Cette *quatorzième* colle vous fera écrire un programme OCaml manipulant un fichier et ses arguments en ligne de commande, et un programme en C manipulant un fichier.

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

Ex.1 Réimplémentations en OCaml des commandes `head` et `tail` - OCaml (25 minutes)

Dans le module `Sys`, il y a la variable `Sys.argv` qui est de type `string array` et contient dans un tableau (de chaîne de caractères) 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 depuis la ligne de commande.

On travaillera dans un premier fichier `head.ml`.

1. Écrire une fonction `head: string -> unit`, qui sera ensuite appelée dans le `main`, qui affiche à l'écran les 10 premières lignes du fichier dont le nom est donné en argument. On pourra utiliser `let fichier_in = open_in nom_fichier in ...` qui ouvre un fichier identifié par son nom ou son chemin (relatif ou absolu) et renvoie un objet de type `in_channel`. En enrobant d'un `try ... with End_of_file -> close_in fichier_in` une boucle `while`, qui lit le fichier ligne après ligne avec `let ligne = input_line fichier_in in ...`, on affichera les 10 premières lignes du fichier, avant de bien penser à le fermer. On rappelle qu'on peut déclencher l'exception manuellement avec `raise End_of_file` (dès que l'on a lu 10 lignes par exemple). Compter ces 10 lignes peut se faire facilement avec une référence entière.
2. Écrire une fonction `main: unit -> unit`, qui sera appelée en fin du fichier OCaml, qui utilise `Sys.argv` pour afficher à l'écran les 10 premières lignes de tous les fichiers dont le nom est donné dans `Sys.argv` à partir de l'indice 1 et suivants.
3. Compiler le fichier en un binaire `head.exe` et vérifier qu'il fonctionne comme la commande `head` 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. On compare avec la commande `head` du terminal, et le dernier `cat` est là pour montrer le contenu de deux fichiers test `fichier_court.txt` et `fichier_long.txt` qui vous pouvez utiliser.

```
$ COMPILATEUR -o echo.exe echo.ml

$ ./head.exe fichier_court.txt
ligne1
ligne2
ligne3

$ head fichier_court.txt
ligne1
ligne2
ligne3
```



```
$ cat fichier_court.txt
ligne1
ligne2
ligne3
```

Et un autre exemple où le ... représente des lignes manquantes :

```
$ ./head.exe fichier_long.txt
ligne1
...
ligne10

$ head fichier_long.txt
ligne1
...
ligne10

$ cat fichier_long.txt
ligne1
...
ligne30
```

4. Que se passe-t-il sur un fichier de moins de 10 lignes ? Vérifier que votre programme s'exécute sans erreur et que son comportement est bien le comportement attendu.
5. (bonus long mais pas très difficile) Faire de même pour implémenter un binaire `tail.exe` qui affiche les 10 dernières lignes du fichier dont le nom est donné en argument du binaire depuis la ligne de commande. On pourra écrire au préalable une fonction `nombre_lignes: string -> int` qui ouvre un fichier identifié par son nom ou son chemin (relatif ou absolu) et calcule son nombre de lignes. Pour cette fonction préalable, on utilisera de même une boucle `while true` entourée d'un `try ... with End_of_file -> ...` et une référence entière pour compter le nombre de ligne. On écrira ensuite une fonction `tail: string -> unit` qui fait comme `head` mais en affichant seulement les 10 dernières lignes du fichier. On a besoin de d'abord compter le nombre de lignes pour savoir quand commencer à afficher les dernières lignes.
6. (bonus assez difficile) Modifier la fonction `main` du fichier `head.ml` pour qu'elle accepte des arguments sous la forme `./head.exe -n13 fichier_long.txt` et affiche non plus 10 mais 13 lignes. On pensera à utiliser `String.sub str debut longueur` et un `int_of_string` sur le premier argument `Sys.arg.(1)` qui est une option sur le nombre de ligne à afficher.

Ex.2 Réimplémentation en C de la commande `wc` - C (30 minutes)

Écrire une fichier C `wc.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) :


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

$ ./wc.exe fichier_court.txt
 5  35 fichier_court.txt
$ wc fichier_court.txt
 5  5 35 fichier_court.txt

$ ./wc.exe fichier_long.txt
 30  231 fichier_long.txt
$ wc fichier_long.txt
 30  30 231 fichier_long.txt
```

Le but de ce binaire `wc.exe` est d'afficher, pour chaque fichier dont le nom ou le chemin (absolu ou relatif) est donné en tant qu'argument de la ligne de commande, le nombre de sauts de lignes `\n` et le nombre de caractères total dans le fichier.

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

1. Écrire une fonction `int wc_sur_fichier(char* nom_fichier)` qui ouvre avec `FILE* fp = fopen(nom_fichier, "r")` le fichier nommé `nom_fichier`, puis qui compte le nombre de caractères et le nombre de saut de ligne de son contenu (avec deux variables entières que l'on incrémentera une par une), caractère par caractère, avec une boucle `while` sur un `c = fgetc(fp)`. 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). A la fin, on affichera sur une ligne au format `" ligne_count  char_count nom_fichier"` le résultat de la fonction, sur une ligne à part.
2. Écrire une fonction `int main(int argc, char* argv[])` qui appelle `wc_sur_fichier(argv[i])` sur chaque argument *sauf le premier* du tableau `argv`. Si un de ces appels résulte en une erreur, on pourra 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 `wc.exe` et vérifier qu'il fonctionne comme la commande `wc` lorsqu'il est exécuté avec un nom de fichier comme argument. On pourra utiliser le fichier de test de l'exercice précédent `fichier_long.txt` contenant 30 lignes pour l'exemple. Pour se comparer à la commande `wc` du terminal, on ignorera le nombre en deuxième position (qui représente le nombre de mots).

```
$ wc Colles_14_fichier_long.txt
 30  30 231 Colles_14_fichier_long.txt
$ ./wc.exe Colles_14_fichier_long.txt
 30  231 Colles_14_fichier_long.txt
```

