Arbres binaires

Dans ce TP, on implémente nos premiers arbres binaires, et on teste les fonctions vues en TD.

Dans la suite, on utilisera, pour implémenter les arbres binaires, le type suivant :

```c
typedef struct noeud{
  int valeur;        /* étiquette du nœud */
  struct noeud *fg; /* adresse du fils gauche */
  struct noeud *fd; /* adresse du fils droit */
} Noeud, *Arbre;
```

0. **Sortir un papier et un crayon** — Si vous ne réussissez pas cet exercice, votre enseignant ne viendra pas vous aider en cas de problème lors d'une des questions qui suivent.
1. **Création d'arbres à la main**
   a. Écrire une fonction `Arbre alloue_noeud(int val)` qui renvoie l'adresse d'un `Noeud` dont l'étiquette est `val`, et dont les champs `fg` et `fd` ont été initialisés avec la valeur `NULL`.
   b. Utiliser directement cette fonction pour construire un arbre de quelques nœuds, par exemple en recopiant les deux exemples ci-dessous :

```c
Arbre a = alloue_noeud(1);
a->fg = alloue_noeud(2);
a->fg->fd = alloue_noeud(42);
```

   ou encore

```c
Arbre a = alloue_noeud(5);
a->fg = alloue_noeud(1);
a->fg->fg = alloue_noeud(11);
a->fg->fd = alloue_noeud(11);
a->fd = alloue_noeud(21);
a->fd->fd = alloue_noeud(42);
a->fd->fd->fd = alloue_noeud(15);
```

2. **Fonctions de mesures d'arbre** — Écrire en C les fonctions vues en TD, et permettant de calculer les caractéristiques suivantes d'un arbre donné en argument :
   a. hauteur ;
   b. nombre de nœuds ;
   c. nombre de feuilles ;
   d. nombre de nœuds internes ;
   e. nombre de nœuds internes à exactement deux fils.
3. **Constuction d'arbres étiquetés par des entiers naturels non nuls** — Chacun des fichiers `arbre1` à `arbre8` contient une suite décrivant un arbre obtenu par parcours en profondeur dans l'ordre préfixe : tout entier `val` strictement positif représente un nœud d'étiquette `val`, et tout entier `0` représente un arbre vide. Deux entiers consécutifs sont séparés par un espace.
   a. Reconstruire ces arbres et, pour chacun d'eux, indiquer les mesures obtenues en utilisant les fonctions codées en question 2.
   b. Écrire et tester sur ces arbres une fonction permettant de déterminer si un arbre est strictement binaire, c'est-à-dire si chacun de ses nœuds a 0 ou bien 2 fils.

4. **Utilisation d'un nouvel encodage pour des arbres à étiquettes quelconques** — On considère maintenant des arbres binaires quelconques, etiquetés par des entiers relatifs. Toute valeur entière peut donc apparaître comme étiquette d'un nœud de l'arbre. Par ailleurs, les nœuds de l'arbre peuvent n'avoir aucun fils, avoir un fils droit mais pas de fils gauche, un fils gauche mais pas de fils droit, ou bien avec un fils gauche et un fils droit.

   Pour décrire un tel arbre, on effectue un parcours en profondeur de l'arbre en indiquant, pour chaque nœud rencontré, s'il dispose de fils gauche et/ou droit, puis en indiquant son étiquette. Pour représenter les relations de filiation, on utilisera :

   — l'entier `0` pour une feuille de l'arbre ;
   — l'entier `1` pour un nœud avec un fils droit mais pas de fils gauche ;
   — l'entier `2` pour un nœud avec un fils gauche mais pas de fils droit ;
   — l'entier `3` pour un nœud avec deux fils.

   Ainsi, l'arbre dessiné en Figure 1 est codé par la suite `3 5 2 0 0 1 1 -7 3 9 0 8 0 11`.

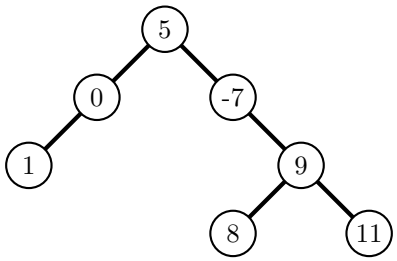

FIGURE 1 – Arbre binaire

   a. Écrire une fonction `Arbre construit_arbre_quelconque(FILE *in)` qui construit un arbre à partir d'une telle suite lue dans le fichier manipulé grâce à `in` et renvoie l'arbre en question.
   b. Écrire une fonction `void ecrit_arbre_quelconque(Arbre a, FILE *out)` qui écrit la suite décrivant l'arbre `a` dans le fichier manipulé grâce à `out`.
   c. Utiliser ces fonctions pour encoder, selon ce nouveau format, les arbres obtenus en question 3. On appellera `arbreX-new` le fichier obtenu à partir du fichier `arbreX`, où `X` décrit tous les entiers entre 1 et 8.