B3405
GENTIL Grégoire
DELAHAYE Etienne

COMPTE RENDU DU TP02 (Héritage et Polymorphisme) − C++ − POO1

A − Description détaillée des classes

Vous trouverez ci-dessous une description des classes de notre projet. Le texte explique leur sémantique.

Sur le graphe d'héritage (*cf annexe*) vous verrez la répartition des attributs et des méthodes. Ce choix est justifié en dessous de la description des classes.

Classe Trajet :

Cette classe est une classe purement abstraite. Elle indique les méthodes qu'un objet de type Trajet (et de ses dérivés) doit implémenter.
Elle représente un trajet (simple ou composé) du catalogue, ainsi le main (fonction d'entrée du programme) ne traite que des objets de type Trajet *.

Classe TrajetSimple :

Un TrajetSimple hérite de la classe Trajet, donc implémente toutes les méthodes déclarées dans Trajet.h. Il représente un trajet simple, c'est à dire un trajet direct sans escale, avec un moyen de transport.

Classe TrajetCompose :

Un TrajetCompose hérite de la classe Trajet. Il représente un trajet avec une ou plusieurs escale(s), et un ou plusieurs moyen(s) de transport différents. À la réalisation, un TrajetCompose est constitué de plusieurs TrajetSimple stockés dans un attribut de type TabTrajet.

Classe TabTrajet :

Un TabTrajet est une implémentation d'un tableau dynamique de pointeurs de Trajet. Il permet les insertions et les récupérations de Trajet *. Il se réalloue automatiquement si le nombre d'objets stockés devient trop grand pour sa capacité.

Classe Catalogue :

Un catalogue représente un objet stockant tous les trajets entrés par l'utilisateur. Il permet aussi la recherche de parcours et l'affichage de tous les trajets stockés. Ceux-ci sont stockés dans un attributs de type TabTrajet *.

Justification de ce choix :

Faire hériter nos classes TrajetSimple et TrajetCompose de la classe abstraite Trajet nous permet de manipuler ces objets − de types différents − de la même manière. Un TrajetCompose étant implémenté sous forme de TabTrajet, il est donc inutile de stocker les charactéristiques (départ, arrivée et moyen de transport) dans la classe TrajetCompose; c'est pourquoi nous avons placé ces attributs dans la classe TrajetSimple plutôt que dans Trajet.

Les objets de type Catalogue n'intéragissent avec le main et avec les TabTrajet qu'à travers des pointeurs de Trajet. Le polymorphisme nous permet donc de traiter n'importe quel Trajet − qu'il soit simple ou composé − de la même façon.

Aussi, ce polymorphisme nous permet de gérer la collection de Trajet (simple ou composé) propre à Catalogue de la même manière que nous gérons la collection ordonnée de TrajetSimple propre à TrajetCompose. Notre TabTrajet ne faisant pas de différence entre un TrajetSimple et un TrajetCompose, nous avons pu l'utiliser à deux reprises, nous simplifiant ainsi la gestion des tableaux et nous permettant de factoriser le code.

B − Description détaillée de la structure de données

cf annexe pour le dessin

La classe TabTrajet a pour rôle la gestion d'une collection ordonnée de trajets. Nous avons choisi comme structure, un tableau de pointeurs de Trajet, que l'on manipule en stockant sa taille maximale ainsi que sa taille utilisée. Les pointeurs étant de type Trajet*, les trajets simples et composés sont donc gérés de la même manière.

La classe dispose de deux méthodes permettant d'accéder à sa taille utilisée ainsi qu'à un trajet qu'elle contient par indice. Elle dispose d'une méthode d'ajout qui permet d'ajouter un trajet au tableau. Lorsque celui-ci est plein, une méthode de réallocation est automatiquement appelée. Cette dernière copie le tableau actuel dans un nouveau tableau de taille double.

C − Code

cf annexe

D − Conclusion

Les options de compilation, notamment "-ansi", "-pedantic" et "-Wall" nous ont permis de corriger les erreurs de syntaxes assez rapidement. Nous avons eu quelques soucis avec les "const", mais puisqu'à chaque fois c'était le compilateur qui nous prévenait, nous avons pu trouver les problèmes en peu de temps; les résoudre en a pris un peu plus.

En revanche, les "Segmentation fault" lors de l'exécution du programme sont plus dures à identifier et à corriger. L'utilisation d'un debugger (gdb) a alors été nécessaire. Cela a été la partie la plus longue du débogage. Certaines segfault venaient d'une non-initialisation des pointeurs, d'autres de fonctions défaillantes.

Le passage par référence d'objets statiques nous a aussi posé quelques soucis, que l'on a réglé en allouant un espace pour chaque objet persistant dans le tas. Cela a aussi permis de gérer plus facilement la destruction des objets construits par le passé.

En terme d'amélioration, nous pourrions améliorer légèrement notre algorithme de recherche avancée, en particulier les procédures pour afficher une combinaison de parcours (voir Catalogue::max et Catalogue::AfficherParcours). Cela reste négligeable face à la complexité de la récursivité.