UNIPAC

Universidade Presidente Antônio Carlos

Bacharelado em Ciência da Computação

Introdução a Programação

Material de Apoio

Parte X – Funções Recursivas

Prof. Nairon Neri Silva

naironsilva@unipac.br

1º sem / 2020

Material cedido pela professora Lívia

Recursividade

- Muitos problemas têm a seguinte propriedade: cada ***instância*** do problema contém uma instância menor do mesmo problema. Dizemos que esses problemas têm *estrutura recursiva*.

- Uma ***instância*** de um problema é um exemplo concreto do problema, com dados específicos. Cada conjunto de dados de um problema define uma instância do problema.

Recursividade

- Em programação, a recursividade é um mecanismo útil e poderoso que permite a uma função chamar a si mesma direta ou indiretamente, ou seja, uma função é dita recursiva se ela contém pelo menos uma chamada explícita ou implícita a si própria.

Recursividade

- A ideia básica de um algoritmo recursivo consiste em diminuir sucessivamente o problema em um problema menor ou mais simples, até que o tamanho ou a simplicidade do problema reduzido permita resolvê-lo de forma direta, sem recorrer a si mesmo.
- Quando isso ocorre, diz-se que o algoritmo atingiu uma condição de parada, a qual deve estar presente em pelo menos um local dentro algoritmo.

Recursividade

- Sem esta condição o algoritmo não para de chamar a si mesmo, até estourar a capacidade da pilha, o que geralmente causa efeitos colaterais e até mesmo o término indesejável do programa.

Recursividade

- Para resolver um tal problema podemos aplicar o seguinte método:

✓ se a instância em questão é pequena,

resolva-a diretamente (use força bruta se necessário);

✓ senão,

reduza-a a uma instância menor do mesmo problema,

aplique o método à instância menor e

volte à instância original.

Recursividade - Exemplo

- Solução Iterativa (Fatorial)

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

Recursividade - Exemplo

- Solução Recursiva (Fatorial)

```
int fat(int n){
    if (n == 0)
     return 1;
    else
     return n * fat(n – 1);
}
```

Recursividade - Exemplo

- Solução Recursiva (Série de Fibonacci)

```
int fib (int n){
   if ((n == 0)  || (n==1))
        return n;
   else
     return fib(n-2) + fib(n-1);
}
```

Recursividade - Exemplo

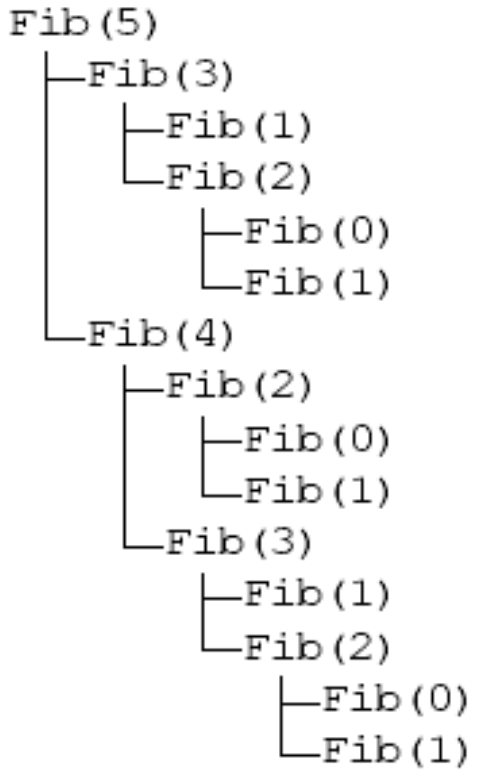


Diagrama de Execução de Fib(5)

Exercícios não avaliativos

1) Dada a implementação da função abaixo:

```
int f(int n){
  if (n < 4)
    return 3*n;
  else
    return 2 * f(n - 4) + 5;
}
```

Quais são os valores de F(3), F(7) e F(8)?

