**TP C++ N°3:**

**GESTION DES ENTRÉES/SORTIES**

**SOMMAIRE**

[I. Description détaillée du format du fichier - 2 -](#_Toc533419990)

[A. Spécification sur les restitutions et sauvergardes - 2 -](#_Toc533419991)

[1. Sélection totale - 2 -](#_Toc533419992)

[2. Sélection selon le type - 2 -](#_Toc533419993)

[3. Sélection selon les villes - 2 -](#_Toc533419994)

[4. Sélection selon un intervalle - 2 -](#_Toc533419995)

[B. Spécification des nouvelles fonctionnalités - 2 -](#_Toc533419996)

[1. Gestion des noms de fichiers - 2 -](#_Toc533419997)

[2. Gestion des cas limites - 3 -](#_Toc533419998)

[II. Contenu du fichier demo.txt - 3 -](#_Toc533419999)

[III. Conclusion - 4 -](#_Toc533420000)

[A. Problèmes rencontrés - 4 -](#_Toc533420001)

[B. Axes d’évolution et d’amélioration - 4 -](#_Toc533420002)

# **Description détaillée du format du fichier**

## Spécification sur les restitutions et sauvergardes

### *Sélection totale*

### *Sélection selon le type*

### *Sélection selon les villes*

### *Sélection selon un intervalle*

### *Présentation du format du fichier*

## Spécification des nouvelles fonctionnalités

Ce TP introduit plusieurs nouvelles fonctionalités à notre application : la première est la possibilité de restituer un ancien catalogue, la deuxième de sauvegarder le catalogue courant.

Ces fonctionnalités permettent alors de créer une ville de départ ou une ville d’arrivée avec un espace dans le nom. Pour une plus grande cohérence du tout, nous avons donc adapté l’ajout de trajets en entrée standard par l’utilisateur pour que ce dernier puisse rentrer des trajets dont un nom est composé.

Or, ces nouvelles fonctionalités sous-entendent une plus grande maîtrise des entrées-sorties qu’auparavant. Ci-dessous quelques spécifications sur ces dernières.

### *Gestion des noms de fichiers*

Nous décidons de permettre à l’utilisateur de saisir le nom du fichier cible (dans lequel il souhaite sauvegarder le catalogue courant) ou source (duquel il veut restituer un ancien catalogue).

Lors de la sauvegarde, si le fichier existe déjà, l’utilisateur doit choisir entre écraser le fichier existant, et saisir à la suite du fichier le contenu souhaité. Lors de la restitution, si le fichier n’existe pas, la demande est annulée. Le programme ne vérifie pas que le fichier sélectionné est aux normes choisies.

Cependant, cette liberté d’entrée du fichier sous-entend une vérification systématique de la chaîne de caractères pour qu’elle soit effectivement acceptée par le système d’exploitation de l’ordinateur. Puisque nous travaillons avant tout sous Linux et Mac, nous avons choisis d’interdire les noms de fichiers commencant par un « . », ou ne contenant que des « . » pour lever toutes ambiguïtées. De même, le caractère « : » est interdit sous Mac, alors que les seules interdictions sous Linux sont « / » et « \0 ». Nous les interdisons donc tous.

La longueur maximale du nom sous Linux est limitée à 255 caractères, cependant, nous limitons l’utilisateur à des saisies de moins de 40 caractères.

Une vérification de l’extension .txt est aussi mise en place, le fichier étant considéré comme non valide si l’extension n’est pas la bonne.

### *Gestion des cas limites*

À plusieurs reprises dans notre programme, nous devons traduire une entrée de l’utilisateur (automatiquement interprétée en string) en un entier. Cela sous-entend une vérification, lors de l’appel à la fonction stoi() que l’entrée peut être interprétée comme un entier. Si une erreur est levée soulevant cette impossibilité, alors la demande de l’utilisateur est abandonnée et ce dernier est redirigé vers le menu principal.

De plus, tous nos menus sont régis par le choix de l’utilisateur d’un entier entre 1 et x, x toujours inférieur à 9 pour que cette donnée puisse être interprétée comme un caractère lors de la redirection vers la fonctionalité choisie.

Si l’utilisateur était amené à saisir plusieurs caractères, alors cela aurait pu amener à quelques incohérences : en effet, la fonction get() ne considèrera que le premier caractère, laissant dans le buffer tout autre caractère. Il nous faut donc nettoyer ce dernier après chaque saisie interprétée par le programme comme un caractère.

De la même manière, si une saisie excède la taille limite fixée à 40 caractères, alors la fonction getline() laissera dans le buffer les caractères excèdant, qui seront utilisés par les prochains getline() ou get() sans que la saisie de l’utilisateur y change quelque chose.

Finalement, il est aussi important de vérifier l’état du flux avant d’y opérer. Nous vérifions donc l’état du goodbit avant utilisations

# **Contenu du fichier demo.txt**

2-1

#S,Lyon,Bordeaux,Train

#S,Lyon,Paris,Auto

#C,Lyon,Paris,2

Lyon,Marseille,Bateau

Marseille,Paris,Avion

# **Conclusion**

## Problèmes rencontrés

## Axes d’évolution et d’amélioration