



Linguagem C

Referência - II

■ Conceitos

- ❖ Uma das **estratégias** para se **resolver problemas complexos** é a **decomposição** dos **mesmos** em **partes menores**, mais **simples** de resolver;
- ❖ As **linguagens de programação de alto nível** (C, Pascal etc.) **oferecem o recurso de sub-rotina** para isso;
- ❖ **Lembrando**, uma **sub-rotina** representa um **conjunto** de **instruções** que possui um **nome associado** e opcionalmente **parâmetros**, e que pode ser **executada** como um **único comando**, sempre que a **solução** por ela **apresentada** for necessária;
- ❖ **Permitem modularizar um programa e reutilizar código.**

Funções em C

■ Conceitos

❖ Tipos de sub-rotinas

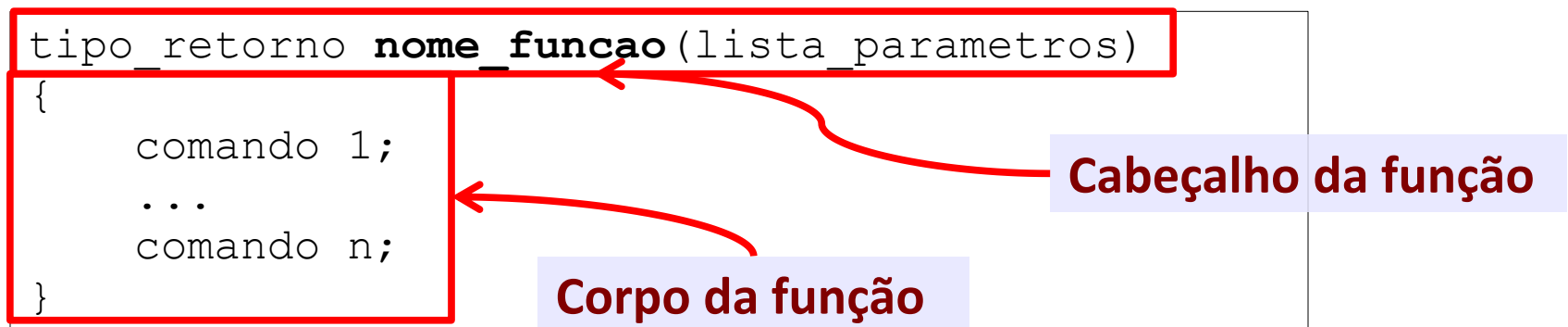
✱ **Função:** executa um conjunto de comandos e sua expressão resulta em um valor que pode ser utilizado diretamente em uma expressão ou atribuída a uma variável.

✧ **Exemplo:** `x = sin(log(y));`

✱ **Procedimento:** executa um conjunto de comandos, mas sua expressão não resulta em um valor particular.

✧ **Exemplo:** `ExibirValorResistor(f1, f2, f3);`

❖ **Em C, sub-rotinas são sempre funções e são genericamente definidas assim:**



■ Tipo de retorno de uma função

- ❖ O **tipo de retorno** de uma função **indica o tipo do valor** que **resultará** da sua **execução**;
- ❖ O **tipo** pode **ser qualquer** um dentre os tipos **primitivos** de C (**int**, **char** etc), **tipos definidos** pelo **usuário** (a ser futuramente estudado) e o **tipo void**;
- ❖ O tipo **void** declarado como **retorno** de uma **função indica** que a **função não resultará** em valor algum – esta é a maneira de como **escrever procedimentos** em **C**!

■ Lista de parâmetros de uma função

- ❖ Uma função pode ter **zero ou mais parâmetros**;
- ❖ Indica-se que uma **função não possui parâmetros** pelo **emprego** do tipo **`void`** em sua lista de parâmetros;
- ❖ Quando possui parâmetros, estes são escritos tal como declarações de variáveis: **nome do tipo seguido pelo nome do parâmetro**;
- ❖ O **tipo do parâmetro** pode ser **qualquer** um dentre os tipos primitivos de C (**`int`**, **`char`** etc), **tipos definidos pelo usuário**;
- ❖ Os parâmetros declarados na definição da função são denominados de **parâmetros formais**.

