

Subprogramação

Programando com funções e procedimentos

**Ponteiros: passando referências para
variáveis como argumento para
funções**

Na aula passada...

- Programação Estruturada
 - Desenvolvimento de algoritmos por fases ou refinamentos sucessivos
 - Uso de um número muito limitado de estruturas de controle em casa fase
 - Decomposição do algoritmo total em módulos desenvolvidos usando **subprogramas**

Dividir para conquistar!



Subprograma

Função ou Procedimento

- **Funções pré-definidas**

- Disponibilizadas juntamente com os compiladores
- Incluídas no programa principal através de *headers*
- `#include <math.h>`

- **Funções desenvolvidas pelo programador**

- Um programa pode incluir diversas funções
- A declaração e implementação dessas funções fica a cargo do programador

Argumentos e Retorno

- Uma função pode **receber** dados de entrada, utilizados localmente para executar os comandos incluídos na função: estes dados são chamados de **parâmetros ou argumentos**
- Uma função pode também **retornar** um valor, o que chamamos de **retorno de função**

Exemplos de chamada:



```
x = pow(3, 5); //Retorna 243.00 em x
y = cos(0);    //Retorna 1.00 em y
```

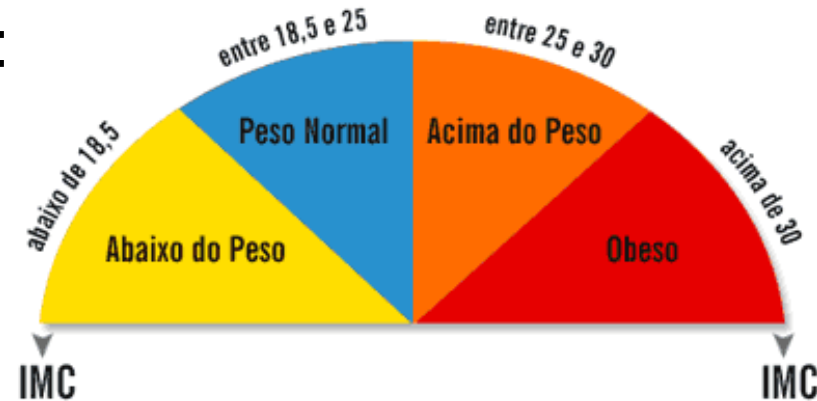
Argumentos e Retorno

- Funções podem ser de 4 tipos

	Sem Argumentos	Com Argumentos
Sem Retorno	Não recebem nada e não retornam nada. Ex: Imprimir uma informação fixa na tela	Recebem um ou mais argumentos e não retornam nada. Ex: Imprime na tela um resultado de uma computação
Com Retorno	Não recebem nada e retornam apenas um valor Ex: Ler um valor do teclado e retorna-lo	Recebem um ou mais argumentos e retornam apenas um valor. Ex: Realiza um cálculo a partir dos argumentos e retorna o resultado

Calculando o IMC

- Fazer uma função para calcular o Índice de Massa Corpórea (IMC)
 - Receber dois argumentos:
 - o peso em quilos (Kg);
 - a altura em metros (m);
 - $IMC = \frac{peso}{altura^2}$
 - Retornar o valor do IMC
- No main, apresentar a mensagem apropriada de acordo com o valor retornado



Declaração de funções com retorno, com ou sem argumentos

- Passando dois argumentos para a função

Tipo de retorno Nome da função Argumentos

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

O tipo de retorno
indica o tipo de valor
retornado pela função

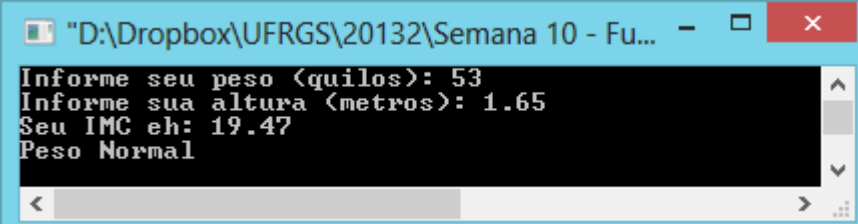
- Chamada da função declarada (a partir do main por exemplo)

```
float i;  
i = imc(65.0, 1.6);
```

Uma variável local do mesmo
tipo pode receber o valor
retornado pela função no main

