TP6 Arrays

Ex 6.1 Separate odd and even numbers

We have a one-dimensional array of integers T.

1. Write a function **EnterTab** that takes three parameters as input; an array T of integers, an integer N that will store the size of the array and an integer NMAX. The function reads the size N and the elements of the array T. The size N must be less than NMAX.

Example :

After calling **EnterTab** (T, &N, 10); the function will behave as follows:

```
Give the size of the array (max = 10): 11
The size of the array must be less than 10.
Give the size of the array (max = 10): 4
    T[0] : 17
    T[1] : 35
    T[2] : 23
    T[3] : 9
```

2. Write a function **OddEven** that takes three arrays as input T, E, and O and then fills the two arrays, E with even numbers and O with odd numbers.

Example : Let the following array T:

T :	-1	9	5	2	0	-3	12	-6	7	1

The function **OddEven** fills the two arrays E and O as follows:

E :	2	0	12	-6

O :	-1	9	5	-3	7	1

3. Write a function **DisplayTab** with two parameters Tab and n, which displays the n elements of the array of integers Tab.
4. Write a main program that enters the array T and displays the two arrays E and O.

Ex 6.2 Two-dimension array

1. Write a fonction **ReadDim** with four parameters L, LMAX, C, CMAX that reads the dimensions L and C of a two-dimensional matrix. The dimensions L and C must be less than LMAX respectively CMAX.
2. Write a function **ReadMatrix** with three parameters MAT, L, and C that reads the components of a matrix MAT of integers and dimensions L and C.
3. Write a function **DisplayMatrix** with three parameters MAT,L and C which displays the components of the matrix of dimensions L and C.
4. Write a function **TranspoMatrix** with five parameters MAT, L, LMAX, C, CMAX which transposes the matrix MAT using the **swap** function. The function **TranspoMatrix** returns a logical value indicating whether the dimensions of the matrix are such that the transposition can be performed.
5. Write a program that allows you to test all of the above functions.
6. Rewrite the above two functions **ReadMatrix** and **DisplayMatrix** in pointer notation (choose the necessary parameters carefully).

Ex 6.3 Pascal's triangle

Write a function that constructs Pascal's triangle of degree N and stores it in a two-dimensional array *P*. The function should also display the Pascal's triangle exactly as shown in the example below:

Example : N = 6 :

```
i=0 :    1
i=1 :    1  1
i=2 :    1  2  1
i=3 :    1  3  3  1
i=4 :    1  4  6  4  1
i=5 :    1  5  10 10 5  1
i=6 :    1  6  15 20 15 6  1
```

Method :

- Calculate and display only values up to the main diagonal (included).
- The values of the first column and the main diagonal are equal to 1.
- The other values are calculated from left to right using the following relationship:

$$P_{i,j} = P_{i-1,j} + P_{i-1,j-1}$$

Ex 6.4 Magic squares

A magic square is a square matrix of size n x n such that the sum of each row, each column, and each main diagonal is equal. A magic square is said to be normal if it contains each integer between 1 and n^2 exactly once.

Example : The following matrix is a normal magic square:

6	7	2
1	5	9
8	3	4

1. Write a function **square** that returns **1** if the array passed as a parameter with its dimensions is a square matrix (which has as many rows as columns), **0** otherwise.
2. Write two functions **SumRow** and **SumColumn** that take as input an array and a row number (or column number) and return the sum of its the elements.
3. Write two functions **SumDiagonal** and **SumAntiDiagonal**, which return the sum of the diagonal (respectively the antidiagonal) of the array passed as a parameter.
4. Write a function **MagicSquare** that returns **1** if the array passed as a parameter is a normal magic square and **0** otherwise.
5. Write a program that asks the user to enter an array, and displays whether it is a normal magic square.