Multidimensional arrays

Plan a: Locate a row and column in which the sum of the values is maximum

Write a plan that defines an array of integers between ROWS rows and COLS columns (set the number of rows to be four and the number of columns to five).

The program will read data to the array in a cross-section of lines and then print it out:
A. The row number in which the sum of the values is the highest
B. The number of the column whose sum of values is the highest.

It can be assumed that the values in the array are really positive.

If there is more than one row or column with the maximum amount, you must display the first row or column.

Plan b: Equilateral triangles in a two-dimensional array

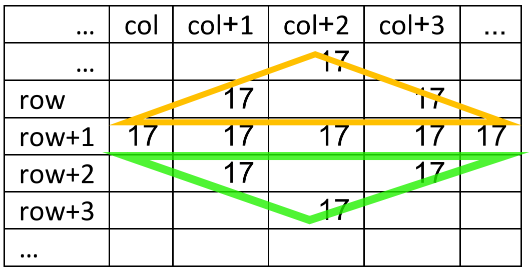
Write a plan that defines an array of integers between ROWS rows and COLS columns (set the number of rows to be 10 and the number of columns to 15). The program will read array data in a cross-section of rows.

In a two-dimensional array of integers we will define an isosceles triangle to be a set of cells, all of which have the same value, and they are arranged in an array in the form of an isosceles triangle as described below:

Equilateral triangle: A cell [row, col] containing the value X is the summit of a triangle triangle if two diagonals of equal length y emerge from it, containing the same value X, and a base that horizontally connects the two diagonals and contains the entire value X in sequence.

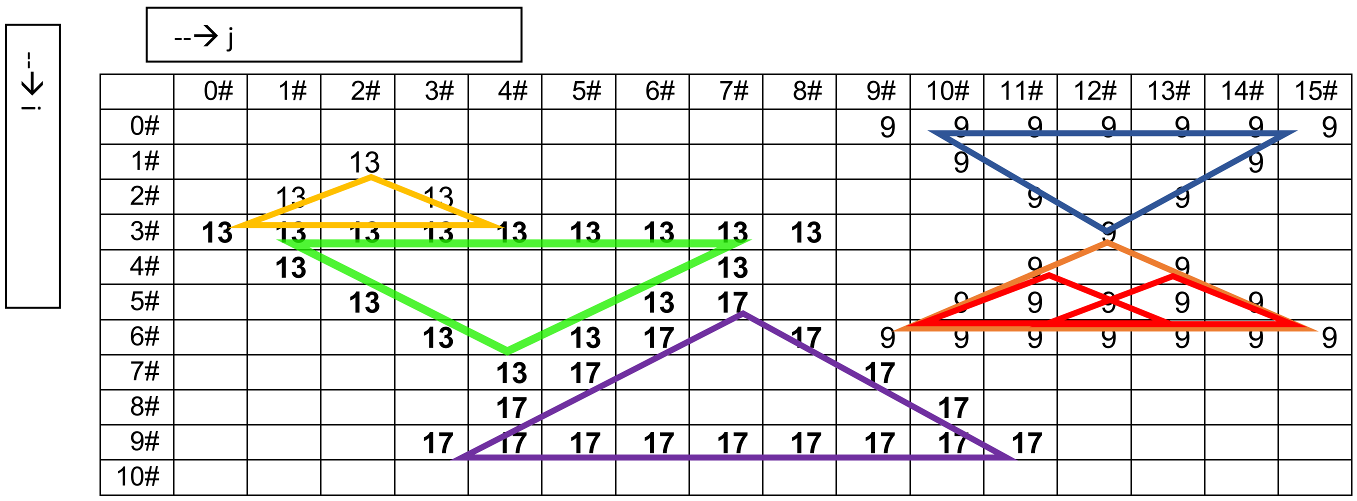
important comments:
.1The diagonals can come down from the summit point and then the triangle stands, or they come up and then the triangle is upside down.
.2The base side must be horizontal on the array and therefore we will not look for diagonal SH triangles even though they are possible.
.3The size of the triangle triangle will be determined by the size of the double side, not by the base.
.4Although it is possible to have a size triangle, we will define that the smallest triangle is size 3. (To avoid counting inner triangles - since each larger triangle contains 2 size triangles).
.5A single cell can form the basis of both a standing triangle and an inverted triangle.
.6If no triangles are found as required, zeros must be presented as described below - at the end of the question.

