Mátrixok használata

1. feladat: Adjunk össze két 3×3-as egész számokat tároló mátrix-ot és tároljunk le egy harmadikban az eredményt. Használjunk függvényt a mátrix elemeinek bekéréséhez.

```
#include <stdio.h>

#define SIZE 3 // a matrix-ok merete
void readMatrix(int m[][SIZE])
{
    for(int sor = 0; sor < SIZE; sor++)
    {
        for(int oszlop = 0; oszlop < SIZE; oszlop++)
        {
            scanf("%d", &m[sor][oszlop]);
        }
    }
}

int main()
{
    int A[SIZE][SIZE]; // Matrix 1
    int B[SIZE][SIZE]; // Matrix 2
    int C[SIZE][SIZE]; // eredmeny Matrix

    printf("Kerem az elso matrix elemeit: \n");
    readMatrix(A);
    printf("Kerem az masodik matrix elemeit: \n");
    readMatrix(B);

    for(int sor = 0; sor < SIZE; sor++)
    {
        for(int oszlop = 0; oszlop < SIZE; oszlop++)
        {
            C[sor][oszlop] = A[sor][oszlop] + B[sor][oszlop];
        }
    }


    printf("\nMatrixok osszege: A+B = \n");
    for(int sor = 0; sor < SIZE; sor++)
    {
        for(int oszlop = 0; oszlop < SIZE; oszlop++)
        {
            printf("%d ", C[sor][oszlop]);
        }
        printf("\n");  // soronkent uj sor
    }
}
```

2. feladat: Az első feladatban a

```
readMatrix()
```

függvényt cseréljük le:

```
void readMatrix(int ** m)
```

-ra és próbáljuk meg mi történik és magyarázzuk meg miért?

3. feladat: Az első feladatban megadott feladatot oldjuk meg pointerekkel is.

```
#include <stdio.h>

#define SIZE 3 // a matrix-ok merete
void readMatrix(int *m)
{
    for(int i = 0; i < SIZE * SIZE; i++)
    {
        scanf("%d", (m + i));
    }
}

int main()
{
    int A[SIZE][SIZE]; // Matrix 1
    int B[SIZE][SIZE]; // Matrix 2

    printf("Kerem az elso matrix elemeit: \n");
    readMatrix(&A[0][0]);
    printf("Kerem az masodik matrix elemeit: \n");
    readMatrix(&B[0][0]);

    int C[SIZE][SIZE]; // eredmeny Matrix

    for(int i = 0; i < SIZE * SIZE; i++)
    {
        *(&C[0][0] + i) = *(&A[0][0] + i) + *(&B[0][0] + i);
    }

    printf("\nMatrixok osszege: A+B = \n");
    for(int sor = 0; sor < SIZE; sor++)
    {
        for(int oszlop = 0; oszlop < SIZE; oszlop++)
        {
            printf("%d ", C[sor][oszlop]);
        }
```

```
        printf("\n");  // soronkent uj sor
    }
}
```

4. feladat: Ellenőrizzük, hogy a felhasználótól bekért 3×3-as mátrix felső háromszög típusú-e?

```
#include <stdio.h>

#define SIZE 3 // a matrix-ok merete
void readMatrix(int m[][SIZE])
{
    for(int sor = 0; sor < SIZE; sor++)
    {
        for(int oszlop = 0; oszlop < SIZE; oszlop++)
        {
            scanf("%d", &m[sor][oszlop]);
        }
    }
}

int main()
{
    int A[SIZE][SIZE];

    printf("Kerem a matrix elemeit: \n");
    readMatrix(A);

    int felsoHaromszog = 1;
    // Hogyan jovunk ra a megoldasra?
    //
    // 1 2 3   [0,0] [0,1] [0,2]
    // 0 4 5 =>[1,0] [1,1] [1,2]
    // 0 0 6   [2,0] [2,1] [2,2]
    //
    // Mi a közös a három 0 érték indexeiben?
    // - az hogy a sor index erteke mindig nagyobb mint
    // az oszlop index!!
    //
    for(int sor = 0; sor < SIZE; sor++)
    {
        for(int oszlop = 0; oszlop < SIZE; oszlop++)
        {
            //
            // a foatlo alatti elemeknek 0-nak kell lenniuk
            //
            if(sor > oszlop && A[sor][oszlop]!=0)
            {
                felsoHaromszog = 0;
            }
```

```
        }
    }
    if(felsoHaromszog)
    {
        printf("felso haromszog matrix");
    }
    else
    {
        printf("nem felso haromszog");
    }
}
```

5. feladat: Írjon C programot, ami meghatározza, hogy egy mátrix ritka mátrix-e? Egy ritka mátrix olyan mártix aminek legalább elemeinek a fele 0.

