Programação de Computadores

Raoni F. S. Teixeira

Aula 18 - Matrizes

1 Introdução

Vimos anteriormente que vetores nos permitem armazenar coleções de valores do mesmo tipo. Por exemplo, para armazenar as notas de 10 questões de uma prova de um aluno:

```
float questoes[10];
int i;

for (i = 0; i < 10; i++) {
    printf("Nota␣da␣questao␣%d:␣", i);
    scanf("%f", &questoes[i]);
}
```

Esta solução funciona perfeitamente para um aluno. Mas e se precisarmos armazenar as notas de 10 questões para 100 alunos?

Uma primeira tentativa seria declarar 100 vetores diferentes:

```
float questoes1[10], questoes2[10], ..., questoes100[10];
int i;

for (i = 0; i < 10; i++) {
    printf("Nota␣da␣questao␣%d␣do␣aluno␣1:␣", i);
    scanf("%f", &questoes1[i]);
}
// ...
for (i = 0; i < 10; i++) {
    printf("Nota␣da␣questao␣%d␣do␣aluno␣100:␣", i);
    scanf("%f", &questoes100[i]);
}
```

Este código é claramente impraticável! Estaríamos repetindo o mesmo padrão 100 vezes, tornando o programa extremamente longo, difícil de manter e impossível de generalizar.

1.1 A Solução: Vetores de Vetores

Já conhecemos a solução para problemas semelhantes:

- **Problema:** Queríamos armazenar 100 variáveis do tipo `float nota`
- **Solução:** Usar um vetor de variáveis: `float notas[100]`

Agora temos um problema análogo:

- **Problema:** Queremos armazenar 100 vetores como `float questoes[10]`
- **Solução:** Usar um *vetor de vetores*: `float questoes[100][10]`

Esta estrutura de dados bidimensional é chamada de **matriz**.

2 Matrizes

2.1 Definição

Uma **matriz** é uma coleção de vetores, ou seja, um vetor bidimensional.

Características das matrizes:

- É um vetor como outro qualquer, mas com duas dimensões.
- Cada *linha* (vetor) é acessada por meio de um **índice primário**.
- Cada *elemento* dentro da linha é acessado por meio de um **índice secundário** (coluna).
- Todos os elementos são do mesmo tipo.
- O tamanho é fixo e definido em tempo de compilação.

2.2 Declaração de Matrizes

A declaração de uma matriz segue a sintaxe:

```
tipo identificador[linhas][colunas];
```

Onde:

- **tipo**: Tipo dos elementos (int, float, char, etc.)
- **linhas**: Número de linhas da matriz

- **colunas**: Número de colunas da matriz

Uma matriz possui linhas $\times$ colunas variáveis do tipo especificado.

Indexação: Em C, tanto as linhas quanto as colunas começam em 0:

- As linhas são numeradas de 0 a linhas $- 1$
- As colunas são numeradas de 0 a colunas $- 1$

Geralmente usamos m para denotar o número de linhas e n para o número de colunas, referindo-nos a uma matriz $m \times n$.

2.3 Exemplo de Declaração

A declaração

```
int matriz[5][4];
```

cria uma matriz com 5 linhas e 4 colunas ($5 \times 4 = 20$ elementos), ilustrada na Figura 1.

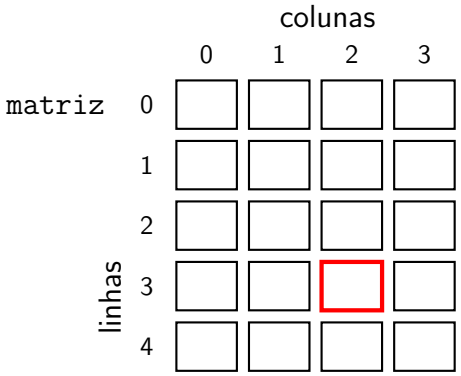

Figura 1: Representação de uma matriz 5×4. O elemento destacado é `matriz[3][2]`.

2.4 Acessando Elementos de uma Matriz

Para acessar ou modificar um elemento da matriz, usamos a notação com dois índices:

```
nome_da_matriz[linha][coluna]
```

Exemplos:

```
matriz[0][0] = 10;          // Primeiro elemento (linha 0, coluna 0)
matriz[4][3] = 25;          // Ultimo elemento de uma matriz 5x4
int x = matriz[2][1];       // Le o elemento da linha 2, coluna 1
```

IMPORTANTE: O compilador **não** verifica se você utilizou valores válidos para a linha e para a coluna. Acessar posições inválidas causa comportamento indefinido, assim como ocorre com vetores unidimensionais.

Exemplo de referência: `matriz[1][10]` refere-se ao elemento na 2ª linha (índice 1) e 11ª coluna (índice 10).

3 Operações com Matrizes

3.1 Lendo uma Matriz

Para ler todos os elementos de uma matriz, usamos laços aninhados. O laço externo percorre as linhas e o laço interno percorre as colunas:

```
int i, j;
int matriz[4][5];

for (i = 0; i < 4; i++) {
    for (j = 0; j < 5; j++) {
        printf("Valor da linha %d, coluna %d: ", i, j);
        scanf("%d", &matriz[i][j]);
    }
}
```

Este código lê uma matriz 4×5 (4 linhas, 5 colunas) do teclado, preenchendo linha por linha.

