VARIÁVEIS COMPOSTAS HOMOGÊNEAS

VETORES UNIDIMENSIONAIS

Até agora, uma variável está associada a uma posição de memória e qualquer referência a ela significa um acesso ao conteúdo de um pedaço de memória cujo tamanho depende de seu tipo.

A partir de agora, um novo tipo de estrutura de dados será utilizado. Esta estrutura possibilita associar um identificador a um conjunto de elementos de mesmo tipo. Em programação, este tipo de estrutura de dados é chamada de `vetor` (ou *`array`*, em inglês) ou, de maneira mais formal `estrutura de dados homogênea`.

Variáveis compostas homogêneas correspondem às posições de memória, identificadas por um mesmo nome individualizadas por índices cujo conteúdo é do mesmo tipo; iniciam com índice `0` (primeiro elemento do vetor) e vão até o último elemento declarado na variável.

A declaração de um vetor é feita usando a seguinte sintaxe:

Sintaxe: `tipo nome[tamanho];`

em que

`tipo` é o tipo dos elementos do vetor: `int`, `float`, `double`, ...;
`nome` é o nome identificador do vetor;
`tamanho` é o tamanho do vetor, ou seja, o número de elementos que o vetor pode armazenar.

Naturalmente, no caso de trabalhar com `N` elementos, então o vetor deve ser declarado com pelo menos `N` posições no `tamanho`.

Exemplo:
```
float exemplo[20];
int idade[100];
float nota[40];
char nome[35];
```

Na declaração de um vetor reserva-se espaço de memória para os elementos de um vetor. A quantidade de memória (em *bytes*) usada para armazenar um vetor é calculada como:

```
quantidade de memória = tamanho do tipo * tamanho do vetor
```

Exemplo: A quantidade de memória utilizada pelos vetores no exemplo anterior, respectivamente é:
```
4*20 = 80 bytes
2*100 = 200 bytes
4*40 = 160 bytes
1*35 = 35 bytes
```

- **Referência a elementos de vetor**

Cada elemento do vetor é referenciado pelo `nome do vetor` seguido de um índice inteiro. O primeiro elemento do vetor tem índice `0` e o último tem índice `tam-1`. O índice de um vetor deve ser inteiro.

Exemplo:

1)
```
#define MAX 15
int i = 8;
float valor[MAX];          // declaração de vetor
valor[1] = 6.645;
valor[MAX-1] = 3.867;
valor[i] = 7.645;
valor[rand() % MAX] = 2.768;
valor[sqrt(MAX)] = 2.705; // NÃO é válido!
```

2)
```
#include <stdio.h>
int main() {
  int i, vetor[10];
  for (i = 0; i < 10; i++) {
    vetor[i] = 2*i;
    printf ("%d\n",vetor[i]);
  }
}
```

- **Iniciação de vetores**

Assim como pode-se inicializar variáveis, por exemplo `int aux = 3;`, pode-se iniciar vetores.

Sintaxe: `tipo nome[tamanho] = {lista de valores};`

em que

`lista de valores` é uma lista, separada por vírgulas, dos valores de cada elemento do vetor.

Exemplo:
```
int dia[7] = {12, 30, 14, 7, 13, 15, 6};
float nota[5] = {8.4, 6.9, 4.5, 4.6, 7.2};
char vogal[5] = {'a', 'e', 'i', 'o', 'u'};
```

Opcionalmente, pode-se iniciar os elementos do vetor enumerando-os um a um.

Exemplo: As duas iniciações a seguir são possíveis:

```
float nota[4] = {5.5, 7.2, 9.7, 2.3};
```

ou

```
float nota[4];
nota[0] = 5.5;
```

```
nota[1] = 7.2;
nota[2] = 9.7;
nota[3] = 2.3;
```

- **Limites de um vetor**

Embora na declaração do vetor, define-se seu tamanho, a linguagem C não verifica se o índice está dentro dos limites válidos.

Se declarar um vetor `int valor[10]`, teoricamente só tem sentido usar os elementos `valor[0]`, `valor[1]`, ..., `valor[9]`. Porém a linguagem C não acusa *erro* se utilizar `valor[18]` em algum lugar do programa.

Desta forma, se o programador não tiver atenção com os limites de validade para os índices ele corre o risco de ter dados sobrescritos ou do computador travar. Vários erros sérios (bugs) terríveis podem surgir.

Estes testes de limite devem ser feitos logicamente dentro do programa.

Exercícios

1) Dado o vetor `vet` definido por:

```
int vet[100];
```

   a) preencha `vet` com o valor 10;
   b) preencha o vetor com os números 1, 2, 3,..., 100;

2) Faça um programa em C que leia uma quantidade `N` de alunos e a nota de cada um dos alunos. O programa deve calcular a média aritmética das notas e, contar quantos alunos estão com a nota acima de 5.0. Se nenhum aluno tirou nota acima de 5, o programa deve imprimir a mensagem: "*Não há nenhum aluno com nota acima de 5".*

3) Faça um programa que leia um vetor `N[20]`. A seguir, encontre o menor elemento do vetor `N` e a sua posição dentro do vetor, mostrando: "*O menor elemento de N é", M, "e sua posição dentro do vetor é:",P*.

