Auteurs : Hugo BLAESS - Octave DUVIVIER
Binôme : B3204

Compte Rendu TP 2 POO1

I. Description simple des classes

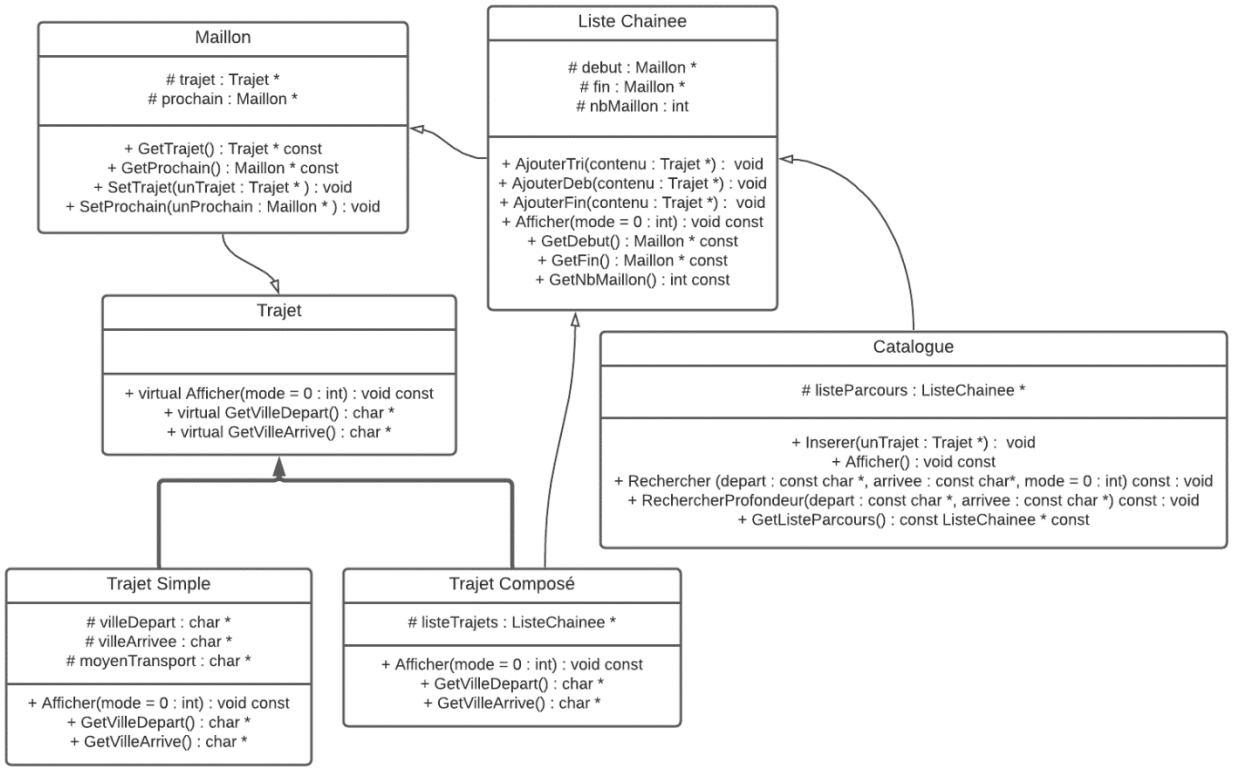

Figure 1 : Diagramme de classes

Pour notre application nous avons choisis d'utiliser les classes suivantes :

- Trajet : classe abstraite dont dérivent les classes Trajet Simple et Trajet Composé. Ses méthodes virtuelles seront définies dans chaque classe fille afin d'avoir un comportement différent en fonction du type de l'appelant. Nous avons fait le choix de ne mettre aucun attribut dans cette classe car ils sont propres aux classes filles.
- Trajet Simple : classe dérivée de Trajet par héritage public. Elle permet de créer un Trajet d'une ville à une autre via un moyen de transport.
- Trajet Compose : classe dérivée de Trajet par héritage public. Elle permet de créer un Trajet d'une ville à une autre par enchainement de trajets simples. Ceux si sont

Auteurs : Hugo BLAESS - Octave DUVIVIER
Binôme : B3204

renseignée dans une liste chainée.

- ❖ Maillon : Elément constitutif de liste chainée permettant de pointer vers un trajet et vers le maillon suivant.
- ❖ Liste chainée : Implémentation d'une liste chainée de trajets avec pointeur de début et de fin.
- ❖ Catalogue : classe qui contient les différents trajets et qui permet de les manipuler.

