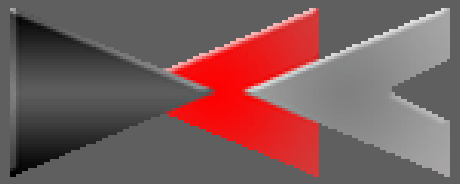
DCC107



Laboratório de Programação II

Tipos Abstratos de Dados Matriz (TAD Matriz) em C

Nesta Aula

- TAD Matriz;
- Exemplos (implementações de TAD Matriz):
  - TADMatriz1 (representação linear);
  - TADMatriz2 (representação por vetor de ponteiros).

Exemplo 1: TADMatriz1

tipo MATRIZ

- domínio: MATRIZ, L, C, VALOR;
- operações:
  - cria_matriz(M, N) → MATRIZ;
  - libera_matriz(MATRIZ);
  - consulta(MATRIZ, L, C) → VALOR;
  - atribui(MATRIZ, L, C, VALOR) → MATRIZ;
- fim-operações;

fim-tipo.

Exemplo 1: TADMatriz1

- Observações:
  - cria_matriz(M, N) → MATRIZ:
    - Cria um TAD matriz bidimensional dado suas dimensões: M → número de linhas e N → número de colunas.
  - consulta(MATRIZ, L, C) → VALOR:
    - Retorna o valor VALOR armazenado na linha L e coluna C da matriz MATRIZ.
  - atribui(MATRIZ, L, C, VALOR) → MATRIZ:
    - Atribui o valor VALOR na linha L e coluna C da matriz MATRIZ.

Exemplo 1: TADMatriz1

- Observações:
  - O TAD Matriz será implementado usando uma representação linear – isto é, vetores. Sendo assim, é necessário uma fórmula para, dados os índices L e C da matriz, calcular o índice K correspondente do vetor. Da teoria, temos:
    - $K = b + t_1 * (L - c_1) + (C - c_2)$.
  - Em C:
    - $c_1 = c_2 = b = 0$ e fazendo $t_1 = M$ (número de colunas = total de elementos por linha).
  - Logo,
    - $K = M * L + C$.

Exemplo 1: TADMatriz1

- Construir dois arquivos para implementar o TAD Matriz:
  - **`matriz.h`** (interface do TAD);
  - **`matriz.c`** (implementação das operações).

Exemplo 1: TADMatriz1

- Arquivo **`matriz.h`**. Contém a definição do tipo e das operações:

```
#ifndef MATRIZ_H_INCLUDED
#define MATRIZ_H_INCLUDED
//Representacao Linear de Matrizes
/* TAD: matriz m por n */

typedef struct matriz Matriz;
/* Funcao cria
** Aloca e retorna uma matriz de dimensao m por n*/
Matriz* cria_matriz (int m, int n);

/* Funcao libera
** Libera a memoria de uma matriz previamente criada.*/
void libera (Matriz* mat);
```

Exemplo 1: TADMatriz1

- Arquivo **`matriz.h`** (continuação):

```
/* Funcao acessa
** Retorna o valor do elemento da linha i e coluna j da matriz
*/
float consulta (Matriz* mat, int i, int j);

/* Funcao atribui
** Atribui o valor dado ao elemento da linha i e coluna j da matriz
*/
void atribui (Matriz* mat, int i, int j, float v);

#endif // MATRIZ_H_INCLUDED
```

Exemplo 1: TADMatriz1

- Arquivo **`matriz.c`**. Contém a `struct` matriz (domínio) e a implementação das operações:

```
#include <stdlib.h> /* malloc, free, exit */
#include <stdio.h> /* printf */
#include "matriz.h"

struct matriz {
    int lin; //numero de linhas da matriz
    int col; //numero de colunas da matriz
    float* v; //vetor que contem os elementos da matriz
};
```

Exemplo 1: TADMatriz1

- Arquivo **`matriz.c`** (continuação):

```
Matriz* cria_matriz(int m, int n)
{
    //aloca memoria para a Matriz
    Matriz* mat = (Matriz*) malloc(sizeof(Matriz));
    if (mat == NULL) {
        printf("Memoria insuficiente!\n");
        exit(1);
    }
    mat->lin = m;
    mat->col = n;
    //aloca memoria para o vetor v
    mat->v = (float*) malloc(m * n * sizeof(float));
    return mat;
}
```

