



INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA  
CAMPUS FLORIANÓPOLIS  
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA

# PROGRAMAÇÃO DE COMPUTADORES I

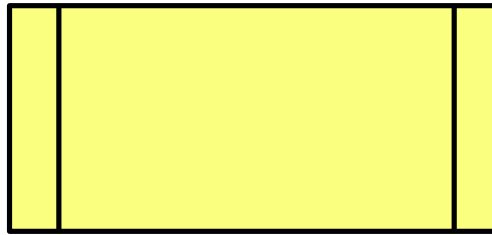
## Undécima Parte

# Capítulo V

## SUBPROGRAMAS & PONTEIROS 1ª Parte

Marco Villaça  
Fernando Pacheco

# Sumário



Símbolo de Subprograma  
para fluxogramas

- Subprogramas
- Funções em C
- Ponteiros

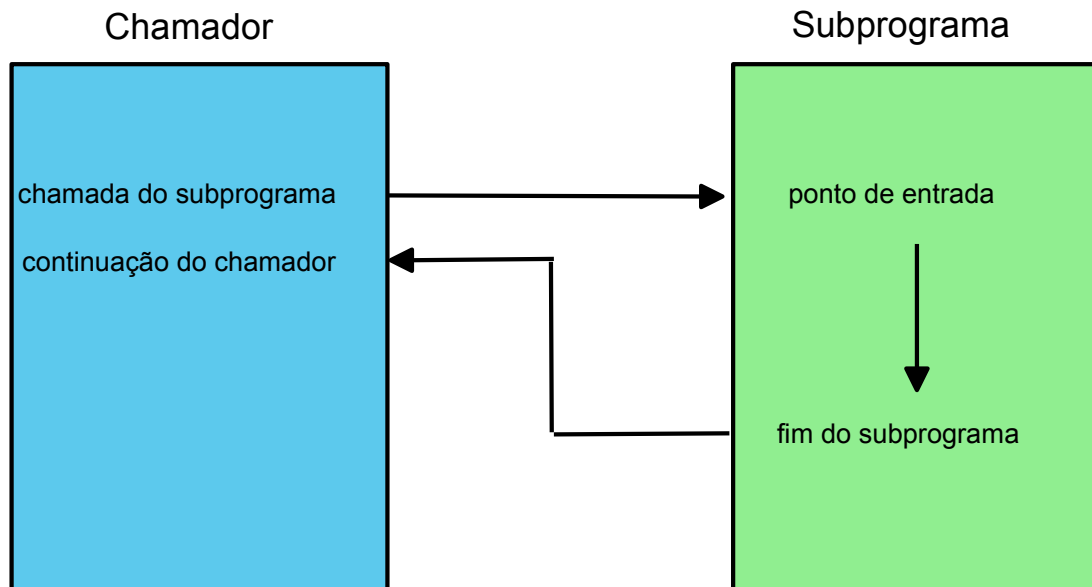
# Subprogramas

- Um subprograma é um conjunto de instruções desenhadas para cumprir uma tarefa particular;
- Subprogramas dividem grandes tarefas de computação em tarefas menores;
- Permitem que os outros programadores os utilizem em seu programas;
- Evita que o programador tenha que escrever o mesmo código repetidas vezes;
  - O conceito economiza espaço e esforço de desenvolvimento e codificação

# Subprogramas

## Características

- Cada subprograma tem um único ponto de entrada;
- A unidade de programa chamadora é suspensa durante a execução do subprograma chamado
- O controle sempre retorna para o chamador quando a execução do subprograma termina



# Subprogramas

## Tipos

- Procedimentos:
  - Conjunto de instruções parametrizadas que definem uma determinada computação;
  - Não retornam valores.
- Funções:
  - Similar aos procedimentos, porém geralmente modelam funções matemáticas;
  - Retorna um valor para o chamador

# Subprogramas

- Tanto em C, como em Fortran, um código executável é criado a partir de um e somente um programa principal, o qual pode invocar subprogramas.
- Em C e Fortran, já utilizamos várias funções:
  - Em C, foram usadas funções das bibliotecas-padrão: `printf`, `scanf`, `strcmp`, `pow`, ...
  - Em Fortran, foram usadas funções intrínsecas: `write`, `read`, `matmul`, ...

# Subprogramas

## Definição

- Subprograma:
  - Cabeçalho → tipo + nome + argumentos formais  
+
  - Corpo → descrição das ações de computação
- Exemplo em C:

The diagram illustrates the components of a C subprogram definition. Three arrows point from the labels 'tipo', 'nome', and 'argumento' to the corresponding parts of the function signature 'float celsius(float fahr)'. A fourth arrow points from the label 'corpo' to the function body enclosed in curly braces.

```
float celsius(float fahr)
{
    float c;
    c = (fahr - 32.0) * 5.0/9.0;
    return c;
}
```



# Subprogramas

## Linguagem C

- A Linguagem C não faz distinção entre procedimentos e funções:
  - O utiliza apenas funções;
  - Uma função tipo `void` não tem retorno.

