Vetores e matrizes

Vejamos como são declarados, inicializados e utilizados vetores e matrizes em C.

Vetores

Como em outras linguagens de programação, C permite declarar e utilizar vetores. Um **vetor** é uma sequência de variáveis de mesmo tipo e referenciadas por um nome único.

As principais características de vetores em C são:

- os valores são acessíveis individualmente através de índices;
- Os elementos do vetor ocupam posições contíguas de memória;
- Os vetores têm tamanho predefinido e fixo; e
- Ao contrário de Pascal, o índice do primeiro elemento é 0 (zero).

Declaração

Um vetor é declarado da seguinte forma em C:

```
  <tipo_base> nome_do_vetor [<tamanho_do_vetor>] ;
```

Alguns exemplos:

```
int   valor[5];         // vetor de 5 inteiros, índices em 0..4
float temperatura[24];  // vetor de 24 temperaturas
char  digito[10];       // vetor de 10 caracteres
```

Acesso aos elementos (i)

O operador pós-fixado **[n]** (par de colchetes) denota o conteúdo da posição **n** do vetor. O índice de um vetor *deve* ser do tipo **int**.

```
int valor[5];         // vetor de 5 inteiros
int n, x, y;

   x = valor[3];      // atribui a x o quarto elemento do vetor
   ...
   n = ...            // n DEVE estar no intervalo 0..4
   y = valor[n];      // atribui a y o n-ésimo elemento do vetor
```

Uma prática recomendada é definir a dimensão do vetor usando uma **macro** de preprocessador, para facilitar futuras alterações no código:

```
#define MAXVET 1000

float v[MAXVET];

for (i=0; i<MAXVET; i++)
  v[i] = 0.0;
```

Inicialização

Os elementos de um vetor podem ser inicializados durante sua declaração, como mostram os exemplos a seguir:

```
short value[5] = { 32, 475, 58, 119, 7442 } ;  // inteiro de 16 bits

float temperatura[24] =
{
  17.0, 18.5, 19.2, 21.4, 22.0, 23.5,
  24.1, 24.8, 25.8, 26.9, 27.1, 28.9,
  29.5, 31.0, 32.3, 33.7, 34.9, 36.4,
  37.0, 38.5, 39.6, 40.5, 42.3, 44.2
} ;

char digito[10] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' } ;
```

Importante: considerando o padrão C ANSI, o compilador precisa saber o tamanho do vetor para reservar memória para ele durante a compilação. Por isso, o número de elementos do vetor deve ser uma constante (ou uma expressão cujo resultado seja constante).

Por exemplo, a seguinte declaração de vetor não é permitida em ANSI C (mas é válida em C99):

```
int func (int size)
{
  int vetor [size]; // proibido em ANSI C, pois "size" não é constante
}
```

Acesso aos elementos (ii)

O acesso aos elementos de um vetor se faz de forma similar a outras linguagens: basta indicar o nome do vetor e a posição (índice) que se deseja acessar.

Em C, o índice de um vetor sempre inicia na posição 0 e termina na posição *size-1*.

Exemplos:

```
value[0] = 73 ;
value[3] = value[2] + 10 ;
value[4]++ ;                // obtém o conteúdo de value[4] e o incrementa

for (i=0; i < 24; i++)  // "i" deve ir de 0 a 23
  temperatura[i] = 20 + i/2 ;
```

Os elementos de um vetor **sempre** ocupam posições contíguas e de mesmo tamanho na memória. Por exemplo, o vetor "value[5]" será alocado na memória RAM da seguinte forma (lembrando que "sizeof(short)" = 2 bytes):

Endereço	addr	addr+2	addr+4	addr+6	addr+8
Elemento	value[0]	value[1]	value[2]	value[3]	value[4]
Conteúdo	32	475	58	119	7442

Para gerar código executável eficiente, o compilador não faz **nenhuma verificação de índices** de vetores nem gera código para essas verificações durante a execução. Por exemplo, uma escrita em "value[8]" modifica a memória na posição *addr+16*, que não pertence ao vetor "value", pois os índices desse vetor vão de 0 a 4, o que pode provocar comportamento instável ou erros de acesso à memória e encerramento inesperado do programa.

É *responsabilidade do programador* garantir que os índices acessados em um vetor estejam sempre dentro dos limites alocados ao vetor.

Finalmente, deve-se observar que não é possível atribuir um vetor a outro diretamente; os elementos devem ser copiados individualmente:

```
int v1[10], v2[10] ;

v1 = { 2, 3, 4, 7, 2, 1, 9, 2, 3, 4 } ;
v2 = v1 ; // NÃO FUNCIONA
```

Vetores e ponteiros

Um vetor pode ser visto como um ponteiro para uma área de memória que contém uma sequência de valores do mesmo tipo.

