Option informatique, DS 1

Durée : 1h30

Rappels de Caml

- Opérateurs sur les entiers : `+`, `-`, `*`, et `mod` pour le modulo.
- Instructions conditionnelles : `if ... then ... else ...`. Type commun dans les blocs `then` et `else`, une seule instruction dans chaque bloc ou encadrement par des parenthèses ou `begin ... end`.
- boucles :
  - `for i = id to if do ... done ;` itère les opérations du corps de boucle (de type `unit`) pour tout $i \in [\![i_d, i_f]\!]$. Si cet ensemble est vide, le corps n'est pas exécuté.
  - `for i = id downto if do ... done ;`. Même chose, mais par pas de -1. On doit avoir $i_d \geq i_f$ pour qu'au moins un tour soit effectué.
  - `while ... do ... done ;` pour une boucle `while`.
- Tableaux : dans la suite, `t` est un tableau.
  - taille d'un tableau : `Array.length t`
  - élements d'un tableau de taille n : `t.(0)`,..., `t.(n-1)`.
  - modifier un élément : `t.(i) <- x`.
  - Création : `Array.make n x` crée un tableau de taille n rempli de x ;
  - Copie : `Array.copy t`.
- Références : impératives quand il s'agit de « modifier » une variable !
  - créer une référence vers `x` : `ref x`. Par exemple : `let i=ref 0 in ...`) ;
  - modifier la valeur pointée par la référence `r` : `r:= ...` ;
  - accéder à la valeur pointée : `!r`.
- Listes chaînées : c'est une structure immuable (pas de modification, simplement la création de nouveaux objets)
  - `[]` : liste vide ;
  - `x::q` liste obtenue par ajout de `x` en tête de `q`.
  - `List.hd q`, `List.tl q` : tête et queue d'une liste `q` ;
  - mais on procède souvent par filtrage sur le motif `x::q`.

Exercice 1. *Reconnaître le type.* Reconnaître le type des fonctions suivantes (on ne demande pas nécessairement de justification, par contre sans justification la moindre erreur est sanctionnée d'un zéro !).

```
let f x = 2 * x + 1 ;;
let g t x = t.(x) * 3 ;;
let h x = x 4 ;;
let i x = x exp ;;
let j x y = y::x ;;
let k x y = if not x then y else 2*y ;;
```

Exercice 2. *Fonctionnelles.* Déclarer les fonctions suivantes, et prévoir leur type. On demande des versions curryfiées.

- la fonction qui à f et g associe $f \circ g \circ f$.
- la fonction qui à f et g associe $x \mapsto \min(f(x), g(x), 0)$

Exercice 3. *Pile d'entiers bornée.* On implémente dans cet exercice une structure de pile bornée, ne pouvant contenir que des entiers. La représentation choisie est très similaire à celle du cours, néanmoins comme les éléments sont des entiers, une pile `p` de capacité c sera vue comme un simple tableau d'entiers de taille $c+1$. Les fonctions à écrire sont les suivantes :

- `creer_pile : int -> int array`, telle que `creer_pile c` crée une pile d'entier (vide) de capacité `c`

— `pile_vide : int array -> bool`, teste si la pile passée en entrée est vide.

— `empiler : int array -> int -> unit`, telle que `empiler p x` rajoute `x` au sommet de `p` si celle-ci n'est pas pleine.

— `sommet : int array -> int`, renvoie le sommet d'une pile non vide ;

— `depiler : int array -> int`, dépile et renvoie le sommet d'une pile non vide.

Dans la représentation choisie, pour une pile `p` :

— `p.(0)` est le nombre d'éléments effectivement présents dans `p` ;

— Avec $n =$ `p.(0)`, si $n > 0$ les éléments présents dans la pile sont `p.(1)`$, \dots,$ `p.(`n`)`, dans l'ordre de la base de la pile au sommet.

Écrire les fonctions de pile, avec cette représentation. Pour les fonctions `empiler`, `sommet` et `depiler`, on pourra supposer que la pile est respectivement non pleine et non vide, sans le vérifier.

Exercice 4. *Pile de Fibonacci.* Dans cet exercice, on suppose donnée une structure de pile d'entiers similaire à la précédente. Néanmoins, vous ne devez manipuler la pile d'entiers que via les fonctions de pile, sans hypothèse sur la représentation effective (j'ai le droit de changer l'implémentation précédente pour une autre si j'ai envie, c'est moi qui écris le sujet !)

1. Écrire une fonction `maj p` prenant en entrée une pile d'entiers, supposée contenir au moins deux éléments, et rajoutant au sommet de `p` la somme des deux éléments actuellement au sommet. Par exemple si `p` contient 5 au sommet et 3 en dessous, alors `maj p` empile 8 sur la pile.
2. La suite de Fibonacci est définie par $F_0 = F_1 = 1$, et $F_n = F_{n-2} + F_{n-1}$ pour $n \geq 2$. Écrire une fonction `pile_fibo n` créant et renvoyant une pile d'entiers contenant les éléments $F_0, \dots, F_n$ (n est un entier qu'on pourra supposé au moins égal à 1), avec F_n au sommet.
3. Écrire une fonction `affiche_pile p` prenant en entrée une pile d'entiers, et affichant à l'écran ses éléments. On s'autorise à vider la pile. On utilisera `print_int` et `print_string`.

```
# afficher (pile_fibo 10) ;;
55 34 21 13 8 5 3 2 1 1 0 - : unit = ()
```

Exercice 5. Les nombres de Catalan satisfont la relation $c_0 = 1$ et à la relation $c_n = c_{n-1}\frac{2(2n-1)}{n+1}$ pour tout $n \geq 1$ (ce sont des entiers). Écrire une fonction récursive `catalan: int -> int` renvoyant c_n.

```
# catalan 5 ;;
- : int = 42
```

Exercice 6. *n-ème élément d'une liste.* Écrire une fonction `nth : int -> 'a list -> 'a` renvoyant l'élément d'indice n dans une liste (l'élément de tête ayant pour indice 0). On renverra une erreur si la liste a strictement moins de $n + 1$ éléments.

```
# nth 3 [7;8;9;10;11] ;;
- : int = 10
# nth 3 [7;8;9] ;;
Exception: Failure "pas assez d'éléments".
```

Exercice 7. *Deux plus petits éléments d'une liste.* Écrire une fonction `min2list : 'a list -> 'a * 'a` qui retourne le couple des deux plus petits éléments. On renverra une erreur si la liste passée en paramètre a strictement moins de deux éléments.

```
# min2list [2;8;7;0;5;6;9;11] ;;
- : int * int = (0, 2)
```

Exercice 8. *Section croissante.* Une sous-suite croissante dans un tableau `t` est une séquence d'éléments consécutifs `t.(`i`)`,`t.(`$i+1$`)`$, \dots,$`t.(`j`)` qui est croissante. On appelle début de la sous-suite l'entier i et longueur de la sous-suite son nombre d'éléments ($j - i + 1$ avec l'exemple précédent). Écrire une fonction `max_sec : 'a array -> int * int` identifiant la plus longue sous-suite croissante dans un tableau, supposé non vide. Le résultat est le couple formé par l'indice de début et la longueur de la sous-suite. Donner la complexité de votre fonction.

```
# max_sec [|2;8;7;1;3;4;5;9;0;8;4;1;6;7|] ;; (* 1,3,4,5,9 demarre a l'indice 3 *)
- : int * int = (3, 5)
```