# Função simples em C

```
/* Celsius.C -- Mostra a escrita da função celsius() */
#include <stdio.h>
#include <stdlib.h>
float celsius(float); /* Protótipo ou declaração da função */
int main() {
    float c, f;
    printf("Digite a temperatura em graus Fahrenheit: ");
    scanf("%f", &f);
    c = celsius(f);          /* Chamada da função */
    printf("Celsius = %.2f\n", c);
    printf("Pressione [ENTER] para finalizar.");
    getchar(); getchar();
    return 0;
}
/* Definição da função */
float celsius(float fahr) {
    float c;
    c = (fahr - 32.0) * 5.0/9.0;
    return c;
}
```

# Função simples

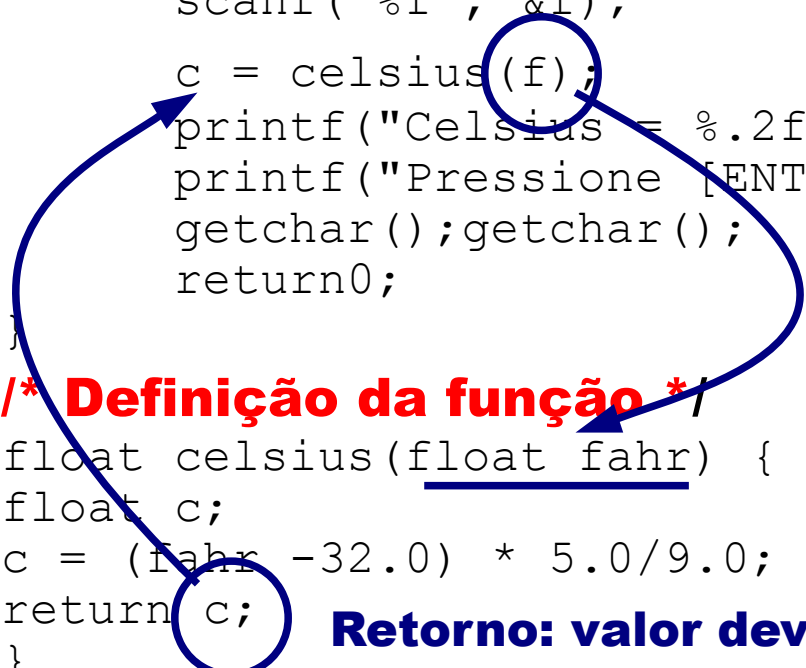
```
/* Celsius.C -- Mostra a escrita da função celsius() */
#include <stdio.h>
#include <stdlib.h>

float celsius(float); /* Protótipo ou declaração da função */

int main() {
    float c, f;
    printf("Digite a temperatura em graus Fahrenheit: ");
    scanf("%f", &f);

    c = celsius(f); /* Chamada da função */
    printf("Celsius = %.2f\n", c);
    printf("Pressione [ENTER] para finalizar.");
    getchar();getchar();
    return0;
}

/* Definição da função */
float celsius(float fahr) {
    float c;
    c = (fahr -32.0) * 5.0/9.0;
    return c; Retorno: valor devolvido pela a função
}
```



# Funções em C

- Definição

```
tipo_retorno    nome_funcao(lista de argumentos)
{
    declaracao de variaveis;
    comandos;
    return(dado_a_retornar);
}
```

- Dessa forma, em C, só há um retorno
  - É possível ter mais de um, se usarmos ponteiros (veremos adiante)

# Funções em C

## O protótipo de uma função

- Uma função não pode ser chamada sem antes ter sido declarada, **assim se a função main( ) usar funções definidas depois dela**, é necessário definir um protótipo
- A declaração de uma função é dita protótipo da função
  - Instrução geralmente alocada no início do programa
  - Estabelece o tipo da função e os argumentos que ela recebe
- O **protótipo da função** permite que o compilador verifique a sintaxe de chamada à função

**float celsius(float);**

- Essa declaração informa que a função de nome celsius() é do tipo float e recebe como argumento um valor float

# Funções sem argumentos em C

- Função pode ter qualquer número de argumentos, inclusive zero

```
ret=imprime_ajuda(); //chamada da função
```

- No caso de zero argumentos, usa-se a palavra-chave `void` para indicar uma lista de argumentos vazia

```
int imprime_ajuda(void); //protótipo
```

# Funções sem retorno em C

- `void` também é usado para indicar que a função não tem retorno
- `void` é um dos tipos básicos em C
  - Indica tamanho zero

```
void imprime_resposta(int resposta)
{
    if (resposta == 0)
    {
        printf("Resposta inválida\n");
        return;
    }
    printf("A resposta é %d\n", resposta);
}
```

# Funções em C

## Exercícios

- Fazer uma função para cálculo da área do círculo e montar um programa exemplo
- Fazer uma função para cálculo da resistência equivalente da associação de 2 resistores (série ou paralelo) e montar um programa exemplo.
  - A função deve receber os valores dos resistores mais um caractere que será s para série e p para paralelo



# Funções em C

## Exercícios

- Fazer uma função para calcular a média de 3 valores
  - Montar um programa exemplo para pedir ao usuário os valores e depois apresentar o resultado.
- Fazer uma função que, chamado a partir da rotina principal, imprima uma mensagem de erro. Montar um programa exemplo.

