UNIVERSITÉ D'ORLÉANS **DEUG MIAS**
IF 2.01 - Informatique **Année 2001-2001**

Feuille d'exercices n°2

Exercice 1

```
let f x = (x = 3*(x/3));;
```

On peut généraliser a la divisibilité par un entier n :

```
let divisible n x = (x = n*(x/n));;
```

Exercice 2

```
let rec succ x = x+1;;
let rec pred x = x-1;;

let rec somme x y = if (y=0)
        then x
        else (somme (succ x) (pred y));;
let rec diff x y = if (y=0)
        then x
        else (diff (pred x) (pred y));;
let rec mult x y = if (y=0)
        then 0
        else somme (mult x (pred y)) x;;
```

Dans ce cas, ajouter x et y coûte $2 * y$ opérations. Idem pour soustraire. Pour multiplier, cela coûte $2x * y$ opérations environ.

Exercice 3

```
let rec pow x y = if (y=0)
        then 1
        else mult (pow x (pred y)) x;;
```

Ce calcul n'est pas très efficace : il coûte une multiplication de x^t par x, à la t-ième opération, et ceci pour t allant de 1 à y. Cela coûte donc

$$\sum_{t=1}^{y} 2 * x^t * x$$

qui est de l'ordre de x^{y+2} opérations.
On peut cependant l'améliorer un peu :

```
let pair = divisible 2;;
let carre x = mult x x;;
let rec newpow x y =  if (y=1)
        then x
        else  if (pair y)
                then carre (newpow x (y/2))
                else mult (newpow x (pred y)) x;;
```

En fait, ceci n'améliore pas le programme. Intuitivement, étant donné que l'on utilise uniquement un incrémenteur et un décrémenteur (excepté pour le calcul de $y/2$, mais c'est un détail), calculer x^y nécessite au minimum d'incrémenter x environ $x^y - x$ fois. La complexité est donc nécessairement de l'ordre (au moins) de x^y.
Utilisons maintenant les opérations de multiplication de CAML pour ce calcul :

```
let pair = divisible 2;;
let carre x = x*x;;
let rec quickpow x y =  if (y=1)
        then x
        else  if (pair y)
                then carre (quickpow x (y/2))
                else x*(quickpow x (y-1));;
```

Cependant, il est toujours préférable d'utiliser la fonction `**` de CAML, qui est appliquée sur des flottants et a donc un plus grand « intervalle d'exactitude ». Calculer par exemple `quickpow 2 30`, et comparer à `2.** 30.`.

Exercice 4

```
let rec div x y = if (y>x)
        then 0
        else 1+div (x-y) y;;
```

Exercice 5

```
open String;;
let rec palindrome s = let l=length(s) in
        if (l<2)
                then true
                else let f x= sub s x 1 and t = sub s 1 (l-2) in
                        (palindrome t && f(0)=f(l-1)) ;;
```

Exercice 6

```
let subdecimal m n = let strm = string_of_int m in
        let l = length strm in
        if (n<1 || n>l)
                then 0
                else int_of_string(sub strm (n-1) 1);;
```

Exercice 7

```
let rec somme f a b = if (b<a)
        then 0.
        else f(a) +. somme f (a+1) b;;
```

Exercice 8

```
let rec integrale f a b h = if ((b<=a)||(h<=0.))
        then 0.
        else h*.f(a) +. integrale f (a+.h) b h;;
```

Calculons alors l'intégrale de la fonction identité sur l'intervalle $[0;5]$. On obtient

```
# let f x = x in integrale f 0. 5. 0.01;;
- : float = 12.525
```

Cependant, graphiquement, il est manifeste qu'on aurait dû obtenir une valeur inférieure à la valeur effective de l'intégrale. C'est un problème d'arrondi (interne) de CAML :

```
# let rec h n a b = if (n=0) then b else h (n-1) a (b+.a);;
val h : int -> float -> float -> float = <fun>
# h 10000 0.0001 0.;;
- : float = 1
# 1. <= h 10000 0.0001 0.;;
- : bool = false
# 1. > h 10000 0.0001 0.;;
- : bool = true
```

Lors du calcul de l'intégrale, CAML ajoute également le « dernier » rectangle, celui pour lequel, théoriquement, `a` = `b`.
Pour éviter ce problème, il faut en quelques sortes redéfinir l'égalité sur les flottants, en disant que « égal » ça veut dire « à peu près égal » :

```
# let valabs x = if (x<=0.) then (-.x) else x;;
val valabs : float -> float = <fun>
# let approx epsilon a b = (valabs(a -. b) <= epsilon);;
val approx : float -> float -> float -> bool = <fun>
# let egal = approx 0.0001;;
val egal : float -> float -> bool = <fun>
# let rec integrale f a b h = if ((b<a)||(egal a b)||(h<=0.))
        then 0.
        else h*.f(a) +. integrale f (a+.h) b h;;
val integrale : (float -> float) -> float -> float -> float -> float = <fun>
# let f x = x;;
val f : 'a -> 'a = <fun>
# integrale f 0. 1. 0.0001;;
- : float = 0.49995
```

Exercice 9

```
# let derivee f x h = (f(x+.h)-. f(x))/. h;;
val derivee : (float -> float) -> float -> float -> float = <fun>
# let abs x = if (x<0.) then -.x else x;;
val abs : float -> float = <fun>
# let precis y e = ((abs y) < e);;
val precis : float -> float -> bool = <fun>
# let ameliore f z h = z -. (f z)/.(derivee f z h);;
val ameliore : (float -> float) -> float -> float -> float = <fun>
# let rec newton f z e h = if (not(precis(f z) e))
        then newton f (ameliore f z h) e h
        else z;;
val newton : (float -> float) -> float -> float -> float -> float = <fun>
```

On peut ainsi calculer $\sqrt{2}$ par exemple :

```
# let f x = x**2. -.2.;;
val f : float -> float = <fun>
# newton f 2. 0.00000000001 0.00001;;
- : float = 1.41421356237
# sqrt 2.;;
- : float = 1.41421356237
```