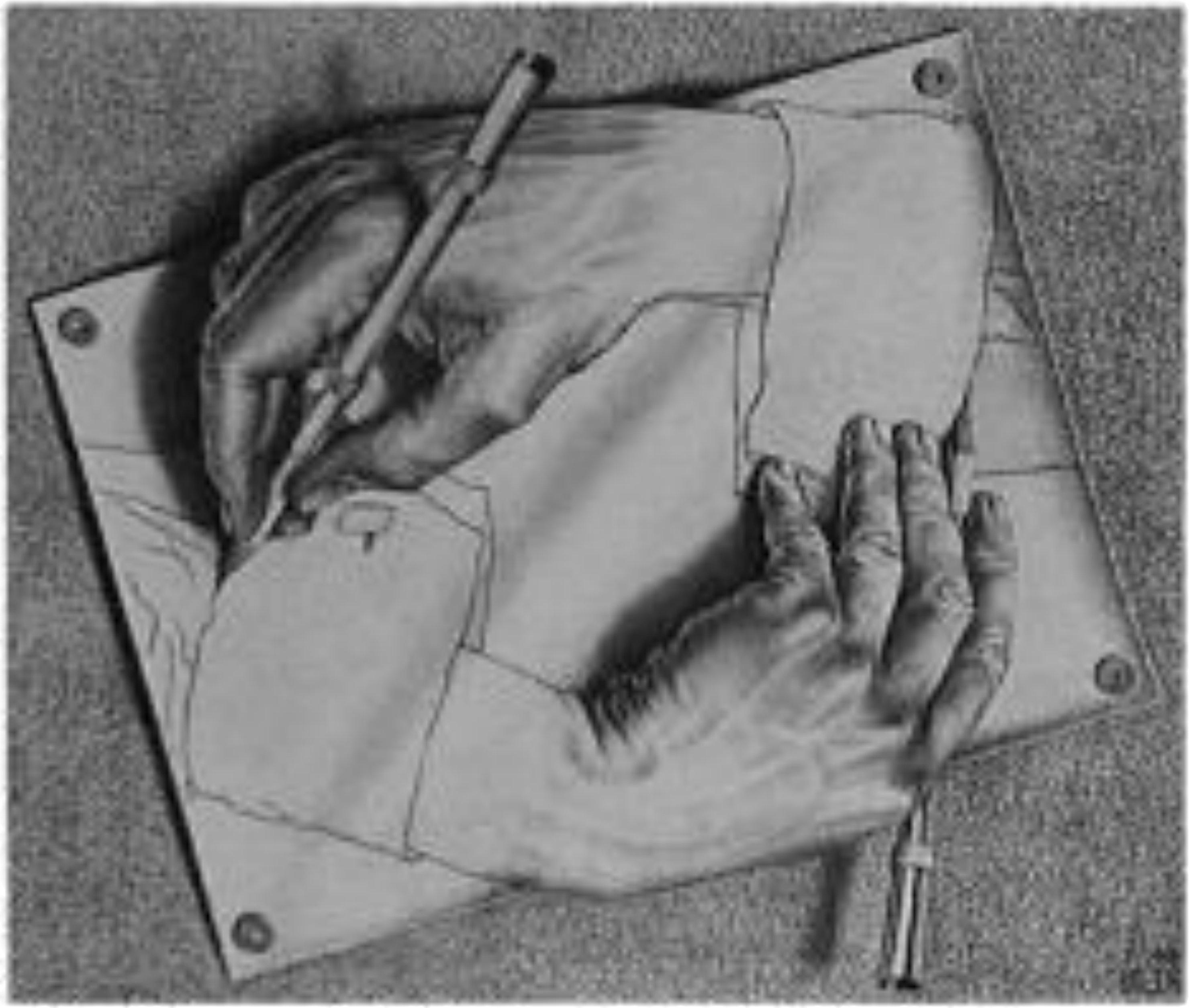
Recursividade

Túlio Toffolo – tulio@toffolo.com.br
www.toffolo.com.br



Outros Exemplos de Recursividade

- Fractal também é um exemplo de recursividade

Quando vale a pena usar recursividade

- Recursividade vale a pena para Algoritmos complexos, cuja a implementação iterativa é complexa e normalmente requer o uso explícito de uma pilha
  - Dividir para Conquistar (Ex. Quicksort)
  - Caminhamento em Árvores (pesquisa, backtracking)

