



## SCC-201 - Capítulo 2

### Recursão

João Luís Garcia Rosa<sup>1</sup>

<sup>1</sup>Departamento de Ciências de Computação  
Instituto de Ciências Matemáticas e de Computação  
Universidade de São Paulo - São Carlos  
<http://www.icmc.usp.br/~joaoluis>

2016

# Sumário

- 1 Recursividade
  - Definições Recursivas
  - Recursão
  - Exemplos
- 2 Busca Binária
  - Busca Binária
  - Exemplo
  - Escrevendo Programas Recursivos
- 3 Simulação da Recursão
  - Soluções Iterativas para Problemas Recursivos
  - O que acontece quando uma função é chamada?
  - O que acontece quando uma função é retornada?

# Sumário

- 1 **Recursividade**
  - Definições Recursivas
    - Recursão
    - Exemplos
- 2 **Busca Binária**
  - Busca Binária
  - Exemplo
  - Escrevendo Programas Recursivos
- 3 **Simulação da Recursão**
  - Soluções Iterativas para Problemas Recursivos
  - O que acontece quando uma função é chamada?
  - O que acontece quando uma função é retornada?

# Definições Recursivas e Processos

- Na matemática, vários objetos são definidos através de um processo que os produz.
- Por exemplo,  $\pi$  é definido como a razão entre a circunferência de um círculo e seu diâmetro.
- Outro exemplo é a função fatorial.
- Dado um número inteiro positivo  $n$  o fatorial de  $n$  é o produto de todos os inteiros entre 1 e  $n$ :
  - $n! = 1$ , se  $n = 0$ .
  - $n! = n * (n - 1) * (n - 2) * \dots * 1$ , se  $n > 0$ .

- Programa **iterativo** para o fatorial de  $n$ :

```
prod = 1;
for (x = n; x > 0; x--)
    prod *= x;
return (prod);
```

# Definições Recursivas e Processos

- O algoritmo anterior é chamado de iterativo pois requer a repetição explícita de um processo até que determinada condição seja satisfeita.
- Este algoritmo pode ser traduzido para uma função em C que retorne  $n!$  quando recebe  $n$  como parâmetro
- Outra definição para o fatorial:
  - $n! = 1$ , se  $n = 0$ .
  - $n! = n * (n - 1)!$ , se  $n > 0$ .
- Observe que a função fatorial está definida em termos de si mesma: o fatorial de  $n$  depende do fatorial de  $n - 1$ . Esta é uma definição **recursiva** do fatorial.

# Definições Recursivas e Processos

- Programa recursivo para cálculo do fatorial:

```
unsigned long fat(int n)
{
    int fato = 1;
    if (n > 1)
        fato = n * fat(n-1);
    return fato;
}
```

# Sumário

1

## Recursividade

- Definições Recursivas
- **Recursão**
- Exemplos

2

## Busca Binária

- Busca Binária
- Exemplo
- Escrevendo Programas Recursivos

3

## Simulação da Recursão

- Soluções Iterativas para Problemas Recursivos
- O que acontece quando uma função é chamada?
- O que acontece quando uma função é retornada?

# Recursão

- A recursão ocorre quando uma função chama a si própria. Quando isto acontece, várias ações ocorrem:
  - A função começa a execução do seu primeiro comando cada vez que é chamada;
  - Novas e distintas cópias dos parâmetros passados por valor e variáveis locais são criadas;
  - A posição que chama a função é colocada em estado de espera, enquanto que o nível gerado recursivamente esteja executando.
- O próximo programa explica estes efeitos.



# Sumário

## 1 Recursividade

- Definições Recursivas
- Recursão
- Exemplos

## 2 Busca Binária

- Busca Binária
- Exemplo
- Escrevendo Programas Recursivos

## 3 Simulação da Recursão

- Soluções Iterativas para Problemas Recursivos
- O que acontece quando uma função é chamada?
- O que acontece quando uma função é retornada?

