Introdução às matrizes

Nesta seção examinaremos resumidamente como utilizar matrizes com a linguagem C.

Da mesma forma que o conceito de *vetor* é útil quando necessita-se de uma grande quantidade de valores associados, todos eles, a um mesmo tipo de dado (como as várias notas de determinado aluno), o conceito de **matriz** é útil quando naturalmente os dados considerados apresentam duas dimensões. Um exemplo é quando precisamos processar dados de um grupo de indivíduos (e.g. alunos), cada um deles com várias medidas (e.g. notas - vide figura 1).

Também vale a pena destacar que, em várias linguagens de programação (como C e *Python*), a implementação de matriz é feita por meio de vetor de vetores. Desse modelo, em ambas pode-se tratar cada linha como um vetor. Por exemplo, se a matriz recebe o nome de `M`, então `M[0]` é sua primeira linha, `M[1]` a segunda linha e assim por diante. Se for necessário pegar um elemento específico da matriz, deve-se usar dois colchetes, como em `M[i][j]`, que devolve o valor da linha *i* na coluna *j* (e.g. `M[0][0]` devolve o valor da primeira posição da matriz e `M[1][0]` devolve o valor do primeiro elemento sua segunda linha).

1. Matrizes: sequência contígua de variáveis

Do ponto de vista computacional a implementação de matrizes segue o princípio dos *vetores*, uma *matriz* ocupa uma porção contígua da memória do computador, servindo para representar o conceito matemático associado. Lembrando que ao representar um *vetor* o acesso ao elemento da posição *i* pode ser feito por meio da sintaxe *vet[i]*, no caso de *matriz* é necessário indicar em qual linha *i* e em qual coluna *j* o elemento está, por exemplo, se a variável tem por nome *Mat*, usaria *Mat[i][j]*.

Este tipo de estrutura de dados é natural quando os dados apresentam 2 atributos. Por exemplo, em uma sala de aula, o professor precisa manter informações dos resultados obtidos pelos alunos em várias atividades, assim pode-se utilizar uma matriz para representar estes dados: as atividades de cada aluno estão em uma única linha da matriz *Mat* (e.g., *Mat[i][0]* é o resultado da atividade *0* para o aluno *i*).

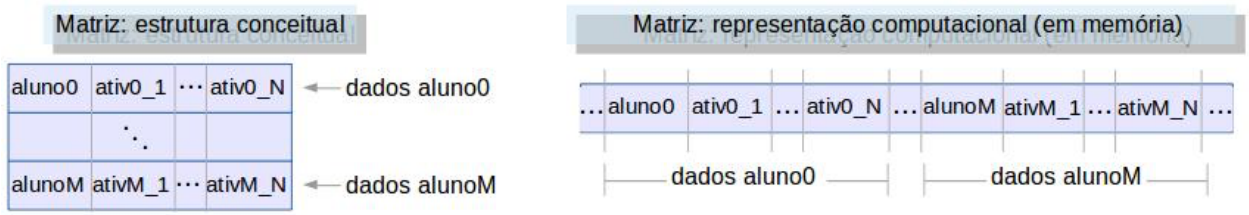

Fig. 1. Ilustração de como uma matriz é representada na "memória" do computador.

Mas como é possível implementar computacionalmente esse tipo de estrutura? Na figura acima está representado uma matriz conceitual, com a ideia de alunos e notas, e à direita como estes dados estão na memória do computador, supondo-se M+1 alunos e N+1 atividades. Se na

representação computacional, o nome da matriz for *Mat*, então os dados correspondentes ao *aluno 0* são: *Mat[0][1]*, *Mat[0][1]* e assim por diante até o *Mat[0][N]*.

1.1. Matrizes: como parâmetro de função

Do mesmo modo que vetores, ao passar uma matriz como parâmetro de uma função, este funcionará como **parâmetro por referência** (ou por **endereço**). Ou seja, é passada uma referência ao *parâmetro efetivo* (em que chamou a função) de modo que qualquer alteração dentro da função, no *parâmetro formal*, significará que o valor na matriz que foi passada como parâmetro (no local que chamou a função e portanto o *parâmetro efetivo*) será alterado.

Na linguagem C as matrizes (e vetores) são passados por referência. O exemplo abaixo ilustra que alterar os dados de uma matriz dentro de uma função implica em alterar a matriz que lhe foi passada (parâmetro efetivo).

Tab. 1. Matrizes passadas via parâmetro para funções (equivale à uma referência à uma matriz "externa").

