Lycée Faidherbe, 2020-2021

T.D. d'informatique

MPSI option informatique

Version du 13 mars 2021

Table des matières

TD I

Variables et fonctions

L'objectif de ce TD est de découvrir les variables en Ocaml et les premières fonctions.

Exercice 1 — Comprendre les types

Pour chacune des entrées suivantes, donner le résultat et le type obtenu :

- `3;;`
- `12.7;;`
- `'a';;`
- `"abcde";;`
- `false;;`
- `let x = 5;;`
- `let y = x * 3;;`
- `x > y;;`
- `let b = true && false;;`
- `let c = b || ( x = y ) ;;`
- `let z = 3.7 *. 2.0 +. 1.3;;`
- `let f1 x = x + 2;;`
- `let f2 x = x ** 2.3;;`
- `let f3 x y = x + y;;`
- `let f4 a b = a || b;;`
- `let f5 (x,y) = x +. y;;`
- `let f6 (a,b) c = a && b && (c<>1);;`
- `let f7 x y = y = 3;;`
- `let f8 x y = (y = 3, x > 7.2);;`
- `let proj1 a b = a;;`
- `let proj2 (a,b) = a;;`
- `let f9 g a b = g a + 3;;`
- `let f10 g h x y = g x +. h y;;`
- `let f11 g h b1 b2 b3 = g (b1 + 2) || h (b2 *. 3.7) && b3;;`
- `f3 2;;`
- `f4 true;;`
- `let f12 x = if x > 5 then 13 else x - 13;;`
- `let f13 a b = if a then b else not b;;`
- `let f14 g a b = if a > b then g a else g b;;`
- `let f15 x y z = (x y) z;;`
- `let f16 x y z = x y z;;`
- `let f17 x y z = x (y z);;`
- `let f18 x y z = (x y) (z x);;`

- `let f19 x y z = x (y z x);;`

Exercice 2 — Implémenter des fonctions de base
Implémenter chacune des fonctions suivantes :

- `f1` teste si trois entiers `a`, `b` et `c` sont tous non nuls,
- `f2` renvoie le maximum de deux paramètres `a` et `b`,
- `f3` renvoie le maximum de trois paramètres `a`, `b` et `c`,
- `f4` retourne le discriminant du polynôme $aX^2 + bX + c$, fourni sous forme d'un triplet `a`, `b`, `c` de coefficients,
- `f5` est la fonction $x \mapsto \dfrac{2x}{1+x^2}$,
- `f6` retourne la valeur absolue d'un flottant,
- `f7` teste (avec un réponse sous forme de booléen) si trois paramètres `a`, `b`, `c` sont tous égaux,
- `f8` retourne le maximum entre x et son inverse.

Exercice 3 — Implémenter une fonction d'un type donné
Implémenter des fonctions dont le type sera dans la liste suivante :

- `int -> int -> int`
- `(int -> int) -> int`
- `float -> bool -> int`
- `float -> (int -> int) -> float`
- `int-> (bool -> bool)`
- `int * bool -> bool`
- `int * float*bool -> bool`

Exercice 4 — Variable locale
A l'aide de `in` évaluer les expressions suivantes en ne faisant appel qu'une fois aux opérateurs mathématiques.

- $\cos\big(\sqrt{2}\big) + \cos^3\big(\sqrt{2}\big) + 7 \times \cos^5\big(\sqrt{2}\big)$.
- Évaluation en $x = e^3$ de $\dfrac{3 + x + 7x^2}{x^2 - 5x^2 + 2}$.
- On note $f \mapsto \cos\big(x^2 + x + 1\big)$ et $g = f \circ f$. Calculer $g\big(\ln(3)\big)$.
- La fonction th.

Exercice 5 — Utilisation de fun et function
Implémenter les fonction suivante :

- `f1` associe à une suite $(u_n)_{n\in\mathbb{N}}$ d'entiers la suite $(u_{n+1} - u_n)_{n\in\mathbb{N}}$
- La composition de fonctions qui à deux fonctions (de variable et à valeurs dans l'ensemble des flottants) f et g associe $f \circ g$

TD II

Fonctions avec filtrage

L'objectif du TD est d'implémenter les fonctions suivantes une première fois sans filtrage et une seconde en utilisant des filtrages.