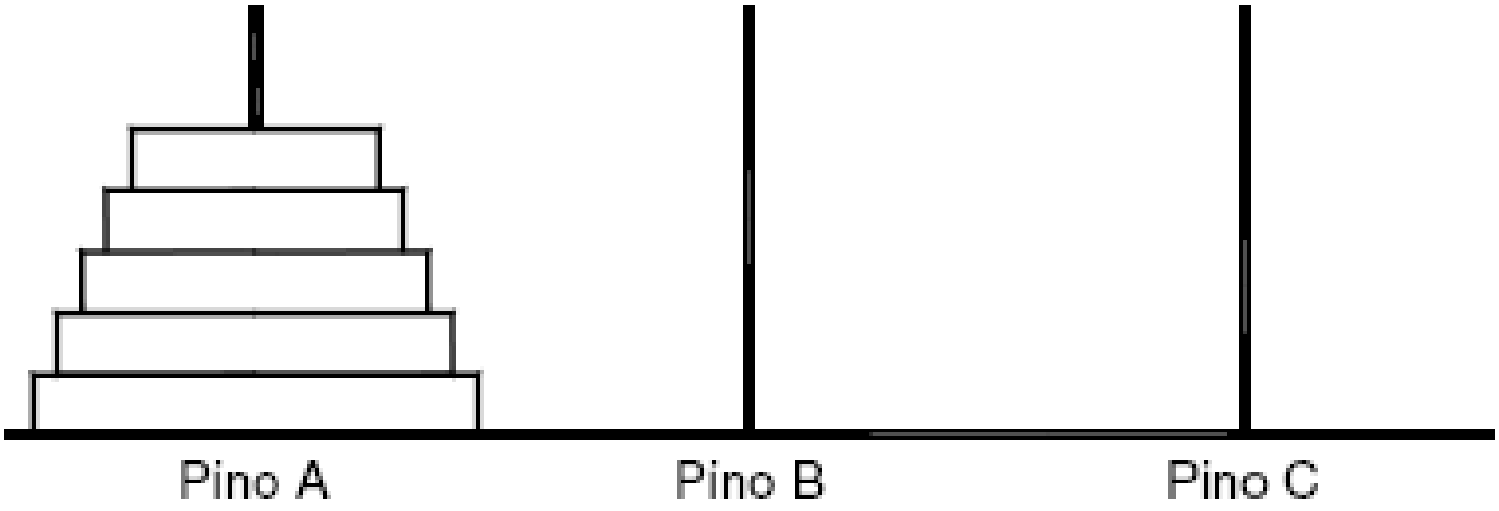
Exercícios não avaliativos

2) Crie uma função recursiva que receba um número n e retorne a soma de todos os anteriores incluindo o próprio número. Ex.: Se o n for 5, o retorno será 15 (5 + 4 + 3 + 2 + 1)

Recursividade - Exemplo

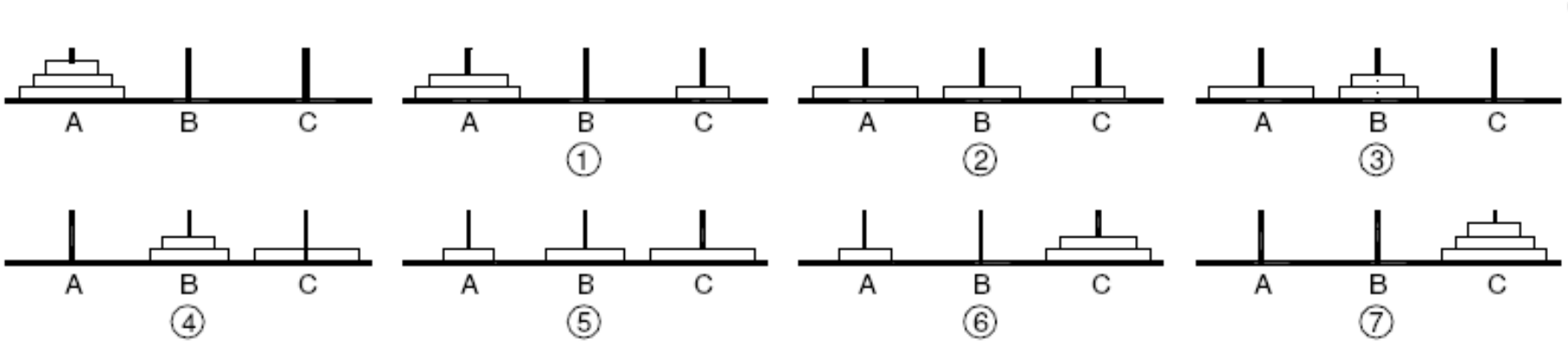
- Problema da Torre de Hanói: publicado em 1883 pelo matemático francês Edouard Lucas, e consiste em transferir, com o menor número de movimentos, a torre composta por *N* discos do pino **A** (origem) para o pino **C** (destino), utilizando o pino **B** como auxiliar. Somente um disco pode ser movimentado de cada vez e um disco não pode ser colocado sobre outro disco de menor diâmetro.