Funções em C

■ Exemplos de definição de funções em C

```
int main(void) {  
    /* ... */  
}
```

Função inteira, sem
parâmetros

```
void exhibir_raizes( double r1, double r2) {  
    /* ... */  
}
```

```
double calcular_pi(void) {  
    /* ... */  
}
```

Função “void”, com dois
parâmetros reais
(procedimento)

Função real, sem parâmetros

```
float calcular_imc(float massa, float altura) {  
    /* ... */  
}
```

Função real, com dois parâmetros reais

Funções em C

■ Corpo de uma função em C

- ❖ O corpo de uma função em C é identificado por um bloco `{ }` dentro do qual se escrevem as definições de variáveis locais e os comandos que serão executados pela função;
- ❖ Dentro do corpo da função, se ela retornar um valor de tipo diferente de `void`, deve-se especificar que valor é esse com o comando `return`;
- ❖ O comando `return` termina imediatamente a função no ponto em que ele foi inserido, produzindo o valor por ele indicado;
- ❖ Pode-se escrever diversos comandos `return` em uma mesma função (em condições, por exemplo).

Funções em C

■Corpo de uma função em C

❖Exemplos

*Calcular o índice de massa corpórea (IMC):

```
float calcular_imc(float massa, float altura ) {  
    float imc;  
    imc = massa / pow(altura, 2);  
    return imc;  
}
```

*Determinar o maior entre dois inteiros:

```
int maior( int a, int b ) {  
    if( a > b)  
        return a;  
    else  
        return b;  
}
```


Funções em C

■ Corpo de uma função em C

❖ Exemplos

* Função “void” que exibe a aprovação de um aluno:

```
void exibir_aprovacao(char nome[], float nota) {  
    if( nota >= 0 && nota <= 10.0 )  
        printf("O aluno %s esta %s.\n", nome,  
              (nota>=6.0)?"APROVADO":"REPROVADO");  
    else  
        printf("ERRO: nota invalida!\n");  
}
```

* A mesma função usando return vazio (analise ...):

```
void exibir_aprovacao(char nome[], float nota) {  
    if( nota < 0 || nota > 10.0 ){  
        printf("ERRO: nota invalida!\n");  
        return;  
    }  
    printf("O aluno %s esta %s.\n", nome,  
          (nota>=6.0)?"APROVADO":"REPROVADO");  
}
```

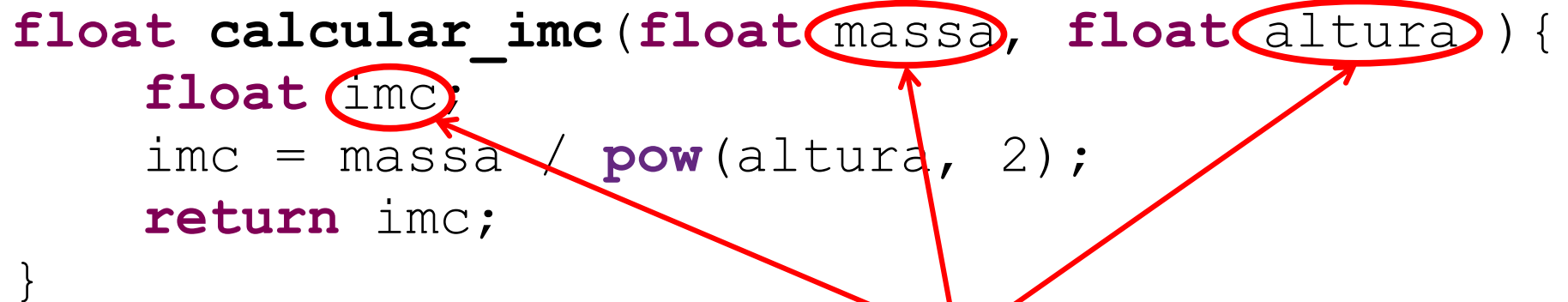
Termina a função aqui!

