Programmation[C] : TP 17, tri et maximier

On reprend les questions du TP 16 mais en utilisant cette fois-ci une structure de donnée **modifiée en place** : lors des manipulations, on ne fabrique pas de nouveaux arbres mais on modifie les nœuds existants.

Codage en C

1. Définir le type `maximier` des arbre maximiers d'entiers. Le maximier vide est codé par le pointeur nul. Rappel : définir un type `struct m` puis un alias

   ```
   typedef struct m * maximier;
   ```

   Écrire la fonction d'allocation `maximier noeud(int, maximier, maximier)` associée.

2. Écrire la fonction

   ```
   void insertion(int comp(int, int), int element, maximier* m);
   ```

   qui insère l'`element` à sa place (pour la fonction de comparaison `comp`) dans le maximier `m` **en le modifiant en place**. Remarques :
   – la fonction manipule un pointeur vers un maximier ;
   – seule l'insertion dans un maximier vide nécessite l'allocation d'un nouveau nœud.

3. Écrire la fonction

   ```
   void ote_sommet(int comp(int, int), maximier* m);
   ```

   qui supprime le sommet d'un maximier. Remarque : libérer (avec la fonction `free()`) tout nœud devenu inutile (dans le cas où l'un des deux sous-arbres est vide).

4. Reprendre une structure de liste classique (par exemple `chaine.c` et `chaine.h` dans `~serveur/PROC/C`) et écrire une fonction de tri en utilisant un maximier. Remarque :
   – on peut trier **en place** en replaçant les éléments dans la liste originale ;
   – on peut tester en triant les arguments de la commande :

   ```
   int main(int argc, char **argv) {
     int i;
     liste l = (liste)NULL;
     for(i = 1; i < argc; i++)
       l = cons(atoi(argv[i]), l);
     tri(l);
     print_liste(l);
     return 0;
   }
   ```