Ponteiros e Vetores

Ponteiros e Vetores

- Vetores como ponteiros

Para que possamos compreender esta similaridade, devemos primeiro entender como a linguagem C trata vetores.

Quando declaramos um vetor da seguinte forma:

tipo_da_variável nome_da_variável [tam1][tam2] ... [tamN];

Ponteiros e Vetores

- Vetores como ponteiros

O compilador C calcula o tamanho, em bytes, necessário para armazenar este vetor. Como vimos, este tamanho é:

tam1 x tam2 x tam3 x ... x tamN x tamanho_do_tipo

O compilador então aloca este número de bytes em um espaço livre de memória. O *nome da variável* declarada é na verdade um ponteiro para o *tipo dos elementos do vetor* que aponta para o endereço inicial da área alocada.

UNIVASF

Ponteiros e Vetores

- Vetores como ponteiros

Mas, aí surge a pergunta: então como é que podemos usar a seguinte notação?

nome_da_variável[índice]

Isto pode ser facilmente explicado desde que você entenda que a notação acima é *absolutamente equivalente* a se fazer:

**(nome_da_variável+índice)*

UNIVASF

Ponteiros e Vetores

- Vetores como ponteiros

Dessa forma, um ponteiro pode ser utilizado, por exemplo, para fazer uma varredura sequencial de uma matriz. Pois, quando temos que varrer todos os elementos de uma matriz de uma forma sequencial, podemos usar um ponteiro, o qual vamos incrementando.

Qual seria a vantagem em se fazer uma varredura sequencial usando ponteiros?

UNIVASF

Ponteiros e Vetores

- Vetores como ponteiros

Considere o seguinte programa para zerar uma matriz:

```
int main ()
{
    float matrx [50][50];
    int i,j;
    for (i=0;i<50;i++)
        for (j=0;j<50;j++)
            matrx[i][j]=0.0;
}
```

Ponteiros e Vetores

- Vetores como ponteiros

Podemos reescrevê-lo usando ponteiros:

```
int main ()
{
   float matrx [50][50], *p;
   int count;
   for (count=0, p=matrx[0]/*p=&matrx[0][0]*/; count<2500;count++)
      *(p++)=0.0;
}
```

Ponteiros e Vetores

Qual seria a vantagem em se fazer uma varredura sequencial usando ponteiros?

```
for (i=0;i<50;i++)
    for (j=0;j<50;j++)
        matrx[i][j]=0.0;
```

***(matrx+i*50+j)**

```
for (count=0, p=matrx[0]/*p=&matrx[0][0]*/; count<2500;count++)
    *(p++)=0.0;
```

Ponteiros e Vetores

- Vetores como ponteiros

Há uma **diferença** entre o nome de um vetor e um ponteiro que deve ser **destacada**: um ponteiro é uma variável. Mas, o nome de um vetor não é uma variável. Isto significa, que não se consegue alterar o endereço que é apontado pelo "nome do vetor". Logo:

```
int vetor[10], *ponteiro, i;
ponteiro = &i;
/* as operações a seguir são inválidas */
vetor = vetor + 2;
vetor++;
vetor = ponteiro;
```

Ponteiros e Vetores

- Vetores como ponteiros

Se testarmos as operações anteriores em nossos compiladores. Eles darão, por exemplo, uma mensagem de erro do tipo:

- Lvalue;
- incompatible types in assignment.

UNIVASF

Ponteiros e Vetores

- Vetores como ponteiros

Exercício:

Construa um programa que declare uma matriz [3,4] de inteiros e a inicializa da seguinte forma:

$$\begin{bmatrix} 01 & 02 & 03 & 04 \\ 05 & 06 & 07 & 08 \\ 09 & 10 & 11 & 12 \end{bmatrix}$$

Depois, a imprima na saída padrão com o layout apresentado. As manipulações da matriz deve ser feitas utilizando um ponteiro.

UNIVASF

```c
#include <stdio.h>
#define nl 3
#define nc 4
int main () {
    int matriz[nl][nc],*p,i;
    for (i=0, p=&matriz[0][0];i<nl*nc;i++)
        *(p++)=i+1;
    for (i=0, p=matriz[0];i<nl*nc;i++)
        if (!(i%nc))
            printf ("| %02d ",*(p+i));
        else
            if (i%4==nc-1)
                printf("%02d |\n",*(p+i));
            else
                printf("%02d ",*(p+i));
}
```

$$\begin{bmatrix} 01_0 & 02_1 & 03_2 & 04_3 \\ 05_4 & 06_5 & 07_6 & 08_7 \\ 09_8 & 10_9 & 11_{10} & 12_{11} \end{bmatrix}$$

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:

UNIVASF

Ponteiros e Vetores

- Ponteiros como vetores

```c
#include <stdio.h>
int main ()
{
    int matrx [] = { 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 constatar que p[2] é equivale a *(p+2).

Ponteiros e Vetores

- Ponteiros como vetores

Contudo, uma observação é necessária:

```
#include <stdio.h>
int 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]);
}
```

p=&matrx[0][0];

UNIVASF

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.

```c
#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);
}
```