3.2 Imprimindo uma Matriz

Para imprimir uma matriz de forma organizada, também usamos laços aninhados:

```
int i, j;
int matriz[4][5];

for (i = 0; i < 4; i++) {
    printf("Linha %d: ", i);
    for (j = 0; j < 5; j++) {
        printf("%d ", matriz[i][j]);
    }
    printf("\n");
}
```

A saída será algo como:

```
Linha 0: 10 20 30 40 50
```

```
Linha 1: 15 25 35 45 55
Linha 2: 12 22 32 42 52
Linha 3: 18 28 38 48 58
```

3.3 Exemplo Completo: Soma de Matrizes

Vamos implementar um programa que lê duas matrizes 3×3 e calcula a matriz soma:

```
#include <stdio.h>
#define LINHAS 3
#define COLUNAS 3

int main() {
    int A[LINHAS][COLUNAS], B[LINHAS][COLUNAS], C[LINHAS][COLUNAS];
    int i, j;

    // Le a matriz A
    printf("Digite os elementos da matriz A:\n");
    for (i = 0; i < LINHAS; i++) {
        for (j = 0; j < COLUNAS; j++) {
            printf("A[%d][%d]: ", i, j);
            scanf("%d", &A[i][j]);
        }
    }

    // Le a matriz B
    printf("Digite os elementos da matriz B:\n");
    for (i = 0; i < LINHAS; i++) {
        for (j = 0; j < COLUNAS; j++) {
            printf("B[%d][%d]: ", i, j);
            scanf("%d", &B[i][j]);
        }
    }

    // Calcula C = A + B
    for (i = 0; i < LINHAS; i++) {
        for (j = 0; j < COLUNAS; j++) {
            C[i][j] = A[i][j] + B[i][j];
        }
    }

    // Imprime a matriz C
    printf("\nMatriz C = A + B:\n");
    for (i = 0; i < LINHAS; i++) {
        for (j = 0; j < COLUNAS; j++) {
            printf("%4d ", C[i][j]);
        }
```

```
        printf("\n");
    }

    return 0;
}
```

4 Vetores Multidimensionais

Embora matrizes bidimensionais sejam as mais comuns, C permite criar vetores com qualquer número de dimensões.

4.1 Declaração Geral

```
tipo identificador[dim1][dim2]...[dimN];
```

Esta declaração cria um vetor N-dimensional com:

- $\text{dim}_1 \times \text{dim}_2 \times \cdots \times \text{dim}_N$ elementos totais
- Cada dimensão i numerada de 0 a $\text{dim}_i - 1$

4.2 Exemplos de Vetores Multidimensionais

```
// Matriz 3D: coordenadas espaciais ao longo do tempo
float posicoes[100][3][2];  // 100 tempos, 3 coordenadas (x,y,z),
                            // 2 objetos

// Cubo 3x3x3
int cubo[3][3][3];

// Hipermatriz 4D
float dados[10][20][15][5];
```

O acesso segue o mesmo padrão:

```
cubo[1][2][0] = 7;
float x = posicoes[50][0][1];  // posicao x do objeto 1 no tempo 50
```

Uso Prático: Vetores com mais de 3 dimensões são raramente usados na prática, pois tornam o código difícil de entender e consomem muita memória. Na maioria das aplicações, matrizes bidimensionais são suficientes.

5 Matrizes em Funções

Assim como vetores unidimensionais, matrizes podem ser passadas para funções.

5.1 Declaração de Função com Matriz

Para declarar uma função que recebe uma matriz, devemos especificar o número de colunas (o número de linhas pode ser omitido):

```
void funcao(int matriz[][COLUNAS], int linhas) {
    // codigo
}
```

O número de colunas é obrigatório porque o compilador precisa calcular corretamente os endereços de memória.

5.2 Exemplo: Função para Imprimir Matriz

```
#include <stdio.h>
#define MAX_COL 10

void imprimir_matriz(int mat[][MAX_COL], int linhas, int colunas) {
    int i, j;
    for (i = 0; i < linhas; i++) {
        for (j = 0; j < colunas; j++) {
            printf("%4d ", mat[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int matriz[5][MAX_COL];
    int i, j;

    // Preenche a matriz
    for (i = 0; i < 5; i++)
        for (j = 0; j < 7; j++)
            matriz[i][j] = i * 10 + j;

    // Imprime usando a funcao
    imprimir_matriz(matriz, 5, 7);

    return 0;
}
```

5.3 Passagem por Referência

Importante: assim como vetores unidimensionais, **matrizes são sempre passadas por referência**. Modificações feitas na matriz dentro da função afetam a matriz original.

```
void zerar_matriz(int mat[][10], int linhas, int colunas) {
    int i, j;
    for (i = 0; i < linhas; i++)
        for (j = 0; j < colunas; j++)
            mat[i][j] = 0;
}
```

Esta função modifica a matriz original, zerando todos os seus elementos.

