TP1 : Arbres binaires de recherche

Le but de ce TP est d'implanter les algorithmes de recherche et de modification dans les ABR vus en cours.

Consignes
Télécharger le contenu du dossier `TP1` sur moodle dans un nouveau dossier de votre espace personnel. Vous trouverez dans ce dossier les fichiers dont vous avez besoin.
Il y a quatre (couples de) fichiers :

`ArbreBinaire.cpp/.h` contient l'implantation d'un arbre binaire (classe `ArbreBinaire`) ;

`ABR.cpp/.h` contient l'implantation d'un arbre binaire de recherche (classe `ABR`) ;

`tests.cpp` est un fichier de tests, qui contient le `main` ;

`Makefile` s'occupe de la compilation : `make` compilera ce qu'il faut !

La commande `make` *produit l'exécutable* `tests` *que vous devez exécuter pour tester vos fonctions.*

Remarques
— Le seul fichier à modifier, et à rendre, est `ABR.cpp`. **Ne pas modifier les autres fichiers.**
— Il est **impératif** de tester chaque fonction que vous écrivez, à l'aide du fichier de tests fourni. *Ne vous contentez pas forcément des tests suggérés !*

Exercice d'échauffement
Cet exercice doit vous permettre de comprendre le fonctionnement de la classe `ArbreBinaire`. Vous n'avez besoin pour cela que de consulter le fichier `ArbreBinaire.h`. Soyez sûrs de savoir répondre aux questions avant de passer à la programmation.

1. Quels sont les champs de la classe `ArbreBinaire` ?
2. Comment crée-t-on un arbre binaire ?
3. Quels sont les champs de la structure `noeud` ?
4. Si `arbre` est un pointeur (type `ArbreBinaire*`) vers l'arbre de la figure 1, comment obtient-on le pointeur vers le nœud de valeur 17 ?

Exercice à rendre
Pour tout l'exercice, on peut tester les fonctions en ***vérifiant le résultat graphiquement sur le fichier*** `arbre.svg`.

1. Compléter la méthode `noeud* ABR::inserer(int k)` qui insère la valeur `k` dans l'arbre et renvoie un pointeur vers le nouveau nœud.
   Exemples de test : insérer des nœuds dans un arbre initialement vide ; par exemple, insérer des nœuds de valeurs 1, 2, …, 15 *(dans cet ordre) ; les insérer dans l'ordre inverse ; les insérer dans l'ordre* 8, 7, …, 1, *puis* 9, 10, …, 15 *; trouver un ordre d'insertion qui construit un arbre de hauteur 3. En choisissant l'arbre test (option 1), le résultat attendu est l'arbre de la figure 1.*
2. Compléter les méthodes `void ABR::parcoursInfixe(noeud* x)` qui affiche un parcours infixe de l'ABR et `noeud* ABR::minimum(noeud* x)` qui renvoie le nœud de valeur minimale dans cet arbre.
3. Compléter la méthode `noeud* ABR::rechercher(int k)` qui recherche un nœud de valeur `k` dans l'arbre. Si un tel nœud existe la fonction renvoie un pointeur sur ce nœud, sinon le pointeur `NULL`.
4. Compléter la méthode `noeud* ABR::successeur(noeud *x)` qui renvoie le nœud successeur du nœud `x` passé en paramètre. Si `x` est le nœud de valeur maximale dans l'arbre, on renvoie le pointeur `NULL`.

5. Compléter la méthode `void ABR::supprimer(noeud* z)` qui supprime le nœud `z`. Penser à libérer la mémoire correspondante lors de la suppression d'un nœud !

   Utiliser la méthode `remplacer` *des* `ArbreBinaires`*, qui implante la fonction vue en cours. En supprimant successivement les nœuds de valeur 23, 16, 6 et 13 de l'arbre test, le résultat doit être celui de la figure 2.*

6. Compléter les méthodes `void ABR::rotationGauche(noeud* z)` qui effectue une rotation gauche sur le nœud `z` et `void ABR::rotationDroite(noeud* z)` qui effectue une rotation droite sur le nœud `z`.

   Après une rotation gauche sur le nœud 6 de l'arbre test, le résultat doit être celui de la figure 3.

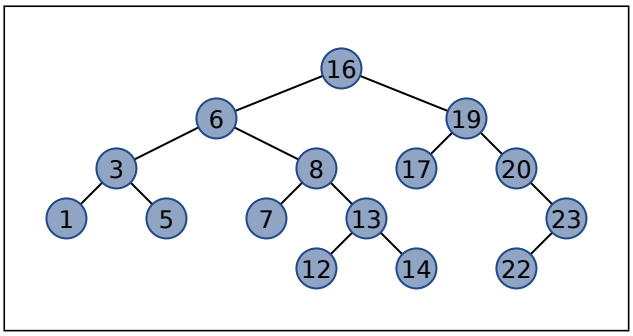

FIGURE 1 – L'arbre test fixé de `test.cpp`.

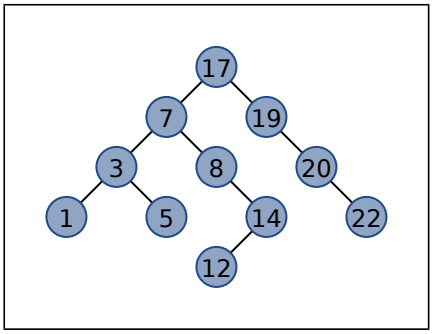

FIGURE 2 – L'arbre de la figure 1 après suppression des nœuds de valeur 23, 16, 6 et 13.

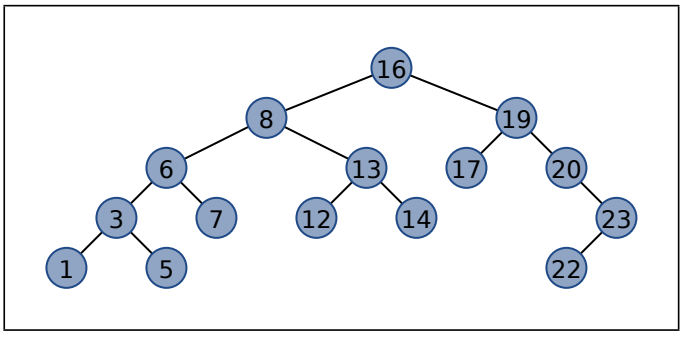

FIGURE 3 – L'arbre de la figure 1 après rotation gauche du nœud de valeur 6.