

POLYTECH TOURS

64 avenue Jean Portalis

37200 TOURS, FRANCE

Tél +33 (0)2 47 36 14 14

[www.polytech.univ-tours.fr](http://www.polytech.univ-tours.fr/)

**Projet C++ 2023/2024**

**Projet Graphe Orienté /**

**Etudiant :**

Yassine ELLATIFI

Milan HERGOTT

**Sommaire**

Table des matières

[1. Introduction 2](#_Toc165236872)

[2. Conception UML (ajouter classe inverse) 3](#_Toc165236873)

[2.1 Le diagramme de classe 3](#_Toc165236874)

[3. Revue de code 4](#_Toc165236875)

[3.1 Classe CgrapheOriente 4](#_Toc165236876)

[3.1.1 Explication de nos choix 4](#_Toc165236877)

[3.1.2 Principales méthodes 4](#_Toc165236878)

[3.2 Classe Cfichier 6](#_Toc165236879)

[3.2.1 Explication 6](#_Toc165236880)

[3.2.2 Principales méthodes 6](#_Toc165236881)

[3.3 Classe CSortie 7](#_Toc165236882)

[3.3.1 Explication 7](#_Toc165236883)

[3.3.2 Principales méthodes 8](#_Toc165236884)

[4. Manuel d’utilisation 9](#_Toc165236885)

[4.1 Paramétrage du fichier 9](#_Toc165236886)

[5 Conclusion 11](#_Toc165236887)

# Introduction

Au cours du 6ème semestre de la 3ème année du cycle ingénieur Polytech. Il nous est demandé au sein du cours de Conception Orientée Objet 2 de réaliser une librairie de classe et de fonction permettant de manipuler des Graphes orienté et non orienté. Pour ce projet tutoré, il est nous également imposé de proposer une implémentation flexible pour permettre à l’utilisateur de spécifier lui-même les informations qu’il veut stocker.

Pour effectuer ce projet nous avons essayé au mieux de suivre les principes fondamentaux du génie logiciel tel que l’encapsulation des données ou le respect d’une convention de nommage. En poursuivant cette volonté nous avons dû dans un premier temps, comprendre au mieux les attentes et objectif de ce projet a l’aide de différents diagrammes UML, seulement après avoir obtenu une compréhension et une visualisation suffisante du projet nous nous sommes lancés dans le développement des différentes classe et méthodes.

C’est pour cela que ce rapport est articulé en quatre parties de la manière suivant : dans un premier temps nous allons nous pencher sur la conception UML du projet ce qui pourrait s’apparenter a la partie descendant du cycle en V.

Ensuite nous allons effectuer une revue de code non exhaustive afin de connaitre le fonctionnement des principales méthodes de ce projet, elle parlera également des tests sur les méthodes qui ont pu être effectuées.

La troisième partie quant à elle est adressée directement à l’utilisateur, cette dernière permet de comprendre comment utiliser le programme de la manière qui est attendus par les développeurs.

Enfin pour finir nous allons conclure sur ce projet discuter de nos principales difficultés de ce qu’on pense de notre projet et enfin de si nous pensons avoir atteint les objectifs.

# Conception UML (ajouter classe inverse)

Pour notre conception UML nous nous sommes limités a un diagramme de classe, pour ce projet nous avons pensés que la réalisation d’un digramme d’activité ou un digramme des cas d’utilisation n’était pas pertinent. D’après nous un diagramme d’activité ici permet pleinement de comprendre le projet.

## 2.1 Le diagramme de classe

Pour notre démarche de conception UML nous avons décidé de faire un diagramme de classe ce qui est d’une grande utilité pour la compréhension des relations entre les classes, et pour le passage au code. Nous nous sommes contentés d’un seul diagramme car nous ne trouvions pas pertinent de modéliser le problème a l’aide d’un diagramme des cas d’utilisation ou autre.

Pour notre diagramme des classes nous avons essayé des suivre au mieux les « mantra » du génie logiciel spécifiquement celui de l’encapsulation des données et l’idée de contrat de service pour les classes.