# Funções em C

## Exercícios

- Fazer uma função que, chamado a partir da rotina principal, imprima quantos dias tem um mês lido no programa principal em função de um valor recebido. Montar um programa exemplo.
  - Por exemplo se receber mês = 1, a função imprime “o mês tem 31 dias”.

# Funções em C

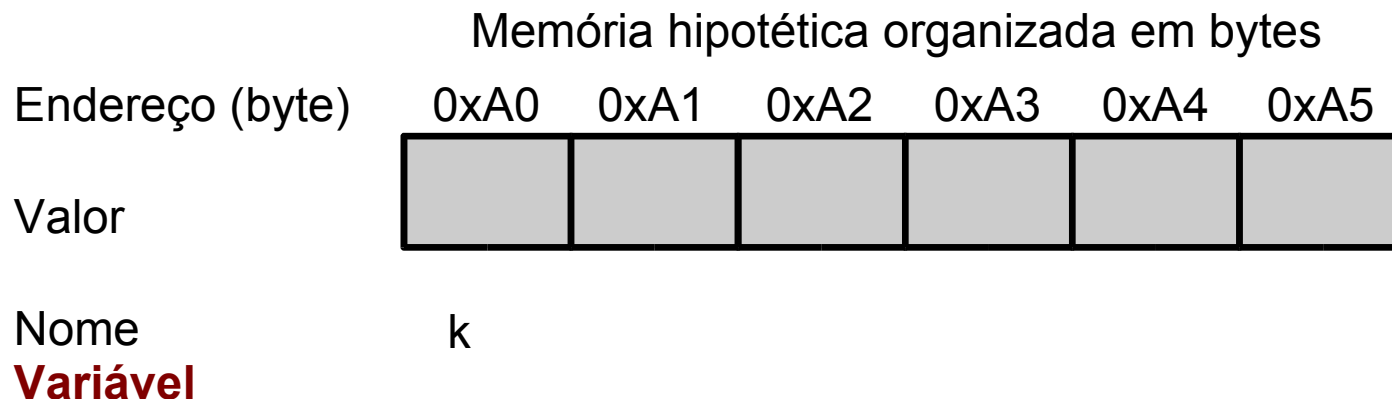
- As funções vistas até agora podiam retornar apenas um valor.
- Utilizando ponteiros, é possível superar esta limitação.
- Assim, antes de apresentarmos como o Scilab e o Fortran tratam os subprogramas, estudaremos os ponteiros e discutiremos seu uso em funções no C.

# Ponteiros

- Um tipo ponteiro é aquele em que as variáveis têm uma faixa de valores que consistem em **endereços de memória**.
- Conceito importantíssimo em C, introduzido no Fortran a partir da versão 90:
  - Extremamente flexíveis, mas devem ser usados com muito cuidado;
  - Podem apontar para qualquer variável, independentemente de onde ela estiver alocada
  - Usado na alocação dinâmica de dados e para emular mais de um retorno em funções em C.

# Ponteiros

- Recapitulando variáveis
  - Variável é um espaço em memória com um nome específico e com valor que pode mudar
  - Tamanho do espaço depende do tipo da variável



# Recapitulando variáveis

- Quando se declara em C

```
char k;
```

- 1 byte (8 bits) de memória é reservado (para guardar um valor inteiro)
- Uma tabela de símbolos mapeia o endereço reservado para o identificador k

# Recapitulando variáveis

- Quando, no programa, define-se

$k = 2;$

- O valor 2 é colocado na porção de memória reservada para  $k$

Endereço (byte)	0xA0	0xA1	0xA2	0xA3	0xA4	0xA5
Valor	00000010					
Nome Variável	k					

# Recapitulando variáveis

- Observe que ao elemento  $k$  estão associadas duas informações
  - O próprio inteiro que está armazenado (2, p.ex.)
  - O “valor” da localização de memória, ou seja, o endereço de  $k$
- Há situações em que o que se deseja armazenar é um endereço
- **Ponteiro** é uma variável que armazena um endereço de memória, ou seja, aponta para um endereço





# Ponteiros

- Um **ponteiro** é uma variável que contém um endereço de memória
  - Esse endereço é normalmente a posição de uma outra variável na memória
- Se uma variável contém o endereço de uma outra, então a primeira variável **aponta** para a segunda
  - Por isso o nome ponteiro

