Feuille de TP n°2

Le but de ce TP est d'implémenter des fonctions de manipulation des arbres binaires de recherche. On représentera les arbres de façon récursive : un arbre est donné par sa racine (munie d'une valeur), son sous-arbre gauche et son sous-arbre droit. Pour implémenter ceci en C, le mieux est d'utiliser des *pointeurs* : au lieu d'inclure la donnée complète des sous-arbres, on se contente de donner les adresses mémoire auxquelles ces sous-arbres se trouvent. Ceci nous amène à distinguer deux types d'objets : les arbres, qui sont des pointeurs sur un nœud, et les nœuds, qui sont des triplets (`valeur`,`gauche`,`droite`) où `valeur` est un entier (par exemple) et où `gauche` et `droite` sont deux arbres. On appellera `noeud` le type de donnée des nœuds ; celui des arbres est alors noté `noeud*` conformément à la syntaxe des pointeurs en C. On définit le type `noeud` par les lignes de code suivantes :

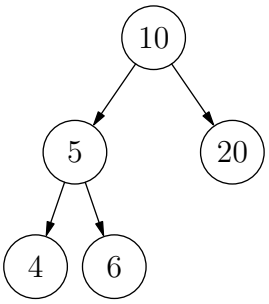
```
typedef struct noeud {
  int valeur;
  struct noeud* gauche;
  struct noeud* droit;
} noeud;
```

Si `arbre` désigne un arbre, on accède à la valeur de sa racine (respectivement son sous-arbre gauche, son sous-arbre droit) à l'aide de la syntaxe `arbre->valeur` (respectivement `arbre->gauche`, `arbre->droit`). L'arbre vide sera représenté par `NULL`.

Téléchargez le fichier squelette disponible à l'adresse `http://perso.univ-rennes1.fr/sandrine.caruso/ALBA/arbres-squelette.c`. Outre la définition du type `noeud`, il contient

- une fonction de création d'un arbre `nouvel_arbre` : elle prend en argument un entier et renvoie l'arbre ayant un unique nœud, étiqueté par cet entier,
- des fonctions `__affiche` et `affiche` qui servent à afficher un arbre (il n'est pas nécessaire d'en comprendre le code),
- des fonctions `insere`, `recherche` et `supprime` à programmer,
- une fonction `main` dans laquelle est programmée une interface pour utiliser les fonctions précédentes.

Exercice 1. Commentez (à l'aide des balises `/*` et `*/`) le contenu de la fonction `main` (excepté le `return 0;`) et utilisez la structure `noeud` et la fonction `nouvel_arbre` pour construire l'arbre suivant :



Affichez-le à l'aide de la fonction `affiche`.

Une fois que cela fonctionne, vous pouvez commenter les lignes que vous venez de taper et décommenter le contenu d'origine.

Exercice 2. Programmer une fonction `insere`, qui, étant donnés un entier et un arbre binaire de recherche (de type `noeud*`), insère l'entier dans l'arbre selon la procédure des arbres binaires de recherche. On pourra programmer, au choix, de manière récursive ou itérative.

Exercice 3. Programmer une fonction `recherche`, qui, étant donnés un entier et un arbre binaire de recherche, renvoie, si l'entier apparait dans l'arbre, un sous-arbre dont la racine est étiquetée par cet entier, et sinon, l'arbre vide `NULL`.

Exercice 4. Programmer une fonction `supprime`, qui, étant donnés un entier et un arbre binaire de recherche, supprime la valeur donnée en argument selon la procédure des arbres binaires de recherche. Si la valeur ne se trouve pas dans l'arbre, la fonction renvoie simplement l'arbre d'origine.

Utilisez l'interface programmée dans `main` pour tester ces fonctions, construire des arbres binaires de recherche, les afficher, rechercher une valeur dedans, supprimer des éléments, etc. Comment construire, par exemple, l'arbre de l'exercice 1 en utilisant la fonction `insere` ?