

# Programozás 2

Két- és többdimenziós tömbök

# Tömbök

## Egydimenziós tömb = Vektor

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
<b>2</b>	<b>18</b>	<b>6</b>	<b>40</b>	<b>19</b>	<b>20</b>	<b>26</b>	<b>25</b>	<b>24</b>	<b>7</b>

Az egydimenziós tömbök jól használhatók egyetlen értékhalmoz kezelésére, azonban sok problémánál szükség van összetettebb adatszerkezetekre.

## Kétdimenziós tömb = Mátrix

- ▶ Sorokból és oszlopokból álló adatszerkezet.
- ▶ Az elemek sor- és oszlopindexek segítségével érhetők el.
- ▶ A C nyelvben az indexelés mindig 0-tól kezdődik!

	0	1	2	3	4
0	m[0][0]	m[0][1]	m[0][2]	m[0][3]	m[0][4]
1	m[1][0]	m[1][1]	m[1][2]	m[1][3]	m[1][4]
2	m[2][0]	m[2][1]	m[2][2]	m[2][3]	m[2][4]

	0	1	2	3	4
0	<b>18</b>	<b>7</b>	<b>41</b>	<b>30</b>	<b>12</b>
1	<b>46</b>	<b>33</b>	<b>20</b>	<b>2</b>	<b>37</b>
2	<b>9</b>	<b>13</b>	<b>32</b>	<b>44</b>	<b>25</b>

# Hogyan van tárolva a mátrix a memóriában?

A mátrix a memóriában **lineáris formában** tárolódik, mivel a memória **egydimenziós címtérként** működik. A **C nyelvben** a tömbök **sorfolytonos tárolás (row-major order)** szerint kerülnek eltárolásra, azaz a mátrix egyes sorai egymás után helyezkednek el a memóriában.

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

Ha pl. a mátrix [0][0] elemének memóriacíme 1000, akkor a tárolás így néz ki (feltételezve, hogy egy int mérete 4 bájtt):

1000	1004	1008	1012	1016	1020	1024	1028	1032	1036	1040	1044
1	2	3	4	5	6	7	8	9	10	11	12

Hogyan számolhatjuk ki a mátrix [i][j] elemének memóriacímét?

$$\text{cím} = \text{báziscím} + \text{elemméret} * (i * \text{oszlopok\_száma} + j)$$

Ha a báziscím=1000, elemméret=4 (int mérete), oszlopok\_száma=4, akkor pl. a mátrix [1][2] elemének memóriacíme (i=1, j=2):

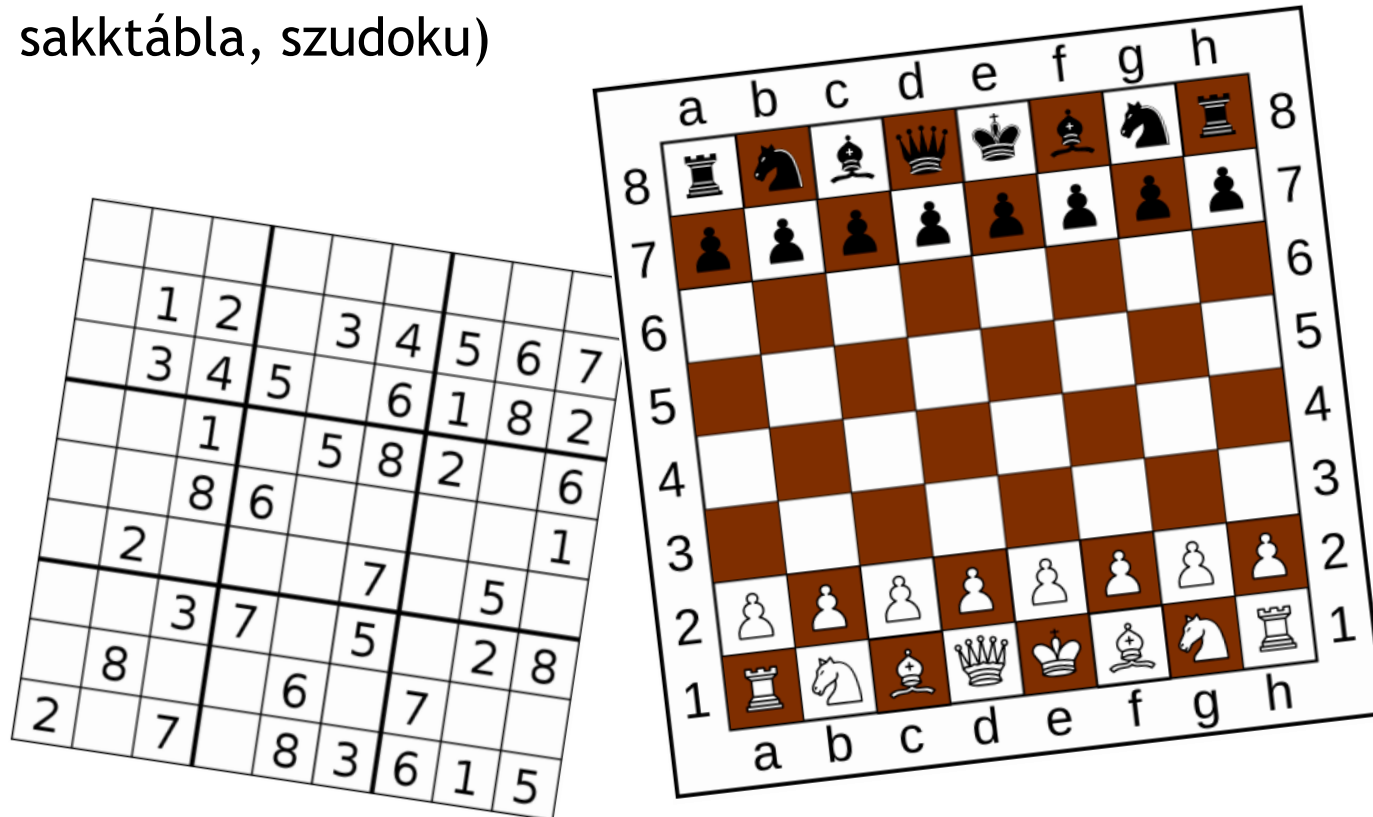
$$\text{cím} = 1000 + 4 * (i * 4 + j) = 1000 + 4 * (1 * 4 + 2) = 1024$$

