

Introdução à Ciência da Computação II



Aula – ponteiros e vetores em C

Alneu de Andrade Lopes

Ponteiros

- # um ponteiro é uma variável que contém o **endereço** um dado
- # declaração: "*" indica que a variável é um ponteiro

*tipo_dado *nome_ponteiro;*

Ex:

```
int x;
```

```
int *pi;    /* compilador sabe que pi é ponteiro */  
            /* pi é um ponteiro para inteiro */
```

Ponteiros

o operador "&" quando aplicado sobre uma variável retorna o seu endereço

Ex:

```
int x = 10, *pi;
```

```
pi = &x;
```

```
printf("&x: %p pi: %p", &x, pi);
```

=> &x: 0x03062fd8 pi: 0x03062fd8

Ponteiros

o operador "*" quando aplicado sobre um ponteiro retorna o dado apontado

Ex:

```
void main () {
```

```
int *tmp_ptr;
```

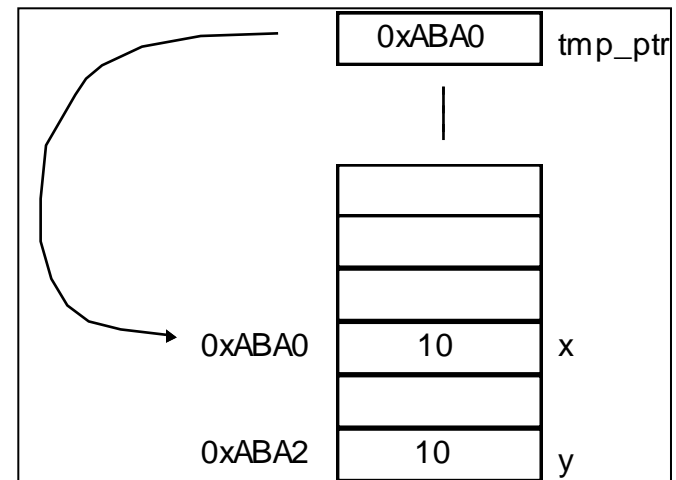
```
int x, y;
```

```
    x = 10;
```

```
    tmp_ptr = &x;
```

```
    y = *tmp_ptr; /* (*tmp_ptr) = 10 */
```

```
}
```



Ponteiros

ponteiros são variáveis tipadas:

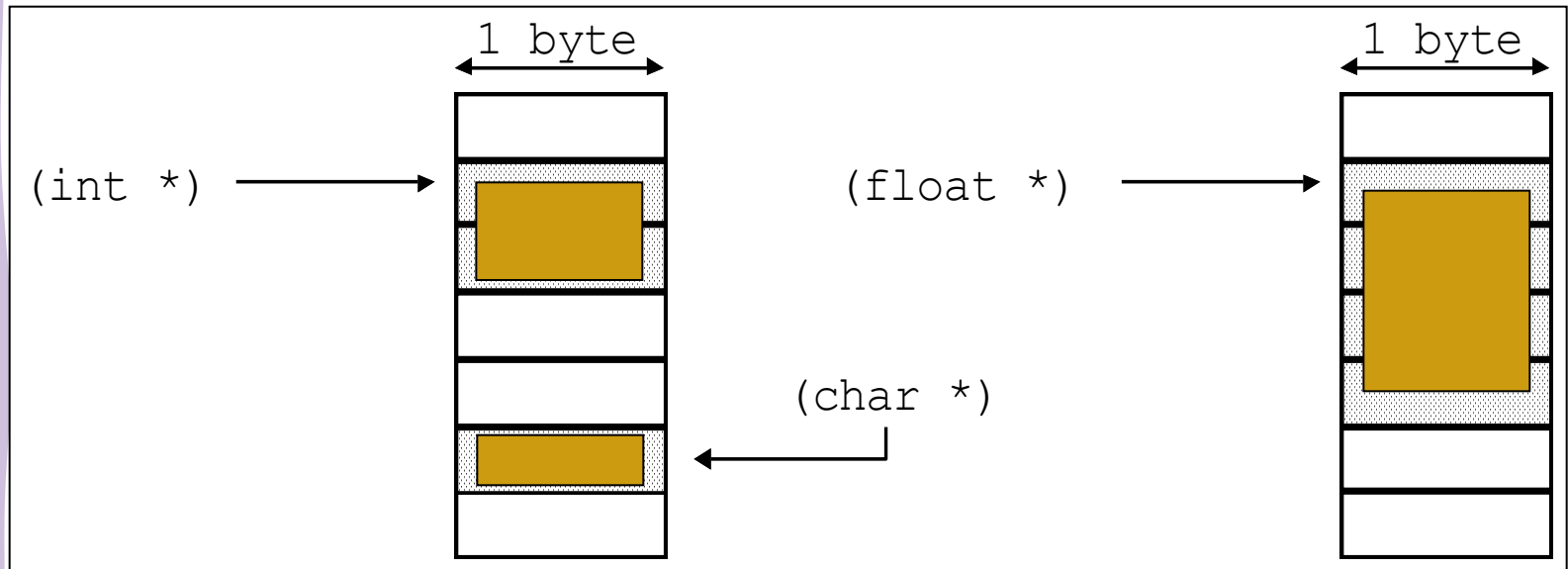
$(\text{int } *) \neq (\text{float } *) \neq (\text{char } *)$

