

TP2. Mise en oeuvre de la classe générique Graphe<S,T> (2h)

1. Mise en oeuvre de la classe (non générique) GElement

GElement (pour Graphe-Element) est la classe de base dont héritent les classes *Sommet*<T> et *Arete*<S,T>.

Elle permet de factoriser les notions communes à ces deux classes. Ici, la seule notion commune à *Sommet*<T> et *Arete*<S,T> est une clef primaire. *GElement* est donc une classe dont le seul attribut est un entier *clef* servant de clef primaire.

Il faut donc :

Ecrire la classe *GElement* ainsi définie. Pour simplifier, le membre *clef* peut être public.

Munir *GElement* d'un constructeur, d'un opérateur de conversion en *string* (fonction membre dont la signature est : **operator string () const**) et de l'opérateur << d'écriture sur un flux (cette dernière méthode est une fonction ordinaire, elle n'est ni membre, ni *friend*, elle peut avantageusement se servir de l'opérateur de conversion en *string*).

Il est inutile d'écrire destructeur, constructeur de copie ou getters et setters.

Ecrire (dans un fichier *TestGraphe.cpp*) une fonction *main()* pour tester les fonctionnalités des classes *GElement*, *Sommet*<T>, *Arete*<S,T> et *Graphe*<S,T>. Tester l'unique constructeur de *GElement*, l'opérateur de conversion en *string* et l'opérateur <<.

2. Mise en oeuvre de la classe générique Sommet<T>

La classe générique *Sommet*<T> représente un sommet dans un graphe.

Elle dérive de *GElement* et contient deux attributs :

degre : un entier représentant le degré du sommet

v : instance de la classe générique *T*, représentant l'information associée au sommet.

Ecrire la classe *Sommet*<T> ainsi définie. Pour simplifier, *degre* et *v* peuvent être publics.

Munir *Sommet*<T> d'un constructeur, d'un opérateur de conversion en *string* (fonction membre dont la signature est : **operator string () const**) et de l'opérateur << d'écriture sur un flux (cette dernière méthode est une fonction ordinaire, elle n'est ni membre, ni *friend*, elle peut avantageusement se servir de l'opérateur de conversion en *string*).

Par défaut, un sommet est créé isolé, donc de degré nul.

Nous faisons l'hypothèse que la classe *T* est munie d'un opérateur << d'écriture sur un flux.

Il est inutile d'écrire destructeur, constructeur de copie ou getters et setters. Tester constructeur, opérateur *string* et opérateur << pour les classes *Sommet*<double> et *Sommet*<string>.

3. Mise en oeuvre de la classe générique Arete<S,T>

La classe générique *Arete*<S,T> représente une arête dans un graphe.

Elle dérive de *GElement* et contient trois attributs :

debut : de type *Sommet*<T>* représentant un pointeur sur l'extrémité initiale de l'arête

fin : de type *Sommet*<T>* représentant un pointeur sur l'extrémité finale de l'arête

v : instance de la classe générique *S*, représentant l'information associée à l'arête.

Ecrire la classe *Arete*<S,T> ainsi définie. Pour simplifier, *debut*, *fin* et *v* peuvent être publics.

Munir *Arete*<S,T> d'un constructeur, d'un opérateur de conversion en *string* et de l'opérateur << d'écriture sur un flux.

Nous faisons l'hypothèse que la classe *S* est munie de l'opérateur << d'écriture sur un flux. Il est suffisant, pour la conversion en *string* d'une arête, d'indiquer les clefs primaires des sommets *debut* et *fin*.

Il est inutile d'écrire destructeur, constructeur de copie ou getters et setters.

L'unique constructeur de *Arete<S,T>* ne fait pas de copie de ses arguments de type *Sommet<T>**. Tester constructeur, operateur string et opérateur << pour la classe *Arete<string,char>*.

Munir la classe *Arete<S,T>* de la méthode suivante :

bool estEgal(**const** Sommet<T> * s1, **const** Sommet<T> * s2) **const**;

qui vérifie si l'arête est égale à l'arête s1 - s2.

Renvoie **true** si *this == s1-s2 ou si *this == s2-s1

Renvoie **false** sinon.

Tester la méthode.

4. Mise en oeuvre de la classe générique *Graphe<S,T>*

La classe générique *Graphe<S,T>* représente un graphe quelconque. *T* est la nature des informations associées aux sommets et *S* celle des informations associées aux arêtes.

Un graphe est défini par une liste de sommets, par une liste d'arêtes et par un générateur de clefs primaires.

