

Linguagens de Programação

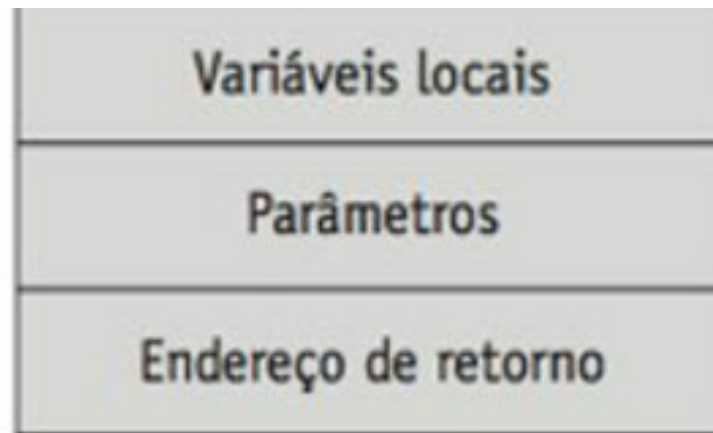
Aula 8

Subprogramação e recursão

2º semestre de 2019
Prof José Martins Junior

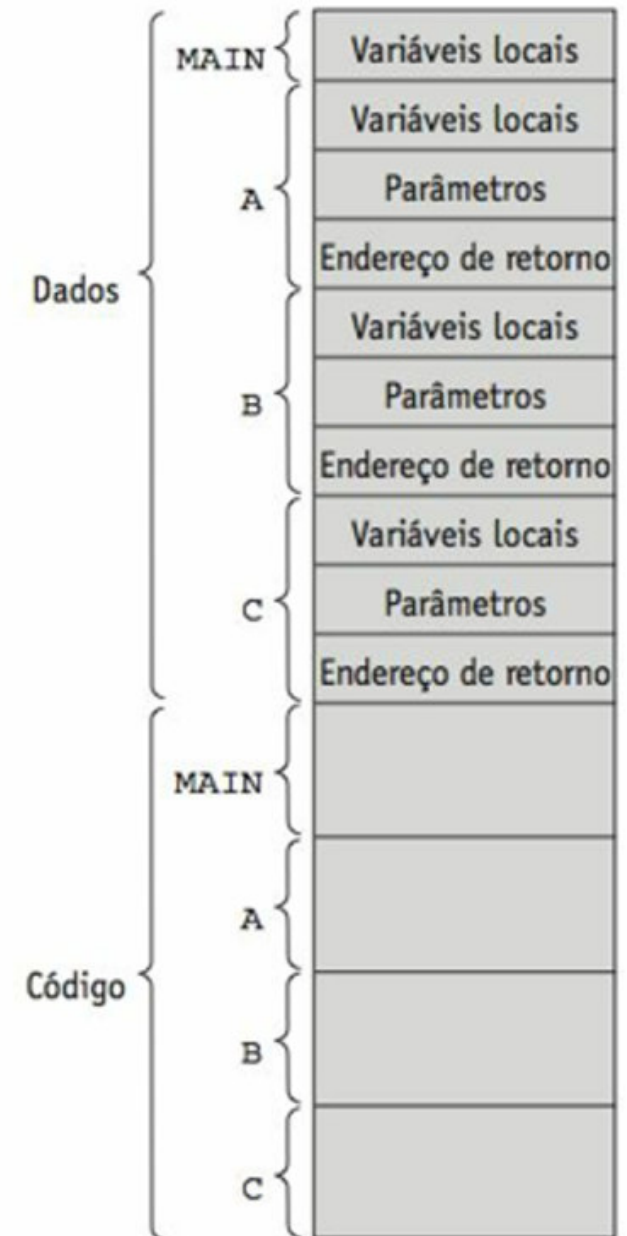
Subprograma

- Subprograma simples consiste em duas partes (tamanhos fixos)
 - Código real do subprograma, que é constante
 - Variáveis locais que podem mudar quando o subprograma é executado
- Registro de ativação
 - Formato, ou layout, da parte que não é código de um subprograma
 - Os dados que descreve são relevantes apenas durante a ativação
 - Sua forma é estática



Registro de ativação simples

- Exemplo um registro de ativação simples
 - Função MAIN e 3 outras (A, B e C)