Funções em C

■ Variáveis locais de uma função

- ❖ Em uma função, as **variáveis declaradas como parâmetros e internas à função** são **locais** → **existem apenas durante a execução da função**;
- ❖ Se no programa houver uma **variável global de mesmo nome daquela local à função**, ela será ignorada, **prevalecendo a local**.

```
float calcular_imc(float massa, float altura) {  
    float imc;  
    imc = massa / pow(altura, 2);  
    return imc;  
}
```



Nomes locais à função

Funções em C

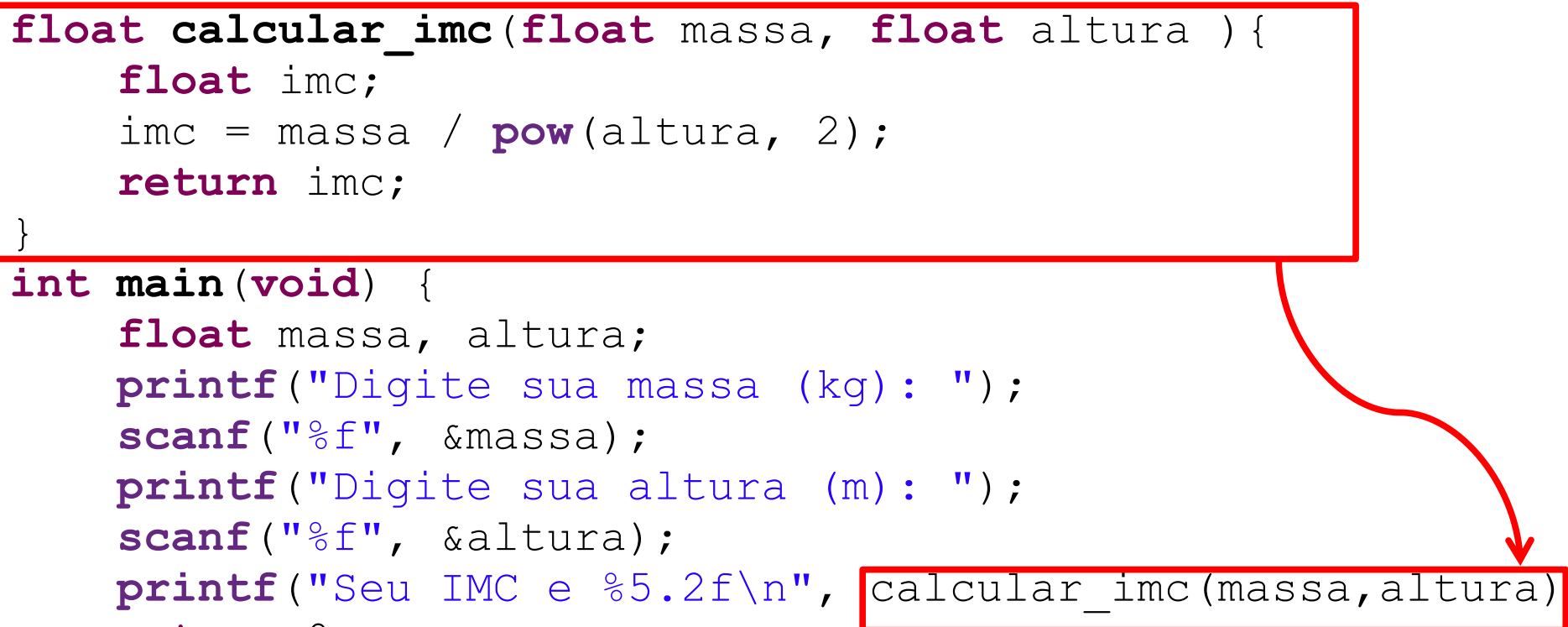
■ Escrevendo funções no programa

❖ Onde escrever as funções?

* R: Antes de serem usadas!

```
#include <stdio.h>
#include <math.h>
float calcular_imc(float massa, float altura ) {
    float imc;
    imc = massa / pow(altura, 2);
    return imc;
}

int main(void) {
    float massa, altura;
    printf("Digite sua massa (kg): ");
    scanf("%f", &massa);
    printf("Digite sua altura (m): ");
    scanf("%f", &altura);
    printf("Seu IMC e %5.2f\n", calcular_imc(massa, altura));
    return 0;
}
```



■ Protótipo de função

- ❖ O **protótipo** de uma **função** é uma **declaração** de função que **omite** o **corpo** mas **especifica** o seu **nome**, **tipo de retorno** e **lista de parâmetros**;
- ❖ Dessa forma, pode-se **definir** uma **função** depois da função `main()` ou mesmo em uma **unidade** de **arquivo** separada;
- ❖ Assim o **código** pode ser **compilado** sem problemas pois o **protótipo** **especifica** **exatamente** o que o **compilador** precisa para **realizar** esta tarefa;
- ❖ Já a **definição** da **função**, ao ser encontrada, **permitirá** a **criação** do **código** **final** **executável**, pois **contém** as **instruções** que **implementam** a **função**.

