Programmation IV Examen de test 27 mai 2003

Exercice 1 [10 points]

Définissez une classe `Stack` polymorphe modélisant une pile de valeurs d'un type donné. Cette classe doit posséder deux méthodes, la première nommée `push` et qui permet de placer une nouvelle valeur au sommet de la pile, et la seconde nommée `pop`, qui supprime et retourne la valeur au sommet de la pile. Une erreur doit être signalée si la pile est vide lors d'un appel à `pop`.

Exercice 2 [25 points]

On modélise les listes d'entiers au moyen des classes suivantes :

```
abstract class IntList with {
  abstract def append(that: IntList): IntList;
  abstract def length: Int;
}
case class IntCons(head: Int, tail: IntList) extends IntList with {
  def append(that: IntList) = IntCons(head, tail.append(that));
  def length = 1 + tail.length;
}
case class IntNil extends IntList with {
  def append(that: IntList) = that;
  def length = 0;
}
```

Montrez par induction sur `l1` qu'on a la propriété suivante, où `l1` et `l2` sont des listes d'entiers quelconques.

```
(l1 append l2).length = l1.length + l2.length
```

(Suite de l'autre côté)

Exercice 3 [20 points]

Dans un programme manipulant les expressions, on a les définitions suivantes :

```
trait Expr;
case class Const(value: double) extends Expr;
case class Plus(left: Expr, right: Expr) extends Expr;
case class Times(left: Expr, right: Expr) extends Expr;
case class Power(expr: Expr, power: int) extends Expr;
```

Écrivez une fonction qui prend en argument une expression et qui retourne une expression équivalente mais dont tous les nœuds `Power` ont été remplacés par des chaînes de nœuds `Times`. On suppose que l'exposant des nœuds `Power` est toujours strictement positif.

Par exemple, l'expression $(1+2)^3$ devra être remplacée par $(1+2)*(1+2)*(1+2)$.

Exercice 4 [25 points]

Écrivez une fonction qui, étant donnée une liste, retourne l'ensemble de toutes ses permutations, sous forme de liste de listes. Cette fonction a le profil suivant :

```
def permute[a](l: List[a]): List[List[a]];
```

Par exemple, appliquée à la liste suivante :

```
List(1,2,3)
```

cette fonction retourne (une permutation quelconque) de la liste ci-dessous :

```
List(List(1,2,3), List(1,3,2), List(2,1,3),
     List(2,3,1), List(3,1,2), List(3,2,1))
```

Exercice 5 [20 points]

Écrivez une fonction `separate` qui prend en argument un flot infini et qui retourne une paire de flots : d'une part le flot constitué des éléments du flot original dont la position est paire, et d'autre part le flot des éléments du flot original dont la position est impaire.

Écrivez aussi une fonction `merge` qui effectue l'opération inverse.

Ces deux fonctions doivent êtres polymorphes, c'est-à-dire qu'elles doivent fonctionner quel que soit le type des éléments des flots.