Colin Troisemaine
Guillaume Bouchard

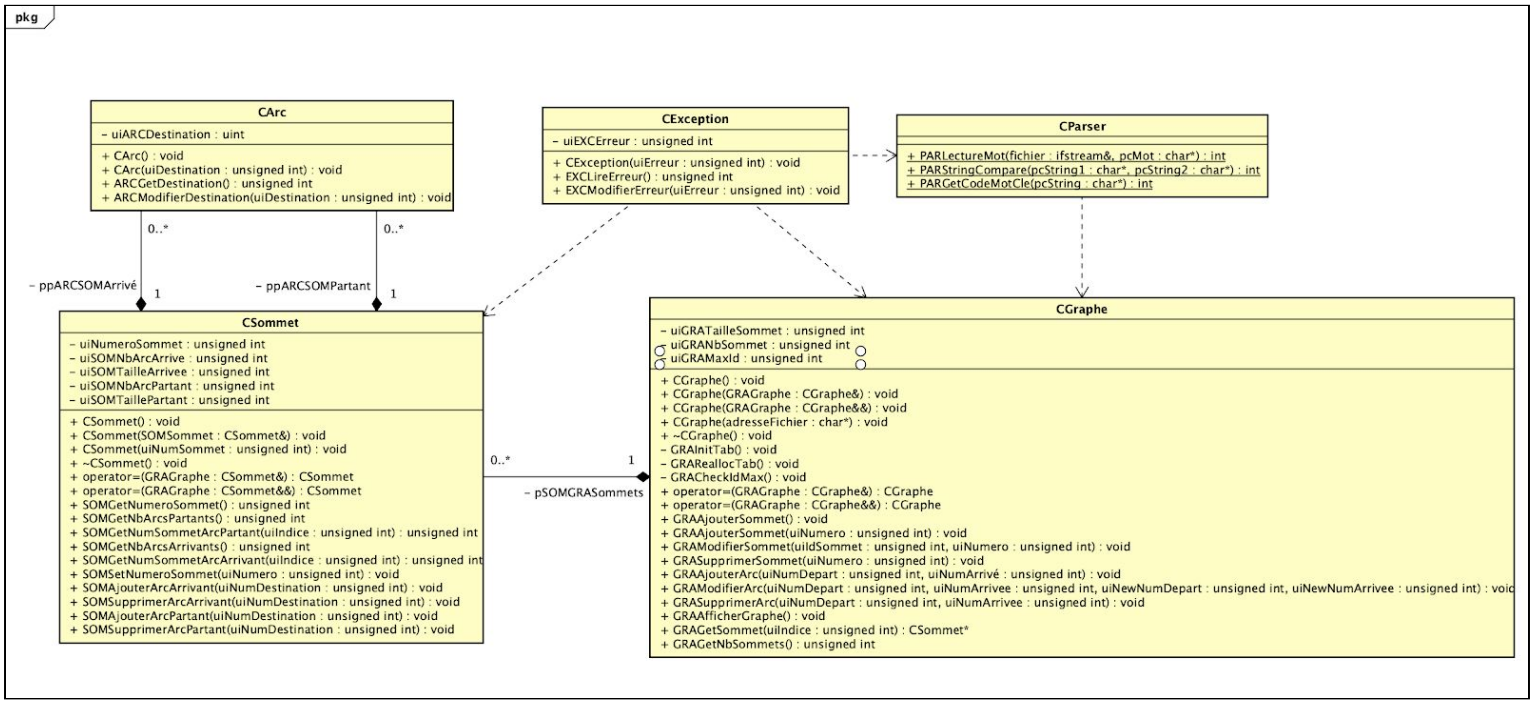
Projet C++: Les graphes

Réalisation d’une librairie de classes et fonctions permettant de manipuler des graphes.

I. Architecture de la librairie

Le but de ce projet était de créer une librairie de classes et fonctions permettant de manipuler des graphes. L'architecture de cette librairie nous est imposée et se présente sous la forme suivante :
Un graphe comporte des sommets qui comportent eux-même des arcs. Chaque classe stocke également un certain nombre de données utiles au fonctionnement de celle-ci, comme le nombre de sommets, la taille du tableau actuellement alloué, le nombre d'arcs partants, etc…
Comme pour le projet des matrices, nous utilisons une classe CException dont se servent les classes CGraphe, CSommet et CParser pour lever des exceptions lors de cas anormaux mais prévisibles de fonctionnement.
Nous avons également pu réutiliser la classe CParser pour la lecture du fichier mot par mot et la création du graphe correspondant.

Voici le diagramme de classes correspondant à notre projet :



II. Justification des choix importants

Classe CArc :
Un CArc est défini par deux paramètres : On considère qu'il **part** du sommet où il est stocké et son sommet de destination est repéré par le numéro que l'arc stocke.

Classe CSommet :
Un CSommet stocke deux tableaux de CArc : Les arcs dits "partants" et les arcs "arrivants". L'avantage de faire comme cela est l'on pourra facilement, et surtout rapidement, parcourir notre CGraphe. En contrepartie, on génère un graphe plus lourd en mémoire.
Pour gagner du temps d'exécution, nous avons également choisi de ne pas stocker des CSommet et des CArc directement, mais des pointeurs de ceux-ci.

