Travaux Pratiques d'algorithmique n°5

Cours d'Informatique de Deuxième Année

—DEUG Sciences S3-MI—

Manipulation d'ensembles

L'objet de ce TP est la manipulation d'ensembles (collections d'éléments **distincts**). Les éléments de ces ensembles (de type `Produit`) sont représentés par des structures. Il faudra donc implémenter d'abord quelques fonctions de base permettant de manipuler ces éléments, puis quelques opérations sur les ensembles.

---

1 Le type `Produit`

Le type `Produit` utilisé durant ce TP est défini de la façon suivante :

```
struct produit
{
  char          nom[MAXSTR];
  char          fournisseur[MAXSTR];
  double        prix;
};
typedef struct produit Produit;
```

où `MAXSTR` est la taille maximale d'une chaîne de caractères.

▸ **Exercice 1.** *Écrire les fonctions suivantes :*

- `SaisieProduit` *permettant la saisie au clavier d'un élément de type* `Produit`*. L'élément à saisir sera fourni en argument de la fonction ;*
- `AfficheProduit` *qui affiche la valeur d'un élément de type* `Produit` *;*
- `CompareProduit` *qui compare deux éléments de type* `Produit` *fournis en argument. La comparaison se fera selon le nom d'abord puis le fournisseur (le prix n'est pas pris en compte ici). La fonction renverra un nombre négatif, nul ou positif selon que le premier élément est "inférieur", "égal" ou "supérieur" au second ;*
- `CopieProduit` *qui recopie un élément dans un autre. Les deux éléments seront fournis en argument ;*
- `EchangeProduit` *qui échange les valeurs des deux éléments de type* `Produit` *fournis en argument.*

2 Le type `Ensemble`

On définit le type `Ensemble` (de `Produit`) de la façon suivante :

```
struct ensemble
{
   unsigned int taille;
   Produit      tab[MAXENS];
};
typedef struct ensemble Ensemble;
```

où `MAXENS` est le nombre maximum d'éléments que peut contenir un ensemble. De plus, le **tableau est trié** selon l'ordre croissant de ses éléments. C'est ce qu'on appelle une représentation d'ensemble par tableau trié. Dans ce qui suit, les ensembles sont représentés de cette façon et toute opération sur un ensemble doit préserver cette propriété.

▸ **Exercice 2.** *Écrire les fonctions suivantes :*

- `InitEnsemble` *qui initialise un ensemble passé en argument à l'ensemble vide ;*
- `AfficheEnsemble` *qui affiche l'ensemble passé en argument ;*
- `RechercheProduit(???)` *qui recherche un produit dans un ensemble. Cette fonction retournera* 1 *ou* 0 *selon que le produit est présent ou pas dans l'ensemble. Elle calculera également l'indice du tableau où se trouve le produit s'il est présent, et l'indice où il devrait se trouver sinon. Vous donnerez deux versions de cette fonction ; l'une réalisant une recherche "naïve" et l'autre la recherche dichotomique ;*
- `int AjouteProduit(???)` *qui ajoute un produit à un ensemble. Si le produit est déjà présent (même nom et même fournisseur), la fonction mettra simplement à jour son prix (il ne doit pas être insérer une deuxième fois) ;*
- `int SaisieEnsemble(???)` *qui permet de créer un ensemble (à partir de l'ensemble vide et par ajouts successifs de ses éléments) ;*
- `SupprimeProduit` *qui supprime un produit d'un ensemble s'il est présent. Ne fait rien sinon.*

▸ **Exercice 3.** *Écrire une fonction qui réalise l'union de deux ensembles. Cette fonction retournera* 1 *ou* 0 *selon que l'opération s'est bien passée ou pas.*

▸ **Exercice 4.** *Évaluer la complexité des fonctions écrites dans les exercices 2 et 3.*

▸ **Exercice 5.** *Implémenter d'autres opérations de base sur les ensembles (intersection, inclusion, ...) et évaluer leurs complexités.*