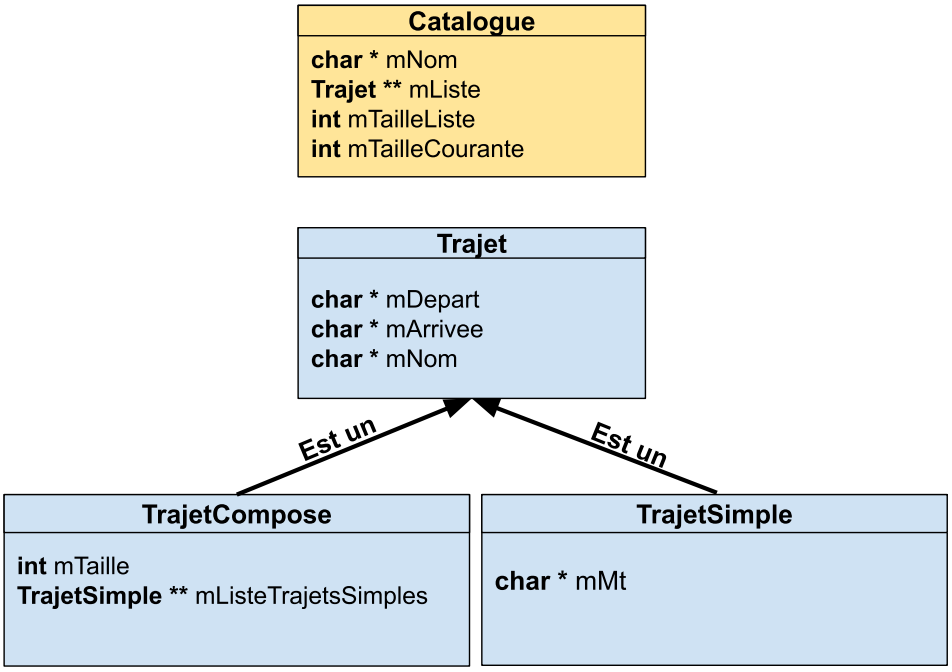
Compte Rendu : TP C++ n°2 - Héritage et Polymorphisme

Binôme :B3110
Noms des étudiants : IICH-LEMSEFFER-ERABHAOUI
Date : 17 Décembre 2024

I. Graphe d'héritage et description des classes :

1. Graphe d'héritage :



Chaque trajet, peu importe son type, possède un départ, une arrivée et un nom. Ces attributs doivent être dans la classe Trajet. De plus, cette répartition facilite les algorithmes de recherche, car on ne s'arrête pas dans les étapes intermédiaires. On ne regarde donc que le départ et l'arrivée d'un trajet, même composé.

2. Description des classes :

a) Trajet (classe de base) : C'est une classe abstraite représentant un trajet.

Attributs protected :
char * mDepart : Ville de départ du trajet.
char * mArrivee : Ville d'arrivée du trajet.
char * mNom : Nom du trajet.

Méthodes publiques :
virtual void Afficher(void) const : Méthode virtuelle pure pour l'affichage.
const char * getters : avoir accès aux attributs de trajet à partir du catalogue sans pouvoir les modifier (mot clé **const**).
Des constructeurs/destructeurs : assurant la bonne gestion des pointeurs.

b) **TrajetSimple (dérivée de Trajet par héritage publique) :** Elle représente un trajet simple entre deux villes avec un moyen de transport.

Attributs protected :
`char * mMt` **:** Le moyen de transport utilisé.

Méthodes publiques :
virtual void Afficher(void) const : Méthode virtuelle pour l'affichage.
const char * getters : avoir accès aux attributs de trajet à partir du catalogue sans pouvoir les modifier (mot clé **const**).
Des constructeurs/destructeurs : assurant la bonne gestion des pointeurs.

