FACAMP
FACULDADES DE CAMPINAS

Estruturas de Dados

Aula 3 – Representação de arranjos de dados: vetores e matrizes.

1º semestre de 2020
Prof. José Martins Jr

Arranjos unidimensionais (arrays)

- Conjuntos (arranjos) de elementos de mesmo tipo
- Um array tem tamanho fixo e pode ser definido em sua declaração

- Exemplo
  - Declaração de um array de números inteiros, com 100 posições na memória

```
int a[100];
```

  - As 100 posições são alocadas sequencialmente na memória e pode-se utilizar um índice para acesso a cada um de seus valores
  - Exemplos: a[0] indica o primeiro elemento do array, enquanto a[99] indica o último

Arranjos unidimensionais (arrays)

- O tamanho de um array também pode ser indicado pelo conjunto de elementos atribuídos

- Exemplo
  - Criar um array de 10 posições na memória

```
int numeros[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

  - E o seguinte laço fará a exibição dos elementos em tela

```
int i;
for (i=0; i<10; i++) {
    printf("numeros[%d]: %d\n", i, numeros[i]);
}
```

Arranjos bidimensionais (matrizes)

- Matrizes, em C, são representadas por arrays com mais de uma dimensão

- Exemplo
  - Declaração de uma matriz 4X5, contendo 20 posições com números inteiros

```
int m[4][5];
```

  - Exemplo: declaração e atribuição de valores para uma matriz 2X3

```
int matriz[2][3] = { {1, 2, 3} , {4, 5, 6} };
```

Matrizes multidimensionais

- Um arranjo pode ser definido com mais de duas dimensões
  - A representação de hipercubos é muitas vezes utilizada em organização de grandes bases de dados analíticas
  - Exemplo: matriz de 4 dimensões 2x2x2x2 (16 posições para inteiros)

```
int m4[2][2][2][2] = {
        {  { {1,  2}, { 3,  4} }, { { 5,  6}, { 7,  8} } },
        {  { {9, 10}, {11, 12} }, { {13, 14}, {15, 16} } }
};
```

  - Exemplo: matriz de 3 dimensões 4x4x4 (64 posições para inteiros)

```
int m5[4][4][4] = {  {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 16}},
          {{17, 18, 19, 20}, {21, 22, 23, 24}, {25, 26, 27, 28}, {29, 30, 31, 32}},
          {{33, 34, 35, 36}, {37, 38, 39, 40}, {41, 42, 43, 44}, {45, 46, 47, 48}},
          {{49, 50, 51, 52}, {53, 54, 55, 56}, {57, 58, 59, 60}, {61, 62, 63, 64}},
};
```

Alocação em memória

- A instância de arranjos na memória ocorre de forma **estática**, ou seja, a quantidade de memória é previamente reservada no momento de sua declaração ou atribuição
  - Quando um arranjo é declarado, informando-se as suas dimensões, é reservada em tempo de execução quantidade de memória equivalente ao número máximo de elementos que ele poderá conter
- Exemplos

```
int ar1[10];
```

  - Serão alocadas 10 posições de 4 bytes; a primeira posição é indicada por ar1[0]

```
int m1[3][4][5];
```

  - Serão reservadas 60 posições (3*4*5) de 4 bytes; a primeira será m1[0][0][0]

Alocação em memória

- Se um arranjo for declarado sem as dimensões, ele assumirá a quantidade de elementos que lhe forem atribuídos na declaração

- Exemplos

```
char *nomes[ ] = {"jose", "maria", "joao", "josefa"};
```

  - 4 posições de ponteiros para strings, e nelas guardados os endereços das constantes armazenadas na memória

```
int a2[ ] = {1, 2};
```

  - São reservadas 2 posições de 4 bytes, e atribuídos os valores dos inteiros

Percurso em arranjos

- A maneira tradicional de se recuperar elementos em um arranjo é através de seu índice
- Tal referência permite o acesso aleatório a qualquer posição do arranjo
  - Exemplo: tomando-se a matriz multidimensional fornecida (m5)
  - m5[1][2][3] refere-se ao elemento localizado na segunda linha (i=1), terceira coluna (j=2), quarta posição na dimensão 3 (k=3)
- Pode-se desejar realizar a busca em sequência dos elementos
  - Situações: quando se deseja a listagem total do arranjo, ou mesmo, quando um arranjo multidimensional for copiado para uma região da memória referenciada por um arranjo unidimensional

Percurso em arranjos

- Nessas situações, o uso da aritmética de ponteiros pode significar alguma vantagem
  - Uma variável do tipo ponteiro, ao ser incrementada, recebe em seu conteúdo o endereço do próximo elemento do tipo que aponta
  - Ou seja, ao incrementar um ponteiro para um inteiro, por exemplo, ele desloca 4 bytes na memória, correspondendo ao offset do tipo inteiro
  - Exemplo: tomando-se novamente a matriz multidimensional fornecida

```
int i, *p = m5;
for (i = 0; i < sizeof(m5)/sizeof(int); i++) {
    printf("%d.o = %d\t\t, no endereco= %u\n", i+1, *p, p);
    p++;
}
```

Cópia de arranjos em blocos

- Uma forma de se otimizar a cópia de um arranjo é realizar a cópia de toda a região da memória onde ele foi alocado, de uma vez só
- Isso pode ser muito útil também quando se recupera um conjunto de dados recebidos através de um buffer (leitura de arquivo ou recepção de um stream pela rede)
- A função memcpy permite realizar isso

```
void *memcpy(void *OUT, const void *IN, size_t N);
```

  - Exemplo:

```
int matriz1[2][3] = { {1, 2, 3} , {4, 5, 6} };
int matriz2[2][3];
memcpy(matriz2, matriz1, sizeof(matriz1));
```

Limpeza de arranjos em blocos

- Uma outra função útil é a memset

```
void *memset(void *OUT, int val, size_t n);
```

- Através dela é possível atribuir um valor para todo um bloco de memória (ou seja, “limpar” a memória)
  - Exemplo:

```
memset(matriz2, 0, sizeof(matriz2));
```

Bibliografia

- KERNIGHAN, B. W., RITCHIE, D. M. C, a linguagem de programação: padrão ANSI. Rio de Janeiro: Campus, 1989. 289 p.
- LOOSEMORE, S.; STALLMAN, R. M. et al. The GNU C Library Reference Manual. Disponível no endereço: http://www.gnu.org/software/libc/manual/