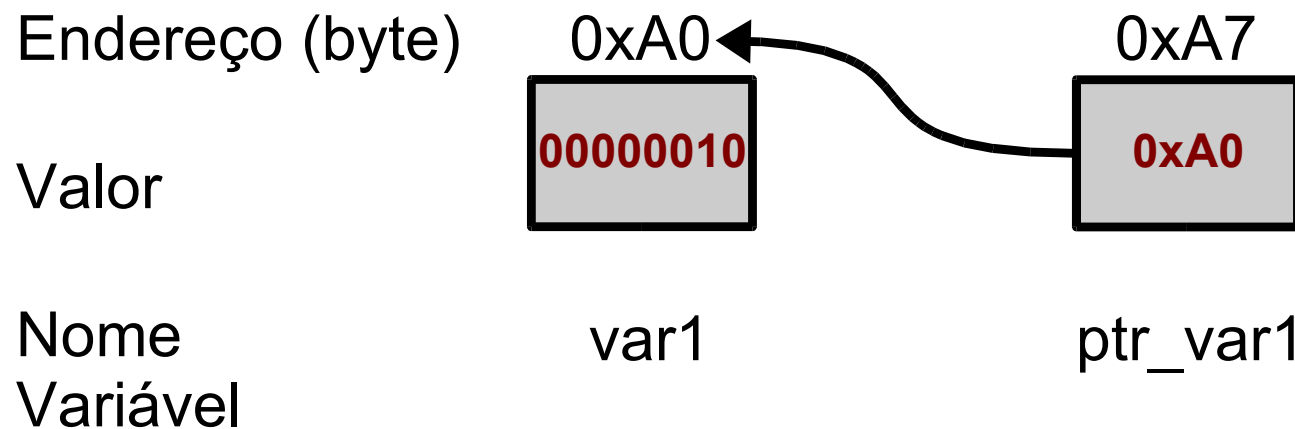
# Ponteiros

- Na figura, a caixa representa um espaço em memória
  - `var1` é o nome de um desses espaços
  - “2” é o valor que está armazenado nesse espaço
  - “0xA0” é o endereço desse espaço na memória

Endereço (byte)	0xA0	0xA7
Valor		
Nome Variável	<code>var1</code>	<code>ptr_var1</code>

# Ponteiros

- `ptr_var1` é uma outra variável
- `ptr_var1` armazena o valor (endereço) `0xA0`
- Neste exemplo, é o endereço da variável `var1`

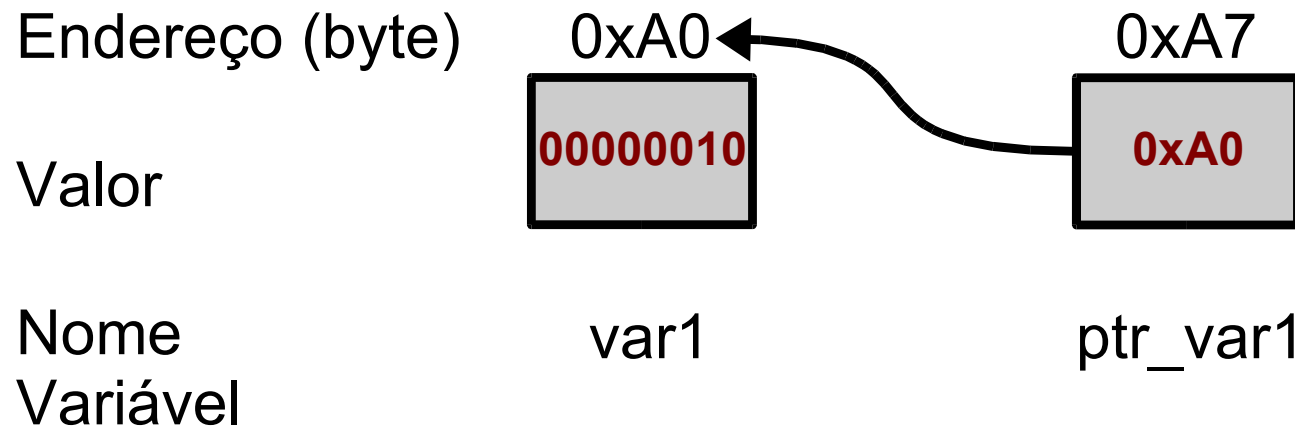


# Ponteiros

- Diz-se que `ptr_var1` aponta para `var1`

`ptr_var1`  $\rightarrow$  `var1`

- Ponteiro também é chamado de variável de endereço



# Ponteiros em C

## Declaração

- Colocar um asterisco na frente do nome da variável

```
int var1; // declara uma variável do tipo int
```

```
int *ptr_num; //declara um ponteiro do tipo int
```

- Cuidado com declaração de mais de um ponteiro na mesma linha

```
int  p,  q,  r; // três variáveis comuns
```

```
int *p,  q,  r; // cuidado! só p será um ponteiro!
```

```
int *p, *q, *r; // agora temos três ponteiros
```