O operador pré-fixado **&** (e-comercial) retorna o *endereço* de seu operando. Para uma variável **x**, a expressão **&x** denota o endereço da variável, e não o seu conteúdo.

O operador pré-fixado * (asterisco) retorna o *conteúdo* de seu operando, que deve ser um endereço. Para uma variável **x**, a expressão ***(&x)** denota o conteúdo da posição cujo endereço é aquele da variável **x**.

O nome "valor" declarado acima é equivalente a "&valor[0]" (endereço do primeiro elemento do vetor).

Dessa forma, o nome de um vetor é visto como um ponteiro (*o endereço*) para dados do tipo definido no vetor e aponta para o endereço do primeiro elemento do vetor.

Em consequência, as seguintes declarações são equivalentes:

```
int *ptr ;     // ponteiro para inteiros
int valor[5] ; // endereço de um vetor de 5 inteiros

ptr = valor;   // atribui a ptr o endereço de valor[0]
...
x = *(ptr+3);  // atribui a x o quarto elemento de valor: *(ptr+3) == valor[3]
```

Matrizes

A linguagem C não oferece um tipo *matriz* nativo. Em vez disso, matrizes são implementadas como vetores de vetores.

Declaração

A forma geral de declaração de uma matriz com N dimensões é a seguinte:

```
  <tipo_base> nome [<tam 1>] [< tam 2 >] ... [< tam N >] ;
```

Alguns exemplos:

```
char  tabuleiro [8][8] ;  // matriz de 8x8 posições (caracteres)
float cf [2][3] ;         // matriz de 2x3 coeficientes reais
int   faltas[31][12][5] ; // matriz de 31 dias x 12 meses x 5 anos
```

Similarmente aos vetores, as matrizes também podem ser inicializadas durante sua declaração:

```
float cf [2][3] = {
                    {  -3.4,  2.1, -1.0 },  // primeira linha
                    {  45.7, -0.3,  0.0 }   // segunda linha
                  } ;
```

A matriz "cf" acima é vista pelo compilador C como um vetor de 2 elementos, sendo cada elemento um vetor de 3 inteiros. Assim, "cf[0]" é o vetor [-3.4, 2.1, -1.0] e "cf[1]" o vetor [45.7, -0.3, 0.0].

A alocação de uma matriz na memória é feita de forma linear e contígua, com um elemento

imediatamente após o outro. Por exemplo, a matriz "cf[2][3]" acima seria alocada na memória desta forma (lembrando que cada "float" ocupa 4 bytes):

```
Endereço   addr                   addr + 3*sizeof(float)
Elemento   cf[0]                  cf[1]
Conteúdo   [ -3.4, 2.1 , -1.0 ]   [ 45.7, -0.3, 0.0 ]
```

Ou, mais detalhadamente:

```
Endereço    addr      addr+4    addr+8    addr+12   addr+16   addr+20
Elemento    cf[0][0]  cf[0][1]  cf[0][2]  cf[1][0]  cf[1][1]  cf[1][2]
Conteúdo    -3.4      2.1       -1.0      45.7      -0.3      0.0
```

Deve-se ter cuidado especial na declaração de matrizes, pois o espaço de memória ocupado por uma matriz cresce exponencialmente com o número de dimensões. Por exemplo:

- float m[100][100] ocupa 40 Kbytes de memória ($100^2 \times 4$ bytes)
- float m[100][100][100][100] ocupa 400 Mbytes ($100^4 \times 4$ bytes).

Acesso

O acesso aos valores de uma matriz se faz de forma similar ao vetor:

```
int matriz[5][5] ;

matriz[0][3] = 73 ;
```

ou

```
#define DIM 8

char tabuleiro[DIM][DIM] ;

// "limpa" o tabuleiro
for (i=0; i<DIM; i++)
  for (j=0; j<DIM; j++)
    tabuleiro[i][j] = ' ' ;
```

Exercícios

Escreva programas em C para:

1. a. Ler um número N e um vetor de N inteiros;
   b. calcular a **média dos valores lidos**;
   c. imprimir essa média;
   d. imprimir os elementos do vetor maiores do que a média calculada.
2. a. Ler um número N e um vetor de N inteiros;
   b. ordenar o vetor lido usando a técnica de **ordenação da bolha**;
   c. imprimir os elementos do vetor ordenado;
   d. melhorar o programa anterior, percorrendo o vetor nos dois sentidos e evitando percorrer as pontas (valores já ordenados).
3. a. Ler um número N e um vetor de N inteiros;
   b. ordenar o vetor lido usando a técnica de **ordenação por seleção**;
   c. imprimir os elementos do vetor ordenado.
4. Ler uma matriz de inteiros 4x4, calcular e imprimir sua **transposta**.
5. Ler duas matrizes de inteiros 4x4, calcular e imprimir seu **produto**.