Ex:

```
main() {  
    int *ip, x;  
    float *fp, z;  
    ip = &x; /* OK */  
    fp = &z; /* OK */  
    ip = &z; /* erro */  
    fp = &x; /* erro */  
}
```

Ponteiros

espaço ocupado pelas variáveis



Utilizando Ponteiros

```
void main() {  
    int x = 10;  
    int *pi;  
  
    pi = &x;    /* *pi == 10 */  
    (*pi)++;    /* *pi == 11 */  
    printf("%d", x);  
}
```

==> 11

ao alterar *pi estamos alterando o conteúdo de x

Utilizando Ponteiros

```
void main() {  
    int x = 10;  
    int *pi, *pj;  
  
    pi = &x;    /* *pi == 10 */  
    pj = pi;    /* *pj == 10 */  
    (*pi)++;    /* (*pi, *pj, x) == 11 */  
    (*pj)++;    /* (*pi, *pj, x) == 12 */  
    printf("%d", x); /* ==> 12 */  
}
```


Prática 1

- # Pratique a declaração e utilização de ponteiros.
 - ▣ defina e inicialize uma variável inteira
 - ▣ defina um ponteiro para inteiro
 - ▣ modifique o valor da variável através do ponteiro
 - ▣ verifique os novos valores da variável usando *printf*

Arrays

- # arrays são agrupamentos de dados adjacentes na memória
- # declaração:

