Capítulo 08: Matrizes

INF1004 e INF 1005 – Programação 1
2012.1

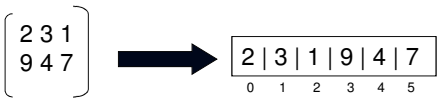
Pontifícia Universidade Católica
Departamento de Informática

Conjuntos Bidimensionais

- Uma matriz representa um conjunto bi-dimensional de valores.
- Similar a variáveis simples e vetores, matrizes devem ser declaradas para que o espaço de memória apropriado seja reservado.
- Como a matriz representa um conjunto bi-dimensional, devemos especificar as duas dimensões na declaração: **o número de linhas e o número de colunas**:
  - float mat[3][4] -> matriz de 3 linhas por 4 colunas (armazenamento de 12 valores do tipo float).
  - O nome da variável mat representa uma referência para o espaço de memória reservado.

Conjuntos Bidimensionais

- Para acessar um elemento da matriz, utilizamos indexação dupla: mat[i][j]
  - Para uma matriz com m linhas e n colunas, os índices usados no acesso aos elementos devem satisfazer: 0<=i<m e 0<=j<n
- Como é armazenada uma matriz em memória:
  - A memória do computador é linear.
  - Uma matriz declarada em C é armazenada na memória linha por linha.



m[i][j] = i *n + j onde n é o número de colunas.
Exemplo: m[1][2] = 1*3 + 2 = 5

Exemplo com Matriz

- Notas obtidas por alunos de uma disciplina.
- Entrada: arquivo com as três notas obtidas por cada aluno.

7.5	8.5	7.8
8.4	9.2	6.8
9.1	10.0	9.5
4.0	5.2	4.6
5.7	3.4	4.3
4.3	6.0	5.8

- Vamos considerar que nosso objetivo seja ler as notas do arquivo e armazená-las na memória do computador para que, posteriormente, seja possível processarmos as notas: calcular a média de cada aluno, a média da disciplina, verificar quantos alunos foram aprovados etc

Exemplo com Matriz

- Precisamos declarar três vetores, um para cada nota:
  - float p1[50];
  - float p2[50];
  - float p3[50];
- Outra alternativa é usar apenas uma estrutura para armazenar todas as notas de todos os alunos: usando matrizes onde teriamos 3 colunas:
  - float notas[50][3];
  - Dessa forma as notas do i-ésimo aluno são representadas por notas[i][0], notas[i][1] e notas[i][2]
- A VANTAGEM É QUE TEMOS TODOS OS DADOS ARMAZENADOS EM UMA ÚNICA ESTRUTURA.

Exemplo com Matriz

Programa 1: ler os dados de um arquivo e armazenar em uma matriz:

```
#include <stdio.h>

int main (void)
{
  int i, j;
  int nalunos;
  float media = 0.0;
  float notas[50][3];
  FILE *f = fopen("notas.txt","r");
  if(f == NULL) {
    printf("Erro na leitura do arquivo.\n");
    return 0;
  }
  /* lê valores do arquivo */
  i = 0;
  while((fscanf(f,"%f %f %f", &notas[i][0],&notas[i][1],
&notas[i][2])==3) && (i<50)){
    i++;
  }
  nalunos = i;
  fclose(f);
  /* calcula média */
  for (i=0; i<nalunos; i++) {
    for (j=0; j<3; j++) {
      media = media + notas[i][j];
    }
  }
  media = media / (3*nalunos);
  /* exibe média calculada */
  printf("Media da disciplina: %.2f\n", media);
  return 0;
}
```

Passagem de Matrizes para Funções

- Essas funções auxiliares recebem como parâmetro uma matriz. PASSAR UMA MATRIZ PARA UMA FUNÇÃO É ANÁLOGO A PASSAR UM VETOR -> Passa-se na verdade uma referência para a matriz.
- Uma diferença importante com relação a vetores é que o parâmetro que representa a matriz deve ter especificado o número de colunas da matriz.
  - O máximo que podemos fazer é codificar uma função que manipula uma matriz com qualquer número linhas, mas o número de colunas deve ser especificado por uma constante no código.
  - Isso é necessário pois o compilador precisa conhecer o número de colunas da matriz para fazer a conta de endereçamento.

Código de função auxiliar que ler valores de um arquivo e armazena em uma matriz:

```
int le_valores (float mat[ ][3])
{
  int i;
  FILE *f = fopen("notas.txt", "r");

  /* lê valores do arquivo */
  i = 0;
  do {
    lidos = fscanf(f,"%f %f %f", &mat[i][0],&mat[i][1],
&mat[i][2]);
    i++;
  } while (i < 50 && lidos == 3);
  fclose(f);
  return i - 1;
}
```

Código de função auxiliar que calcula a média:

```
#define QTD_COL 3
float media (int n, float mat[ ][QTD_COL])
{
  int i, j;
  float soma = 0.0;
  for (i=0; i<n; i++) {
    for (j=0; j<QTD_COL; j++) {
      soma = soma + mat[i][j];
    }
  }
  return soma /(QTD_COL*n);
}
```

Código de função principal:

```
int main (void)
{
  int n;
  float m;
  float notas[50][3];

  n = le_valores(notas);
  m = media(n,notas);

  printf("Media da disciplina: %.2f", m);

  return 0;
}
```

