Cette dixième colle vous fera écrire une fonction simple en OCaml sur des listes, et un exercice en C sur des calculs sur des caractères ASCII.

Ex.1 Autour de la fonctionnelle map - OCaml (25 minutes)

1. Écrire la fonctionnelle `map : ('a -> 'b) -> 'a list -> 'b list` qui à une fonction `f : 'a -> 'b` et à une liste `[x0; x1; x2; ...; xn]` de type `'a list` associe la liste `[f (x0); f (x1); f (x2); ...; f (xn)]` de type `'b list`.
2. Démontrer très soigneusement et rigoureusement la terminaison et la correction de votre fonction. S'il est nécessaire de faire des hypothèses, explicitez-les clairement.
3. On note C_n le nombre d'appels à la fonction `map` lors de l'appel `map f liste` pour une liste (`'a list`) de longueur $n \in \mathbb{N}$. Déterminer C_n pour tout $n \in \mathbb{N}$.

Ex.2 Autour de l'alphabet et des majuscules et minuscules - C (30 minutes)

Écrire une fichier C `colle10.c` important `stdio.h`, `stdbool.h`.

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 colle10.exe colle10.c
$ ./colle10.exe
```

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

On rappelle qu'en C, le type `char` représente des entiers sur 8 bits, c'est-à-dire un octet, soit des entiers entre `0` et `255`. Il est utilisé pour représenter les caractères ASCII étendu (256), qui peuvent être affiché à l'écran avec le drapeau `%c` dans un `printf`, et sur lesquels on peut faire des opérations arithmétiques comme `'a'+1` qui donnera `'b'`. En particulier, on peut faire des opérations et des comparaisons arithmétiques entre `char`, comme on le ferait avec des entiers. Par contre, on utilise pas un entier comme un `char` et inversement.

4. Afficher ligne par ligne tous les caractères ASCII 128 imprimables en affichant leur valeur entière et le caractère associé, soit les caractères compris entre 32 et 126 (inclus), sous la forme suivante :

```
ASCII # 32 = ' '
ASCII # 33 = '!'
...
ASCII # 126 = '~'
```

5. En observant que les caractères de `'A'` à `'Z'` se suivent, écrire une fonction `char lettre_suivante(char lettre)` qui à `'A'` associe `'B'`, à `'B'` associe `'C'` etc, et à `'Z'` associe `'A'`.

6. Les caractères minuscules `'a'` à `'z'` se suivent aussi. Écrire une fonction `char mise_en_majuscule(char lettre)` qui à `'a'` associe `'A'` et ainsi de suite jusqu'à `'z'` qui est associé à `'Z'`. On supposera que `lettre` se trouve dans l'intervale entre `'A'` et `'Z'` ou entre `'a'` et `'z'`.
7. Faire une fonction inverse `char mise_en_minuscule(char lettre)` qui met en minuscule une lettre, sous les mêmes hypothèses.
8. En utilisant la fonction précédente et le fait qu'une chaîne de caractère (de type `char[]` et pas `char*`) n'est rien d'autre qu'un tableau de `char`, écrire une fonction `void mise_nom_en_majuscule(char nom[])` qui agit par effet de bord pour mettre `nom` entièrement en majuscule.
9. Faire de même pour `void mise_prenom_commencant_majuscule(char prenom[])` qui agit par effet de bord pour mettre la première lettre en majuscule et les suivantes en minuscule.

On aura par exemple les résultats suivants :

```
char nom[] = "dupont";
mise_nom_en_majuscule(nom);
printf("mise_nom_en_majuscule(\"%s\") -> \"%s\"", "dupont", nom);
// affiche : mise_nom_en_majuscule("dupont") -> "DUPONT"

char prenom[] = "TiNtIN";
mise_prenom_commencant_majuscule(prenom);
printf("mise_prenom_commencant_majuscule(\"%s\") -> \"%s\"", "TiNtIN", prenom);
// affiche : mise_prenom_commencant_majuscule("TiNtIN") -> "Tintin"
```

