2. ELMÉLETI LAP

Áttekintő:

Egyszerű függvények implementálása

```
1. int** allocateMemoryForMatrix1(int rows, int cols);
vagy void allocateMemoryForMatrix2(int rows, int cols, int **dpMatrix);
2. void readMatrix(int *pRows, int *pCols, int **dpArray, const char *in);
3. void printMatrix(int rows, int cols, int **pMatrix, const char *out);
4. int minimumValueOfColumn(int rows, int columnIndex, int **pMatrix);
5. void deallocateMemoryForMatrix(int rows, int **dpMatrix);
```

Megoldás:

Deklaráció:

```
int** allocateMemoryForMatrix1(int rows, int cols);
```

Függvény dokumentáció:

```
/**
 * Lefoglalja a memóriát rows * cols egész számnak és visszatérít egy pointert, a lefoglalt memória címét
 * @param rows sorok száma
 * @param cols oszlopok száma
 * @return mutató a matrix címe felé
 */
```

Definíció:

```
int **allocateMemoryForMatrix1(int rows, int cols) {
    int **matrix = (int**) calloc(rows, sizeof (int*));
    if(!matrix)
    {
        printf(MEMORY_ALLOCATION_ERROR_MESSAGE);
        exit(MEMORY_ALLOCATION_ERROR_CODE);
    }
    for (int i = 0; i < rows; ++i) {
        matrix[i] = (int*) calloc(cols, sizeof (int));
```

```
        if(!matrix[i])
        {
            printf(MEMORY_ALLOCATION_ERROR_MESSAGE);
            exit(MEMORY_ALLOCATION_ERROR_CODE);
        }
    }
    return matrix;
}
```

Kódmagyarázat

A függvény kétdimenziós egész számú mátrixnak foglal helyet (calloc-al, ami nullázza az elemeket), először a sorok, majd minden sor oszlopainak memóriáját foglalja le, hibák esetén pedig leállítja a programot kiírva egy hibaüzenetet.

Megjegyzések (Big O):

- időbeli komplexitás **O(n * n)**
- térbeli komplexitás **O(n * n)**

Deklaráció:

```
void allocateMemoryForMatrix2(int rows, int cols, int **dpMatrix);
```

Paraméterek:

```
/**
 * Lefoglalja a memóriát rows * cols egész számnak és visszatérít egy pointert, a lefoglalt memória címét a dpMatrix cím szerinti paraméterében
 * @param sorok száma
 * @param oszlopok száma
 * @param dpMatrix dupla mutató a mátrix fele
 */
```

Definíció:

```
void allocateMemoryForMatrix2(int rows, int cols, int ***dpMatrix) {
    *dpMatrix = (int**) calloc(rows, sizeof (int*));
    if(!(*dpMatrix))
    {
        printf(MEMORY_ALLOCATION_ERROR_MESSAGE);
        exit(MEMORY_ALLOCATION_ERROR_CODE);
```

```
    }
    for (int i = 0; i < rows; ++i) {
        (*dpMatrix)[i] = (int*) calloc(cols, sizeof (int));
        if(!(*dpMatrix)[i])
        {
            printf(MEMORY_ALLOCATION_ERROR_MESSAGE);
            exit(MEMORY_ALLOCATION_ERROR_CODE);
        }
    }
}
```

Kódmagyarázat

A függvény dinamikusan foglal le egy *rows x cols* méretű egész szám mátrixot úgy, hogy a *dpMatrix paraméteren keresztül adja vissza az eredményt. Először lefoglalja a sorok mutatóit, majd minden sorhoz lefoglalja az oszlopok memóriáját, hibakezeléssel.

