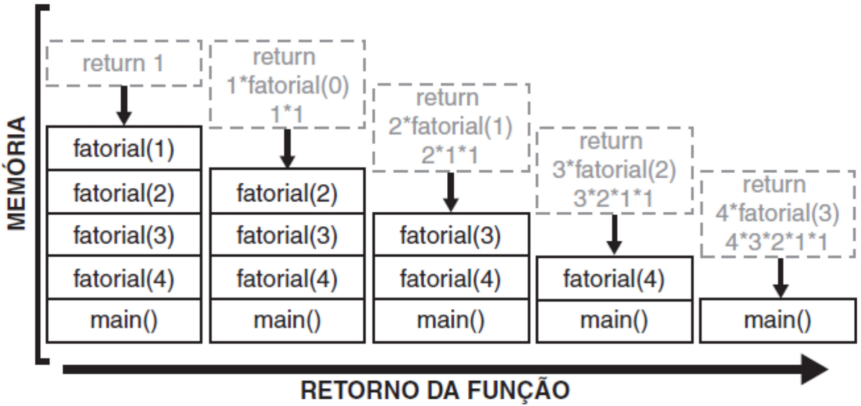
Recursivo

```
int Fatorial(int N)
{
    if (N == 0 || N == 1)
        return(1);
    else
        return(N * Fatorial(N-1));
}
```

Recursividade



Recursividade



Recursividade

- Um outro exemplo clássico de uso de recursão é a sequência matemática chamada série de Fibonacci.

Recursividade

- Um outro exemplo clássico de uso de recursão é a sequência matemática chamada série de Fibonacci.
- Na série de Fibonacci, cada número, a partir do terceiro, é igual à soma dos dois números anteriores.

Recursividade

- Um outro exemplo clássico de uso de recursão é a sequência matemática chamada série de Fibonacci.
- Na série de Fibonacci, cada número, a partir do terceiro, é igual à soma dos dois números anteriores.
- Eis a série de Fibonacci:

  $1, 1, 2, 3, 5, 8, 13, 21, 34...$

Recursividade

- Um outro exemplo clássico de uso de recursão é a sequência matemática chamada série de Fibonacci.
- Na série de Fibonacci, cada número, a partir do terceiro, é igual à soma dos dois números anteriores.
- Eis a série de Fibonacci:

  $1, 1, 2, 3, 5, 8, 13, 21, 34...$
- Geralmente, o que se deseja é determinar qual o n-ésimo número da série.

Recursividade

- Para solucionar o problema, precisamos examinar com cuidado a série de Fibonacci.

Recursividade

- Para solucionar o problema, precisamos examinar com cuidado a série de Fibonacci.
- Os primeiros dois elementos são iguais a 1. Depois disso, cada elemento subsequente é igual à soma dos dois anteriores.

Recursividade

- Para solucionar o problema, precisamos examinar com cuidado a série de Fibonacci.
- Os primeiros dois elementos são iguais a 1. Depois disso, cada elemento subsequente é igual à soma dos dois anteriores.
- Por exemplo, o sétimo número é igual à soma do sexto com o quinto. Ou, dito de um modo genérico, o n-ésimo número é igual à soma do elemento $n - 1$ com o elemento $n - 2$, desde que $n > 2$.

Recursividade

- Para evitar desastres, uma função recursiva precisa ter uma condição de parada.

Recursividade

- Para evitar desastres, uma função recursiva precisa ter uma condição de parada.
- Alguma coisa precisa acontecer para fazer com que o programa encerre a cadeia recursiva, senão ela se tornará infinita.

Recursividade

- Para evitar desastres, uma função recursiva precisa ter uma condição de parada.
- Alguma coisa precisa acontecer para fazer com que o programa encerre a cadeia recursiva, senão ela se tornará infinita.
- Na série de Fibonacci, essa condição é $n < 3$.