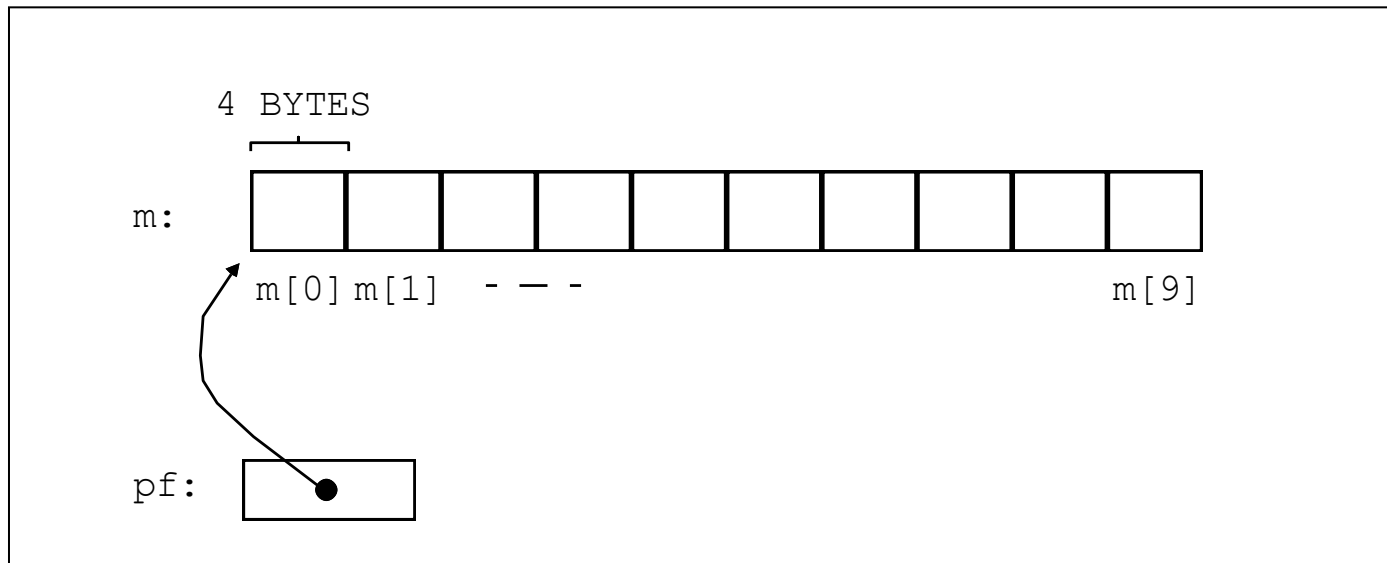
`tipo_dado nome_array[<tamanho>];`

define um arranjo de < *tamanho* > elementos adjacentes na memória do tipo *tipo_dado*

Arrays

Ex:

```
float m[10], *pf;  
pf = m;
```



Referenciando Arrays

- # em `float m[10]` *m* é uma constante que endereça o primeiro elemento do array
- # portanto, não é possível mudar o valor de *m*
- # Ex:

```
float m[10], n[10];
```

```
float *pf;
```

```
m = n;    /* erro: m é constante ! */
```

```
pf = m;   /* ok */
```

Referenciando Elementos

- # pode-se referenciar os elementos do array através do seu nome e colchetes:

```
m[5] = 5.5;
```

```
if (m[5] == 5.5)  
    printf("Exito");
```

```
else  
    printf("Falha");
```

Referenciando Elementos

- # Pode-se referenciar os elementos de um array através de ponteiros:

```
float m[] = { 1.0, 3.0, 5.75, 2.345 };  
float *pf;
```

```
pf = &m[2];  
printf("%f", *pf);      /* ==> 5.75 */
```

Referenciando Elementos

Pode-se utilizar ponteiros e colchetes:

```
float m[] = { 1.0, 3.0, 5.75, 2.345 };
```

```
float *pf;
```

```
pf = &m[2];
```

```
printf("%f", pf[0]);    /* ==> 5.75 */
```

Note que o valor entre colchetes é o deslocamento a ser considerado a partir do endereço de referência

pf[n] ==> indica enésimo elemento a partir de pf

Aritmética de Ponteiros

- # É possível fazer operações aritméticas e relacionais entre ponteiros e inteiros
- # Soma: ao somar-se um inteiro n a um ponteiro, endereçamos n elementos a mais (n positivo) ou a menos (n negativo)

pf[2] equivale a $*(pf+2)$

$*(pf + n)$ endereça n elementos a frente

$*(pf - n)$ endereça n elementos atrás

pf++ endereça próximo elemento array

pf-- endereça elemento anterior array

Exemplo

```
void main ()  
{  
  int arint[] = { 1,2,3,4,5,6,7 };  
  int size = 7; /* tamanho do array */  
  int i, *pi;  
  
  for (pi=arint, i=0; i < size; i++, pi++)  
    printf("\ %d ", *pi);  
}
```

==> 1 2 3 4 5 6 7

Exemplo - variação

```
void main ()  
{  
  int arint[] = { 1,2,3,4,5,6,7 };  
  int size = 7;    /* tamanho do array */  
  int i, *pi;  
  
  for (pi=arint, i=0; i < size; i++)  
    printf("\ %d ", *pi++);  
}
```

==> 1 2 3 4 5 6 7

Exemplo - variação

```
void main () {  
  int arint[] = { 1,2,3,4,5,6,7 };  
  int size = 7; /* tamanho do array */  
  int i, *pi;  
    pi = arint;  
    printf("\ %d ", *pi); pi += 2;  
    printf("\ %d ", *pi); pi += 2;  
    printf("\ %d ", *pi); pi += 2;  
    printf("\ %d ", *pi);  
}
```

==> 1 3 5 7

Operações Válidas Sobre Ponteiros

É válido:

