Programmation Fonctionnelle en Scheme

Examen L1-I/MI

Université Nice Sophia-Antipolis, Mai 2017

LE FORMULAIRE EN DERNIÈRE PAGE EST LE SEUL DOCUMENT AUTORISÉ. Tâchez de soigner l'indentation, la limpidité et l'exactitude du code. Répondez aux questions sur la copie d'examen qui vous est fournie, ne joignez aucune autre feuille ni intercalaire. N'hésitez pas à utiliser le résultat d'une question précédente même si vous ne l'avez pas traitée. Et ne perdez pas de temps...

1 Exécution d'une Fonction Récursive [1 pt]

Quelle est la valeur de l'expression `(foo 12 empty)` si l'on définit la fonction `foo` comme ci-dessous ?

```
(define (foo x L)
  (cond ((= x 1) L)
        ((= (modulo x 2) 0) (cons x (foo (/ x 2) L)))
        (else (foo (+ (* 3 x) 1) (cons x L)))))
```

2 Programmation et Nombres Premiers [7 pts]

a) On considère la fonction suivante en Python, qui retourne le plus petit diviseur ≥ 2 de l'entier `n` ≥ 2. Expliquez pourquoi le résultat de la fonction `ppdiv` est nécessairement un nombre premier.

`ppdiv(15) == 3`   et   `ppdiv(13) == 13`

```
def ppdiv(n) :                                    # on suppose n entier ≥ 2
    k = 2
    while n % k!= 0 :
        k = k + 1
    return k
```

b) Traduisez cette définition Python en une fonction Scheme `(ppdiv n)`, dont le processus de calcul soit *itératif* – mais sans instruction d'affectation bien entendu.

c) On suppose que `(ppdiv n)` a été programmée en b). Utilisez-la pour programmer une fonction `(facteurs-premiers n)` retournant la liste croissante des diviseurs premiers *distincts* de `n`, en complétant le code itératif ci-dessous, qui extrait les facteurs premiers un par un pour les placer dans un accumulateur `acc`. N'écrivez dans la feuille de réponses que les expressions <1> à <6>.

`(facteurs-premiers 4840)` $\leadsto$ `(2 5 11)`   ; $4840 = 2^3 \times 5 \times 11^2$

```
(define (facteurs-premiers n)     ; on suppose n >= 2
  (local [(define (iter n acc)    ; acc accumule les facteurs premiers solutions
            (if <1>               ; fini ?
                <2>               ; le résultat
                (local [(define d (ppdiv n))]   ; plus petit facteur premier de n
                  (iter <3> <4>)))))]           ; et on boucle...
    (iter <5> <6>)))
```

d) Programmez la fonction `(exposant p n)` retournant le nombre de fois que l'entier `p` divise l'entier `n` (peu importe que `p` soit premier ou pas). Exemples :

`(exposant 2 40)` $\rightsquigarrow$ 3
`(exposant 3 40)` $\rightsquigarrow$ 0

e) Tout nombre entier $n \geq 1$ se décompose de manière unique en un produit $n = p_1^{\alpha_1} \times \ldots \times p_k^{\alpha_k}$ de puissances de nombres premiers où chaque α_i est > 0. En utilisant les fonctions `facteurs-premiers` et `exposant`, programmez – si possible en une ligne – la fonction `(factorise n)` retournant la liste `((`$\texttt{p}_1$ α_1`)...(`$\texttt{p}_k$ α_k`))`. Exemple :

`(factorise 4840)` $\rightsquigarrow$ `((2 3) (5 1) (11 2))`   ; $4840 = 2^3 \times 5^1 \times 11^2$

3 Retour sur le Tri par Insertion [7 pts]

a) Programmez par récurrence une fonction `(insertion x LD)` prenant une liste d'entiers `LD` triée en ordre *décroissant* et retournant une liste toujours en ordre *décroissant* obtenue en insérant l'entier `x` à sa juste place dans `LD`.

```
(check-expect (insertion 10 '(22 19 15 8 6 1)) '(22 19 15 10 8 6 1))
```

b) Utilisez la fonction `insertion` pour programmer par récurrence la fonction `(tri-ins L)` retournant une copie triée en ordre *décroissant* d'une liste d'entiers `L`.

```
(check-expect (tri-ins '(15 1 8 22 6 19)) '(22 19 15 8 6 1))
```

c) Reprogrammez la fonction `tri-ins` en une fonction `tri-ins-it` qui soit *itérative*.

