Multidimensional arrays exercises - Taken from pbinfo.ro

Clasa IX-a => Tablouri Bidimensionale(matrice) => Parcurgerea matricelor patratice => Mediu

1. Given a matrix with n rows and n columns, create a C++ program which will compute the sum of the elements which are on the two diagonals which are neighbor with the main diagonal.
    - Sample Input:
    - n = 5

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

    - Sample Output: 30
    - Solution:

```
#include <iostream>
const int n = 5;

using namespace std;

int main() {
    int mat[n][n] = {
            {3, 1, 8, 5, 4},
            {7, 8, 5, 1, 2},
            {2, 2, 6, 7, 3},
            {9, 8, 1, 3, 6},
            {7, 5, 3, 1, 7},
    };

    int sum = 0;
    for(int i = 0; i < n; i++) {
    if(i == 0) {
        sum += mat[i][i+1];
    } else if (i == (n-1)) {
        sum += mat[n-1][n-2];
    } else {
        sum += mat[i][i-1];
        sum += mat[i][i+1];
    }
    }
```

```
        cout<< sum;

        return 0;
    }
```

2. Given a matrix with n rows and n columns, create a C++ program which will generate a new matrix, which will be symetrical with respect to the second diagonale of the given matrix

    - Sample Input:
        - 4

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

    - Sample Output:

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

    - Solution:

```
#include <iostream>
const int n = 4;

using namespace std;

int main() {
    int mat[n][n] = {
            {3, 1, 8, 5},
            {7, 8, 5, 1},
            {2, 2, 6, 7},
            {9, 8, 1, 3},
    };

    int result[n][n] = {0};

    for(int i = 0; i < n; i++) {
        for (int j = 0; j <  n; j++) {
            result[i][j] = mat[i][j];
        }
    }

    for (int i = 0; i < n; i++) {
```

```
            for (int j = 0; j < n; j++) {
                if(j < (n - i -1)) {
                    int temp = result[i][j];
                    result[i][j] = mat[n - j -1][n-1-i];
                    result[n-j-1][n-1-i] = temp;
                }
            }
        }

        for(int i =0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                cout << result[i][j] <<" ";
            }
            cout << endl;
        }
        return 0;
    }
```

3. Given a matrix with n rows and n columns. Create a C++ program which will display the elements by scrolling through the matrix, starting with the element on the first row and the first column as in the example:

    - Sample Input:

    ```
     1  3  4 10
     2  5  9 11
     6  8 12 15
     7 13 14 16
    ```

    - Sample Output:

        - 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

    - Solution:

    ```
            #include <iostream>
            using namespace std;

            const int n = 4;
            bool isElementInsideMatrixBoundary(int row, int col);
            int main()
            {
                int mat[n][n]={
                        {1,3,4,10},
                        {2,5,9,11},
                        {6,8,12,15},
                        {7,13,14,16}
                };
    ```

```
            int currentRow = 0;
            int currentCol = 0;
            int direction = 0; // 0 => going down, 1 => going up
            while (currentRow < n || currentCol < n) {
                int temporaryRow = currentRow;
                int temporaryCol = currentCol;
                if(direction == 0) {
                    while(isElementInsideMatrixBoundary(temporaryRow,
temporaryCol)) {
                        currentRow = temporaryRow;
                        currentCol = temporaryCol;
                        cout << mat[currentRow][currentCol] << " ";
                        temporaryRow +=1;
                        temporaryCol -=1;
                    }
                    direction = 1;
                    if(isElementInsideMatrixBoundary(currentRow+1,
currentCol)) {
                        currentRow++;
                    } else {
                        currentCol++;
                    }
                } else if (direction == 1) {
                    while(isElementInsideMatrixBoundary(temporaryRow,
temporaryCol)) {
                        currentRow = temporaryRow;
                        currentCol = temporaryCol;
                        cout << mat[currentRow][currentCol] << " ";
                        temporaryRow -=1;
                        temporaryCol +=1;
                    }
                    direction = 0;
                    if(isElementInsideMatrixBoundary(currentRow,
currentCol+1)) {
                        currentCol++;
                    } else {
                        currentRow++;
                    }
                }
            }
            return 0;
        }

        bool isElementInsideMatrixBoundary(int row, int col) {
            return (row >= 0 && row < n) && (col >= 0 && col < n);
        }
```

4. Given an array with n rows and n columns and natural number elements, create a C++ program which will Display, in ascending order, the sums of the elements in the four areas delimited by diagonals.
    - Sample Input:

