

## TP n° 4

**Consignes** les exercices ou questions marqués d'un  $\star$  devront être rédigés sur papier (afin de se préparer aux épreuves écrites de l'examen). En particulier, il est recommandé d'être dans les mêmes conditions qu'en examen : pas de document ni de calculatrice. Tous les TPs se font sous Linux.

### 0 Prise en main de l'environnement

Pour faciliter les TPs en mode distanciel, nous utiliserons le serveur Jupiter hébergé par le LAL (Laboratoire Accélérateur Linéaire). Dans cet environnement, vous avez accès, via un navigateur Web à :

- un espace de stockage
- un terminal Unix
- un éditeur de texte avec la coloration syntaxique pour OCaml
- les commandes vues en cours : `ocamlc`, `ocaml`, ...

1. Se rendre sur <https://jupytercloud.lal.in2p3.fr/>
2. Cliquer sur se connecter
3. Utiliser l'authentification centralisée de Paris-Saclay (si vous êtes déjà connecté via eCampus, il est possible que l'authentification se fasse automatiquement)
4. Si vous êtes redirigé vers la page de garde : <https://jupytercloud.lal.in2p3.fr/>, c'est un bug connu. Effacez vos cookies ou ouvrez l'URL dans une fenêtre de navigation privée
5. Arrivé sur l'interface, choisissez « Start My Server » (bouton vert)

Vous vous retrouvez dans un navigateur de fichier. Vous pouvez créer des répertoires et des fichiers. Le bouton « New » à droite comporte plusieurs options dont :

- Create Terminal : ouvre un terminal dans un autre onglet de votre navigateur
- Create Texte File : ouvre un éditeur de texte

Dans l'éditeur de texte, si vous renommez le fichier en lui donnant l'extension « `.ml` » la coloration syntaxique pour OCaml est activée.

Il est **vivement recommandé** de télécharger vos fichiers après chaque exercice, pour en avoir une copie sur votre ordinateur personnel. Les sauvegardes sur cet espace de travail expérimental ne sont pas garanties, et c'est un espace distinct de vos répertoires personnels sur les machines de l'Université.

### 1 Re-tours de Hanoï

Nous allons revisiter l'exercice sur les tours de Hanoï, dans le but d'afficher en temps réel (à chaque mouvement) l'état de chaque piquet.

On rappelle la solution récursive du problème :

```

1 (* On rappelle la fonction de hanoi du TD 2 *)
2 let hanoi n =
3   let rec hanoi_aux dep mil arr n =
4     if n > 0 then begin
5       hanoi_aux dep arr mil (n - 1);
6       Printf.printf "%s -> %s\n" dep arr;
7       hanoi_aux mil dep arr (n - 1)
8     end
9   in
10  hanoi_aux "dep" "mil" "arr" n

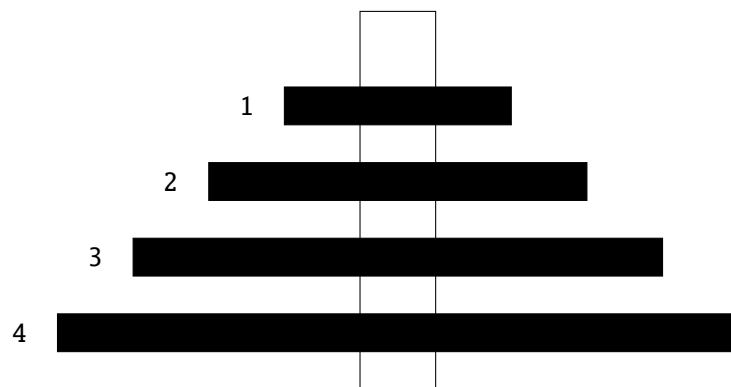
```

Le principe récursif est le suivant :

- cas de base  $n = 0$  : on n'a rien à faire (pas de **else**)
- cas récursif : on utilise récursivement la fonction **hanoi\_aux** pour déplacer  $n - 1$  disques du piquet de départ vers le piquet du milieu en utilisant le piquet d'arrivée comme zone temporaire. Une fois que c'est fait on est donc dans une configuration où :
  - le plus grand disque est sur le piquet de départ (car on a bougé les  $n - 1$  au dessus de lui ;
  - tous les autres disques sont empilés dans le bon ordre sur le piquet du milieu ;
  - le piquet d'arrivée est vide.

On peut donc déplacer librement le plus grand disque du piquet de départ au piquet d'arrivée (ligne 6). Ensuite, on peut de façon symétrique déplacer les  $n - 1$  disques du piquet du milieu vers le piquet d'arrivée en utilisant le piquet de départ (maintenant vide) comme zone temporaire.