# Hol használhatók fel a mátrixok?

- ▶ Táblázatok, matematikai mátrixok reprezentálása
- ▶ Képfeldolgozás: szürkeárnyaltos kép tárolására  
(minden pixel értéke 0-255 közötti egész szám a mátrixban)
- ▶ Játéktáblák tárolására  
(pl. sakktábla, sudoku)
- ▶ ...

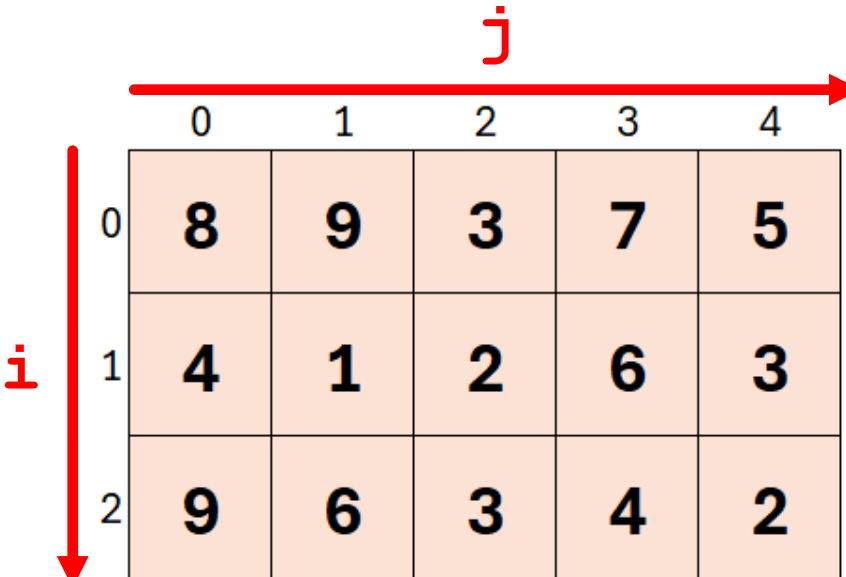
$$M = \begin{bmatrix} 27 & 9 & 14 \\ 17 & 22 & 8 \end{bmatrix}$$

	0	1	2
0	27	9	14
1	17	22	8



## Mátrix deklarálása és inicializálása:

```
int matrix[3][5] = {  
    { 8, 9, 3, 7, 5 },  
    { 4, 1, 2, 6, 3 },  
    { 9, 6, 3, 4, 2 }  
};
```

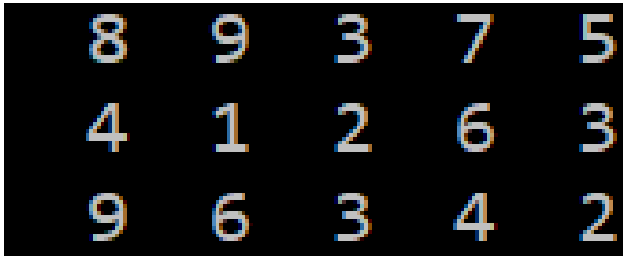


A 3x5 matrix is shown with row index *i* (0, 1, 2) and column index *j* (0, 1, 2, 3, 4). The matrix contains the following values:

	0	1	2	3	4
0	8	9	3	7	5
1	4	1	2	6	3
2	9	6	3	4	2

## Mátrix kiírása a képernyőre:

```
for (int i=0; i<3; i++) {  
    for (int j=0; j<5; j++) {  
        printf("%3d", matrix[i][j]);  
    }  
    printf("\n");  
}
```