Exemplo 1: TADMatriz1

- Arquivo **`matriz.c`** (continuação):

```
void libera (Matriz* mat){
    //desaloca memoria do vetor v
    free(mat->v);
    //desaloca memoria de Matriz
    free(mat);
}
float consulta(Matriz* mat, int i, int j) {
    int k; // indice do elemento no vetor
    if (i < 0 || i >= mat->lin || j < 0 || j >= mat->col) {
        printf("Acesso invalido!\n");
        exit(1);
    }
    k = i * mat->col + j; //k: indice vetor; i,j: indice matriz
    return mat->v[k];
}
```

Exemplo 1: TADMatriz1

- Arquivo **`matriz.c`** (continuação):

```
void atribui (Matriz* mat, int i, int j, float v)
{
    int k; // indice do elemento no vetor
    if (i < 0 || i >= mat->lin || j < 0 || j >= mat->col)
    {
        printf("Atribuicao invalida!\n");
        exit(1);
    }
    k = i * mat->col + j;
    mat->v[k] = v;
}
```

Exemplo 1: TADMatriz1

- Arquivo **`main.c`**. "Cliente" do TADMatriz:

```c
#include <stdio.h>
#include <stdlib.h>
#include "matriz.h"

int main()
{
    Matriz* mat33;
    int i,j;
    //cria uma matriz, em mat33, com 3 linhas e 3 colunas
    mat33 = cria_matriz(3, 3);
    for(i = 0; i < 3; i++)
        for(j = 0; j < 3; j++)
            atribui(mat33, i, j, (i + 1) * (j + 1));
```

