Auteurs : CACHARD Sylvain, NICOLAS Julien, LEONG Loris

Compte rendu TP C++ n°2 : Héritage Polymorphisme

I. Description des classes et de l'application

Afin de manipuler et de comprendre le fonctionnement d'un Héritage en C++, nous avons construit dans ce TP une application permettant la création, la gestion et l'affichage de trajets entre deux villes. Suivant un cahier des charges précis, notre priorité a été d'éviter la redondance d'information en choisissant la meilleure architecture des classes possible et en exploitant le principe d'Héritage. Ainsi, le trajet simple, à savoir celui reliant une ville de départ et une ville d'arrivée en utilisant un moyen de transport, est la classe mère du trajet composé, classe permettant de réaliser une liste chaînée de trajets simples. On peut donc utiliser indifféremment les méthodes de ces deux classes dans la classe Catalogue grâce au polymorphisme. La classe Catalogue permet de stocker, de rechercher ou de créer de nouveaux trajets.

De manière plus précise, notre application s'articule autour de quatre classes différentes :
- **une classe Catalogue** qui est la classe centrale de l'application. Instanciée dans *MenuCatalogue.cpp*, elle permet de créer de nouveaux trajets simples ou composés, de rechercher des trajets particuliers via les villes de départ et d'arrivée voulues, ou encore d'afficher l'ensemble des trajets répertoriés. L'ensemble de ces actions est exécuté à la suite d'une commande utilisateur, ici simplement réduite à un nombre allant de 0 à 5.
- **une classe File** qui n'est autre que l'implémentation d'une liste chaînée de type *First-in First-out*. Utilisée par la classe Catalogue, elle permet stocker les trajets insérés afin de pouvoir réaliser les opérations de recherche par la suite.
- **une classe TrajetSimple** ayant pour attribut une ville de départ, une ville d'arrivée et un moyen de transport utilisé pour relier ces deux villes. Cette classe étant la classe mère de la classe représentant les trajets composés, la plupart de ces méthodes sont *virtual* et seront redéfinies dans la classe fille. Les attributs étant protégés, elle possède aussi différents getters pour accéder à ces derniers depuis d'autres classes n'ayant pas le droit d'accès.
- **une classe TrajetComp** héritant de la classe TrajetSimple et possédant comme attribut supplémentaire un pointeur sur un élément de type TrajetComp. En effet, ce type de trajet n'étant rien d'autre que plusieurs trajets simples de suite, ce pointeur permet d'indiquer quel TrajetComp suit celui actuellement traité dans ce trajet composé. Cette classe possède un setter permettant de fixer son pointeur à la création du trajet.

Le graphe d'héritage Figure 1 représente les liens entre les objets manipulés ainsi que la répartition de leurs caractéristiques.

Tous les attributs généraux d'un trajet (ville de départ, ville d'arrivée, moyen de transport) font parties de la classe TrajetSimple décrivant un trajet simple d'une ville de départ à une ville d'arrivée en utilisant un moyen de transport.

Le deuxième type de trajet est un trajet composé, autrement dit une succession de trajets simples. D'un point de vue programmation et gestion des classes, un trajet composé est un ensemble TrajetSimple + Pointeur, permettant de connaître le trajet composé suivant. Il a ainsi fallu créer une classe TrajetComp héritant de TrajetSimple possédant en plus un attribut pointeur de TrajetComp.

La classe Catalogue permet de stocker plusieurs instances de TrajetComp et TrajetSimple dans une liste chaînée. Cette liste est liée à Catalogue grâce à deux attributs de type pointeurs, un de début de liste et

l'autre de fin. Un troisième attribut tailleCatalogue permet de gérer l'ensemble des données regroupées dans le catalogue de façon optimale.