1. Valeur absolue d'un flottant
2. Maximum de deux éléments
3. Maximum d'un couple d'éléments
4. Fonction en escalier avec deux paliers
5. Fonction maximum de deux fonctions fournies en paramètre
6. Le "et" booléen
7. Le "ou" booléen
8. Détermine si au moins un élément d'un triplet est nul
9. Fonction qui retourne le nombre de jours d'un mois identifié par son numéro.
10. Fonction qui détermine si une année est bissextile.
11. Renvoyer le premier terme non nul d'un triplet d'entiers
12. Renvoyer le nombre d'éléments positifs dans un couple d'entiers
13. Déterminer si trois entiers fournis en paramètre forment un triplet pythagoricien
14. Classer dans l'ordre décroissant un triplet d'entiers
15. Déterminer si dans un triplet d'entiers l'un est égal à la différence des deux autres

TD III

Programmation impérative et références

Exercice 1 — Calcul de sommes

- Implémenter une fonction `somme` qui, prenant en paramètre un entier n, retourne $\sum_{k=1}^{n} k$
- Implémenter une fonction `somme_puiss` qui, prenant en paramètre deux entiers n et p, retourne $\sum_{k=1}^{n} k^p$
- Implémenter une fonction `seuil` qui, prenant en paramètre un flottant x, retourne le plus petit entier naturel non nul n tel que $\sum_{k=1}^{n} \frac{1}{k} > x$
- Implémenter une fonction `somme2` qui, prenant en paramètre un entier n, retourne la somme de tous les entiers naturels inférieurs ou égaux à n qui ne sont divisibles ni par 2 ni par 3.

Exercice 2 — Arithmétique

- Implémenter une fonction `test_prime` qui teste si un nombre n fourni en paramètre est premier.
- En déduire une fonction `prop_prime` qui, prenant en paramètre un entier n, retourne la proportion de nombres premiers parmi les entiers naturels inférieurs ou égaux à n.
- Implémenter une fonction `pgcd` qui retourne le pgcd de deux entiers non nuls fournis en paramètres, à l'aide d'une simple boucle.
- Implémenter une fonction `pgcd_euclide` qui répond à la même question à l'aide de l'algorithme d'Euclide.
- Implémenter une fonction `somme_carre` qui, prenant en paramètre un entier n, détermine s'il peut s'écrire comme la somme de deux carrés d'entiers.

Exercice 3 — Suites

- Implémenter une fonction `suite_rec_2` qui prenant en paramètre quatre flottants a,b,c et d et un entier n, retourne u_n où la suite est définie par $u_0 = a$, $u_1 = b$ puis $u_{n+2} = c.u_{n+1}+d.u_n$.
- En déduire une fonction `somme_fibo` qui calcule la somme des n premiers nombres de Fibonacci.
- Implémenter une fonction `test_fibo` qui détermine si un entier n, fourni en paramètre, est un nombre de Fibonacci.
- Implémenter une fonction `test_syr` qui teste la conjecture de Syracuse jusqu'à un entier n fourni en paramètre.

Exercice 4 — Palindrome

- Implémenter une fonction `test_palyn` qui détermine si une chaîne de caractères fournie en paramètre est un palindrome.
- Déterminer la somme des entiers inférieurs à un million dont l'écriture en base 10 est un palindrome.

Exercice 5 — Base 2

- Implémenter une fonction `base_10_to_2` qui prenant en paramètre un entier n, retourne son écriture en base 2 sous forme d'une chaîne de caractères.
- Implémenter la fonction réciproque `base_2_to_10`.
- Tester qu'elle sont bien réciproque jusque $N = 10000$.

Exercice 6 — Décomposition d'entier

On se donne un entier n et trois entiers a, b, c non nuls. On souhaite implémenter une fonction qui détermine de combien de manières on obtenir n comme somme de nombres égaux à a, b ou c.

- Proposer une version naïve.
- Proposer une version optimisée de complexité linéaire (utilisant un tableau).

Évaluer la différence de complexité.

