COMPTE RENDU

TP C++ : Gestion des entrées/sorties

SOMMAIRE

[I. Introduction 1](#_Toc534501958)

[II. Description du format de notre fichier : 2](#_Toc534501959)

[III. Spécification des nouvelles fonctionnalités 2](#_Toc534501960)

[III.1. La sauvegarde : 3](#_Toc534501961)

[III.2. Le chargement : 4](#_Toc534501962)

[IV. Conclusion 6](#_Toc534501963)

[IV.1. Problèmes : 6](#_Toc534501964)

[IV.2. Améliorations : 6](#_Toc534501965)

1. Introduction

Dans un premier temps, nous devions fournir à un client une mini application fonctionnelle donnant la possibilité à un utilisateur de gérer un catalogue de trajets. Et ce insérant lui-même des trajets simples ou des trajets composés, en effectuant des recherches de trajets…

Aujourd’hui le temps est venu de faire évoluer cette application (Traj’App++ 😊). L’utilisateur peut dorénavant sauvegarder le contenu de son catalogue dans un fichier et le aussi charger le catalogue à partir d’un fichier.

La sauvegarde et le chargement peuvent ainsi se faire selon différents critères :

-Sans critère de sélection (Sauvegarde/chargement total d’un catalogue dans/à partir d’un fichier).

-Selon la ville de départ et/ou d’arrivée.

-Selon le type de trajet (simple ou composé).

-Selon une sélection de trajet (Sauvegarde/chargement d’un nombre précis de trajets se situant dans un intervalle)

Afin d’être capable de répondre à ces besoins, le choix du format de fichier s’avère important.

1. Description du format de notre fichier :

Remarque : Notre choix pour le caractère ‘|’ comme délimiteur se justifié par le fait que c’est un caractère que le n’on obtient pas « directement » au clavier, mais en faisant Alt Gr +6. Ce qui fait que si l’utilisateur entre un nom de ville erroné, il y’a très peu de chances que le caractère en plus soit la barre verticale ‘|’. Contrairement à d’autres caractères qui sont potentiellement le choix par défaut des délimiteurs ( ‘\_’, ’ ;’, ’,’ …)

1. Spécification des nouvelles fonctionnalités

Dès que l’utilisateur exécute le programme, un menu s’affiche à l’écran, lui donnant la possibilité d’effectuer différentes actions.

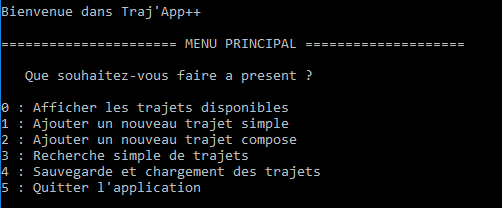


Figure 1 : Menu principal

La nouveauté par rapport au TP précédent est l’**option 4 :** La sauvegarde et le chargement des trajets.

En choisissant cette option, l’utilisateur est redirigé vers un autre menu lui permettant de faire des sauvegardes et des chargements selon les critères différents.

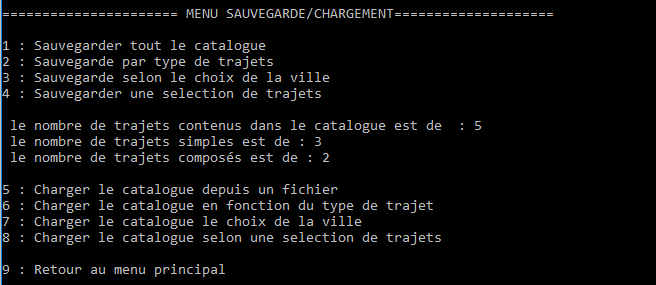


Figure 2 : Menu sauvegarde/chargement

* 1. La sauvegarde :

Pour sauvegarder un fichier dans le dossier dédié « Fichiers\_Saved », nous demandons à l’utilisateur de choisir un nom. On vérifie si le fichier existe déjà ou pas. Si le dossier contient un fichier du même nom ; trois possibilités lui sont présentés :

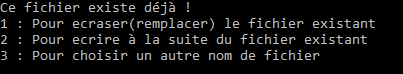


Figure 3 : Menu ouverture du fichier

Il peut ainsi choisir d’écraser le contenu du fichier existant, d’écrire à sa suite ou encore changer de nom pour le fichier. Nous avons implémenté une méthode Ouverture : *void Ouverture(ofstream &fout,string nomFichier) const* qui adapte le mode d’ouverture selon le choix de l’utilisateur.