Recursividade



Recursividade

- Exemplo:

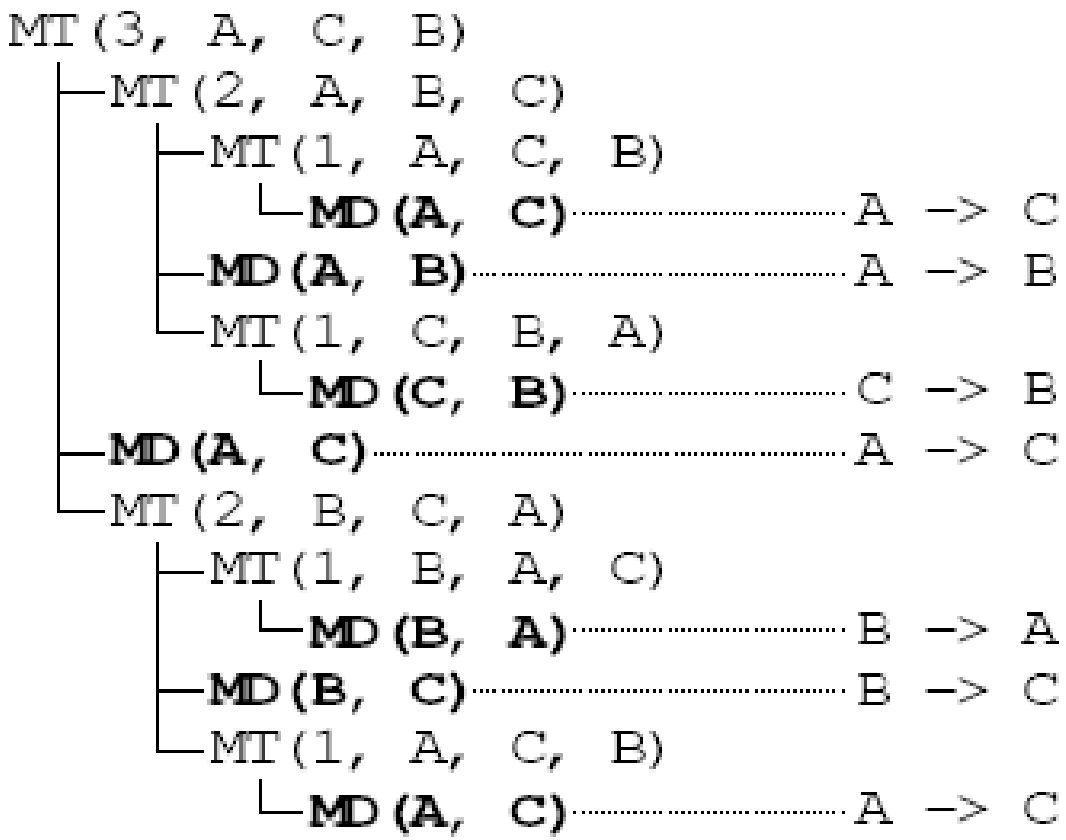


Recursividade

```
procedimento MoveTorre(N : natural; Orig, Dest, Aux : caracter)
início
  se N = 1 então
    MoveDisco(Orig, Dest)  senão
  início
    MoveTorre(N - 1, Orig, Aux, Dest)
    MoveDisco(Orig, Dest)
    MoveTorre(N - 1, Aux, Dest, Orig)
  fim
fim

procedimento MoveDisco(Orig, Dest : caracter)
início
  Escreva("Movimento: ", Orig, " -> ", Dest)
fim
```

Recursividade



Recursividade

- Observe que o algoritmo MoveTorre faz duas chamadas a si mesmo, portanto o número de movimentos cresce exponencialmente com o número de discos.

Recursividade

- Mais precisamente, a solução do problema com N discos requer 2^N-1 movimentos. Assim, se uma pessoa se dispusesse a resolver o problema com 25 discos, com dedicação exclusiva de 8 horas por dia e realizando um movimento por segundo, esta "pobre criatura" gastaria 3354431 segundos, ou seja, mais de 3 anos para solucionar o problema. Esse tempo dobraria a cada disco acrescentado!

Atividade Prática

(durante a aula)

Formação dos Grupos de Trabalho

Atividade Prática (em grupo)

1) A multiplicação de dois números inteiros pode ser feita através de somas sucessivas. Proponha um algoritmo recursivo int multpRec(n1,n2) que calcule a multiplicação de dois inteiros. (Ex.: $6 * 4 = 4 + 4 + 4 + 4 + 4 + 4$). No método main() realize o teste da função.