TD IV

Tableaux

Exercice 1 — Fonctions de base sur les tableaux
Implémenter les fonctions suivantes pour des tableaux :

- `max_tab` qui détermine la valeur maximale d'un tableau.
- `ind_max` qui détermine l'indice de la valeur maximale d'un tableau.
- `somme_tab` qui retourne la somme des valeurs d'un tableau d'entiers.
- `retourne_tab_en_place` qui symétrise un tableau (sur place) et `retourne_tab` qui fait la même chose en créant un nouveau tableau.
- `permut_circ` qui décale vers la droite (sur place) tous les termes d'un tableau, le dernier revenant en première position.
- `map_tabl` qui prenant en paramètre un tableau `t` et une fonction f, applique f sur tous les éléments du tableau `t`. (une version sur place et une autre créant un nouveau tableau)
- `positif_tab` qui prenant en paramètre un tableau d'entiers, remplace toutes les valeurs négatives par 0. (sur place)
- `positif2_tab` qui prenant en paramètre un tableau crée un nouveau tableau ne contenant que les termes positifs du tableau fourni en paramètre.
- `ajoute_tab` qui prend en paramètre un tableau `t`, un élément x et un indice i et qui retourne un nouveau tableau correspondant au tableau `t` dans lequel x est inséré en position i.
- `supp_tab` qui prenant en paramètres un tableau `t` et un indice i, retourne un couple `t', x` où `t'` correspond au tableau `t` dans lequel l'élément d'indice i est supprimé, et x cet élément.
- `test_sym` qui teste si un tableau est symétrique.
- `test_cr` qui teste si un tableau est classé dans l'ordre croissant.
- `tabl_entiers` qui prenant en paramètre un entier n, retourne le tableau `[|1; 2; ...; n|]`.
- `sous_tabl_pos` qui, prenant en paramètre un tableau d'entiers `t`, retourne le plus long sous tableau constitués d'éléments positifs consécutifs.

Exercice 2 — Recherche d'élément
Implémenter les fonctions suivantes :

- `recherche`, qui prenant en paramètre un tableau `t` et un élément x, teste la présence de x dans ce tableau.
- `recherche_dicho` qui réalise la même chose sur un tableau trié par ordre croissant, en procédant par dichotomie.

Exercice 3 — Coefficients binomiaux et suite de Fibonacci
Implémenter les fonctions suivantes :

- `Fibo_tab` qui calcule des n premiers nombres de Fibonacci à l'aide d'un tableau.
- `fact` qui, prenant en paramètre un entier n, retourne $n!$
- `coef_binom1` qui, prenant en paramètre deux entiers k et n, retourne $\binom{n}{k}$. Quel est le gros défaut de cette méthode ?

- `coef_binom2` qui calcule $\binom{n}{k}$ en utilisant le triangle de Pascal et un unique tableau.

Exercice 4 — Statistiques
Implémenter des fonctions qui calculent la moyenne, la variance et l'écart-type d'un tableau d'entiers.

Exercice 5 — Alea jacta est
Implémenter les fonctions suivantes en utilisant le module `random`.
- `elem_alea` qui prenant en paramètre un tableau, retourne un élément de ce tableau choisi pseudo-aléatoirement.
- `transp_alea` qui transpose (en place) deux éléments aléatoirement dans un tableau.
- `permut_alea` qui prenant en paramètre un tableau, permute ses éléments de manière aléatoire. On ne demande pas à ce que toutes les permutations soient équiprobables, mais on demande à ce qu'elles soient toutes possibles.
- Réfléchir à une manière de traiter la question précédente avec une équiprobabilité sur les permutations possibles du tableau. (Bonus +++)

Exercice 6 — Arithmétique
Implémenter les fonctions suivantes :
- `crible`, qui prenant en paramètre un entier n, retourne un tableau de booléens de taille $n+1$, chaque case indiquant si l'entier correspondant est premier ou non.
- `tabl_prime` qui retourne un tableau contenant les nombres premiers inférieurs ou égaux à n.
- `test_Golbach` qui prenant en paramètre un entier n, vérifie la conjecture de Golbach jusque cette valeur.
- `euler12bis` qui détermine le plus nombre triangulaire, c'est à dire la de la forme $\frac{n(n+1)}{2}$, possédant strictement plus de 1000 diviseurs. Cette fonction devra retourner le résultat en quelques secondes au plus.

Exercice 7 — Somme de cubes
Implémenter une fonction `test_somme_cubes` qui détermine si un entier fourni en paramètre peut s'écrire comme somme de cubes d'entiers différents de 1.