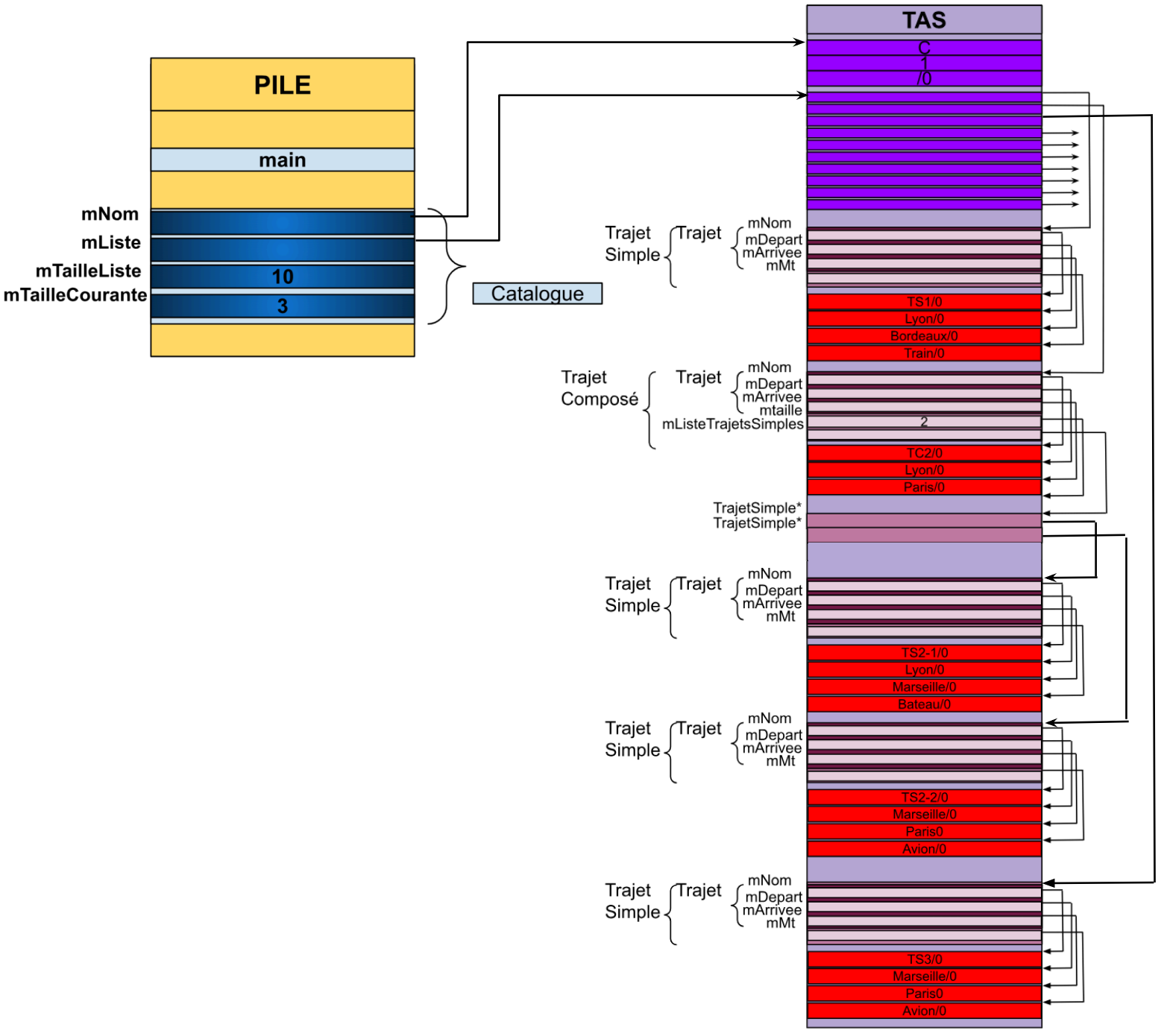
c) **TrajetCompose (dérivée de Trajet par héritage publique) :** Elle représente un trajet composé de plusieurs trajets simples.

Attributs protected :
TrajetSimple ** mListeTrajetsSimples **:** liste de pointeurs vers trajets simples.
int mTaille **:** taille de la liste des pointeurs vers trajets simples.
Méthodes publiques :
virtual void Afficher(void) const : Méthode virtuelle pour l'affichage.
Des constructeurs/destructeurs : assurant la bonne gestion des pointeurs.

d) **Catalogue (classe à part) :** Elle gère la collection ordonnée de trajets (simples ou composés).