Fibonacci - Sem recursividade

```
1  int fibo(int n){
2      int i,f0,f1,f2;
3      i = 2; f0 = 0; f1 = 1;
4      while(i <= n){
5          f2 = f0 + f1;
6          f0 = f1;
7          f1 = f2;
8          i  = i + 1
9      }
10     return(f2);
11 }
```

Fibonacci - Recursiva

```
1 int fibo(int n){
2     if (n <= 1)
3         return(1);
4     else
5         return( fibo(n-1) + fibo(n-2));
6 }
```

Recursividade - Fibonacci

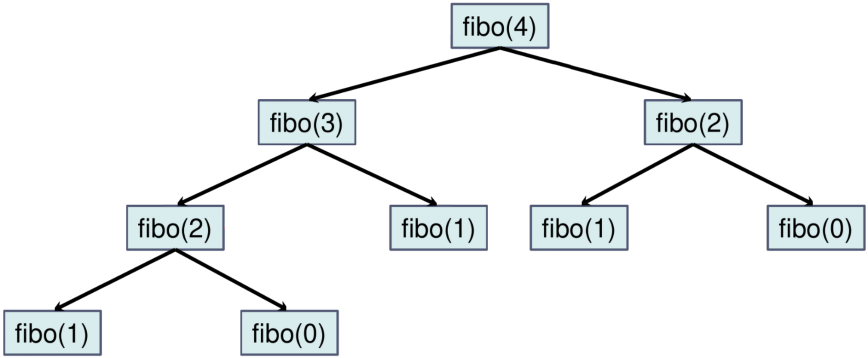
Sem Recursividade

```
1  int fibo(int n){
2      int i,f0,f1,f2;
3      i = 2; f0 = 0; f1 = 1;
4      while(i <= n){
5          f2 = f0 + f1;
6          f0 = f1;
7          f1 = f2;
8          i  = i + 1
9      }
10     return(f2);
11 }
```

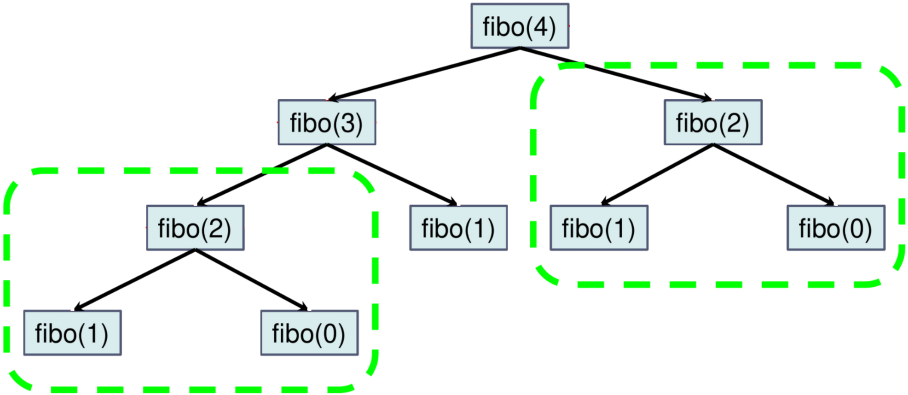
Recursivo

```
1 int fibo(int n){
2     if (n <= 1)
3         return(1);
4     else
5         return( fibo(n-1) + fibo(n-2));
6 }
```

