Travail pratique #01

Objectifs

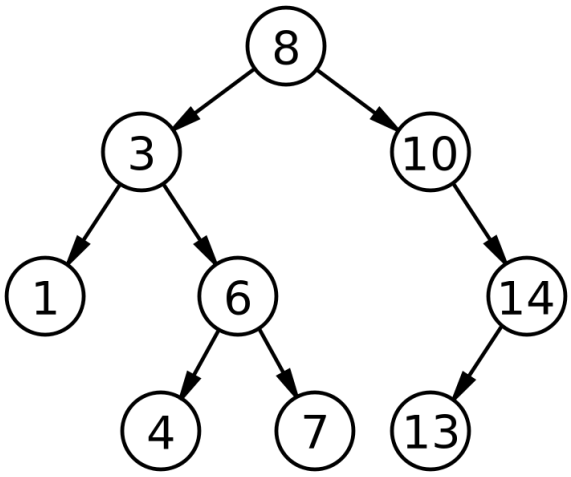
— Pratiquer l'allocation et la libération de mémoire dynamique
— Créer une structure de donnée en mémoire
— Révision des notions de pointeurs
— Utilisation de template pour rendre le code générique
— Compilation multi-plateforme

Règles importantes

1. Chaque classe doit avoir son propre fichier *.h* et *.cpp*
2. N'oubliez pas les *"include guard"*
3. Prenez grand soin de respecter la casse pour les noms de classe, méthode, etc
4. Les règles de l'encapsulation doivent être respectées au maximum, seulement ce qui doit absolument être public peut l'être
5. Veuillez commenter votre code *intelligemment*

Instructions

Vous devez implémenter en c++ une application console comportant une classe conteneur de type *arbre binaire de recherche* (ABR, BST). Votre classe doit être capable de contenir n'importe quel type de donnée (template).



Un arbre binaire de recherche comporte les caractéristiques suivantes :

1. Il ne peut contenir de doublons
2. Les données sont arrangées sous forme d'arbre. Cet arbre contient un noeud principal nommé *noeud racine* sous lequel tout les autres noeuds sont placés en hiérarchie.
3. Chaque noeud (incluant le noeud racine) peut avoir maximum 2 noeuds enfants (0, 1 ou 2 enfants pour chaque noeud)
4. Pour chacun des noeuds, tout noeud à sa gauche a une valeur plus petite que la sienne (numériquement ou alphabétiquement)
5. Pour chacun des noeuds, tout noeud à sa droite a une valeur plus grande que la sienne (numériquement ou alphabétiquement)

6. Les opérations d'ajout, supression et de recherche (méthodes `Ajouter`, `Enlever` et `Contient`) doivent se faire de façon optimale et doivent avoir une complexité algorithmique de `O(logn)`

Il est fortement suggéré de consulter la page suivante : `https://msdn.microsoft.com/en-us/library/ms379572(v=vs.80).aspx`

Avant la remise, assurez-vous de respecter les consignes suivantes :

1. Votre classe doit s'appeler obligatoirement `ArbreBinaire`
2. Votre classe `ArbreBinaire` doit être un template
3. Vous devez programmer vous-même le code qui manipule l'arbre binaire de recherche. Vous ne pouvez pas utiliser des classes conteneurs qui existent déjà dans la librairie standard c++ (comme `std::set`, etc).
4. Votre code doit compiler en Windows (Visual Studio 2019) et sous linux (g++) **sans modifications**.
   (a) `g++ *.cpp -o tp01`
5. Attention de ne pas causer de fuites mémoires (tout ce qui est alloué doit être libéré dans le destructeur).
6. Votre code doit être robuste et être capable de faire face aux imprévus sans planter (exemple : enlever un item inexistant, essayer d'ajouter un item déjà présent, afficher un arbre vide, etc)
7. Votre code doit toujours être le plus optimal possible (usage judicieux de la mémoire et du cpu)

Voici les méthodes que vous devez implémenter au **minimum** dans votre classe **ArbreBinaire** :

Listing 1 – Classe ArbreBinaire

```
template <class T>
class ArbreBinaire
{
    public:
        ArbreBinaire();                         // Construit un arbre vide
        ~ArbreBinaire();                        // Libere toute la memoire allouée

        void Ajouter(const T& valeur);          // Ajouter un item
        void Enlever(const T& valeur);          // Enlever un item
        bool Contient(const T& valeur) const;   // Retourne vrai si l'arbre contient l'item recherché
        int Nombre() const;                     // Retourne le nombre d'items dans l'arbre

        T Minimum() const;              // Retourne la plus petite valeur
        T Maximum() const;              // Retourne la plus grande valeur

        void AfficherCroissant() const;         // Affiche le contenu de l'arbre en ordre croissant (avec un espace entre chaque item)
        void AfficherDecroissant() const;       // Affiche le contenu de l'arbre en ordre décroissant (avec un espace entre chaque item)

    private:
        // ...
};
```

Points bonus

Cette section est facultative mais donne la possibilité de recevoir des points bonus sur ce travail (pour récupérer des points perdus si c'est le cas) si elle est terminée en temps. Dans tout les cas il est fortement recommandé de la faire.

Veuillez, pour chacune des fonctions suivante, créer le code qui effectue l'opération demandée en commentaire (on assume que le type *int* a une taille de 32 bit) :

Listing 2 – Code à implémenter

```
int CompterBit1(unsigned int valeur)
{
    // Retourne le nombre de bit setté à 1 dans la valeur reçue
}

bool EstPair(unsigned int valeur)
{
    // Retourne true si la valeur reçue est paire, false sinon
    // Vous devez utiliser l'opérateur ET binaire (&)
}

bool EstPuissanceDeux(unsigned int valeur)
{
```

```
    // Retourne true si la valeur reçue est une puissance de 2, false sinon
}

unsigned int InverseBit(unsigned int valeur)
{
    // Inverse l'ordre des bit de la valeur reçue et retourne le résultat
    // Le premier bit devient le dernier, le deuxième devient l'avant dernier, etc
    // Exemple 1: 01001011 -> 11010010
    // Exemple 2: 10101010 -> 01010101
    // Exemple 3: 11100110 -> 01100111
}
```

Remise

Pour ce travail vous devez me remettre sur vortex les fichiers source **seulement** (*.cpp, *.h).