Auteurs : FADILI Zineb, FORLER Corentin

Compte rendu TP de C++ n°3 : Gestion des entrées / sorties

I. Introduction

L'objectif de ce TP est de manipuler les entrées-sorties en utilisant la bibliothèque standard. Deux fonctionnalités ont été ajoutées au programme : chargement et sauvegarde de fichiers. Le projet peut être compilé à l'aide d'un ***makefile***, l'absence de fuites de mémoire est assurée par l'utilisation de ***valgrind***, et la collaboration est assurée via un répertoire ***git***.

II. Format de fichier choisi

Nous avons choisi d'utiliser un format de fichier plein-texte, permettant l'édition manuelle du fichier de sauvegarde ainsi qu'une plus grande interopérabilité. En outre, la plupart des données codées sont composées de texte (ville de départ, ville d'arrivée, mode de transport). Nous avons donc décidé de coder une information par ligne.

II.1. Spécification du format

II.1.a. Trajet simple

Les trajets simples sont stockés dans le fichier sous la forme de 4 lignes :

- Un caractère dénotant le type de trajet (trajet simple) : '>'
- Le nom de la ville de départ
- Le nom de la ville d'arrivée
- Le mode de transport

II.1.b. Trajet composé

Les trajets composés possédant N sous-trajets ($N = 1$ équivalent à un trajet simple $A \rightarrow B$) sont stockés dans le fichier sous la forme de $2 \cdot N + 3$ lignes :

- Un caractère dénotant le type de trajet (trajet composé) : '@'
- Le nombre de sous-trajets N stocké sous la forme d'une suite de chiffres en décimal (et non pas sous forme binaire)
- Le nom de la ville de départ du ***premier*** sous-trajet
- Puis pour chaque sous-trajet, ***k*** allant de *1* à *N* :
  - Le nom de la ville d'arrivée du sous-trajet ***k***
  - Le mode de transport du sous-trajet ***k***

II.2. Nouvelles fonctionnalités

II.2.a. Gestion du nom du fichier

Le chemin d'accès au fichier de sauvegarde ou de téléchargement est explicitement demandé à l'utilisateur, lorsque depuis le menu principal il choisit soit l'option « *Charger dans un fichier* » ou « *Sauvegarder dans un fichier* ». Si l'utilisateur a choisi de sauvegarder le contenu du catalogue dans un fichier, c'est la méthode menuSauvegarder() qui s'occupe de cette demande. Lors du téléchargement c'est la méthode menuCharger(). Dans les deux cas, le programme attend que l'utilisateur entre le chemin d'accès sur le terminal, suivi d'un retour à la ligne. C'est la méthode std::getline() qui permet de le récupérer, le chemin d'accès est stocké dans un std::string.

En mode sauvegarde, un objet de type ofstream est créé, le chemin d'accès au fichier est fourni comme paramètre formel au constructeur de cette classe. (Remarque : si le fichier n'existe pas, il est créé). En mode chargement, un objet de type ifstream est créé, le chemin d'accès au fichier est fourni comme paramètre formel au constructeur de cette classe.

Par la suite, une vérification de l'état du flot est effectuée par la méthode fail() qui vérifie l'état du failbit (qui, si il est à true signifie une erreur logique entrée/sortie, un problème d'ouverture de fichier).

II.2.b. Gestions des trajets vis-à-vis de fichiers

Voici les deux nouvelles fonctionnalités principales pour l'utilisateur en ce qui concerne la gestion des trajets concernant un fichier :

- La possibilité de charger des trajets écrits dans un fichier de manière sélective ou non ;
- La possibilité de sauvegarder les trajets du catalogue de manière sélective ou non.

Si l'utilisateur souhaite filtrer les trajets, il peut le faire selon différents critères (ou aucun) :

- Le type de trajet : simple ou composé ;
- La ville de départ, ou la ville d'arrivée, ou les deux ;
- L'indice du trajet.

