Cette *treizième* colle n'est pas maudite. Elle vous fera écrire un programme OCaml manipulant ses arguments en ligne de commande, et un programme en C affichant son propre code source (un quine).

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

Ex.1 Réimplémentation en OCaml de la commande `echo` - OCaml (20 minutes)

1. Dans le module `Sys`, il y a la variable `Sys.argv` qui est de type `string array` et contient dans un tableau les arguments de la ligne de commande ayant servi à appeler le binaire. Le premier argument est le nom du dit binaire, et les suivants sont les arguments donnés à ce binaire.

On travaillera dans un fichier `echo.ml`.

2. Écrire une fonction `print_each: string array -> unit` qui à l'aide de `print_string: string -> unit` et `print_newline: unit -> unit` affiche chaque argument de `print_each`, séparés par une espace, et avec un retour à la ligne à la fin.
3. Écrire une fonction `main: unit -> unit` qui sera appelée en fin du fichier OCaml, qui utilise `Sys.argv` pour afficher à l'écran tous les arguments (*sauf le premier*). On pourra utiliser `Array.sub un_array debut longueur` qui extraie un sous tableau, et `print_each`.
4. Compiler le fichier en un binaire `echo.exe` et vérifier qu'il fonctionne comme la commande `echo` du terminal.

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 echo.exe echo.ml
$ ./echo.exe arg1 arg2 ... argN
arg1 arg2 ... argN
```

Ex.2 Réimplémentation en C de la commande `cat` et un quine - C (35 minutes)

Écrire une fichier C `colle13.c` important `stdio.h` (pour `printf` et `scanf`) 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 colle13.exe colle13.c
$ ./colle13.exe
```


Si une ligne affiche un résultat qui vous semble bizarre, commenter le.

On travaillera dans un premier fichier `cat.c`.

1. Écrire une fonction `int affiche_fichier(char* nom_fichier)` qui ouvre avec `FILE* fp = fopen(nom_fichier, "r")` le fichier nommé `nom_fichier`, puis qui affiche à l'écran son contenu, caractère par caractère, avec une boucle `while` sur un `c = fgetc(fp)` et `putc(c, stdout)` ou `printf("%c", c)`. On veillera à bien gérer les erreurs possibles lors de l'ouverture ou de la lecture du fichier, avec des tests `if` et des `return EXIT_FAILURE;` en cas d'erreur. On renverra `return EXIT_SUCCESS;` en cas de réussite. On pensera bien à fermer les flux de fichiers ouverts avec `fclose(fp)` (qui lui aussi peut échouer).
2. Écrire une fonction `int main(int argc, char* argv[])` qui appelle `affiche_fichier(argv[i])` sur chaque argument *sauf le premier* du tableau `argv`. Si un de ces appels résulte en une erreur, il faudra quitter directement la fonction `main` avec un `return EXIT_FAILURE;`, sinon on terminera par un `return EXIT_SUCCESS;` à la fin.
3. Compiler le fichier en un binaire `cat.exe` et vérifier qu'il fonctionne comme la commande `cat` lorsqu'il est exécuté avec un nom de fichier comme argument. On pourra créer un petit fichier `petit_fichier.txt` contenant quelques lignes pour l'exemple.

```
$ cat petit_fichier.txt
Lycée
Kléber
$ ./cat.exe petit_fichier.txt
Lycée
Kléber
```

4. Copier-coller le fichier `cat.c` précédent dans un fichier `quine.c` et modifier là où c'est nécessaire pour faire en sorte que l'exécution du binaire `quine.exe` affiche le code source de `quine.c`, sans besoin d'arguments en ligne de commande :

```
$ ./quine.exe
// Solution quine.c colle 13 semaine 10/01/2022
#include <stdio.h>
...
}
```