Exercice 8 — Matrices
On modélise une matrice à l'aide d'un tableau de tableaux, chaque sous-tableau représentant une ligne de la matrice. Implémenter les opérations suivantes :
- `mat_nulle` retourne une matrice nulle de taille n, p.
- `identite` retourne la matrice identité de taille n.
- `somme_mat` réalise la somme de deux matrices fournies en paramètre quand cela est possible.
- `prod_mat` réalise le produit de deux matrices.
- `transpose` transpose (sur place) une matrice fournie en paramètre.

Exercice 9 — Les deux plus proches
Proposer une fonction qui prenant en paramètre une liste contenant au moins deux entiers, détermine les deux éléments les plus proches.
- Proposer une version naïve et préciser sa complexité.
- Proposer une version améliorée, sachant que l'on peut trier un tableau en complexité $\mathcal{O}\big(n.\ln(n)\big)$.

Exercice 10 — Décomposition d'entier
On se donne un entiers n et trois entiers a, b, c non nuls. On souhaite implémenter une fonction qui détermine de combien de manières on obtenir n comme somme de nombres égaux à a, b ou c.
- Proposer une version naïve.
- Proposer une version optimisée de complexité linéaire.

Évaluer la différence de complexité.

TD V

Récursivité sans liste

Exercice 1 — Exercices de base

Implémenter chacune des fonctions suivantes de manière récursive :

- `puiss` calcule a^n où a et n sont des entiers fournis en paramètre.
- `expo_rap` réalise la même chose mais à l'aide de la méthode d'exponentiation rapide.
- `fact` calcule $n!$ où n est un entier fourni en paramètre.
- `Syr_rec` prend en paramètre un entier a et retourne le nombre d'itérations nécessaires dans la suite de Syracuse pour revenir la première fois à 1 en partant de a.
- `somme_puiss` prend en paramètre deux entiers n et p et retourne $\sum_{k=1}^{n} k^p$.
- `fibo_rec` retourne le n-ième nombre de Fibonacci. Que pensez-vous de la complexité obtenue ?
- `coef_binom_rec` calcule de manière récursive $\binom{n}{k}$. Que penser de la complexité ?
- `hanoi` prend en paramètre un entier n et retourne le nombre minimal de déplacements nécessaires pour résoudre le problème de la tour de Hanoï à n disques. On pourra ensuite afficher la liste des déplacements.
- `heron` prends en paramètre un entier n et retourne le n-ième terme de la suite définie par $u_0 = 1$ puis $u_{n+1} = \frac{1}{2}.\left(1 + \frac{2}{u_n}\right)$
- `suite_rec` prend en paramètre une fonction f, un flottant a et un entier n et retourne le n-ième terme de la suite définie par $u_0 = a$ puis $u_{n+1} = f(u_n)$.
- `modulo` prend en paramètre deux entier n et d ($\neq 0$) et retourne le reste de la division euclidienne de n par d
- `pgcd` prend en paramètre deux entiers et retourne leur pgcd, calculé de manière récursive grâce à l'algorithme d'Euclide.
- `test_palin` détermine si une chaîne de caractères fournie en paramètre est un palindrome. (utiliser `sub_string`)

Exercice 2 — Pour aller un peu plus loin
Implémenter chacune des fonctions suivantes de manière récursive :

- `tchebychev` prend en paramètres deux flottants `cosx` et `sinx` (désignants respectivement $\cos(x)$ et $\sin(x)$), un entier naturel n, et retourne le couple $cos(nx), sin(nx)$.
- `iteration` prend en paramètre une fonction f et un entier naturel n et retourne f^n (au sens de la composition).
- Il est possible d'attribuer à chaque rationnel positif $\frac{a}{b}$ un unique entier positif (son numéro) et cela de manière bijective. `num_rat` prend en paramètre deux entiers positifs a et b et retourne un numéro associé de manière unique à la fraction $\frac{a}{b}$.
- `monnaie` détermine de combien de manières on peut donner n euros avec des pièces de 3, 7 et 13 euros. On fera intervenir dans la récursion le nombre de types de pièces utilisées.

