université PARIS-SACLAY | FACULTÉ DES SCIENCES D'ORSAY

semaine du 9 septembre 2024

Introduction à la programmation fonctionnelle
L2 Info – LDD2 IM

TP n° 1

Les exercices marqués ⋆ sont à faire sur un cahier ou dans un fichier, pour en garder une trace écrite.

0 Environnement de travail

0.1 Session de secours

Les sessions de secours étant recrées tous les jours, il est conseillé de travailler de la façon suivante :
— télécharger l'archive ZIP contenant les fichiers à compléter et la décompresser. Cette dernière contient un unique répertoire `ipf_l2_info_tpxx` où `xx` est le numéro du TP.
— ouvrir un terminal et se placer dans le répertoire en question `cd chemin/vers/le/repertoire`
— exécuter le script d'initialisation `./init_tp.sh`
— fermer le terminal et en ouvrir un nouveau
— travailler (voir la section 0.2)
— sauvegarder régulièrement et au moins une fois en fin de séance les fichiers du TP (soit en ligne sur un espace personnel [1], soit en les stockant sur une clé USB)

0.2 Utilisation de Visual Studio Code

Une fois configuré correctement (par le script d'initialisation), VSCode est un éditeur confortable pour du code OCaml (moins lourd que Netbeans ou Eclipse en particulier). Voici l'ensemble minimal des commandes pour les TPs :
— Création d'un nouveau Fichier : `CTRL-N` (ou « *File* → *New file* ». Attention, le fichier nouvellement créé n'a pas de nom. Il convient alors de l'enregistrer (`CTRL-S`) ou *Save...*) en lui donnant un nom se terminant par `.ml`
— Évaluation d'une expression OCaml : il est possible de surligner (SHIFT+↑ / ↓ ou en utilisant la souris) une portion de code OCaml puis de l'envoyer dans un interpréteur avec `SHIFT-ENTER`. Attention, il convient d'évaluer les définitions dans l'ordre.

0.3 Utilisation de l'environnement

Au début de chaque séance on fera les actions suivantes :

1. ouvrir un terminal
2. (éventuellement se placer dans un sous-répertoire de votre choix)
3. créer un répertoire pour le TP :

   ```
   mkdir tp01
   ```

4. se placer à l'intérieur

   ```
   cd tp01
   ```

5. démarrer Visual Studio code sur ce répertoire

   ```
   code .
   ```

1. a priori seul le HTTP et HTTPS sortant est autorisé à l'heure actuelle, il est donc possible que l'utilisation de `git` via SSH ne fonctionne pas

1 Lecture de code

⋆ Pour chacun des programmes OCaml suivants, dire ce qu'ils affichent.

(a)
```
1 let x = 3
2 let y = 4
3 let z = x + y
4 let () = Printf.printf "%d\n" z
```

(b)
```
1 let f x = (x + 1) * (x - 1)
2 let u = f (f 2)
3 let () = Printf.printf "%d\n" u
```

(c)
```
1 let x = 47
2 let y =
3    if x mod 2 = 0 then
4      3
5    else
6    4
7
8 let () = Printf.printf "%d\n" y
```

(d)
```
1  let f x = x / 17
2  let g x =
3     if x mod 2 = 0 then
4     "pair"
5     else
6     "impair"
7
8  let u = f 42
9  let () =
10    Printf.printf
11    "%d est %s\n" u (g u)
```

(e)
```
1
2 let x = 3.14159
3 let x2 = x *. x
4 let x4 = x2 *. x2
5 let () = Printf.printf "%f\n" x4
```

(f)
```
1 let rec f n =
2    if n >= 0 then begin
3        Printf.printf "%d\n" n;
4        f (n-1)
5     end
6 let () = f 10
```

2 Entrées/Sorties

Écrire un programme qui attend en argument sur sa ligne de commande un nom de fichier, qui teste si ce fichier existe et qui teste son type. Plus précisément :

- S'il n'y a pas d'argument sur la ligne de commande, le programme affiche `"Pas assez d'arguments"`.
- S'il y a un argument mais qu'il ne correspond pas à un nom de fichier existant, le programme affiche `"Le fichier n'existe pas"`
- Si le fichier existe, le programme affiche `"Le fichier existe"` ou `"Le répertoire existe"` selon le cas.