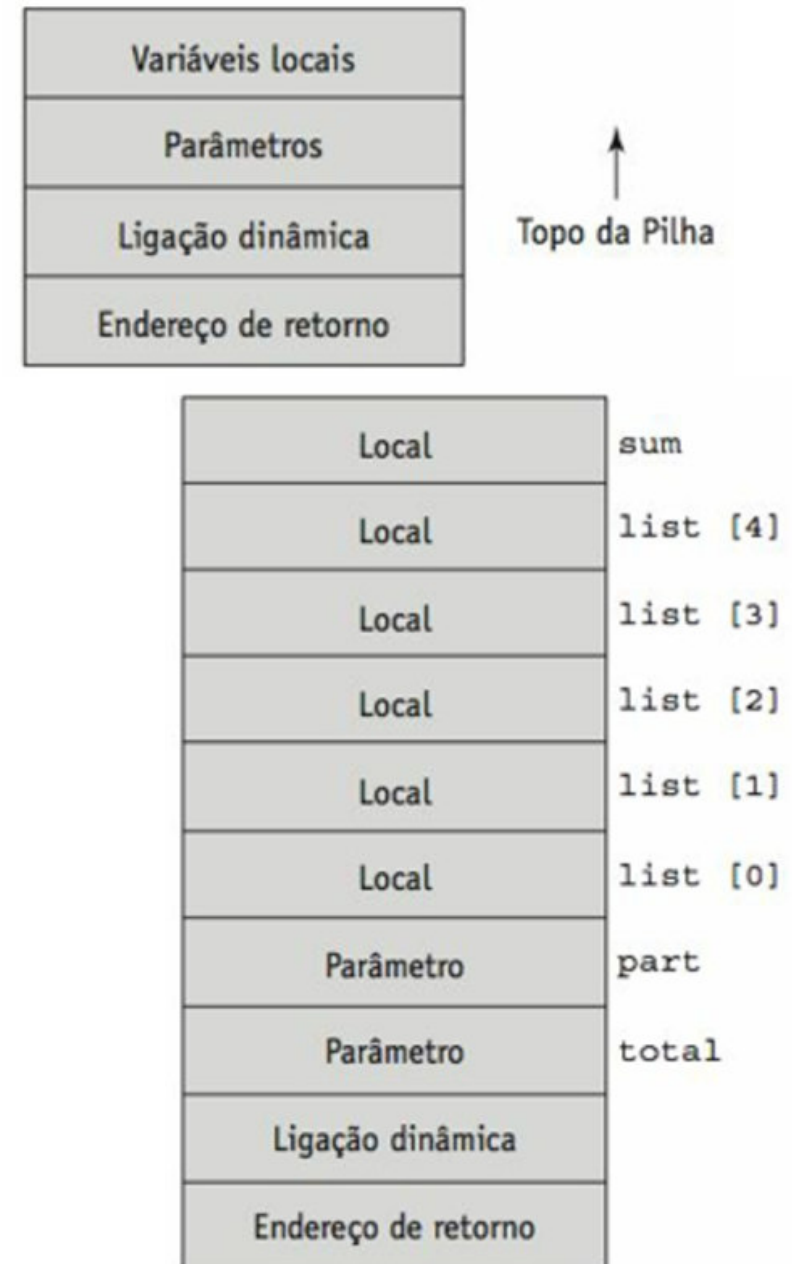


Registro de ativação mais complexo

- Ligação dinâmica
 - Ponteiro para a base da instancia de registro de ativação do chamador

- Exemplo em C

```
void sub(float total, int part) {  
    int list[5];  
    float sum;  
    ...  
}
```