4) Escreva um programa que leia dois vetores de 10 posições e faça a multiplicação dos elementos de mesmo índice, colocando o resultado em um terceiro vetor. Mostre o vetor resultante.

- **Passagem de vetores como parâmetros para funções**

Vetores, assim como variáveis, podem ser usados como argumentos de funções.

Na *passagem de vetores* para funções utiliza-se a seguinte sintaxe:

Sintaxe: `nome_da_funcao (nome_do_vetor)`

em que

`nome_da_funcao` é o nome da função que se está chamando;
`nome_do_vetor` é o nome do vetor que se deseja passar. Indica-se **apenas** o nome do vetor, **sem** índices.

Na *declaração de funções* que recebem vetores por parâmetros:

Sintaxe:
```
tipo_funcao nome_função (tipo_vetor nome_vetor[]) {
  /* ... */
}
```

em que

`tipo_funcao` é o tipo de retorno da função;
`nome_funcao` é o nome da função;
`tipo_vetor` é o tipo dos elementos do vetor;
`nome_vetor` é o nome do vetor (depois do nome do vetor há um índice vazio [] para indicar que está se recebendo um vetor).

Exemplo:

Na declaração da função:
```
float soma (int n, float vetor[]) {
  /* ... */
}
```

Na chamada da função:
```
int main() {
  float valor[50]; // declaração do vetor
  /* ... */
  s = soma (n,valor); // passagem do vetor para a função
  /* ... */
}
```

Observação: Ao contrário das variáveis comuns, o conteúdo de um vetor pode ser modificado pela função chamada. Isto significa que pode se passar um vetor para uma função e alterar os valores de seus elementos. Isto ocorre porque a passagem de vetores para funções é feita de modo especial dito *passagem por referência*. Portanto, deve-se *ter cuidado* ao manipular os elementos de um vetor dentro de uma função para não modifica-los por descuido.

Exemplo:

```
#include <stdio.h>
// Rotina auxiliar - troca
void troca (int *a, int *b) {
  int aux;
  aux = *a;
  *a = *b;
  *b = aux;
}
// Impressao do vetor
void mostraVetor (int tam, int v[]) {
  int i;
  for (i = 0; i < tam; i++)
    printf ("%d ",v[i]);
  printf ("\n");
}
// Busca em um vetor nao ordenado
int busca (int tam, int v[], int elem) {
  int i;
  for (i = 0; i < tam; i++)
    if (v[i] == elem) return 1;
  return 0;
}
// Busca em um vetor ordenado
int buscaOrd (int tam, int v[], int elem) {
  int i;
  for (i = 0; i < tam && v[i]<=elem; i++)
    if (v[i] == elem) return 1;
  return 0;
}
// Ordena vetor
int ordenaVetor (int tam, int v[]) {
  int i, j;
  for (i = 0; i < tam-1; i++)
    for (j = i+1; j < tam; j++)
      if (v[i] > v[j]) troca(&v[i],&v[j]);
}
// Programa principal
int main() {
  int i, j;
  int vetor[10] = {3, 5, 4, 6, 1, 2, 0, 8, 9, 7};
  printf ("%d \n\n",busca (10,vetor,13));
  mostraVetor (10,vetor);
  ordenaVetor (10,vetor);
  mostraVetor (10,vetor);
  printf ("%d \n\n",buscaOrd (10,vetor,3));
  printf ("\n\n");
}
```

Observação: Ordenação por Troca – Bubble Sort

Um método simples de ordenação por troca é a estratégia conhecida como bolha que consiste, em cada etapa "borbulhar" o maior elemento para o fim da lista. Inicialmente percorre-se a lista dada da esquerda para a direita, comparando pares de elementos consecutivos, trocando de lugar os que estão fora da ordem.

No exemplo abaixo, em cada troca, o maior elemento é deslocado uma posição para a direita.

Varredura	v[0]	v[1]	v[2]	v[3]	Troca
1	10	9	7	6	0 e 1
	9	10	7	6	1 e 2
	9	7	10	6	2 e 3
	9	7	6	10	Fim da Varredura 1

Após a primeira varredura o *maior* elemento encontra-se alocado em sua posição definitiva na lista ordenada. Pode-se deixá-lo de lado e efetuar a segunda varredura na sub lista `v[0],v[1],v[2]`. Segue a continuação do exemplo:

Varredura	v[0]	v[1]	v[2]	v[3]	Troca
2	9	7	6	10	0 e 1
	7	9	6	10	1 e 2
	7	6	9	10	Fim da Varredura 2

Após a segunda varredura, o *maior* elemento da sublista `v[0],v[1],v[2]` encontra-se alocado em sua posição definitiva. A próxima sublista a ser ordenada é `v[0],v[1]`. Segue a continuação do exemplo:

Varredura	v[0]	v[1]	v[2]	v[3]	Troca
3	7	6	9	10	0 e 1
	6	7	9	10	Fim da Varredura 3

Função Bubble Sort:

```
void BubbleSort (int n, int v[]) {
  int i, j, aux;
  int trocado = 1;
  for (i=0; i<n-
1 && trocado; i++) {
    trocado = 0;
    for (j=0; j<n-i-1; j++) {
      if (v[j] > v[j+1]) {
        trocado = 1;
        troca (&v[j], &v[j+1]);
      }
    }
  }
}
```

VETORES MULTIDIMENSIONAIS

Vetores podem ter mais de uma dimensão, isto é, mais de um índice de referência. Podemos ter vetores de duas, três, ou mais dimensões.

