```cpp
#include <iostream>
#include <vector>
#include <iomanip>
#include "ex01-library.h"


using namespace std;


// Exercise 1 (a) Check and correct if necessary
// Allocate an n x m matrix of int WITHOUT initializing it

int ** createMatrix(UINT n, UINT m) {
    // Correct Code:
    int ** A = new int * [n];
    for (unsigned int i = 0 ; i < n ; i++) {
        A[i] = new int[m];
    }

    return A;

    //**********************************************************************
    // Given Code: 3 mistakes:
    // int ** A = new int * [n];
    // for (unsigned int i = 0; i <= n; i++) {    // i <= n  ->  i < n
    //      A[i] = new int[n];                    // int[n]  ->  int[m]
    // }
                                                  // No return
}


// NOTE: Other options:
//
// int * A = new int[n * m];
// Then, to address the elements:
// for (int i = 0; i < n ; i++)
//      for (int j = 0; j < m; j++)
//          *(A + i*n + j) = value;

// C++11 would allow the following if m was a constant:
// int A = new int[n][m];


////////////////////////////////////////////////////////////////////////////


// Exercise 1 (b) Implement this function
// Copy a matrix A to another matrix B and return B


int ** duplicateMatrix(int ** A, UINT n, UINT m) {
    int ** B = createMatrix(n , m);
    for (unsigned int i = 0 ; i < n ; i++)
        for (unsigned int j = 0 ; j < m ; j++)
            B[i][j] = A[i][j];
    return B;
}

```

```cpp

////////////////////////////////////////////////////////////////////////////

// Exercise 1 (c) Implement this function
// Initialize a given matrix to 0


void initMatrix(int ** A, UINT n, UINT m) {
    for (unsigned int i = 0 ; i < n ; i++)
        for (unsigned int j = 0 ; j < m ; j++)
            A[i][j] = 0;
}


////////////////////////////////////////////////////////////////////////////

// Exercise 1 (d) Implement this function
// Deallocate a matrix previously allocated


void deallocateMatrix(int ** A, unsigned int n) {
    for(unsigned int i = 0 ; i < n ; i++)
        delete [] A[i];
    delete [] A;
}

// NOTE: To be completely robust a check should be made as trying to free
// memory that wasn't allocated results in undefined behaviour.


////////////////////////////////////////////////////////////////////////////

// Exercise 1 (e) Implement this function
// Perform a convolutional thersholding to create a binary image.


int ** makeBitonal(int ** A, UINT n, UINT m, int threshold) {
    int ** B = createMatrix(n , m);
    for (unsigned int i = 0 ; i < n ; i++)
        for (unsigned int j = 0 ; j < m ; j++)
            B[i][j] = (A[i][j] >= threshold) ? 255 : 0;
    return B;
}
```

```
///////////////////////////////////////////////////////////////////////////

// Do not modify


void printMatrix(int ** A, UINT n, UINT m, std::string description) {
    std::cout << "Printing: " << description << std::endl;

    for (unsigned int i = 0 ; i < n ; i++) {
        for (unsigned int j = 0 ; j < m ; j++) {
            std::cout << setw(5) << A[i][j] << " ";
        }
        std::cout << std::endl;
    }
    std::cout << std::endl;
}

///////////////////////////////////////////////////////////////////////////

// NOTE: I cannot find a way to protect a 2D array when passing it to a
// function by using const like with vectors...
```