- *somar* ou *subtrair* um inteiro a um ponteiro ($pi \pm int$)
- *incrementar* ou *decrementar* ponteiros ($pi++$, $pi--$)
- *subtrair* ponteiros (produz um inteiro) ($pf - pi$)
- *comparar* ponteiros ($>$, $>=$, $<$, $<=$, $==$)

Não é válido:

- somar ponteiros ~~$(pi + pf)$~~
- multiplicar ou dividir ponteiros ~~$(pi * pf, pi / pf)$~~
- operar ponteiros com *double* ou *float* ~~$(pi \pm 2.0)$~~

Prática 2



- # Escreva um programa que imprima um array de inteiros na ordem inversa endereçando os elementos com um ponteiro

Cuidados...

✦ **C não** controla os limites dos arrays, o programador deve fazê-lo

✦ Ex:

✦ encontrar o erro:

```
void main () {  
int arint[] = { 1,2,3,4,5,6,7 };  
int size = 7, i, *pi;  
for (pi=arint, i=0; i < size; i++, pi += 2)  
    printf(" %d ", *pi);  
}
```

Cuidados...

```
void main ()
```

```
{
```

```
int arint[] = { 1,2,3,4,5,6,7 };
```

```
int size = 10;
```

```
int i;
```

```
for (pi=arint, i=0; i < size; i++)
```

```
    printf(" %d ", arint[i]);
```

```
}
```

Cuidados...

Um ponteiro deve *sempre* apontar para um local válido *antes* de ser utilizado

Ex:

```
void main ()
```

```
{
```

```
  int i=10, *pi;
```

```
  *pi = i;      /*erro ! pi nao tem endereco valido*/
```

```
}
```


Ponteiros Genéricos

- # Um ponteiro genérico é um ponteiro que pode apontar para qualquer tipo de dado
- # Define-se um ponteiro genérico utilizando-se o tipo *void*:

```
void *pv;
```

```
int x=10;
```

```
float f=3.5;
```

```
pv = &x; /* aqui pv aponta para um inteiro */
```

```
pv = &f; /* aqui, para um float */
```

Ponteiros Genéricos

- # O tipo de dado apontado por um *void pointer* deve ser controlado pelo usuário
- # Usando um *type cast* (conversão de tipo) o programa pode tratar adequadamente o ponteiro

```
pv = &x;  
printf("Inteiro: %d\n", type cast*(int *)pv); /*=> 10*/  
pv = &f;  
printf("Real: %f\n", *(float *)pv); /*=> 3.5*/
```

Ponteiros e Strings

- # *strings* são arrays de caracteres e podem ser acessados através de *char **

```
void main ()
```

```
{
```

```
char str[]="abcdef", *pc;
```

```
    for (pc = str; *pc != '\0'; pc++)
```

```
        putchar(*pc);
```

```
}
```

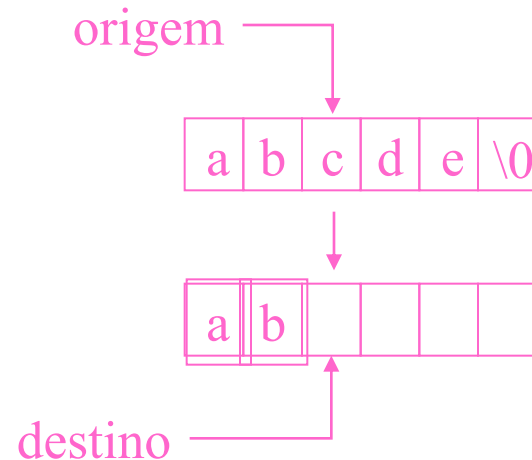
=> abcdef

- # o incremento de pc o posiciona sobre o próximo character (byte a byte)

Ponteiros e Strings

operações sobre *strings* com ponteiros

```
void StrCpy (char *destino, char *origem)
{
    while (*origem) /* *origem=='\0' encerra while */
    {
        *destino=*origem;
        origem++;
        destino++;
    }
    *destino='\0';
}
```



Ponteiros e Strings

variação de *strcpy*:

```
void strcpy (char *destino, char *origem)
{
    while ((*destino = *origem) != '\0')
        destino++, origem++;
}
```