En effet ici nous avons choisi de créer des classes séparées pour l’affichage du graphe, la lecture d’un graphe depuis un fichier et enfin pour inverser un Graphe. Tout cela pour respecter le principe de contrat de service de la classe CGrapheOriente.

Nous avons pris la liberté de modifier le type du nom des Sommet, en effet d’après le sujet le nom des sommets devait être un entier mais par volonté de rendre le code réutilisable nous avons élargi cette perspective en permettant de donner des chaines de caractère.

Une image contenant texte, diagramme, Police, document

Description générée automatiquement

Figure 1 : Diagramme de classe

# Revue de code

## Classe CgrapheOriente

Dans un premier temps nous nous concentrerons sur la classe CgapheOriente. Nous ne nous intéresserons pas à la classe Cgraphe car elle hérite simplement de CgrapheOriente en appliquant deux fois certaines méthodes pour avoir des arcs réversibles. Ces méthodes sont les méthodes associées à la création et à la suppression d’arc entre sommets.

### 3.1.1 Explication de nos choix

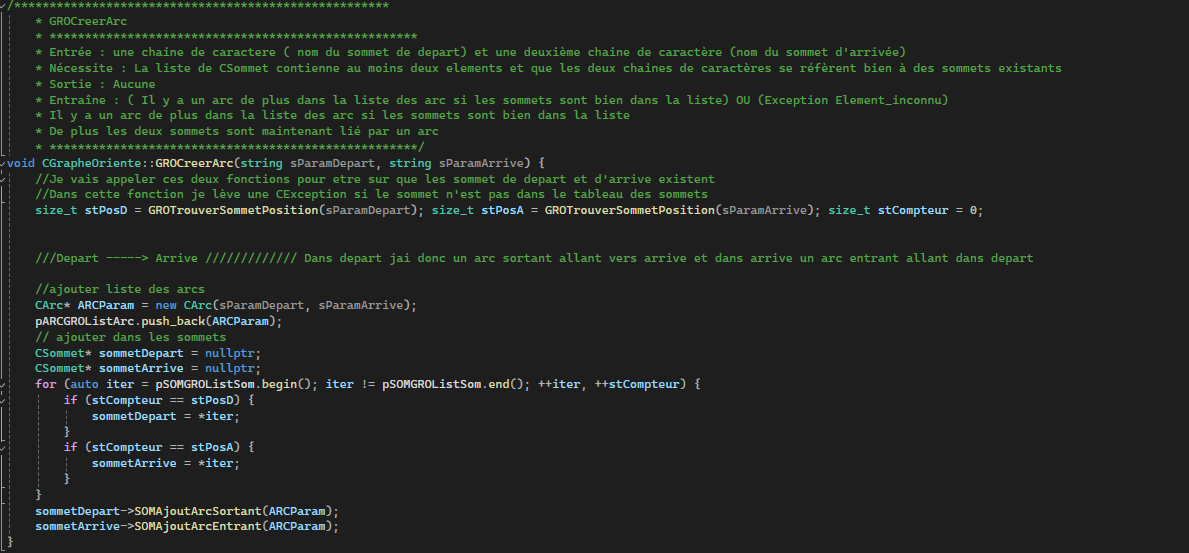
Afin d’avoir une implémentation flexible nous avons préférés avoir comme attributs, des listes de pointeurs, d’arcs et de sommets. Nous avons choisis de passer par des pointeurs car de manière évidente, cela permet de réduire la surcharge de la mémoire. Seul l’adresse de nos objets est stockés dans nos listes, et dans des structures telles que les graphes c’est primordial car on peut avoir de nombreux objets.

Pour stocker nos objets nous avions le choix entre beaucoup de conteneurs différents, en effet le C++, avec la STL offre de nombreuses options. Nous avons principalement réfléchis entre les listes (std::list) et les vecteurs (std::vector). Après recherches et discussion avec notre tuteur, notre choix s’est finalement porté sur les listes pour plusieurs raisons. Les listes permettent d’accéder plus facilement aux éléments. C’est vraiment un gros point positif quand on commence à avoir beaucoup d’objets. L’autre gros avantages des listes est la façon dont elles sont stockées en mémoire, par rapport aux vectors. Les vectors sont stockées sous forme de tableau dynamique tandis que les listes sont sous formes de listes chainées avec l’élément à stocker et un pointeur vers l’élément suivant. Il est donc clair ici étant donné que la taille des données va évoluer dans le programme que les listes sont plus adaptées.