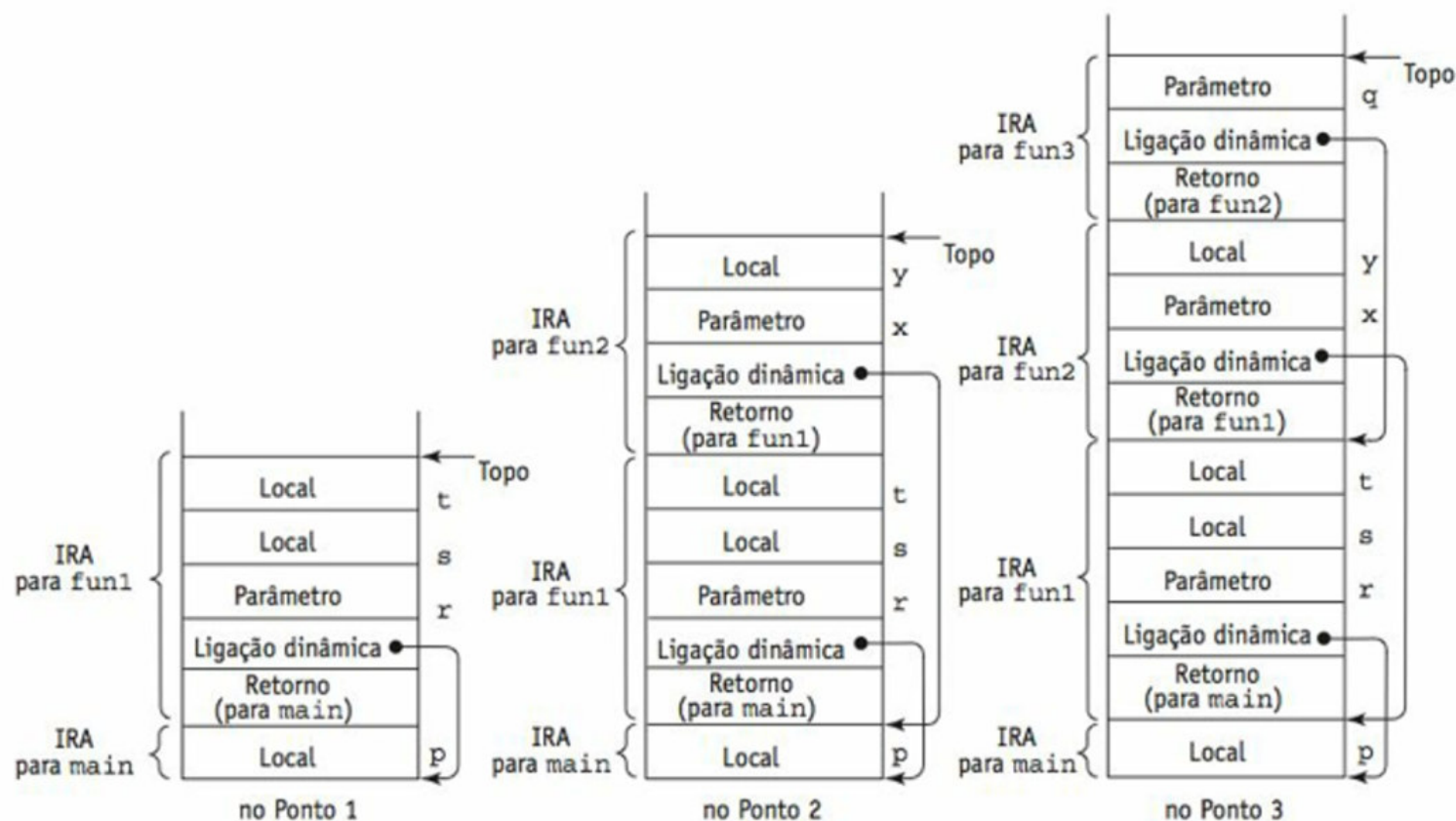


Processo geral

- Ações do subprograma chamador
 1. Criar uma instância de registro de ativação
 2. Gravar o estado da execução da unidade de programa atual
 3. Calcular e passar os parâmetros
 4. Passar o endereço de retorno para o subprograma chamado
 5. Transferir o controle para o subprograma chamado
- Ações do prologo do subprograma chamado
 1. Salvar o PE (ponteiro para a base do registro de ativação do programa principal) antigo na pilha como a ligação dinâmica e criar o novo valor
 2. Alocar variáveis locais
- Ações do epílogo do subprograma chamado
 1. Se existirem parâmetros com passagem por valor-resultado ou passagem por modo de saída, os valores atuais desses parâmetros são movidos para os parâmetros reais correspondentes
 2. Se o subprograma é uma função, o valor funcional é movido para um local acessível ao chamador
 3. Restaurar o ponteiro da pilha configurando-o para o valor do PE atual menos um e configurar o PE para a ligação dinâmica antiga
 4. Restaurar o estado de execução do chamador
 5. Transferir o controle de volta para o chamador

