

```
1  #include <iostream>
2  #include <vector>
3  #include <iomanip>
4  #include "ex01-library.h"
5
6
7  using namespace std;
8
9
10 // Exercise 1 (a) Check and correct if necessary
11 // Allocate an n x m matrix of int WITHOUT initializing it
12
13 int ** createMatrix(UINT n, UINT m) {
14     // Correct Code:
15     int ** A = new int * [n];
16     for (unsigned int i = 0 ; i < n ; i++) {
17         A[i] = new int[m];
18     }
19
20     return A;
21
22     //*****
23     // Given Code: 3 mistakes:
24     // int ** A = new int * [n];
25     // for (unsigned int i = 0; i <= n; i++) {    // i <= n -> i < n
26     //     A[i] = new int[n];                    // int[n] -> int[m]
27     // }
28
29     // No return
30 }
31
32 // NOTE: Other options:
33 //
34 // int * A = new int[n * m];
35 // Then, to address the elements:
36 // for (int i = 0; i < n ; i++)
37 //     for (int j = 0; j < m; j++)
38 //         *(A + i*n + j) = value;
39
40 // C++11 would allow the following if m was a constant:
41 // int A = new int[n][m];
42
43
44 ///////////////////////////////////////////////////////////////////
45
46
47 // Exercise 1 (b) Implement this function
48 // Copy a matrix A to another matrix B and return B
49
50
51 int ** duplicateMatrix(int ** A, UINT n, UINT m) {
52     int ** B = createMatrix(n , m);
53     for (unsigned int i = 0 ; i < n ; i++)
54         for (unsigned int j = 0 ; j < m ; j++)
55             B[i][j] = A[i][j];
56     return B;
57 }
58
```

```
59
60 ///////////////////////////////////////////////////
61
62 // Exercise 1 (c) Implement this function
63 // Initialize a given matrix to 0
64
65
66 void initMatrix(int ** A, UINT n, UINT m) {
67     for (unsigned int i = 0 ; i < n ; i++)
68         for (unsigned int j = 0 ; j < m ; j++)
69             A[i][j] = 0;
70 }
71
72
73 ///////////////////////////////////////////////////
74
75 // Exercise 1 (d) Implement this function
76 // Deallocate a matrix previously allocated
77
78
79 void deallocateMatrix(int ** A, unsigned int n) {
80     for(unsigned int i = 0 ; i < n ; i++)
81         delete [] A[i];
82     delete [] A;
83 }
84
85 // NOTE: To be completely robust a check should be made as trying to free
86 // memory that wasn't allocated results in undefined behaviour.
87
88
89 ///////////////////////////////////////////////////
90
91 // Exercise 1 (e) Implement this function
92 // Perform a convolutional thersholding to create a binary image.
93
94
95 int ** makeBitonal(int ** A, UINT n, UINT m, int threshold) {
96     int ** B = createMatrix(n , m);
97     for (unsigned int i = 0 ; i < n ; i++)
98         for (unsigned int j = 0 ; j < m ; j++)
99             B[i][j] = (A[i][j] >= threshold) ? 255 : 0;
100     return B;
101 }
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
```

```
117 ///////////////////////////////////////////////////////////////////
118
119 // Do not modify
120
121
122 void printMatrix(int ** A, UINT n, UINT m, std::string description) {
123     std::cout << "Printing: " << description << std::endl;
124
125     for (unsigned int i = 0 ; i < n ; i++) {
126         for (unsigned int j = 0 ; j < m ; j++) {
127             std::cout << setw(5) << A[i][j] << " ";
128         }
129         std::cout << std::endl;
130     }
131     std::cout << std::endl;
132 }
133
134 ///////////////////////////////////////////////////////////////////
135
136 // NOTE: I cannot find a way to protect a 2D array when passing it to a
137 // function by using const like with vectors...
```