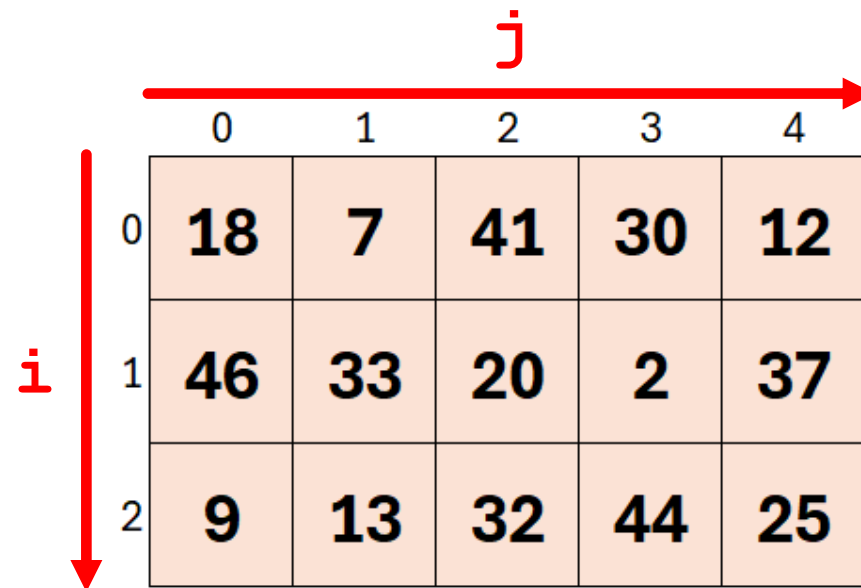
The output of the code is a 3x5 matrix of numbers displayed on a black background:

8	9	3	7	5
4	1	2	6	3
9	6	3	4	2

## Mátrix feltöltése véletlen számokkal és kiírása a képernyőre:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    const int M=3, N=5;
    int m[M][N];
    // matrix feltoltese veletlen számokkal es kiirasa
    srand(time(NULL));
    for (int i=0; i<M; i++) {
        for (int j=0; j<N; j++) {
            m[i][j] = rand() % 50 + 1;
            printf("%3d", m[i][j]);
        }
        printf("\n");
    }
}
```



A diagram illustrating a 3x5 matrix. The rows are indexed by 'i' (0, 1, 2) and the columns by 'j' (0, 1, 2, 3, 4). The matrix contains the following values:

	0	1	2	3	4
0	18	7	41	30	12
1	46	33	20	2	37
2	9	13	32	44	25

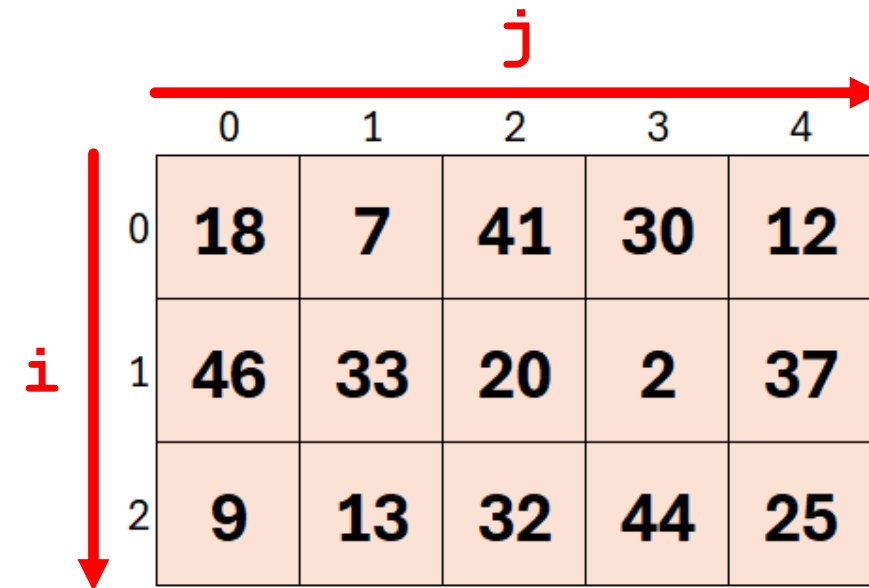
# Műveletek kétdimenziós tömbökön

## Mátrix elemeinek összege és átlaga

Feladat: Határozzuk meg a mátrix elemeinek összegét és átlagát!

```
int osszeg=0;
double atlag;
for (int i=0; i<M; i++) {
    for (int j=0; j<N; j++) {
        osszeg = osszeg + m[i][j];
    }
}
```

