Option Informatique

TP $n°1$: prise en main de Caml

Important : gardez une copie de votre code sur votre clé. L'extension d'un fichier Caml est `.ml`

Note : même lorsque ce n'est pas demandé, écrire des fonctions auxilliaires n'est jamais une mauvaise idée.

Quelques types en Caml

Exercice 1.

1. Déclarer une variable `a` de valeur `4611686018427387903`. *Passion : copier-coller.*
2. Comparer `a-1` et `a+1` à l'aide de `<`, `=` et `>`. Les afficher pour comprendre ce qui se passe !

Exercice 2. Comparer les flottants `4611686018427387902.` et `4611686018427387904.` à l'aide de `<`, `=` et `>`. Expliquer.

Exercice 3. Vérifier que && et || ont bien la table de vérité attendue.

Exercice 4.

1. Stocker la chaîne de caractère `"Hello, world !"` dans une variable `bonjour`.
2. Extraire le cinquième caractère. Est-ce un `'o'` ? Pourquoi ?
3. Définir une chaine de caractère composée de mille `'a'`.

Exercice 5. Voyons si on a compris comment marche le *pattern-matching* : **réécrire** la fonction ||. L'appeler `ou`.

Exercice 6. Plus sérieux : *un peu d'ordre supérieur.*
Écrire une fonction (récursive) `itere : ('a → 'a) → int → 'a → 'a` prenant comme argument une fonction f et un entier n, et retournant la fonction $\underbrace{f \circ f \circ \cdots \circ f}_{n \text{ fois}}$.
Exemple : d'après le cours de maths, `itere sin n 1.` *doit être de plus en plus proche de* 0. *lorsque* n *augmente.*

Exercice 7.

1. Faire afficher le type de `()`.
2. Écrire une fonction prenant un argument de type arbiraire, et ne retournant rien.
3. Écrire une fonction constante prenant 0 argument et retournant l'entier 2.
   Attention, ce n'est pas pareil qu'une variable égale à 2.

Exercice 8. Écrire une fonction sans argument, qui ne retourne rien, mais qui a pour effet de bord d'afficher le message `Hello, world !` sur la sortie standard. **Attention, ça n'a rien à voir avec une fonction qui retourne une chaîne de caractères !**

Exercice 9. Essayons de comprendre comment sont codées certaines fonctions bien utiles, c'est un classique de concours de nous demander de les réécrire.

1. Que fait `List.length` ? Écrire une fonction `longueur` ayant le même comportement.
2. Que fait `List.map` ? Écrire une fonction `mapper` ayant le même comportement.
3. Que fait `List.concat` ? Ah ah, elle n'est pas dans l'aide mémoire celle-là. Écrire une fonction `concatener` ayant le même comportement.

Exercice 10. Écrire une fonction `recherche_dichotomique : 'a → 'a vect → bool` qui prend comme argument un élément de type `'a` et un tableau de type `'a array` **supposé trié**, et rend comme résultat le booléen indiquant si l'élément est ou pas dans le tableau, en utilisant une recherche dichotomique. On peut le faire avec une boucle while, mais faisons l'effort de le coder par récursivité.

Quelques aspects impératifs

Exercice 11.

1. Sans tester, quel est le type et le comportement de cette fonction ?

```
let test1 n =
  let a = n-1 in
  let b = a-1 in
  a+b;;
```

   Vérifier.

2. Idem avec :

```
let test2 x =
  let aux n =
    let a = n-1 in
    a in
  let b = x+1 in
  aux b;;
```

3. Idem avec :

```
let test3 n =
  let aux a b = match a with
    |0 -> b
    |_ -> aux (a-1) (b+1) in
  aux n 0;;
```

Exercice 12. Créer une variable `n` de type `int ref` pointant vers la valeur entière de votre mois de naissance (entre 1 et 12). Modifiez-la en lui ajoutant votre jour de naissance (entre 1 et 31).

Exercice 13. Écrire une fonction `v2 : int → int` qui calcule la valuation 2-adique d'un entier. C'est très facile (plus facile) à faire par récursivité mais faisons l'effort d'utiliser une boucle.

Exercice 14. Créer un tableau de longueur 100 contenant les entiers de 0 à 99 dans cet ordre. On peut le faire par récursivité mais faisons l'effort d'utiliser une boucle.

Exercice 15. Réécrire une fonction `factorielle : int → int` avec une boucle et une variable de type `int ref`.

Exercice 16. Écrire une fonction `degre2 : float →float →float →float list` telle que `degre2 a b c` donne la liste des racines réelles du trinôme $aX^2 + bX + c$.

Exercice 17. Proposer un exemple où il est agréable de pouvoir écrire plusieurs instructions dans une (ou plusieurs) des branches d'une instruction conditionnelle. L'implémenter.

Exercice 18.

1. Écrire une fonction `division_euclidienne : int → int → int*int` telle que `division_euclidienne a b` retourne le couple `(q,r)` obtenu avec `mod` et `quo`.
2. Cette fonction provoque une exception pour `b`$= 0$. La modifier à l'aide d'une instruction conditionnelle pour qu'elle retourne le couple $(-1,-1)$ dans ce cas.
3. Plutôt que d'utiliser une instruction conditionnelle, obtenez le même résultat en rattrapant l'exception.

Exercice 19. Écrire une fonction `est_premier : int → bool` qui prend comme argument un entier n et retourne un booléen indiquant si l'entier est ou pas premier : tester 2 à part, puis les diviseurs impairs inférieurs à $\lfloor \frac{n}{2} \rfloor$, en s'arrêtant dès qu'un diviseur est trouvé. Ceci peut être réalisé par récursivité ou avec une boucle while, mais pour le sport on peut aussi rattraper une exception dans une boucle for.