Example: The following cells in the array form two isosceles triangles:
Standing triangle: In the cell [row] [col + 2] is the apex of an equilateral triangle with 2 diagonals and a horizontal series of 5 cells that extend continuously on a row +1 row, and all have the same value.
Inverted triangle: The horizontal series of five cells that extend consecutively on a row +1 row, is also the base of an inverted isosceles triangle that is apexed in the cell [row + 3, col + 2]



...	col	col+1	col+2	col+3	...
...			17		
row		17		17	
row+1	17	17	17	17	17
row+2		17		17	
row+3			17		
...					

לדוגמה, במערך:



	0#	1#	2#	3#	4#	5#	6#	7#	8#	9#	10#	11#	12#	13#	14#	15#
0#										9	9	9	9	9	9	9
1#			13								9				9	
2#		13		13								9		9		
3#	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**				9			
4#		**13**						**13**				9		9		
5#			**13**				**13**	**17**			9	9	9	9	9	
6#				**13**		**13**	**17**		**17**	9	9	9	9	9	9	9
7#					**13**	**17**				**17**						
8#					**17**						**17**					
9#				**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**				
10#																

There are seven triangles that are total. (Assuming the empty cells have different random numbers that do not form other triangles as well. In the example we left the cells empty just for clarity)
The largest triangles are size 5, and there are two such. The first is at the top in cell [5] [7] and is painted purple, and the second in cell [7] [4] and is painted green.
In cell [12] [3] we see a base point common to two triangles where one is standing and one is inverted. Of course the information has to be displayed for the two triangles.
We also see that this point is the summit of two standing triangles. One in size 3 and one in size 4. In this case we will ignore the smaller triangle and print only the details for the larger triangle.
The program must locate and display the following information:
First, a list with the row number and column number of the summit point of each triangle we found as well as the size of the triangle, its description: standing or inverted standing \ inverted followed by the value indicating the boundaries of the triangle.
After the list of triangles we will present the following information in one line:
A. Several maximum-sized triangles exist in the array.
B. Maximum triangle size.

In the example above, the output will be:

```
1 2: 3 standing. Value: 13
3 12: 4 standing. Value: 9
3 12: 4 inverted. Value: 9
4 11: 3 standing. Value: 9
4 13: 3 standing. Value: 9
5 7: 5 standing. Value: 17
7 4: 5 inverted. Value: 13
2 5
```

If no triangles are found as required two zeros should be displayed as follows: 0 0.

Notice exactly how the information should be printed. (As shown above) Row number, space, column number, colon, space, triangle size, space, triangle type (one of the following words: standing, inverted), dot, print "Value:" (note the colon and space), value Indicating the boundaries of the triangle, new line.
Explain in the documentation what the running time of the program is

Plan c: Finding a pattern according to rules

Write a plan that defines an array of integers between ROWS rows and COLS columns (set the number of rows to be 5 and the number of columns to 17).
The program also defines a one-dimensional array of size N (set N to be 6) that will be called the "set of rules" containing enum pattens_t values defined as follows:

```
enum patterns_t
{   EVEN_PT, POSITIVE_PT, DIVIDE_BY_3_MPT};
```

