AULA e05
Algoritmos e Estruturas de Dados I

Matrizes Especiais

Luciano Antonio Digiampietri

Matriz

Uma matriz bidimensional é um conjunto de elementos (ou tabela) composta por *m* linhas e *n* colunas.

Matriz

Uma **matriz bidimensional** é um conjunto de elementos (ou tabela) composta por ***m*** **linhas** e ***n*** **colunas**.

- Em computação utilizamos matrizes para representar diferentes tipos de dados (dados numéricos, imagens, etc.)

Criado uma matriz em C

Criado uma matriz em C

```
int matriz1[2][3];
```

Criado uma matriz em C

```
int matriz1[2][3];

matriz1[0][0] = 1;
matriz1[0][1] = 2;
matriz1[0][2] = 3;
matriz1[1][0] = 4;
matriz1[1][1] = 5;
matriz1[1][2] = 6;
```

Criado uma matriz em C

```
int matriz1[2][3];

matriz1[0][0] = 1;
matriz1[0][1] = 2;
matriz1[0][2] = 3;
matriz1[1][0] = 4;
matriz1[1][1] = 5;
matriz1[1][2] = 6;
```

Como entendemos a matriz:

1	2	3
4	5	6

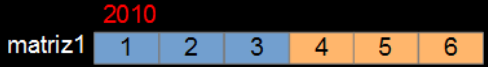
Criado uma matriz em C

```
int matriz1[2][3];

matriz1[0][0] = 1;
matriz1[0][1] = 2;
matriz1[0][2] = 3;
matriz1[1][0] = 4;
matriz1[1][1] = 5;
matriz1[1][2] = 6;
```

Como entendemos a matriz:

1	2	3
4	5	6

Como ela é representada na memória:



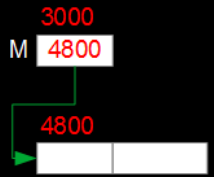
Maneira alternativa

Maneira alternativa

```
int** M
```

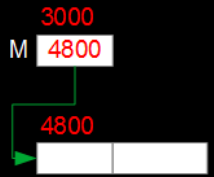
Maneira alternativa

```
int** M = (int**) malloc(sizeof(int*)*2);
```



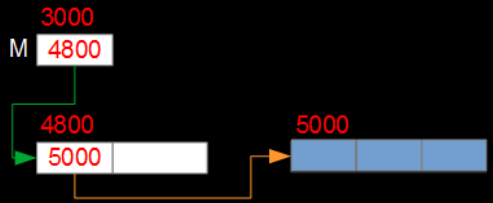
Maneira alternativa

```
int** M = (int**) malloc(sizeof(int*)*2);

for (i=0; i<2; i++){
    M[i] = (int*) malloc(sizeof(int)*3);
}
```



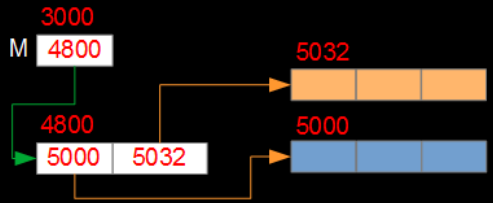
Maneira alternativa

```
int** M = (int**) malloc(sizeof(int*)*2);

for (i=0; i<2; i++){
    M[i] = (int*) malloc(sizeof(int)*3);
}
```



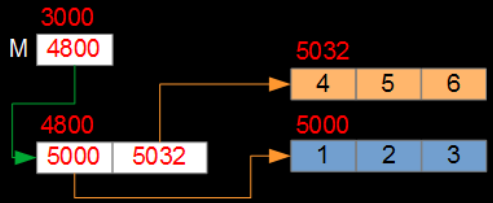
Maneira alternativa

```
int** M = (int**) malloc(sizeof(int*)*2);

for (i=0; i<2; i++){
    M[i] = (int*) malloc(sizeof(int)*3);
}
```



Maneira alternativa

```
int** M = (int**) malloc(sizeof(int*)*2);

for (i=0; i<2; i++){
    M[i] = (int*) malloc(sizeof(int)*3);
}

M[0][0] = 1;
M[0][1] = 2;
M[0][2] = 3;
M[1][0] = 4;
M[1][1] = 5;
M[1][2] = 6;
```



