Programmation fonctionnelle

Répétition 1

12 février 2015

Avant propos

Informations pratiques :

— Assistant : Stéphane Lens
— `lens @ montefiore.ulg.ac.be`, 04/3662619, bureau I82a.
— `http://www.montefiore.ulg.ac.be/~lens`

— L'interpréteur scheme conseillé est *Racket*. (`http://racket-lang.org/download/`)

Quelques points délicats (qui seront répétés, rerépétés, ...) :

— Attention à ne pas confondre `cons`, `list`, et `append`.
— Savoir spécifier est aussi important que savoir programmer. Les énoncés des exercices sont de bons exemples de spécifications.
→ **+-50% des points pour les spécifications !!!**
— Il faut privilégier les solutions simples et courtes.
— Les solutions doivent avoir une complexité acceptable.

Évaluations simples

Exercice 1.
Donner le résultat de l'évaluation des expressions suivantes :

```
3
#t
(+ 1 2)
(/ 2 3)
(+ (* 3 4) 10)
(* 3 (- 12 5))
(+ (+ 2 3) (+ 4 5))
(- (+ 5 8) (+ 2 4))
(define a 4)
a
(quote a)
'a
(define b a)
```

```
b
(define a 6)
a
b
(define c (quote a))
c
(define d #t)
(define Robert 'Bob)
Robert
(car '(a b))
(car (quote (a b)))
(cdr '(a b))
(car (cdr '(a b)))
(cdr (cdr '(a b)))
(cons 'a '())
(cons 'a '(b))
(cons '() '())
(cons '(a) '(b))
(list '(a) '(b))
(append '(a) '(b))
(cons 'a (cons 'b (cons 'c '())))
(car (cons 'a '()))
(cdr (cons 'a '()))
(cons a '(a b))
(cons x '(a b))
(cons (car '(a b c)) (cons
                       (car (cdr '(a b c)))
                       (cons (car (cdr (cdr '(a b c)))) '())))
(cadr '(a b c d))
(cadar '((a b) (c d) (e f)))
```

Premières formes

Exercice 2.
Calculer en une seule forme SCHEME le nombre de secondes dans une année (non bissextile).

Exercice 3.
Écrire une forme qui rend `un` si `x` est égal à 1, ..., `cinq` si `x` est égal à 5 et `inconnu` sinon.

De là, définir une fonction *sayit* qui rend `un` si l'argument est 1, ..., `cinq` si l'argument est égal à 5 et `inconnu` sinon.

Récursion sur les nombres

Exercice 4.
Écrire une fonction $sumOfIntegers$ prenant en argument un naturel n et calculant la somme des naturels inférieurs ou égaux à n.

Exercice 5.
Définir la fonction `mod` qui renvoie le modulo de deux nombres.

Remarque : la fonction *modulo* est prédéfinie, nous la redéfinissons sous un autre nom à titre d'exercice.

Exercice 6.
Définir une fonction `pgcd` qui renvoie le plus grand commun diviseur de deux nombres.

Exercice 7.
Écrire une fonction prenant en argument deux nombres x et n et renvoyant x^n.

Remarque : la fonction *expt* est prédéfinie, nous la redéfinissons sous un autre nom à titre d'exercice.

Récursion sur les listes

Exercice 8.
Définir la fonction *sumList* qui calcule la somme des éléments d'une liste de nombres.

Exercice 9.
Écrire une fonction `removeFirst` à deux arguments `l` et `n` dont les valeurs sont respectivement une liste et un nombre entier, qui retourne la liste privée de la première occurrence de `n`.

`(removeFirst '(2 7 1 7 3 1) 1)` $\Rightarrow$ `(2 7 7 3 1)`

Exercices proposés

Exercice 10.
La variable `phrase` est définie comme suit :

```
(define phrase
  '(((e) x (e r (c (i c e) c (o m (p l (i q))) u e)))))
```

Extraire le `m` au moyen de `car`, `cdr`, `cadr`, ...

Exercice 11.
Construire la liste `(s c h e m e)` au seul moyen de la liste `ls` définie comme suit :

```
(define ls '(C E H M S))
```

et des fonctions `cons`, `car`, `cdr`, `cadr`, ...

Exercice 12.
Définir la fonction `min` qui renvoie le minimum de la liste non vide de nombres donnée en argument.

Exercice 13.
Spécifier la fonction suivante :

```
(define xxx
  (lambda (u)
    (if (null? u)
        0
        (if (> (car u) 0)
            (+ (car u) (xxx (cdr u)))
            (xxx (cdr u))))))
```