Função com computa soma de 2 matrizes em C

Exemplo em C

```
#include <stdio.h>
#define NL 20 // constante para maximo de linhas
#define NC 20 // constante para maximo de colunas
// Funcao para gerar nova matriz MA + MB
void somaMat (int MA[][NC], int MB[][NC], int MC[][NC], int nL, int nC) {
  int i, j; // auxiliar, para indexar os matrizes
  for (i=0; i< nL; i++)   // "Ler" nL x nC valores
    for (j=0; j< nC; j++) // inteiros armazenando-os
      MC[i][j] = MA[i][j] + MB[i][j];
  }
// Funcao para entrar matriz nL x nC, linha por linha
void lerMatriz (int mat[][NC], int nL, int nC) {
  int i,j; // declara indices
  for (i=0; i<nL; i++)   // i entre 0 e nL-1
    for (j=0; j<nC; j++) // j entre 0 e nC-1
      scanf("%d", &mat[i][j]);
  }
// Funcao para imprimir matriz nL x nC, linha por linha
void imprimeMatriz (int mat[][NC], int nL, int nC) {
  int i,j; // declara indices
  for (i=0; i< nL; i++) { // i entre 0 e nL-1
    printf("%2d: ", i); // "Imprime" numero desta linha
    for (j=0; j< nC; j++) // j entre 0 e nC-1
      printf("%3d ", mat[i][j]); // em 3 espacos
    printf("\n"); // Mude de linha
    }
  }
int main (void) {
  int i, j; // auxiliar, para indexar os matrizes
  int m, n; // tamanho util das matrizes
  int matA[NL][NC], matB[NL][NC], matC[NL][NC]; // Matriz para inteiros (ate' NL*NC elementos)
  scanf("%d %d", &m, &n);
  printf("Entra matriz A:\n"); lerMatriz(matA, m, n);
  printf("Entra matriz B:\n"); lerMatriz(matB, m, n);
  printf("Gera matriz C:\n"); somaMat(matA, matB, matC, m, n); // parametros efetivos

  printf("Matriz A:\n"); imprimeMatriz(matA, m, n); // 'matA' e' parametro efetivo
  printf("Matriz B:\n"); imprimeMatriz(matB, m, n); // aqui e' 'matB'
```

```
printf("Matriz C:\n"); imprimeMatriz(matC, m, n); // aqui e' 'matC'
return 0;
}
```

1.2. Matrizes: uma linha equivale a um vetor

Uma vez que os elementos em uma linha da matriz estão em posições contíguas da memória, eles podem ser olhados como um vetor, novamente o *contexto* determina o significado dos dados. Assim, se tivermos uma função *soma_vetor* que recebe como parâmetro um vetor (e sua dimensão) e que gera a soma de seus elementos, pode-se fazer a seguinte chamada: *soma_vetor(mat[i],n)*, para qualquer *i* entre *0* e *M* do exemplo acima.

Desse modo, o exemplo abaixo ilustra uma função preparada para somar elementos de um vetor (`soma += vet[i];`) sendo usada para somar as linhas de uma matriz. Então, na execução do laço dentro função, o comando usando o *parâmetro formal soma*, `soma += vet[i]` equivalerá ao código `soma += mat[k][i];` no trecho que invocou a função *soma_vetor*.

Tab. 2. Uma linha de um matriz é na verdade um vetor.

Função de função para somar elementos de vetor sendo usado com linhas de matrizes em C
Exemplo em C

```
#include <stdio.h>
...
int soma_vetor (int vet[], int n) {
  int i, soma = 0;
  for (i=0; i < n; i++)
    soma += vet[i];
  return soma;
  }
int main (void) {
  int mat[NL][NC]; ...
  ...
    print("soma linha %d : %d\n", i, soma_vetor(mat[i], n));
  ...
  return 0;
  }
```

2. Matrizes em C

Como em C deve-se sempre declarar o tipo da variável, no caso de matriz deve-se declarar seu tipo e seu tamanho. No exemplo abaixo ilustramos as declarações e uso de matrizes dos tipos básicos *int*, *char* e *float*.

Tab. 3. Exemplos de tratamento de matrizes em C (com inteiros, caracteres e "flutuantes").

