ENSEIRB Filiere informatique premiere annee 2006/2007

# Examen de programmation fonctionnelle

Durée 2h, documents autorisés, livres interdits

1. Soit la session suivante. Indiquer les réponses de l'interprète aux trois dernières expressions à évaluer. Expliquer.

```
(defparameter i 0)

(defun f(x)
  (let ((i 2))
    (g x)))

(defun g(a)
  (+ a i))

(defun h(i)
  (g i))

(h 10)
(f 1)
(g 0)
```

2. Soit la session suivante.

```
(defparameter a 10)

(defun f(a)
  (lambda(x y)
    (+ a y x)))

(defparameter g (f 0))
```

   - Quel est le type et la valeur du symbole g?
   - On considère les applications suivantes :

```
(funcall g 1 2)

(g 1 2)

(apply g 1 2)
```

   - Indiquer quelles sont parmi ces applications celles qui sont erronées et pourquoi. Donner le résultat de celle qui est correcte.

- Indiquer quelle valeur de a est utilisée pour le calcul de l'application et grâce à quel mécanisme elle est retrouvée.

3. Écrire une fonction `arg-max` admettant en argument une liste de nombres et renvoyant une paire pointée dont le car est le maximum de la liste et le cdr son indice. Proposer une écriture récursive terminale et justifier cette propriété.

4. Expliquer ce que signifie en lisp la notion de forme spéciale et pour quelle raison elle est nécessaire dans la construction du langage.

5. On souhaite écrire une fonction `every` admettant une fonction `f` et une liste `L` en arguments et calculant la conjonction des résultats des applications de la fonction `f` aux éléments de la liste `L`. Par exemple

```
* (every #'evenp '(2 4 6 8))
T
* (every #'evenp '(1 2 3 4 5))
NIL
```

   - Expliquer pourquoi la définiton suivante ne convient pas :

```
(defun every(f L)
  (apply #'and (mapcar f L)))
```

   - Proposer deux écritures différentes, l'une utilisant la forme `map` appelée `every1` et l'autre basée sur une écriture récursive traditionnelle appelée `every2`.
   - Comparer les fonctions `every1`, `every2` et la forme `and` :
     (a) Comment les arguments sont-ils évalués dans les 3 cas? Donner un exemple explicatif pour chaque cas.
     (b) Le calcul de la conjonction s'effectue-t-il jusqu'au bout de la liste des arguments dans les 3 cas? Donner un exemple explicatif pour chaque cas.
   - Écrire une macro effectuant le même calcul et expliquer quelle est la différence de fonctionnement avec les fonctions.

6. On donne la fonction suivante :

```
(defun append-map (f L)
  (apply #'append (mapcar f L)))
```

   - Expliquer le fonctionnement général de cette fonction;
   - Donner le résultat de l'application suivante :

```
(append-map #'cdr '((1 2 3 4) (2 3 4 5) (3 4 5 6) (4 5 6 7)))
```

   - Écrire la fonction `append-mapn` généralisant la précédente en admettant en arguments une fonction d'arité n, $n > 0$ et $n$ listes.

```
(append-mapn #'list '((1 2 3 4) (2 3 4 5)) '((3 4 5 6) (4 5 6 7)))
```

   - Donner le résultat de l'application de l'exemple précédent de `append-mapn`.

7. Étant données les listes $(x_0\ x_1\ \ldots\ x_n)$ et $(y_0\ y_1\ \ldots\ y_n)$, et une fonction binaire $f$, écrire une fonction `tableau` qui construit la liste $((f\ x_0\ y_0)\ \ldots\ (f\ x_0\ y_n)\ (f\ x_1\ y_0)\ \ldots\ (f\ x_1\ y_n)\ \ldots\ (f\ x_m\ y_0)\ \ldots\ (f\ x_m\ y_n))$