d) Les deux fonctions `tri-ins` et `tri-ins-it` ont-elles la même *complexité* (dans le pire des cas), si l'on mesure le nombre d'appels à la primitive `cons` ?

e) Si vous avez répondu *OUI* à la question précédente, précisez (sans la justifier) cette complexité si `n` désigne la longueur de `L`. Si vous avez répondu *NON*, expliquez pourquoi.

f) Reprogrammez la fonction `tri-ins` en une fonction `tri-ins-os` utilisant un schéma d'ordre supérieur en une ligne.

4 Question de cours audio [1 pt]

Un musicien souhaite enregistrer numériquement un morceau de musique dont la fréquence monte jusqu'à 15000 Hz. A quelle fréquence d'horloge minimale doit-il régler son convertisseur analogique-digital (ADC) chargé de prélever les échantillons ?

5 Animation d'un Système de Particules [7 pts]

a) Le `FOND` est constitué d'un rectangle 600×300 de couleur bleu clair *(lightblue)* figurant le ciel, posé sur un rectangle 600×300 de couleur sable *(wheat)* figurant la plage. En plus un soleil jaune de rayon 20 est déjà haut dans le ciel. Définissez le `FOND` de l'animation.

b) Un ballon sera modélisé par une **structure** (x, y, r, c, v) comportant sa position x, y, son rayon `r`, sa couleur `c` et sa vitesse d'ascension `v`. Définissez le type de structure `ballon`.

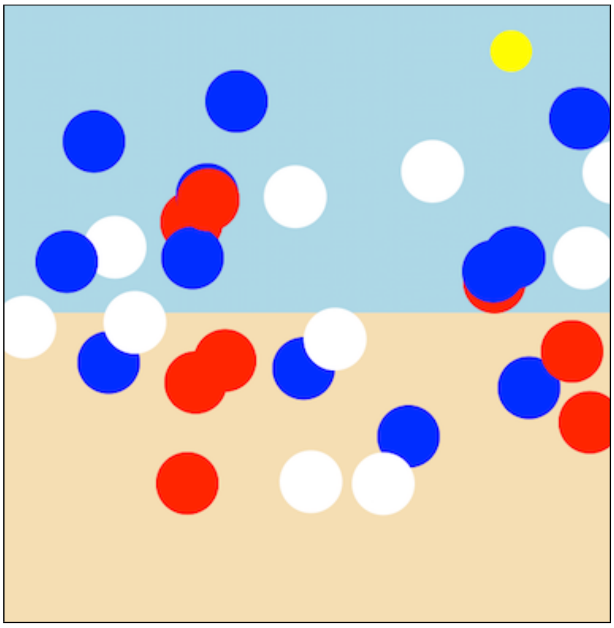
c) Définissez la fonction `(random-ballon)` retournant un **ballon aléatoire** : posé au hasard sur la plage, de rayon 4, de couleur aléatoire choisie parmi *blue*, *white* et *red*, et de vitesse ascensionnelle *inexacte* choisie au hasard entre `#i3` et `#i8`.

d) Le **monde** sera une liste de ballons. Définissez le monde initial `INIT` qui sera une liste de 30 ballons aléatoires.

e) Programmez la fonction `(dessiner L)` retournant la scène courante correspondant au monde `L`.

f) Au début de l'animation, les ballons commencent à gonfler tous en même temps, les rayons augmentant de `0.1` pixel par top d'horloge. Dès qu'un ballon voit son rayon dépasser 30 pixels, il s'élève à la verticale avec sa vitesse ascensionnelle (constante). Programmez la fonction `(suivant L)`.

g) L'animation termine lorsque plus aucun ballon n'est visible dans le ciel. Programmez la fonction `(final? L)`.



```
(big-bang INIT
          (on-tick suivant)
          (on-draw dessiner)
          (stop-when final?)
          (name "Exam Mai 2017"))
```