Tipo Abstrato de Dados (Matriz)

Desenvolver um tipo abstrato de dados para armazenar e manipular matrizes de diferentes dimensões.

Vamos implementar o TAD considerando duas formas de organização dos dados. A primeira forma será desenvolvida no arquivo `TAD-matriz1.h` e a segunda forma no arquivo `TAD-matriz2.h`

DADOS

Os dados da matriz serão organizados em uma *struct* denominada `Matriz`.

Estrutura 1

```
typedef struct{
    int** mat;
    int linhas;
    int colunas;
} Matriz;
```

Estrutura 2

```
typedef struct{
    int** mat;
    int linhas;
    int colunas;
}Matriz;
```

OPERAÇÕES

Todas as operações previstas para a matriz serão mapeadas para funções. Todas as funções serão desenvolvidas especificamente para manipular a estrutura definida `Matriz`. Portanto, cada função terá duas implementações, uma no arquivo `TAD-matriz1.h` e outra no arquivo `TAD-matriz2.h`.

Importante: *Você deve ser capaz de validar se sua função funciona ou não. Para isso, crie pequenos casos de teste para validar as funções.*

Para facilitar a identificação, utilizaremos o prefixo `"matriz_"` para cada função.

Criar a matriz

```
Matriz* matriz_cria(int linhas, int colunas);
```

Destruir a matriz

```
void matriz_destroi(Matriz *m);
```

Recuperar um elemento da matriz

```
int matriz_acessa1(Matriz *m, int lin, int col);

void matriz_acessa2(Matriz *m, int lin, int col, int *end);
```

Atribuir um elemento da matriz

```
void matriz_atribui(Matriz *m, int lin, int col, int valor);
```

Imprimir a matriz

```
void matriz_imprime(Matriz *m);
```

Recuperar a quantidade de linhas e colunas da matriz

```
int matriz_linhas(Matriz* m);

int matriz_colunas(Matriz* m);
```

Prover uma forma de aplicar uma alteração à todos os elementos da matriz.

Essa funcionalidade pode ser obtida por meio de uma função que recebe por parâmetro o ponteiro da função que realizará a alteração em cada elemento. Essa funcionalidade já foi discutida em sala de aula.

Podemos optar por um dos dois protótipos a seguir:

```
void matriz_map1(Matriz* m, void (*funcao)(int*));
void matriz_map2(Matriz* m, int (*funcao)(int));
```

A função recebe a matriz e o ponteiro da função que será invocada para cada elemento.

Obter um vetor `(int*)` a partir de uma determinada LINHA da matriz.

Essa funcionalidade não deve alterar os dados originais da matriz. Um vetor deve ser criado com as cópias dos valores da linha.

Para essa funcionalidade, pense em um protótipo que satisfaça a necessidade de que utilizará a função. A função deverá devolver 2 infomações:

- o endereço do vetor.
- a quantidade de elementos.

Obter um vetor `(int*)` a partir de uma determinada COLUNA da matriz.

Essa funcionalidade não deve alterar os dados originais da matriz. Um vetor deve ser criado com as cópias dos valores da coluna.

Para essa funcionalidade, pense em um protótipo que satisfaça a necessidade de que utilizará a função. A função deverá devolver 2 infomações:

- o endereço do vetor.
- a quantidade de elementos.

Multiplicar duas matrizes.

Para essa funcionalidade a função deve receber duas matrizes e devolver uma nova matriz com o resultado da multiplicacao.

Lembrando que a multiplicação de matriz somente é possível se o número de linhas em uma matriz for igual ao número de colunas da outra matriz.

```
Matriz* matriz_multiplica(Matriz* a, Matriz* b);
```

Verificar se a matriz é quadrada.

Uma matriz é quadrada quando o número de colunas é igual ao número de linhas.

```
int matriz_ehQuadrada(Matriz* m);
```

A função deve devolver:

- `1` caso a matriz `m` seja quadrada e
- `0`, caso contrário.

Verificar se a matriz é diagonal.

Uma **matriz diagonal** é uma matriz quadrada em que todos os elementos que não pertencem a diagonal principal são nulos.

Veja um exemplo aqui.

```
int matriz_ehDiagonal(Matriz* m);
```

Verificar se a matriz é identidade.

Uma **matriz identidade** é uma matriz quadrada em que todos os elementos que não pertencem a diagonal principal são nulos e os elementos da diagonal principal são 1

Veja um exemplo aqui.

```
int matriz_ehIdentidade(Matriz* m);
```

Obter uma cópia da transposta da matriz

Uma matriz transposta é uma matriz resultante da troca ordenadamente de linhas pelas colunas de outra matriz.

Veja um exemplo aqui

```
Matriz* matriz_transposta(Matriz* m);
```

Obter uma cópia da oposta da matriz

Uma matriz transposta é uma matriz que é obtida trocando os sinais dos elementos da matriz.

Veja um exemplo aqui

```
Matriz* matriz_oposta(Matriz* m);
```