# Ponteiros em C

## Operadores

Operador	Significado
*	Dereference (dado um ponteiro, obtém o elemento referenciado)
&	Address_of (dado um elemento, aponta para o mesmo)

# Ponteiros em C

## Operador & e \*

- O operador **&** pode ser imaginado como “**o endereço de**”, assim como o comando abaixo significa “p recebe o endereço de count”:

- `p = &count ;`

- O operador **\*** é o complemento de **&**. É um operador unário que devolve o valor da variável localizada no endereço que o segue

- `q = *p;`

- O operador **\*** pode ser imaginado como “**no endereço apontado por**”, assim o comando acima significa “q recebe o valor no endereço apontado por p”

# Ponteiros em C

## Operadores

```
int var1; // declara variável do tipo int com o nome var1  
var1 = 4;
```

```
int *ptr_var1; //declara um ponteiro do tipo int
```

```
ptr_var1 = &var1; //ponteiro aponta para var1, ou seja,  
armazena o endereço de var1
```

```
*ptr_var1 = 5; //altera o valor que está armazenado no  
endereço apontado pelo ponteiro para 5
```

Código C	Descrição
<b>var1</b>	Variável simples
<b>&amp;var1</b>	Ponteiro para a variável var1 (endereço de var1)
<b>ptr_var1</b>	Ponteiro para um inteiro (neste exemplo, aponta para a variável var1)
<b>*ptr_var1</b>	Um inteiro



# Exercício 1

- Determine o que ocorre em cada linha do trecho de programa em C
  - Qual o valor final de c e d?

```
int *a, *b, c = 3, d = 7;  
a = &c;  
b = &d;  
*a = 5;  
*b = 8;  
*a = *b;  
*a = 2;  
b = a;  
*b = 0;
```

# Resposta do Exercício 1

```
int *a, *b, c = 3, d = 7;
a = &c;    // a apontará para c
b = &d;    // b apontará para d
*a = 5;    // o valor em c é alterado (c=5)
*b = 8;    // o valor na variável d é alterado (d=8)
*a = *b;   // o valor de d (apontado por b) é
           // copiado para c (apontado por a) (c=8)
*a = 2;    // o valor na variável c é alterado (c=2)
b = a;     // b aponta para o mesmo lugar que a,
           // ou seja, para c
*b = 0;    // o valor de c é alterado (c=0)
```

# Exercício 2

- Determine o que ocorre em cada linha do trecho do programa em C
  - Qual o valor final de a e d?



```
int a=5, *b, *c, d=8;  
b = &a;  
a = 7;  
c = &a;  
*c = 9;  
b = &d;  
*b = *c;  
*c = *b + *b;  
d = a;  
*b = 1;
```

# Ponteiros em C

## Inicialização

- Ponteiros devem ser inicializados antes de serem usados, ou seja, têm que apontar para um endereço específico antes do uso
  - Fazer o seguinte levará a uma falha de segmentação

```
int *p; /*ponteiro não inicializado*/  
  
*p = 9;  
  
/*o endereço físico para guardar o  
número 9 pode não ser válido*/
```

# Ponteiros no Scilab

- O Scilab permite que rotinas ou funções escritas em FORTRAN ou C sejam utilizados dentro de seu ambiente:
  - Através do comando link, em um processo chamado de ligação dinâmica;
  - Através de programas de interface ou *gateways*;
  - Através da adição de uma nova função ao código do Scilab.
- Neste caso, poderá haver a necessidade do Scilab manipular ponteiros

# Ponteiros em Fortran

- O Primeiro passo para usar ponteiros em Fortran é determinar as variáveis as quais será necessário associar ponteiros

- Deve-se usar o atributo target na declaração:

```
real, target :: x, y
```

```
real, dimension(30), target :: a
```

```
real, dimension(8,8), target :: b
```

# Ponteiros em Fortran

- A declaração de ponteiros anexa o atributo `pointer`:
  - `real, pointer :: ptx, pty, pta(:), ptb(:, :)`
- Observe que o tipo e a dimensão do ponteiro devem coincidir com o tipo da variável alvo.
- Em qualquer parte do código é permitido associar o ponteiro com a variável alvo, como na declaração:
  - `px => x`

# Ponteiros em Fortran

- Após a declaração, pode se usar uma atribuição tal como
  - `a(i) = px*b(i,i)`
- Que equivale a
  - `a(i) = x*b(i,i)`
- É permitido, também
  - `px = 3.141592`  
`print *, 'x = ', x`
- Que imprimirá no console o resultado 3.141592, porque alterar `px` altera `x`



# Ponteiros em Fortran

- Um ponteiro pode ser reassociado a qualquer outra variável:
  - `px => y`
- ou, desassociado de qualquer variável:
  - `nullify (px)`

# Ponteiros em Fortran

- É possível descobrir se um ponteiro está associado a alguma variável:
  - `if (associated(px)) then`  
    `print *, 'px está associado '`  
    `end if`
- Ou a uma variável específica:
  - `if (associated(px, target=x)) then`  
    `print *, 'px está associado a x'`  
    `end if`