Exercice 3 — Pour aller nettement plus loin (et à titre culturel avant tout)
Implémenter chacune des fonctions suivantes de manière récursive :

- `pb_8_reines` détermine de manière récursive le nombre de solutions au problème des 8 reines. On utilisera pour cela une fonction auxiliaire avec en paramètre la liste des reines déjà placées.
- `test_sudoku` prend en paramètre un tableau de tableaux, modélisant une grille de sudoku, et détermine si cette grille est possible. On créera une fonction copiant une grille avant de commencer cet exercice.
- `pb_cavalier` détermine le nombre de solutions du problème du cavalier sur un échiquier, ce dernier partant initialement de la case `a1`.
- `plus_proche` détermine les deux points les plus proches dans un nuage de points, de manière optimisée. (cela peut faire l'objet d'un TP complet)
- `text` prend en paramètre un entier n pair, et détermine le nombre de manière de paver un quadrillage de taille $n \times n$ avec des dominos de taille 2×1.

TD VI

Liste

s

Exercice 1 — Fonctions de base sur les listes

Implémenter chacune des fonctions suivantes sur les listes (on ne se soucie pas pour l'instant d'optimiser la complexité, ni de chercher la récursivité terminale) :

- `long` retourne la longueur de la liste.
- `somme` retourne la somme des éléments d'une liste d'entiers.
- `test_pres` teste la présence d'un élément x fourni en paramètre dans une liste.
- `avant_dern` retourne l'avant dernier élément (s'il existe) d'une liste.
- `map_liste` applique une fonction f sur tous les éléments d'une liste.
- `test_cst_x` détermine si une liste est constante égale à x.
- `test_cst` détermine si une liste est constante.
- `test_croiss` détermine si une liste est triée par ordre croissant.
- `mirror` symétrise une liste.
- `test_mono` détermine si une liste est monotone.
- `max_liste` retourne le maximum d'une liste.
- `ind_max` retourne la position du maximum dans une liste.
- `ajout_deb` ajoute un élément x au début d'une liste.
- `ajout_fin` ajoute un élément x à la fin d'une liste.
- `supprim_fin` supprime le dernier élément d'une liste.
- `supprim_deb` supprime le premier élément d'une liste.
- `enleve` supprime toutes les occurences d'un élément x fourni en paramètre d'une liste.
- `concat` réalise la concaténation de deux listes.
- `fusion` réalise la fusion de deux listes triées dans l'ordre croissant, le résultat étant aussi une liste croissante.
- `union` réalise l'union de deux listes au sens ensembliste.
- `inter` réalise l'intersection de deux listes au sens ensembliste.
- `supprime_n` supprime les n premiers éléments d'une liste.
- `test_doublon` teste si une liste contient un doublon.
- `supp_doublon` supprime les doublons dans une liste.
- `suff_liste` retourne la liste des suffixes d'une liste.
- `pref_liste` retourne la liste des préfixes d'une liste.
- `plus_proche` détermine l'élément le plus proche dans une liste d'entiers, d'un flottant x fourni en paramètre.
- `seuil` prend en paramètre un entier x et sépare une liste en deux listes constituées respectivement des éléments strictement inférieurs à x et de ceux supérieurs ou égaux à x.

Exercice 2 — Listes et tableaux
Implémenter les fonctions suivantes :

- `tabl_to_list` "transforme" un tableau en liste.
- `list_to_tabl` "transforme" une liste en tableau.
- `rand_list` génère des listes aléatoires d'entiers (à vous de choisir les paramètres)
- `rand_tabl` génère des tableaux aléatoires de flottants.
- `test_conv` prend un paramètre un entier n et réalise n tests dans les deux sens, sur des listes et des tableaux aléatoires, pour vérifier que les deux premières fonctions sont bien réciproques l'une de l'autre.

Exercice 3 — Changement de base
On souhaite modéliser un nombre en base 2 par une liste de booléens (`true` pour 1, `false` pour 0) les chiffres étant classés dans la liste dans l'ordre inverse de leur écriture. Implémenter les fonctions suivantes :

- `base_10_to_2` associe à un entier fourni en paramètre, une liste de booléens représentant son écriture en base 2.
- `base_2_to_10` réalise l'opération inverse.
- `test_binaire` réalise des tests aléatoires sur le même modèle que celui de l'exercice précédent.

Exercice 4 — Meilleure coupure
On dispose d'une liste et d'un tableau de booléen. On souhaite savoir où "couper" la structure pour que le nombre de `true` à gauche de la coupure plus le nombre de `false` à droite soit maximal.

- `coupure_max_tab` répondra à cette question pour un tableau.
- `coupure_max_list` répondra à cette même question pour une liste.