2025/11/18 06:50 1/5 Vetores e matrizes

Vetores e matrizes

Neste módulo veremos a foma 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
- As entradas do vetor ocupam posições contíguas de memória
- Os vetores têm tamanho predefinido e fixo
- Os vetores sempre iniciam na posição 0

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
float temperatura[24];  // vetor de 24 temperaturas
char digito[10];  // vetor de 10 caracteres
```

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 ;</pre>
```

Inicialização

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

```
short int valor[5] = { 32, 475, 58, 119, 7442 } ;

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' };
```

Acesso

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:

```
short int valor[5] ;

valor[0] = 73 ;
valor[3] = valor[2] + 10 ;
valor[4]++ ;

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

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

```
short int valor[5] = { 32, 475, 58, 119, 7442 };
```

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

Para gerar um 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 valor[5] irá modificar a memória na posição addr+10, que não pertence ao vetor valor (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 mesmo.

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

```
#define SIZE 10
int v1[SIZE], v2[SIZE] ;
v1 = { 2, 3, 4, 7, 2, 1, 9, 2, 3, 4 } ;
```

```
// NÃO FUNCIONA...
v2 = v1 ;

// FUNCIONA!
for (int i = 0; i < SIZE; i++)
   v2[i] = v1[i] ;

// usando cópia de memória (mais rápida)
memcpy (v2, v1, SIZE * sizeof(int)) ;</pre>
```

Vetores de tamanho variável

No padrão C89, o compilador precisa saber qual o tamanho do vetor para poder reservar memória para ele durante a compilação. Por isso, o número de elementos do vetor deve ser uma constante ou uma expressão/macro cujo resultado seja constante:

```
#define MAX 10

int vet1[MAX} ; // válido em C89, pois MAX é constante
int vet2[100] ; // válido em C89, pois 100 é constante

int func (int size)
{
   int vet3[size]; // inválido em C89, pois "size" não tem valor definido durante a compilação
   ...
}
```

O padrão C99 introduziu a definição de vetores de tamanho variável, ou VLA - Variable-Lenght Arrays, que permitem definir vetores com tamanho variável em tempo de execução, como vet3 no exemplo acima.

Algumas observações importantes sobre VLAs:

- VLAs não podem ser variáveis globais ou estáticas, somente variáveis locais.
- VLAs são alocados na área de memória chamada "pilha de execução", que costuma ser pequena (alguns MBytes).
- VLAs são fontes potenciais de bugs de segurança e **devem ser evitados** quando possível.

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> 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 coef[2][3] =
{
    {       -3.4,     2.1, -1.0 },
    {       45.7, -0.3,      0.0 }
} ;
```

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

A alocação de uma matriz na memória é feita de forma linear e contígua, ou seja, 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)		
Entrada	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
Entrada	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 $(100^2 \times 4 \text{ bytes})$
- float m[100][100][100][100] ocupa 400 Mbytes (100⁴ × 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] = ' ';</pre>
```

2025/11/18 06:50 5/5 Vetores e matrizes

Exercícios

Escrever programas em C para:

- 1. Leitura e escrita:
 - 1. ler um número N e um vetor de N inteiros
 - 2. Escrever o vetor na saída no formato [n1 n2 n3 ...]
 - 3. Inverter o vetor e imprimi-lo: [... n3 n2 n1] (inverter antes de imprimir)
- 2. Cálculo da média:
 - 1. ler um número N e um vetor de N inteiros
 - 2. calcular a média dos valores lidos
 - 3. imprimir essa média
 - 4. imprimir os elementos do vetor que forem maiores que a média calculada
- 3. Ordenação I:
 - 1. ler um número N e um vetor de N inteiros
 - 2. ordenar o vetor lido usando a técnica de ordenação da bolha
 - 3. imprimir os elementos do vetor ordenado
- 4. Melhorar o programa anterior, percorrendo o vetor nos dois sentidos e evitando percorrer as pontas (ou seja, valores já ordenados)
- 5. Ordenação II:
 - 1. ler um número N e um vetor de N inteiros
 - 2. ordenar o vetor lido usando a técnica de ordenação por seleção
 - 3. imprimir os elementos do vetor ordenado
- 6. Ler uma matriz, calcular e imprimir sua transposta
- 7. Ler duas matrizes, calcular e imprimir sua multiplicação

From:

https://wiki.inf.ufpr.br/maziero/ - Prof. Carlos Maziero

Permanent link:

https://wiki.inf.ufpr.br/maziero/doku.php?id=c:vetores

Last update: 2024/10/01 17:30