Matrizes especiais

Matrizes especiais

Algumas matrizes possuem características específicas e os algoritmos podem considerar essas características em suas implementações

Matrizes especiais

Algumas matrizes possuem características específicas e os algoritmos podem considerar essas características em suas implementações

- Há matrizes quadradas

Matrizes especiais

Algumas matrizes possuem características específicas e os algoritmos podem considerar essas características em suas implementações

- Há matrizes quadradas
- Há matrizes simétricas

Matrizes especiais

Algumas matrizes possuem características específicas e os algoritmos podem considerar essas características em suas implementações

- Há matrizes quadradas
- Há matrizes simétricas
- Há matrizes esparsas

Matrizes especiais

Algumas matrizes possuem características específicas e os algoritmos podem considerar essas características em suas implementações

- Há matrizes quadradas
- Há matrizes simétricas
- Há matrizes esparsas
- ...

Matrizes Triangulares Inferiores

Todos os elementos acima da diagonal principal são nulos.

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

Matrizes Triangulares Inferiores

Todos os elementos acima da diagonal principal são nulos.

- Só alocaremos memória para os demais elementos.

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

Matrizes Triangulares Inferiores

Representação

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

Matrizes Triangulares Inferiores

Representação

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Matrizes Triangulares Inferiores

Representação

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

0	A_{12}	A_{13}	A_{14}	A_{15}
1	2	A_{23}	A_{24}	A_{25}
3	4	5	A_{34}	A_{35}
6	7	8	9	A_{45}
10	11	12	13	14

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Matrizes Triangulares Inferiores

Representação

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

0	A_{12}	A_{13}	A_{14}	A_{15}
1	2	A_{23}	A_{24}	A_{25}
3	4	5	A_{34}	A_{35}
6	7	8	9	A_{45}
10	11	12	13	14

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A_{11}	A_{21}	A_{22}	A_{31}	A_{32}	A_{33}	A_{41}	A_{42}	A_{43}	A_{44}	A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

Matrizes Triangulares Inferiores Implementação

Matrizes Triangulares Inferiores Implementação

Nossa estrutura manterá um **arranjo** no qual os valores serão armazenados;

Matrizes Triangulares Inferiores Implementação

Nossa estrutura manterá um **arranjo** no qual os valores serão armazenados;

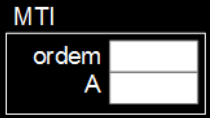
Teremos uma variável chamada **ordem** que conterá a ordem da matriz (número de linhas e de colunas).

Modelagem

```
#include <stdio.h>
#include <malloc.h>
#define true 1
#define false 0
typedef int bool;

typedef struct {
  int ordem;
  int* A;
} MTI;
```

Modelagem

```c
#include <stdio.h>
#include <malloc.h>
#define true 1
#define false 0
typedef int bool;

typedef struct {
  int ordem;
  int* A;
} MTI;
```



Funções de gerenciamento

Implementaremos funções para:

- **Inicializar a matriz (“new** $matriz[m][n]$**”)**
- **Atribuir um valor (**$matriz[x][y] = valor$**)**
- **Acessar valor (**$valor = matriz[x][y]$**)**

Inicialização

Para inicializar nossa matriz, nós precisamos:

Inicialização

Para inicializar nossa matriz, nós precisamos:

- **Atualizar o valor do campo *ordem* (a partir da ordem da matriz passada pelo usuário).**

Inicialização

Para inicializar nossa matriz, nós precisamos:

- **Atualizar o valor do campo *ordem* (a partir da ordem da matriz passada pelo usuário).**
- **Alocar memória para o arranjo com o tamanho correto (a partir da ordem da matriz).**

Inicialização

Para inicializar nossa matriz, nós precisamos:

- Atualizar o valor do campo *ordem* (a partir da ordem da matriz passada pelo usuário).
- Alocar memória para o arranjo com o tamanho correto (a partir da ordem da matriz).
- Precisamos também colocar zeros em cada posição do arranjo.

Qual será o tamanho do arranjo?

Qual será o tamanho do arranjo?