Exemplo do IMC completo

```
1  /* Escrita de numeros inteiros */
2  #include <stdio.h>
3  #include <math.h>
4  float imc(float peso, float altura){
5      float imc;
6      imc = peso / pow(altura, 2);
7      return imc;
8  }
9  int main(){
10     float p, a, i;
11     printf("Informe seu peso (quilos): ");
12     scanf("%f", &p);
13     printf("Informe sua altura (metros): ");
14     scanf("%f", &a);
15     i = imc(p, a);
16     printf("Seu IMC eh: %.2f\n", i);
17     if (i < 18.5f){
18         printf("Abaixo do Peso\n");
19     }else if(i <= 25.0f){
20         printf("Peso Normal\n");
21     }else if(i <= 30.0f){
22         printf("Acima do Peso\n");
23     }else{
24         printf("Obeso\n");
25     }
26     return 0;
27 }
```



```
D:\Dropbox\UFRGS\20132\Semana 10 - Fu...
Informe seu peso (quilos): 53
Informe sua altura (metros): 1.65
Seu IMC eh: 19.47
Peso Normal
```

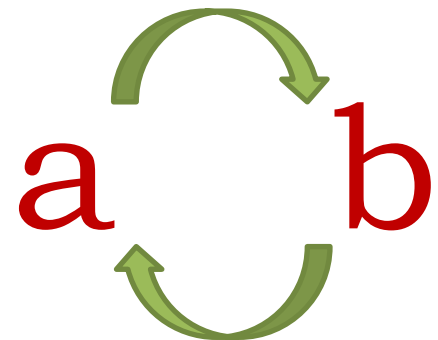

... até agora vimos

- Um argumento passado para uma função funciona da mesma forma que uma variável **local** à função
- Manipular ou alterar o valor dos argumentos dentro do escopo de uma função **não altera seus valores originais** no programa que chamou a função

Exemplo 1: Trocar os valores de duas variáveis

- Escrever uma função que receba dois argumentos inteiros (a e b) e troque o valor das duas variáveis
 - Os valores devem “retornar” alterados ao programa que chamou a função
 - Lembre que não é possível retornar mais de um valor de uma função

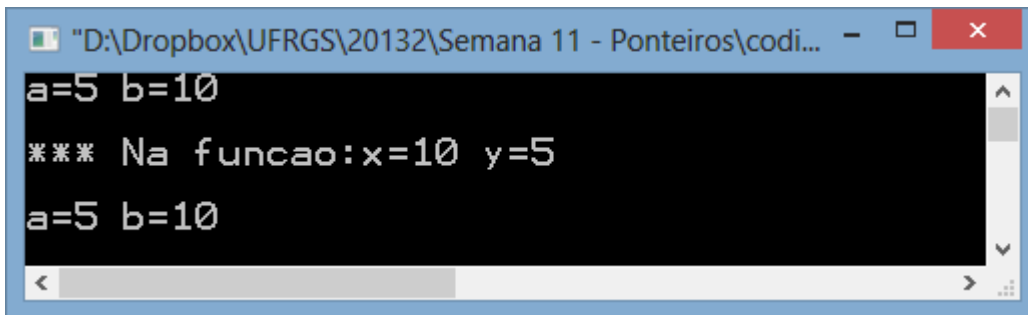
Como escrever esse tipo de função?



Uma tentativa frustrada ☹️

```
void troca(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
    printf("\n*** Na funcao:");
    printf("x=%d y=%d\n", x, y);
}
```

```
int main()
{
    int a=5, b=10;
    printf("a=%d b=%d\n", a, b);
    troca(a, b);
    printf("\na=%d b=%d\n", a, b);
}
```



```
"D:\Dropbox\UFRGS\20132\Semana 11 - Ponteiros\codi...
a=5 b=10
*** Na funcao:x=10 y=5
a=5 b=10
```

Vamos precisar
usar ponteiros!