Megjegyzések:

- időbeli komplexitás **O(n * n)**
- térbeli komplexitás **O(n * n)**

Deklaráció:

```
void readMatrix(int *pRows, int *pCols, int **dpArray, const char *in);
```

Függvény dokumentáció:

```
/**
 * Reads beolvassa a mátrix elemeit állományból vagy billentyűzetről
 * @param pointer a sorok száma felé
 * @param pointer az oszlopok száma felé
 * @param dpArray double pointer a mátrix felé
 * @param bemenet
 */
```

Definíció:

```
void readMatrix(int *pRows, int *pCols, int ***dpArray, const char *input) {
    if(!freopen(input, "r", stdin))
```

```
    {
        printf(FILE_OPENING_ERROR_MESSAGE);
        exit(FILE_OPENING_ERROR_CODE);
    }
    scanf("%i%i", pRows, pCols);
    allocateMemoryForMatrix2(*pRows, *pCols, dpArray);
    for (int i = 0; i < *pRows; ++i) {
        for (int j = 0; j < *pCols; ++j) {
            scanf("%i", &((*dpArray)[i][j]));
        }
    }
    freopen("CON", "r", stdin);
}
```

Kódmagyarázat

A függvény beolvassa a mátrix méreteit és elemeit egy megadott fájlból. Először a standard bemenetet a fájlra irányítja, beolvassa a sor- és oszlopszámot, lefoglalja a megfelelő memóriát, majd a mátrix elemeit, végül visszaállítja a standard bemenetet a konzolra.

Megjegyzések:

- időbeli komplexitás **O(n * n)**
- térbeli komplexitás **O(n * n)**

Deklaráció:

```
void printMatrix(int rows, int cols, int **pMatrix, const char *out);
```

Függvény dokumentáció:

```
/**
 * kiírja a mátrix elemeit állományba vagy a standard output-ra
 * @param sorok
 * @param oszlopok
 * @param mátrix
 * @param kimenet
 */
```

```
void printMatrix(int rows, int cols, int **pMatrix, const char
*output) {
```

```
    freopen(output, "w", stdout);
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            printf("%5i ", pMatrix[i][j]);
        }
        printf("\n");
    }
    freopen("CON", "w", stdout);
}
```

Kódmagyarázat

A függvény a mátrixot a megadott fájlba vagy képernyőre írja ki. Először a standard kimenetet átirányítja az *output* fájlra, majd soronként, oszloponként kiírja a mátrix elemeit 5 karakter széles formázással, sorvégekkel, végül visszaállítja a kimenetet a konzolra.

Megjegyzések:

- időbeli komplexitás **O(n * n)**
- térbeli komplexitás **O(1)**

Deklaráció:

```
int minimumValueOfColumn(int rows, int *pCols);
```

Függvény dokumentáció:

```
/**
 * Egy adott sorszámú oszlop elemei közül téríti vissza a legkisebbet
 * @param sorok
 * @param pointer az adott oszlop fele
 * @return a legkisebb érték
 */
int minimumValueOfRow(int rows, int *pCols) {
    int minimum = INT_MAX;
    for (int i = 0; i < rows; ++i) {
        if(pCols[i] < minimum)
        {
            minimum = pCols[i];
```

```
        }
    }
    return minimum;
}
```

Kódmagyarázat

Inicializálja a minimumot a legnagyobb egész szám értékével (INT_MAX), majd végigiterál a tömbön, és ha egy elem kisebb, mint az aktuális minimum, frissíti azt. Végül visszaadja a megtalált legkisebb értéket.

Megjegyzések:

- időbeli komplexitás **O(n)**
- térbeli komplexitás **O(1)**

Deklaráció:

```
void deallocateMemoryForMatrix(int rows, int **dpMatrix);
```

Függvény dokumentáció:

```
/**
 * felszabadítja a lefoglalt memóriahelyet
 * @param dupla pointer, a mátrix
 */
```

Definíció:

```
void deallocateMatrix(int rows, int ***dpMatrix) {
    for (int i = 0; i < rows; ++i) {
        free((*dpMatrix)[i]);
    }
    free(*dpMatrix);
    *dpMatrix = NULL;
}
```

Kódmagyarázat

A függvény felszabadítja a dinamikusan lefoglalt kétdimenziós mátrix memóriáját. Először egy ciklusban minden sorhoz tartozó memóriaterületet szabadít fel, majd a sorok mutatóit tartalmazó

tömböt, végül a megadott pointert NULL-ra állítja, hogy jelezze a felszabadítást.

Megjegyzések:

- időbeli komplexitás **O(n)**
- térbeli komplexitás **O(1)**

[redacted]