CENTRO PAULA SOUZA

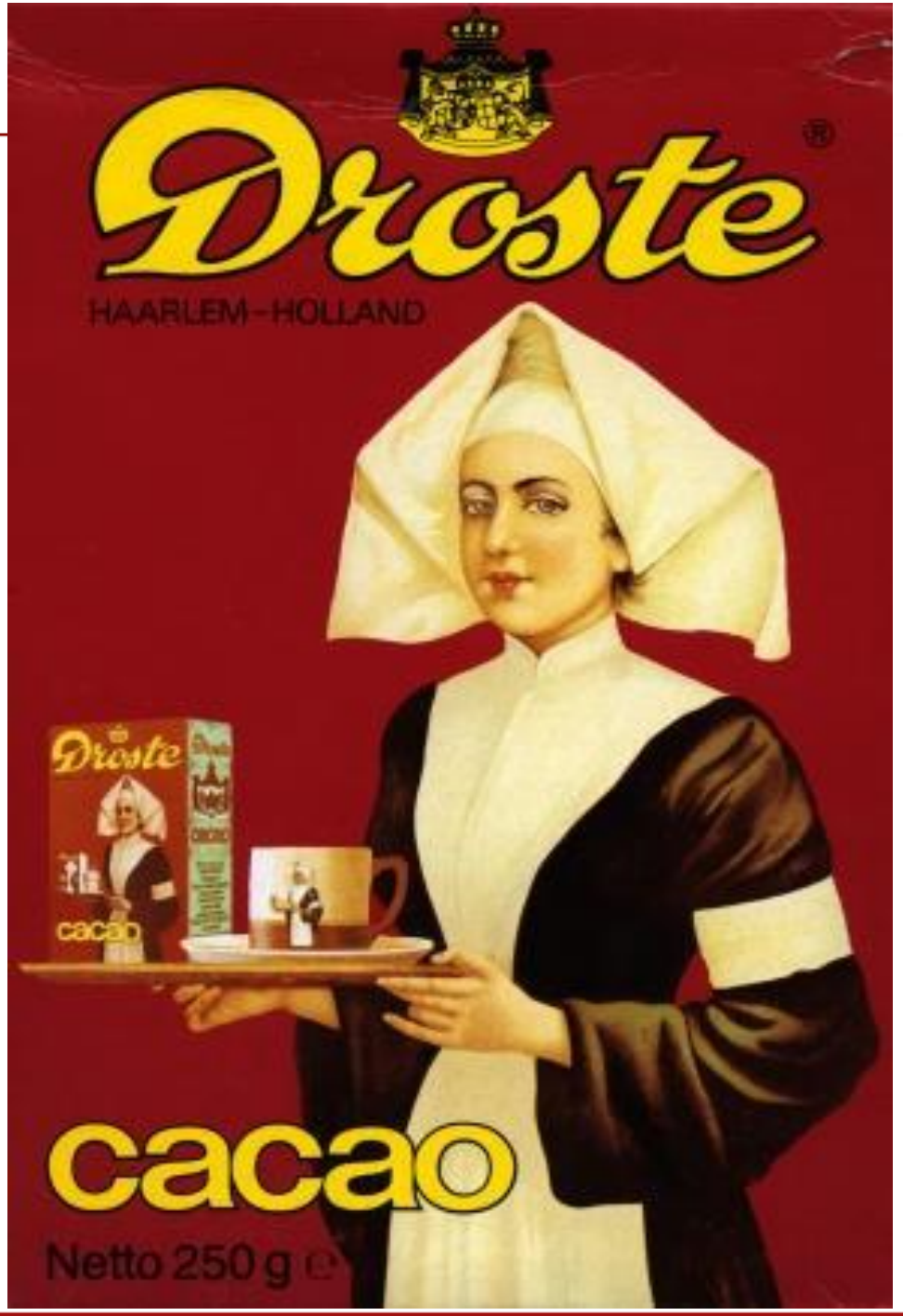
Linguagem de Programação

Estrutura de um programa

- Funções Recursivas
- Material: LP_Aula07

GOVERNO DO ESTADO SÃO PAULO

Funções Recursivas



Recursividade

- Capacidade que uma linguagem de programação tem de permitir que uma função possa invocar a si mesma.
- A recursividade pode ser direta ou indireta.
- Direta:
  - Quando uma função invoca a si mesma no seu corpo da função.
- Indireta:
  - Quando uma função ***f*** invoca uma outra função ***g*** que, por sua vez, volta a invocar a função ***f***.

Clássico: Fatorial de um número

- Problema:
  - Implemente uma função fatorial que calcula o valor de
    - n! = n * (n – 1) * (n – 2) * ….. * 2 * 1
  - Sabendo-se que 0!=1.