Função de Complexidade

- Exemplo de recorrência:

$T(n) = n + T(n-1)$

$T(1) = 1$

$T(n) = n + T(n-1)$

$T(n) = n + (n-1) + T(n-2)$

$T(n) = n + (n-1) + (n-2) + T(n-3)$

...

$T(n) = n + (n-1) + (n-2) + ... + 2 + T(1)$

$T(n) = n + (n-1) + (n-2) + ... + 2 + 1$

No exemplo do fatorial

```
int fatorial(int n) {
    if (n == 1)
        return 1;
    else {
        return n * fatorial(n-1);
    }
}
```

Análise da Função Fatorial

- Qual a equação de recorrência que descreve a complexidade da função fatorial vista?

$T(n) = 1 * T(n-1)$

$T(1) = 1$

$T(n) = 1 * T(n-1)$

$T(n-1) = 1 * T(n-2)$

$T(n-2) = 1 * T(n-3)$

...

$T(2) = 1 * T(1)$

Análise de Funções Recursivas

- Além da análise de custo do tempo, deve-se analisar também o custo de espaço
- Qual a complexidade de espaço da função fatorial (qual o tamanho da pilha de execução)?
  - **Proporcional ao número de chamadas?**

Alguns somatórios úteis

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=0}^{k} a^i = \frac{a^{k+1}-1}{a-1} (a \neq 1)$$

$$\sum_{i=0}^{k} 2^k = 2^{k+1} - 1$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=0}^{k} \frac{1}{2^i} = 2 - \frac{1}{2^k}$$

Recursão

Quando usar: quando o problema pode ser definido recursivamente de forma natural

- Como usar
- 1º ponto: definir o problema de forma recursiva, ou seja, em termos dele mesmo

- 2º ponto: definir a condição de término (ou *condição básica*)

- 3º ponto: a cada chamada recursiva, deve-se tentar garantir que se está mais próximo de satisfazer a condição de término (caso mais simples)

- Caso contrário, qual o problema?

Recursão

Problema do fatorial

- 1º ponto: definir o problema de forma recursiva
  - n! = n * (n-1)!

- 2º ponto: definir a condição de término
  - n=1

- 3º ponto: a cada chamada recursiva, deve-se tentar garantir que se está mais próximo de satisfazer a condição de término
  - A cada chamada, n é decrementado, ficando mais próximo da condição de término

Recursão vs Iteração de Fatorial

Quem é melhor?

```
//versão iterativa
int fatorial(int n) {
    int i, fat=1;
    for (i=2;i<=n;i++)
        fat=fat*i;
    return(fat);
}
```

```
//versão recursiva
int fatorial(int n) {
    int fat;
    if (n==0)
        fat=1;
    else fat=n*fatorial(n-1);
    return(fat);
}
```

Recursão vs Iteração de Fibonacci

Quem é melhor?

```
//versão recursiva
int fib(int n) {
    int resultado;
    if (n<2)
        resultado=n;
    else
        resultado =fib(n-1)+fib(n-2);
    return(resultado);
}
```

```
//versão iterativa
int fib(int n) {
    int i=1, k, resultado=0;
    for (k=1;k<=n;k++)
    {
        resultado=resultado+i;
        i=resultado-i;
    }
    return(resultado);
}
```

Exercício

Crie um vetor de inteiros com alocação dinâmica. O tamanho do vetor e os valores serão passados pelo usuário. Então, crie 2 funções recursivas:

- Uma que faça a soma de todos os elementos do vetor e retorne o valor total da soma. Exiba esse valor na principal
- Outra que inverta os elementos do vetor e retorne o vetor (invertido) para o exibir na principal.