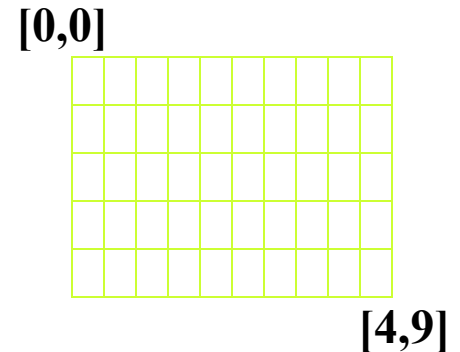
Arrays Multidimensionais

- # Arrays podem ter diversas dimensões, cada uma identificada por um par de colchetes na declaração

Ex:

char matriz[5][10];

- # declara uma matriz de 5 linhas e 10 colunas:
- # na memória, entretanto, os caracteres são armazenados linearmente:



Array de Caracteres

Percorrendo *array* com ponteiro:

```
void main () {  
  char matriz[5][10];  
  char *pc;  
  int i;  
  
  for (i=0, pc=matriz[0]; i < 50; i++, pc++)  
    *pc = ` `;  
}
```

Array de Caracteres

Percorrendo *array* com índices:

```
void main () {  
  char matriz[5][10];  
  int i, j;  
  
    for (i=0, j=0; i<5; i++)  
        for (; j<10; j++)  
            matriz[i][j] = ` `;  
}
```

as colunas (dimensões mais a direita) mudam mais rápido

Array de Inteiros

Exemplo: considere o problema de conversão de data
dia_do_ano: um dos 365 dias do ano, convertido a partir do
mes e dia do mes

Tabela que indica dias dos meses incluindo bissexto

```
static char tabela_dias[2][13] =  
{  
    { 0,31,28,31,30,31,30,31,31,30,31,30,31 }  
    { 0,31,29,31,30,31,30,31,31,30,31,30,31 }  
};
```

Conversão de Data

Organização lógica e física da tabela:

tabela_dias

0	31	28	31	30	31	31	30	31	30	31	30	31
0	31	29	31	30	31	31	30	31	30	31	30	31

memória

0	31	28	31	...	0	31	28	31	...		31	30	31
---	----	----	----	-----	---	----	----	----	-----	--	----	----	----

Conversão de Data

`# /* dia_do_ano: calcula dia do ano a partir
do dia do mes */`

`int dia_do_ano(int ano, int mes, int dia)`

`{`

`int i, bis;`

`bis = (ano%4)==0 && (ano%100)!=0 ||
 (ano%400)==0;`

`for (i = 1; i < mes; i++)`

`dia += tabela_dias[bis][i];`

`return dia;`

`}`

Array de *Strings*

Neste caso, cada elemento do *array* é um ponteiro para um caracter

Declaração:

```
char *arstr[] = {"Joao", "Maria", "Antonio",  
                 "Zacarias", "Carlos"};
```

arstr é um *array* de ponteiros para *char*, iniciado com os *strings* indicados

Array de Strings

Comparando *array* de *string* com matriz de *char*

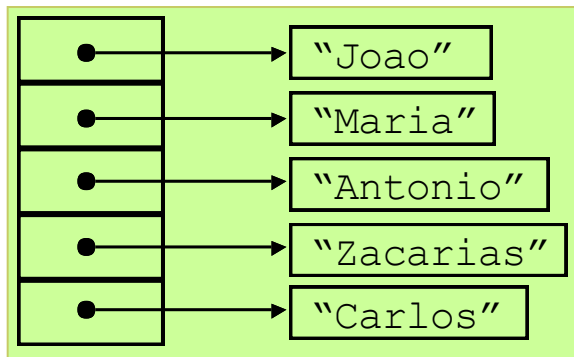
```
char *as[] =
```

```
    {"Joao","Maria","Antonio","Zacarias","Carlos"};
```

```
char ma[5][10] =
```

```
    {"Joao","Maria","Antonio","Zacarias","Carlos"};
```

Ponteiros (as)



Matriz (ma)

J	o	a	o		W0				
M	a	r	i	a		W0			
A	n	t	o	n	i	o		W0	
Z	a	c	a	r	i	a	s		W0
C	a	r	l	o	s		W0		