Une fois le fichier ouvert, on fait appel à la méthode en réponse à la requête de l’utilisateur.

Pour chaque critère, nous avons créé une méthode adéquate. Toutes les méthodes font appel à une fonction polymorphique « Enregistrer » : *virtual void Enregistrer(ofstream & fout) const*

Cette fonction réagit différemment selon si elle est appelée par un trajet simple ou un trajet composé.

Elle a pour mission d’afficher sur le flux de sortie « fout », avec le format spécifié précédemment, les trajets retenus relativement à chaque critère.

-**Sauvegarde total** :

*bool SauvegardeAll(ofstream & fout) const*

Sauvegarde tout le catalogue en parcourant sa collection de trajets.

**-Sauvegarde selon le type des trajets :**

*bool SauvegardeType(ofstream & fout, bool type) const;*

Nous avons créé une fonction *virtual bool EnvoyerType () const* (définie dans le .h ) qui renvoie 0 pour un trajet simple et 1 pour un trajet composé.

Donc dans cette sauvegarde-là, on applique cette fonction à tous les trajets du catalogue et l’on ne retient que ceux correspondants au choix de l’utilisateur. ( Saisie au clavier de 0 ou 1 comme valeur du paramètre d’entrée ‘type’).

-**Sauvegarde selon le choix des villes :**

*bool SauvegardeVD(ofstream & fout, const char\* vd) const;*

*bool SauvegardeVA(ofstream & fout, const char \*va) const;*

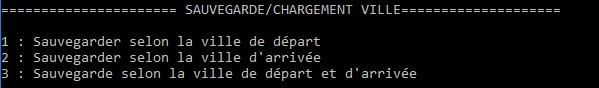
 *bool SauvegardeVDA(ofstream & fout, const char\* vd,const char \*va ) const;*

Figure 4 Menu sauvegarde/Ville

Ces trois fonctions sont très similaires. Chacune d’entre elles, on vérifie avec la fonction ‘strcmp’ que la ville saisie par l’utilisateur est la même que celle présente dans le catalogue.

On ne gère pas les cas où la ville existe dans le catalogue avec une majuscule au début et que l’utilisateur la saisit sans majuscule. Les deux villes doivent être syntaxiquement pareils pour être insérés dans le fichier.

-**Sauvegarde par sélection :**

*bool SauvegardeSelec(ofstream & fout,unsigned const int n,const unsigned int m) const;*

Afin de mettre en place cette méthode, nous avons ajouté à notre classe Catalogue deux attributs ‘private’ : unsigned cptTrajetsSimples, unsigned cptTrajetsComposes

qui sont incrémentés à chaque fois que l’on ajoute un trajet dans le catalogue.

Pour ne sauvegarder que les trajets contenus dans l’intervalle déterminé par l’utilisateur (Gestion de la validité de l’intervalle dans le menu), on parcourt simplement la collection de trajets du catalogue de l’indice n-1 à l’indice m.

* 1. Le chargement :

De même que pour la sauvegarde, nous demandons à l’utilisateur d’entrer le nom du fichier à partir duquel il veut charger son catalogue courant. Le fichier est directement recherché dans le dossier Fichiers\_Saved.

Dans le cas où le fichier n’existe pas, ou est déjà ouvert par l’utilisateur, un message d’erreur est affiché sur le terminal.

Tant que nous sommes pas arrivée à la fin du fichier, on récupère les lignes du fichier. Les informations qui nous intéressent se trouvent entre les délimiteurs ‘|’. Nous avons donc utilisé la fonction ‘getline()’, qui prend en paramètre un flux d’entrée, un string et un char (délimiteur), pour accéder aux attributs qui nous permettront la constitution des trajets. Pour créer un trajet on fait appel aux méthodes suivantes :

*TrajetSimple\* creerTrajetSimple(std::ifstream &fichierEntree);*

Cette méthode retourne un pointeur sur TrajetSimple. Sa constitution se fait par appel au constructeur qui prend en paramètre les données récupérés du fichier.

