TP C++ n°2 : Héritage – Polymorphisme

Constitution de voyages à partir d'un catalogue de trajets

1. Introduction

L'objectif de ce TP est de vous familiariser avec les concepts et mécanismes de base des langages à objet (classes, héritage, polymorphisme...) et avec leur expression en langage C++. Le point le plus intéressant du TP est la **manipulation d'un objet correspondant à une collection d'objets hétérogènes** (c'est-à-dire de types différents) **qu'il faut gérer sans distinction de leur nature en exploitant le polymorphisme et la liaison dynamique** (cf. les cours et TD). L'application proposée est un simple support et ne prétend pas être ni réaliste, ni complète.

Ce deuxième TP d'initiation au langage C++ reprend également les ingrédients du premier TP – pointeur et classe dynamique – en les accentuant. Il exige une extrême vigilance sur la gestion de la mémoire en particulier pour la structure de données principale associée à votre application. Ce TP met aussi en œuvre les principes élémentaires de développement d'un logiciel avec la notion d'interface et de réalisation de classe (application du Guide de Style C++ proposé dans le cours). Enfin, il constitue une belle application à l'écriture d'un fichier *makefile* paramétré (variables) et utilisant un *pattern*.

Il s'appuie exclusivement sur les concepts abordés dans le cadre du module IF-3-POO1 même si l'usage de classes issues de la STL ou l'emploi de la généricité s'imposeraient !

2. Problème

A partir d'un catalogue de **trajets**, il s'agit de proposer des parcours pour un voyage défini par une ville de départ et une ville d'arrivée. Un parcours est formé d'une suite de trajets et un voyage de **`A`** à **`B`** est choisi parmi tous les parcours permettant à partir de la ville **`A`** de rejoindre la ville **`B`**.

Les **trajets du catalogue** sont soit des **trajets simples** d'une ville à une autre, soit des **trajets composés**. Un **trajet composé** est une **collection ordonnée de trajets simples ou composés** tels que la ville d'arrivée d'un trajet est obligatoirement la ville de départ du trajet suivant dans la collection ordonnée. Le premier trajet de la collection ordonnée donne la ville de départ du trajet composé, tandis que sa ville d'arrivée est la ville d'arrivée du dernier trajet de la collection ordonnée. Un trajet composé est **à prendre en totalité** (même si la ville destination recherchée est une escale…). Chaque trajet est caractérisé par un moyen de transport et pour un trajet composé, chaque trajet simple doit avoir son propre mode de transport.

3. Cahier des charges – Définition du problème

L'application comporte plusieurs fonctionnalités essentielles :

1. La **construction du catalogue**, c'est-à-dire l'ajout de trajets au catalogue courant : pour réaliser cette tâche, il faut être capable de saisir de nouveaux trajets qui pourront être simples ou composés ;
2. L'**affichage du catalogue** courant : à tout instant, il faut être en mesure d'afficher le catalogue courant (affichage d'objets hétérogènes – trajets simples ou composés). Le rôle principal de cet affichage est de permettre la mise au point et la validation de votre travail ;
3. La **recherche de parcours** dans le catalogue courant : pour un voyage donné, défini par une ville de départ et d'arrivée, il faut retrouver dans le catalogue courant tous les parcours qui peuvent répondre à la demande. Cette recherche de parcours pourra s'envisager en 2 étapes.
    a. Pour la première étape (version simple), on recherchera uniquement des parcours constitués d'un trajet simple ou d'un trajet composé sans envisager de composition de trajets. Cette recherche devra être opérationnelle pour tous les binômes ;
    b. Dans un deuxième temps (version avancée), on combinera des trajets simples et/ou composés pour construire les parcours Cette recherche est beaucoup plus difficile à réaliser et à mettre au point. Elle constitue un beau *challenge* mais elle peut exiger un long temps de développement…

Exemple : Contenu du catalogue de trajets (simples ou composés) :

```
1 : TS1 = de A à B en MT1
2 : TC1 = de B à Y en MT3 – de Y à C en MT2
3 : TS2 = de B à C en MT1
4 : TC2 = de A à Z en MT2 – de Z à C en MT1
```

Voyage recherché : `de A à C`

Version simple : `1` seul parcours possible : `(TC2)`

Version avancée : `3` parcours possibles : `(TS1+TC1), (TS1+TS2), (TC2)`

Note : le nommage des trajets (`TS1`, `TC1`…) **ne doit pas être envisagé dans votre solution**. Il est présent ici pour expliquer les 2 versions de recherche.

Pour enchaîner ces fonctionnalités, on utilisera un simple menu avec une dernière option de sortie de l'application. **Dans le cadre de ce TP, l'aspect interface utilisateur est clairement secondaire**. On pourra, par exemple, saisir un chiffre pour effectuer le choix entre les différentes options du menu. Il faudra se contenter de cette saisie très rustique et **faire les choses simplement** au niveau de l'interface. En effet, l'objectif du TP est clairement fixé sur l'héritage et sa sémantique et la gestion dynamique de la mémoire avec de très nombreuses manipulations de pointeurs. En revanche, une interface ergonomique et facile à utiliser est demandée : pensez à l'utilisateur qui doit tout recommencer s'il se trompe ou qui doit deviner ce qu'il doit faire dans le menu.

