TP n° 5 : Programmation impérative

Quelques rappels :

- Les références sont créées avec `let identifiant = ref expression`, le contenu de la référence s'obtient avec `!identifiant` et sa modification avec `identifiant := expression` (qui est de type `unit`). L'entier sur lequel pointe une référence de type `int ref` peut être incrémenté ou décrémenté à l'aide des fonctions `incr` et `decr`.
- On peut exécuter des séquences d'expressions, séparées par des virgules, lesquelles seront exécutées dans l'ordre d'affichage. Toutes les expressions sauf la dernière doivent être de type `unit`. L'expression qui en résulte est du type de la dernière expression de la séquence. Si nécessaire, les séquences peuvent être entourées par `begin` et `end`.

```
begin
  expression1 ;
  expression2 ;
  ... ;
  expressionN
end
```

- Les boucles non conditionnelles s'écrivent :

```
for indice = debut to fin do
  expression ou séquence d'expr.
done;
```

```
for indice = debut downto fin do
  expression ou séquence d'expr.
done;
```

- Les boucles conditionnelles s'écrivent :

```
while condition do
  expression ou séquence d'expressions
done;
```

On s'efforcera de réfléchir à des invariants de boucle (en entrée ou en sortie) ainsi qu'à des justifications de terminaison de boucles (conditionnelles) pour chaque boucle écrite.

On indiquera le type des fonctions lorsqu'il n'est pas précisé dans l'énoncé.

1 Autour des références

EXERCICE 1 (*évaluation des références*)

Anticiper, tester et comprendre les résultats produits par les instructions suivantes.

```
let b = 2;;
let double x = b * x;;
let b = 145;;
double 3;;
```

```
let b = ref 2;;
let double x =! b * x;;
b := 145;;
double 3;;
```

EXERCICE 2 (*échange de contenus*)

Déterminer le type [1] puis écrire une fonction `echange` échangeant les contenus de deux références.

1. Ne trichez pas !

2 Un grand classique

EXERCICE 3 (*Suite de Fibonacci*)

1. Écrire une fonction impérative `fibonacci : int -> int` calculant le terme d'indice n de la suite de Fibonacci définie par $F_0 = 0$, $F_1 = 1$ et pour tout $n \geqslant 0$, $F_{n+2} = F_n + F_{n+1}$, en utilisant deux références.
2. Écrire une fonction impérative `fibonacci_bis : int -> int` calculant le terme d'indice n de la suite de Fibonacci en utilisant une seule référence.

3 Un peu d'arithmétique

EXERCICE 4 (*Algorithme d'Euclide*[2])

1. Écrire une fonction `euclide : int -> int -> int` programmant la version impérative de l'algorithme d'Euclide.
2. Modifier la fonction pour qu'elle affiche les divisions euclidiennes successives.

```
euclide_affiche (fibonacci 11) (fibonacci 10);;
89 = 55 x 1 + 34
55 = 34 x 1 + 21
34 = 21 x 1 + 13
21 = 13 x 1 + 8
13 = 8 x 1 + 5
8 = 5 x 1 + 3
5 = 3 x 1 + 2
3 = 2 x 1 + 1
2 = 1 x 2 + 0
- : int = 1
```

EXERCICE 5 (*Nombre et somme des chiffres en b*)

Écrire des fonctions `nombre_de_chiffres : int -> int -> int` et `somme_des_chiffres : int -> int -> int` programmant la version impérative du calcul du nombre de chiffres et de la somme des chiffres d'un entier dans une certaine base.

Ces fonctions afficheront les chiffres de l'entier au fur et à mesure qu'ils sont calculés.

```
somme_des_chiffres 145792 10;;
2 9 7 5 4 1
- : int = 28
```

Écrire une fonction `nombre_de_chiffres_bis : int -> int -> int` calculant ce nombre de chiffres sans utiliser de boucle.

EXERCICE 6 (*Test de primalité*)

Écrire une fonction `est_premier : int -> bool` testant si un entier est premier.

EXERCICE 7 (*Nombre et somme des diviseurs, nombres parfaits*)

Écrire des fonctions `nombre_diviseurs : int -> int` et `somme_diviseurs : int -> int` renvoyant respectivement le nombre et la somme des diviseurs positifs d'un entier et une fonction `est_parfait : int -> int` testant si un nombre est parfait, c'est-à-dire s'il est égal à la somme de ses diviseurs stricts.

2. Savez-vous que le **théorème de Lamé** dit que le nombre de divisions nécessaires pour calculer le pgcd de a et de $b < a$ par l'algorithme d'Euclide est majoré par $1 + \lfloor \ln b / \ln \varphi \rfloor$ (qui est lui-même plus petit que 5 fois le nombre de chiffres de b en base 10) où φ est le nombre d'or, majorant atteint pour le calcul du pgcd de deux termes successifs de la suite de Fibonacci ?

4 Exponentiation : encore et toujours...

EXERCICE 8 (*Exponentiations lente et rapide itératives*)

1. Écrire une version itérative du calcul de x^n de manière naïve.
2. On souhaite implémenter l'exponentiation rapide[3] en itératif. Le principe sera le suivant : on remarque que
$$x^{2^{k+1}} = \left(x^{2^k}\right)^2$$
Il suffit donc d'une multiplication pour passer de x^{2^k} à $x^{2^{k+1}}$.
L'idée est alors de décomposer n en base 2. Si n s'écrit
$$n = b_0 + b_1 \cdot 2 + b_2 2^2 + \cdots + b_k 2^k$$
avec $b_0, \cdots, b_k \in \{0, 1\}$ (écriture en base 2) alors
$$x^n = x^{b_0} \left(x^2\right)^{b_1} \cdots \left(x^{2^k}\right)^{b_k}$$
le calcul de x^{2^p} s'effectuant à partir de celui de $x^{2^{p-1}}$.
Par exemple,
   - $19 = 1 + 2 + 2^4$ et
   $$x^{19} = x \times x^2 \times \left(\left(\left(x^2\right)^2\right)^2\right)^2$$
   (6 multiplications nécessaires),
   - $39 = 1 + 2 + 2^2 + 2^5$ et
   $$x^{39} = x \times x^2 \times \left(x^2\right)^2 \times \left(\left(\left(\left(x^2\right)^2\right)^2\right)^2\right)^2$$
   (8 multiplications nécessaires).

5 Des chaînes de caractères

EXERCICE 9 (*Palindrome*)

Écrire une fonction impérative `palindrome : string -> bool` renvoyant `true` si la chaîne passée en argument est palindromique et `false` sinon, faisant le minimum de comparaison de caractères.

```
palindrome "kayak";;
- : bool = true

 palindrome "caml";;
- : bool = false
```

EXERCICE 10 (*Sous-chaine*)

Écrire une fonction impérative `sous_chaine : string -> string -> int` tel que l'appel de `sous_chaine mot texte` renvoie la position de la première apparition de la chaîne `mot` dans la chaîne `texte` si elle y apparaît, et `-1` sinon.

3. C'est en fait la dérécursification de la version récursive (terminale) de l'exponentiation rapide.