*TrajetCompose\* creerTrajetCompose(std::ifstream &fichierEntree);*

Cette méthode récupère les caractéristiques d’un trajet composé, en créant les différents trajets simples le constituant ( Appel à creerTrajetSimple())

L’allocation dynamique du Trajet Composé retourné se fait une seule fois grâce au premier trajet simple le formant. Puis les autres trajets simples sont ajoutés au fur et à mesure du parcours de la ligne.

**-Chargement total :**

bool ChargementAll(std::string chemin);

On parcourt tout le fichier et on insère directement dans la collection de trajets du catalogue.

Remarque : Pour chaque ajout de trajet dans le catalogue on veillera à incrémenter les attributs

cbTrajetsSimples et cbTrajetComposes.

**-Chargement selon le type de trajet :**

bool ChargementType(std::string chemin,bool type);

Pour chaque ligne, la création de trajets dépend seulement de la première chaine de caractère récupérée « TS » ou « TC ».

**-Chargement selon le choix des villes :**

bool ChargementVD(std::string chemin,const char\* vd);

bool ChargementVA(std::string chemin,const char\* va);

bool ChargementVDA(std::string chemin,const char\* vd,const char \*va);

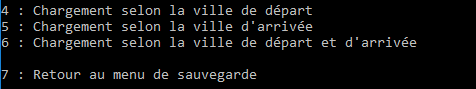


Figure 5 : Menu chargement/Ville

Ces trois méthodes sont gérées de la même manière. Tous les trajets contenus dans le fichier sont créés. Ceux qui ne vérifient la condition (strcmp des villes), et qui ne sont donc pas inséré dans le catalogue sont libérés de la mémoire ‘delete’.

**-Chargement par sélection :**

bool ChargementSelection(std::string chemin, const unsigned int n,const unsigned int m);

On commence par parcourir tout le catalogue, en incrémentant une variable locale à chaque ligne parcourue.

On ne commence à ajouter les trajets au catalogue que lorsque la variable locale est égale à ‘n’ et on sort de la méthode lors cette variable locale à ‘m’. De même que pour la sauvegarde, la gestion de la validité est faite directement dans le menu. La méthode n’est exécuté que lorsque l’intervalle [n,m] est mathématiquement valide.

1. Conclusion
   1. Problèmes :
   2. Améliorations :

* Pour les critères de sélection relatifs aux choix des villes, on stocke dans la mémoire tout le contenu du fichier puis on supprime ceux qui ne nous intéressent pas. Ce qui induit un temps plus long d’exécution. Il serait plus judicieux, de trouver un moyen de comparer les villes de départ et/ou d’arrivée avant de créer le trajet.
* Nous effectuons la comparaison du nom des villes caractère pas caractère. Dans le cas, par exemple où un utilisateur veut sauvegarder les trajets à destination de « lyon » alors que dans le catalogue nous avons des trajets à destination de « Lyon » (Première lettre en majuscule), aucun trajet ne sera sauvegardé ! Alors que l’on ne se doute pas que dans son intention il veut aller à la ville de « Lyon/lyon ». On aurait pu, avant de faire la comparaison, transformer les deux noms en majuscules. On se retrouvera alors à comparer « LYON » et « LYON » et donc à sauvegarder un fichier non vide.
* Plus d’interactions avec l’utilisateur, suivi complet de ses actions : Par exemple, pour la sauvegarde, lui transmettre un message explicitant le nombre de trajets sauvegardés dans le fichier et la proportion des trajets simples et des trajets composés. Ou encore que ces informations figurent dans le fichier en entête. Cependant dans le chargement, nous allons toujours ignorer la première ligne.
* Le cahier de charge ne nous donne aucune contrainte vis-à-vis de la gestion des doublons. Ce choix est justifié par le fait que beaucoup de clients utilisent notre application. Il n’y a donc aucune raison que deux personnes différentes choisissent le même trajet. Et pour ce faire, il doit être présent plusieurs fois dans le catalogue. Cependant, des mêmes trajets sont stockés plusieurs fois dans le catalogue, il y’a donc une perte de mémoire qu’on pourrait bien éviter. Il est possible de ne créer le trajet qu’une seule fois, mais de définir une variable donnant le nb d’occurrence. De même dans le fichier, on affiche à la fin de la ligne ce nombre -là. Ce qui donnera plus de visibilité à la lecture du fichier.