Passagem de argumentos para funções

- Por valor

- passa uma **cópia do valor** da variável original no momento da chamada da função
- não existe relação entre a variável utilizada na chamada e dentro da função
- ao final da execução da função o valor original da variável usada na chamada permanece o mesmo

- Por referência

- passa um **endereço** de memória, o qual é associado ao parâmetro, localmente
- alterações feitas no conteúdo do endereço **afetam** a variável referida na chamada
- para passar parâmetros por referência em C precisamos **usar ponteiros**

Ponteiros

- Mecanismo para referenciar algo indiretamente
 - Diferenciar entre o **endereço** de um assento no show e a **pessoa** que está sentada no assento específico



Maria está sentada no assento
50 da fila 118

Código do local: a50f118



Variáveis e Endereços

- Toda variável utilizada por um programa reside em determinado **endereço de memória**
- O acesso ao endereço de uma variável pode ser feito **simbolicamente** através de seu nome
- Essa referência é simbólica porque o **endereço de memória** propriamente dito **não é especificado**

A declaração:

int x;



Variável	Valor da Variável	Endereço de Memória
x	?	AFA1

O comando:

x = 3;

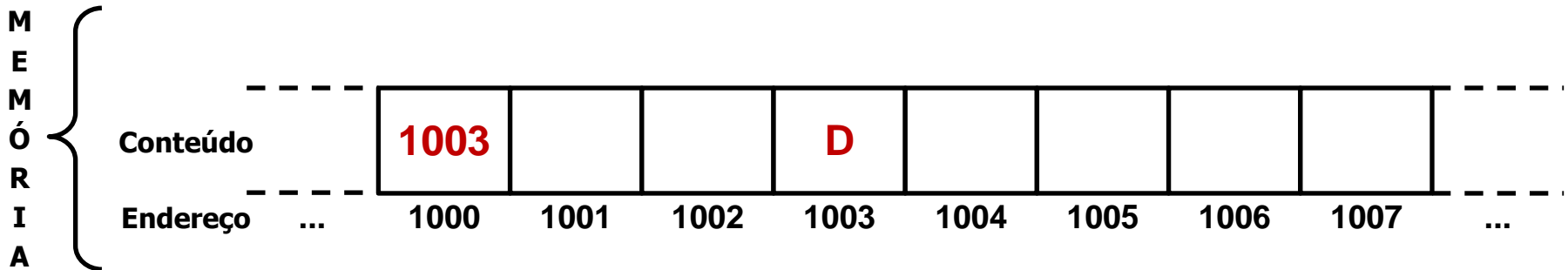


Variável	Valor da Variável	Endereço de Memória
x	3	AFA1

AFA1 = 1010 1111 1010 0001 = 44961

O que são ponteiros?

- Ponteiros são **variáveis** que guardam **endereços de memória**



- ✓ A posição de memória 1000 armazena o endereço de memória 1003, cujo conteúdo é o caractere D
- ✓ O endereço é a posição de uma outra variável na memória
- ✓ Se uma variável **a** armazena o endereço de uma variável **b**, dizemos que **a aponta para b**

Declarando ponteiros

- Sintaxe

tipo *****nome;

Qualquer tipo válido em C

Nome da variável

Asterisco indica que a variável armazena um **endereço de memória**, cujo conteúdo é do tipo especificado

- Exemplos:

```
// f é um ponteiro para variáveis do tipo float
```

```
float *f;
```

```
// p é um ponteiro para variáveis do tipo inteiro
```

```
int *p;
```

```
// pode declarar junto com variáveis de mesmo tipo
```

```
char a, b, *p;
```

Nota: Não confundir o * com o operador aritmético de multiplicação

Operadores de Ponteiros

& operador unário, que devolve o endereço de memória de seu operando

Exemplo:

```
int count, q, *m;  
m = &count;
```

Lê-se: “***m** recebe o endereço de **count***”

***** operador unário que devolve o valor da variável localizada no endereço que o segue

Exemplo:

```
int count, q, *m;  
q = *m;
```

Lê-se: “***q** recebe o valor contido no endereço **m***”

Voltando ao show do U2

Maria está sentada no
assento 50 da fila 118



Qual o “conteúdo” de
a50f118?

***a50f118 = Maria**

João está sentado no
assento 43 da fila 77



Onde está sentado Joao?

&Joao = a43f77

O ingresso no assento
23 da fila 102 não foi
vendido



Qual o “conteúdo” de
a23f102?