# Ponteiros em Fortran

- Os ponteiros podem ser associados a arrays ou porções destas:
  - `pa => a`
- Nesse caso, `pa` é um vetor de 30 elementos. Na atribuição:
  - `pa => a(21:30)`
- Nesse caso, `pa` é um vetor de 10 elementos, com `pa(1)` equivalendo `a(21)`. E se
  - `pb => b(2:4, 3:5)`
- Resulta em `pb` com dimensão 3 x 3, com `pb(1,1)` equivalendo a `b(2,3)`

# Resumo até aqui

- Até aqui, vimos o que são ponteiros e como operá-los
- Vamos ver, agora, uma aplicação importante em C
  - Funções com mais de um retorno através do uso de ponteiros

# Ponteiros como argumentos de funções em C

- Em C, parâmetros são passados para uma função através de uma chamada por valor
- Valor do argumento é copiado para dentro da função

```
#include <stdio.h>
mult(int,int);
void main(void)
{  int a=4, b=5;
   printf(" O valor da multiplicação de %d por %d é %d\n", a, b, mult(a,b));
}

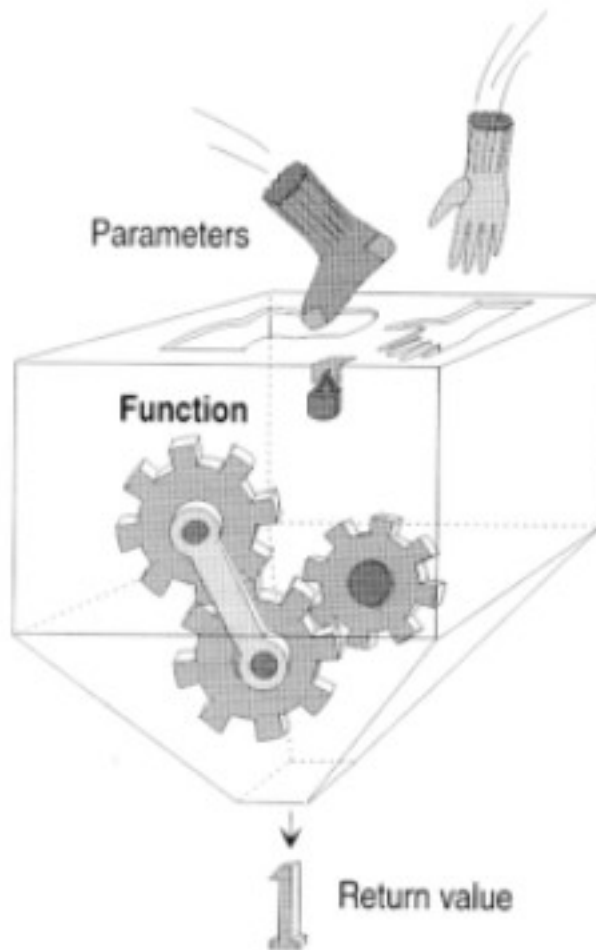
mult(int x, int y)
{  int resultado;
   resultado = x * y;
   return resultado;
}
```

argumentos

parâmetros formais

# Ponteiros como argumentos de funções em C

- Na chamada por valor o caminho só tem uma direção



# Ponteiros como argumentos de funções em C

- Mas usando ponteiros, é possível alterar o próprio parâmetro de entrada
- Assim, função poderá emular a existência de mais de um retorno
- Passagem por referência
- Vamos ver um exemplo prático

# Ponteiros como argumentos de funções em C

- Programa para trocar o valor de duas variáveis

```
#include <stdio.h>
int main()
{
    int a = 5;
    int b = 10;
    int temp;
    printf ("%d %d\n", a, b);

    temp = a;
    a = b;
    b = temp;

    printf ("%d %d\n", a, b);
    return 0;
}
```



# Ponteiros como argumentos de funções em C

- E se essa operação tiver que ser repetida várias vezes? Como colocá-la em uma função?
- Verifique se o código seguinte faria isso.

```
#include <stdio.h>
void swap(int i, int j)
{
    int temp;
    temp = i;
    i = j;
    j = temp;
}
int main()
{
    int a, b;
    a = 5;
    b = 10;
    printf ("%d %d\n", a, b);
    swap (a, b);
    printf ("%d %d\n", a, b);
    return 0;
}
```

# Ponteiros como argumentos de funções em C

- Como fazer usando ponteiros

```
#include <stdio.h>
void swap (int *i, int *j) {
    int temp;
    temp = *i;
    *i = *j;
    *j = temp;
}
int main () {
    int a, b;
    a = 5;
    b = 10;
    printf ("\n\nValem a=%d, b=%d\n", a, b);
    swap (&a, &b);
    printf ("\n\nAgora, valem a=%d, b=%d\n", a, b);
    return 0;
}
```


