

TP n° 2

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 Lecture de code

\star Dire si les programmes suivants sont bien typés ou mal-typés. S'ils sont bien typés, donner le type de toutes les variables et fonctions globales du fichier. S'ils sont mal typés, indiquer précisément la ligne et la nature de l'erreur de typage.

(a)

```
1 let x = 3 ;;
2 let y = 4 ;;
3 let z =
4   string_of_int x + y ;;
```

(b)

```
1 let f x y = (x + 1) * (y - 1)
2 ;;
3 let u = f (f 2 3) (f 4 5) ;;
```

```

1 let x = 47 ;;
2 let y =
3   if x mod 2 = 0 then
4     3.0
5   else
6     4.0
7 let z = x + y;;
8 ;;

```

```

1 let f x = x / 17 ;;
2 let g x =
3   if x mod 2 = 0 then
4     "pair"
5   else
6     false
7   ;;
8 let u = f 42;;

```

```

1 let rec f n =
2   if n <= 0 then 1
3   else n * f (n-1)
4 ;;
5 let g n =
6   Printf.printf
7   "%d! = %d\n" n (f n)
8 ;;

```

```

1 let rec f n =
2   if n >= 0 then begin
3     Printf.printf "%d\n" n;
4     f (n-1)
5   end
6 ;;

```

2 Entrées/Sorties

Le but de l'exercice est de s'inspirer du programme 2 de la feuille de TP 1 et de le complexifier pour qu'il reproduise le comportement de la commande Unix : `ls -R d`. Cette dernière affiche récursivement le contenu d'un répertoire et de tous les répertoire contenu dans ce dernier.

1. Écrire une fonction `affiche_tab a` qui prend en argument un tableau de chaînes de caractères `a` et qui affiche le contenu des cases, chacun sur une ligne avec un retour à la ligne et décalé de 4 espaces par rapport au début de la ligne. On pourra utiliser une fonction récursive `affiche_tab_aux i a` qui prend en argument l'indice courant.
2. Écrire du code de test qui appelle `affiche_tab Sys.argv` et vérifier que toutes les arguments de votre ligne de commande sont bien affichés.
3. Créer deux fonctions mutuellement récursives :

```

1 let rec affiche_dir d =
2   ...
3 and affiche_dir_iter i t =
4   ...
5 ;;

```

où `affiche_dir d` :

cas de base : ne fait rien si `d` n'est pas un répertoire (tester avec `Sys.is_directory`)

cas récursif :

- Affiche `d` en début de ligne, suivi de « : » et un retour à la ligne.
- Récupère le contenu du répertoire `d` au moyen de `Sys.readdir`. Cette fonction prend en argument le nom d'un répertoire et renvoie son contenu sous la forme d'un tableau `t`.
- Affiche `t` au moyen de `affiche_tab`
- Fait de `d` le répertoire courant (en utilisant `Sys.chdir`)
- appelle récursivement `affiche_dir_iter 0 t`.
- Remonte dans le répertoire initial avec `Sys.chdir ".."`

La fonction `affiche_dir_iter i t` appelle récursivement `affiche_dir t.(j)` pour toutes les indices `j` compris entre `i` et la taille de `t`.

Enfin, faire en sorte que votre programme appelle `affiche_dir "."` (i.e. affiche récursivement le répertoire courant).

4. Créer (dans le terminal, pas en OCaml) un ensemble de répertoires :

```

$ mkdir -p a/b/c
$ mkdir -p a/d/e
$ touch a/f1.txt a/b/f2.txt a/d/e/f3.txt

```

Tester votre programme en l'exécutant dans le répertoire courant. Il doit afficher tous les fichiers s'y trouvant ainsi que les répertoires `a`, `b`, `c`, `d`, `e` et leur contenu.

5. Pour chacune des fonctions, `affiche_tab_aux`, `affiche_dir`, `affiche_dir_iter`, dire si elle est récursive terminale ou non en justifiant.

3 Tours de Hanoï

Le problème des Tours de Hanoï est un célèbre problème possédant une élégante solution sous forme de fonction récursive. Le jeu consiste en trois piquets, *A*, *B*, *C* sur lesquels on peut enfiler des disques de tailles décroissantes. Le but du jeu est de partir d'une pyramide de *n* disques sur le piquet *A* et de déplacer tous les disques vers le piquet *C*. Une contrainte est qu'il est interdit de placer un disque sur un disque plus petit. La figure 1 indique une configuration. Dans cette partie, il est interdit de déplacer le disque 4 sur le disque 1.

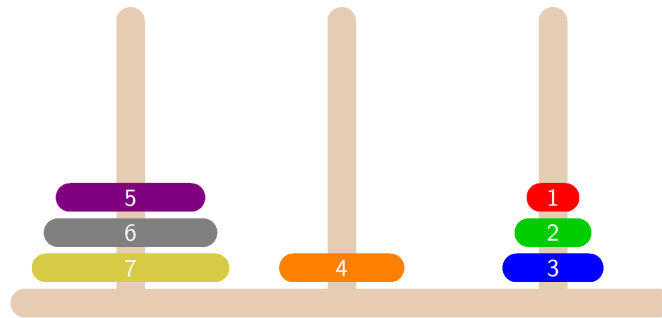


FIGURE 1 – Les tours de Hanoï

Pour déplacer le disque 4 sur le piquet de droite, on peut effectuer la séquence de coups suivante :

- déplacer 1 sur 5 (à gauche)
- déplacer 2 sur 4 (au milieu)
- déplacer 1 sur 2 (au milieu)
- déplacer 3 sur 5 (à gauche)
- déplacer 4 sur le piquet de droite.

1. On veut résoudre ce problème à l'aide d'un programme, c'est à dire étant donné le nombre *n* de disques au départ, afficher la liste des mouvements à faire. Compléter le squelette de code proposé :

```

1  let rec hanoi dep aux dest n =
2    if n > 0 then begin
3      (* ... *)
4      (* ... *)
5      (* ... *)
6    end
7
8  let () = hanoi "A" "B" "C" 5

```

La fonction affiche les coups à jouer sous la forme `A -> C` pour indiquer qu'il faut déplacer le disque se trouvant au sommet du piquet `A` vers le piquet `C`.

2. Transformer le code pour n'avoir qu'une seule fonction `hanoi n` qui appelle la fonction récursive auxiliaire `hanoi_aux "A" "B" "C" n`.
3. La fonction `Sys.time ()` renvoie un nombre flottant représentant le nombre de secondes écoulées depuis le début du programme. Modifier le programme pour mesurer le temps pris par la fonction `hanoi` pour s'exécuter. À partir de quelle valeur de `n` dépasse-t-on 10 secondes ?