Exemplo não recursivo em C

```
void fun1(float r) {  
    int s, t;  
    ...           ← 1  
    fun2(s);  
    ...  
}  
  
void fun2(int x) {  
    int y;  
    ...           ← 2  
    fun3(y);  
    ...  
}  
  
void fun3(int q) {  
    ...           ← 3  
}  
  
void main() {  
    float p;  
    ...  
    fun1(p);  
    ...  
}
```




Recursão

- Recursão é o ato de uma função chamar ela mesma
 - Uma função que chama a ela mesma é chamada função recursiva
- Características de um programa recursivo
 - Chamada recursiva
 - Chama a si mesmo para resolver parte do problema
 - Condição de parada
 - Existe pelo menos um caso base que é resolvido sem chamar a si mesmo
- Toda função recursiva possui duas regras de recorrência
 - Resolver parte do problema (caso base) ou
 - Reduzir o tamanho do problema (caso geral)

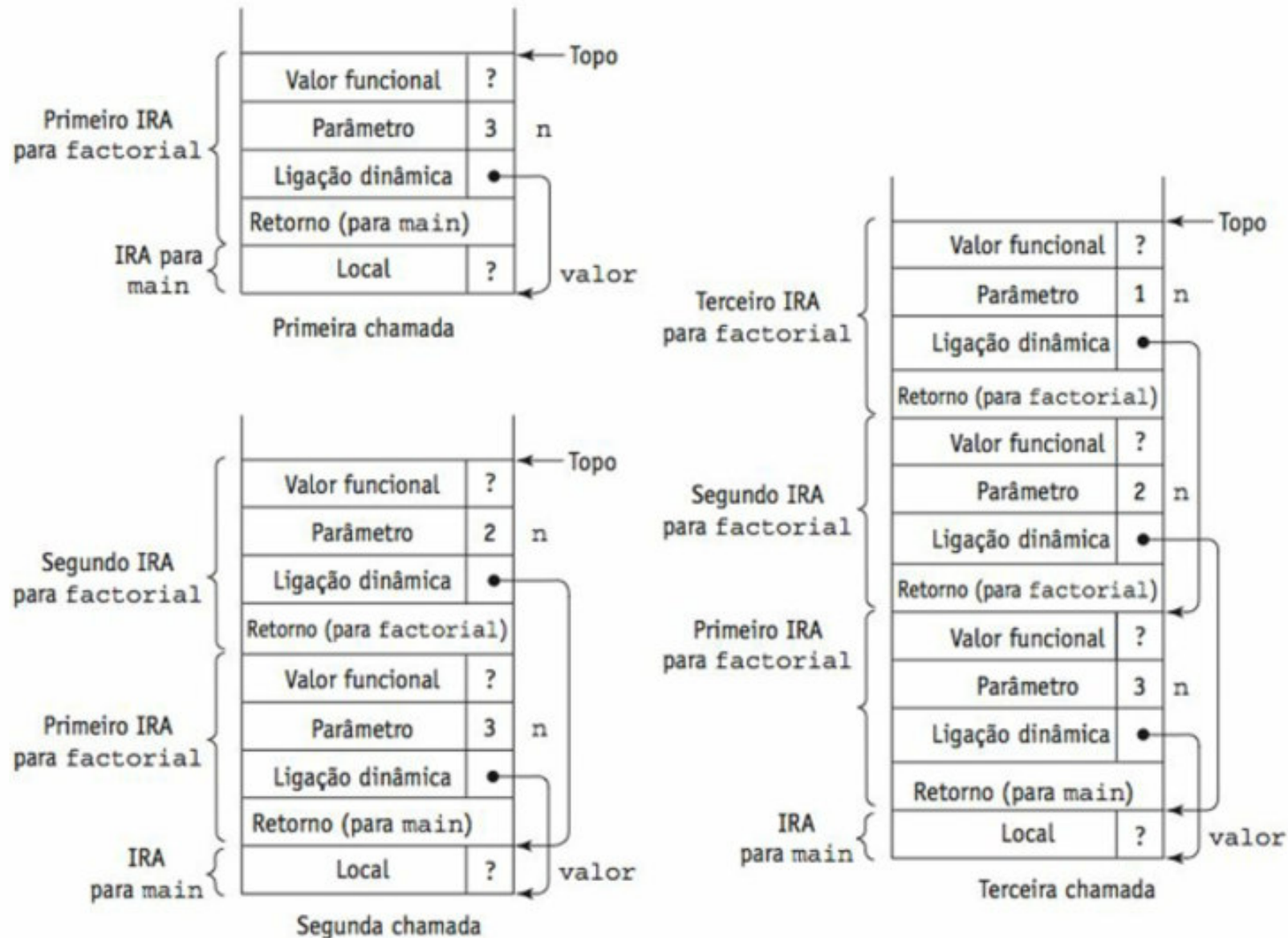
Exemplo de recursão

- Exemplo padrão: fatorial de um número
 - Produto dos números inteiros de 1 até o número
fatorial de 3 = $3 * 2 * 1$
 - Se você observar com cuidado verá que
fatorial de 3 = $3 * \text{fatorial de } 2$
fatorial de 2 = $2 * \text{fatorial de } 1$
fatorial de 1 = $1 * \text{fatorial de } 0$
 - factorial(0) é o **caso base**
 - Condição de parada



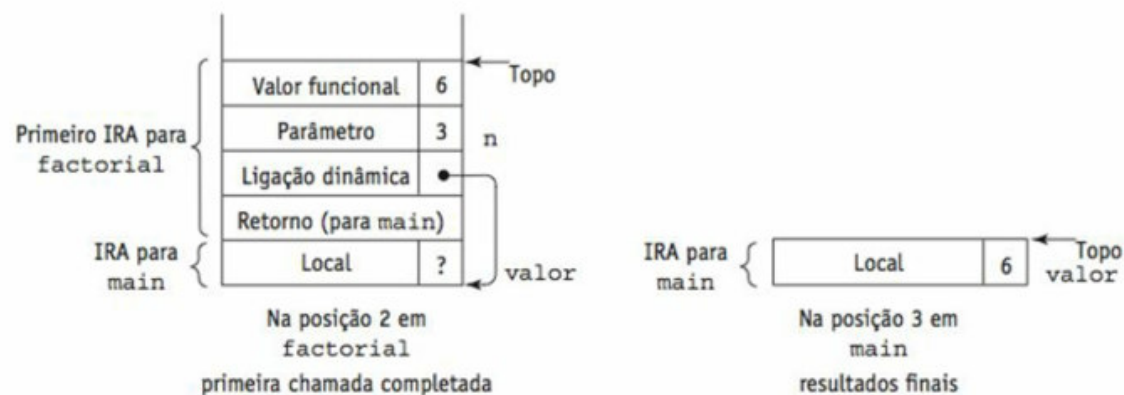
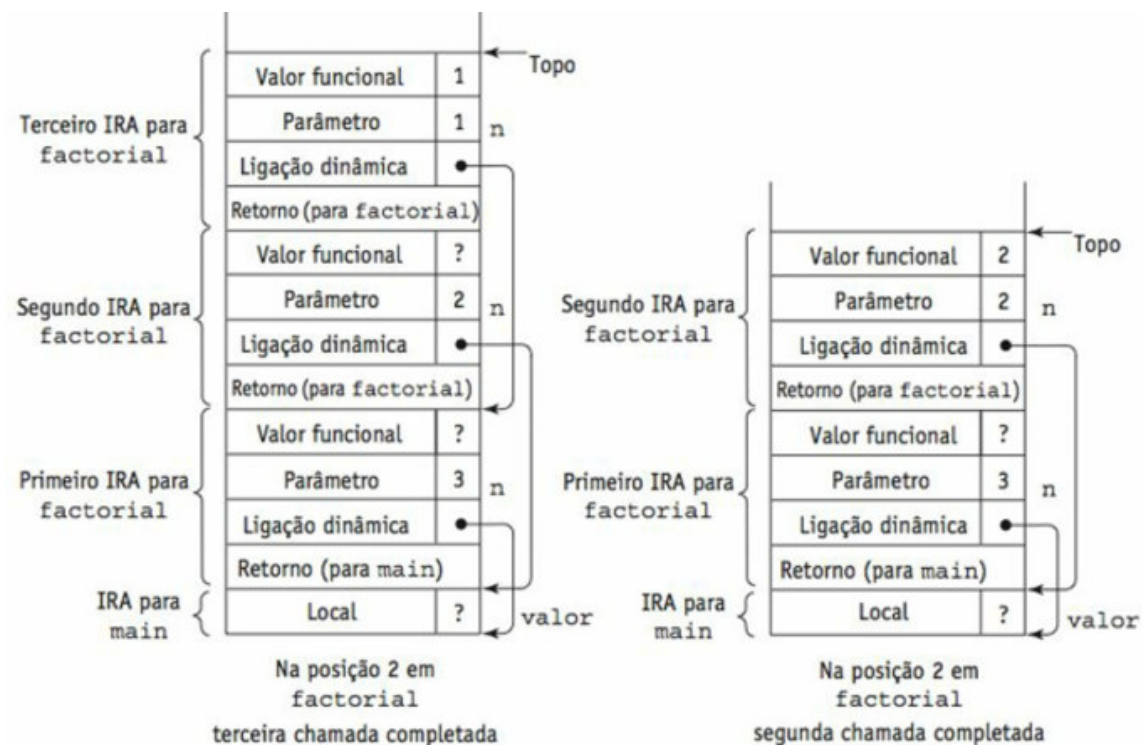
```
int factorial(int n) {  
    <-----1  
    if (n <= 1)  
        return 1;  
    else return (n * factorial(n - 1));  
    <-----2  
}  
void main() {  
    int value;  
    value = factorial(3);  
    <-----3  
}
```