Funções Algébricas

- Em muitas aplicações computacionais, fazemos uso de **matrizes quadradas**, isto é, matrizes em que o número de linhas é igual ao número de colunas.
- Neste caso, como o número de colunas, e consequentemente o número de linhas, tem que ser pré-estabelecido, nossos códigos ficam particularizados para determinada dimensão de matriz.
- Nos códigos, vamos assumir que a dimensão das matrizes é N x N, onde N é uma constante simbólica. Por exemplo, se quisermos que nosso código seja usado para matrizes 4 x 4, fazemos:
  - #define N 4

Verificar se uma matriz é simétrica: retorna 1 se TRUE e 0 se FALSE:

```
int simetrica (double A[ ][N])
{
  int i, j;
  for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
      if (A[i][j] != A[j][i]) {
        return 0;
      }
    }
  }
  return 1;
}
```

Calcular a transposta de uma matriz:

```
void cria_transposta (double A[ ][N],
double T[ ][N])
{
  int i, j;
  for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
      T[j][i] = A[i][j];
    }
  }
}
```

Transpor uma matriz:

```
void transpoe (double A[ ][N])
{
  int i, j;
  for (i=0; i<N; i++) {
    for (j=0; j<i; j++) {
      double t = A[i][j];
      A[i][j] = A[j][i];
      A[j][i] = t;
    }
  }
}
```

Multiplicar uma matriz por um escalar:

```
void mult_matriz_escalar (double A[ ][N], double s, double B[ ][N])
{
  int i, j;
  for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
      B[i][j] = s * A[i][j];
    }
  }
}
```

Multiplicar uma matriz por um vetor:

```
void mult_matriz_vetor (double A[ ][N], double v[ ], double w[ ])
{
  int i, j;
  for (i=0; i<N; i++) {
    w[i] = 0.0;
    for (j=0; j<N; j++) {
      w[i] = w[i] + A[i][j] * v[j];
    }
  }
}
```

Multiplicação de matrizes:

```
void mult_matriz_matriz (double A[ ][N], double B[ ][N], double C[ ][N])
{
  int i, j, k;
  for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
      C[i][j] = 0.0;
      for (k=0; k<N; k++) {
        C[i][j] = C[i][j] + A[i][k] * B[k][j];
      }
    }
  }
}
```

Representação de Tabelas

- Muitas aplicações precisam organizar informações na forma de tabelas, e matrizes são naturalmente estruturas de dados adequadas para representação de tabelas.
- Para exemplicar, vamos considerar uma tabela de um campeonato de futebol. Uma tabela deste tipo e ilustrada a seguir, onde cada linha armazena as informações de um determinado time do campeonato: numero de pontos ganhos (PG), numero de jogos (J), numero de vitorias (V), saldo de gols (SG) e gols próprios (GP).

Time	PG	J	V	SG	GP
Time 0	10	8	3	-4	12
Time 1	17	8	5	10	19
Time 2	10	8	3	-5	11
Time 3	11	8	3	-1	15
Time 4	19	8	6	13	23

gerando a matriz →

10	8	3	-4	12
17	8	5	10	19
10	8	3	-5	11
11	8	3	-1	15
19	8	6	13	23

Representação de Tabelas

- Para o código ficar mais legível, podemos definir constantes simbólicas como:

```
#define PG  0
#define J   1
#define V   2
#define SG  3
#define GC  4
```

- Na verdade, quando temos a definição de várias constantes simbólicas relacionadas, em geral optamos por definir as constantes usando uma **enumeração**:

```
enum {
   PG = 0,
   J,
   V,
   SG,
   GC
};
```

Critérios para o problema dos TIMES

- Um critério usualmente usado para classificação dos times é: número de pontos ganhos, número de vitórias, saldo de gols e, finalmente, número de gols próprios. Assim, o líder do campeonato é o time que tem o maior número de pontos; se dois times tem o mesmo número de pontos, usa-se o maior número de vitórias como critério de desempate; se o número de vitórias também for igual, usa-se o saldo de gols; por fim, usa-se o número de gols próprios.

- Podemos então codificar uma função que recebe como parâmetros o número de times e a matriz representando a tabela do campeonato e retorna o número do time que é líder. Uma possível codificação desta função é mostrada a seguir. A função consiste em um cálculo de máximo de um conjunto, com critérios de desempates.

```
int lider (int n, int t[][5])
{
   int l = 0;  /* assume inicialmente time 0 como líder */
   for (i = 1; i < n; i++) {
      if (t[i][PG] > t[l][PG]) {
         l = i;
      }
      else if (t[i][PG] == t[l][PG]) {
         if (t[i][V] > t[l][V]) {
            l = i;
         }
         else if (t[i][V] == t[l][V]) {
            if (t[i][SG] > t[l][SG]) {
               l = i;
            }
            else if (t[i][SG] == t[l][SG] && t[i][GP] >
 t[l][GP]) {
               l = i;
            }
         }
      }
   }
   return l;
}
```

Atualizando a Tabela do Campeonato

```
void atualiza (int n, int t[][5], int a, int b, int na, int nb)
{
   /* Atualiza pontos ganhos e número de vitórias */
   if (na > nb) {       /* time a foi o vencedor */
      t[a][PG] += 3;
      t[a][V] += 1;
   }
   else if (na < nb) {  /* time b foi vencedor */
      t[b][PG] += 3;
      t[b][V] += 1;
   }
   else {               /* houve empate */
      t[a][PG] += 1;
      t[b][PG] += 1;
   }

   /* Atualiza número de jogos, saldo de gols e gols próprios */
   t[a][J] += 1;
   t[b][J] += 1;
   t[a][SG] += na - nb;
   t[b][SG] += nb - na;
   t[a][GP] += na;
   t[b][GP] += nb;
}
```