Le filtrage repose sur trois méthodes de la classe Parser : FiltreParType(ListOfTrips*, bool, Trip::TYPE), FiltreParIndex(ListOfTrips*, bool, unsigned int, unsigned int) et FiltreParNom(ListOfTrips*, bool, const char*, const char*). **Remarque : l'explication des paramètres et du fonctionnement détaillé de ces méthodes est présent dans Parser.h.** Ces méthodes se trouvent dans la classe Parser pour des raisons arbitraires (éviter de créer encore une autre classe/fichier, de plus le fichier *App.cpp* est déjà long).

Ces méthodes sont appelées depuis la méthode menuFiltrer(ListOfTrips*, bool) de la classe App. Selon les cas, cette méthode demande à l'utilisateur d'entrer certaines données :

- Dans le cas d'un filtrage par type, l'utilisateur doit choisir entre conserver les trajets simples ou les trajets composés.
- Dans le cas d'un filtrage par ville, l'utilisateur doit renseigner la ville de départ, ou la ville d'arrivée, ou les deux. S'il ne souhaite pas porter de critère sur l'une d'elle il lui suffit de saisir un retour à la ligne.
- Dans le cas d'un filtrage par index, l'utilisateur doit saisir l'indice du premier trajet à conserver et l'indice du dernier.

II.2.c. Cas limites

Voici la liste des cas limites lors de la gestion du fichier que nous avons identifiés :

- **Problème lors de l'ouverture du fichier de chargement ou sauvegarde :** détection des erreurs expliquée dans le partie **II.2.a**. Si une erreur est détectée à ce stade, un message d'erreur est affiché sur le terminal : « *Impossible d'ouvrir le fichier de sauvegarde* » ou « *Impossible d'ouvrir le fichier à charger »* et l'utilisateur est renvoyé au menu principal.
- **Problème lors de la lecture du fichier à charger :** Le fichier est mal écrit, il ne respecte pas la spécification du format détaillée en **II.1**. Deux cas de figures :
    - Les indications « @ » et « > » ne sont pas respectées pour les trajets simples et composés : un message d'erreur est affiché : « *Erreur de lecture : caractère invalide à la ligne [n°]. '@' ou '>' attendu mais [erreur] trouvé. »*
    - Un trajet composé est mal écrit : un message d'erreur est affiché « *Erreur de lecture : la ligne [n°] doit ne contenir qu'un nombre mais [reste de la ligne] trouvé après le nombre lu ([nbre])*. »

  Dans les deux cas, le chargement est annulé et l'utilisateur est renvoyé au menu principal.
- **Si le fichier à charger est vide :** Aucune erreur n'est signalée, il n'y aura juste aucun trajet ajouté au catalogue.
- **Si le fichier de sauvegarde existe déjà :** Les données présentes seront écrasées et remplacées par les nouvelles données. S'il n'existe pas, alors il est créé au chemin d'accès indiqué par l'utilisateur.

Cas limites lors de la gestion du filtrage :

- **Les chaines de caractères entrées par l'utilisateur contiennent des espaces :** la récupération des mots entrés par l'utilisateur sont faits grâce à la méthode cin.getline(), les espaces sont donc également récupérés (tous les caractères jusqu'à la fin de ligne).
- **Le choix des types de trajet :** ne pose pas de problème car l'utilisateur n'entre pas directement le type, mais le choisit parmi une liste.
- **L'utilisateur fournit un retour à la ligne pour la ville de départ et/ou la ville d'arrivée :** tout se passe comme s'il n'y avait aucun filtrage, les chaînes vides étant interprétées comme un choix de l'utilisateur de ne pas filtrer par ville de départ/d'arrivée.
- **L'index minimum est égal à l'index maximum** : un seul trajet est gardé, celui dont l'index est égal aux indices minimum *et* maximum (identiques) donnés par l'utilisateur.
- **Les index entrés par l'utilisateur n'entrent pas dans l'intervalle des indices des trajets existants :** annulation de la sauvegarde ou du chargement avec affichage du message d'erreur « *Chargement annulé* » ou « *Sauvegarde annulée* ».

