# **Introduction to programming** `(NF05A)` — **exercices** 2020-2021 —

*Université de technologie de Troyes*

# Tutorial 4

## Exercise 1. Array operations

1. Write a procedure to enter a vector of positive real elements of size $N$ (the value of $N$ will be asked to the user and the array is allocated dynamically)
2. Write three functions to calculate :
   - The sum of two vectors entered by the user.
   - The subtraction of these two vectors.
   - The scalar product of these vectors.
3. Write a procedure that displays the result of the operations.

## Exercise 2. Pointers and structures

1. Give the declaration of the following structure:
   **Student**
   - first and last name.
   - Midterm exam grade.
   - Final exam grade.
   - The average grade.
2. Using a dynamic array of type **Student**, write a program to enter a list of $N$ students (the value of $N$ will be entered by the user). The average grade is given by weighting by 0.4 and 0.6 the median and final grades, respectively.
3. Write a procedure/function to sort this list of students in ascending order according to the first and last name field. Add another one that sorts the list according to the average grade in descending order.

# Additional exercises - To go further

### Exercise 3. Matrix operations

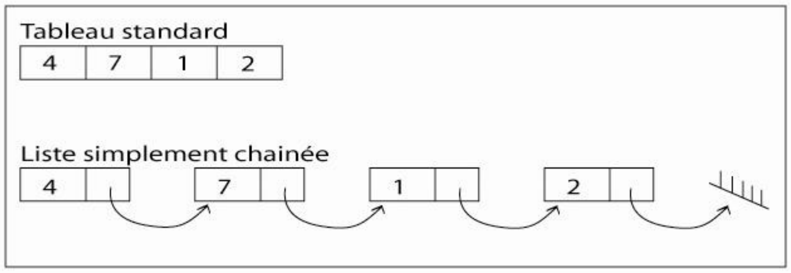
Write a procedure to read two matrices of any size (the user enters the size) and then the function to add, subtract and multiply the two matrices.

Write a procedure that allows to return the transpose of a matrix of any size (the user enters the size).

### Exercise 4. Linked lists

The most commonly used data structure is the array. An array becomes impractical if the elements of the array are subject to change during the program. A deletion of an element in the middle will force the user to shift all elements that come after the deleted element to keep the data structure consistent.

A linked list is a different data structure in the sense that the elements of the list are distributed in the memory and linked together by pointers. You can add and remove elements from a linked list at any point, at any time, without having to recreate the entire list.



The list is characterized by a pointer that points to the first element. The list is empty at the beginning, that is, it contains no element and the pointer is set to `NULL`. An element of the list is composed of the data and a pointer that points to the next element. The last element has no successor and its pointer is set to `NULL`. To simplify the implementation, we consider that each element contains an integer.

1. Write the structure `Element` of an element of the list and define the type `List`
2. Write the procedure `void insertElement(List l, struct Element e)` which allows to insert the element `e` in the list `l`.

3. Write the function `struct element * successor(List l, struct Element e)` which returns a pointer to the successor of the element `e`. If the element is the last or does not exist in the list, the function must return `NULL`

4. Write the function `struct element * predecessor(List l, struct Element e)` which returns a pointer to the predecessor of the element `e`. If the element is the first or does not exist in the list, the function must return `NULL`

5. Write the procedure `void deleteElement(List l, struct Element e)` which allows to remove the element `e` from the list. The list remains unchanged if the element does not exist in the list.

6. Write the procedure `void sortList(list L)` that sorts the list in descending order.

7. Modify the function `insertElement` to allow an ordered insertion.