Classe CGraphe :
Pour supprimer un arc du graphe, il faut que l'utilisateur spécifie le numéro de départ et le numéro d'arrivée. Cette méthode de suppression implique l'existence d'un méthode dans CSommet qui supprime un CArc de l'un de ses tableaux (et en mode public). C'est alors de la responsabilité de l'utilisateur de choisir de s'en servir ou pas, même si nous avons précisé dans les commentaires de la méthode qu'il fallait éviter de s'en servir, au risque de compromettre l'intégrité du graphe.
Il existe plusieurs autres méthodes pouvant compromettre l'intégrité du graphe. Nous avons pris le soin de le préciser dans les commentaires d'en tête de celles-ci.

Un autre choix important que nous avons dû faire est celui de ne pas autoriser l'utilisateur à ajouter au CGraph un CArc ou un CSommet qu'il aurait créé en dehors de celui-ci.
En effet si l'utilisateur crée un sommet en dehors du graphe, plusieurs soucis peuvent se poser : D'abord le choix du numéro du sommet. Si l'utilisateur le définit, il entrera très probablement en conflit avec les numéros définis automatiquement par le CGraphe. Le dernier problème que nous avons envisagé est le suivant : Si le graphe est détruit, ses sommets le sont également puisqu'ils sont alloués par le graphe. Mais si il y avait un sommet ajouté par l'utilisateur, celui-ci conservera un pointeur sur un objet détruit, ce qui est dangereux.

La solution que nous avons adoptée est la suivant :

- Impossible d'ajouter un sommet que l'on a créé en dehors d'un graphe.
- Uniquement possible de créer un sommet appartenant à un graphe dont le numéro sera géré automatiquement, même si l'utilisateur peut par la suite modifier son numéro avec une méthode de CGraphe.

Constructeur CGraphe(char * pcAdresseFichier) :

Afin de factoriser le code du constructeur et d'augmenter la clarté du code, nous avons choisi de définir 4 méthodes supplémentaires :

- OCMLectureMot : On lit le mot suivant du ifstream passé en paramètre.
- OCMGetCodeMotCle : On récupère le numéro d'un mot-clé (ici "NBSommets", "NBArcs", "Sommets", "Arcs", "Numero", "Debut", "Fin", "[", "]").
- OCMStringCompare : On compare si deux chaînes de caractères sont identiques. Cette méthode sert dans les 2 méthodes de renvoi de code pour comparer les mots-clés ou les valeurs lues par OCMLectureMot.

Toutes ces méthodes ont été crées dans l'optique de pouvoir être facilement adaptées à un fichier avec une disposition et des balises différentes, notamment grâce au tableau de mots-clés.

Le code de cette méthode a été pensé pour qu'il puisse tout de même pouvoir générer un graphe avec un fichier ne respectant pas exactement la mise en forme imposée.Par exemple il pourra générer un graphe **quel que soit l'ordre des lignes** ! Il est ainsi possible de créer un fichier qui déclare les sommets avant le nombre de sommets ou d'arcs, du moment que les valeurs correspondent au nombre d'éléments du graphe. Les espaces et les retours chariot en trop seront également ignorés et n'empêcherons pas la génération du graphe.

Les autres cas comme une valeur manquante ou en trop, l'absence d'une balise ou bien la présence d'un caractère imprévu lèveront tous des exceptions appropriées.

Constructeur/Destructeurs :

Dans notre classe CGraphe, nous manipulons un tableau de pointeurs d'objets de la classe CSommet : *CSommet ** ppSOMGRASommet*. Nous avons donc dû surcharger le constructeur de recopie (ainsi que les opérateurs d'affectation) pour qu'il ne crée pas une autre référence aux sommets du tableau que l'on copie mais bien une copie de chaque sommet (même chose pour la classe CSommet).

Nous avons également surchargé le destructeur pour qu'il explore tous les sommets du graphe et qu'il détruise chaque arc de chaque sommet (grâce au destructeur de CSommet) avant de détruire le sommet lui-même.

Opérateur d'affectation :

Notre classe CGraphe comportant un tableau de pointeurs de CSommets, il nous fallait surcharger l'opérateur d'affectation pour bien recopier les objets et non les pointeurs.

Lorsque nous l'avons surchargé de la même manière que vu en cours, c'est à dire "*CGraphe operator=(CGraphe & GRAGraphe);*", nous avons remarqué que certaines opérations nous étaient impossibles "*graphe = CGraphe(argv[1]);*". Nous avons alors appris que cette simple surcharge ne permettait de n'affecter que des lvalues à une matrice, alors que la ligne que nous avons citée en exemple est ce que l'on appelle une rvalue car elle correspond au résultat d'une opération (avec p un int, "*p+5*" est une rvalue par exemple).

