Cette douzième colle vous fera écrire deux fonctions simples en OCaml sur des listes de paires et des paires de listes, et un exercice en C sur des structures avec des tableaux et des chaînes de caractères.

Ex.1 Autour de listes de paires - OCaml (20 minutes)

Dans le module `List` il y a les deux fonctions suivantes :

— `val split : ('a * 'b) list -> 'a list * 'b list` : *Transform a list of pairs into a pair of lists* : `split [(a1,b1); ...; (an,bn)]` est `([a1; ...; an], [b1; ...; bn])`.

— `val combine : 'a list -> 'b list -> ('a * 'b) list` : *Transform a pair of lists into a list of pairs* : `combine [a1; ...; an] [b1; ...; bn] is [(a1,b1); ...; (an,bn)]`. Raises `Invalid_argument` if the two lists have different lengths.

1. Écrire ces deux fonctions.
2. Tester les sur plusieurs listes.
3. Quelle est la complexité asymptotique temporelle de chaque fonction ?

Ex.2 Sur une structure à plusieurs champs - C (30 minutes)

Écrire une fichier C `colle12.c` important `stdio.h` (pour `printf` et `scanf`), et `stdlib.h` (pour `malloc` et `free`).

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 colle12.exe colle12.c
$ ./colle12.exe
```

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

1. Définir une structure `eleve` ayant cinq champs `prenom` et `nom` des chaînes de caractères `char[]`, `age` un entier, `nombre_notes` un entier et enfin `notes` un tableau de `double` (un `double*`). Vous pouvez choisir de restreindre les chaînes à une longueur maximale donnée (par exemple `20`) mais pas besoin pour les notes.
2. Dans votre fonction `main`, définir un exemple de variable ayant ce type `struct eleve` en initialisant à la main les cinq champs pour votre propre identité. On commencera par initialiser à la main un tableau de notes : par exemple avec `double notes[5] = {12, 14, 16, 18, 20};`.
3. Définir une fonction `double moyenne(int nombre_notes, double notes[])` qui calcule la moyenne (arithmétique) des notes d'un élève donné. On supposera que `nombre_notes` > 0.

4. Définir une fonction `void affiche_eleve(struct eleve id)` qui affiche une identité au format suivant : `<Prenom> <Nom> (<AGE> ans) a comme moyenne <MOYENNE>` (sur une ligne à part).
5. Définir une fonction `void creer_notes(int nombre_notes, double* notes)` qui prend un tableau `notes` déjà déclaré sur le tas (avec un `malloc` préalable), de taille donnée `nombre_notes`, puis demande chaque note à l'utilisateur avec des `scanf("%lf", &note)` suivis de `notes[i] = note` dans une boucle.
6. Définir une fonction `struct eleve creer_eleve(void)` qui déclare une nouvelle variable de type `struct eleve`, et effectue quatre demandes d'informations au clavier avec des `scanf("drapeau", pointeur)` pour les quatre premiers champs, et un appel à `creer_notes()` pour les notes, précédemment allouées sur le tas avec un `malloc` de la bonne taille (`nombre_notes * sizeof(double)`).
7. Dans votre `main`, faites un test de cette fonction de saisie d'un élève `autre_eleve`, puis afficher cet élève ainsi saisis par l'utilisateur au clavier.
8. En fin de votre fonction `main`, on pensera bien à désallouer les blocs précédemment alloués sur le tas.

