
Universidade de São Paulo – São Carlos
Instituto de Ciências Matemáticas e de Computação

# Mais sobre Ponteiros em C

**Material preparado pela profa**
**Silvana Maria Affonso de Lara**

**2º semestre de 2010**

# Roteiro da Aula

- Aula passada:
  - Definição de ponteiros
  - Como utilizar ponteiros
  - Exemplos de ponteiros
  - Definição de Arrays
  - Como referenciar arrays
  - Como referenciar elementos
  - Operações válidas sobre ponteiros
- Cuidados!!
- Ponteiros genéricos
- Ponteiros e Strings
- Arrays Multidimensionais
- Ponteiros para ponteiros

# 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

type cast

```
pv = &x;
printf("Inteiro: %d\n", *(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);
}
```

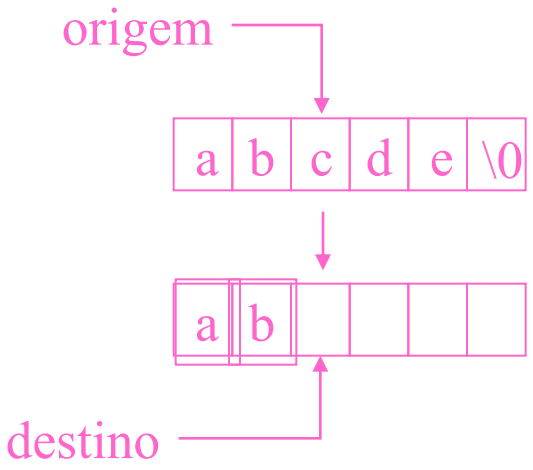
==> abcdef

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

# Ponteiros e Strings

- operações sobre *strings* com ponteiros. Ex.:

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



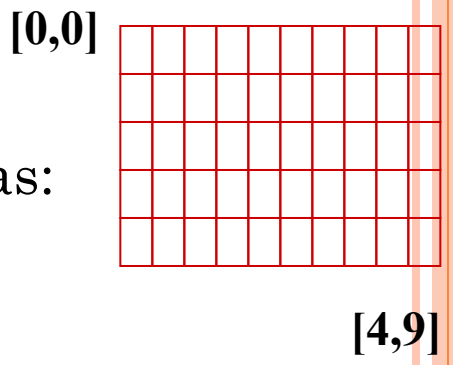
# Ponteiros e Strings

- variação de *strcpy:*

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

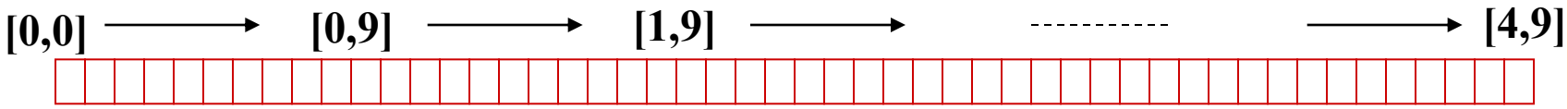
# Arrays Multidimensionais

- *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 = ' ';
} /* inicializa todos os elementos da matriz com
espaço em branco */
```

# Array de Caracteres

Percorrendo *array* com índices:

```
void main () {
char matriz[5][10];
int i, j;

  for (i=0; i<5; i++)
                  for (j=0; j<10; j++)
                              matriz[i][j] = ' ';
}
```

# 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 | 29 | 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 | \0 | | | |
| M | a | r | i | a | \0 | | |
| A | n | t | o | n | i | o | \0 |
| Z | a | c | a | r | i | a | s | \0 |
| C | a | r | l | o | s | \0 | |

# 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**

# Exercício

Considerando as variáveis e ponteiros definidos abaixo; quais são as atribuições permitidas?

```
int    i, *pi, **ppi;
float  f, *pf, **ppf;
```

a) `i = f;`
b) `pf = &i;`
c) `*pf = 5.9;`
d) `*ppi = &pi;`
e) `*pf = 10;`
f) `f = i;`
g) `pi = &f;`
h) `*pi = 7.3;`
i) `ppf = &pf;`
j) `**ppi = 100;`

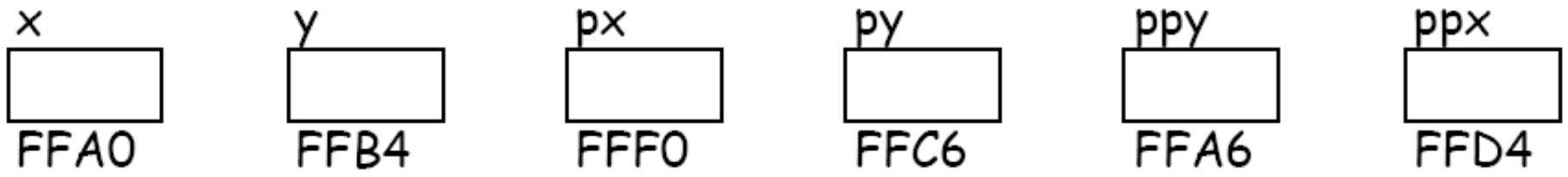
## Resposta

c) `*pf = 5.9;`  e) `*pf = 10;`  i) `ppf = &pf;`  j) `**ppi = 100;`

# Exercício

Dadas as declarações abaixo; qual é o valor dos itens:

```
int  x = 10,  *px = &x,  **ppx = &x;
float  y = 5.9,  *py = &y,  **ppy = &py;
```




a) x =
b) *py =
c) px =
d) &y =
e) *px =
f) y =
g) *ppx =
h) py =
i) &x =
j) py++ =
k) *px-- =
l) **ppy =
m) &ppy =
n) *&px =
o) **ppx++ =
p) px++ =
q) &ppx =

# Resposta

a) x = 10
b) *py = 5.9
c) px = FFA0
d) &y = FFB4
e) *px = 10
f) y = 5.9
g) *ppx = FFA0
h) py = FFB4
i) &x = FFA0
j) py++ = FFB8
k) *px-- = 9
l) **ppy = 5.9
m) &ppy = FFA6
n) *&px = FFA0
o) **ppx++ = 11
p) px++ = FFA2
q) &ppx = FFD4

# Exercício

- Fazer um programa em C que leia uma matriz de dimensão 3 X 5 de caracteres, e imprima-a na ordem inversa, usando ponteiros

# Exercício

- Fazer um programa em C que leia uma matriz de dimensão 3 X 5 de caracteres, e imprima-a na ordem inversa, usando ponteiros

```
void main () {
   char matriz[3][5], *pc;
   int i, j;
      pc=matriz[0];
     for (i=0; i<3; i++)
                        for (j=0; j<5; j++, pc++)
                                                scanf("%c",pc);
     pc=&matriz[2,4];'
      while (pc >= matriz[0])
            {
             printf("  %c",*pc);
            pc--;
            }
   }
```