TP2 - Fonctions récursives sur les entiers

> **Remarque** pour ceux qui voulaient installer rapidement et tester Tuareg/Merlin/learn-ocaml-mode: il y avait un souci de configuration lié au paquet opam `conf-libssl.3` qui a retardé la release de `learn-ocaml-client.0.13.0` (requise par learn-ocaml-mode). Cette release vient d'être faite… donc on vous informera sur Moodle dès que le tuto d'installation sera finalisé (comme ça vous pourrez bénéficier de l'IDE local dès la semaine prochaine, avant le TP3 :)

Cet exercice en ligne est un peu plus long que les précédents, mais vous avez quinze jours jusqu'au prochain TP pour le terminer.

Si vous avez déjà abordé la récursion en TD, vous pouvez commencer cet exercice en ligne directement par la partie B.

Mais si pour votre groupe ce créneau de TP survient avant votre créneau de Cours/TD consacré à la récursion, nous vous suggérons de **commencer par la partie A**.

N'hésitez pas à solliciter votre encadrant pour toute question durant la séance, en cas de doute, ou pour lui demander une **revue de votre code**.

Alternativement, vous pouvez aussi cliquer **sur le bouton `Noter!`** (qui ressemble au bouton `Sync` mais envoie sur la plateforme PFITAXEL à la fois votre code + le résultat des tests) et demander à votre encadrant qu'il relise votre code dans le dashboard enseignant de PFITAXEL, après lui avoir rappelé **votre pseudonyme `prenomN@tp00`**.

Partie A - Fonctions et n-uplets

Exercice A.1 - Synthèse d'expressions

Écrivez des expressions ayant exactement le type spécifié ci-dessous :

- `e1 : (int * bool) * int`
- `e2 : ('a -> 'a) * char`

Exercice A.2 - Synthèse de fonctions

Écrivez des fonctions ayant exactement le type spécifié ci-dessous (sans utiliser d'annotation de type) :

- `f1 : int -> bool -> (int * bool)`
- `f2 : 'a -> bool`
- `f3 : 'a -> 'a -> int`
- `f4 : 'a * 'b -> 'b * 'a`
- `f5 : ('a -> 'b * 'c) -> ('b -> 'c -> 'd) -> ('d * bool -> 'e) -> 'a -> 'a -> 'e`
- `f6 : ('a -> 'b) -> ('a -> 'b -> 'c) -> 'a -> 'c`
- `f7 : ('a -> 'b) * ('a * 'b -> 'c) * 'a -> 'c`
- `f8 : 'a -> 'a -> 'b -> 'b -> 'a * 'b`

Exercice A.3 - Écriture de petites fonctions (Thème 3 / Exercice 5)

1) Écrire les définitions des fonctions suivantes :

- `cube` qui calcule le cube d'un entier
- `quad` qui calcule le quadruple de son argument
- `estPair` qui indique par un booléen si son argument est un entier pair ou non
- `max3` qui retourne le plus grand de 3 nombres
- `discriminant` qui, étant donnés les coefficients entiers `a`, `b` et `c` d'une équation du second degré calcule le discriminant `Delta = b² - 4ac`

2) En utilisant la fonction `estPair`, écrire la fonction `estImpair` qui indique si son argument est un entier impair.

3) En utilisant les fonctions ci-dessus, écrire une fonction `choix` à un argument, qui calcule le cube d'un entier s'il est pair et le quadruple sinon.

4) En utilisant la fonction `discriminant`, écrire la fonction `nbRac : int -> int -> int -> int` qui indique le nombre de racines réelles (sans les calculer) d'une équation du second degré à coefficients entiers.

Vous ne devrez pas effectuer plusieurs fois le même calcul.