La solution que nous avons trouvée a été de surcharger une nouvelle fois l'opérateur d'affectation de la manière suivante : "*CGraphe operator=(CGraphe && GRAGraphe);*"

Ce qui nous a par la suite permis d'effectuer n'importe quelle opération.

Les mêmes raisons nous ont poussées à surcharger deux fois l'opérateur d'affectation dans la classe CSommet.

III. Tests unitaires

Afin de vérifier le bon fonctionnement de toutes les méthodes et fonctions que nous avons codées, nous avons créé une fonction temporaire appelée testsUnitaires qui affiche un graphe au fur et à mesure que l'on applique toutes les méthodes de notre librairie pour vérifier leur bon comportement.
Voici quelques exemples des tests que cette fonction comporte :

```
int testsUnitaires() {
    CGraphe graphe1 = CGraphe();

    graphe1.GRAAjouterSommet(1);
    graphe1.GRAAjouterSommet(2);
    graphe1.GRAAjouterSommet(3);
    graphe1.GRAAjouterSommet(4);
    graphe1.GRAAjouterSommet();

    graphe1.GRAAjouterArc(1, 2);
    graphe1.GRAAjouterArc(2, 1);
    graphe1.GRAAjouterArc(2, 4);
    graphe1.GRAAjouterArc(3, 4);

    graphe1.GRAAfficherGraphe();
    graphe1.GRASupprimerSommet(1);
    graphe1.GRAAfficherGraphe();
    graphe1.GRAModifierSommet(2, 7);
    graphe1.GRAAfficherGraphe();
    graphe1.GRAModifierArc(7, 4, 7, 3);
    graphe1.GRAAfficherGraphe();
    graphe1.GRASupprimerArc(7, 3);
    graphe1.GRAAfficherGraphe();

    return 0;
}
```

Cette fonction nous a permis de déceler quelques erreurs : Nous avions notamment oublié de supprimer les arcs allant vers un sommet que l'on supprime, cette fonction nous a donc révélé cet oubli.

IV. Utilisation de la librairie

La librairie s'utilise grâce à un unique fichier header. En effet notre classe CGraphe inclut déjà toutes les classes avec les méthodes nécessaires au bon fonctionnement de celle-ci.

CGraphe.h :

Pour créer un nouveau graphique, il existe plusieurs solutions. D'abord de manière statique :

- CGraphe nom;
- CGraphe nom(graphe); Avec graphe un autre graphe.
- CGraphe nom
- CGraphe nom

Et de manière dynamique :

- CGraphe * nom = new CGraphe();
- CGraphe * nom = new CGraphe(graphe); Avec graphe un autre graphe.
- CGraphe * nom = new CGraphe(adresseFichier); Avec adresseFichier le chemin d'un fichier contenant un graphique respectant le format imposé.

Il est également possible de faire :
graphe1 = graphe2; Ce qui appellera l'opérateur d'affectation et copiera correctement le graphe2 dans le graphe1 après avoir vidé le graphe1.

Pour détruire un graphique, il suffit d'appeler le destructeur grâce à la méthode delete qui appellera notre surcharge qui s'occupe de complètement vider le graphe en détruisant sommets et arcs.

Il existe ensuite quelques méthodes pour modifier la structure du graphe :

- GRAAjouterSommet();
- GRAAjouterSommet(unsigned int uiNumero);
- GRAModifierSommet(unsigned int uiIdSommet, unsigned int uiNumero);
- GRASupprimerSommet(unsigned int uiNumero);
- GRAAjouterArc(unsigned int uiNumDepart, unsigned int uiNumArrivee);
- GRAModifierArc(unsigned int uiNumDepart, unsigned int uiNumArrivee, unsigned int uiNewNumDepart, unsigned int uiNewNumArrivee);
- GRASupprimerArc(unsigned int uiNumDepart, unsigned int uiNumArrivee);
- GRAGetSommet(unsigned int uiIndice);
- GRAGetNbSommets();

Enfin, il est possible d'afficher un graphe à l'aide de la fonction GRAAfficherGraphe qui listera les sommets avec tous leurs arcs partants et arrivants.

Concernant le constructeur prenant un char * en paramètre, il faut que le fichier lu respecte la mise en forme suivante :

```
NBSommets=<Nombre_de_sommets_du_graphe>
NBArcs=<Nombre_d_arcs_du_graphe>
Sommets=[
Numero=<Numéro_sommet>
Numero=<Numéro_sommet>
…
Numero=<Numéro_sommet>
]
Arcs=[
Debut=<Sommet_départ_arc>, Fin=<Sommet_arrivée_arc>
Debut=<Sommet_départ_arc>, Fin=<Sommet_arrivée_arc>
…
Debut=<Sommet_départ_arc>, Fin=<Sommet_arrivée_arc>
]
```

Cependant, nous avons construit cette méthode de sorte qu'elle puisse générer un graphe même si les attributs ne sont pas dans le bon ordre, qu'il manque un retour chariot entre deux lignes des sommets/arcs, qu'il y ait des espaces en trop, etc… Mais s'il manque un élément important comme une ligne entière ou un signe '=', la méthode lèvera l'erreur appropriée.