***a23f102= ?**

LIXO! Não podemos garantir o conteúdo

Atribuições com ponteiros

- Como qualquer variável, um ponteiro pode ser usado no lado direito de um comando de atribuição para passar seu valor para um outro ponteiro
- Exemplo:

```
int x = 200, *p1, *p2;
```

```
// p1 aponta para x
```

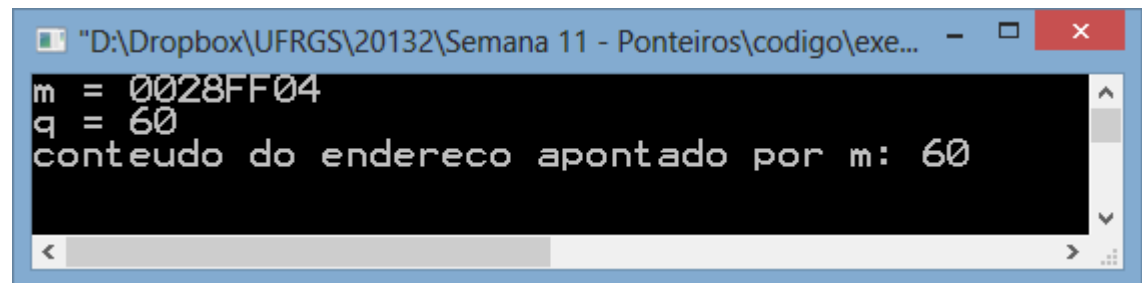
```
p1 = &x;
```

```
// p2 recebe p1 e também passa a apontar para x
```

```
p2 = p1;
```

Exemplo de atribuição de ponteiros

```
1  #include <stdio.h>
2  int main()
3  {
4      int cont, q, *m;
5
6      m = &cont;    // recebe endereço de count
7
8      cont = 60;
9      q = *m;        // recebe valor apontado por m
10
11     printf("m = %p\n", m); // %p para imprimir ponteiro (endereço apontado)
12     printf("q = %d\n", q);
13     // conteúdo apontado por m
14     printf("conteudo do endereço apontado por m: %d\n\n", *m);
15     return 0;
```



```
"D:\Dropbox\UFRGS\20132\Semana 11 - Ponteiros\codigo\exe...
m = 0028FF04
q = 60
conteudo do endereço apontado por m: 60
```

Outro Exemplo

```
1  #include <stdio.h>
2  int main()
3  {
4      int a, *x, *y;
5      a = 10;
6      x = &a;
7      y = x;
8      printf("a = %d, *x = %d, *y = %d \n", a, *x, *y);
9      printf("a = %d, x = %p, y = %p\n\n", a, x, y);
10     *x = 20;
11     printf("a = %d, *x = %d, *y = %d \n", a, *x, *y);
12     printf("a = %d, x = %p, y = %p\n\n", a, x, y);
13     return 0;
14 }
```

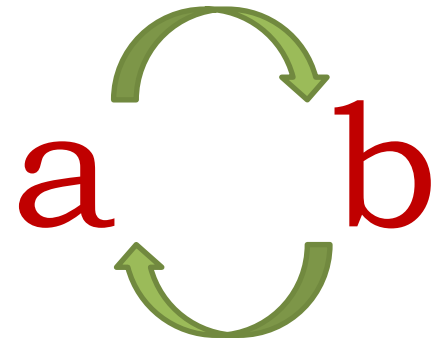
```
"D:\Dropbox\UFRGS\20132\Semana 11 - Ponteiros\codi...
a = 10, *x = 10, *y = 10
a = 10, x = 0028FF04, y = 0028FF04

a = 20, *x = 20, *y = 20
a = 20, x = 0028FF04, y = 0028FF04
```

Retomando a função que troca os valores de duas variáveis

- Escrever uma função que receba dois argumentos inteiros (a e b) e troque o valor das duas variáveis
 - Os valores devem “retornar” alterados ao programa que chamou a função
 - Lembre que não é possível retornar mais de um valor de uma função

Resolver usando ponteiros!!!



