Programação I - 2ª fase

Equivalência entre matrizes e ponteiros

Equivalência entre ponteiros e matrizes

- Há algumas similaridades entre ponteiros e matrizes. Eles não são iguais, mas a forma como um programa em C opera estes elementos internamente, acaba criando uma relação entre eles.

- A principal característica é o fato de que operações com matrizes são sempre baseadas em ponteiros. Em um programa em C, o **nome de uma matriz é sempre tratado como um ponteiro para o primeiro elemento da matriz**, exceto na declaração (definição) da variável.

Equivalência entre ponteiros e matrizes

- Exemplo:

```
int vetor[10]; /* “vetor” é o nome da matriz */
int *ptrVetor; /* “ptrVetor” é um ponteiro para um inteiro */
int *ptrInt;   /* “ptrInt” é um ponteiro para um inteiro */

/* as linhas abaixo têm o mesmo significado:
   UM PONTEIRO RECEBE UMA MATRIZ DO MESMO TIPO */
ptrVetor = vetor;
ptrVetor = &vetor[0]; /* mesmo significado da linha anterior */
ptrInt   = vetor;
ptrInt   = &vetor[0]; /* mesmo significado da linha anterior */
```

- Depois de obter o endereço de memória do primeiro elemento do vetor, o ponteiro pode ter acesso a todos os outros elementos.

Equivalência entre ponteiros e matrizes

```
ptrVetor = vetor;
ptrVetor = &vetor[0]; /* mesmo significado da linha anterior */

ptrInt   = vetor;
ptrInt   = &vetor[0]; /* mesmo significado da linha anterior */
```

O que é preciso considerar para entender as linhas acima:

1) Os ponteiros “ptrVetor” e “ptrInt” são ponteiros do tipo inteiro (int);
2) A matriz “vetor” é uma matriz de inteiros;
3) A operação de atribuição (=) é permitida entre ponteiros de mesmo tipo;
4) A linguagem C trata o nome de uma matriz como se fosse um ponteiro para o primeiro elemento da matriz;
5) Então um ponteiro para “int” pode receber uma matriz unidimensional do tipo “int”.

Equivalência entre ponteiros e matrizes

- **É importante ressaltar que quando a matriz é de um tipo e o ponteiro é de outro, não é possível fazer a atribuição.**

```
int vetor[10];    /* “vetor” é o nome da matriz */
float *ptrVetor;  /* “ptrVetor” é um ponteiro para um
                     “float” */


/* as linhas abaixo ESTÃO INCORRETAS, pois a matriz e o
    ponteiro são de tipos diferentes */
ptrVetor = vetor;  //ERRADO
ptrVetor = &vetor[0];  //ERRADO
```

Equivalência entre ponteiros e matrizes

- Assim como em uma matriz, um ponteiro também pode ter um índice ([i]).
  - É desta forma que podemos usar ponteiros para acessarmos os elementos de uma matriz

```
int vetor[10]; /* “vetor” é o nome da matriz */
int *ptrVetor; /* “ptrVetor” é um ponteiro para um inteiro */

ptrVetor = vetor;
for (i=0; i < 10; i++)
    scanf("%d", &ptrVetor[i]); //lendo elementos
for (i=0; i < 10; i++)
    printf("%d", ptrVetor[i]); //mostrando elementos
```

Equivalência entre ponteiros e matrizes

- O que você não pode fazer?
  - atribuir um vetor a outro:

```
int v1[10], v2[10];
v1 = v2; /* INCORRETO */
```

  - atribuir um ponteiro a um vetor:

```
int a[10], *pInt;
v = pInt; /* INCORRETO */
```

Equivalência entre ponteiros e matrizes

- A equivalência entre ponteiros e matrizes pode ser explorada na passagem de parâmetros para funções:
  - Se uma matriz pode ser atribuída a um ponteiro então eu posso criar funções cujos parâmetros são ponteiros que receberão valores.
  - Observe que no exemplo abaixo temos uma passagem por referência na função "lerMatrizr". Portanto, a equivalência entre ponteiros e matrizes explica o conceito de que "toda matriz é passada por referência para uma função".

```
main() {
  int vetor[10];
/* uma função para ler
   um vetor */
  lerMatriz(vetor);
...
}
```

```
/* implementação da função
lerVetor */
void lerMatriz (int *pInt) {
    int i;
    for (i = 0; i < 10; i++)
        scanf("%d", &pInt[i]);
}
```