6 Representação na Memória

6.1 Ordem de Armazenamento

Embora pensemos em uma matriz como uma estrutura bidimensional, ela é armazenada linearmente na memória. Em C, as matrizes são armazenadas em *ordem de linhas* (*row-major order*): todos os elementos da primeira linha são armazenados consecutivamente, seguidos por todos os elementos da segunda linha, e assim por diante.

A Figura 2 ilustra como uma matriz 3×4 é armazenada na memória.

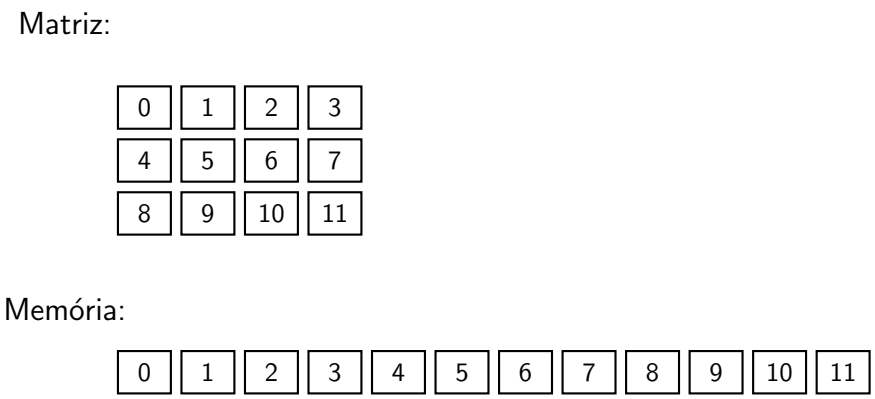

Figura 2: Matriz 3×4 e sua representação linear na memória.

6.2 Cálculo de Endereços

Para uma matriz `mat[m][n]`, o elemento `mat[i][j]` está na posição:

$$\text{posição} = i \times n + j$$

Este é o motivo pelo qual o compilador precisa conhecer o número de colunas ao passar matrizes para funções.

7 Resumo

Pontos-chave desta aula:

- Matrizes são vetores bidimensionais (vetores de vetores)
- Declaração: `tipo nome[linhas][colunas];`
- Indexação começa em [0][0] e vai até [linhas-1][colunas-1]
- Acessos fora dos limites não são verificados pelo compilador
- Matrizes são armazenadas linearmente na memória (ordem de linhas)
- Matrizes são sempre passadas por referência para funções
- Ao passar matriz para função, o número de colunas deve ser especificado
- Use laços aninhados para percorrer todos os elementos
- C permite vetores com qualquer número de dimensões

8 Exercícios

1. Escreva um programa que leia uma matriz de caracteres de dimensões 15×20 e depois leia uma palavra. O programa deverá contar o número de vezes que a palavra aparece na matriz (apenas na horizontal, da esquerda para a direita).

   Exemplo de matriz:

   ```
   OEAIAGBOOL
   IIWAXHHLHN
   PADUCAPNOC
   ZBMOUIZSAS
   OXEZOKOEUA
   QCRMAAPAOH
   DHOMEMTUFO
   HOOAJCMVGM
   ```

```
NMFOANGMAE
JEVJVCCSNM
```

Dica: Para cada linha, percorra as colunas verificando se os caracteres subsequentes formam a palavra procurada.

2. Escreva um programa que lê uma matriz quadrada de dimensões 5×5 e verifica se ela é simétrica. Uma matriz é simétrica se $A[i][j] = A[j][i]$ para todo i e j.

3. Escreva uma função que calcula a soma dos elementos da diagonal principal de uma matriz quadrada $n \times n$.

4. Escreva uma função que encontra o maior elemento de uma matriz e retorna sua posição (linha e coluna) através de ponteiros.

5. Escreva uma função que gera a matriz identidade $n \times n$. A matriz identidade tem 1 na diagonal principal e 0 nas demais posições.

6. Escreva uma função que rotaciona uma matriz quadrada $n \times n$ em 90 graus no sentido horário.

7. Implemente um jogo da velha usando uma matriz 3×3 de caracteres. O programa deve:
   - Ler as jogadas alternadamente de dois jogadores
   - Imprimir o tabuleiro após cada jogada
   - Verificar se há um vencedor ou empate

8. Escreva um programa que verifica se uma matriz 4×4 é uma solução válida para um Sudoku simplificado. Em um Sudoku 4×4:
   - Cada linha deve conter os números de 1 a 4, sem repetição
   - Cada coluna deve conter os números de 1 a 4, sem repetição
   - Cada sub-matriz 2×2 deve conter os números de 1 a 4, sem repetição

9. Escreva um programa que:
   (a) Lê dois números m e n menores que 100
   (b) Lê uma matriz de dimensões $m \times n$
   (c) Imprime a matriz transposta (dimensões $n \times m$)

   A matriz transposta A^T é obtida trocando linhas por colunas: $A^T[i][j] = A[j][i]$.