INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PIAUÍ
CAMPUS PICOS

ESTRUTURA DE DADOS

MAÍLA DE LIMA CLARO

claromaila@gmail.com

FUNÇÃO EM C

- Agrupa um conjunto de comandos e associa a ele um nome
  - O uso deste nome é uma chamada da função
- Após sua execução, programa volta ao ponto do programa situado imediatamente após a chamada
  - A volta ao programa que chamou a função é chamada de retorno.
- A chamada de uma função pode passar informações (argumentos) para o processamento da função
  - Argumentos = lista de expressões
    - Lista pode ser vazia
    - Lista aparece entre parênteses após o nome da função
    - Ex.
    - int Soma (int x, int y) {
    - }

O RETORNO DA FUNÇÃO

- No seu retorno, uma função pode retornar resultados ao programa que a chamou.
  - return (resultados);
  - O valor da variável local *resultados* é passado de volta como o valor da função.
- Valores de qualquer tipo podem ser retornados.
  - Funções predicado: funções que retornam valores
  - Procedimentos: funções que não retornam valores
  - Exemplo: void function (int x)

DEFINIÇÕES DE FUNÇÃO

- Funções são definidas de acordo com a seguinte sintaxe:

```
tipo_de resultado nome (lista de parâmetros)
{
        corpo de função
}
```

```
int MDC (int a, int b) {
    int aux;
    if (a < b) {
        aux = a;
        a = b;
        b = aux;
    }
    while (b != 0) {
        aux = b;
        b = a % b;
        a = aux;
    }
    return (a);
}
```

FUNÇÕES

- Definições de funções
  - Tipo de resultado
    - Quando a função é um procedimento, usa-se a palavra chave void
    - Procedimento não retorna valor
  - Lista de parâmetros
    - Funcionam como variáveis locais com valores iniciais
    - Quando função não recebe parâmetros, a lista de parâmetros é substituída pela palavra void

FUNÇÕES

- Funcionamento de uma chamada:
  - Cada expressão na lista de argumentos é avaliada
  - O valor da expressão é convertido, se necessário, para o tipo de parâmetro formal
    - Este tipo é atribuído ao parâmetro formal correspondente no início do corpo da função
  - O corpo da função é executado

FUNÇÕES

- Funcionamento de uma chamada:
  - Se um comando return é executado, o controle é passado de volta para o trecho que chamou a função
  - Se um comando return inclui uma expressão, o valor da expressão é convertido, se necessário, pelo tipo do valor que a função retorna
    - O valor então é retornado para o trecho que chamou a função
  - Se um comando return não inclui uma expressão nenhum valor é retornado ao trecho que chamou a função
  - Se não existir um comando return, o controle é passado de volta para o trecho que chamou a função após o corpo da função ser executado

PASSAGEM DE INFORMAÇÕES

- Exemplo:

```
double mesada (double notas, int idade) {
      double total;

      if (idade > 10)
          return (idade * 20.0);
      else{
          total =  notas*idade*20;
          return total;
      }
}
```

PASSAGEM DE INFORMAÇÕES

- Argumentos são passados **por valor**
  - Quando chamada, a função recebe o valor da variável passada
    - Quando argumento é do tipo atômico, a passagem por valor significa que a função não pode mudar seu valor
  - Os argumentos deixam de existir após a execução do método

FUNÇÕES

- Mecanismo do processo de chamada de função
- 1. Valor dos argumentos é calculado pelo programa que está chamando a função
- 2. Sistema cria novo espaço para todas as variáveis locais da função (estrutura de pilha)
- 3. Valor de cada argumento é copiado na variável parâmetro correspondente na ordem em que aparecem
  - 3.1 Realiza conversões de tipo necessárias

FUNÇÕES

- Mecanismo do processo de chamada de função
- 4. Comandos do corpo da função são executados até:
  - 4.1 Encontrar comando return
  - 4.2 Não existirem mais comandos para serem executados
- 5. O valor da expressão return, se ele existe, é avaliado e retornado como valor da função
- 6. Pilha criada é liberada
- 7. Programa que chamou continua sua execução

FUNÇÕES

- Uma função pode retornar qualquer valor válido em C, sejam de tipos pré-definidos (*int*, *char*, *float*) ou de tipos definidos pelo usuário (*struct*, *typedef*)
- Uma função que não retorna nada é definida colocando-se o tipo *void* como valor retornado
- Pode-se colocar *void* entre parênteses se a função não recebe nenhum parâmetro

