TP n° 3 : Listes

Dans ce TP, nous vous proposons de travailler avec la structure naturellement récursive de liste chaînée en CAML.

On rappelle qu'une liste, de type `'a list`,

- peut être vide, et est alors notée `[]`,
- a tous ses éléments de même type,
- possède, lorsqu'elle n'est pas vide une tête (premier élément) et une queue (liste des autres éléments), auxquelles on accède en temps constant par filtrage `tete :: queue`.
- `[1; 2; 3; 4]` est une écriture raccourcie pour `1 :: 2 :: 3 :: 4 :: []`.

1 Fonctions de base sur les listes vues en cours

EXERCICE 1 *Longueur d'une liste*

Écrire une fonction `list_length : 'a list -> int` qui renvoie la longueur d'une liste. Quelle est sa complexité en fonction de la taille de la liste ?

EXERCICE 2 *Concaténation*

Écrire une fonction `concat : 'a list -> 'a list -> 'a list` qui renvoie la liste concaténée à partir de deux listes. Quelle est sa complexité en fonction des tailles des deux listes ?

Par exemple, `concat [1; 2] [3; 4; 5]` doit renvoyer `[1; 2; 3; 4; 5]`.

2 D'autres fonctions

EXERCICE 3 *Maximum*

Écrire une fonction `maxi : 'a list -> 'a` qui renvoie l'élément maximal d'une liste (d'éléments comparables).

EXERCICE 4 *n^e élément*

Écrire une fonction `element_numero : int -> 'a list -> 'a` qui, à partir d'un entier n et d'une liste, renvoie le n^e élément de la liste.

EXERCICE 5 *Liste arithmétique*

Écrire une fonction `liste_arithm : int -> int -> int -> int list` qui prend en entrée des entiers n, a et b et qui renvoie une liste de longueur n dont les éléments sont les n premiers termes de la suite arithmétique de premier terme a et de raison b.

Exercice 6 *Fonction mystère*

Quel est le type de la fonction suivante, et que fait-elle ?

```
let rec mystere f liste =
  match liste with
  | [] -> []
  | tete :: queue -> (f tete) :: mystere f queue
;;
```

Exercice 7 *Découplage*

Écrire une fonction `decouple : ('a * 'b) list -> 'a list * 'b list` qui à partir d'une liste de couple renvoie un couple de listes

Exercice 8 *for_all*

Écrire une fonction `for_all : ('a -> bool) -> 'a list -> bool` prenant en argument un prédicat qui a un élément de type `'a` associe un booléen et une liste, et renvoyant `true` si le prédicat est vérifié pour tous les éléments de la liste, faux sinon.

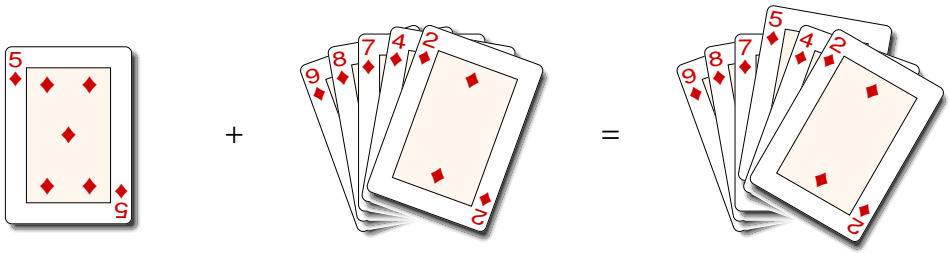
Par exemple,

- `for_all (function x -> x > 2) [1; 2; 3]` renvoie `false`
- `for_all (function x -> x > 2) [4; 7; 9]` renvoie `true`.

3 Un tri

Exercice 9

On se propose de programmer le tri insertion (*insertion sort*) sur les listes. Le principe de ce tri est celui que vous utilisez pour trier un jeu de cartes en main : au fur et à mesure du tri, vous insérez un nouvel élément parmi ceux déjà triés.



1. Écrire une fonction `insere : 'a -> 'a list -> 'a list` prenant en argument un objet et une liste triée par ordre croissant et renvoyant la liste obtenue en insérant l'objet dans la liste en conservant son caractère trié.

