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

1. Sérialisation en OCaml - 50 minutes

On rappelle que l'on peut compiler avec `COMPILATEUR = ocamlc` ou `= ocamlopt` et la ligne de commande suivante, et ensuite que la seconde ligne permet de l'exécuter.

```
$ COMPILATEUR -o serialisation.exe serialisation.ml
$ ./serialisation.exe 10 fichier_sortie.txt
```

1. Écrire une fonction `ecrire_fichier (nom_fichier: string) (chaine: string) -> unit` qui écrit la chaîne donnée en argument dans le fichier identifié par son nom (ou son chemin relatif ou absolu). On rappelle que l'on ouvre un fichier avec `let fichier_out = open_out nom_fichier in ...` qui donne un objet de type `out_channel`, sur lequel on peut utiliser `output_string fichier_out chaine` pour écrire une chaîne dans ce fichier. On fermera ensuite le fichier avec un appel à `close_out fichier_out`.
2. Écrire une fonction `main`, qui sera appelée ensuite en fin de fichier, qui prend son premier argument et le convertit en un entier (avec `int_of_string Sys.argv.(1)`) et écrit dans le fichier dont le nom est donné en deuxième argument du binaire la chaîne `"[0;1;...;n-1]"` (sans les points de suspension évidemment). On pourra utiliser et abuser de `ecrire_fichier` sans chercher à être économe en son nombre d'appel.
3. Tester et vérifier que l'appel à la ligne de commande suivante produit bien un fichier `fichier_sortie.txt` qui contient `[0;1;2;3;4;5;6;7;8;9]` :

```
$ ./serialisation.exe 10 fichier_sortie.txt
$ cat fichier_sortie.txt
[0;1;2;3;4;5;6;7;8;9]
```

4. (bonus, plus long et assez difficile) On travaillera depuis un autre fichier. On cherche à réaliser l'opération de sérialisation inverse, à savoir écrire une fonction qui permet d'obtenir une liste d'entiers depuis un fichier dans lequel cette liste est écrite au format précédent.
   - Écrire une fonction `lecture_liste (nom_fichier: string) -> int list` qui lit depuis le fichier une liste d'entiers, formatée comme précédemment.
   - On rappelle que l'on ouvre un fichier en lecture avec `let fichier_in = open_in nom_fichier in ...` qui donne un objet de type `in_channel`, sur lequel on peut utiliser `let ligne_lue = input_line fichier_in in ...` pour lire une ligne et obtenir une chaîne de caractère. Après avoir lu une seule ligne, on fermera le fichier avec `close_in fichier_in;`.
   - On commencera par créer une fonction `count (chaine: string) (car: char) -> int` qui compte le nombre de présence du caractère `car` dans la chaîne donnée en argument. Appliquée à la `ligne_lue` avec le `char car = ';'` cela permet de trouver la longueur de la chaîne (en comptant le nombre de `;` plus un).

— Ensuite on peut utiliser deux fonctions du module `String` pour faire des recherches et des extractions de sous-chaîne : `String.index_from chaine position car` qui trouve la position du `car` à partir d'une position de départ (on cherchera `;` sauf si on est au dernier entier auquel cas on cherchera `]`), et ce pour trouver la position de fin de l'entier en cours de lecture ; et `String.sub chaine position (position_fin - position)` pour extraire la sous-chaîne. La dite sous-chaîne peut se convertir en un entier avec simplement `int_of_string`, et on rajoute l'entier ainsi obtenu dans une (référence vers une) liste, qui sera renversée par un `List.rev` à la fin (car on ajoute au début pour rester en temps linéaire).

— Enfin on écrira une fonction `main` (qui sera appelée en fin de fichier) On écrira aussi une fonction pour afficher sur la sortie standard une liste d'entiers, afin de vérifier que la fonction `lecture_liste` est correcte.