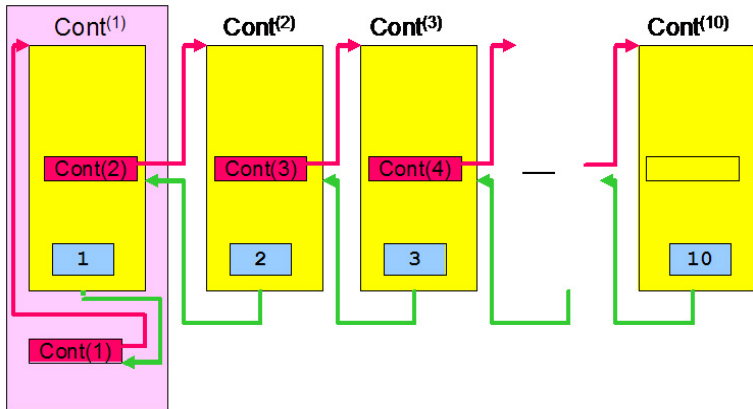
# Recursão

- Contador recursivo:

```
#include <stdio.h>
void cont (int);
main()
{
    cont(1);
}
void cont(int n)
{
    if (n < 10)
        cont(n+1);
    printf("%d\n", n);
}
```

# Recursão

## ContRecursivo



# Recursão

- A **seqüência de Fibonacci** é a seqüência de inteiros:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

- Cada elemento nesta seqüência é a soma dos dois elementos anteriores. Se  $fib(0) = 0$  e  $fib(1) = 1$ :

- $fib(n) = n$ , se  $n = 0$  ou  $n = 1$
- $fib(n) = fib(n - 2) + fib(n - 1)$ , se  $n \geq 2$

- Por exemplo, para calcular  $fib(6)$ , pode-se aplicar a definição recursivamente:

$$\begin{aligned}
 fib(6) &= fib(4) + fib(5) = fib(2) + fib(3) + fib(5) = \\
 &fib(0) + fib(1) + fib(3) + fib(5) = 0 + 1 + fib(3) + fib(5) = \\
 &1 + fib(1) + fib(2) + fib(5) = 1 + 1 + fib(0) + fib(1) + fib(5) = \\
 &2 + 0 + 1 + fib(5) = 3 + fib(3) + fib(4) = \\
 &3 + fib(1) + fib(2) + fib(4) = 3 + 1 + fib(0) + fib(1) + fib(4) = \\
 &4 + 0 + 1 + fib(4) = 5 + fib(2) + fib(3) = \\
 &5 + fib(0) + fib(1) + fib(3) = 5 + 0 + 1 + fib(3) = \\
 &6 + fib(1) + fib(2) = 6 + 1 + fib(0) + fib(1) = 7 + 0 + 1 = 8
 \end{aligned}$$

# Recursão

- Seqüência de Fibonacci:

```
int Fib(int n)
{
    if (n < 2)
        return n;
    else
        return Fib(n-2) + Fib(n-1);
}
```

# Sumário

1

## Recursividade

- Definições Recursivas
- Recursão
- Exemplos

2

## Busca Binária

- Busca Binária
- Exemplo
- Escrevendo Programas Recursivos

3

## Simulação da Recursão

- Soluções Iterativas para Problemas Recursivos
- O que acontece quando uma função é chamada?
- O que acontece quando uma função é retornada?

# Busca Binária

- **Busca binária** consiste em buscar um elemento em um vetor classificado, dividindo-o em duas metades
- Caso o elemento procurado seja o elemento do meio do vetor, a operação é terminada
- Caso o elemento procurado seja menor que o elemento do meio do vetor, procura-se na metade inferior
- Caso o elemento procurado seja maior que o elemento do meio do vetor, procura-se na metade superior
- Trata-se portanto de um procedimento recursivo: o método de busca é definido em termos de si mesmo, com um vetor menor como entrada
- A busca terminará quando o tamanho deste vetor for 1 (operação não recursiva)

# Sumário

1

## Recursividade

- Definições Recursivas
- Recursão
- Exemplos

2

## Busca Binária

- Busca Binária
- Exemplo
- Escrevendo Programas Recursivos

3

## Simulação da Recursão

- Soluções Iterativas para Problemas Recursivos
- O que acontece quando uma função é chamada?
- O que acontece quando uma função é retornada?



