Travaux Pratiques d'algorithmique n°1

Cours d'Informatique de Deuxième Année

—DEUG Sciences S4-MI—

Arbres

L'objet de cette feuille d'exercices est l'implémentation des opérations de base sur une structure de données récursive : la structure d'*arbre binaire de recherche*. Nous utilisons ici cette structure pour classer des événements en fonction de leur date (toutes les dates seront supposées différentes).

On considère les déclarations suivantes :

```
struct evenement
{
  int Annee;
  char Info[255];
};

typedef struct evenement Evenement;

struct noeud
{
  Evenement Evt;                /* donnees liees au noeud           */
  struct noeud * FilsGauche;    /* pointeur sur le sous-arbre gauche  */
  struct noeud * FilsDroit;     /* pointeur sur le sous-arbre droit   */
};

typedef struct noeud Noeud;     /* nouveau type : Noeud             */

typedef Noeud * ArbreBin;       /* pointeur sur un Noeud            */
```

Un arbre binaire, représenté par la structure `ArbreBin` ci-dessus, est dit *arbre binaire de recherche* (ou encore ABR) si les conditions suivantes sont vérifiées :

- ▷ l'arbre vide est un ABR ;
- ▷ les deux fils de la racine de l'arbre sont des ABR et la valeur à la racine de l'arbre est strictement plus grande (resp. petite) que toutes les valeurs du sous-arbre gauche (resp. droit).

▸ **Exercice 1.** ***(Fonctions de base)*** *Afin de pouvoir manipuler une telle structure de données, nous demandons d'écrire les fonctions élémentaires suivantes :*

1. `Noeud * CreerFeuille(int annee, char info[255])` *qui retournera un pointeur sur un nœud contenant l'événement lié aux données passées en argument ;*
2. `int EstVide(ArbreBin abr)` *qui testera si l'arbre pointé par* `abr` *est vide ;*
3. `int EstFeuille(Noeud * n)` *qui testera si le nœud pointé par* `n` *est une feuille ;*
4. `int DetruireNoeud(Noeud * n)` *qui libère l'espace mémoire associée au nœud* `n`*. Renvoie* `1` *si la libération s'est effectuée sans problème,* `0` *sinon ;*
5. `void AfficheEvt(Evenement evt)` *qui affiche le contenu d'un événement.*

▸ **Exercice 2.** ***(Insertion)*** *On considère l'opération d'insertion d'un événement dans un arbre binaire de recherche. Si* `abr` *est un arbre binaire de recherche et* `evt` *un événement, alors le résultat de l'insertion de* `evt` *dans l'arbre* `abr` *est encore un arbre binaire de recherche.*
Les comparaisons entre événements seront effectuées par rapport aux valeurs du champ `Annee` *de la structure* `evenement`*.*

1. *Implémenter la fonction* `InsertionBin` *prenant en paramètre un événement* `evt` *et un arbre binaire de recherche* `abr` *réalisant l'algorithme d'insertion (les doublons ne sont pas autorisés) ;*
2. *évaluer la complexité de cette fonction.*

▸ **Exercice 3.** ***(Parcours)*** *Donner les fonctions de parcours permettant d'afficher tous les événements d'un arbre binaire de recherche, par ordre chronologique et par ordre chronologique inverse. Ensuite, écrire une fonction qui affiche par ordre chronologique les événements qui se sont déroulés après une année donnée.*

▸ **Exercice 4.** ***(Destruction d'un arbre)*** *Donner une fonction* `SuppressionArbre` *qui libère tout l'espace mémoire associé à l'arbre binaire de recherche passé en argument.*

▸ **Exercice 5.** ***(Suppression d'un nœud)*** *Donner une fonction de nom* `SuppressionBin` *de suppression d'un événement ayant eu lieu à la date* `annee` *dans un arbre binaire de recherche pointé par* `abr`*. Le résultat de cette suppression sera bien évidemment encore un arbre binaire de recherche.*