```
atlag = (double)osszeg / (M*N);
printf("Osszeg: %d\nAtlag:%lf\n", osszeg, atlag);
```



A 3x5 matrix is shown with row index *i* (0, 1, 2) and column index *j* (0, 1, 2, 3, 4). The matrix elements are:

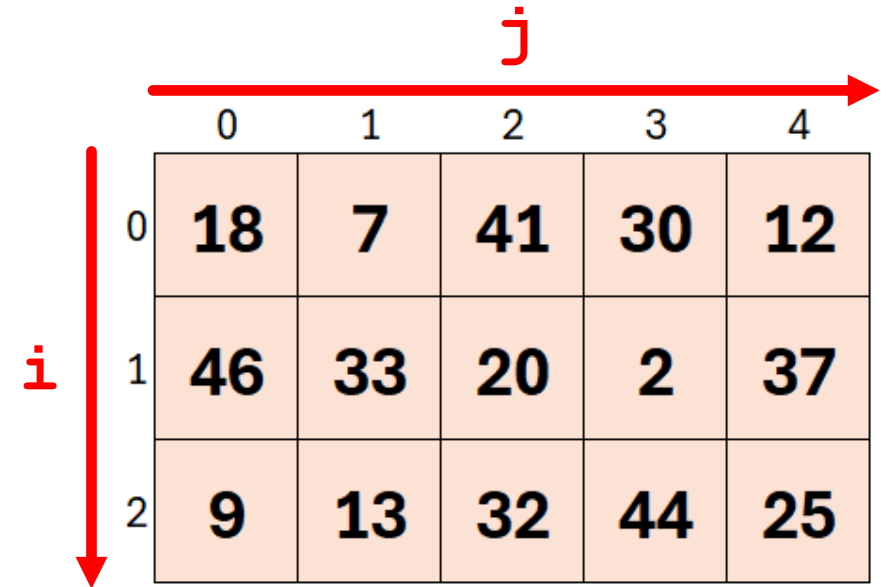
	0	1	2	3	4
0	18	7	41	30	12
1	46	33	20	2	37
2	9	13	32	44	25

## Feltételnek megfelelő elemek darabszáma

Feladat: Határozzuk meg mennyi 10-nél kisebb szám van a mátrixban!

```
int drb=0;
for (int i=0; i<M; i++) {
    for (int j=0; j<N; j++) {
        if (m[i][j] < 10) {
            drb++;
        }
    }
}

printf("10-nel kisebb elemek darabszama: %d\n", drb);
```



A 3x5 matrix is shown with row index  $i$  (0, 1, 2) and column index  $j$  (0, 1, 2, 3, 4). The matrix contains the following values:

	$j$	0	1	2	3	4
0	18	7	41	30	12	
1	46	33	20	2	37	
2	9	13	32	44	25	



## Feltételnek megfelelő elemek módosítása

Feladat: A 10-nél kisebb elemeket szorozzuk meg kettővel!

```
for (int i=0; i<M; i++) {  
    for (int j=0; j<N; j++) {  
        if (m[i][j] < 10) {  
            m[i][j] = m[i][j] * 2;  
        }  
    }  
}
```

		j				
		0	1	2	3	4
i	0	18	14	41	30	12
	1	46	33	20	4	37
	2	18	13	32	44	25

# Felhasználó által megadott szám keresése és indexének meghatározása

Feladat: Keressük meg a mátrixban a felhasználó által megadott  $k$  számot!

```
int k;  
printf("Keresett szám: ");  
scanf("%d", &k);  
int sorindex=-1, oszlopindex=-1;  
for (int i=0; i<M; i++) {  
    for (int j=0; j<N; j++) {  
        if (m[i][j] == k) {  
            sorindex = i;  
            oszlopindex = j;  
        }  
    }  
}  
  
if (sorindex != -1) {  
    printf("A keresett szám sorindexe: %d, oszlopindexe: %d\n",  
        sorindex, oszlopindex);  
} else {  
    printf("A keresett szám nem található a mátrixban.\n");  
}
```

A 3x5 matrix is shown with row index  $i$  and column index  $j$ . The matrix contains the following values:

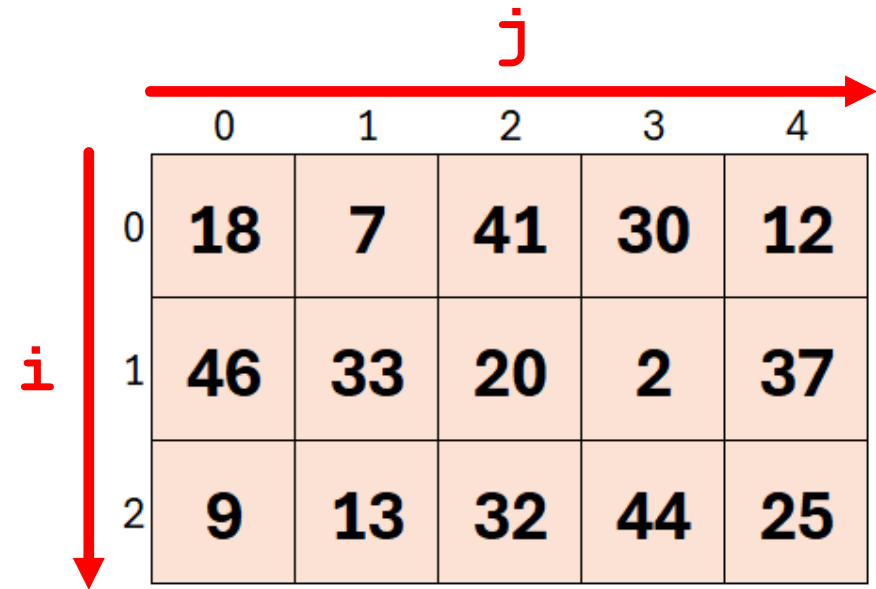
	0	1	2	3	4
0	18	7	41	30	12
1	46	33	20	2	37
2	9	13	32	44	25

# Minimum és maximum érték keresése és indexének meghatározása

Feladat: Keressük meg a mátrix legkisebb és legnagyobb elemét és indexeit!