Equivalência entre ponteiros e matrizes

- Quando você usar strings como parâmetros de funções também poderá usar ponteiros para o tipo `char` em vez de uma matriz de `char`
  - **Lembre-se que strings em C nada mais são do que matrizes (vetores) de char**
  - **Observe que no exemplo abaixo temos a passagem de um string para a função "lerPalavra".**
  - **Enquanto você não aprender alocação dinâmica de memória, não use declarações de string do tipo "`char *string`" para variáveis locais. Use-as somente para parâmetros de funções cujos argumentos sejam strings declarados como vetores de `char`.**

```
main() {
  char palavra[31];
  /* uma função para ler
  uma palavra (string) */
  lerPalavra(palavra);
...
}
```

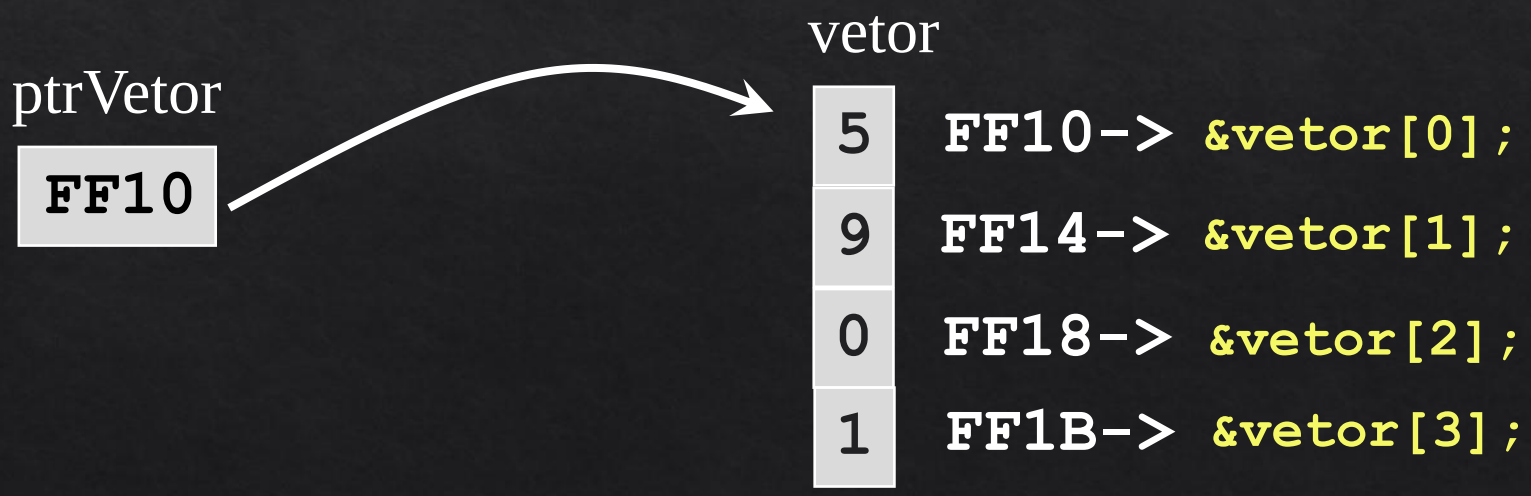
```
/* implementação da função
lerPalavra */
void lerPalavra (char *palavra) {
    scanf("%s", palavra);
}
```

Equivalência entre ponteiros e matrizes

- Resumo (1)

```
int vetor[10];
int *ptrVetor;

ptrVetor = vetor;
OU
ptrVetor = &vetor[0];
```



Equivalência entre ponteiros e matrizes

- Resumo (2)

```
int vetor[10];
int *ptrVetor;

ptrVetor = vetor;
OU
ptrVetor = &vetor[0];
```