0	A_{12}	A_{13}	A_{14}	A_{15}
1	2	A_{23}	A_{24}	A_{25}
3	4	5	A_{34}	A_{35}
6	7	8	9	A_{45}
10	11	12	13	14

Qual será o tamanho do arranjo?

0	A_{12}	A_{13}	A_{14}	A_{15}	1
1	2	A_{23}	A_{24}	A_{25}	2
3	4	5	A_{34}	A_{35}	3
6	7	8	9	A_{45}	4
10	11	12	13	14	5

Qual será o tamanho do arranjo?

0	A_{12}	A_{13}	A_{14}	A_{15}	1
1	2	A_{23}	A_{24}	A_{25}	2
3	4	5	A_{34}	A_{35}	3
6	7	8	9	A_{45}	4
10	11	12	13	14	5

Neste exemplo o arranjo terá 15 posições:

1 + 2 + 3 + 4 + 5

Qual será o tamanho do arranjo?

0	A_{12}	A_{13}	A_{14}	A_{15}
1	2	A_{23}	A_{24}	A_{25}
3	4	5	A_{34}	A_{35}
6	7	8	9	A_{45}
10	11	12	13	14

1 2 3 4 5

Neste exemplo o arranjo terá 15 posições:

1 + 2 + 3 + 4 + 5

Se a ordem da matriz for n, teremos:

1 + 2 + 3 + 4 ... + n -1 + n

Qual será o tamanho do arranjo?

0	A_{12}	A_{13}	A_{14}	A_{15}
1	2	A_{23}	A_{24}	A_{25}
3	4	5	A_{34}	A_{35}
6	7	8	9	A_{45}
10	11	12	13	14

1
2
3
4
5

Neste exemplo o arranjo terá 15 posições:

1 + 2 + 3 + 4 + 5

Se a ordem da matriz for n, teremos:

1 + 2 + 3 + 4 ... + n -1 + n

Isto equivale à soma dos termos de uma progressa aritmética Soma_PA:

Soma_PA = (Primeiro+Último)*(Número de Elementos)/2

Qual será o tamanho do arranjo?

0	A_{12}	A_{13}	A_{14}	A_{15}
1	2	A_{23}	A_{24}	A_{25}
3	4	5	A_{34}	A_{35}
6	7	8	9	A_{45}
10	11	12	13	14

1
2
3
4
5

Neste exemplo o arranjo terá 15 posições:
1 + 2 + 3 + 4 + 5

Se a ordem da matriz for n, teremos:
1 + 2 + 3 + 4 ... + n -1 + n
Isto equivale à soma dos termos de
uma progressa aritmética Soma_PA:
Soma_PA = (Primeiro+Último)*(Número de Elementos)/2

Para uma matriz de ordem n:
Soma_PA = (1 + n)*(n)/2 = (n + n*n)/2

Inicialização

```
void inicializarMatriz(int n, MTI* matriz){




}
```

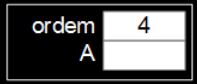
Inicialização

```
void inicializarMatriz(int n, MTI* matriz){
  matriz->ordem = n;



}
```

Inicialização

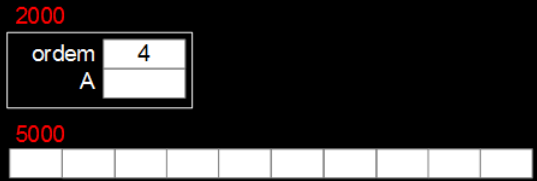
```
void inicializarMatriz(int n, MTI* matriz){
  matriz->ordem = n;
  int numeroDeElementos = (n+n*n)/2;



}
```

2000



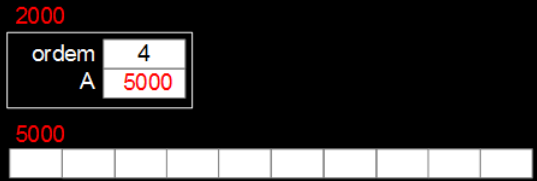
Inicialização

```
void inicializarMatriz(int n, MTI* matriz){
  matriz->ordem = n;
  int numeroDeElementos = (n+n*n)/2;
  matriz->A = (int*) malloc(sizeof(int)*numeroDeElementos);


}
```