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

- Sample Output:
  - `10 18 19 20`
- Solution:

```
#include <iostream>
using namespace std;

const int n = 5;
int main()
{
    int mat[n][n]={
            {3,1,8,5,4},
            {7,8,5,1,2},
            {2,2,6,7,3},
            {9,8,1,3,6},
            {7,5,3,1,7}
    };

    int sumNord = 0;
    int sumEast = 0;
    int sumSouth = 0;
    int sumWest = 0;
    int sumsArray[4] = {};

    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++) {
            if(j > i && j < n-i-1) {
                sumNord += mat[i][j];
            } else if (j > i && j > (n -i-1) ) {
                sumEast += mat[i][j];
            } else if (i > j && j > (n-i-1)) {
                sumSouth += mat[i][j];
            } else if (i > j &&  j < (n-i-1) ) {
                sumWest += mat[i][j];
            }
        }
    }

    sumsArray[0] = sumNord;
    sumsArray[1] = sumEast;
    sumsArray[2] = sumSouth;
    sumsArray[3] = sumWest;
```

```
        for(int i = 0; i < 4; i++) {
            for (int j = 0; j < 3;j++) {
                if(sumsArray[j] > sumsArray[j+1]) {
                    int temp = sumsArray[j];
                    sumsArray[j] = sumsArray[j+1];
                    sumsArray[j+1] = temp;
                }
            }
        }

        for(int i = 0; i < 4; i++) {
            cout << sumsArray[i] << " ";
        }
        return 0;
    }
```

5. Give a matrix with n rows and n columns and natural elements, create a C++ program which will determine how many elements of the matrix are on rows and columns which have the same sum.

    - Sample Input:

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

    - Sample Output: 2

    - Solution:

```
#include <iostream>
using namespace std;

const int n = 5;
int main()
{
    int mat[n][n]={
            {3,1,8,5,4},
            {7,8,5,1,2},
            {2,2,6,7,3},
            {9,8,1,3,6},
            {7,5,3,1,7}
    };

    int rowsSum[n] = {0};
    int columnsSum[n] = {0};
    int count = 0;
    for (int i = 0; i < n; i++) {
```

```
            for(int j = 0; j < n; j++) {
                rowsSum[i] += mat[i][j];
                columnsSum[i] += mat[j][i];
            }
        }

        for (int i = 0; i < n; i++) {
            for(int j = 0; j < n; j++) {
                if(rowsSum[i] == columnsSum[j]) {
                    count++;
                }
            }
        }
        cout << count;
        return 0;
    }
```

6. Given a matrix with n rows and n columns, create a C++ program which will sort each matrix row, ascending, and then, each column ascending.

    - Sample Input:

```
12 7 1 8
20 9 11 2
15 4 5 13
3 18 10 6
```

    - Sample Output:

```
1 5 8 12
2 6 10 15
3 7 11 18
4 9 13 20
```

    - Solution:

```
    #include <iostream>
    using namespace std;

    const int n = 4;
    int main()
    {
        int mat[n][n]={
                {12,7,1,8},
                {20,9,11,2},
                {15,4,5,13},
                {3,18,10,6}
```

```
        };

        for(int i =0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for(int k = 0; k < n-1; k++) {
                    if(mat[j][k] > mat[j][k+1]) {
                        int temp = mat[j][k];
                        mat[j][k] = mat[j][k+1];
                        mat[j][k+1] = temp;
                    }
                }
            }
        }

        for(int i =0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for(int k = 0; k < n-1; k++) {
                    if(mat[k][j] > mat[k+1][j]) {
                        int temp = mat[k][j];
                        mat[k][j] = mat[k+1][j];
                        mat[k+1][j] = temp;
                    }
                }
            }
        }

           for(int i = 0; i< n; i++) {
               for (int j = 0; j < n; j++) {
                   cout << mat[i][j] << " ";
               }
               cout << endl;
           }

           return 0;
       }
```

7. Create a C++ program which will read a matrix with n rows and n columns from the keyboard and then will display all the values which are strictly greater than their neighbors.
    - Sample Input:

```
n = 4;
1 5 1 1
2 1 2 3
1 3 4 2
2 1 2 1
```

    - Sample Output: 5 2 3 4 2

- Solution:

```
#include <iostream>
using namespace std;

bool isCurrentElementHigherThanNeighbors(int mat[20][20], int n, int
i, int j);

int main()
{

    cout<<"Enter the dimension of the matrix: ";
    int n;
    cin >> n;
    int mat[20][20] = {0};

    for(int i = 0; i < n;i++) {
        for (int j = 0; j < n; j++) {
            cout << "Enter element Mat["<<i<<","<<j<<"]: ";
            cin >> mat[i][j];
            cout << endl;
        }
    }

    for(int i = 0; i < n;i++) {
        for (int j = 0; j < n; j++) {
            if(isCurrentElementHigherThanNeighbors(mat, n, i, j)){
                cout << mat[i][j]<<" ";
            }
        }
    }

    return 0;
}

bool isCurrentElementHigherThanNeighbors(int mat[20][20], int n, int
i, int j) {
    if(i -1 >= 0  && mat[i-1][j] > mat[i][j]) {
        return false;
    }

    if(j+1 < n && mat[i][j+1] > mat[i][j]){
        return false;
    }

    if(i+1 < n && mat[i+1][j] > mat[i][j]) {
        return false;
    }

    if(j-1 >= 0 && mat[i][j-1] > mat[i][j]) {
        return false;
    }
```

```
        return true;
    }
```

8. Given a square matrix with n rows and n columns, create a C++ program which will update the matrix as follows: all the elements which are on rows that contain the maximum value from the matrix, will be increased with the minimum value from the matrix.