Matrizes em C		
Matriz de inteiros	**Matriz de caracteres**	**Matriz de flutuantes**

`#include <stdio.h>`
`#define NL 20 // constante para maximo de linhas`
`#define NC 20 // constante para maximo de colunas`
`int main (void) {`
	`#include <stdio.h>`
	`#define NL 20 // constante para maximo de linhas`
	`#define NC 20 // constante para maximo de colunas`
	`int main (void) {`
	`  int i, j; // auxiliar, para indexar os matrizes`
		`#include <stdio.h>`
		`#define M 20 // usado para constante`
		`int main (void) {`
		`  int i, j; // auxiliar, para indexar os matrizes`
		`  int nL, nC; // tamanho util dos matrizes`

```
  int i, j; // auxiliar, para
indexar os matrizes
  int nL, nC; // tamanho util das
matrizes
  int  mat[NL][NC]; // Matriz
para inteiros (ate' NL*NC
elementos)
  scanf("%d %d", &nL, &nC);
  for (i=0; i< nL; i++)   //
"Ler" nL x nC valores
    for (j=0; j< nC; j++) //
inteiros armazenando-os
      scanf("%d", &mat[i][j]);
// na matriz
  for (i=0; i< nL; i++) { //
"Imprimir" os nL x nC
    printf("%2d: ", i); //
"Imprime" numero desta linha
    for (j=0; j< nL; j++)   //
valores inteiros, ajustando
      printf("%3d ", mat[i][j]);
// em 3 espacos
    printf("\n"); // Mude de
linha
    }
  return 0;
  }
```

```
  int nL, nC; // tamanho util
dos matrizes
  char matC[NL][NC]; // Matriz
para caracteres
  scanf("%d %d", &nL, &nC);
  for (i=0; i< nL; i++)   //
"Ler" nL x nC valores
    for (j=0; j< nC; j++) //
tipo "char" armazenando-os
      scanf("%c",
&matC[i][j]);  // na matriz
  for (i=0; i< nL; i++) { //
"Imprimir" os nL x nC
    printf("%2d: ", i); //
"Imprime" numero desta linha
    for (j=0; j< nC; j++)
// valores tipo "char",
      printf("%2c",
matC[i][j]); // ajustando em 2
espacos
    printf("\n"); // Mude de
linha
    }
  return 0;
  }
```

```
  float matF[NL][NC]; // Matriz
para caracteres
  scanf("%d %d", &nL, &nC);
  for (i=0; i< nL; i++)      //
"Ler" nL x nC valores
    for (j=0; j< nC; j++)    //
tipo "char" armazenando-os
      scanf("%f",
&matF[i][j]); // na matriz
  for (i=0; i< nL; i++) { //
"Imprimir" os nL x nC
    printf("%2d: ", i); //
"Imprime" numero desta linha
    for (j=0; j< nC; j++)
// valores tipo "float",
      printf("%5.1f",
matF[i][j]); // usando 5 esp. e
1 dec.
    printf("\n"); // Mude de
linha
    }
  return 0;
  }
```

No C padrão NÃO é possível declarar as dimensões da matriz usando a variável que será usada para informar o número efetivo de posições "úteis" na matriz, ou seja, **não** tente fazer algo como: `int m, n; float mat[m][n];`, pode ser que em seu equipamente a versão C utilizada aceite esse construção dinâmica, mas em várias outras **não** funcionará, logo evite. A razão disso é que C, por ser uma linguagem compilada, durante a **fase de compilação** deve-se reservar o espaço máximo a ser usado pelo matriz. Já as variáveis `m` e `n` só serão conhecidas durante a **execução** do programa.

Apesar de algumas implementações de compiladores C permitirem esse tipo de declaração "dinâmica", NÃO usem sob pena de seu programa não funcionar em outros compiladores.

Vejamos o exemplo da função *soma_vetor* sendo chamada com cada linha de uma matriz, com código completo em C.

```
// Exemplo de matriz em Python passando linha em funcao que trabalha com
vetor
#include <stdio.h>

#define MAXL  20 // usar no maximo 20 linhas
#define MAXC  20 // usar no maximo 20 colunas

int soma_vet (int vet[], int n) {
  int soma = 0, i;
  for (i=0; i<n; i++)
    soma += vet[i];
  return soma;
  }

int main (void) {
```

```
  int mat[MAXL][MAXC]; // declara que existira uma matriz (vetor de vetor)
  int i, j, m = 3, n = 4; // dimensoes fixas por simplicidade
  for (i=0; i<m; i++) {
    for (j=0; j<n; j++) {
      mat[i][j] = i*n + j; // como ja' existe espaco reservado atribua "i*m +
j" para a posicao linha i e coluna j
      printf(" %3d", mat[i][j]); // imprime sem mudar de linha
      }
    printf("\n"); // muda de linha
    }

  printf("Imprime somas das linhas\n");
  for (i=0; i<m; i++)
    printf("Linha %d tem soma: %3d\n", i, soma_vet(mat[i], n)); // %3d ajusta
3 casas 'a direita
  return 1;
  }
```

Bem, em resumo é isso.

Leônidas de Oliveira Brandão
http://line.ime.usp.br

Alterações