II. Description de la structure de donnée

Pour gérer la collection ordonnée de trajets nous avons choisis d'implémenter une liste chainée. En effet celle-ci nous paraissait être la plus simple à utiliser pour les opérations d'insertion et de parcours que nous allions effectuer. En utilisant des Maillons pointant sur un type trajet cela nous a permis d'insérer des Trajets Simple tout comme des Trajets Compose à l'intérieur de la liste. Cela nous a également permis via les méthodes virtuelles pure de Trajet de faire des appels sur les éléments de la liste tout en gardant les comportements spécifiques des classes TS ou TC.

Chaque Maillon de la liste pointe sur le trajet qu'il contient et sur son Maillon suivant. Nous avons choisi d'implémenter une liste avec pointeur vers le premier et le dernier Maillon pour faciliter les insertions en queue ainsi que les recherches des villes de départ et d'arrivée pour les trajets composés.

Nous avons fait le choix d'implémenter deux méthodes d'insertion. Une méthode d'insertion en queue, que nous utiliserons pour insérer des Trajets Simple au sein d'un trajet compose et une méthode d'insertion trié par ordre alphabétique que nous utiliserons pour ajouter un Trajet au catalogue. La méthode d'ajout en début n'est pour l'instant utiliser que pour l'ajout trié.

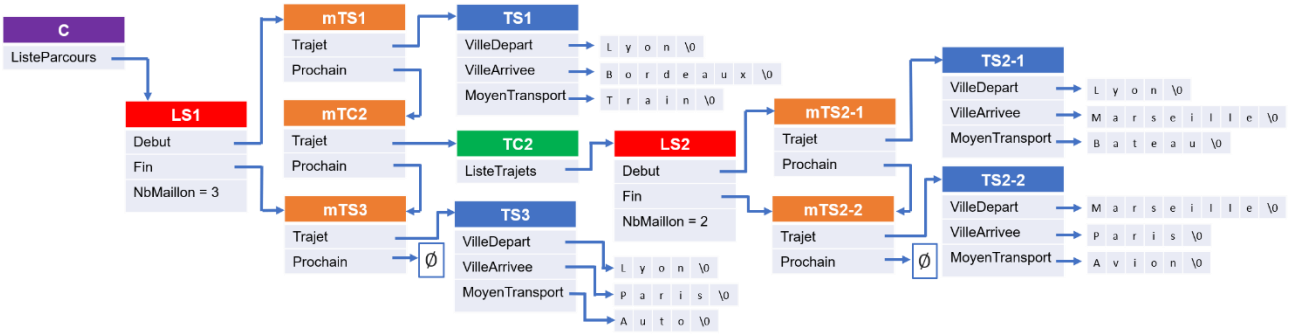

Figure 2 : Schéma mémoire de la structure de données

Auteurs : Hugo BLAESS - Octave DUVIVIER
Binôme : B3204

III. Réalisation

Durant la réalisation de notre projet nous avons rencontré de nombreux problèmes. L'utilisation des éditeurs de code nous a permis d'éviter beaucoup d'erreur en nous prévenant avant le preprocessing, la compilation et l'édition des liens de certaines d'entre elles.
Les fuites mémoires ainsi que les lectures et écritures en dehors de la mémoire accessible on dut être régler avec d'autre outils. Pour se faire nous avons utilisé valgrind et la trace de notre squelette ainsi que l'option de compilation -g qui nous a permis de compiler tout en gardant des informations de débugage dans l'exécutable.

```
oduvivi@LAPTOP-R4FCIDSJ:/mnt/d/Desktop/C++/Projet/master$ make MAP -B
g++ -c -g -ansi -pedantic -Wall -std=c++11 -DMAP Catalogue.cpp
g++ -c -g -ansi -pedantic -Wall -std=c++11 -DMAP ListeChainee.cpp
g++ -c -g -ansi -pedantic -Wall -std=c++11 -DMAP Maillon.cpp
g++ -c -g -ansi -pedantic -Wall -std=c++11 -DMAP Trajet.cpp
g++ -c -g -ansi -pedantic -Wall -std=c++11 -DMAP TrajetSimple.cpp
g++ -c -g -ansi -pedantic -Wall -std=c++11 -DMAP TrajetCompose.cpp
g++ -c -g -ansi -pedantic -Wall -std=c++11 -DMAP main.cpp
g++ -o Projet_Jours Catalogue.o ListeChainee.o Maillon.o Trajet.o TrajetSimple.o TrajetCompose.o
main.o -ansi -pedantic -Wall -std=c++11 -DMAP
```