The values in the one-dimensional array indicate the "rules" of a particular template that we want to find:
EVEN_PT: It means even numbers.
POSITIVE_PT: It means positive numbers (greater than zero)
DIVIDE_BY_3_MPT: It means numbers divided by three.
The program will first call data to the two-dimensional array, then it will call data to the "rules array" and then it will call a function called find_patterns.
The user will enter data as follows:
First ROWS * COLS data for the two-dimensional array (to be read in a cross-section of rows).
The user will then enter N-1 values to read into the rule set: zero will indicate that he is looking for EVEN_PT, 1 will indicate that he is looking for POSITIVE_PT, 2 will indicate that he is looking for DIVIDE_BY_3_MPT.
Example of input to be fed into a set of rules (if we set N to be 6): 0 1 0 1 2
The find_patterns function accepts the following parameters:
A two-dimensional array of integers, consisting of ROWS rows and COLS columns, which already includes data.
.2One-dimensional array (size N) - "Rules set" containing enum pattens_t values.
)And additional parameters as described below and at your discretion.(
The function must look for a row in the matrix that meets the rules as they appear in the set of rules in a quantity that matches the index in the set of rules.

For example for the following set of rules:

#0	#1	#2	#3	#4	#5
	EVEN_PT	POSITIVE_PT	EVEN_PT	POSITIVE_PT	DIVIDE_BY_3_MPT

The function will look for a row that has it (not necessarily at the beginning, but wherever there is a sequence of cells that includes values as described):

1. One even number. (Women note that cell # 0 is not used, we are only interested in cells # 1 and above. Since cell number one has all the pairs we are looking for one pair.)
2. Immediately afterwards (i.e. in cells immediately after the even number) -> 2 positive numbers. (Because Index 2 has the positive rule so we will look for it to be applied to two numbers).
3. Immediately afterwards (i.e. in cells after the two positive numbers) -> 3 even numbers. (Because in index 3 there is again the rule of pairs and we will look for it to be applied to three numbers).
4. Immediately afterwards -> 4 positive numbers.
5. Immediately afterwards -> 5 numbers divided by three.

The return value of the function will be the sum of the patterns found in the matrix. The function in the reference parameter will also return the maximum amount of values in the format found.
The program will print as output the return value of the function followed by (with a single gain) the maximum amount returned in the reference parameter.
We will continue the example we started. Given the set of rules above and the following matrix:

	#0	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15	#16
#0	3	-1	3	3	3	2	3	3	3	3	3	3	3	3	3	3	3
#1	2	1	3	2	3	4	0	0	7	7	6	77	6	6	66	6	0
#2	1	-2	3	2	4	-8	252	6	7	8	9	-6	6	9	-6	9	-4
#3	1	3	5	-5	5	3	17	12	12	3	-2	5	7	9	12	66	7
#4	12	3	24	4	8	-2	12	3	3	3	3	3	3	3	3	3	3

It can be seen that there are two templates that correspond to the given set of rules:
1. The first begins in cell [2] [1]. The sum of the values in the template: 293.
The logic of the division is:

#0	#1	#2	#3	#4	#5
	EVEN_PT	POSITIVE_PT	EVEN_PT	POSITIVE_PT	DIVIDE_BY_3_MPT
	-2	3 2	4 -8 252	6 7 8 9	-6 6 9 -6 9

2. The second begins in cell [4] [0]. The sum of the values in the format: 85.
The logic of the division is:

#0	#1	#2	#3	#4	#5
	EVEN_PT	POSITIVE_PT	EVEN_PT	POSITIVE_PT	DIVIDE_BY_3_MPT
	12	3 24	4 8 -2	12 3 3 3	3 3 3 3 3

In the call to find_patterns function given the matrix and set of rules above the return value of the function will be 2 (because two templates were found), in the reference parameter the value 293 will return which is the maximum sum of the values in the template.

Additional Comments:
• If no suitable templates are found, the return value will be 0, in the reference parameter the value 1- will return.
• We will define that N (the size of the set of rules) is equal to 2. (When as stated from cell # 0 we ignore).

• It is possible that a certain value (in a certain box in the matrix) will belong to several different patterns in different places in the pattern.

Take care of the modularity.