Université Bordeaux 1 Licence S&TS Mention Informatique S5 2008/2009

UE INF353 **Programmation 3**

Programmation Fonctionnelle et Symbolique
Devoir surveillé No 2

Tous documents autorisés. Mercredi 26 Novembre 2008
Durée : 1h20.
Le barème est donné à titre **indicatif**.
Chacun des trois exercices doit être **rédigé** sur une feuille **séparée** sur laquelle doivent être inscrits votre nom et votre numéro de groupe. Vous devez rendre exactement **une feuille** (ou copie), éventuellement blanche, **par exercice**.

Exercice 1 *(5pts)*

Soit la fonction `decomposition (entier base)` suivante :

```
(defun decomposition (entier base)
  (loop
     for quotient = (floor entier base) then (floor quotient base)
     and chiffres = (list (mod entier base)) then (cons (mod quotient base) chiffres)
     until (zerop quotient)
     finally (return chiffres)))
```

1. Que retourne l'appel `(decomposition 13 2)` ?
2. Que retourne l'appel `(decomposition 11 3)` ?
3. Expliquer en une phrase ce que fait la fonction `decomposition`.
4. Réécrire la fonction `decomposition` en utilisant une boucle de type `do`.

On rappelle que l'appel `(floor n p)` retourne le quotient de la division entière de `n` par `p`.

Exercice 2 *(5pts)*

Écrire une fonction impérative (sans récursion) `my-pi (epsilon)` qui calcule la valeur approchée de π à partir du développement en série de $\pi/4$:

$$\pi/4 = 1 - \frac{1}{3} + \frac{1}{5} - \ldots = \sum_{n=0}^{\infty} (-1)^n \frac{1}{2n+1}$$

On pourra arrêter la somme quand le terme courant devient strictement inférieur à `epsilon/10`.

Exercice 3 *(10pts)*

Pour cet exercice le style de programmation n'est pas imposé. On pourra écrire dans un style fontionnel, impératif ou une combinaison des deux.
On considère l'ensemble des applications de $[0, n[$ dans $\mathbb{N}$.
Chaque application a est représentée par un tableau d'entiers `a` de taille n tel que

$$\forall i \in [0, n[,\ \texttt{a}[i] = a(i).$$

Une telle application est *injective* si

$$\forall i \in [0, n[, \forall j \in [0, n[,\ a(i) = a(j) \Rightarrow i = j.$$

1. Écrire un prédicat `injectivep (a)` qui retourne vrai si a est injective, faux sinon.

```
CL-USER> (injectivep #(1 0 4 3 5 2))
T
CL-USER> (injectivep #(1 0 4 2 5 2))
NIL
```

Une application est *interne* si $\forall i \in [0, n[,\ a(i) \in [0, n[$.

2. Écrire un prédicat `internep (a)` qui retourne vrai si a est interne, faux sinon.

```
CL-USER> (internep #(1 0 4 2 5 2))
T
CL-USER> (internep #(1 0 4 6 5 2))
NIL
```

Une application est une permutation si elle est à la fois interne et injective.

3. Écrire un prédicat `permutationp (a)` qui retourne vrai si a est une permutation.

```
CL-USER> (permutationp #(1 0 4 2 5 2))
NIL
CL-USER> (permutationp #(1 0 4 3 5 2))
T
```

Une *inversion* dans une permutation a est un couple $(i, j) \in [0, n[\times[0, n[$ tel que $i < j$ et $a(i) > a(j)$.
Étant donnée une permutation a, on considère l'application

$$\begin{array}{rccl} inv_a : & [0, n[& \to & \mathbb{N} \\ & i & \mapsto & Card\big\{j \in [0, n[\mid (i, j) \text{ est une inversion}\big\} \end{array}$$

On représente l'application inv_a par un tableau de taille n.

4. Écrire une fonction `inv-a (a)` qui étant donnée une permutation a retourne le tableau représentant l'application inv_a.

```
CL-USER> (inv-a #(1 2 6 4 0 8 7 3 5))
#(1 1 4 2 0 3 2 0 0)
CL-USER> (inv-a #(0 1 2 3 4 5))
#(0 0 0 0 0 0)
CL-USER> (inv-a #(5 4 3 2 1 0))
#(5 4 3 2 1 0)
```

FIN