```
int min=m[0][0], mini=0, minj=0;
int max=m[0][0], maxi=0, maxj=0;
for (int i=0; i<M; i++) {
    for (int j=0; j<N; j++) {
        if (m[i][j] < min) {
            min = m[i][j];
            mini = i;
            minj = j;
        }
        if (m[i][j] > max) {
            max = m[i][j];
            maxi = i;
            maxj = j;
        }
    }
}
```

```
printf("Minimum: %d, sorindexe: %d, oszlopindexe: %d\n", min, mini, minj);
printf("Maximum: %d, sorindexe: %d, oszlopindexe: %d\n", max, maxi, maxj);
```



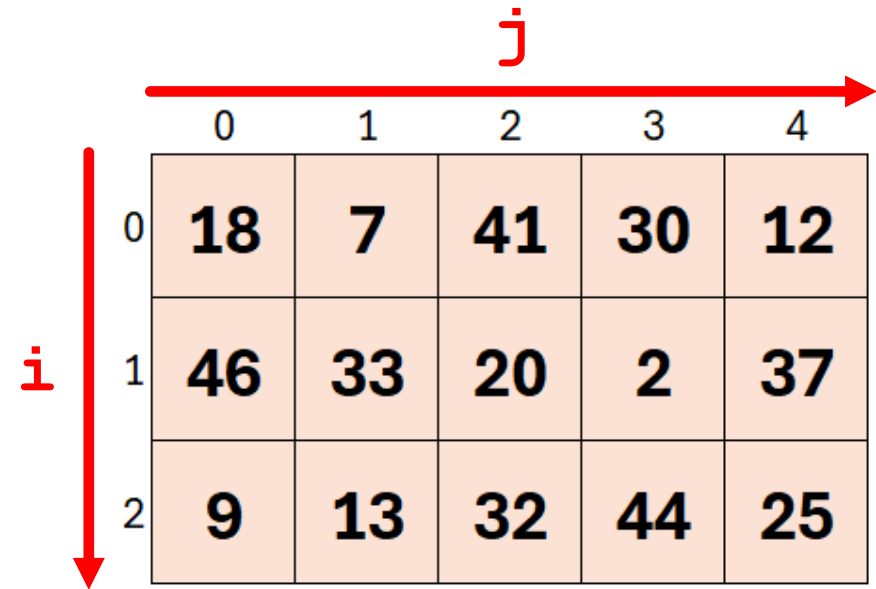
A 3x5 matrix is shown with row indices 0, 1, 2 and column indices 0, 1, 2, 3, 4. A red arrow labeled 'i' points downwards along the left side, and a red arrow labeled 'j' points to the right along the top. The matrix contains the following values:

	0	1	2	3	4
0	18	7	41	30	12
1	46	33	20	2	37
2	9	13	32	44	25

Felesleges külön megjegyezni a minimum és maximum értékét ( `min` és `max` változók ), mivel azok az indexeik segítségével bármikor elérhető ( `m[mini][minj]` és `m[maxi][maxj]` elemekként)!

```
int mini=0, minj=0;
int maxi=0, maxj=0;
for (int i=0; i<M; i++) {
    for (int j=0; j<N; j++) {
        if (m[i][j] < m[mini][minj]) {
            mini = i;
            minj = j;
        }
        if (m[i][j] > m[maxi][maxj]) {
            maxi = i;
            maxj = j;
        }
    }
}
```

```
printf("Minimum: %d, sorindexe: %d, oszlopindexe: %d\n",
      m[mini][minj], mini, minj);
printf("Maximum: %d, sorindexe: %d, oszlopindexe: %d\n",
      m[maxi][maxj], maxi, maxj);
```



A 3x5 matrix is shown with row index *i* (0, 1, 2) and column index *j* (0, 1, 2, 3, 4). The matrix contains the following values:


	0	1	2	3	4
0	18	7	41	30	12
1	46	33	20	2	37
2	9	13	32	44	25

## Mátrix adott sorának vagy oszlopának módosítása

Feladat: Módosítsuk a mátrix 1. sorának (1-es sorindexű) elemeit 0-ra!

Mivel csak egy sor elemeit szeretnénk végigjárni, ezt megtehetjük egyetlen for ciklussal:

```
for (int i=0; i<N; i++) {  
    m[1][i] = 0;  
}
```




	0	1	2	3	4
0	18	7	41	30	12
1	0	0	0	0	0
2	9	13	32	44	25

Feladat: Módosítsuk a mátrix 3. oszlopának (3-as oszlopindexű) elemeit 0-ra!

Mivel csak egy oszlop elemeit szeretnénk végigjárni, ezt is megtehetjük egyetlen for ciklussal:

```
for (int i=0; i<M; i++) {  
    m[i][3] = 0;  
}
```




	0	1	2	3	4
0	18	7	41	0	12
1	46	33	20	0	37
2	9	13	32	0	25

## Két sor vagy oszlop felcserélése

Feladat: Cseréljük fel a mátrix 0. és 1. indexű sorát!