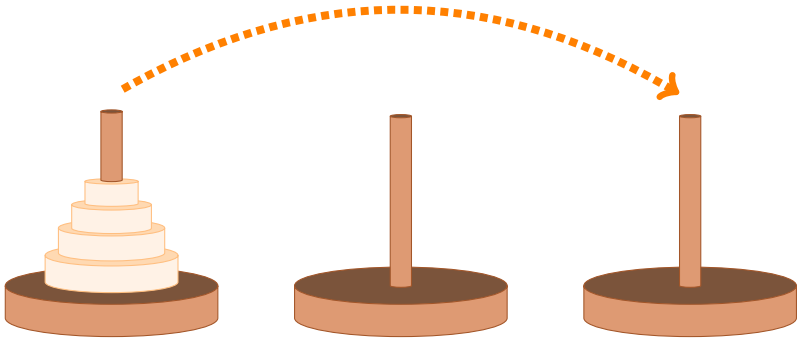
   Par exemple, `insere 5 [2; 4; 7; 8; 9]` renvoie `[2; 4; 5; 7; 8; 9]`.

2. Écrire une fonction `tri_insertion : 'a list -> 'a list` triant la liste : lorsque c'est encore possible, il suffit d'insérer la tête dans la queue qui aura été triée récursivement.

3. Quelle est la complexité temporelle dans le meilleur et le pire des cas ? Préciser des listes pour lesquelles ces cas sont atteints.

4 Un jeu

Exercice 10



Le jeu des **tours de Hanoï** a été inventé par le mathématicien Édouard Lucas [1] au dix-neuvième siècle. Le principe est le suivant : il faut déplacer un pile de n disques d'un emplacement initial à un emplacement final en utilisant un emplacement intermédiaire, en respectant les règles :

- on ne peut déplacer que le disque en haut d'une pile,
- on ne peut empiler un disque que sur un disque plus grand.

1. Écrire une fonction

```
deplacement : int -> string -> string -> string
```

qui prend en argument un entier `n` et des chaînes de caractères `origine` et `destination` et renvoyant la chaîne de caractère `"Déplacer le disque {n} de {origine} à {destination}."` où `{n}`, `{origine}` et `{destination}` sont remplacés par leur valeur.

2. Écrire une fonction récursive

```
hanoi_liste : int -> string -> string -> string -> string list -> string list
```

qui prend en argument :

- un nombre entier `n` de disques à déplacer,
- des chaînes de caractères `debut`, `fin` et `intermediaire` décrivant les piquets de départ et d'arrivée pour le déplacement des `n` disques, ainsi que le troisième piquet,
- une liste de déplacements (chaînes de caractères) `liste_depl` déjà effectués

et qui :

- renvoie la liste de déplacements s'il n'y en a pas d'autre à faire,
- résout sinon le problème de déplacements des n disques [2], à l'aide de deux déplacements de $n-1$ disques (appels récursifs) et du déplacement d'un disque (appel à `deplacement`) et renvoie la liste de tous les déplacements effectués.

1. 
Édouard Lucas (1842 - 1891) est un arithméticien français également connu pour ses récréations mathématiques. Il entre à l'École Normale Supérieure puis devient professeur de Spéciales à Paris. Ses travaux mathématiques concernent la géométrie euclidienne non élémentaire, et surtout la théorie des nombres. Sa principale contribution est celle faite aux tests de primalité. Il a en particulier prouvé que le nombre de Mersenne $2^{127}-1$ est premier, ce qui reste le plus grand nombre premier découvert sans l'aide d'un ordinateur. Lucas est aussi connu pour être l'inventeur de nombreuses récréations mathématiques. La plus répandue d'entre elles est le problème des tours de Hanoï. Lucas est mort au cours d'un banquet : une assiette portant un couteau est tombée et lui a transpercé la gorge.
2. Si vous ne voyez vraiment pas, tournez la page.

3. Écrire une fonction `hanoi : int -> string list` renvoyant la liste de tous les déplacements à effectuer pour résoudre le problème avec un nombre donné de disque.