Attributs protected :
char * mNom : nom du catalogue.
Trajet ** mListe : Liste des trajets dans le catalogue.
int mTailleListe : Taille de la liste des trajets.
int TailleCourante : Remplissage courant du catalogue.
Méthodes protected :
void dfs (const char * current, const char * depart, const char * arrivee,
bool * visited, const char * villes[ ], int & nbVilles, Trajet * trajetsParcourus[ ], int & nbTrajets) const : Explore les trajets entre villes avec l'algorithme DFS.
bool villeDejaRencontree (const char * ville, const char * villes[ ], int nbVilles) const : Vérifie si une ville existe déjà.
void ajouterVille (const char * ville, const char * villes[ ], int & nbVilles) const : Ajoute une ville si elle n'y est pas encore.
Méthodes publiques :
void Afficher(void) const : Cette méthode parcourt la liste des trajets et les affiche un par un.
void Redimensionner(void) : Cette méthode redimensionne la liste des trajets de catalogue si notre catalogue est plein.
void RechercheAvancee(const char *depart, const char *arrivee) const : Cette méthode permet d'effectuer une recherche avancée avec le principe de transitivité multiple.
void RechercheSimple(const char *depart, const char *arrivee) const :Cette méthode effectue une recherche simple en vérifiant les correspondances exactes entre le départ et l'arrivée.
void AjouterTrajet(Trajet * trajet) : Cette méthode ajoute un objet Trajet à la liste des trajets du catalogue.
Des constructeurs/destructeurs : assurant la bonne gestion des pointeurs.

II. Dessin de la mémoire :



⚠ **En rouge dans le tas, ce n'est une vraie représentation de la mémoire parce que dans chaque bloc on met le mot en entier, la bonne représentation est de mettre chaque caractère de chaque mot fini par un nullterminator \0, comme pour le nom du catalogue**

III . Problèmes rencontrés et améliorations :

Problème 1 :

Pour chaque trajet, on devait saisir un nom. Du coup, le menu est devenu trop chargé. Pour résoudre ce problème, on a implémenté dans le “main.cpp” une fonction pour convertir les int en char* et donc automatiser artificiellement le nommage à l’aide de la fonction strcat.

Problème 2 :

Dans la partie Recherche Avancée, on a trouvé un peu de difficulté pour implémenter l’algorithme de recherche en profondeur (DFS) sans la bibliothèque standard. On est parvenu à contourner le problème en subdivisant le problème en sous problèmes qu’on a traité chacun individuellement.

Problème 3 :

C'était inconfortable de gérer la mémoire dynamique et d’assurer l’allocation et la libération correcte des objets donc il fallait utiliser régulièrement valgrind pour s’assurer d’aucune perte de mémoire. Pour ce faire, dans le makefile, on a créé la cible “valgrind”, il suffit donc d'exécuter dans le terminal la commande “make valgrind” pour faire un test.

Problème 4 :

Au début, on faisait des get dont le retour était des pointeurs non constants. On s’est rendu compte qu’il fallait ajouter const a chaque pointeur, pour ne pas donner d’accès aux attributs “protected” en dehors de la classe.

Problème 5 :

Dans le menu, on ne peut pas entrer un nom avec un espace. Ainsi, il est obligatoire de mettre des tirets pour des villes au nom composé, comme par exemple “Le_Creusot”.

Problème 6 :

Pour activer l’option MAP, on devait recompiler tous les fichiers source à la main. On a décidé d’ajouter l’option dans le makefile. Il suffit d’écrire dans le terminal : make MAP=1

Améliorations :

Actuellement, on est obligés de parcourir les trajets composés en entier, ce qui n’est pas forcément le meilleur choix. Pour améliorer notre programme, on pourrait regarder dans la liste des trajets simples de chaque trajet composé.

De même, on pourrait implémenter une méthode pour supprimer un trajet du catalogue.