5) En se référant aux tables de vérités, définir les fonctions permettant de déterminer la combinaison de deux valeurs booléennes `a` et `b` par les opérateurs `xor` (ou exclusif), `nor` et `nand` :

```
a     b     a xor b  a nor b  a nand b
false false   false    true     true
false true    true     false    true
true  false   true     false    true
true  true    false    false    false
```

Partie B - Récursion

Exercice B.1 - Numération décimale

a) Écrire la fonction `sommeChiffres` prenant en argument un entier `n>=0` et calculant la somme des chiffres contenus dans `n`. On lèvera une exception avec `failwith` en cas de valeur négative.

b) Écrire la fonction `sommeIteree` prenant en argument un entier `n>=0` et itérant le calcul effectué par `sommeChiffres` jusqu'à ce que le résultat soit inférieur à `10`.

```
sommeChiffres 456 => 15
sommeIteree 456 => 6
```

On lèvera une exception avec `failwith` en cas de valeur négative.

Écrire les fonctions suivantes, prenant en argument un entier `n>=0`
(cette fois-ci il n'est pas demandé de tester le cas d'erreur `n<0`) :

1. `dernierCh`, qui renvoie le dernier chiffre d'un entier non nul
2. `toutSaufDer`, qui renvoie son argument privé de son dernier chiffre

3. `premierCh`, qui renvoie le premier chiffre d'un entier non nul
4. `toutSaufPrem`, qui renvoie son argument privé de son premier chiffre
5. `estPalindrome`, qui teste si un entier (ne contenant pas le chiffre 0) est un palindrome
6. `nbOccs : int -> int -> int`, qui compte le nombre d'occurrence d'un chiffre dans l'écriture décimale d'un entier.
   (par exemple, `nbOccs 0 100 => 2`)

Exercice B.2 - Fonction d'itération

1) Écrire la fonction `iterer` qui, étant donné un entier `n`, une fonction unaire `f` et un argument `x`, réalise l'application `n` fois de `f` (si `n = 0` le résultat est l'argument `x`).

Par exemple : `iterer 2 (fun x -> x+10) 3` renvoie `23`.

(*Remarque :* il y a 2 solutions possibles suivant la position de l'appel récursif.)

Autre version de iterer :

2.1) Définir la fonction unaire `id` correspondant à la fonction identité. Vérifier son type.

2.2) Définir la fonction `compose` dont le type est `('a ->'b) -> ('c -> 'a) -> 'c -> 'b`.

2.3) Écrire, en réutilisant `compose` et `id`, la fonction `iterer2` à 2 arguments (un entier `n` et une fonction unaire `f`), renvoyant la fonction réalisant `n` fois l'application de `f`; si `n = 0`, le résultat est l'identité.

Par exemple, `let f = iterer2 4 (fun x -> x+10) in f 3` renvoie `43`.

3.1) Écrire la fonction `itererBis` prenant en argument une fonction unaire `f`, un prédicat *unaire* `p` (c.-à-d. une fonction *à un argument* renvoyant un booléen) et un argument `x`, et itérant l'application de `f` à `x` jusqu'à-ce-que `x` vérifie le prédicat `p`, le résultat alors renvoyé étant `x`.

Exemple : prendre la moitié de `x` jusqu'à obtenir une valeur `< 10`.

```
itererBis (fun x -> x / 2) (fun x -> x < 10) 45 => 5
```

3.2) Définir la fonction `sommeIteree2`, utilisant `itererBis` pour effectuer le même calcul que `sommeIteree`.

Exercice B.3 - Fonctions classiques

1) (Commencez par écrire son type, puis) implémentez la fonction `qqsoit` prenant en argument un entier `n` et un prédicat unaire `p` et renvoyant `true` ssi tous les éléments de `{1, 2, …, n}` vérifient le prédicat `p`.

2) Écrire la fonction d'Ackermann définie sur les entiers naturels comme suit :

```
ack(m,n) = n + 1                 si m = 0
ack(m,n) = ack(m-1,1)            si m > 0 et n = 0
ack(m,n) = ack(m-1,ack(m,n-1))   si m > 0 et n > 0
```

Attention, la fonction d'Ackermann croît très rapidement... Ainsi `ack(4,2)` contient 19729 chiffres et dépasse le nombre d'atomes estimé dans l'univers. Éviter de le calculer.