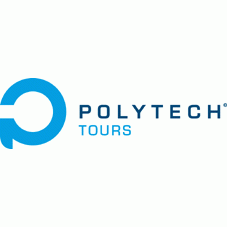
PERVEYRIE Hugo DI3 Groupe 2

DURET Romuald

Compte-rendu de projet :

# Librairie de graphe (C++)



30/05/2021

# Introduction :

Dans ce rapport vous trouverez tout le nécessaire à la compréhension de notre démarche sur la conception et la réalisation de ce projet. Par ailleurs, comme vu précédemment, il y a une partie consacrée à l’utilisation de la librairie afin d’accompagner tout utilisateur qui voudrait l’utiliser.

Tout d’abord, la première partie vous permettra de comprendre notre réflexion sur la conception du projet. Ce travail en amont nous a permis de pouvoir travailler dans de bonnes conditions et d’une façon structurée.

Ensuite, la seconde partie expliquera comment s’est déroulé le développement de la librairie. Nous parlerons donc de la prise en main de l’IDE, les difficultés que nous avons rencontrées ainsi que des différentes méthodes de développement que nous nous sommes employés à utiliser.

Enfin, nous conclurons sur ce que ce projet nous a apporté par rapport à nos connaissances personnelles et celles vu tout au long de ce semestre à Polytech.

# I – Conception

Pour la conception de la structure du projet nous avons décidé de séparer les opérations élémentaires du graphe, du sommet et de l’arc des autres opérations. Ces classes contiennent donc les constructeurs et accesseurs et quelques méthodes simples d’ajout et de suppression d’objet dans une liste. Ces méthodes sont contenues dans ces classes pour des raisons de sécurité concernant les allocations d’espace dans les listes. En effet on souhaite que tous les traitements de liste soient effectués en interne pour éviter des problèmes de pointeurs tels que l’écriture dans un espace non alloué.

Les opérations non élémentaires sont contenues dans l’Interface CGrapheOperation qui contient toutes les méthodes sur le traitement des graphes, sommets et arcs tels que le traitement de l’ajout d’un arc ou sommet dans un graphe ou la suppression de celui-ci. Cette séparation permet une bien meilleure lisibilité du code et des ajouts bien plus simple pour de futures méthodes complexes.

En raison de la ressemblance des listes d’arcs et de sommets, nous avons décidé d’implémenter une Interface Template CListeOperation prenant en charge toutes les opérations élémentaires sur des pointeurs de liste de pointeurs. On y trouvera les méthodes d’allocations de listes, d’ajout et de suppression d’objet et les affichages de listes. Cette séparation aide encore le code à être plus lisible et nous permets de regrouper les opérations de traitement de liste.

Pour la lecture de fichier, nous avons créer une classe Parser qui contient un parser pour récupérer les informations importantes d’un fichier et une méthode compare pour comparer deux chaines de caractères afin d’étudier la syntaxe du fichier.

Nous avons aussi ajouté une classe CException qui nous permet de lever nos propres exceptions concernant le traitement des graphes.

Vous pouvez retrouver ci-dessous le diagramme de classe du projet :