# Busca Binária

- Busca binária:

```

1:  if (baixo > alto)
2:    return(-1);
3:  meio = (baixo + alto)/2;
4:  if (x == a[meio])
5:    return(meio);
6:  if (x < a[meio])
7:    procura x em a[baixo] até a[meio-1];
8:  else
9:    procura x em a[meio+1] até a[alto];

```

- Exemplo: seja o seguinte vetor:

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
12	25	33	37	<b>48</b>	57	86	92

- Suponha que se queira procurar o elemento 48, isto é,  $x = 48$ , entre o item 0 (*baixo* = 0) e 7 (*alto* = 7).

# Busca Binária

- Aplicando o algoritmo, tem-se:
  - linha 1:  $0 > 7$ ? Falso, executa linha 3
  - linha 3:  $meio = (0 + 7)/2 = 3$
  - linha 4:  $x = a[3]$ ? 48 não é igual a 37, executa linha 6
  - linha 6:  $x < a[3]$ ? 48 não é menor que 37, executa linha 9
  - linha 9: repita com  $baixo = meio + 1 = 4$  e  $alto = 7$
  - linha 1:  $4 > 7$ ? Falso, executa linha 3
  - linha 3:  $meio = (4 + 7)/2 = 5$
  - linha 4:  $x = a[5]$ ? 48 não é igual a 57, executa linha 6
  - linha 6:  $x < a[5]$ ? 48 é menor que 57, executa linha 7
  - linha 9: repita com  $baixo = 4$  e  $alto = meio - 1 = 4$
  - linha 1:  $4 > 4$ ? Falso, executa linha 3
  - linha 3:  $meio = (4 + 4)/2 = 4$
  - linha 4:  $x = a[4]$ ? 48 é igual a 48, retorna  $meio = 4$  como resposta.

# Sumário

1

## Recursividade

- Definições Recursivas
- Recursão
- Exemplos

2

## Busca Binária

- Busca Binária
- Exemplo
- Escrevendo Programas Recursivos

3

## Simulação da Recursão

- Soluções Iterativas para Problemas Recursivos
- O que acontece quando uma função é chamada?
- O que acontece quando uma função é retornada?

# Propriedades das Definições Recursivas

- Recursão significa repetição.
- Da mesma forma que num comando repetitivo a parada é uma preocupação, o mesmo ocorre com a recursão.
- A função recursiva  $f$  deve ser definida de modo a não envolver  $f$  para um argumento ou grupo de argumentos.
- Esta será a “saída” da seqüência de chamadas recursivas.
- Nos exemplos vistos, as partes **não**-recursivas das definições são:

- fatorial:  $0! = 1$
- seqüência de Fibonacci:  $fib(0) = 0$ ;  $fib(1) = 1$
- busca binária:

```
if (baixo > alto)
    return(-1);
if (x == a[meio])
    return(meio);
```

# Escrevendo Programas Recursivos

- O desenvolvimento de uma solução recursiva **sem** o algoritmo nem sempre é fácil
- Normalmente, não há porque propor uma solução recursiva
- Entretanto, para alguns problemas a solução recursiva é mais elegante
- Há problemas que são recursivos já na sua definição
- Para estes, a solução recursiva é mais natural.

# Torres de Hanói

- Considere o problema das **Torres de Hanói**. Há três torres  $A$ ,  $B$  e  $C$  e há  $n$  discos de diâmetros diferentes dispostos na torre  $A$ , sempre com o disco de maior diâmetro abaixo de um disco de menor diâmetro. Como colocar todos os discos na torre  $C$ , usando a torre  $B$  como intermediária, passando um único disco de cada vez e sem inverter a ordem dos diâmetros (maior abaixo do menor) em nenhuma torre.
- Se uma solução para  $n - 1$  discos for encontrada, tem-se a solução para  $n$  discos.
- No caso trivial de  $n = 1$ , a solução é simples.
- Se for possível declarar uma solução para  $n$  discos em termos de  $n - 1$ , haverá uma solução recursiva.