Ativação de cada chamada recursiva



IRA = instância de registro de avaliação

Retorno da pilha de recursão



IRA = instância de registro de avaliação

Problemas recursivos

- Existem problemas que são naturalmente recursivos
 - Ex.: Fatorial: $n! = n(n-1)(n-2)\dots 1$, ou seja, $n! = n * (n - 1)!$
 - Mas mesmo esses podem ser escritos em programas iterativos

Recursivo:	Iterativo:
<pre>public int fatorial(int f) { if (f == 0) return 1; else return (f * fatorial(f - 1)); }</pre>	<pre>public int fatorial(int f) { int i, aux; aux = f; if (f == 0) return 1; else { for(i = f - 1; i > 0; i--) aux = aux * i; return aux; } }</pre>

Problemas não recursivos

- Mesmo definições iterativas podem ser resolvidas com funções recursivos
- Exemplo: Calcular a soma dos números pares de 0 até N

```
public int somaPares(int n) {  
    int soma = 0;  
    if (n == 0)  
        return 0;  
    else {  
        if(n % 2 == 0)  
            soma = n;  
        return (soma + somaPares(n - 1));  
    }  
}
```

Problema recursivo: Fibonacci

- A Sequência de Fibonacci
 - Primeiros termos são os números 0 e 1
 - Termos subsequentes são a soma dos dois termos predecessores
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...
- Proporção áurea (1,618...)
 - Razão entre dois números consecutivos da sequência
- Regras de recorrência
 - $\text{Fibonacci}(0) = 0$ (condição de parada)
 - $\text{Fibonacci}(1) = 1$ (condição de parada)
 - $\text{Fibonacci}(n) = \text{Fibonacci}(n-1) + \text{Fibonacci}(n-2)$ (caso geral)

Fibonacci recursivo (Java)

- A implementação recursiva é muito mais lenta que a iterativa
 - O motivo é que a cada recursão são duplicadas as chamadas

```
public static long fiboRec(int n) {  
    if (n < 2) {  
        return n;  
    } else {  
        return fiboRec(n - 1) + fiboRec(n - 2);  
    }  
}
```

Fibonacci iterativo (Java)

- Apesar de menos elegante, é muito mais rápida para valores maiores de n

```
public static long fiboIter(int n) {  
    long atu = 0;  
    long ant = 0;  
    for (int i = 1; i <= n; i++) {  
        if (i == 1) {  
            atu = 1;  
            ant = 0;  
        } else {  
            atu += ant;  
            ant = atu - ant;  
        }  
    }  
    return atu;  
}
```

Vantagens e desvantagens

- Vantagens da recursão
 - Redução do tamanho do código fonte
 - Maior clareza do algoritmo para problemas de definição naturalmente recursiva
- Desvantagens da recursão
 - Baixo desempenho na execução devido ao tempo para gerenciamento das chamadas
 - Dificuldade de depuração dos subprogramas recursivos, principalmente se a recursão for muito profunda

Bibliografia

- SEBESTA, R. W. Conceitos de Linguagens de Programação. Porto Alegre: Bookman, 2011. 792p.