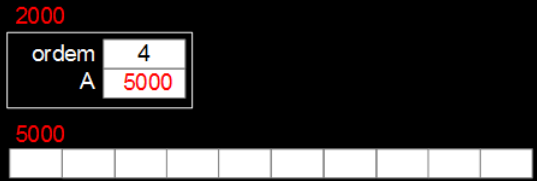
Inicialização

```
void inicializarMatriz(int n, MTI* matriz){
  matriz->ordem = n;
  int numeroDeElementos = (n+n*n)/2;
  matriz->A = (int*) malloc(sizeof(int)*numeroDeElementos);


}
```



Inicialização

```
void inicializarMatriz(int n, MTI* matriz){
  matriz->ordem = n;
  int numeroDeElementos = (n+n*n)/2;
  matriz->A = (int*) malloc(sizeof(int)*numeroDeElementos);
  int i;
  for (i=0;i<numeroDeElementos;i++) matriz->A[i]=0;
}
```



Inicialização

```
void inicializarMatriz(int n, MTI* matriz){
  matriz->ordem = n;
  int numeroDeElementos = (n+n*n)/2;
  matriz->A = (int*) malloc(sizeof(int)*numeroDeElementos);
  int i;
  for (i=0;i<numeroDeElementos;i++) matriz->A[i]=0;
}
```

2000

ordem	4
A	5000

5000

0	0	0	0	0	0	0	0	0	0

Atribuir valor

O usuário nos passa: o endereço da matriz, a linha, a coluna e o valor a ser colocado na respectiva posição da matriz:

Atribuir valor

O usuário nos passa: o endereço da matriz, a linha, a coluna e o valor a ser colocado na respectiva posição da matriz:

- **Se for um endereço inválido retornaremos *false*.**

Atribuir valor

O usuário nos passa: o endereço da matriz, a linha, a coluna e o valor a ser colocado na respectiva posição da matriz:

- **Se for um endereço inválido retornaremos *false*.**
- **Caso contrário, realizaremos a atribuição na posição correspondente do arranjo e retornaremos *true*.**

Qual a posição correta?

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

0	A_{12}	A_{13}	A_{14}	A_{15}
1	2	A_{23}	A_{24}	A_{25}
3	4	5	A_{34}	A_{35}
6	7	8	9	A_{45}
10	11	12	13	14

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A_{11}	A_{21}	A_{22}	A_{31}	A_{32}	A_{33}	A_{41}	A_{42}	A_{43}	A_{44}	A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

A atribuição ocorrerá na linha lin e coluna col.

Qual a posição correta?

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

0	A_{12}	A_{13}	A_{14}	A_{15}
1	2	A_{23}	A_{24}	A_{25}
3	4	5	A_{34}	A_{35}
6	7	8	9	A_{45}
10	11	12	13	14

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A_{11}	A_{21}	A_{22}	A_{31}	A_{32}	A_{33}	A_{41}	A_{42}	A_{43}	A_{44}	A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

A atribuição ocorrerá na linha lin e coluna col.

Qual a posição correta?

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

0	A_{12}	A_{13}	A_{14}	A_{15}
1	2	A_{23}	A_{24}	A_{25}
3	4	5	A_{34}	A_{35}
6	7	8	9	A_{45}
10	11	12	13	14

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

A_{11}	A_{21}	A_{22}	A_{31}	A_{32}	A_{33}	A_{41}	A_{42}	A_{43}	A_{44}	A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

A atribuição ocorrerá na linha lin e coluna col.

Qual a posição correta?

A atribuição ocorrerá na linha `lin` e coluna `col`.

```
int retornaPosicao(int lin, int col){
  return                  +        ;
}
```

Qual a posição correta?

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

0	A_{12}	A_{13}	A_{14}	A_{15}
1	2	A_{23}	A_{24}	A_{25}
3	4	5	A_{34}	A_{35}
6	7	8	9	A_{45}
10	11	12	13	14

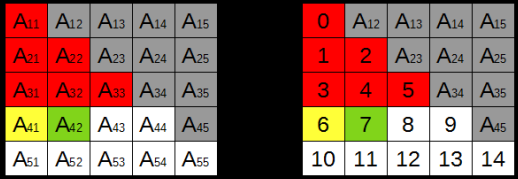
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A_{11}	A_{21}	A_{22}	A_{31}	A_{32}	A_{33}	A_{41}	A_{42}	A_{43}	A_{44}	A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

A atribuição ocorrerá na linha lin e coluna col.