# Torres de Hanói

- Mover  $n$  discos de  $A$  para  $C$ , usando  $B$  como auxiliar:
  - 1 Se  $n = 1$ , mova o único disco de  $A$  para  $C$  e pare.
  - 2 Mova os  $n - 1$  discos de  $A$  para  $B$ , usando  $C$  como auxiliar.
  - 3 Mova o último disco de  $A$  para  $C$ .
  - 4 Mova os  $n - 1$  discos de  $B$  para  $C$ , usando  $A$  como auxiliar.

- Programa:

```
void hanoi(char de, char para, char via, int n)
{
    if (n >= 1)
    {
        hanoi(de, via, para, n-1);
        printf("%c => %c\n", de, para);
        hanoi(via, para, de, n-1);
    }
}
```

# Sumário

## 1 Recursividade

- Definições Recursivas
- Recursão
- Exemplos

## 2 Busca Binária

- Busca Binária
- Exemplo
- Escrevendo Programas Recursivos

## 3 Simulação da Recursão

- Soluções Iterativas para Problemas Recursivos
- O que acontece quando uma função é chamada?
- O que acontece quando uma função é retornada?



# Simulando a Recursividade

- Todo problema com solução recursiva pode ser resolvido iterativamente
- Algumas linguagens de programação não permitem recursão
- Basta examinar com detalhes os mecanismos usados para implementar a recursividade para que seja possível simulá-los usando técnicas não-recursivas
- A solução recursiva geralmente é mais cara computacionalmente do que a solução não-recursiva
- A possibilidade de gerar uma solução não-recursiva a partir de um algoritmo recursivo é muito interessante em várias situações.

# Sumário

## 1 Recursividade

- Definições Recursivas
- Recursão
- Exemplos

## 2 Busca Binária

- Busca Binária
- Exemplo
- Escrevendo Programas Recursivos

## 3 Simulação da Recursão

- Soluções Iterativas para Problemas Recursivos
- O que acontece quando uma função é chamada?
- O que acontece quando uma função é retornada?

# Simulando a Recursividade

- Antes de examinar as ações de uma rotina recursiva, deve-se examinar a ação de uma rotina **não**-recursiva.
- Para o comando  
`rota(x);`
- onde *rota* é definida como  
`rota(a)`
- *x* é referido como um **argumento** (da função chamadora) e *a* como um **parâmetro** (da função chamada).
- O que acontece quando uma função é chamada?
  - 1 Passagem de argumentos,
  - 2 Alocação e iniciação de variáveis locais,
  - 3 Transferência do controle para a função.

# Simulando a Recursividade

- Cada uma das três etapas:
  - 1 **Passagem de argumentos:** para um parâmetro em C, uma cópia do argumento é criada localmente dentro da função.
  - 2 **Alocação e iniciação de variáveis locais:** depois que os argumentos são passados, as variáveis locais da função serão alocadas. Incluem as declaradas diretamente na função e as temporárias criadas durante a execução.
  - 3 **Transferência do controle para a função:** neste ponto, o **endereço de retorno** deve ser passado à função (armazenado na pilha), para que seja possível passar também o controle. Este controle deverá ser devolvido à função chamadora ao final da execução da função, daí a necessidade do endereço de retorno.

# Sumário

- 1 Recursividade
  - Definições Recursivas
  - Recursão
  - Exemplos
- 2 Busca Binária
  - Busca Binária
  - Exemplo
  - Escrevendo Programas Recursivos
- 3 Simulação da Recursão
  - Soluções Iterativas para Problemas Recursivos
  - O que acontece quando uma função é chamada?
  - O que acontece quando uma função é retornada?

# Retorno de uma Função

- Quando uma função termina sua execução, o controle retorna para a função chamadora. Três ações são executadas:
  - 1 O endereço de retorno é recuperado da pilha e armazenado em um lugar seguro
  - 2 A área de dados da função é liberada. Essa área contém todas as variáveis locais (incluindo as cópias locais dos argumentos), as temporárias e o endereço de retorno
  - 3 Desvia-se para o endereço de retorno salvo anteriormente. Isso devolve o controle à função chamadora no ponto imediatamente posterior ao comando que efetuou a chamada. Se a função retorna um valor, o mesmo será colocado num lugar seguro (registrador de hardware) de onde a função chamadora poderá recuperá-lo.

