Cette *quinzième* 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 Réimplémentation en OCaml de la commande `sort` - OCaml (35 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 `sort.ml`.

1. Écrire une fonction `sort: string -> unit`, qui sera ensuite appelée dans le `main`, qui affiche à l'écran les lignes du fichier dont le nom est donné en argument, triées par ordre (lexicographique) croissant. 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 stockera ces lignes dans une liste mutable (une référence sur une `string list`), avant de bien penser à fermer le fichier. Une fois que l'on a lu toutes les lignes du fichier, on dispose d'une liste de lignes. On peut la convertir en un tableau (`string array`) avec un appel à `Array.of_list`, puis trier le tableau obtenu avec un appel en place à `Array.sort compare array_lignes` (en place veut dire que cet appel modifie directement le tableau donné en argument). La fonction `compare` doit être de type `'a -> 'a -> int` telle que `compare x y` vaut `+1` si $x > y$, `-1` si $x < y$ et `0` en cas d'égalité. La fonction `compare` polymorphe du module `Stdlib` fonctionnera bien, inutile d'en implémenter une soi-même.
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 lignes triées 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 `sort.exe` et vérifier qu'il fonctionne comme la commande `sort` 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 `sort` 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 sort.exe sort.ml

$ ./sort.exe fichier_court.txt
ligne1
ligne2
ligne3
ligne4
```



```
ligne5

$ sort fichier_court.txt
ligne1
ligne2
ligne3
ligne4
ligne5

$ cat fichier_court.txt
ligne5
ligne3
ligne4
ligne1
ligne2
```

4. Que se passe-t-il sur un fichier d'exactement une ligne ? Vérifier que votre programme s'exécute sans erreur et que son comportement est bien le comportement attendu.
5. (bonus ni long ni très difficile) Faire de même pour implémenter un binaire `sort-inverse` qui trie ses lignes dans l'ordre inverse.

Ex.2 Réimplémentation en C de la commande `cat -n` - C (20 minutes)

Écrire une fichier C `catn.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 catn.exe catn.c

$ ./catn.exe fichier_court.txt
     1  ligne1
     2  ligne2
     3  ligne3
$ cat -n fichier_court.txt
     1  ligne1
     2  ligne2
     3  ligne3

$ ./catn.exe fichier_long.txt
     1  ligne1
...
   100  ligne100
```



```
$ cat -n fichier_long.txt
     1  ligne1
...
   100  ligne100
```

Le but de ce binaire `catn.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 contenu du fichier avec le numéro de ligne devant, comme illustré dans les exemples précédents.

1. Écrire une fonction `int catn_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 saut de ligne de son contenu (avec une variable entière que l'on incrémentera une par une), caractère par caractère, avec une boucle `while` sur un `c = fgetc(fp)`. On affichera le numéro de ligne avant d'afficher un `\n` et on affichera tous les autres caractères directement. On fera attention à utiliser le drapeau `"%6i  "` pour afficher un entier avec des espaces au début pour qu'il tienne au moins sur 6 caractères (comme dans les exemples ci-dessus), et deux espaces à la fin avant de continuer à afficher la suite de la ligne. 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 `catn_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 `catn.exe` et vérifier qu'il fonctionne comme la commande `cat -n` lorsqu'il est exécuté avec un ou plusieurs noms de fichier comme argument(s). On pourra utiliser le fichier de test de l'exercice précédent `fichier_long.txt` contenant 30 lignes pour l'exemple (ou on peut créer avec un petit programme C un fichier contenant plus de 100 lignes pour un test plus approfondi).