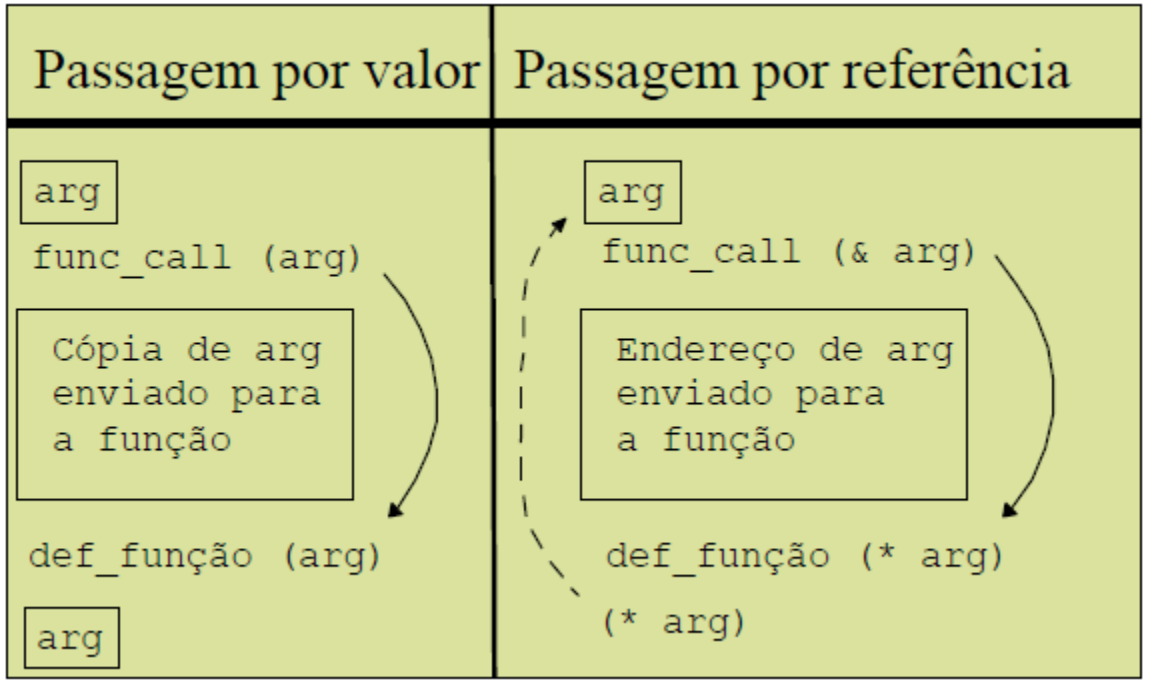
PASSAGEM DE PARÂMETROS

- Em C os argumentos para uma função são sempre passados por valor (*by value*), ou seja, *uma cópia* do argumento é feita e passada para a função

```
• void loop_count( int i ) {
• printf( "Em loop_count, i = " );
• while( i < 10 )
• printf ( "%d ", i++); ==> i = 2 3 4 5 6 7 8 9
• }
• void main( ) {
• int i = 2;
• loop_count( i );
• printf( "\nEm main, i = %d.\n", i ); ==> i = 2.
• }
```

PASSAGEM DE PARÂMETROS

- Como, então, mudar o valor de uma variável?
- Passagem de parâmetro por referência
- enviar o ***endereço*** do argumento para a função



PASSAGEM DE PARÂMETROS

- Passagem por referência:

```
void loop_count( int *i ) {
printf( "Em loop_count, i = " );
while( *i < 10 )
printf ( "%d ", (*i)++); ==> i = 2 3 4 5 6 7 8 9
}
void main( ) {
int i = 2;
loop_count( &i );
printf( "\nEm main, i = %d.\n", i ); ==> i = 10.
}
```

PRÁTICA: FUNÇÃO TROCA

- Fazer uma função *troca(px, py)* que recebe como parâmetros 2 ponteiros para inteiros e troca o conteúdo deles
- int x = 10, y = 20;
- troca(&x, &y);
- printf("x=%d y=%d", x, y) => x=20 y=10

```
void troca (int *px, int *py)
{
int temp;
temp=*px;
*px=*py;
*py=temp;
}
```

RECURSÃO EM C

- Uma função recursiva quando dentro do seu código existe uma chamada para si mesma.
- Em uma função recursiva, a cada chamada é criada na memória uma nova ocorrência da função com comandos e variáveis "isolados" das ocorrências anteriores.
- A função é executada até que todas as ocorrências tenham sido resolvidas.

RECURSÃO EM C

- A recursão é uma técnica que define um problema em termos de uma ou mais versões menores deste mesmo problema.
- Esta ferramenta pode ser utilizada sempre que for possível expressar a solução de um problema em função do próprio problema.
- Exemplo clássico: Fatorial
- n! = n* (n-1)!

