Cette *seizième* colle vous fera écrire un programme en OCaml implémentant le tri par dénombrement, et deux programme en C manipulant leurs arguments en ligne de commande.

On travaillera depuis la machine virtuelle ClefAgreg2019, et on compilera les fichiers écrits avant d'exécuter les binaires produits.

Ex.1 Tri par dénombrement - OCaml (30 minutes)

Le tri par dénombrement peut être utilisé dans le cas où l'on doit trier des entiers positifs, pas nécessairement distincts, dont on connaît un majorant. Il est efficace si ce majorant n'est pas trop grand par rapport au nombre d'entiers à trier. Son principe est le suivant : on parcourt les données et pour chaque valeur qui apparaît, on maintient à jour un compteur qui nous permet de savoir combien de fois chaque valeur a été vue. Avec ces nombres d'occurrences, on peut à la fin reconstruire les données dans l'ordre croissant.

1. Écrire un programme OCaml qui lit sur l'entrée standard un entier naturel $n \leqslant 1000$ puis n entiers naturels dans $[0; 1000]$ et qui affiche sur la sortie standard ces n entiers triés dans l'ordre croissant en utilisant le tri par dénombrement proposé ci-dessus.

On rappelle que l'on peut lire une ligne depuis le clavier, comme une chaîne de caractères, avec `let ligne = input_line stdin in ...`, et la convertir en un entier avec `let entier = int_of_string ligne in ...`. On utilisera un tableau qui permet de compter les occurrences de chaque entier vu au fur et à mesure de leur lecture, puis on affichera à la fin le résultat du tri.

2. Si le majorant sur les entiers est fixé égal à k une constante, et qu'il y a n entiers à trier, quelle est la complexité temporelle asymptotique de cet algorithme de tri par dénombrement en fonction de k et n ? (exprimé comme un $\mathcal{O}(\dots)$)

Ex.2 Sur des tableaux en C - C (25 minutes)

Écrire une fichier C `main_qX.c` par question ($X = 1, 2, 3$), important `stdio.h` (pour `printf` et les fonctions sur les fichiers) et `stdlib.h` (pour `EXIT_SUCCESS` et `EXIT_FAILURE`).

On rappelle qu'on compile ce fichier avec `COMPILATEUR = gcc` ou `clang` avec la ligne de commande suivante, puis on exécute le binaire produit avec la deuxième ligne (sans les dollars qui représentent le prompt de la ligne de commande du terminal) :

```
$ COMPILATEUR -O0 -Wall -Wextra -Wvla -Werror -fsanitize=address
  -fsanitize=undefined -pedantic -std=c11 -o main.exe main.c
```

1. Écrire un programme C qui lit sur l'entrée standard un entier naturel $n \leqslant 1000$ puis n entiers relatifs, et qui affiche sur sa sortie standard ces n éléments mais *dans l'ordre inverse d'apparition.*
2. Écrire un programme C qui lit sur l'entrée standard un entier naturel $n \leqslant 1000$, puis n flottants $x_1, x_2, \dots, x_n$, puis encore n flottants $y_1, y_2, \dots, y_n$ et qui affiche la valeur absolue des différences, terme à terme, de ces flottants, c'est-à-dire $|x_1 - y_1|, |x_2 - y_2|, \dots, |x_n - y_n|$. On définira sa propre fonction de prototype `double abs_float(double x)`.

- a. Pour lire ces flottants, on utilisera le spécificateur de format `%lf` avec `scanf` pour lire un *long float* ou `double`.
- b. Cette fonction valeur absolue `abs_float` existe sous le nom `fabs` dans la bibliothèque `math`.

3. Écrire une fonction `int maximum(int n, int m, int mat[n][m])` qui renvoie la valeur maximale d'une matrice de taille $n \times m$ avec $n, m \geqslant 1$.