Funções em C

■ Protótipo de função

❖ Exemplo

```
#include <stdio.h>
#include <math.h>
/* Este é um protótipo de função */
void exibir_diagnosticos(float imc);
/* Outra forma: omitir os nomes de parâmetros */
float calcular_imc(float, float );
int main(void) {
    float massa, altura, imc;
    printf("Digite sua massa (kg): "); scanf("%f", &massa);
    printf("Digite sua altura (m): "); scanf("%f", &altura);
    imc = calcular_imc(massa, altura);
    exibir_diagnosticos(imc);
    return 0;
}

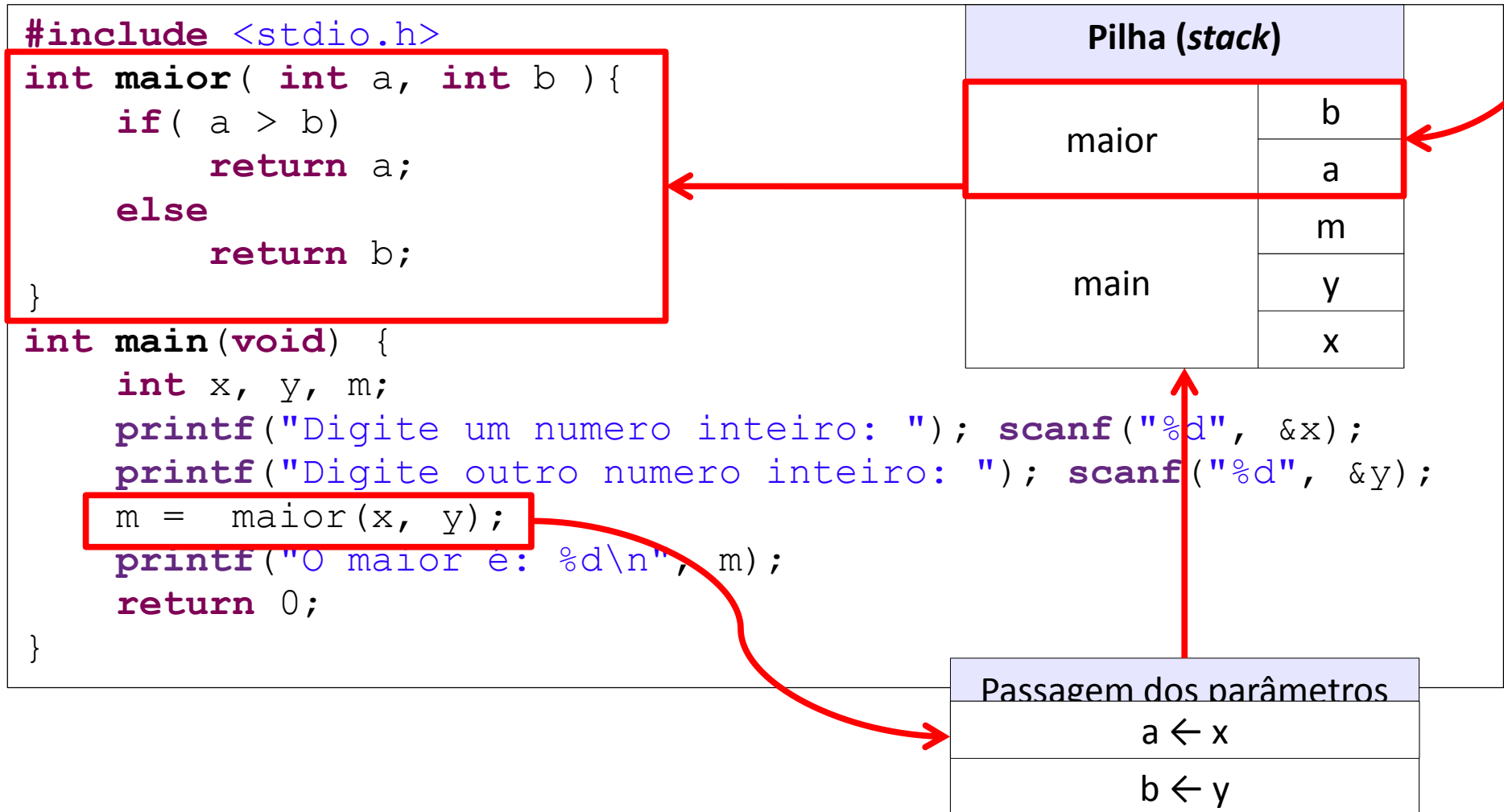
float calcular_imc(float massa, float altura ){
    float imc;
    imc = massa / pow(altura, 2);
    return imc;
}

void exibir_diagnosticos(float imc){
    if( imc < 17 ) printf("Muito abaixo do ideal");
    else if(imc < 18.49 ) printf("Abaixo do ideal");
    else if(imc < 24.99 ) printf("Ideal");
    else if(imc < 29.99 ) printf("Acima do ideal");
    else if(imc < 34.99 ) printf("Obesidade I");
    else if(imc < 39.99 ) printf("Obesidade II");
    else if(imc > 39.99 ) printf("Obesidade III");
}
```