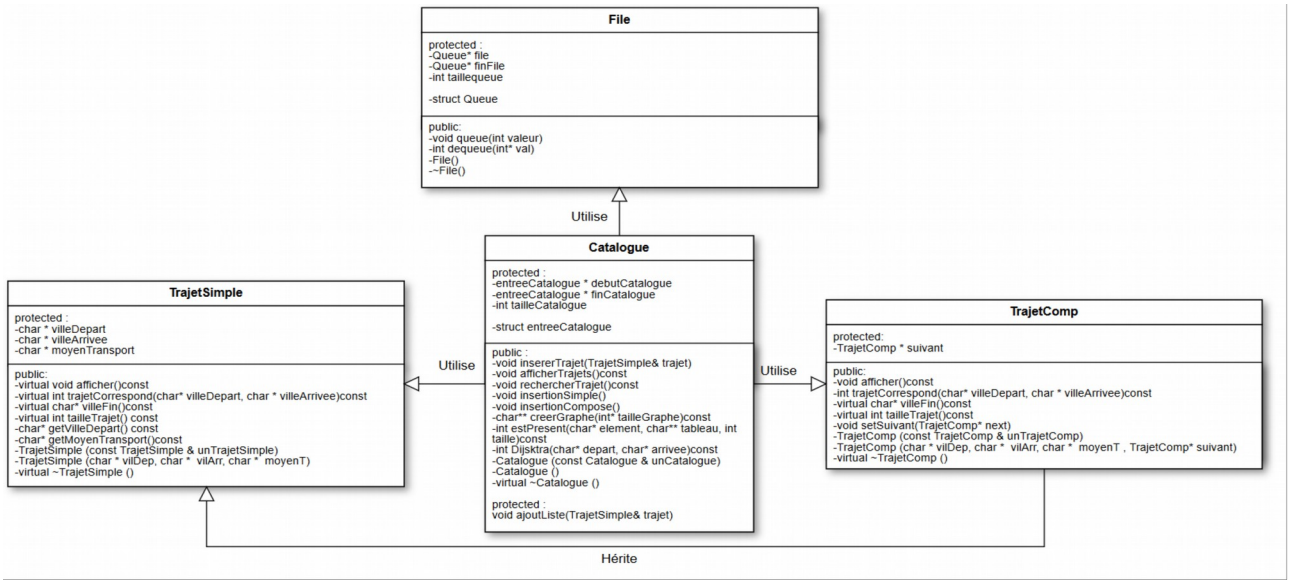

Figure 1 : Graphe d'héritage de l'application

II. Description de la structure de données

Le Catalogue est composé d'une liste d'entreeCatalogue qui permettent d'avoir accès au contenu d'un trajet via un pointeur trajet de type trajetSimple*. Le trajet pointé peut être soit un TrajetSimple, soit un TrajetComp, sachant le TrajetComp pointé est le 1er trajet d'une liste de TrajetComp constituant un trajet composé.

Voir StructMem.pdf

III. Listing des classes

L'application utilise les classes suivantes :

- Catalogue.h, Catalogue.cpp
- TrajetSimple.h, TrajetSimple.cpp
- TrajetComp.h, TrajetComp.cpp
- File.h, File.cpp

Le rôle de chacune des classes et son fonctionnement général est directement expliqué dans les entêtes de classe. Le mode d'emploi de chaque méthode et les explications nécessaires à son utilisation sont décrits dans le code.
Il est possible d'activer le traçage des appels aux constructeurs/destructeurs et méthodes à partir du fichier Makefile : DEFINEMAP= -DMAP

IV. Conclusion

Lors de notre travail sur ce TP, nous avons fait face à différents problèmes :

- Temps pris pour mettre en commun le travail effectué
- Difficulté de la recherche complexe
- Impossibilité d'utiliser des librairies simplifiant l'écriture comme par exemple String et List ou vector

Nous pouvons voir différents axes d'évolution et d'amélioration de l'application :

- Optimiser les algorithmes de recherche : ceux-ci ont pour le moment une complexité importante et ne seraient pas adaptés à des graphes très grands.

- Résoudre le problème de boucle infinie pour la recherche avancée (par ex pour aller de a vers e avec les trajets a→b, b→ c, c→ d, d→ a ; ou aller de a vers d avec les trajets a→b, b→ c, c→ a, d→ a)

- Vérification de saisie utilisateur → chiffres au menu et lettres pour les villes et moyens de transport
  → empêcher la saisie une ville de départ égale a celle d'arrivée
  → rendre possible la saisie de ville avec un nom composé (avec espaces)

- Ajouter la possibilité de supprimer un trajet ajouté en fonction de la ville de départ, de la ville d'arrivée et du moyen de transport

- Ajouter des poids aux trajets (prix, distance, emprunte carbone…) et trier les trajets en fonction de ces poids