RICM1 - 2005/06

Langage et Programmation 2

TP1

Préliminaires

Sur le site internet des TPs, (`http://www-verimag.imag.fr/~peron/`, puis suivre LP2), récuperez l'archive du tp1.

Configuration

Nous allons utiliser EMACS comme environnement de travail, couplé au mode *tuareg* qui permet une mise en valeur et une indentation automatique du code *ocaml*, et donne accès aux fonctions d'interprétation et de compilation directe du code dans EMACS.

1. Extraire les fichiers de l'archive sur votre compte.
2. Installer le mode *tuareg*, selon les instructions contenues dans le fichier `mode.tuareg`.

Exercice 1

Pour la dernière question se reporter au *précis de compilation caml.*

1. Écrire une fonction `int_somme` qui rend la somme des éléments d'une liste d'entiers.
2. Exécuter et tester cette fonction à l'aide de l'interpréteur *ocaml.*
3. Utiliser les fonctions du fichier `read.list.ml` pour lire une liste d'entiers sur l'entrée standard, puis en calculer la somme et enfin en afficher le résultat.
4. Exécuter et tester cette fonction à l'aide du compilateur `ocamlc`.

Algorithmes élémentaires de tri

On désigne par « tri » l'opération consistant à ordonner (on choisira en ordre croissant) un ensemble d'éléments en fonction de clés sur lesquelles est définie une relation d'ordre. Voici deux principes de tri différents.

- le *tri par insertion* qui pour chaque élément de la liste de départ l'insère en bonne place dans la liste des éléments déja triés.
- le *tri minimum* qui recherche le plus petit élément de la liste pour le mettre en premier, puis considère le reste de la liste, recherche son plus petit élément pour le mettre en second, etc.

Exercice 2

Implantez en *caml* ces deux algorithmes de tri sur des listes associatives.

1. Donnez la complexité de chaque algorithme implanté.
2. Réfléchissez à ce qu'est le *tri maximum.* Implanter le et discuter les différences avec le *tri minimum.*

Chaînes de caractères

Exercice 3
Écrire une fonction qui permet de compter le nombre d'occurrences du caractère `'a'` dans une chaîne de caractères.

Exercice 4
Une phrase peut cacher, dans l'ordre, les lettres d'un mot. Par exemple, la chaîne : « Examens de programmation fonctionnelle à Grenoble » contient le mot « espoir ».

1. Écrire une fonction `mot_cache` qui teste si un mot donné est caché dans une chaîne de caractères.
2. Écrire une fonction `nb_mot_cache` qui compte le nombre d'occurrences d'un mot caché dans une chaîne de caractères.

Arbres binaires

Exercice 5
Écrire une fonction `liste_feuilles` qui rend la liste des feuilles d'un arbre.

Un arbre binaire de recherche est un arbre binaire dont les noeuds vérifient la propriété suivante :
- tout nœud du sous-arbre droit d'un nœud a une valeur supérieure à la valeur de ce nœud.
- tout nœud du sous-arbre gauche d'un nœud a une valeur inférieure à la valeur de ce nœud.

Exercice 6
Écrire la fonction `recherche` qui vérifie si un élément donné appartient un arbre binaire de recherche donné.