### 3.1.2 Principales méthodes

Cette classe étant la classe contenant les arcs et les sommets et celles les administrant, elle va beaucoup appeler les méthodes de ces derniers pour les manier.

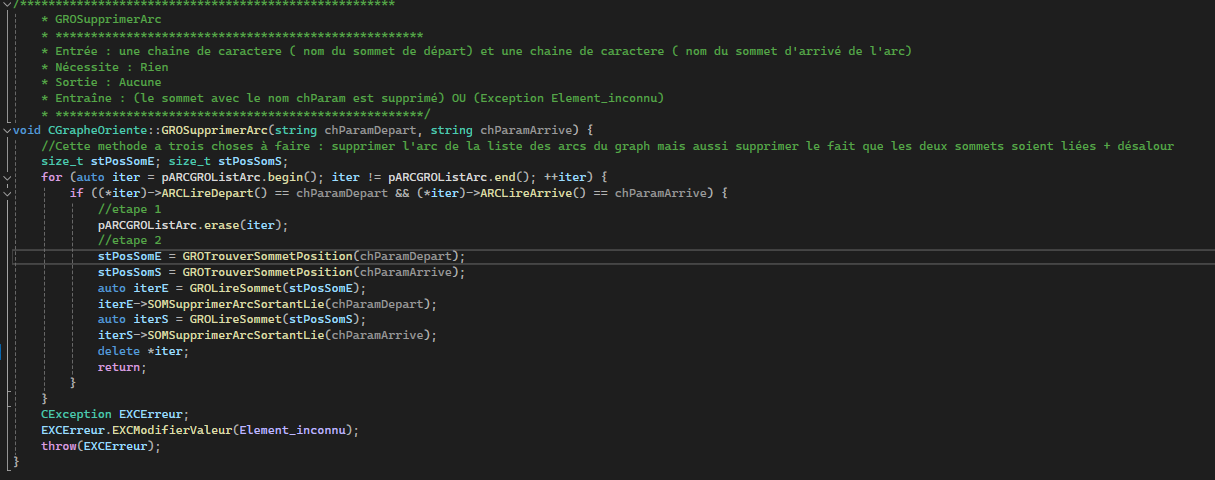
Nous avons choisi d’expliquer les méthodes GROCreerArc, GROSupprimerArc



Cette première méthode a pour objectif comme son nom l’indique de créer un arc, elle prend deux paramètres, deux chaines de caractères correspondant aux noms des sommets à relier.

En appelant GROTrouverSommetPosition sur les deux chaines de caractères on s’assure que les deux sommets font bien parti de la liste des sommets que le graphe connait.

On va commencer par créer un pointeur sur un arc avec un constructeur de confort qu’on implémenter dans la classe CArc. Ce constructeur prend deux chaines de caractères, une pour le sommet de départ et une pour celui d’arrivé. On va ensuite ajouter cet Arc dans notre list d’arcs de graphe orienté. Il nous reste maintenant à indiquer dans le sommet de départ qu’il est lié avec le sommet d’arrivé et qu’il pointe vers lui et l’inverse pour le sommet d’arrivé. C’est ce qu’on va faire avec des iterators et en appelant les méthodes adaptées.



La seconde méthode se charge de la suppression des Arcs. Comme l’insertion elle va devoir supprimer les arcs crées à plusieurs endroits. Pour commencer elle va parcourir la liste de arcs connu par le graphe. Si l’arc n’est pas trouvé on sortira de la boucle et on renverra une exception.

Supposons qu’on est trouvé l’arc dans la liste. On va devoir le supprimer de cette liste. Ensuite il faut aller dans le sommet du quel partait l’arc et faire en sorte qu’il ne soit plus lié au sommet qui recevait l’arc. On doit procéder de la même manière pour supprimer l’arc depuis le sommet récepteur.