```
for (int i=0; i<N; i++) {  
    int tmp = m[0][i];  
    m[0][i] = m[1][i];  
    m[1][i] = tmp;  
}
```



	0	1	2	3	4
0	46	33	20	2	37
1	18	7	41	30	12
2	9	13	32	44	25

Feladat: Cseréljük fel a mátrix 2. és 3. indexű oszlopát!

```
for (int i=0; i<M; i++) {  
    int tmp = m[i][2];  
    m[i][2] = m[i][3];  
    m[i][3] = tmp;  
}
```



	0	1	2	3	4
0	18	7	30	41	12
1	46	33	2	20	37
2	9	13	44	32	25

# Mátrix tükrözése vízszintes vagy függőleges tengelye mentén

Feladat: Tükrözzük a mátrixot függőleges tengelye mentén!

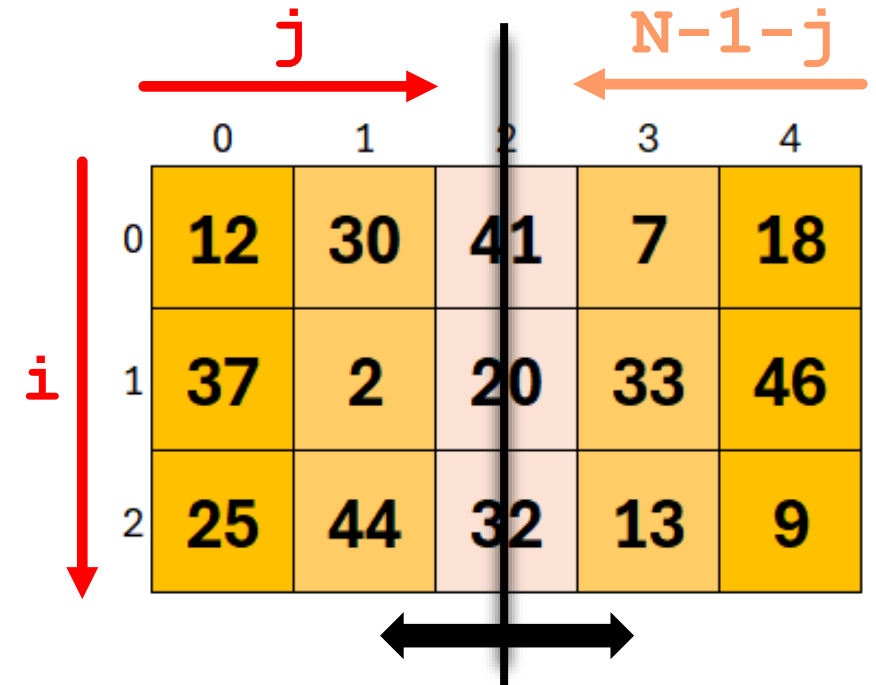
Melyik oszlopokat kell kicserélnünk?

- ▶ 0. oszlopot 4. oszloppal (N-1. oszloppal)
- ▶ 1. oszlopot 3. oszloppal (N-2. oszloppal)

Általánosan:

- ▶ j. oszlopot N-1-j. oszloppal

```
for (int i=0; i<M; i++) {  
    for (int j=0; j<N/2; j++) {  
        int tmp = m[i][j];  
        m[i][j] = m[i][N-1-j];  
        m[i][N-1-j] = tmp;  
    }  
}
```



# Mátrix transzponálása (sorok felcserélése az oszlopokkal)

Feladat: Készítsük el az  $m$  mátrix transzponáltját, az eredményt tároljuk a  $t$  mátrixban!