6. feladat: Írjon C programot, ami bekér a felhasználótól egy 3×3-as mátrixot és kiírja a transzponáltját.

```
#include <stdio.h>

#define SIZE 3 // a matrix-ok merete
void readMatrix(int m[][SIZE])
{
    for(int sor = 0; sor < SIZE; sor++)
    {
        for(int oszlop = 0; oszlop < SIZE; oszlop++)
        {
            scanf("%d", &m[sor][oszlop]);
        }
    }
}

int main()
{
    int A[SIZE][SIZE];
    int E[SIZE][SIZE];
    printf("Kerem a matrix elemeit: \n");
    readMatrix(A);

    // Hogyan jovunk ra a megoldasra?
    // Transzpontalt matrix pelda:
    //
    // 1 2 3    1 4 7
    // 4 5 6 => 2 5 8
    // 7 8 9    3 6 9
    //
```

```
    // A matrix [i][j] elemét [j][i]-re kell cserelni
    for(int sor = 0; sor < SIZE; sor++)
    {
        for(int oszlop = 0; oszlop < SIZE; oszlop++)
        {
            E[sor][oszlop] = A[oszlop][sor];
        }
    }

    for(int sor = 0; sor < SIZE; sor++)
    {
        for(int oszlop = 0; oszlop < SIZE; oszlop++)
        {
            printf("%d ", E[sor][oszlop]);
        }
        printf("\n");
    }
}
```

7. feladat: Írjon C programot, ami összeszoroz két mátrixot.

```
#include <stdio.h>

#define MAX_SOR 3
#define MAX_OSZLOP 3
void readMatrix(int m[][MAX_OSZLOP])
{
    for(int sor = 0; sor < MAX_SOR; sor++)
    {
        for(int oszlop = 0; oszlop < MAX_OSZLOP; oszlop++)
        {
            scanf("%d", &m[sor][oszlop]);
        }
    }
}

int main()
{
    int A[MAX_SOR][MAX_OSZLOP];
    int B[MAX_SOR][MAX_OSZLOP];
    int C[MAX_SOR][MAX_OSZLOP];
    printf("Kerem az A matrix elemeit: \n");
    readMatrix(A);
    printf("Kerem a B matrix elemeit: \n");
    readMatrix(B);


    for (int sor = 0; sor < MAX_SOR; sor++)
    {
```

```
        for (int oszlop = 0; oszlop < MAX_OSZLOP; oszlop++)
        {
            int sum = 0;
            for (int i = 0; i < MAX_OSZLOP; i++)
            {
                sum += A[sor][i] * B[i][oszlop];
            }
            C[sor][oszlop] = sum;
        }
    }


    for(int sor = 0; sor < MAX_SOR; sor++)
    {
        for(int oszlop = 0; oszlop < MAX_OSZLOP; oszlop++)
        {
            printf("%d ", C[sor][oszlop]);
        }
        printf("\n");
    }
}
```

8. feladat: Oldjuk meg nem négyzetes mátrixokkal is a mátrix összeadós feladatot, valamint dinamikus memória kezeléssel:

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

// Dinamikus mátrix létrehozás
int **allocateMatrix(int rows, int cols) {
    int **matrix = (int **)malloc(rows * sizeof(int *));
    for (int i = 0; i < rows; i++) {
        matrix[i] = (int *)malloc(cols * sizeof(int));
    }
    return matrix;
}

// Mátrix felszabadítás
void freeMatrix(int **matrix, int rows) {
    for (int i = 0; i < rows; i++) {
        free(matrix[i]);
    }
    free(matrix);
}

// Mátrix beolvasás
```

```
void readMatrix(int **m, int rows, int cols) {
    printf("Adja meg a mátrix elemeit (%d x %d):\n", rows, cols);
    for (int sor = 0; sor < rows; sor++) {
        for (int oszlop = 0; oszlop < cols; oszlop++) {
            scanf("%d", &m[sor][oszlop]);
        }
    }
}

// Mátrixok összeadása
void addMatrices(int **A, int **B, int **C, int rows, int cols) {
    for (int sor = 0; sor < rows; sor++) {
        for (int oszlop = 0; oszlop < cols; oszlop++) {
            C[sor][oszlop] = A[sor][oszlop] + B[sor][oszlop];
        }
    }
}

void printMatrix(int **matrix, int rows, int cols) {
    printf("\nMátrix:\n");
    for (int sor = 0; sor < rows; sor++) {
        for (int oszlop = 0; oszlop < cols; oszlop++) {
            printf("%d ", matrix[sor][oszlop]);
        }
        printf("\n");
    }
}

int main() {
    int rows, cols;

    printf("Adja meg a mátrix sorainak és oszlopainak számát: ");
    scanf("%d %d", &rows, &cols);

    int **A, **B, **C;