1. Reporter la fonction **hanoi** ci-dessus au début d'un fichier **hanoi\_list.ml** et la tester sur une tour à 4 éléments.
2. On modélise les piquets par une paire de type **string \* int list** : le nom du piquet (qui sera **"dep"**, **"mid"** ou **"arr"**) et la liste des disques sur ce piquets. Les disques sont représentés par des entiers proportionnels à leur diamètre. Ainsi, la liste **[1;2;3;4]** représente le piquet :



Définir deux types :

```
1 type piquet = string * int list ;;
2 type jeu = piquet list;;
```

3. Écrire une fonction **affiche\_piquet** : **piquet** → **unit** qui affiche un piquet dans la console. Par exemple, le piquet (**"mid"**, **[1;2;5]**) sera affiché comme ceci :

```
mid|5-2-1-
```

C'est à dire le nom du piquet suivi d'une barre verticale, suivi de chaque nombre suivi d'un tiret. Attention, l'affichage doit commencer par la fin de la liste, il est donc judicieux de la renverser avec **List.rev**.

4. Écrire une fonction **choix\_piquet** : **jeu** → **string** → **piquet**. Étant donné une liste de piquets et un nom, la fonction renvoie le piquet ayant ce nom. Par exemple :

```
1 let piquets = [ ("dep", [2;3;4]);
2                ("mid", []);
3                ("arr", [1])]
4 ;;
5 let m = choix_piquet piquets "mid"
6 ;;
7 (* m vaut ("mid", []) *)
```

Attention, on ne connaît pas l'ordre des piquets dans la liste (ce n'est pas forcément **"dep"** en premier. On pourra utiliser la fonction **List.assoc**.

5. Écrire une fonction **affiche\_jeu** : **jeu** → **unit**. Cette dernière doit utiliser **Printf.printf** pour :
  - Afficher la chaîne de caractères **"\x1b[2J\x1b[H"**<sup>1</sup>.

1. Pour les curieux ou les curieuses, vous pouvez vous renseigner sur les séquences d'échappement ANSI, des caractères spéciaux qui permettent de contrôler le curseur du terminal

- Afficher d'abord le piquet `"dep"`, puis le piquet `"mid"` puis le piquet `"arr"`. Évidemment, il convient de réutiliser `affiche_piquet` et `choix_piquet`.
  - Afficher la séquence spéciale `"%!"`. Cette dernière, indique de forcer l'affichage dans le terminal.
6. Écrire un fonction `deplace_sommet : piquet -> piquet -> piquet * piquet`. Cette fonction prend deux piquets en argument et déplace le disque au sommet du premier vers le sommet du second piquet. La fonction renvoie les nouveaux piquets. Dans les cas suivant, la fonction lève une exception au moyen de la fonction `failwith` :
    - Le premier piquet est vide.
    - L'entier au sommet du premier piquet est supérieur à celui au sommet du second piquet.
  7. Écrire une fonction `joue : jeu -> string -> string -> string -> jeu` telle que `joue piquets src ds` déplace le sommet du piquet `src` au sommet du piquet `dst` et reconstitue le jeu avec :
    - le piquet `autre`
    - les deux piquets `src` et `dst` modifiés.
  8. Écrire une fonction `gen_list : int -> int list` telle que `gen_list n` crée une liste contenant les entiers `[1; 2; ... ; n]`.
  9. Enfin, écrire une fonction `hanoi_list n`. Cette dernière résout le problème des tours de Hanoï pour  $n$  disques.

La fonction utilise en interne une fonction auxiliaire

`hanoi_aux : jeu -> string -> string -> string -> int -> jeu`

Cette fonction a la même structure récursive que la version simple en début de fichier mais elle doit renvoyer le `jeu` (c'est à dire la liste des trois piquets modifiés).

On appellera `hanoi_aux` sur la configuration initiale :

```

1  let hanoi_list n =
2    let rec hanoi_aux piquets dep mil arr n =
3      if n > 0 then begin
4        ...
5      end else ...
6    in
7    let final =
8      hanoi_aux [ ("dep", gen_list n);
9                  ("mil", []);
10                 ("arr", []) ] "dep" "mil" "arr" n
11  in
12  affiche_jeu final

```

L'affichage étant trop rapide pour être suivi, on pourra ralentir le programme en mettant, dans le corps de la fonction `hanoi_aux` l'instruction `Unix.sleepf 0.5`; juste après l'affichage du jeu, ce qui permet au programme de se mettre en pause pendant 0.5s. Pour utiliser cette fonction, vous devez compiler votre programme avec la ligne de commande :

```
$ ocamlc -o hanoi_list unix.cma hanoi_list.ml
```

ou

```
$ ocamlc -o hanoi_list unix.cmxa hanoi_list.ml
```