

Ponteiros e Vetores

- 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 e Vetores

- Ponteiros como vetores

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

OBS.: Podemos constatar que $p[2]$ é equivalente a $*(p+2)$.

Ponteiros e Vetores

- Ponteiros como vetores

Contudo, uma observação é necessária:

```
#include <stdio.h>
int main () {
    int matr[2][10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    int *p;
    p=&matr[0][0];
    p=matr[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]);
}
```

Ponteiros e Vetores

- 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.

Ponteiros e Vetores

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.

```
#include <stdio.h>
#define tamanho 10
#define comprimento_max 100
int main () {
    char vetor_strings[tamanho][comprimento_max], *p;
    int i;
    for (i=0, p=vetor_strings[0]; i<tamanho; i++)
    {
        printf ("\nEntre com a string[%d]: ", i+1);
        scanf("%99[^\n]", p); /*caso seja necessário lembre-se de limpar o buffer de entrada*/
        p+=comprimento_max;
    }
    printf ("\nString digitadas:");
    for (i=0, p=&vetor_strings[0][0]; i<tamanho; p+=comprimento_max, i++)
        printf ("\nString[%d]: %s", i+1, p);
}
```

Ponteiros

**Strings Constantes – Modificador de
Acesso *const***

Ponteiros

- Inicializando Ponteiros

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

Para isto, precisamos entender como a linguagem de programação C trata as strings constantes.

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

String Constante 1

String Constante 2



`strcmp (char *, char *)`

`strcmp ("EU", "VAMOS")`

Ponteiros

- Inicializando Ponteiros

O que isto tem a ver com a inicialização de ponteiros?

É que, para uma string constante 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. Pois, se o alterarmos vamos perder a referência para a string e se o usarmos para alterar a string podemos facilmente corromper o banco de strings constantes que o compilador criou.

Ponteiros

- Inicializando Ponteiros

OBS.: Em C existem modificadores de acesso, 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 endereço apontado pelo ponteiro não pode ser alterado. Mas, ainda podemos corromper o banco de strings constantes.

Ponteiros

Funções strchr() e strstr() - string.h

Ponteiros

Agora podemos concluir nosso estudo das Funções Básicas para manipulação de Strings.

- strchr

Sua forma geral é:

```
char *strchr (const char *str, int ch);
```

A função *strchr()* devolve um ponteiro para a primeira ocorrência do byte menos significativo de *ch* na string apontada por *str*. Se não for encontrada nenhuma coincidência, será devolvido um ponteiro nulo. (string.h)

Ponteiros

/* Exemplo strchr */

```
#include <string.h>
int main()
{
    char *p;
    p = strchr("Isto eh um teste.", ' ');
    puts(p);
}
```

Ponteiros

- strstr

Sua forma geral é:

*char *strstr (const char *str1, const char *str2);*

A função *strstr()* devolve um ponteiro para a primeira ocorrência da string apontada por *str2* na string apontada por *str1*. Ela devolve um ponteiro nulo se não for encontrada nenhuma coincidência. Obs.: Função presente em *string.h*.

Ponteiros

/* Exemplo strstr */

```
#include <string.h>
int main()
{
    char *p;
    p = strstr("Isto é um teste", "te");
    puts(p);
}
```

Ponteiros

Exercício:

Construa um programa que receba duas strings fornecidas pelo usuário, verifique se a segunda string está presente na primeira e, caso esteja retorne a posição do caractere na primeira string onde a primeira ocorrência da segunda string inicia, caso contrário retorne zero.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main() {
    char string1[100], string2[100], *p;
    puts("Forneca a primeira string: ");
    scanf("%99[^\n]", string1);
    puts("Forneca a segunda string: ");
    setbuf (stdin, NULL);
    scanf("%99[^\n]", string2);
    p = strstr(string1, string2);
    if (p)
        printf("%d\n", (int)(p-string1)+1);
    else
        puts ("0");
}
```

Vetores de Ponteiros e Ponteiros para Ponteiros

Ponteiros

- 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 *pmat [10];
```