    - Sample Input:

```
n = 4
2 5 3 2
2 2 4 4
3 2 2 2
5 3 5 2
```

    - Sample Output:

```
4 7 5 4
2 2 4 4
3 2 2 2
7 5 7 4
```

    - Solution:

```
#include <iostream>
using namespace std;

const int n = 4;

int main()
{
    int mat[n][n]={
            {2,5,3,2},
            {2,2,4,4},
            {3,2,2,2},
            {5,3,5,2}
    };

    int min = mat[0][0];
    int max = mat[0][0];
    int rowsContainingMax[n] = {0}; // 0 means it does not contain
the max, 1 means it contains the max

    for(int i = 0; i < n;i++) {
        for (int j = 0; j < n; j++) {
            if(mat[i][j] < min) {
                min = mat[i][j];
```

```
                }

                if (mat[i][j]> max) {
                    max = mat[i][j];
                }
            }
        }

        for(int i = 0; i < n;i++) {
            for (int j = 0; j < n; j++) {
                if(mat[i][j] == max) {
                    rowsContainingMax[i] = 1;
                    break;
                }
            }
        }

        for(int i = 0; i < n;i++) {
            for (int j = 0; j < n; j++) {
                if(rowsContainingMax[i] == 1) {
                    mat[i][j] += min;
                }
            }
        }

        for(int i = 0; i < n;i++) {
            for (int j = 0; j < n; j++) {
                cout << mat[i][j] << " ";
            }
            cout << endl;
        }

        return 0;
    }
```

9. Given a square matrix with `n` rows and `n` columns and natural elements smaller than 1000, create a C++ program which will display in ascending order, the elements placed below the main diagonale and above the second diagonale. If a value appears multiple time in that area, it will be displayed only once.
    - Sample Input:

```
n = 6

10 8 5 8 4 2
6 5 3 1 3 8
8 1 4 7 8 8
5 1 9 6 6 1
8 9 10 1 3 6
8 2 3 3 9 6
```

- Sample Output: 1 5 6 8
- Solution:

```
#include <iostream>
using namespace std;

const int n = 6;

int main()
{
    int mat[n][n]= {
            {10, 8, 5,  8, 4, 2},
            {6,  5, 3,  1, 3, 8},
            {8,  1, 4,  7, 8, 8},
            {5,  1, 9,  6, 6, 1},
            {8,  9, 10, 1, 3, 6},
            {8,  2, 3,  3, 9, 6}
    };

       int elements[n*n/2] = {0};
       int matchingElementsSize = 0;
       for(int i = 0; i < n;i++) {
           for (int j = 0; j < n; j++) {
               if ( i > j && j < (n-i-1)) {
               elements[matchingElementsSize++] = mat[i][j];
               }
           }
       }

       // Sort elements which correspond to the problem's statement
       for(int i =0; i < matchingElementsSize; i++) {
           for (int j = 0; j < matchingElementsSize-1;j++) {
               if(elements[j] > elements[j+1]) {
                   int temp = elements[j];
                   elements[j] = elements[j+1];
                   elements[j+1] = temp;
               }
           }
       }

       // Eliminate duplicates (if any)
       for(int i = 0; i <matchingElementsSize -1; i++) {
           if(elements[i] == elements[i+1]) {
               for(int j = i+1; j< matchingElementsSize; j++) {
                   elements[j] = elements[j+1];
               }
               matchingElementsSize--;
               i--;
           }
       }

       // Display the elements
```

```
            for(int i = 0; i < matchingElementsSize; i++) {
                cout << elements[i] <<" ";
            }

            return 0;
        }
```

10. Given a square matrix with n rows and n columns and natural elements which are smaller than 1000, create a C++ program which will display in ascending order, the values which appear below the main diagonal and below the second diagonal at least two times. Each value will be displayed only once.

    - Sample Input:

```
n = 6
10 8 5 8 4 2
6 5 3 1 3 8
8 1 4 7 8 8
5 1 9 6 6 1
8 9 3 2 3 6
8 9 3 3 9 6
```

    - Sample Output: 3 9

    - Solution:

```
#include <iostream>
using namespace std;

const int n = 6;

int main()
{
    int mat[n][n]= {
            {10, 8, 5,  8, 4, 2},
            {6,  5, 3,  1, 3, 8},
            {8,  1, 4,  7, 8, 8},
            {5,  1, 9,  6, 6, 1},
            {8,  9, 3, 2, 3, 6},
            {8,  9, 3,  3, 9, 6}
    };

    int matchingElementsFrequency[1000] = {0};

    for(int i = 0; i < n;i++) {
        for (int j = 0; j < n; j++) {
            if(i > j && j > (n - i - 1)) {
                matchingElementsFrequency[mat[i][j]]++;
            }
        }
```

```
        }

        for(int i = 0; i < 1000; i++) {
            if(matchingElementsFrequency[i] >= 2) {
                cout <<i<<" ";
            }
        }

        return 0;
    }
```