# Ponteiros como argumentos de funções em C

- Outro exemplo

```
//contador é incrementado dentro da função
#include <stdio.h>
void inc_contador(int *conta_ptr)
{
    (*conta_ptr)++;
}

int main()
{
    int    conta = 0;
    while (conta < 10)
        inc_contador(&conta);
    printf("conta = %d\n", conta);
    return (0);
}
```

# Exercício

- Fazer **uma** função que receba o raio de um círculo e devolva o comprimento da circunferência e a área 
  - Não use variáveis globais
  - Use ponteiros
- main() pergunta o raio para o usuário, chama a função e depois imprime o resultado

# Ponteiros como argumentos de funções em C

- Além de permitir que mais de um valor retorne, a passagem por referência tem outra vantagem
  - Não há cópia dos valores para dentro da função
  - Mais rápido e eficiente
    - Principalmente para manipulação de vetores e estruturas
  - Observe o exemplo seguinte

# Ponteiros como argumentos de funções

```
#include <stdio.h>

void troca(char* string_ptr) {
    //frase não é copiada para dentro da função. É
    passado um ponteiro (endereço do início da string).
    *string_ptr='0'; //altera a primeira letra
    string_ptr++;
    *string_ptr='1'; //altera a segunda letra
}

int main () {
    char frase[150]="Esta é uma string grande...";
    printf ("\n\nfrase=%s\n", frase);
    troca(frase);
    printf ("\n\nAgora, frase=%s\n", frase);
    return 0;
}
```

# Aritmética de Ponteiros em C

- A definição do tipo em ponteiros serve para
  - que o compilador saiba quantos bytes copiar para uma posição de memória

```
int *ptr, var1; //plataforma com int 32  
ptr = &var1;  
*ptr = 2;
```

- Neste exemplo, indica que 32 bits representando o número 2 serão copiados para a área de memória apontada
- Além disso, também serve para fazer operações aritméticas com ponteiros



# Aritmética de Ponteiros em C

- Vamos imaginar que ptr do exemplo anterior aponta para o endereço (em bytes) 100 (decimal)
- Para qual endereço aponta (ptr+1)?

# Aritmética de Ponteiros em C

- Se ptr é do tipo int (4 bytes), o ponteiro irá apontar para o próximo endereço com um inteiro
- Neste caso hipotético, para o endereço 104

# Aritmética de Ponteiros em C

- Os operadores
  - ++, --
  - comparação: >, <, >=, <=, ==, !=
  - São válidos com ponteiros e operam sobre os **endereços**

# Ponteiros e Vetores (Arrays) em C

- Podemos continuar estudando a aritmética de ponteiros a partir de vetores
- Em C, os elementos de um vetor são guardados sequencialmente, a uma “distância” fixa um do outro
- Seja o seguinte (pont\_vet1.c)

```
char array[5] = {5, 10, 15, 20, 25};  
char *array_ptr = &array[0];
```

# Ponteiros e Vetores (Arrays) em C

- Considerando a organização hipotética de memória da figura

```
char array[5] = {5, 10, 15, 20, 25};
```

```
char *array_ptr = &array[0];
```

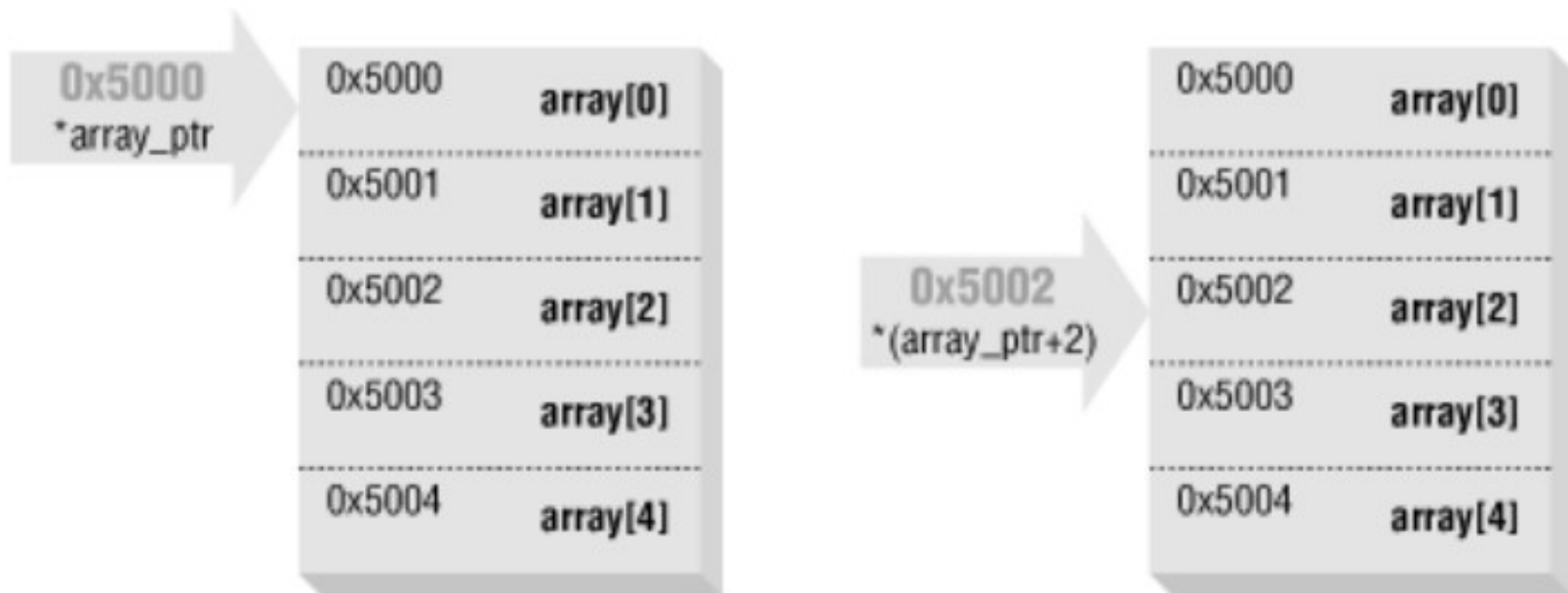
- Verifique o que é obtido com

```
printf("%d %d", *array_ptr, *(array_ptr+1));
```

