Introdução a Programação
Recursividade

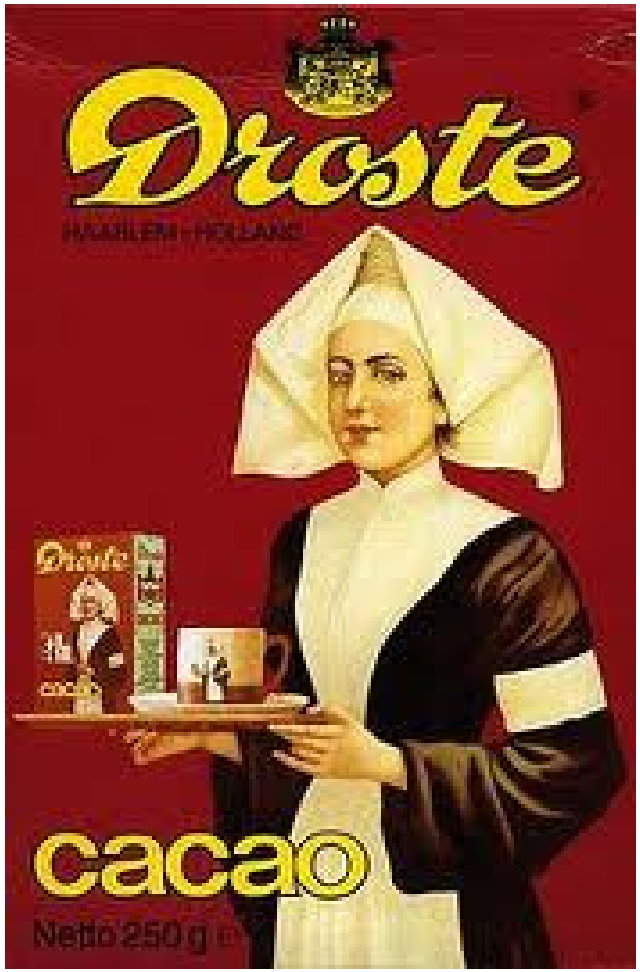
Prof. Hebert Coelho

Instituto de Informática
Universidade Federal de Goiás

Roteiro

- Recursão

Recursão



Recursão

- Funções em C podem ser usadas recursivamente, isto é uma função pode chamar a si mesmo.
- É como se procurássemos no dicionário a definição da palavra recursão e encontrássemos o seguinte texto:
  recursão: s.f. Veja a definição em recursão
- Um exemplo simples de função que pode ser escrita com chamadas recursivas é o fatorial de um número inteiro. O fatorial de um número, sem recursão, é definido como

$$n! = n * (n - 1) * (n - 2) * \cdots * 2 * 1$$

Fatorial sem recursão

```
1 unsigned long int fat (unsigned long int num) {
2       unsigned long int fato=1, i;
3
4       for (i=num; i>1; i--) fato = fato * i;
5
6       return fato;
7 }
```

Fatorial com recursão

Alternativamente, o fatorial pode ser definido como o produto deste número pelo fatorial de seu predecessor, ou seja

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

Fatorial com recursão

```
1 unsigned long int fat (unsigned long int num) {
2           if (num == 0)
3          return 1;
4      else
5           return num * fat (num-1);
6 }
```

Comentários

- Um ponto importante é que toda função recursiva deve prever cuidadosamente como o processo de recursão deve ser interrompido.
- No caso da função `fat` o processo é interrompido quando o valor do número passado como parâmetro vale 0.
- É importante notar que recursão não traz obrigatoriamente economia de memória porque os valores sendo processados tem de ser mantidos em pilhas.
- Nem será mais rápido, e as vezes pode ser até mais lento porque temos o custo de chamada as funções.
- As principais vantagens da recursão são códigos mais compactos e provavelmente mais fáceis de serem lidos.

Potenciação x^n, $n \geq 0$

```
1  float Elevar(float x, int n) {
2      if (n == 0) {
3          return 1;
4      }
5      if (n == 1) {
6          return x;
7      }
8      else {
9          return x * Elevar(x, n-1);
10     }
11 }
```

Antes ou Depois?

```c
#include<stdio.h>
void imprime1(int n) {
    if (n < 0) return;
    printf("%2d, ", n);
    imprime1(n-1);
}
void imprime2(int n) {
    if (n < 0) return;
    imprime2(n-1);
    printf("%2d, ", n);
}
int main (void) {
    imprime1(10);
    printf("\n");
    imprime2(10);
    printf("\n");
    return 0;
}
```

Recursão o que é?

Recursão

É um processo que quebra um problema em problemas menores, que serão quebrados em problemas menores ainda, até que chegamos no menor problema possível e na sua solução (caso básico), neste ponto começamos a retornar começando pelo caso básico.

Passos para escrever uma função recursiva

- Escreva um protótipo da função recursiva.
- Escreva um comentário que descreve o que a função deve fazer.
- Determine o caso base (pode haver mais de um) e a solução deste caso.
- Determine qual é o problema menor do que o atual a ser resolvido.
- Use a solução do problema menor para resolver o problema maior.

Aplicando os passos

- Considere que vamos resolver a soma de um vetor de *n* elementos.
- Escreva um protótipo da função recursiva:
  `int soma(int vetor[], int n);`
- Escreva um comentário que descreve o que a função deve fazer:
  "A função deve somar *n* elementos de um vetor inteiro".
- Determine o caso base (pode haver mais de um) e a solução deste caso:
  O caso base seria a soma de um vetor do elemento que restou que é ele próprio.