Solução utilizando ponteiros

```
1 // função recebe endereços e troca seus conteúdos
2 void troca(int *x, int *y) {
3     int temp;
4     temp=*x;
5     *x = *y;
6     *y = temp;
7     printf("\n*** Na funcao:\n");
8     printf("Valores: x = %d; y = %d\n", *x, *y);
9     printf("Enderecos: x = %p; y = %p\n", x, y);
10 }
11 int main() {
12     int a=5, b=10;
13     printf("\n*** No main:\n");
14     printf("Valores: a = %d; b = %d\n", a, b);
15     printf("Enderecos: a = %p; b = %p\n", &a, &b);
16     troca(&a, &b);
17     printf("\n*** De volta ao main:\n");
18     printf("Valores: a = %d; b = %d\n", a, b);
19     printf("Enderecos: a = %p; b = %p\n", &a, &b);
20     return 0;
21 }
```

A função recebe dois ponteiros para inteiros como argumentos

Na passagem dos argumentos usamos os endereços de memória das variáveis declaradas no main

Solução utilizando ponteiros

- Saída do programa implementado

```
*** No main:  
Valores: a = 5; b = 10  
Enderecos: a = 0028FF0C; b = 0028FF08  
*** Na funcao:  
Valores: x = 10; y = 5  
Enderecos: x = 0028FF0C; y = 0028FF08  
*** De volta ao main:  
Valores: a = 10; b = 5  
Enderecos: a = 0028FF0C; b = 0028FF08
```

Execução
da função

Execução
do main

Os valores foram trocados, pois a passagem de parâmetros foi feita por **referência**.

Dentro do subprograma, x e y receberam os **endereços de memória** de a e b, o seus conteúdos são alterados.

Exemplo 2: Calcular as raízes de uma equação

- Escreva uma função em C que receba três valores reais **a**, **b** e **c**, e que calcule as duas raízes **x1** e **x2** da equação:

$$ax^2 + bx + c = 0$$

- As raízes deverão ser repassados ao programa que chama a função
- Assuma que a equação de segundo grau possui raízes reais

Como resolver, se uma função tipada pode retornar apenas um único valor???

Exemplo 2: Calcular as raízes de uma equação

- **Solução:** usar passagem de parâmetros por **referência** para repassar os valores das duas raízes
- **Estrutura da função:** os parâmetros **a**, **b** e **c** devem ser passados por valor, pois não serão alterados
- Os parâmetros **x1** e **x2** devem ser passados por referência, para que possam ser acessados pelo programa que chama a função

```
void bhaskara(float a, float b, float c, float *x1, float *x2)
{
...
}
```

Exemplo 2: Calcular as raízes de uma equação

```
1  /* Exemplo de rotina que acha raízes reais de uma equação de 2º grau
2     Entradas: valores a,b,c dos coeficientes
3     Saídas: valores das raízes x1 e x2 (referências) */
4  #include<stdio.h>
5  #include<math.h>
6
7  void bhaskara(float, float, float, float*, float*); //protótipo
8
9  int main() {
10     float r1, r2;
11     bhaskara(-1, 3, 9, &r1, &r2);
12     printf("x1 = %f -- x2 = %f", r1, r2);
13 }
14
15 // função bhaskara. Entradas a,b,c (passagem por valor) e saídas x1, x2
16 // Assume-se que as raízes são reais
17 void bhaskara(float a, float b, float c, float* x1, float* x2) {
18     float raizdisc;
19     raizdisc = sqrt(b*b-4*a*c); // raiz do discriminante
20     *x1 = (-b + raizdisc)/(2*a);
21     *x2 = (-b - raizdisc)/(2*a);
22 }
```

Ponteiros e Arranjos

- Ponteiros são utilizados na manipulação de arranjos
- O nome de um vetor é um **ponteiro** que aponta para o **primeiro elemento** do arranjo
 - Se **v** for um vetor, então **v == &v[0]**
- Considerando:
 - `int v[3] = {10, 20, 30}; //vetor c/ 3 inteiros`
 - `int *ptr; //ponteiro p/ inteiro`
- Existem 2 formas de fazer com que **ptr** aponte para **v**:
 - `ptr = &v[0];`
 - `ptr = v;`
- O ponteiro **v** não deveria ser alterado durante a execução do programa; mas **ptr** PODE!

Exemplo 3: Ordenar um vetor

- Escreva uma função que ordene em ordem crescente um vetor de 10 inteiros passado como argumento
 - Ler os valores do vetor no programa principal
 - Ordenar o vetor usando a função definida
 - Utilize um método de ordenação qualquer (ex: *bubble sort*)
 - Imprimir o vetor ordenado novamente no programa principal
- Dicas:
 1. O nome do vetor já é um ponteiro para a sua primeira posição (não é necessário usar o & para passa-lo para a função)
 2. Dentro da função podemos acessar as posições do vetor normalmente usando os colchetes []

Tente fazer em casa!