Cuidados com *Strings*

- # É comum esquecer de alocar uma área para armazenamento de caracteres

```
void main() {
```

```
char *pc; char str[] = "Um string";
```

```
    strcpy(pc, str); /* erro! pc indeterminado */
```

```
    ...  
}
```

Ponteiros para Ponteiros

É possível definir ponteiros para ponteiros até um nível arbitrário de indireção

Ex:

```
char *pc;    /* ponteiro para char */
```

```
char **ppc; /* ponteiro para ponteiro para char */
```

```
pc = "teste";
```

```
ppc = &pc;
```

```
putchar(**ppc); /* ==> 't' */
```

Ponteiros para Ponteiros

Ponteiro para ponteiro para ponteiro...

Ex:

```
char *pc, **ppc, ***pppc;
```

**Um ponteiro permite modificar o objeto apontado
ou apontar para outro objeto do mesmo tipo**

Passagem de Parâmetros

em C os argumentos para uma função são, via de regra, 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.  
}
```

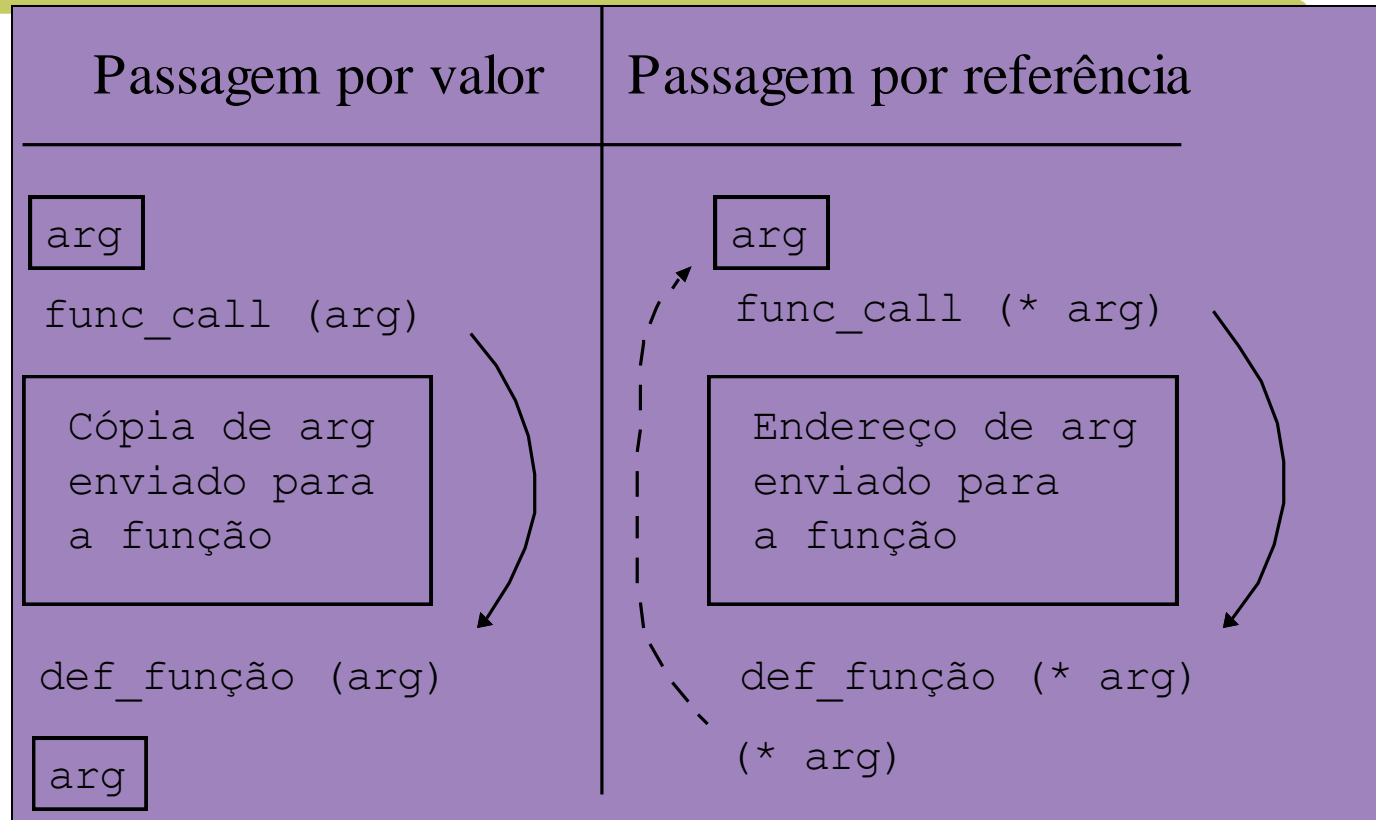
Passagem de Parâmetros

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 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

ex:

```
int x = 10, y = 20;
```

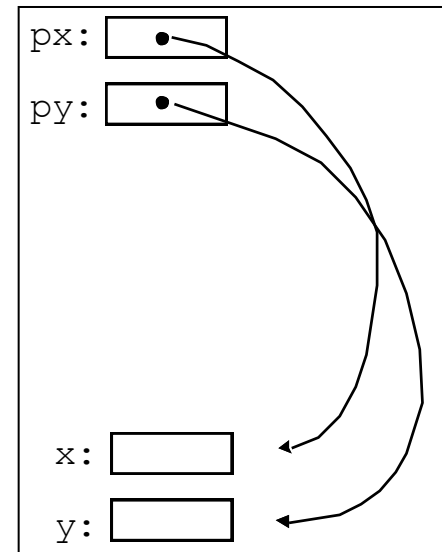
```
troca(&x, &y);
```

```
printf("x=%d y=%d", x, y) => x=20 y=10
```

Prática: função *troca*

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

    temp=*px;
    *px=*py;
    *py=temp;
}
```



Retornando Valores

* uma função retorna um valor através do comando *return*

* Ex:

```
int power (int base, int n) {  
    int i,p;  
  
    p = 1;  
    for (i = 1; i <= n; ++i)  
        p *= base;  
    return p;  
}
```

Funções

- # o valor retornado por uma função é sempre copiado para o contexto de chamada (retorno *by value*)

```
x = power(2, 5);           /* atribuição */
if (power(7, 2) > 12543) /* comparação */
    printf("Numero grande!");
x = 10*power(2,3);         /* expressão */
array[get_index()];        /* índice */
funcao( get_arg() );       /* argumento */
```


Ex: Concatena Strings

```
char *concatena( char cabeca[], char cauda[] )  
{  
    int i, j;  
  
    for (i = 0; cabeca[i] != '\0'; i++);  
    for (j = 0; (cabeca[i] = cauda[j]) != '\0'; i++, j++);  
    cabeca[i] = '\0';  
    return cabeca;  
}
```

Exemplo (cont.)

```
int main( )  
{  
    char nome[80] = "Santos";  
    char sobrenome[] = " Dumont";  
  
    printf( "O nome é %s.\n",  
           concatena( nome, sobrenome ) );  
  
    return 0;  
}
```

Prática: Localiza *char* em *string*

Fazer uma função que procura um caracter em um *string* e retorna o seu endereço caso o encontre, senão retorna NULL (ponteiro nulo)

Ex:

```
char *achachar (char *str, char c) {...}
```

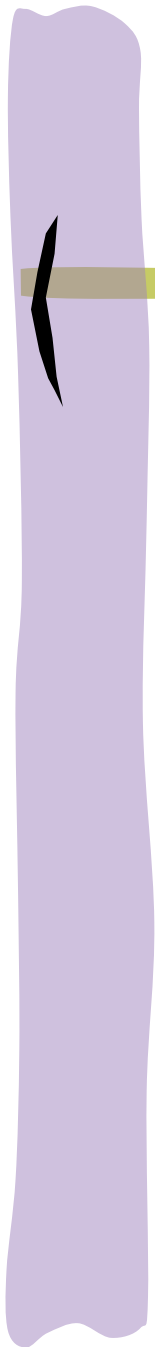
```
char str[] = "abcd5678";
```

```
achachar(str, 'c');
```

=> retorna endereço do terceiro caracter do *string*: &str[2]

Achachar

```
char *achachar (char *str, char c) {  
    char *pc = str;  
  
    while (*pc != c && *pc != '\0') pc++;  
    return *pc ? pc : NULL;  
}
```



Fim