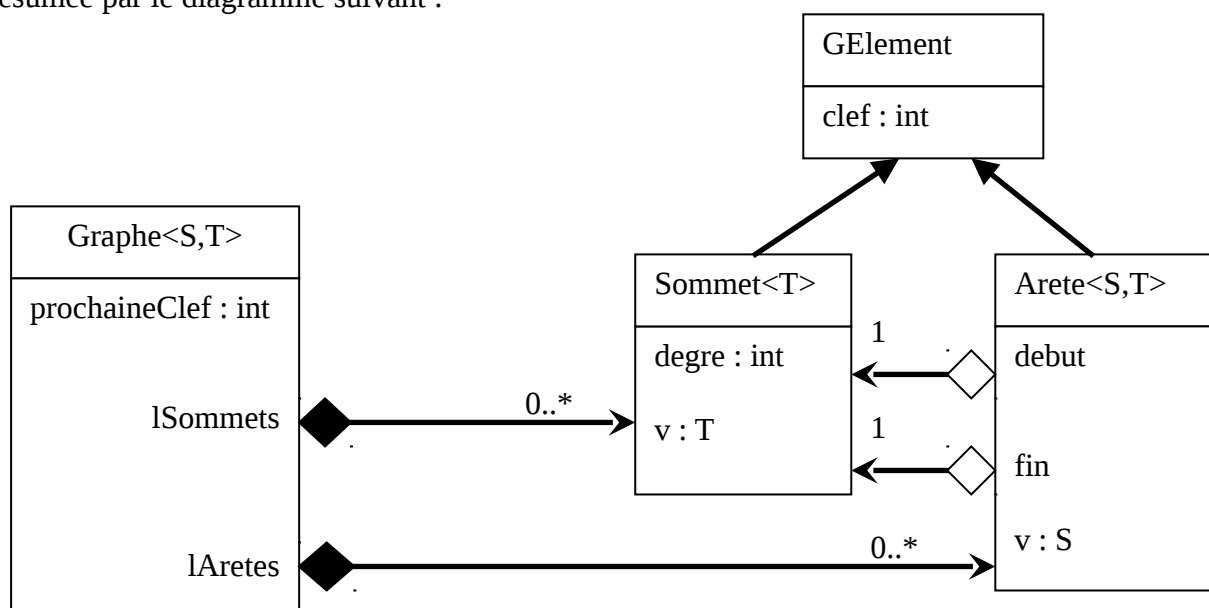
La classe *Graphe<S,T>* est donc munie de trois attributs :

lSommets : de type *PElement< Sommet<T> > **, qui contient la liste des sommets.

lAretes : de type *PElement<Arete<S,T> > **, qui contient la liste des arêtes.

prochaineClef : de type *int* qui définit la clef primaire du prochain élément (sommets ou arête) qui sera inséré dans le graphe. *prochaineClef* est incrémenté à chaque insertion d'un élément.

L'organisation des classes *GElement*, *Sommet<T>*, *Arete<S,T>* et *Graphe<S,T>* peut être résumée par le diagramme suivant :



4.1 Ecrire la définition de la classe Graphe <S,T>

Ecrire la définition de la classe avec ses attributs.

4.2 Munir la classe Graphe <S,T> du constructeur suivant :

Graphe();

qui crée un graphe vide. Initialise de façon cohérente les attributs *prochaineClef*, *ISommets* et *IAretes*.

Note : On peut aussi mettre en oeuvre un constructeur de copie, un destructeur et l'opérateur d'affectation =. Ces 3 méthodes sont nécessaires à une gestion rigoureuse de la mémoire. C'est ici obligatoire car la classe comporte des membres alloués dynamiquement par ses soins. Cependant, par manque de temps, nous ne le ferons pas pendant ce TP.

4.3 Munir la classe Graphe<S,T> de la méthode suivante :

Sommet<T> * creeSommet(const T & info);

qui crée, dans le graphe, un sommet isolé muni de l'information *info*. Un pointeur sur le sommet créé est retourné.

4.4 Munir la classe Graphe<S,T> de la méthode suivante :

Arete<S,T> * creeArete(Sommet<T> * debut, Sommet<T> * fin, const S & info);

qui crée, dans le graphe, une arête reliant les sommets *debut* et *fin*. A l'appel, on suppose que *debut* et *fin* sont déjà contenus dans le graphe. L'arête créée contient l'information *info*. La méthode ne fait pas de copie de *debut* ou de *fin*. Les degrés de *debut* et de *fin* sont mis à jour par la méthode. Un pointeur sur l'arête créée est retourné.

4.5 Munir la classe Graphe<S,T> de la méthode suivante :

int nombreSommets() const;

qui renvoie le nombre de sommets contenus dans le graphe.

4.6 Munir la classe Graphe<S,T> de la méthode suivante :

int nombreAretes() const;

qui renvoie le nombre d'arêtes contenues dans le graphe.

4.7 Munir la classe Graphe<S,T> des opérateurs suivants :

opérateur de conversion en *string* et opérateur << d'écriture sur un flux.

4.8 Munir la classe Graphe<S,T> de la méthode suivante :

Arete<S,T> * getAreteParSommets(const Sommet<T> * s1, const Sommet<T> * s2) const;

qui retourne l'arête s'appuyant sur *s1* et *s2* (on rappelle qu'une arête n'est pas orientée).

Retourne NULL si l'arête *s1* - *s2* n'existe pas dans le graphe.

4.9 Munir la classe Graphe<S,T> de la méthode suivante :

```
PElement< pair< Sommet<T> *, Arete<S,T>* > > *  
adjacences(const Sommet<T> * sommet) const;
```

qui construit, pour le sommet *sommet*, la liste de toutes les associations (voisin de *sommet*, arête incidente en *sommet*).

De cette liste, on peut donc déduire facilement, par exemple, la liste des voisins de *sommet* ou encore la liste des arêtes adjacentes à *sommet*.

Cette liste est essentielle au déroulement de A^* .

4.10 Tester les fonctionnalités de Graphe<S,T>

Tester les fonctionnalités de Graphe<S,T> **au fur et à mesure** qu'elles sont écrites.



Créer un Graphe<string,char> à 5 sommets, dont un isolé, puis tester dans l'ordre le constructeur de graphe vide, l'opérateur <<, les méthodes *creeSommet()*, *creeArete()*, *getAreteParSommets* puis enfin *adjacences()*.