Funções em C

■ Mecanismo de chamada de função

Removido da pilha quando a função `maior` terminar



- a e b são os parâmetros formais;
- x e y são os parâmetros reais

Funções em C

■ Passagem por valor e referência

❖ Passagem por valor

• A função faz uma **cópia** dos **valores** dos **parâmetros reais** para os **parâmetros formais**;

• Os **parâmetros reais** e **formais** são **independentes**!

```
...  
int x, y, m;  
...  
m = maior(x, y);  
...
```

} Em main()

```
int maior( int a, int b ) {  
    if ( a > b )  
        return a;  
    else  
        return b;  
}
```

} Em maior()

Memória	
m	?
y	3
x	4

Antes de chamar maior()

Passagem dos parâmetros	
a	← x
b	← y

Durante maior()

Memória	
b	3
a	4
m	?
y	3
x	4

Memória	
m	4
y	3
x	4

Depois de chamar maior()

Funções em C

■ Passagem por valor e referência

❖ Passagem por referência

- ✱ Na **passagem de valor por referência**, o **parâmetro formal** torna-se uma **referência** para o **parâmetro real**, ou seja, ele “aponta” para a **mesma posição de memória** que **parâmetro real** representa;

- ✱ **Consequência**: ao se **alterar o conteúdo** da **memória** referenciada pelo **parâmetro formal**, altera-se também o **conteúdo** da **memória** do **parâmetro real**;

✱ Aplicações:

- ✧ **Funções** que precisam **retornar** valores por meio de **parâmetros** – a função `scanf` de `stdio.h` é um exemplo;

- ✧ **Passagem mais eficiente de parâmetros**: uma **referência** é na realidade o **endereço de memória** que uma **variável** representa. **Endereços** são sempre **números inteiros**. Logo, para **passar grandes massas de dados** para dentro de **funções**, **referência** é a **melhor opção**, pois consiste apenas em um inteiro sem sinal.

Funções em C

■ Passagem por valor e referência

❖ Passagem por referência

- ✱ Indica-se que um parâmetro de um tipo T será passado por referência à uma função por meio de um ponteiro para o tipo T , escrito assim: T^* ;
- ✱ Ponteiros em C são variáveis que armazenam endereços de memória de outras variáveis;
- ✱ Então $T^* p$ indica que p é um ponteiro que pode armazenar (ou apontar para) o endereço de um outra variável do tipo T ;
- ✱ Em C, para obter um endereço de uma variável utiliza-se o operador “endereço de”: $\&$;