Versão tradicional – Sem uso de recursividade.

```c
#include <stdio.h>
#include <stdlib.h>

unsigned fat(int n){
    unsigned i;
    int res = 1; /* Pois vamos realizar multiplicações */
    for(i = 1; i <=n; i++){
        res*=i;
    }
    return res;
}
int main()
{
    int num;
    printf("Fatorial do número dado pelo usuário\n");
    printf("Digite o número: \n");
    scanf("%d",&num);
    printf("%d! = %d", num, fat(num));
    return 0;
}
```

Observe

- Ao observar a definição da função fatorial:
  - $n! = n * (n – 1) * (n – 2) * …. * 2 * 1$
- Nota-se que é equivalente a:
  - $n! = n * (n – 1)!$
- Pois:
  - $(n – 1)! = (n – 1) * (n – 2) * … * 2 * 1$
- Dessa forma, a definição de fatorial é realizada à custa do próprio fatorial:
  - $n! = n * (n – 1)!$

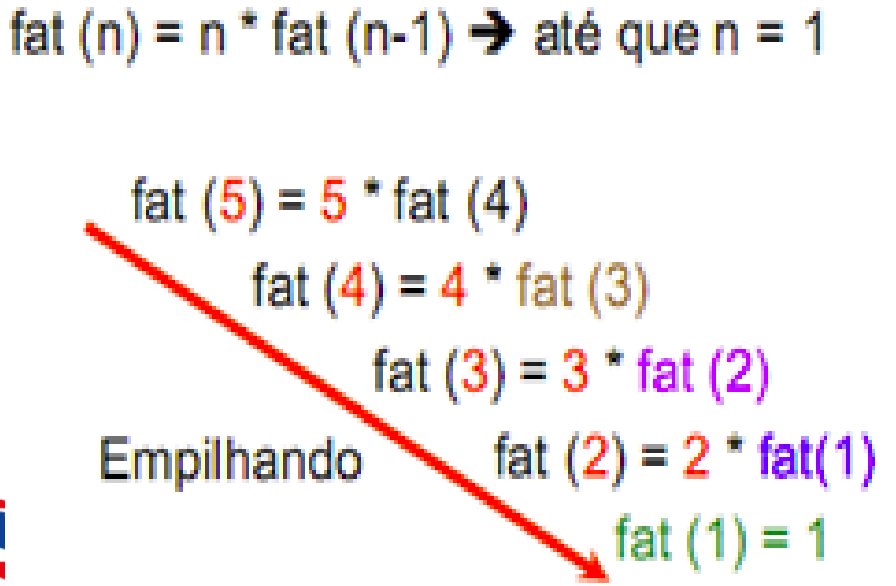
Uso de Recursividade

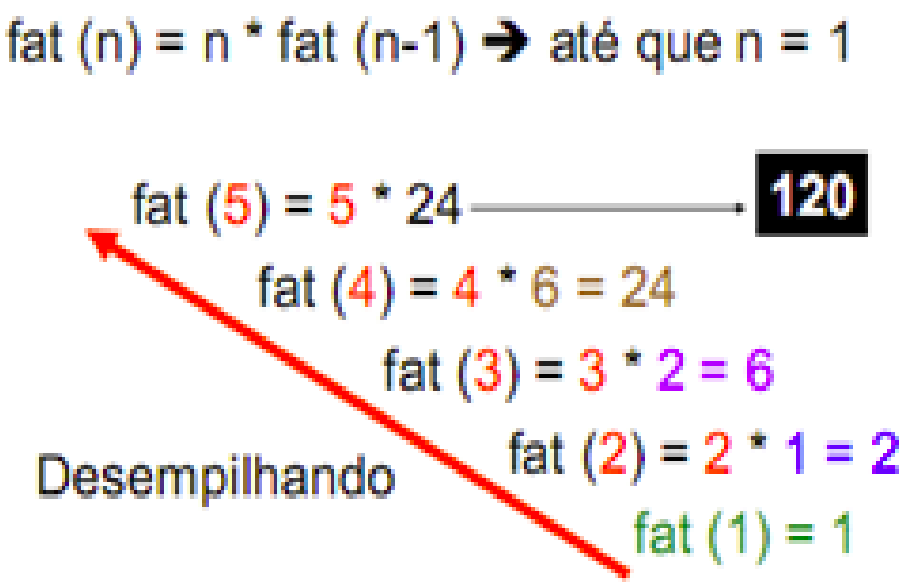
```c
#include <stdio.h>
#include <stdlib.h>

unsigned fat(int n){
    if(n==1)
        return 1; //Criterio para termino
    else
        return n * fat(n-1); // Chamada Recursiva
}
int main()
{
    int num;
    printf("Fatorial do número dado pelo usuário\n");
    printf("Digite o número: \n");
    scanf("%d",&num);
    printf("%d! = %d", num, fat(num));
    return 0;
}
```

Funcionamento da Chamada Recursiva

- Na execução do cálculo houve um empilhamento de chamadas da função fatorial(int n), até que a mesma atingisse a condição de saída, ou seja **numero = 1**. Acompanhe este processo nas imagens abaixo:





Regras para Escrita de Funções Recursivas

- 1) A primeira instrução deve ser a implementação de um critério de términos das chamadas.
- 2) Só depois de escrito o crtiério de término é que se deverá escrever a chamada recursiva da função.
- 3) Com a recursividade ganha-se, devido à menor quantidade de código escrito, maior legibilidade. Perde-se no entanto, em termos de performance.