- E com 

```
printf("%d",(*array_ptr)+1);
```

 ?



# Ponteiros e Vetores (Arrays) em C

- Programa Exemplo
  - Definir vetor
  - Usar ponteiro para acessar cada valor
  - (No próximo exemplo, será vista a equivalência com vetor)

# Ponteiros e Vetores (Arrays) em C

- Verifique a saída do programa (pont\_vetor2.c)

```
#include <stdio.h>
int main ()
{
    int i;
    int vetor[3] = {4, 7, 1};
    int *ptr;
    ptr = vetor; // Em C é igual a ptr = &vetor[0]
    printf("Tam. do int nessa plataf.: %d\n\n", sizeof(int));
    printf("End. vetor: %p\n", vetor);
    printf("End. apontado por ptr: %p\n", ptr);
    printf("End. onde está ptr: %p\n", &ptr);
    for (i = 0; i < 3; i++) {
        printf("O end. do índice %d do vetor é %p\n", i, &ptr[i]);
        printf("O valor do índice %d do vetor é %d\n", i, ptr[i]);
    }
    return 0;
}
```

# Ponteiros e Vetores (Arrays) em C

- C trata ponteiros e vetores da mesma forma
- Assim, são equivalentes (programa anterior)

```
vetor[i]; //”acesso” padrão vetor  
ptr[i];
```

```
*(vetor + i); //”acesso” padrão ponteiro  
*(ptr + i);
```

- Teste cada um deles e confira a equivalência!

```
printf("O valor do índice %d do vetor é %d\n", i, vetor[i]);  
...  
printf("O valor do índice %d do vetor é %d\n", i, *(ptr+i));
```



# Ponteiros e Vetores (Arrays) em C

- O programa seguinte verifica quantos elementos estão no vetor antes que apareça um 0
- Não usa ponteiros de forma explícita

```
#include <stdio.h>

int array[] = {4, 5, 8, 9, 8, 1, 0, 1, 9, 3};
int indice;

int main()
{
    indice = 0;
    while (array[indice] != 0) {
        ++indice;
    }
    printf("Numero de elementos antes de zero %d\n", indice);
    return (0);
}
```

# Ponteiros e Vetores (Arrays) em C

- E este é o equivalente, usando ponteiros
- Verifique como funciona

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

```
int array[ ] = {4, 5, 8, 9, 8, 1, 0, 1, 9, 3};  
int *array_ptr;
```

```
int main()  
{  
    array_ptr = array;  
    while ((*array_ptr) != 0)  
        ++array_ptr;  
    printf("Numero de elementos antes do zero %d\n",  
array_ptr - array);  
    return (0);  
}
```

# Bibliografia e Crédito das Figuras

- OUALLINE, S. *Practical C Programming*. 3. ed. O'Reilly, 1997.
- SEBESTA, R. **Conceitos de Linguagens de Programação**. 5a ed. Porto Alegre: Bookman, 2003.
- CHIVERS, I. e SLEIGHTHOLME, J. **Introduction to Programming with Fortran**. Londres: Springer-Verlag, 2006.

# Bibliografia e Crédito das Figuras

- [http://help.scilab.org/docs/5.5.0/pt\\_BR/index.html](http://help.scilab.org/docs/5.5.0/pt_BR/index.html)
- <http://home.netcom.com/~tjensen/ptr/pointers.htm>
- <http://br.geocities.com/cesarakg/pointers.html>
- [http://pt.wikibooks.org/wiki/Programar\\_em\\_C/Ponteiros](http://pt.wikibooks.org/wiki/Programar_em_C/Ponteiros)