A declaração de um vetor de mais de uma dimensão é feita semelhante ao vetor unidimensional, ou seja, utiliza-se usando a seguinte sintaxe:

Sintaxe: `tipo nome[tam_1][tam_2]...[tam_n];`

em que

`tipo` é o tipo dos elementos do vetor: `int`, `float`, `double`, ...;
`nome` é o nome identificador do vetor;
`[tam_1][tam_2]...[tam_n]` é o tamanho de cada dimensão do vetor.

Os índices dos vetores multidimensionais, também começam em `0`. Dessa forma, `vet[0][0]` é o primeiro elemento do vetor.

Exemplos:

1)
```
#define lin 20
#define col 30
int matriz[lin][col];
```

2)
```
#include <stdio.h>
#include <stdlib.h>
int main() {
  int i, j;
  int matriz[10][10];
  // Preenchimento da matriz
  for (i = 0; i < 10; i++)
    for (j = 0; j < 10; j++)
      matriz[i][j] = i*j;
  // Impressao da matriz
  for (i = 0; i < 10; i++) {
    for (j = 0; j < 10; j++)
      printf ("%3d ",matriz[i][j]);
    printf ("\n");
  }
  printf ("\n\n");
  system ("pause");
}
```

- **Iniciação de vetor multidimensional**

A iniciação de vetores multidimensionais é feita de modo semelhante aos vetores unidimensionais.

Sintaxe: `tipo nome[tam_1][tam_2]...[tam_n]={{lista},{lista},...{lista}};`

em que

`tipo` é o tipo dos elementos do vetor;
`nome` é o nome do vetor;
`[tam_1][tam_2]...[tam_n]` é o tamanho de cada dimensão do vetor;
`{{lista},{lista},...{lista}}` são as listas de elementos.

Exemplos:

1)
```
float valor[5][4] = {{8.4,7.4,5.7,2.5},
                     {6.9,2.7,4.9,3.3},
                     {4.5,6.4,8.6,9.9},
                     {6.4,4.6,8.9,6.3},
                     {1.4,7.2,3.6,7.7}};
```

2)
```
int tabela[2][3][2] = {{{10,15}, {20,25}, {30,35}},
                       {{40,45}, {50,55}, {60,65}}};
```

3)
```
#include <stdio.h>
int main() {
  int i, j;
  float mat[3][4] ={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 4; j++)
      printf ("%.2f ",mat[i][j]);
    printf ("\n");
  }
}
```

4)
```
#include <stdio.h>
int main() {
  int i, j;
  float mat[][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 4; j++)
      printf ("%.2f ",mat[i][j]);
    printf ("\n");
  }
}
```

Observação: Apenas a primeira dimensão pode ser omitida!

~~`float mat[][] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};`~~

- **Passagem de vetores multidimensionais para funções**

A sintaxe para *passagem* de vetores multidimensionais para funções é semelhante à passagem de vetores unidimensionais: chama-se a função e passa-se o **nome** do vetor, **sem** índices.

A única mudança ocorre na declaração de funções que recebem vetores. Na *declaração* de funções que recebem vetores:

Sintaxe:
```
tipo_f funcao (tipo_v vetor[tam_1][tam_2]...[tam_n]) {
  /* ... */
}
```

Note que depois do nome do vetor tem-se os índices com contendo os tamanhos de cada dimensão do vetor.

Exemplo:

Na declaração da função:
```
float soma (int n, int m, float vetor[6][10]) {
  /* ... */
}
```

Na chamada da função:
```
int main() {
  float valor[6][10]; // declaração do vetor
  /* ... */
  s = soma (l,c,valor); // passagem do vetor para a função
  /* ... */
}
```

Exercícios

1) Escreva um programa que leia os elementos de uma matriz 3x4 de números reais e, em seguida imprima os elementos no seguinte formato:
```
x.xx   x.xx   ...   x.xx
x.xx   x.xx   ...   x.xx
...    ...    ...   ...
x.xx   x.xx   ...   x.xx
```

2) Faça um programa que leia duas matrizes 2x3 de números inteiros e imprima a soma destas duas matrizes.

Definição de tipos: `typedef`

Sintaxe: `typedef tipo identificador;`

Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
#define max 100

typedef int inteiro;
typedef int vetor[max];

void imprime (int n, vetor v) {
  int i;
  for (i = 0; i < n; i++)
    printf ("%d ",v[i]);
}

int main() {
  inteiro i, n = 10;
  vetor vet = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  imprime (10, vet);
  printf ("\n\n");
  system ("pause");
}
```