Travaux Dirigés sur le langage C n^o.7

Cours d'Informatique de Deuxième Année

—Deug SSM M3—

L'idée de ce TD est la manipulation d'une nouvelle structure : la structure d'arbre binaire de recherche. L'algorithmique que l'on utilise sur ces structures est voisine de celle que l'on utilise sur les listes chaînées à ceci près que l'on ne parle plus exactement du `suivant` d'une cellule mais plutôt du `fils gauche` et `fils droit` d'un noeud. De même, on ne parle plus du `debut` de la liste mais plutôt de la `racine` de l'arbre. Pour chaque algorithme, on s'efforcera de donner les complexités en temps et espace.

Arbres binaires de recherche...

On considère les déclarations suivantes :

```
typedef struct noeud
{ int Val;                    /* valeur dans le noeud               */
  int NbOcc;                  /* nombre d'occurences de cette valeur */
  struct noeud * Gauche;      /* pointeur sur le sous-arbre gauche   */
  struct noeud * Droit;       /* pointeur sur le sous-arbre droit    */
} Noeud;

typedef Noeud * PtrNoeud;     /* pointeur sur un noeud               */
```

sans oublier la propriété que l'on a imposée à ces arbres et qui dit que ce sont des arbres binaires de **recherche**...

▸ **Exercice 1**. Afin de pouvoir manipuler une telle structure, on demande d'écrire les fonctions élémentaires suivantes :

- ◇ `CreerFeuille(val, nbocc)` qui retournera un pointeur sur un noeud contenant `nbocc` occurences de la valeur `val`,
- ◇ `EstFeuille(abr)` qui testera si le noeud pointé par `abr` est une feuille,
- ◇ `EstVide(abr)` qui testera si l'arbre pointé par `abr` est vide.

▸ **Exercice 2**. On considère l'opération d'insertion d'un élément dans un arbre binaire de recherche. Comme pour les listes chaînées triées, cette opération se caractérise par la propriété suivante : si *abr* est un arbre binaire de recherche et x un élément, alors le résultat de l'insertion de x dans l'arbre *abr* est encore un arbre binaire de recherche. On demande tout d'abord de construire à la main les arbres binaires de recherche correspondant aux insertions successives dans l'arbre vide, des valeurs $3, 5, 7, 2, 4, 1$.

▸ **Exercice 3**. Déduire des exemples précédents l'algorithme d'insertion d'un élément `x` dans un arbre binaire de recherche pointé par `abr`. On écrira alors la fonction `InsertionBin()` prenant en paramètre `x` et `abr`, réalisant cet algorithme.

▸ **Exercice 4**. Écrire une fonction `RechercheBin()` prenant en argument un entier `x` et un arbre binaire de recherche pointé par `abr` et retournant le nombre d'occurences de l'élément `x` dans l'arbre pointé par `abr`.

▸ **Exercice 5**. Déduire de ce qui précède, la fonction `TriBin()` de tri par insertion dans un arbre binaire de recherche. Cette fonction prendra en argument un tableau d'entiers `t` à trier et le nombre `n` de ces entiers. Le résultat du tri sera stocké dans ce même tableau `t`.

▸ **Exercice 6**. Écrire une fonction `SuppressionBin()` de suppression d'un élément `x` dans un arbre binaire de recherche pointé par `abr`. La fonction supprimera une seule occurence de `x`. Le résultat de cette suppression sera bien évidemment encore un arbre binaire de recherche.