Aplicando os passos

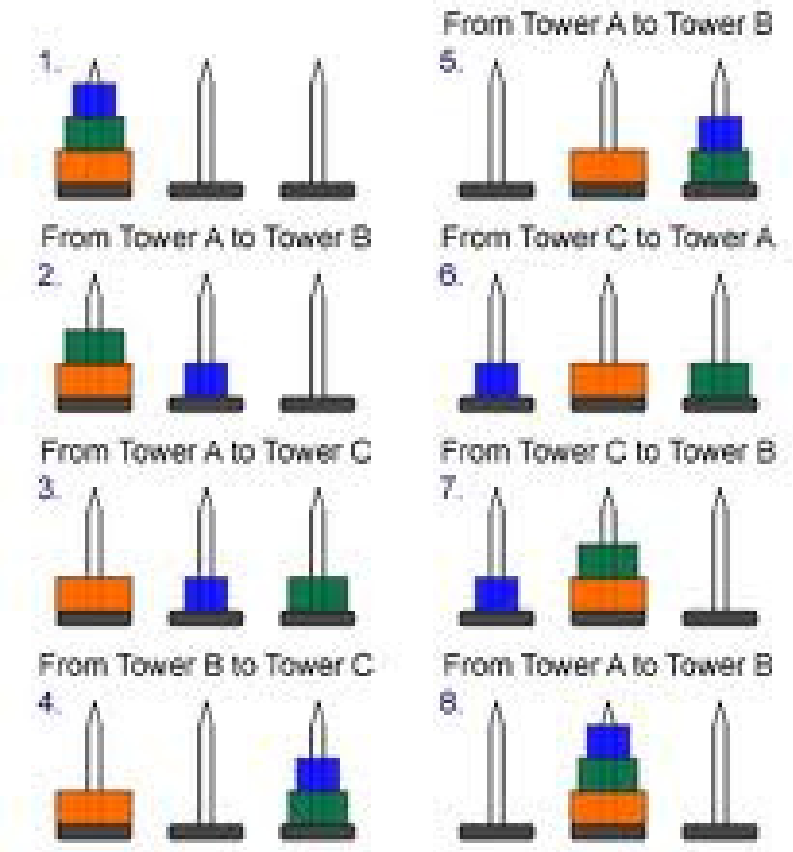
- Determine qual é o problema menor a ser resolvido:
  O problema menor a ser resolvido é `soma(vetor, n-1)`.
- Use a solução do problema menor para resolver o problema maior.
  vetor[n-1] + soma(vetor, n-1)
- Lembre-se que em C o elemento 1 está na posição 0, portanto o elemento n está na posição $n - 1$.

Solução

```
1  #include<stdio.h>
2
3  int soma(int v[], int n) {
4      if (n == 1) {
5          return v[0];
6      }
7      else {
8          return v[n-1] + soma(v, n-1);
9      }
10 }
11 int main (void) {
12     int v[5] = {7, 2, -2, 4, 15};
13
14     printf("%d\n", soma(v, 5));
15     return 0;
16 }
```

Torres de Hanói

Torres de Hanói



Algoritmo recursivo

- if n==1 mova este disco de A para B e pare.
- Mova os n-1 discos superiores de A para C usando B como auxiliar.
- Mova o disco restante de A para B.
- Mova os n-1 discos de C para B, usando A como auxiliar.
- MÁGICA! NÃO PRECISA SABER COMO MOVER.....

Algoritmo em C

```
1  #include <stdio.h>
2  void towers(int nDiscos, char dePino, char paraPino,
3                           char auxPino) {
4  /* Se um disco somente, mova-o e retorne */
5      if(nDiscos==1) {
6          printf("Move disco 1 do pino %c para pino %c\n",
7                 dePino, paraPino);
8              return;
9      }
10 /* Move n-1 discos superiores de A para C, usando B */
11     towers(nDiscos-1, dePino, auxPino, paraPino);
12
13 /* Move discos restantes de A para B */
14     printf("Move disco %d do pino %c para pino %c\n",
15            nDiscos, dePino, paraPino);
16
17 /* Move n-1 discos de C para B usando A como auxiliar */
18     towers(nDiscos-1, auxPino, paraPino, dePino);
19 }
```

Algoritmo em C

```c
int main() {
    int n;
    printf("Entre o numero de discos: ");
    scanf("%d",&n);
    printf("O algoritmo faz os seguintes movimentos:\n");
    towers(n, A, B, C);
    printf("\n");
    return 0;
}
```

Recursão de 2a. Ordem

Example (A Sequencia Fibonacci)

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

Maior Divisor comum

Example (Maior Divisor comum)

```
int gcd(int a, int b)
{
    if (b == 0)
        return a;
    else
        return gcd(b, a % b);
}
```

Coeficientes Binomiais

Example (Coeficientes Binomiais)

```
int binom(int n, int r)
{
    if (r == 0 || r == n)
        return 1;
    else
        return binom(n - 1, r) + binom(n - 1, r - 1);
}
```

Vantagens e Desvantagens

- Vantagens
  - O código pode ser mais fácil de escrever.
  - Algumas situações são naturalmente recursivas.

Vantagens da Recurção

- Algumas Estruturas são naturalmente recursivas:
  - Listas Encadeadas.
  - Árvores Binárias.
- Problemas naturalmente recursivos:
  - Deslocamento em listas encadeadas.
  - Caminhamento em árvores binárias.
  - Avaliação de expressões.

Desvantagens da Recursão

- Desvantagens
  - As funções recursivas geralmente são mais lentas do que as funções não recursivas.
  - Recursão excessiva pode transbordar a pilha de tempo de execução.
    - Às vezes, cada chamada de função gera duas ou mais chamadas recursivas nesse nível.
  - É preciso ter muito cuidado ao escrever funções recursivas; Elas podem ser complicadas.