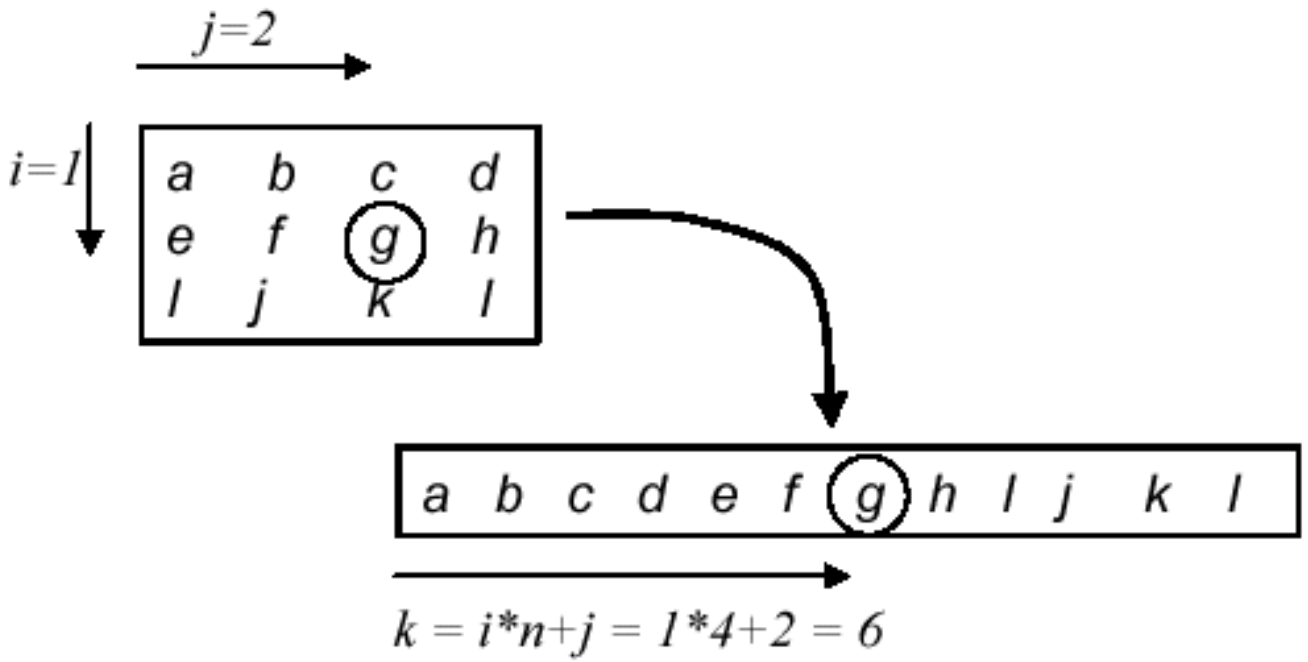
Exemplo 1: TADMatriz1

- Arquivo **`main.c`** (continuação):

```
    for(i = 0; i < 3; i++)
    {
        printf("\n");
        for(j = 0; j < 3; j++)
            printf("%f ", consulta(mat33, i, j));
    }
    libera(mat33);
    return 0;
}
```

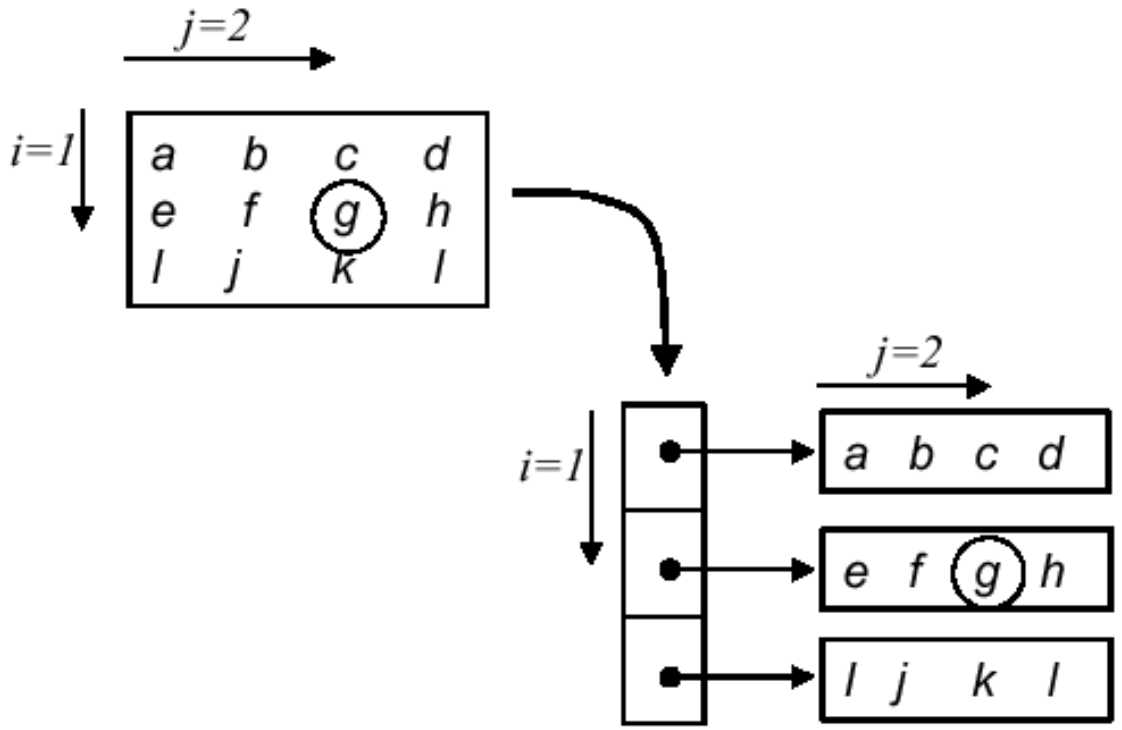
Exemplo 2: TADMatriz2

- Matriz representada por um vetor simples (TADMatriz1: implementado previamente):



Exemplo 2: TADMatriz2

- Matriz representada por um vetor de ponteiros (TADMatriz2), no qual cada elemento armazena o endereço do primeiro elemento de cada linha:



Exemplo 2: TADMatriz2

- Arquivo **`matriz2.h`**. Definição do tipo e das operações (idêntico ao arquivo **`matriz.h`**):

```
#ifndef MATRIZ_H_INCLUDED
#define MATRIZ_H_INCLUDED
//Representacao Linear de Matrizes
/* TAD: matriz m por n */

typedef struct matriz Matriz;
/* Funcao cria
** Aloca e retorna uma matriz de dimensao m por n*/
Matriz* cria_matriz (int m, int n);

/* Funcao libera
** Libera a memoria de uma matriz previamente criada.*/
void libera (Matriz* mat);
```