```
for (int i=0; i<M; i++) {  
    for (int j=0; j<N; j++) {  
        t[j][i] = m[i][j];  
    }  
}
```

`int m[M][N]`

	$j$				
	0	1	2	3	4
$i$ 0	18	7	41	30	12
1	46	33	20	2	37
2	9	13	32	44	25

`int t[N][M]`

	$i$		
	0	1	2
$j$ 0	18	46	9
1	7	33	13
2	41	20	32
3	30	2	44
4	12	37	25



# Mátrix sorainak vagy oszlopainak rendezése

Feladat: Rendezzük a mátrix minden sorát külön-külön növekvő sorrendbe!

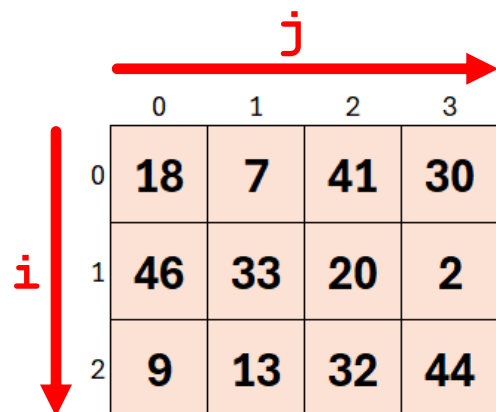
```
for (int i=0; i<M; i++) {  
    // az i. sor rendezese egyszeru cseres rendezessel  
    for (int j=0; j<N-1; j++) {  
        for (int k=j+1; k<N; k++) {  
            if (m[i][j] > m[i][k]) {  
                int tmp = m[i][j];  
                m[i][j] = m[i][k];  
                m[i][k] = tmp;  
            }  
        }  
    }  
}
```

A 3x5 matrix is shown with rows indexed 0 to 2 and columns indexed 0 to 4. A red arrow labeled 'i' points downwards along the first column. Two orange arrows at the top point to the right: the top one is labeled 'j' and starts at column 0, the bottom one is labeled 'k' and starts at column 1.

	0	1	2	3	4
0	7	12	18	30	41
1	2	20	33	37	46
2	9	13	25	32	44

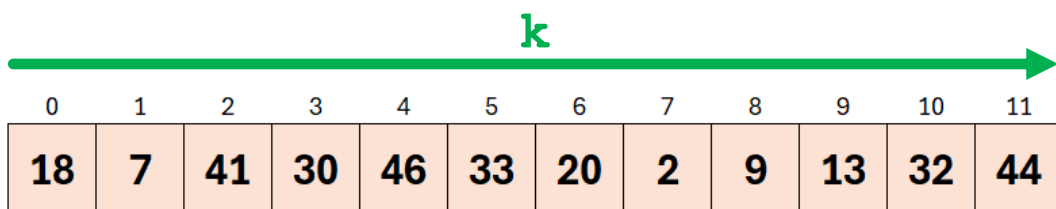
## Teljes mátrix rendezése (átmásolva egydimenziós tömbbe, majd visszamásolva)

Feladat: Rendezzük a mátrix elemeit  
növekvő sorrendbe!



A 3x4 matrix with row index *i* (0 to 2) and column index *j* (0 to 3). The elements are:

	0	1	2	3
0	18	7	41	30
1	46	33	20	2
2	9	13	32	44



A 1D array with index *k* (0 to 11). The elements are:

0	1	2	3	4	5	6	7	8	9	10	11
18	7	41	30	46	33	20	2	9	13	32	44

Hogyan fejezhetjük ki *k*-t az *i* és *j*  
változók segítségével?

$$k = i * \text{oszlopok\_száma} + j$$

// matrix atmasolasa egydimenzios tombbe

```
int a[M*N];  
for (int i=0; i<M; i++) {  
    for (int j=0; j<N; j++) {  
        a[i*N+j] = m[i][j];  
    }  
}
```

// egydimenzios tomb rendezese

```
for (int i=0; i<M*N-1; i++) {  
    for (int j=i+1; j<M*N; j++) {  
        if (a[i] > a[j]) {  
            int tmp = a[i];  
            a[i] = a[j];  
            a[j] = tmp;  
        }  
    }  
}
```

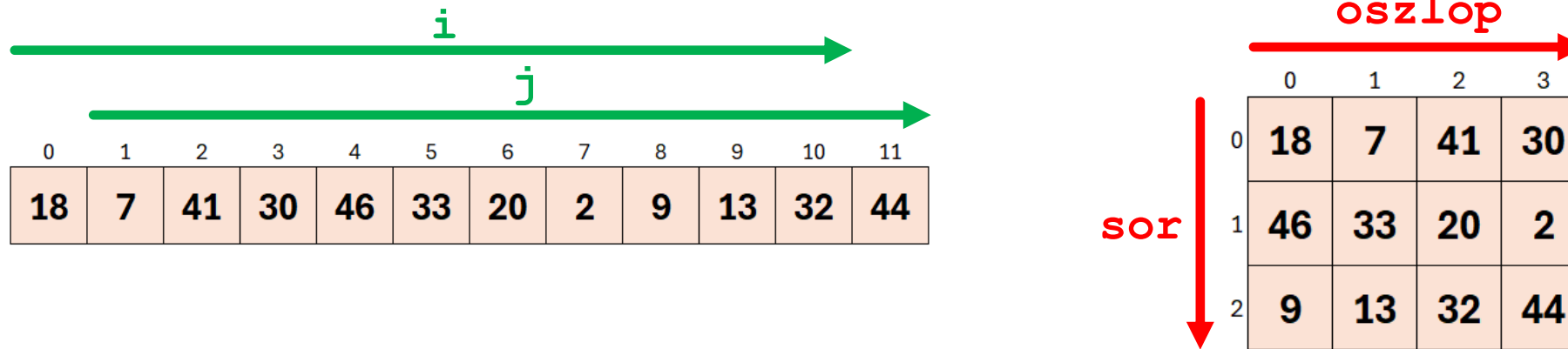
// egydimenzios tomb visszamasolasa matrixba

```
for (int i=0; i<M; i++) {  
    for (int j=0; j<N; j++) {  
        m[i][j] = a[i*N+j];  
    }  
}
```

# Teljes mátrix rendezése (helyben, egyszerű cserés rendezéssel)

Feladat: Rendezzük a mátrix elemeit növekvő sorrendbe.

- ▶ A rendezési algoritmusokat egydimenziós tömbökön tudjuk jól használni.
- ▶ Pl. egyszerűs cserés rendezésnél  $i$  és  $j$  ciklusváltozók segítségével tudjuk rendezni a tömböt:



Hogyan tudnánk meghatározni, hogy az egydimenziós tömb valamelyik eleme a mátrix melyik elemének felel meg? Pl. az  $i$  segítségével hogyan határozható meg az adott elem sora és oszlopa a mátrixban?

