- Ponteiros como vetores

Sabemos agora que:

- o nome de um vetor é um ponteiro constante;
- podemos indexar o nome de um vetor.

Logo, podemos também indexar um ponteiro qualquer.

O programa mostrado a seguir funciona perfeitamente:

- Ponteiros como vetores (continuação)

```
#include <stdio.h>
main ()
{
  int matrx [10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
  int *p;
  p=matrx;
  printf ("O terceiro elemento do vetor e: %d",p[2]);
}
```

OBS.: Podemos ver que p[2] equivale a *(p+2).

- Ponteiros como vetores (continuação)

Contudo uma observação é necessária:

```
#include <stdio.h>
main ()
{
  int matrx [2][] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
  int *p;
  p=matrx[0];
  printf ("O terceiro elemento da segunda linha da
matriz eh: %d",p[7]);
  printf ("O terceiro elemento da segunda linha da
matriz eh: %d",p[1][2]);
}
```

- Strings

Seguindo o raciocínio anterior, nomes de strings, são do tipo **char***. Isto nos permite explorar os conceitos apresentados para resolver problemas como, por exemplo, o apresentado no exercício a seguir.

Exercício:

Construa um programa que declare um vetor de strings com 10 elementos e o inicialize com nomes fornecidos pelo usuário através da entrada padrão e em seguida o retorne na saída padrão. A manipulação do vetor deve ser feita por meio de um ponteiro.

- Endereços de elementos de vetores

Nesta seção vamos apenas ressaltar que a notação

&nome_da_variável[índice]

no caso de uma declaração do tipo

tipo_elemento nome_da_variável[num_elementos];

é válida e retorna o endereço do ponto do vetor indexado por índice.

- Vetores de ponteiros

Podemos construir vetores de ponteiros como declaramos vetores de qualquer outro tipo. Um exemplo de declaração de um vetor de ponteiros inteiros é:

```
int *pmatrx [10];
```

No caso acima, **pmatrx** é um vetor que armazena 10 ponteiros para inteiros.

- Inicializando Ponteiros

Podemos inicializar, ponteiros de um jeito, no mínimo interessante.

Precisamos, para isto, entender como o C trata as strings constantes.

Toda string que o programador insere no programa é colocada num banco de strings que o compilador cria. No local onde está uma string no programa, o compilador coloca o endereço do início daquela string (que está no banco de strings).

- Inicializando Ponteiros (continuação)

É por isto que podemos usar **strcpy()** do seguinte modo:

```
strcpy (string,"String constante.");
```

strcpy() solicita dois parâmetros do tipo **char***. Como o compilador substitui a string **"String constante."** pelo seu endereço no banco de strings, os argumentos da função **strcpy()** estão coerentes.

- Inicializando Ponteiros (continuação)

O que isto tem a ver com a inicialização de ponteiros? É que, para uma string que vamos usar várias vezes, podemos fazer:

```
char *str1="String constante.";
```

Aí poderíamos, em todo lugar que precisarmos da string, usar a variável **str1**. Devemos apenas tomar cuidado ao usar este ponteiro. Se o alterarmos vamos perder a string. Se o usarmos para alterar a string podemos facilmente corromper o banco de strings que o compilador criou.

- Inicializando Ponteiros (continuação)

OBS.: Em C existem modificadores de acesso, um exemplo é o modificador *const* que permite criar uma constante.

Exemplo:

```
const int numero = 32;
```

Logo, podemos fazer:

```
const char *str1="String constante.";
```

Desta forma o conteúdo apontado pelo ponteiro não pode ser alterado.