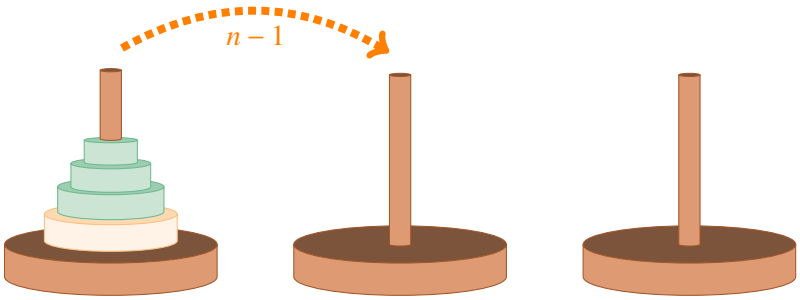
   Exemple :

```
hanoi 5;;
- : string list =
 ["Deplacer le disque 1 de gauche a droite.";
  "Deplacer le disque 2 de gauche a milieu.";
  "Deplacer le disque 1 de droite a milieu.";
  "Deplacer le disque 3 de gauche a droite.";
  "Deplacer le disque 1 de milieu a gauche.";
  "Deplacer le disque 2 de milieu a droite.";
  "Deplacer le disque 1 de gauche a droite.";
  "Deplacer le disque 4 de gauche a milieu.";
  "Deplacer le disque 1 de droite a milieu.";
  "Deplacer le disque 2 de droite a gauche.";
  "Deplacer le disque 1 de milieu a gauche.";
  "Deplacer le disque 3 de droite a milieu.";
  "Deplacer le disque 1 de gauche a droite.";
  "Deplacer le disque 2 de gauche a milieu.";
  "Deplacer le disque 1 de droite a milieu.";
  "Deplacer le disque 5 de gauche a droite.";
  "Deplacer le disque 1 de milieu a gauche.";
  "Deplacer le disque 2 de milieu a droite.";
  "Deplacer le disque 1 de gauche a droite.";
  "Deplacer le disque 3 de milieu a gauche.";
  "Deplacer le disque 1 de droite a milieu.";
  "Deplacer le disque 2 de droite a gauche.";
  "Deplacer le disque 1 de milieu a gauche.";
  "Deplacer le disque 4 de milieu a droite.";
  "Deplacer le disque 1 de gauche a droite.";
  "Deplacer le disque 2 de gauche a milieu.";
  "Deplacer le disque 1 de droite a milieu.";
  "Deplacer le disque 3 de gauche a droite.";
  "Deplacer le disque 1 de milieu a gauche.";
  "Deplacer le disque 2 de milieu a droite.";
  "Deplacer le disque 1 de gauche a droite."]
```

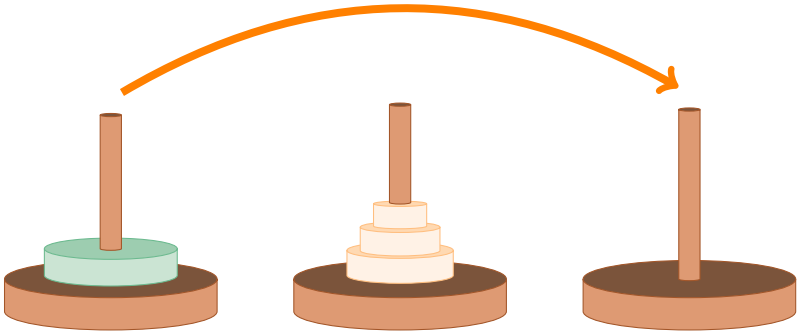
4. Combien de déplacements sont nécessaires pour déplacer n disques ?

C'est un problème facile à résoudre récursivement : il suffit de :

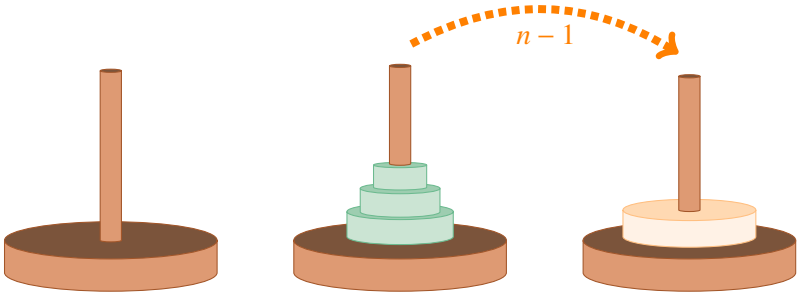
1. Déplacer les $n-1$ premiers disques sur la tige intermédiaire,



2. Puis le plus grand disque sur la tige finale



3. Et enfin de nouveau les $n-1$ premiers disques vers la tige finale.



Cela ne viole aucune règle du jeu.