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

Ponteiros

- Ponteiros para Ponteiros

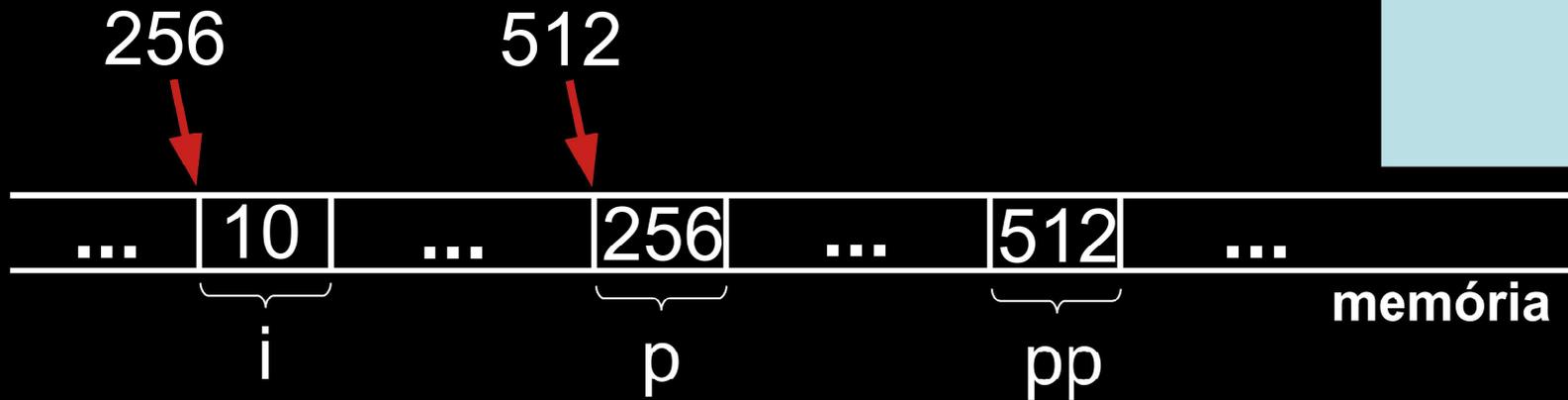
Podemos declarar um ponteiro para um ponteiro com a seguinte notação:

```
tipo_da_variável **nome_da_variável;
```

Algumas considerações:

****nome_da_variável** é o conteúdo final da variável apontada;

***nome_da_variável** é o conteúdo do ponteiro intermediário.



```
int i=10;
```

```
int *p;
```

```
int **pp;
```

```
p=&i;
```

```
pp=&p;
```

```
printf ("%d", **pp);
```

Ponteiros

- Ponteiros para Ponteiros (continuação)

Na linguagem C podemos declarar ponteiros para ponteiros para ponteiros, ou então, ponteiros para ponteiros para ponteiros para ponteiros e assim por diante.

Para fazer isto basta aumentar o número de asteriscos na declaração.

Para acessar o valor desejado apontado por um ponteiro para ponteiro, o operador asterisco deve ser aplicado duas vezes, como mostrado anteriormente e no exemplo a seguir:

Ponteiros

- Ponteiros para Ponteiros (continuação)

```
#include <stdio.h>
int main()
{
    float pi = 3.1415, *pf, **ppf;
    pf = &pi;
    ppf = &pf;
    printf("\n%.4f", **ppf);
    printf("\n%.4f", *pf);
}
```

Ponteiros

Exercício:

Verifique o programa abaixo. Encontre o(s) seu(s) erro(s) e corrija-o(s) para que o mesmo escreva o número 10 na tela.

```
#include <stdio.h>
int main()
{
    int x, *p, **q;
    p = &x;
    q = &p;
    x = 10;
    printf( "\n%d\n" , **q );
}
```