Pour se faire on va appeler une multitude de méthodes de la classe CSommet tel que GROTrouverSommetPosition, GROLireSommet, SOMSupprimerArcSortantLie et SOMSupprimerArcSortantLie qui sont toutes assez explicites.

## Classe Cfichier

### 3.2.1 Explication

Pour ce projet la seule réel interaction que l’utilisateur peut avoir avec le projet se fait a travers un fichier. En effet, pour utiliser notre application il faut disposer d’un fichier suivant un formalisme très précis. Pour en savoir plus sur comment utiliser notre application, un manuel d’utilisation est proposé plus loin dans le rapport.

### 3.2.2 Principales méthodes

L’une des principales attentes de ce projets est la lecture d’un graphe depuis un

fichier suivant un format précis. Le formalisme que respecte le fichier est composé

de balise tel que “ Numéro=1” ou encore “Début=1, Fin=4”. Par conséquent la

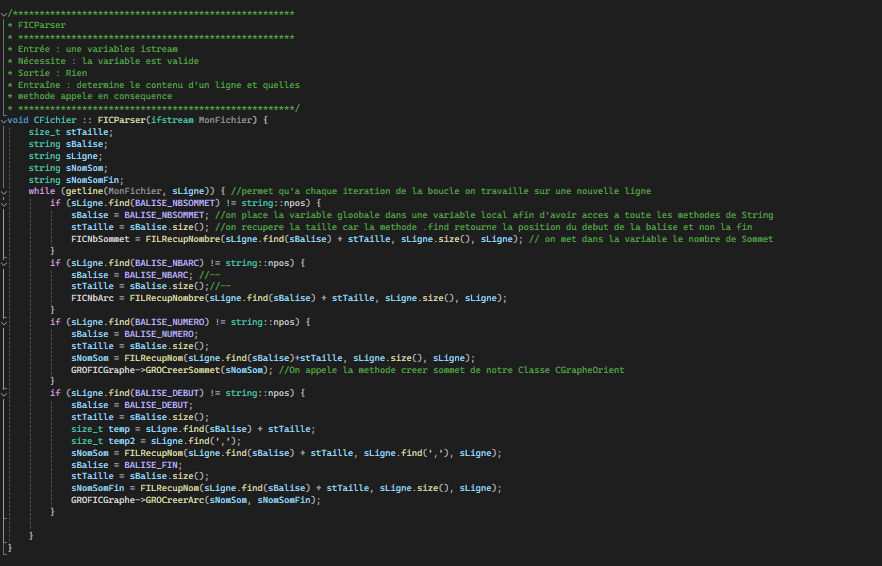
principale méthodes de cette classe est l’analyse syntaxique (parser) d’un fichier

pour la création d’un graphe donné par l’utilisateur.

Pour réaliser ces méthodes nous avons utilisé les bibliothèques fstream et iostream

qui permettent de parcourir des fichier avec des méthodes tel que getline ou encore

find.



Pour cette méthode nous avons pris une liberté vis-à-vis du sujet. En effet, nous avons décidé d’ajouter une balise supplémentaire dans le fichier du graphe. Cette dernière permet de déterminer si le graphe est un graphe Orienté ou Non Oriente. Evidement nous aurions pu permettre à l’utilisateur de paramétrer ceci directement depuis le terminal à l’aide des commandes cin et d’une verfication syntaxique de l’entrée. Cependant comme les seules entrée de l’utilisateur se faisait à travers le fichier il nous est paru logique de continuer dans cette lancée.

Cette nouvelle balise se présente comme suit :

**TypedeGraphe= <NonOriente>**

**Ou**

**TypedeGraphe= <Oriente>**

Si cette balise n’est pas correctement ou n’est pas implémenté dans le fichier alors le graphe sera par défaut un graphe orienté

## Classe CSortie

### Explication

Nous avons décidé de gérer l’affichage de notre graphe dans une fonction à part entière. Nous l’avons mis dans une classe à part afin de suivre les préceptes du génie logiciel. En effet chaque classe ne doit servir qu’à une seule tâche et nous trouvions que l’affichage se devait d’être séparé. De plus cela permet d’avoir un code plus facilement compréhensible, plutôt qu’une seule fonction qui fournirait tous les services. Cela permet aussi d’avoir un code réutilisable, évolutif et encapsulé.