Problema

- Implemente de forma recursiva, a função up que escreve na tela os n primeiros números de forma crescente.
- A função terá um único parâmetro e não retorna qualquer tipo de resultado.
- void up(int n)

Função up(int n)

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   void up(int n){
5       if(n<1) return;
6       up(n-1);
7       printf("%d\n", n);
8   }
9   int main()
10  {
11      int n;
12      printf("Digite o teto: ");
13      scanf("%d", &n);
14      up(n);
15  }
16
```

Função down(int n)

- Implemente de forma recursiva, a função down que escreve na tela os n primeiros números de forma decrescente.
- A função terá um único parâmetro e não retorna qualquer tipo de resultado.
- void down(int n)

Passagem de Parâmetros

Relembrando

- Uma função é composta por quatro partes distintas:
  - 1) Nome da Função
  - 2) Tipo de Retorno
  - 3) Lista de Parâmetros
  - 4) Corpo da Função

Número de argumentos e tipo

- O número e o tipo de argumentos passdos a uma função devem corresponder ao número e ao tipo de parâmetros com que essa foi definida.

Tipo de Retorno

- Uma função possui apenas um tipo de retorno (ou nenhum, se for procedimento – void).
- Essa característica traz alguns problemas, caso seja necessário passar para fora da função, mais do que um valor.
- A título de exemplo, suponhamos que queiramos implementar a função f, que calcula o número de elementos de um vetor de floats que são menores que zero, e retorne ainda o menor deles.

Tipo de Retorno (2)

- Para resolver esse problema passam-se as variáveis destinatárias dos resultados para a própria função, de forma que seja a própria função que carregue os valores nas variáveis.
- Para isso, necessitamos diferenciar os tipos de passagem:
  - Passagem de **parâmetros por valor**;
  - Passagem de **parâmetros por referência**.

Passagem por valor

- Passagem de parâmetros por valor é aquela que não envia a variável ou expressão para a função, e sim uma **cópia de seu valor**.

Passagem por referência

- Neste caso o que é enviado para a função não é uma cópia do valor da variável, mas sim a própria variável ou uma referência a esta (não se pode aplicar a expressões, apenas a variáveis).
- Dessa forma qualquer alteração nos parâmetros da função corresponde, na realidade, a uma alteração nas próprias variáveis referenciadas na invocação à função.

Função para troca de valores

```
 1: #include <stdio.h>
 2:
 3: void troca(int *a,int *b);  /* Protótipo da Função */
 4:
 5: main()
 6: {
 7:      int n,k;
 8:      puts("Introd. dois N°s Inteiros");
 9:      scanf("%d %d",&n,&k);
10:      printf("Antes da troca  n=%d e k=%d\n",n,k);
11:      troca(&n,&k);              /* Trocar n com k */
12:      printf("Depois da troca n=%d e k=%d\n",n,k);
13: }
14:
15: void troca(int *a , int *b)
16: {
17:      int tmp;
18:      tmp = *a;
19:      *a = *b;
20:      *b = tmp;
21: }
22:
```

Passagem de vetores para funções

- O nome de um vetor corresponde ao ENDEREÇO do seu primeiro elemento. Assim se s for um vetor:
  - s == &s[0]
- Em outras palavras, sempre que passamos um vetor para funções é enviado apenas o endereço original do vetor, e não o vetor na sua totalidade.

Programa para ordenar 10 valores de um vetor de inteiros

```c
#include <stdio.h>
#include <stdlib.h>
void carregarVetor(int num[10]);
void ordenarVetor(int num[10]);
void mostrarVetor(int num[10]);
int main()
{
    int num[10];
    printf("Carregando o Vetor de 10 posicoes\n");
    carregarVetor(num);
    printf("Ordenando Vetor......\n");
    ordenarVetor(num);
    printf("Mostrando o Vetor..... \n");
    mostrarVetor(num);
    return 0;
}
void carregarVetor(int num[10])
{
    int i;
    for(i=0;i<10;i++) {
        printf("Digite o %do valor: ", i+1);
        scanf("%d", &num[i]);
    }
}
```

[...]continuação

```
25    void ordenarVetor(int num[10])
26    {
27        int i, j, aux;
28        for(i=0;i<10;i++)
29            for(j=i;j<10;j++) {
30                if(num[i]>num[j]){
31                    aux = num[i];
32                    num[i] = num[j];
33                    num[j] = aux;
34                }
35            }
36    }
37    void mostrarVetor(int num[10])
38    {
39        int i;
40        for(i=0;i<10;i++)
41            printf("Vetor num[%d] = %d \n", i, num[i]);
42    }
43
```

Prof. Me. Marcos Roberto de Moraes, o Maromo

FIM

Referências Bibliográficas

DAMAS, L. M. D. **Linguagem C**. LTC, 2007.

HERBERT, S. **C completo e total**. 3a. ed. Pearson, 1997.