Figure 3 : Compilation avec le debuger et la trace

```
Appel a la méthode RechercherProfondeur de <Catalogue> pour A
==74== Invalid write of size 8
==74==    at 0x10A79C: Catalogue::RechercherProfondeur(char const*, char const*) const (Cata
logue.cpp:140)
==74==    by 0x10A567: Catalogue::Rechercher(char const*, char const*, int) const (Catalogue
.cpp:83)
==74==    by 0x10C98D: Menu(Catalogue*) (main.cpp:139)
==74==    by 0x10C240: main (main.cpp:19)
```

```
total heap usage: 97 allocs, 97 frees, 75,999 bytes allocated
```

Figure 4 et 5 : Utilisation de valgrind

La réalisation de la recherche avancée à elle aussi était une épreuve. Nous avons réussi à gérer la mémoire sans trop de difficultés mais l'algorithme à était plus compliqué à mettre en place. Il semblait assez clair que pour réaliser cette recherche nous allions utiliser un arbre ayant pour sommet les différentes villes et comme liaisons le trajet les reliant.
On commençait la création de notre arbre depuis la ville de départ puis on descendait dans l'arbre en recherchant récursivement les destinations atteignables dans le catalogue de chacune des destinations respectives.
Pour éviter les boucles infinies il était nécessaire de marquer les villes par lesquels nous étions déjà passer. En construisant un tableau catégorisant la ville ainsi que celle par laquelle on était arrivée. Cela me permettait de connaitre les sommets déjà visiter et également de garder une trace du chemin parcouru, ce qui nous a était utile lors de

Auteurs : Hugo BLAESS - Octave DUVIVIER
Binôme : B3204

l'affichage.

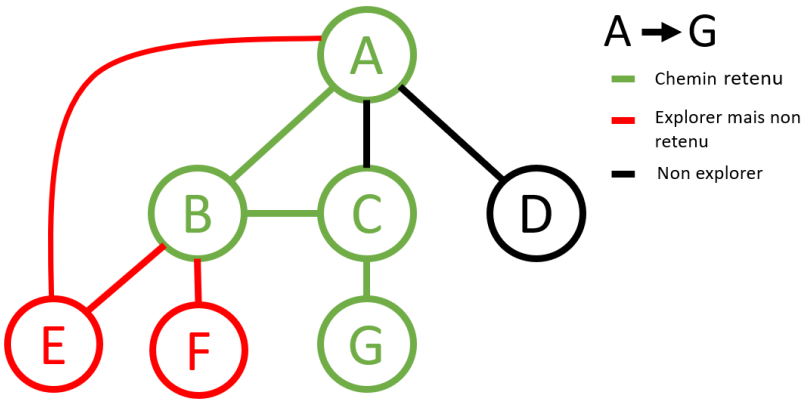

Figure 6 : Arbre de recherche en profondeur

IV. Conclusion

Notre projet est bien sur très limité et possèdent encore bien des axes d'améliorations.

L'impossibilité de supprimer ou de modifier des trajets du catalogue rend le catalogue difficilement utilisable. Ces fonctionnalités pourraient être des méthodes ajouter en vue d'une amélioration de l'application.

L'IHM est également très sommaire, bien que cela ne fasse pas vraiment partie du projet. L'affichage très simpliste rend l'application difficile d'utilisation.

En revenant sur certaine fonction je me suis rendu compte qu'elle n'était pas très optimisée. La structure générale du code pourrait être améliorer.

L'utilisation des getteur et setteur pourrait être optimisé afin d'optimiser la qualité du code et de préserver la protection des variables, notamment au niveau des const.

La recherche avancée pourrait être améliorer de plusieurs manières. Tout d'abord la combinaison des trajets à effectuer devrait s'afficher dans le bon sens (de haut en bas) afin d'en faciliter la compréhension. Comme montrer dans la figure 6 l'algorithme actuelle ne permet d'afficher qu'une solution, il serait intéressant de le développer pour afficher le trajet le plus court ou l'ensemble des trajets réalisables. En therme d'optimisation nous pourrions utiliser un autre algorithme que celui de parcours en profondeur nous pourrions utiliser l'algorithme de parcours en largeur qui nous éviterai de partir dans une branche sans savoir où celle-ci nous mène.