Exemplo 2: TADMatriz2

- Arquivo **`matriz2.h`** (continuação):

```
/* Funcao acessa
** Retorna o valor do elemento da linha i e coluna j da matriz
*/
float consulta (Matriz* mat, int i, int j);

/* Funcao atribui
** Atribui o valor dado ao elemento da linha i e coluna j da matriz
*/
void atribui (Matriz* mat, int i, int j, float v);

#endif // MATRIZ_H_INCLUDED
```

Exemplo 2: TADMatriz2

- Arquivo **`matriz2.c`**. Contém a `struct` matriz (domínio) e as implementação das operações:

```
#include <stdlib.h> /* malloc, free, exit */
#include <stdio.h> /* printf */
#include "matriz2.h"

struct matriz {
    int lin; //numero de linhas da matriz
    int col; //numero de colunas da matriz
    float** v; //vetor que contem os endereços das linhas da matriz
};
```

Exemplo 2: TADMatriz2

- Arquivo **`matriz2.c`** (continuação):

```
Matriz* cria_matriz(int m, int n)
{
    int i;
    Matriz* mat = (Matriz*) malloc(sizeof(Matriz));
    mat->lin = m;
    mat->col = n;
    mat->v = (float**) malloc(m * sizeof(float*));
    for (i = 0; i < m; i++)
        mat->v[i] = (float*) malloc(n * sizeof(float));
    return mat;
}
```

Exemplo 2: TADMatriz2

- Arquivo **`matriz2.c`** (continuação):

```
void libera (Matriz* mat) {
    int i;
    for (i=0; i<mat->lin; i++)
        free(mat->v[i]);
    free(mat->v);
    free(mat);
}
float consulta(Matriz* mat, int i, int j) {
    if (i<0 || i>=mat->lin || j<0 || j>=mat->col)
    {
        printf("Acesso invalido!\n");
        exit(1);
    }
    return mat->v[i][j];
}
```

Exemplo 2: TADMatriz2

- Arquivo **`matriz2.c`** (continuação):

```
void atribui (Matriz* mat, int i, int j, float v)
{
    if (i<0 || i>=mat->lin || j<0 || j>=mat->col)
    {
        printf("Atribuicao invalida!\n");
        exit(1);
    }
    mat->v[i][j] = v;
}
```

Exemplo 2: TADMatriz2

- Arquivo **`main.c`**. "Cliente" do TADMatriz2:

```c
#include <stdio.h>
#include <stdlib.h>
#include "matriz2.h"

int main()
{
    Matriz* mat33;
    int i,j;
    //cria uma matriz, em mat33, com 3 linhas e 3 colunas
    mat33 = cria_matriz(3, 3);
    for(i = 0; i < 3; i++)
        for(j = 0; j < 3; j++)
            atribui(mat33, i, j, (i + 1) * (j + 1));
```

Exemplo 2: TADMatriz2

- Arquivo **`main.c`** (continuação):

```
    for(i = 0; i < 3; i++)
    {
        printf("\n");
        for(j = 0; j < 3; j++)
            printf("%f " , consulta(mat33, i, j));
    }
    libera(mat33);
    return 0;
}
```

Exemplo 2: TADMatriz2

- Observações:
  - Observe que, independentemente da representação da matriz, a função **`main.c`** continua a mesma. Isto acontece pois **`main.c`** não tem acesso aos dados internos do TAD.
  - Os arquivos **`.h`** também continuam iguais já que o acesso ao TAD é igual.
  - Implementam a idéia de proteção e invisibilidade.

Exercícios

1. Implemente uma função que, dada uma matriz, crie dinamicamente a matriz transposta correspondente, fazendo uso das operações básicas discutidas acima.
2. Implemente uma função que determine se uma matriz é ou não simétrica quadrada, também fazendo uso das operações básicas.
3. Implemente um procedimento para imprimir a 7ª coluna de uma matriz.

Exercícios

4. Implemente um procedimento para ler índice de linha, índice de coluna e um valor real e alterar a posição correspondente de M.
5. Implemente uma função que determine o maior valor da diagonal secundária de uma matriz.