RECURSÃO EM C

```
#include <stdio.h>

int fatorial(int n)
{
    if(n == 0)
        return 1;
    else if(n<0)
    {
        printf("Erro: fatorial de número negativo!\n");
    }
    return n * fatorial(n-1);
}
```

EXEMPLO FATORIAL

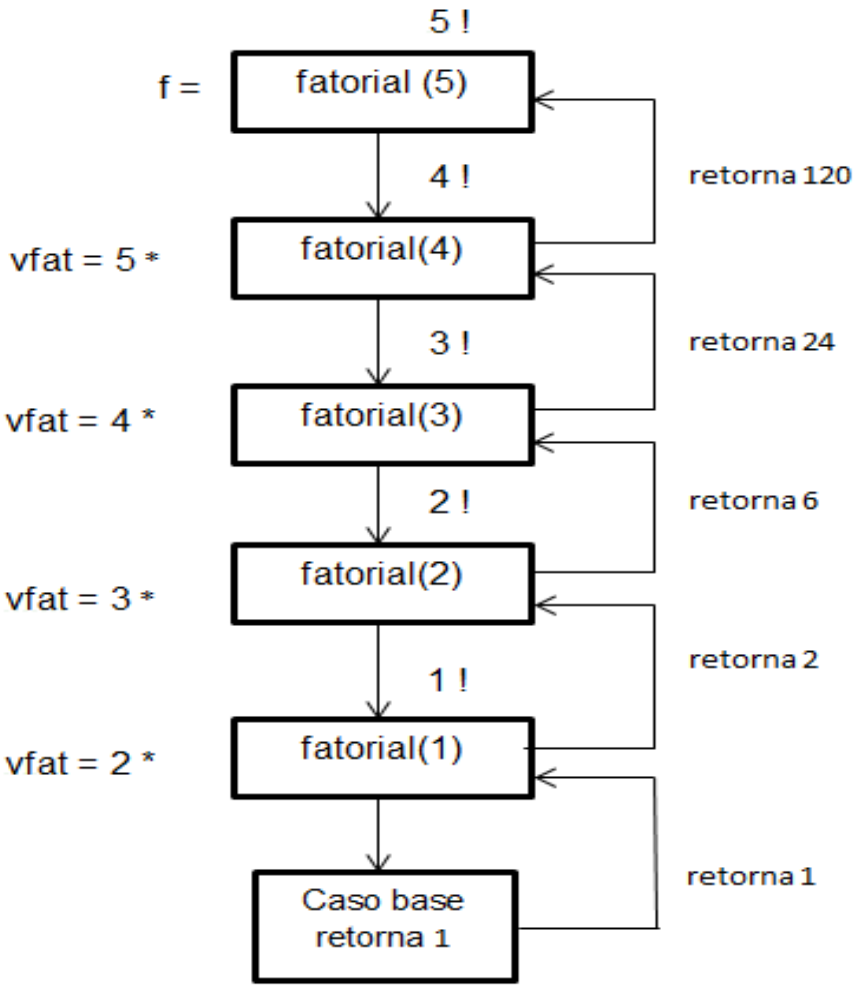
```
fatorial(5)
    => (5 ° 0)
       return 5 • fatorial(4)
       => (4 ° 0)
          return 4 • fatorial(3)
          => (3 ° 0)
             return 3 • fatorial(2)
             => (2 ° 0)
                return 2 • fatorial(1)
                => (1 ° 0)
                   return 1 • fatorial(0)
                   => (0 == 0)
                   <=  return 1
                <= return 1 • 1    (1)
             <= return 2 • 1       (2)
          <=  return 3 • 2         (6)
       <= return 4 • 6             (24)
    <= return 5 • 24               (120)
120
```

RECURSÃO EM C

- Porém um problema que surge ao usar a recursividade é como fazê-la parar. Caso o programador não tenha cuidado é fácil cair num loop infinito recursivo o qual pode inclusive esgotar a memória…
- Toda recursividade é composta por um **caso base** e pelas **chamadas recursivas**.
- **Caso base:** é o caso mais simples. É usada uma condição em que se resolve o problema com facilidade.
- **Chamadas Recursivas:** procuram simplificar o problema de tal forma que convergem para o caso base.

EXPLICANDO O CÓDIGO

- No programa acima, se o número n for menor ou igual a 1 o valor 1 será retornado e a função encerrada, sem necessidade de chamadas recursivas. Caso contrário dá-se início a chamadas recursivas até cair no caso mais simples que é resolvido e assim, as chamadas retornam valores de forma a solucionar o cálculo.
- Veja a figura a seguir que representa as chamadas recursivas para o cálculo de 5!



VANTAGENS E DESVANTAGENS

- **Vantagens da recursividade**
  - Torna a escrita do código mais simples e elegante, tornando-o fácil de entender e de manter.
- **Desvantagens da recursividade**
  - Quando o loop recursivo é muito grande é consumida muita memória nas chamadas a diversos níveis de recursão, pois cada chamada recursiva aloca memória para os parâmetros, variáveis locais e de controle.
  - Em muitos casos uma solução iterativa gasta menos memória, e torna-se mais eficiente em termos de performance do que usar recursão.

EXERCÍCIOS

1. Escreva uma função recursiva para calcular o valor de uma base x elevada a um expoente y.
2. Escrever uma função recursiva que retorna o tamanho de um *string*.
3. Fazer uma função recursiva que conta o número de ocorrências de um determinado caracter.
4. Escreva uma função recursiva que produza o reverso de um *string*.

CONTATO:

Maíla de Lima Claro

(claromaila@gmail.com)