Les fonctionalités utiles pour réaliser cet exercice sont :

- `Sys.argv` : le tableau des arguments passés au programme. Il contient toujours au moins une entrée : le nom du programme.
- `Array.length` : une fonction renvoyant la taille d'un tableau passé en argument.
- `Sys.file_exists` : une fonction qui teste si un nom de fichier existe.
- `Sys.is_directory` : une fonction qui teste si un nom de fichier est un répertoire.

3 Fonctions récursives

1. Écrire une fonction `somme_entiers n` qui renvoie la somme des entiers entre `n` et `0`. On pourra s'inspirer de la fonction (f) de l'exercice 1, en traitant le cas **`else`** (*i.e.* que faire lorsque `n` vaut 0).
2. Écrire une fonction `somme_carres n` qui renvoie la somme des i^2 pour i entre `0` (inclus) et `n` (exclu).
3. Écrire une fonction `leibniz n` qui calcule l'approximation de $\frac{\pi}{4}$ en utilisant la formule de Leibniz :

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots + (-1)^i \times \frac{1}{2i+1} + \ldots$$

4. Tester ces fonctions en les appelant sur différentes valeurs.

4 Dates

1. Écrire une fonction OCaml `bissextile a` qui renvoie `true` si `a` est bissexstile et `false` sinon. Quel est le type de cette fonction ?
2. Écrire une fonction `jour_mois m a` qui prend en argument un entier `m` compris entre 1 et 12 et qui renvoie le nombre du jour dans le mois `m` de l'année `a`.

   Remarque plutôt que d'utiliser 11 `if then else`, on peut remarquer que, si on exclut le mois de février alors :
   — les mois 8 à 12 se comportent comme les mois 1 à 8.
   — pour les mois entre 1 et 8, si le mois est impair il a 31 jour et s'il est pair il a 30 jours.
3. Écrire une fonction `jour_date j m a` qui prend trois arguments représentant un jour `j` (à partir de 1), un mois `m` (entre 1 et 12) et une année `a` et qui renvoie le nombre de jour écoulé entre cette date et le premier janvier de l'année.

   Remarque on pourra utiliser une fonction récursive pour parcourir les mois, en s'inspirant de l'exercice 3. En particulier :

   Cas de base : si le mois est janvier, il suffit renvoyer le nombre de jour de la date (le 4/1/2019 est bien le 4$^{\text{ème}}$ jour de l'année 2019).

   Cas récursif : pour une date `j m a`, le nombre de jours est la somme du nombre de jour dans le mois précédant `m` et (récursivement) le nombre de jour à la date du mois précédent : `j (m-1) a`.
4. Écrire un programme qui teste vos fonctions sur plusieurs dates.

5 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 ont peut enfiler des disques de tailles décroissantes. Le but du jeu est de partir d'une pyramide de n disque 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.

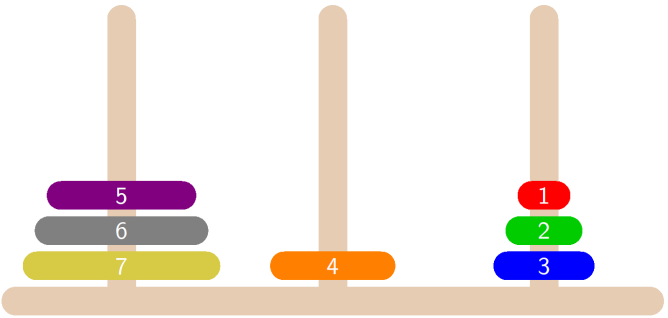

FIGURE 1 – Les tours de Hanoï

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 se problème à l'aide d'un programme, c'est à dire étant donné le nombre n de disque 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 mesure le temps pris par la fonction **`hanoi`** pour s'exécuter. À partir de quelle valeur de **`n`** dépasse t'on 10 secondes ?