---

2. Des tableaux en arguments en C - 30 minutes

Écrire une fichier C `TP13.c` 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). Cette fois, on enlèvera le warning `-Wvla` car certaines signatures de fonctions seront des VLA (*variable-length array*).

```
$ COMPILATEUR -O0 -Wall -Wextra -Werror -fsanitize=address
  -fsanitize=undefined -pedantic -std=c11 -o TP13.exe TP13.c
```

> *Rappel :* Quand on passe un tableau en paramètre à une fonction, ce n'est pas l'ensemble du tableau qui est transmis, mais simplement l'adresse de la première case du tableau. Il n'y a donc pas de copie du tableau (comme c'est le cas avec une variable scalaire) et la fonction peut agir directement sur les cases du tableau.

5. Écrire en C, une fonction de prototype `void affiche_tableau(int n, int* tab)` qui affiche un tableau de taille `n`, dont on passe l'adresse de la première case en paramètre. Rappelons que même si ici `tab` est un pointeur, `tab[i]` désigne la valeur à l'indice `i`, comme pour un tableau. Vérifier avec le main suivant :

```
int main(void) {
    int tab[5] = {4, 3, 1, 6, 3};
    affiche_tableau(5, &tab[0]);
    affiche_tableau(5, tab);
}
```

6. Expliquer pourquoi les deux appels à `affiche_tableau` reviennent au même. Quelle syntaxe préférez-vous ?

7. Modifier la signature en `void affiche_tableau(int n, int tab[])`. Vérifier et commenter. Modifier la signature en `void affiche_tableau(int n, int tab[n])`. Quelle syntaxe préférez-vous et pourquoi ? Vérifier et commenter. Est-il possible d'avoir le prototype `void affiche_tableau(int tab[n])` ?
8. Essayer d'écrire une fonction `int* creer_tableau()` qui tente de créer un tableau de taille `1000` dont la case d'indice `i` est initialisée avec l'entier `i` (comme un `range` en Python) et qui renvoie ce tableau. Que se passe-t-il ? Essayer de renvoyer plutôt un pointeur vers la première case du tableau. Que se passe-t-il ? Si besoin, exécutez votre programme sur C Tutor (`http://pythontutor.com/c.html#mode=edit`) et assurez-vous d'avoir bien compris le problème et pourquoi cette approche ne peut pas fonctionner.

> Un fonction ne peut pas renvoyer un tableau. On peut renvoyer un pointeur vers la première case du tableau, mais il faut alors que le tableau soit alloué en dehors de la fonction, ou sur le tas avec un 'malloc'.

9. Écrire une fonction `int* initialise_tableau(int n, int tab[n])` qui initialise les cases d'un tableau de taille `n` passé en paramètre de manière à avoir l'entier `i` dans la case d'indice `i`. Vérifier et tester.
10. Expliquer pourquoi on pourrait également, pour cette fonction, utiliser une signature `void initialise_tableau(int n, int tab[n])` et ce que cela change. Laquelle préférez-vous ?

---

3. Tableaux à plusieurs dimensions en C - 35 minutes

Un tableau à plusieurs dimensions en C est aplati en un tableau de tableau, et on peut accéder à la case d'indice (i, j) avec la notation `mat[i][j]`, si `mat` est déclaré comme un tableau de taille `n * m` avec `int mat[n][m]` dans la signature de la fonction.

11. Écrire une fonction `void initialise_matrice(int n, int m, int mat[n][m])` qui initialise une matrice avec la valeur `i + j` pour la case de coordonnées `(i, j)`, pour $0 \leqslant i < n$ et $0 \leqslant j < m$.
12. Écrire une fonction `void affiche_matrice(int n, int m, int mat[n][m])` qui affiche une matrice de taille `n * m`.
13. Écrire une fonction `void somme_matrice(int n, int m, int mat1[n][m], int mat2[n][m], int res[n][m])` qui effectue la somme terme à terme de tous les éléments de deux matrices `mat1` et `mat2` passés en paramètres et qui range le résultat dans la matrice `res` passé en paramètre. Tester et vérifier.
14. Pourquoi a-t-il été nécessaire de passer la matrice `res` en paramètre supplémentaire ? Peut-on faire autrement ?