    // Mátrixok allokálása a megadott méret szerint
    A = allocateMatrix(rows, cols);
    B = allocateMatrix(rows, cols);
    C = allocateMatrix(rows, cols);

    // Mátrixok beolvasása
    printf("Kerem az elso matrix elemeit:\n");
    readMatrix(A, rows, cols);

    printf("Kerem a masodik matrix elemeit:\n");
    readMatrix(B, rows, cols);

    // Összeadás
    addMatrices(A, B, C, rows, cols);
```

```
    printf("\nMatrixok osszege: A+B = \n");
    printMatrix(C, rows, cols);

    // Memória felszabadítása
    freeMatrix(A, rows);
    freeMatrix(B, rows);
    freeMatrix(C, rows);

    return 0;
}
```

9. feladat : vizsgáljuk meg a llama chat model c implementációját: https://github.com/karpathy/llama2.c/blob/master/run.c a 217.es sortól kezdődő függvényt.

Ellenőrző kérdések

Mi a helyes deklaráció egy 3×3-as egész mátrix létrehozására C-ben?

```
A) int matrix[3][3];
B) int matrix[9];
C) int matrix[3,3];
D) matrix int[3][3];
```

Megoldás: A

Mi lesz a következő kódrészlet eredménye?

```
int matrix[2][2] = {{1,2}, {3,4}};
printf("%d", matrix[1][0]);
```

```
A) 1
B) 2
C) 3
D) 4
```

Megoldás: C (az első sor nulladik eleme, ami a 3, hiszen a sorok is 0-tól indulnak.

Milyen ciklusszerkezet alkalmas egy kétdimenziós mátrix minden elemének bejárására?

```
A) Egyetlen for-ciklus
```

```
B) Egymásba ágyazott for-ciklusok
C) while ciklus, egyetlen feltétellel
D) Nincs szükség ciklusra
```

Megoldás: B

Mi lesz a kód eredménye?

```
int arr[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
printf("%d", arr[1][2]);
```

```
A) 2
B) 4
C) 6
D) 8
```

Megoldás: C –> 2. sor 3. eleme, ami a 6-os.

Melyik állítás igaz az alábbiak közül?

```
A) A C-ben a mátrix sorainak száma opcionális deklarációkor.
B) Mátrixokat nem lehet paraméterként átadni függvényeknek.
C) Mátrix elemei mindig ugyanazt a típust tárolják.
```

Megoldás: C

Hogyan adunk át egy mátrixot paraméterként egy függvénynek C-ben helyesen?

```
A) void fuggveny(int matrix[][])
B) void fuggveny(int **matrix)
C) void fuggveny(int matrix[][3])
D) void fuggveny(matrix int[3][3])
```

Megoldás: C –> második dimenzió méretét kötelező megadni

Hogyan érhető el az alábbi mátrix legutolsó eleme (9) az alábbi deklarációból?

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

```
A) mat[9]
B) mat[3][3]
C) mat[2][2]
D) mat[3][2]
```

Megoldás: C –> mivel az indexelés 0-tól indul

Mit csinál az alábbi kód?

```
int mat[3][3] = {0};
```

```
A) Lefoglal egy 3x3-as mátrixot véletlenszerű értékekkel.
B) Lefoglal egy 3x3-as mátrixot nullákkal feltöltve.
C) Csak a mátrix első elemét inicializálja 0-ra.
D) Csak a mátrix első sorát tölti fel nullával.
```

Megoldás: B → a modern c-ben ezzel a szintaktikával lehet nullákkal feltölteni egy mátrixot.

Hogyan lehet helyesen dinamikusan lefoglalni egy 5×5-ös mátrixot?

```
A) int mat[5][5] = malloc(25 * sizeof(int));
B)
```

```
int **mat = malloc(5 * sizeof(int *));
for(int i = 0; i < 5; i++)
    mat[i] = malloc(5 * sizeof(int));
```

```
C) int *mat[5] = malloc(sizeof(int) * 25);
D) int mat[][] = malloc(5*5*sizeof(int));
```

Megoldás: B

Hogyan kell helyesen felszabadítani a memóriát egy dinamikusan foglalt kétdimenziós mátrix esetén?

```
A) free(mat);
B)
```

```
for (int i = 0; i < sorok; i++)
    free(mat[i]);
free(mat);
```

```
C) free(mat[][]);
D) free(mat);
```

Megoldás: B