L’affichage du graphe est lisible grace a des “balise” predefini assez intuitive qui garantit un affichage clair.

* + 1. Principal methodes

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

La principale methode de la classe Csortie; SORAfficher\_Graphe permet l’affichage de tous les arc lié a un sommet en suivant un format d’affichage clair et lisible a l’aide de fleche.

Une image contenant texte, capture d’écran, Police, menu

Description générée automatiquement

### 3.3.2 Principales méthodes

La classe CSortie possède une méthode essentielle de ce projet.



Dans cette méthode on veut afficher notre graphe. Pour se faire on va parcourir la liste des sommets avec un pointeur sur la liste des sommets. Pour chaque sommet on va parcourir ses arcs sortant et les afficher puis les arcs entrant en appelant des méthodes simples de CSommet.

## Tests

Nous avons effectué bon nombre de tests dans une classe Ctest, ces tests vont principalement visés les classes CgrapheOriente, CSommet et Carc. Les tests vont visés à tester le comportement de nos fonctions dans tout les cas d’utilisation possible.

# Manuel d’utilisation

## 4.1 Paramétrage du fichier

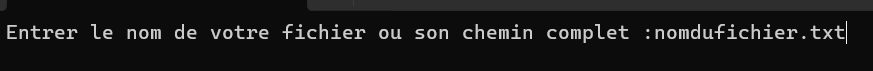
Une image contenant texte, Police, capture d’écran, typographie

Description générée automatiquementComme expliqué précédemment le fichier contenant le graphe doit suivre un formalisme précis, nous allons maintenant nt l’expliquer plus en détails.

Comme le montre l’image ci-dessus il existe 8 balise différentes dont une ajouté par nous-même.

Ainsi pour créer un fichier qui sera compris par notre application il faut suivre le format spécifié dans le sujet du projet et rappelé par la figure précédente. Quant a notre balise TypedeGraphe=… elle accepte seulement deux valeur possible qui permettent de déterminer le type de Graphe du fichier. Ces valeurs sont “Oriente” si c’est un Graphe Orienté, et “NonOriente” si il s’agit d’un graphe non Orienté. Si cette balise n’est pas correctement employé ou est absente alors le graphe sera par défaut un graphe orienté.

Apres avoir créé le fichier, il est temps de lancer l’exécution du programme. Lors de l’exécution du programme l’application demandera le nom du fichier ou son chemin complet il suffit d’entrer.



Enfin il suffit de lancer d’appuyer sur entrée pour valider, si le nom du fichier est incorrect il vous sera proposé de réessayer. Si tout se passe on obtient un affichage qui se décomposent en 2 partie .

La première affichage notre tel qu’il est, plus de précision sont donnée dans la partie CSortie du rapport.

Une image contenant texte, capture d’écran

Description générée automatiquement

La seconde affiche le graphe inversé.

Une image contenant texte, capture d’écran

Description générée automatiquement

# 5 Conclusion

Pour conclure, nous pensons avoir plutôt bien respecté les principes du génie logiciel, nous avons dans ce but effectué des classes pour que chaque classe ne fasse qu’une seule chose. Nous avons fait le choix d’avoir plus de classes mais avec des méthodes le plus simples possible, nous avons aussi respecté une convention de nommage assez lourde mais qui nous a bien aidé à comprendre les objets que nous manipulions. Cela nous a permis de comprendre bien plus facilement le travail de l’autre. Afin de travailler en parallèle nous avons mis en place un git hub qui nous a beaucoup aidé afin de pouvoir travailler à deux. Chacun pouvait avancer sur ces branches puis on a pu regrouper le travail facilement.

Nous avons beaucoup apprécié la première partie et nous attendons la deuxième avec impatience.