# Retorno de uma Função

- A seqüência de endereços de retorno forma uma pilha: se uma função *A* chama uma função *B*, que chama uma função *C*, a pilha conterá no topo o endereço de retorno da função *C* (endereço dentro de *B*), depois da função *B* (endereço dentro de *A*).
- Ou seja, de *C* só é possível retornar para a função *B* (função chamadora de *C*) e de *B* só é possível retornar para a função *A* (função chamadora de *B*).
- Conclusão: chamar uma função corresponde a colocar um endereço na pilha (*push*) e retornar corresponde a retirar um endereço da pilha (*pop*).
- E no caso da função recursiva?

# Implementando uma Função Recursiva

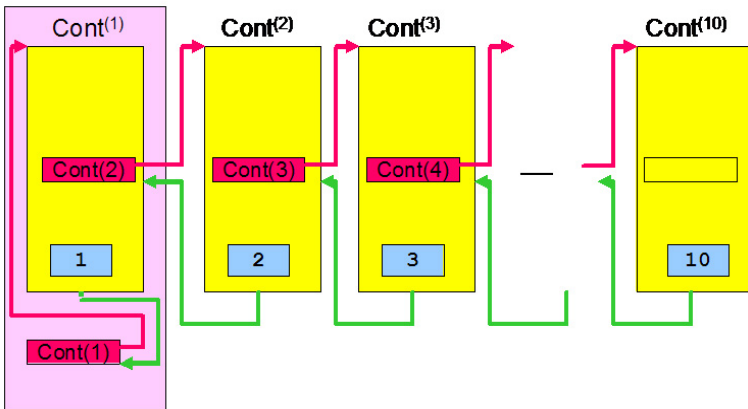
- Cada vez que uma função recursiva chama a si mesma, uma área de dados totalmente nova precisa ser alocada,
- Essa área de dados precisa ter todos os parâmetros, variáveis locais, temporárias e um endereço de retorno,
- No caso da recursividade, uma área de dados está associada não a uma função isolada, mas a uma chamada dessa função,
- Cada chamada acarreta a alocação de uma nova área de dados e toda a referência a um item na área de dados da função destina-se à área de dados da chamada mais recente,
- Da mesma forma, todo retorno provoca a liberação da atual área de dados e a área de dados alocada anteriormente torna-se a área atual.



# Implementando uma Função Recursiva

- Veja novamente a figura do contador recursivo:

**ContRecursivo**



# Eficiência da Recursividade

- Geralmente uma versão não-recursiva (iterativa) de um programa executará com mais eficiência, em termos de tempo e espaço, do que uma versão recursiva,
- Isso acontece porque o trabalho extra dispendido para entrar e sair de um bloco é evitado na versão não-recursiva,
- Num programa não-recursivo muita atividade de empilhamento e desempilhamento pode ser evitada,
- Contudo, muitas vezes, uma solução recursiva é o método mais natural e lógico de solucionar um problema,
- Alguns problemas são recursivos por natureza e sua solução recursiva é muito mais simples (por exemplo, as Torres de Hanói).

# Referências I

- [1] Engelbrecht, Angela  
Estrutura e Recuperação de Informação II.  
*Apostila*. Engenharia de Computação. PUC-Campinas, 2000.
- [2] Horowitz, E., Sahni, S. Rajasekaran, S.  
*Computer Algorithms*.  
Computer Science Press, 1998.
- [3] Rosa, J. L. G.  
Recursão. SCC-201 Introdução à Ciência da Computação II (capítulo 2).  
*Slides*. Ciência de Computação. ICMC/USP, 2009.

# Referências II

- [4] Tenenbaum, A. M., Langsam, Y., Augestein, M. J.  
*Estruturas de Dados Usando C.*  
Makron Books, 1995.