III. Contenu du fichier démo

```
>
Lyon
Bordeaux
Train
@
2
Lyon
Marseille
Bateau
Paris
Avion
>
Lyon
Paris
Auto
```

IV. Conclusion

En conclusion, ce TP nous a permis de mettre en pratiques les connaissances acquises lors des cours magistraux de Programmation Orientée Objet 2. Nous avons ainsi pu continuer à développer nos compétences en programmation ainsi qu'en conception de logiciels, et surtout nous avons pu mettre en pratique de nouvelles compétences concernant la gestion des fichiers et la lecture de données légèrement structurées selon un format de fichier.

Par ailleurs, ce TP nous a permis développer nos capacités d'adaptation, étant donné que l'un de nous a dû s'approprier un code écrit par d'autres personnes et continuer à travailler dessus, et l'autre a dû expliquer et justifier les choix et les démarches réalisées auparavant.

Voici les problématiques majeures auxquelles nous avons fait face :

- Le choix du format des fichiers de sauvegarde. Il a fallu choisir une spécification de format à la fois simple et non ambiguë. Notre solution est détaillée en **II.1.**
- La gestion des cas limites pour les fichiers : problèmes d'ouverture, de création, fichiers vides. Il a particulièrement fallu être vigilant à toutes ces possibilités d'erreurs. Notre solution est détaillée en **II.2.c.**
- Le filtrage a également été un axe sur lequel il a fallu être vigilant. Par exemple lors de la sauvegarde, étant donné que l'on filtre les éléments du catalogue, il ne faut pas supprimer en mémoire les trajets qui ne correspondent pas au filtrage souhaité par l'utilisateur car cela les supprimera également pour le catalogue (et causerai ensuite des erreurs de segmentation…). Or, lors du chargement, tous les trajets issus du fichier sont créés en mémoire. Ainsi, lors du filtrage, il faut supprimer en mémoire ceux qui ne correspondent pas au critère de l'utilisateur (car ils sont inutiles et causeraient des fuites de mémoire). Pour éviter la duplication de code, et ne pas réécrire des méthodes de filtrage quasi-identiques pour chacun de cas (sauvegarde ou filtrage) à la suppression près, il a fallu introduire une variable booléenne appelée shouldFreeMemory qui indique aux méthodes si les trajets doivent être supprimés en mémoire. Cette variable indique, en quelque sorte, si l'on est en mode "sauvegarde" ou mode "enregistrement". C'est une solution malheureusement plutôt inélégante.

Voici les axes d'évolution que nous envisageons pour notre programme :

- Actuellement, lorsque l'on charge un fichier, on crée en mémoire un trajet pour tous ceux stockés dans le fichier. Cette solution nous semble acceptable dans le cadre du TP, une application non réaliste, où l'on sait que le nombre de trajets reste raisonnable. Cependant, il est évident qu'il serait beaucoup plus efficace de filtrer au niveau du fichier avant même de créer les trajets (notamment lors de l'application de filtre pouvant être très sélectifs comme par index ou type).
- Il peut être intéressant d'implémenter la recherche par motif lors du filtrage par ville (par exemple, « par* » pour chercher « Paris » et autres villes de même préfixe).
- Une autre amélioration serait d'implémenter une manière d'éviter les doublons de trajets lors de chargements consécutifs.
- Il peut être intéressant d'implémenter une compression des fichiers de sauvegarde directement dans le logiciel.
- Enfin, il peut aussi être intéressant d'implémenter d'une autre manière l'algorithme d'analyse des fichiers de sauvegarde pour simplifier l'évolution du format de fichier. On peut par exemple utiliser des outils de génération de code comme que flex/bison.