À la réalisation, **pour simplifier la saisie**, **un trajet composé ne contient que des trajets simples**. Autrement dit, la saisie d'un trajet composé doit vérifier immédiatement la contrainte sur les villes de départ et d'arrivée des trajets simples consécutifs de façon à **ne stocker dans le catalogue que des trajets valides**. C'est une **exigence de réalisation et non pas de spécification / conception de la solution**.
Il faudra également réfléchir à l'architecture générale de votre application et veiller à un **découpage de cette application en classes sémantiquement homogènes**, en évitant les redondances d'information. Le point le plus intéressant est le catalogue qui est une collection ordonnée d'objets hétérogènes (les trajets qui peuvent être simples ou composés) qu'il faut impérativement gérer sans distinction de leur nature.

4. Réalisation

Par rapport au premier TP C++, la gestion dynamique de la mémoire est plus complexe et l'algorithmie est plus pointue (surtout avec la recherche avancée). Clairement, le TP est plus difficile. Aussi, il faudra faire preuve de **beaucoup de rigueur et de méthode** (faire des petits pas !) pour arriver sans trop de déboires à la fin de la réalisation.
Rappel : Lorsqu'on manipule un pointeur, il y a le pointeur et il y a l'objet pointé ! L'usage du mot clé `const` sur les paramètres des méthodes (surtout avec les pointeurs) mais également sur les méthodes des classes qui ne modifient pas leurs attributs peut s'avérer d'une grande utilité (surveillance de la part du compilateur).

Pour la compilation de votre code C++, il est impératif d'utiliser, **au minimum**, les options de compilation suivantes pour le compilateur `g++` de la plate-forme Linux : `-ansi -pedantic -Wall -std=c++11`.

Attention : contrainte de réalisation

L'utilisation d'une fonction issue d'une quelconque bibliothèque standard du C / C++ est **rigoureusement interdite**, hormis 2 bibliothèques :

1. La bibliothèque de manipulation des flux d'entrées / sorties **iostream** (**iostream** ne doit servir qu'à faire des **cin**, **cout** ou **cerr** !) ;
2. La bibliothèque de manipulation des chaînes de caractères **cstring** pour une utilisation de **strlen**, **strcpy**… (cf. TD IF-3-POO1 Pointeur et Mémoire).

Note :

Comme pour le premier TP C++, la bonne gestion de la mémoire dynamique sera validée par l'outil **valgrind**. (cf. TP IF-3-OP-1). **Au fur et à mesure de l'avancée de votre réalisation**, il faudra s'assurer de l'absence de fuite de mémoire et de la bonne gestion des pointeurs. Une compilation de votre code en mode débug pourra s'avérer d'une grande utilité.

5. Récupération des informations

Pour réaliser ce TP, il faut utiliser les squelettes de classe présentés et utilisés lors des cours magistraux et du premier TP C++. Par conséquent, vous réutiliserez les squelettes que vous avez déjà copiés lors de ce premier TP.

Rappel :

Les squelettes sont disponibles sous Moodle dans la partie relative au TP C++ 1 ainsi que dans le répertoire :

/shares/public/tp/TP1C++-POO1/SQUELETTES

6. Livrables

Le livrable final a pour échéance le 11 décembre 2023 à 14h et sera constitué de 3 éléments :

1) Le code source commenté, accompagné du fichier makefile (**inutile de fournir les fichiers compilés, bien évidemment**) permettant à l'équipe enseignante de compiler, exécuter et tester votre application,
2) Un document écrit au format pdf (1 page),
3) Une démonstration ou un test de l'application par l'équipe enseignante

Au plus tard le 11 décembre 2023 14h dernier délai, chaque groupe déposera son travail sous moodle, sous la forme d'un fichier zip contenant le document écrit au format pdf et le code source de l'application (sources et makefile. Ne pas intégrer les fichiers compilés). Ce fichier zip devra être nommé avec les numéros de binômes et les noms des étudiant(e)s sous la forme : « POO1-TP2-numéro du binôme 1-numéro du binôme 2-nom1-nom2-nom3-nom4.zip ». Au terme des deux séances de TP, des zones de dépôts seront prévues sous Moodle : une zone par groupe (attention à bien déposer dans la zone correspondant à votre groupe/jours de TP).

Le document au format pdf devra contenir simplement la description simple de toutes les classes de votre application, en particulier le graphe d'héritage avec sa justification (notamment la répartition des caractéristiques des objets). Et bien entendu, **n'oubliez pas de préciser vos noms et numéro de binôme sur votre document**.

Les sources de l'application doivent notamment comporter les classes définies dans votre application (fichiers *interface* `*.h` et *réalisation* `*.cpp`). Dans l'entête de chaque classe, vous expliquerez clairement le rôle de la classe et son fonctionnement général. Vous donnerez le mode d'emploi pour chaque méthode avec les explications nécessaires à son utilisation en décrivant correctement les paramètres formels (cf. les squelettes). Dans la cadre de ce TP, **l'usage et le respect du squelette constitue une obligation stricte**. Le traçage des appels aux constructeurs / destructeur doit être possible avec la constante **`MAP`** et prévu par le fichier **`makefile`**.

L'exécutable créé par le fichier makefile devra se nommer « **trajets** » (donc il suffira de seulement deux commandes pour tester l'application : « make » et « ./trajets »).

La compilation grâce au makefile, l'exécution et le test de chaque application seront réalisés par l'équipe enseignante sur les machines du département soit durant une démonstration, soit par la suite en fonction du temps disponible. A cette occasion, chaque fonctionnalité sera testée, ainsi que des enchaînements de fonctionnalités dans des scénarii préétablis. L'appel des fonctionnalités sera réalisé grâce à votre menu. Comme précisé précédemment, ce dernier doit être simple mais suffisamment clair pour qu'une personne souhaitant utiliser votre application puisse comprendre tout de suite quelle fonctionnalité sera exécutée par chaque option).