```
sor = i / oszlopok_száma
```

```
oszlop = i % oszlopok_száma
```

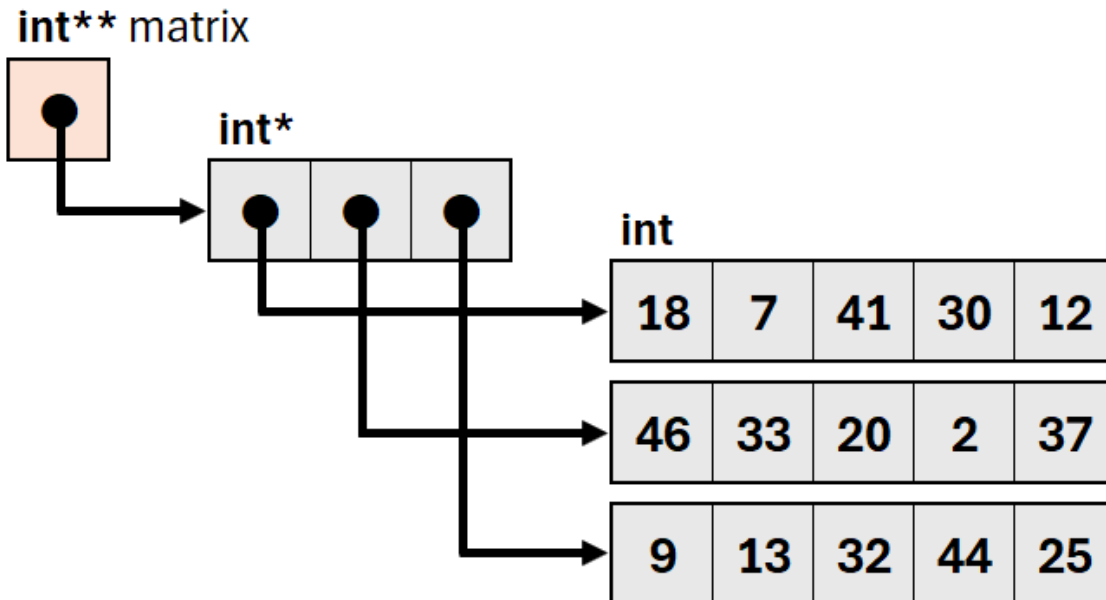
Mátrix rendezése helyben, egyszerű cserés rendezési algoritmussal:

```
for (int i=0; i<M*N-1; i++) {  
    for (int j=i+1; j<M*N; j++) {  
        int i_sor = i / N;  
        int i_oszlop = i % N;  
        int j_sor = j / N;  
        int j_oszlop = j % N;  
        if (m[i_sor][i_oszlop] > m[j_sor][j_oszlop]) {  
            int tmp = m[i_sor][i_oszlop];  
            m[i_sor][i_oszlop] = m[j_sor][j_oszlop];  
            m[j_sor][j_oszlop] = tmp;  
        }  
    }  
}
```

# Mátrix létrehozása dinamikus memórafoglalással

Ha a mátrix mérete futási időben derül ki, a memóriát dinamikusan kell lefoglalni.

Ebben az esetben a mátrixot nem egybefüggő memóriaterületen tároljuk, hanem **egy pointerek tömbje mutat soronként külön lefoglalt memóriaterületekre.**



```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

```
int main() {
    // sorok (M) es oszlopok (N) szama
    int M = 3, N = 5;
    // memoria lefoglalasa
    int** matrix = (int**)malloc(M * sizeof(int*));
    for (int i = 0; i < M; i++) {
        matrix[i] = (int*)malloc(N * sizeof(int));
    }
    // matrix generalasa es kiirasa
    srand(time(NULL));
    for (int i=0; i<M; i++) {
        for (int j=0; j<N; j++) {
            matrix[i][j] = rand() % 50 + 1;
            printf("%3d", matrix[i][j]);
        }
        printf("\n");
    }
    // memoria felszabaditasa
    for (int i = 0; i < M; i++) {
        free(matrix[i]);
    }
    free(matrix);
}
```

# Három- és többdimenziós tömbök

A többdimenziós tömbök hasznosak lehetnek olyan problémák megoldásában, ahol az adatokat térbeli, időbeli vagy más többváltozós jellemzők mentén kell tárolni és kezelni.

- ▶ **RGB képek tárolása** - a színes kép piros (R), zöld (G) és kék (B) csatornája egy ilyen 3D tömbben tárolható:

```
int kep[szelesseg][magassag][3];
```

- ▶ **Animáció vagy videó tárolása** - egy videó sok egymás után következő képkockából áll, minden képkocka egy színes képet tartalmaz, ezért egy 4D tömbben tárolható:

```
int video[kepckak_szama][szelesseg][magassag][3];
```

- ▶ **Mesterséges intelligencia, mélytanulás, tudományos szimulációk, többváltozós adatok** - 4D és 5D+ tömbök

