TP : Arborescence binaire

La démarche préalable à suivre

Dans l'espace pédagogique téléchargez les fichiers *AB.h*, *AB.cpp* et *SortieLatex.cpp*.
Compilez ce dernier fichier, dans le dossier où vous allez réaliser votre TP, avec l'ordre de compilation :
g++ -c SortieLatex.cpp

Vous aurez à implémenter le fichier *AB.cpp*.

travail demandé

Il s'agit de représenter un arbre binaire, étiqueté sur les sommets par des objets de type *valeur*.
Idéalement, il faudrait utiliser un type paramétré, mais pour simplifier la programmation,
nous utiliserons la définition de type

```
typedef int Valeur;
```

pour étiqueter par des *int*.

1. soient les déclarations et définitions suivantes :

   ```
   class Sommet;
   typedef Sommet* AB;

   class Sommet {
     protected:
     Valeur racine;
     AB SAG, SAD;
     public:
   // la suite aux questions suivantes
   ```

   — Où se trouve l'etiquette d'un sommet ?
   — comment voit-on que l'arborescence est binaire ?

2. On définit maintenant un constructeur

   ```
   Sommet(Valeur v);
   ```

   qui fabrique une arborescence réduite à un seul sommet étiqueté par la valeur v.
   Écrire la méthode correspondante.

3. Écrire la méthode du constructeur par recopie

   ```
   Sommet(Sommet& s);
   ```

   qui recopie toute l'arborescence de racine le sommet s.

4. Écrire la méthode

   ```
   bool FeuilleP();
   ```

   qui indique si l'objet avec lequel est invoquée la méthode est réduit à une feuille.

5. Écrire les deux méthodes

```
void SupprimerSAG();
void SupprimerSAD();
```

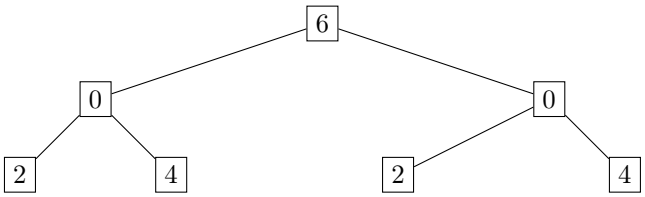
qui suppriment respectivement les sous arborescences gauche et droite de l'objet avec lesquelles elles sont invoquées

6. puis les deux méthodes

```
void GrefferSAG(AB p);
void GrefferSAD(AB p);
```

qui remplacent les sous arborescences respectivement gauche et droite de l'objet avec lesquelles elles sont invoquées par le paramètre *p*.

7. Construire avec les méthodes implémentées jusqu'à présent l'arborescence



Travail à rendre

Dans le fichier que vous rendrez, le `main` devra utiliser les deux méthodes de greffe pour fabriquer une arborescence, utiliser le constructeur par recopie sur la racine et utiliser **sur cette copie** chacune des deux méthodes de suppression.
Il devra aussi tester la méthode `FeuilleP`

Dessin des arborescences binaires

Pour pouvoir tester vos résultats, vous utiliserez la fonction

void SortieLatex(AB Ar) ;

déclarée dans le header *AB.h* et implémentée dans le fichier *SortieLatex.cpp*

`Comprenez vous pourquoi cette fonction est déclarée` ***`friend`*** `de la classe` ***`Sommet`*** `?`

exemple d'utilisation de cet outil de dessin :

exemple d'utilisation de cet outil de dessin :

— acceptons que le code ci dessous du *main* du fichier *AB.cpp* fabrique une arborescence *A0* que vous voulez dessiner.

```
int main() {
  // fabrication de A0
  AB A0=new Sommet(0), A1=new Sommet(1), A2=new Sommet(2),
        A3=new Sommet(3),A4=new Sommet(4), A5=new Sommet(5);
  A3->GrefferSAG(A4);  A3->GrefferSAD(A5);  A0->GrefferSAG(A2);  A0->GrefferSAD(A4);
  // impression
  SortieLatex(A0);
  return 1;
}
```

— compilez avec l'ordre *g++ SortieLatex.o AB.cpp*, puis exécutez par l'ordre *./a.out*

— vous voyez s'afficher le fichier *fig.pdf* qui contient le dessin de votre arborescence.

— **Attention** : on ne peut dessiner qu'un arbre à la fois !

8. On veut maintenant rajouter (en *protected*)
   — un champs *Pere* tel que chaque sommet aie un pointeur sur son père (s'il n'a pas de père, le contenu de ce champs devra être `NULL`)
   — un champs *FGP* qui, quand le contenu du champs *Pere* n'est pas `NULL`, indique si le sommet est fils gauche de son père.

   Modifier toutes les méthodes en conséquence.

9. Écrire alors la méthode (publique)

   `void RemplacerPourLePerePar(AB);`

   telle que après l'invocation $\mathcal{A}_D \rightarrow RemplacerPourLePerePar(\mathcal{A}_P)$ l'arborescence de gauche soit devenue l'arborescence de droite.