Soma_PA: de 1 até lin - 1
Soma_PA: (1 + lin-1)*(lin-1)/2
Soma_PA: (lin)*(lin-1)/2
Soma_PA: (lin*lin - lin)/2

```
int retornaPosicao(int lin, int col){
  return (lin*lin-lin)/2 +        ;
}
```

Qual a posição correta?



A atribuição ocorrerá na linha `lin` e coluna `col`.

Soma_PA: de 1 até lin - 1
Soma_PA: (1 + lin-1)*(lin-1)/2
Soma_PA: (lin)*(lin-1)/2
Soma_PA: (lin*lin - lin)/2

Precisaremos somar *col* na posição

```
int retornaPosicao(int lin, int col){
  return (lin*lin-lin)/2 + col    ;
}
```

Qual a posição correta?

A atribuição ocorrerá na linha `lin` e coluna `col`.

Soma_PA: de 1 até lin - 1
Soma_PA: (1 + lin-1)*(lin-1)/2
Soma_PA: (lin)*(lin-1)/2
Soma_PA: (lin*lin - lin)/2

Precisaremos somar *col* na posição

Precisaremos descontar 1,
pois arranjos começam na posição zero

```
int retornaPosicao(int lin, int col){
  return (lin*lin-lin)/2 + col - 1;
}
```

Atribuir valor

```
bool AtribuiMatriz(int lin, int col, int val, MTI* mat) {



}
```

Atribuir valor

```
bool AtribuiMatriz(int lin, int col, int val, MTI* mat) {
  if ((lin<1) || (lin>mat->ordem) || (col<1) || (col>mat->ordem) || (lin<col))
      return false;



}
```

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

Atribuir valor

```
bool AtribuiMatriz(int lin, int col, int val, MTI* mat) {
  if ((lin<1) || (lin>mat->ordem) || (col<1) || (col>mat->ordem) || (lin<col))
      return false;
  int posicao = retornaPosicao(lin,col);


}
```

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

Atribuir valor

```
bool AtribuiMatriz(int lin, int col, int val, MTI* mat) {
  if ((lin<1) || (lin>mat->ordem) || (col<1) || (col>mat->ordem) || (lin<col))
      return false;
  int posicao = retornaPosicao(lin,col);
  mat->A[posicao]=val;
  return true;
}
```

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

Acessar valor

O usuário nos passa: o endereço da matriz, a linha e a coluna e a função deve retornar o valor da respectiva posição:

Acessar valor

O usuário nos passa: o endereço da matriz, a linha e a coluna e a função deve retornar o valor da respectiva posição:

- Se as coordenadas estiverem acima da diagonal principal
  então retornar zero;

Acessar valor

O usuário nos passa: o endereço da matriz, a linha e a coluna e a função deve retornar o valor da respectiva posição:

Se as coordenadas estiverem acima da diagonal principal então retornar zero;
Caso contrário, retornar o valor da posição correspondente no arranjo.

Acessar valor

```
int ValorMatriz(int lin, int col, MTI* mat){



}
```

Acessar valor

```
int ValorMatriz(int lin, int col, MTI* mat){
  if ((lin<1) || (lin>mat->ordem) || (col<1) || (col>mat->ordem))
      return -1;


}
```

Acessar valor

```
int ValorMatriz(int lin, int col, MTI* mat){
  if ((lin<1) || (lin>mat->ordem) || (col<1) || (col>mat->ordem))
      return -1;
  if (lin<col) return 0;

}
```

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

Acessar valor

```
int ValorMatriz(int lin, int col, MTI* mat){
  if ((lin<1) || (lin>mat->ordem) || (col<1) || (col>mat->ordem))
      return -1;
  if (lin<col) return 0;
  return mat->A[retornaPosicao(lin,col)];
}
```

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}

AULA e05
Algoritmos e Estruturas de Dados I

Matrizes Especiais

Luciano Antonio Digiampietri