✱ Exemplo:

```
int q = 10;  
int *p = &q;
```

p “aponta” para q

Memória		
Posição	Valor	
...
3411	p	5671
...
5671	q	10

Funções em C

■ Passagem por valor e referência

❖ Passagem por referência

- Se **p** é uma variável ponteiro que já foi definida, a expressão **p** representa apenas o endereço que está armazenado em **p**;
- Se **p** é uma variável ponteiro que já foi definida, a expressão ***p** representa o conteúdo (valor) armazenado no endereço representado pelo ponteiro:

```
#include <stdio.h>
int main(void) {
    int q = 10;
    int *p = &q;
    printf("Valor armazenado em q: %d\n", q);
    printf("Endereco de q: %p\n", &q);
    printf("Valor armazenado em p: %p\n", p);
    printf("Conteúdo do endereco de memoria "
           "armazenado em p: %d\n", *p);
    return 0;
}
```

Execute o programa ao lado e então responda:

- O que significa `int *p`?
- O que é `&q`?
- Qual a relação entre `p` e `&q`?
- O que está armazenado em `p`?
- Qual a relação entre `*p` e `q`?

■ Passagem por valor e referência

❖ Passagem por referência

★ Exemplo: lembra do procedimento `inc` do Pascal?

```
#include <stdio.h>
void inc(int *x) {
    *x = *x + 1; /*ou apenas (*x)++ */
}
int main(void) {
    int a = 10;
    printf("Valor original de a: %d\n", a);
    inc( &a );
    printf("Valor alterado de a: %d\n", a);
    return 0;
}
```

★ A variável `a` da função `main` é manipulada indiretamente pelo ponteiro `x` da função `inc`!

Funções em C

■ Passagem por valor e referência

❖ Passagem por referência

* Exemplo: trocar o valor armazenado em duas variáveis.

```
#include <stdio.h>
void trocar(int *x, int *y) {
    int t = *x; /* salvar o conteúdo de uma delas */
    *x = *y;     /* copiar o conteúdo de uma para outra */
    *y = t;     /* recuperar o valor salvo anteriormente */
}
int main(void) {
    int a = 10, b = 20;
    printf("Valores originais: de a = %d e b = %d\n", a, b);
    trocar( &a, &b );
    printf("Depois da troca: de a = %d e b = %d\n", a, b);
    return 0;
}
```

* Por utilizar **ponteiros**, a **troca é efetuada e permanece quando a função trocar é finalizada**. Tente **escrever trocar sem o uso de ponteiros e veja a diferença ...**

■ Funções recursivas

❖ **Recursão** permite **resolver** um **problema complexo** pela sua **redução** em **subproblemas** que são

(1) Idênticos em **estrutura** ao **problema original**;

(2) Mais simples de **resolver**.

❖ É uma **técnica** que é **muito empregada** na **Matemática**:

$$n! = \begin{cases} n(n-1)! & n > 0 \\ 1 & n = 0 \end{cases}$$

Munido de lápis e papel, calcule 5! com a definição acima.

■ Funções recursivas

- ❖ Em C, uma função recursiva é aquela que em algum ponto da função executa ela própria, com parâmetros e retornos adequados;
- ❖ Deve-se sempre especificar um caso que pare a recursão – caso contrário seu programa entrará em uma repetição infinita e travará ...
- ❖ O que acontece com as chamadas da função durante uma recursão?
- ✱ R: elas são empilhadas. Somente quando a função atinge o caso de parada (resultado concreto) é que os valores intermediários são desempilhados e o resultado vai sendo formado.

■ Funções recursivas

❖ Exemplo: função fatorial.

$$n! = \begin{cases} n(n-1)! & n > 0 \\ 1 & n = 0 \end{cases}$$

```
#include <stdio.h>
int fatorial( int n ) {
    if( n > 0 )
        return n*fatorial(n-1);
    else
        return 1;
}
int main(void) {
    int n;
    printf("Digite um numero inteiro: ");
    scanf("%d", &n);
    printf("Seu fatorial é: %d\n", fatorial(n));
    return 0;
}
```

Compare os dois lados ...

Bibliografia

- DEITEL, H. M. **C: Como programar**. Trad. de João Eduardo Nóbrega Tortello; rev. téc. Alvaro Rodrigues Antunes. São Paulo, SP: Pearson Education, 2003. 1153 p. ISBN 8534614598.
- SCHILDT, Herbert. **C Completo e total**. [Título original: C: the complete reference]. Trad. e rev. téc. Roberto Carlos Mayer. 3. ed. São Paulo, SP: Pearson Education do Brasil, 2011. 827 p. ISBN 9788534605953.