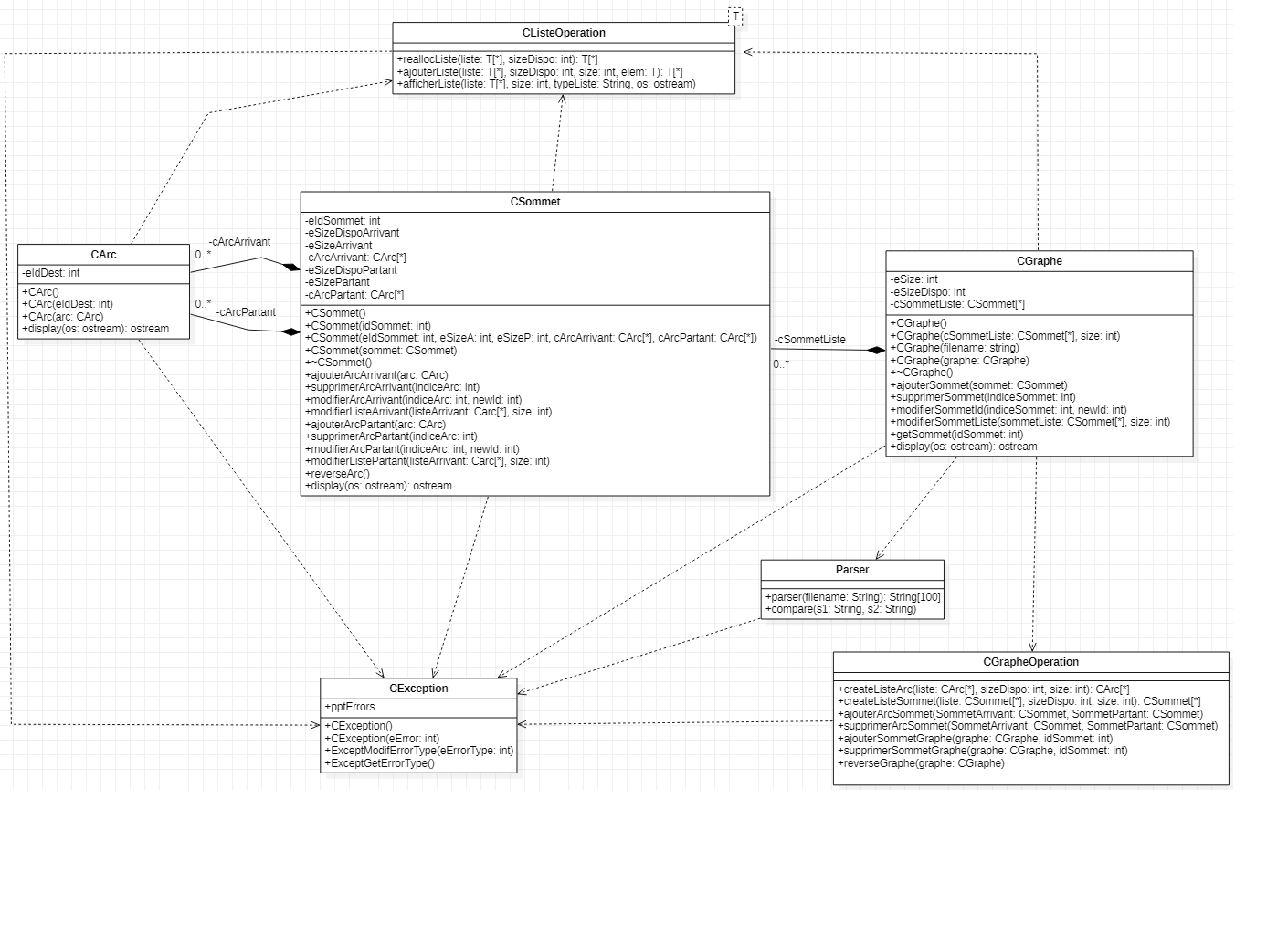


Image 1 : Vue globale du diagramme de classe

# II – Développement

Pour le développement de ce projet, il était important de choisir des outils adéquats pour sa réalisation. Tout au long de l’année on nous a appris à compiler manuellement ou en utilisant le site Repl.it donc sans utiliser d’IDE, or pour ce projet, l’utilisation de Visual Studio 2017 nous était imposé. De plus, Visual Studio 2017 proposant des outils Git, il aurait été dommage de ne pas les utiliser. Ainsi tout notre projet est disponible sur Git Hub. L’utilisation du git fut pratique car elle nous a permis de travailler en collaboration sans se soucier de la synchronisation du projet. L’expérience que nous as procuré le projet Matrice nous a permis de travailler le projet plus rapidement que sur ce dernier.

Pour ce qui est du développement même, nous nous sommes beaucoup inspirés du projet précédent pour l’organisation et de nombreux forums tels que stackoverflow et les forums Microsoft. L’analyse du projet nous a pris plus de temps que ce que l’on pensait car le projet était assez conséquent et une mauvaise analyse pouvait vite nuire à la lisibilité du code. Nous avons également profité des cours de Génie Logiciel et d’Algorithme Orientée Objet pour respecter au mieux les normes de nommage ainsi que les bonnes manières pour développer. Nous avons donc essayé au mieux d’appliquer ces méthodes en tentant d’anticiper pour chaque fonction les potentiels problèmes qui pourraient être gérés en pré ou postcondition.

La gestion de ces problèmes s’est faite via la classe CException qui permet de lever des exceptions personnalisées. Ainsi à travers notre analyse en amont nous avons pu anticiper la plupart des erreurs que l’utilisation de la librairie sur des graphes peut engendrer. Cette classe possède 2 attributs : l’indice d’erreur et le tableau des messages d’erreurs. Les différentes classes du projet possède de nombreuses constantes correspondant à chaque erreur possible (*Exemple : CGrapheEmptyFile vaut 1 et correspond à l’erreur d’un fichier vide*). // ¨PARTIE ROMUChaque indice d’erreur est classé, par exemple, les constantes de 1 à 10 correspondent à la création d’une matrice via un fichier, à partir de 51, elles correspondent à des erreurs liées aux calculs. Donc, il y a une section try-catch dans le main qui dès qu’il y une CException qui est levée dans CMatrice elle arrive dans le main qui s’occupe de chercher via l’indice d’erreur le message associé dans le tableau de message d’erreurs.

Le projet Matrice nous a permis de comprendre l’utilité des classes Template, ce qui nous as amenés à gérer les opérations sur les listes de cette façon. Ainsi nous avons pu centraliser toutes les méthodes de gestion de pointeurs et réduit de façon conséquente la taille de notre code.

Finalement le cours de gestion des fuites de mémoires en génie logiciel nous a été très utile. La librairie CRT de Visual Studio nous a permis de traquer toutes les fuites de mémoires de notre projet lors de la suppressions d’un graphe et d’un sommet. Celles -ci ont toutes été résolus ce qui rend notre projet très robuste lors d’une longue utilisation et lors de l’implémentation d’une grande quantité de sommets/graphes.

# Conclusion :

Pour ce projet, nous avons du assemblés une grosse partie des connaissances des cours de C++, de Génie Logiciel et d’UML. Ce travail assez conséquent était l’un des plus intéressant tant du côté de la gestion des pointeurs que de l’analyse UML. Nous l’avons travaillé comme une continuité du projet Matrice avec l’envie d’améliorer les différents points qu’il manquait dans ce dernier. Pour cela, la revue de code du projet Matrice envoyé par M. T’KINDT nous a été très utile notamment pour l’implémentation du Parseur et de l’analyseur syntaxique. Finalement, ce projet nous a aidé à comprendre de mieux en mieux le langage C++ et l’utilisation des pointeurs ce qui est utile dans un langage qui est pour nous, l’un des plus complexes vu en cours.