Multi-ensembles (4 séances, à rendre)

Un ensemble est une structure de données sans ordre et sans répétition. Un multi-ensemble (appelé multi-set ou bag en anglais) est une structure de données sans ordre mais avec répétition possible, qui répond par exemple à la spécification (partielle) suivante. Les paramètres soulignés correspondent au paramètre *donnée modifiée* pour un implémentation en modification (demandée ici).

```
Constructeurs : (ME pour Multi-Ensemble, T est le type des éléments)
      vide : → ME
      ajoute : ME x T → ME       // ajoute une occurrence de plus

Deconstructeurs :
      oteUn : ME x T → ME    // enlève une occurrence de l'élément
      oteTous : ME x T → ME  // enlève toutes les occurrences

Reconnaisseur :
      estVide : ME  → booléen

Utilitaires :
      nbOcc : ME  x T → entier         // nombre d'occurrences d'un élément
      ajoute : ME x ME → ME            // union de deux multi-ensembles
      intersecte : ME x ME → ME        // intersection de deux multi-ensembles
      enleve :  ME x ME → ME           // retire les éléments de l'autre ME
      egal : ME x ME → booléen         // égalité de deux multi-ensembles


Axiomes (incomplet)
      oteUn(vide,_)=ε ;                oteUn(ajouter(mul,x),x)=mul
      oteUn(ajouter(mul,y),x)= ajouter(oteUn(mul,x),y) si x≠y
      oteTous(vide,_)=vide ;           oteTous(ajouter(mul,x),x)=oteTous(mul,x)
      oteTous(ajouter(mul,y),x)= ajouter(oteTous(mul,x),y) si x≠y
      estVide(vide)=VRAI ;             estVide(ajouter(_,_))=FAUX
      nbOcc(vide,_)=0 ;                nbOcc(ajouter(mul,x),x)=1+nbOcc(mul,x)
      nbOcc(ajouter(mul,y),x)=nbOcc(mul,x) si x≠y
```

Le travail demandé consiste à implémenter de deux façons une structure de multi-ensemble, par modification sur place (et non par création d'un nouveau multi-ensemble renvoyé en résultat). La première implémentation doit utiliser le chaînage décrit ci-contre (→), la seconde doit utiliser une table de hachage.

```
struct _maille {
  T elt ;
  _maille * suiv ;
}
```

Vous devrez aussi écrire un programme minimaliste utilisant votre structure pour la tester. Ce programme doit pouvoir compiler et fonctionner à l'identique avec l'une ou l'autre des implémentations, sans modification (sauf le #include pour importer la structure). Les deux implémentations doivent donc porter le même nom et avoir les mêmes profils de méthodes.

Un document de travail doit être rédigé **au préalable et en parallèle**, précisant les ambiguïtés du sujet (en particulier en donnant les axiomes manquants de la spécification). Ce document présentera aussi les choix effectués (constantes, comportement des méthodes non précisé dans l'énoncé, …) et les limites d'utilisation. Il est aussi, si ce n'est plus important, que le code.

Il devra comporter un comparatif de vos deux implémentations en terme de complexité temporelle (l'ordre de grandeur du coût de chaque méthode doit être renseigné) et d'encombrement mémoire. Au moins une troisième implémentation (suffisamment différente) doit être présentée (la réalisation n'en est pas demandée) et comparée aux deux autres.

Le travail est à déposer sur MADOC au plus tard le 14 mars 2014 à 23h55 sous forme d'une archive zip contenant le rapport (pdf) et le code.

Le code doit être clair et concis, commenté, avec en particulier les pré- et post-conditions. Le rapport doit être bien écrit (français correct, attention à l'orthographe!) et agréable à lire, il doit comporter introduction et conclusion et des annexes pour les détails techniques (graphiques, code, etc.)

Vous pouvez proposer quelques méthodes supplémentaires si leur coûts sont très différents d'une implémentation à l'autre (mais impérativement les mêmes méthodes dans toutes les implémentations). Vous pouvez les implémenter ou seulement les évoquer dans le rapport, avec leurs coûts.

Vous décrirez (et prouverez les propriétés données) les résultats des expressions :

```
nbOcc(oteUn(A,x))                          nbOcc(oteTous(A,x))

intersecte(enleve(A,B),B)                  ajoute(enleve(A,intersecte(A,B)),B)
```