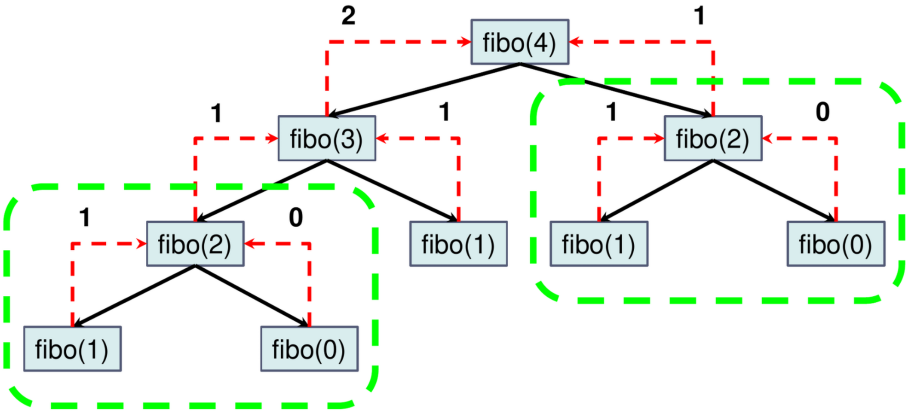
Recursividade - Fibonacci



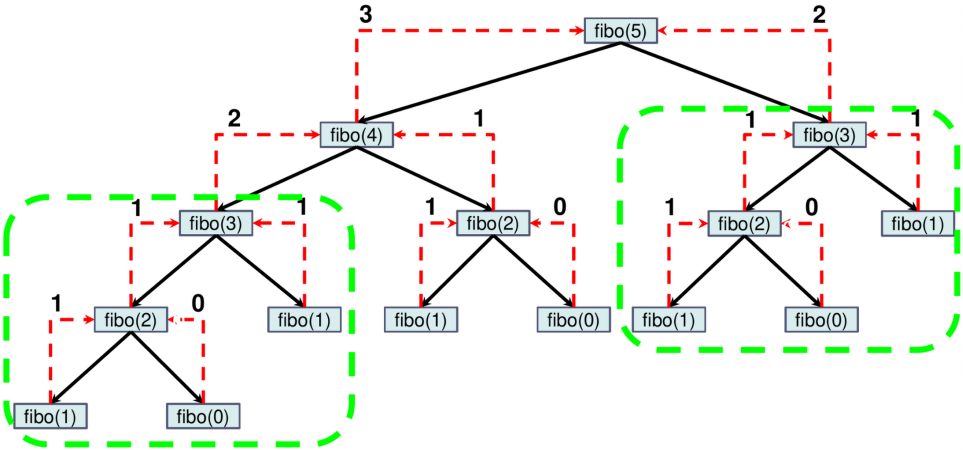
Recursividade - Fibonacci



Recursividade - Fibonacci



Recursividade - Fibonacci



Recursividade vs Não Recursividade

- Quando a natureza do problema é estritamente recursiva, pois assim, diminui as chances de erros na programação do algoritmo;

Recursividade vs Não Recursividade

- Quando a natureza do problema é estritamente recursiva, pois assim, diminui as chances de erros na programação do algoritmo;
- Porém, a recursão requer uma grande demanda de memória para efetuar as chamadas recursivas, podendo gerar stack overflow

Recursividade vs Não Recursividade

- Quando a natureza do problema é estritamente recursiva, pois assim, diminui as chances de erros na programação do algoritmo;
- Porém, a recursão requer uma grande demanda de memória para efetuar as chamadas recursivas, podendo gerar stack overflow
- Programas recursivos podem ser convertidos em programas não-recursivos.

Recursividade vs Não Recursividade

- Quando a natureza do problema é estritamente recursiva, pois assim, diminui as chances de erros na programação do algoritmo;
- Porém, a recursão requer uma grande demanda de memória para efetuar as chamadas recursivas, podendo gerar stack overflow
- Programas recursivos podem ser convertidos em programas não-recursivos.
- Funções não-recursivas costumam ser mais eficazes que as funções recursivas, menos quando é necessário efetuar múltiplas comparações e tentativas;

Recursividade vs Não Recursividade

- Quando a natureza do problema é estritamente recursiva, pois assim, diminui as chances de erros na programação do algoritmo;
- Porém, a recursão requer uma grande demanda de memória para efetuar as chamadas recursivas, podendo gerar stack overflow
- Programas recursivos podem ser convertidos em programas não-recursivos.

- Funções não-recursivas costumam ser mais eficazes que as funções recursivas, menos quando é necessário efetuar múltiplas comparações e tentativas;
- Funções não-recursivas não provocam *stack overflow*;

Recursividade vs Não Recursividade

- Quando a natureza do problema é estritamente recursiva, pois assim, diminui as chances de erros na programação do algoritmo;
- Porém, a recursão requer uma grande demanda de memória para efetuar as chamadas recursivas, podendo gerar stack overflow
- Programas recursivos podem ser convertidos em programas não-recursivos.

- Funções não-recursivas costumam ser mais eficazes que as funções recursivas, menos quando é necessário efetuar múltiplas comparações e tentativas;
- Funções não-recursivas não provocam *stack overflow*;
- Programas não-recursivos nem sempre podem ser convertidos para programas recursivos.

Exercícios

1. Elabore uma função recursiva para realizar a função de exponenciação ab, onde a e b são inteiros:
   `eleva(2,5) - retorna 32`
   `eleva(3,4) - retorna 81`

2. Crie uma função recursiva para efetuar somatório de 1 até n:
   `Soma(8) - retorna 36 (1+2+3+4+5+6+7+8 = 36)`

Faculdade de Computação - FACOM

Bacharelado em Sistemas de Informação

Prof. Thiago Pirola Ribeiro