**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*

La sélection totale ne fait aucune distinction entre les différents trajets. Il doit cependant lui être possible – lors de la restitution – de faire appel aux constructeurs de TrajetSimple et TrajetCompose. Ainsi, les informations concernant le type de Trajet, la ville de départ et d’arrivée doivent être indiquées pour tous les trajets. Pour les trajets simples, le mode de transport doit aussi figurer et pour les trajets composés, les informations portant sur chaque étape doivent aussi être explicitées.

### *Sélection selon le type*

Cette sélection doit permettre de sélectionner tous les trajets simples ou tous les trajets composés. Pour faciliter cette sélection, nous rajoutons en tête de fichier les informations sur le nombre de trajets contenu par le fichier par type. Ainsi, si l’utilisateur souhaite restituer tous les trajets composés d’un fichier mais que celui-ci n’est composé que de trajets simples, alors le programme n’a pas besoin de parcourir tout le fichier : il s’arrête à la première ligne.

Pour la restitution d’un trajet composé, il pourrait être très utile de faire figurer le nombre d’étapes composant le trajet composé pour faciliter la lecture du fichier.

### *Sélection selon les villes*

Aucune information de plus ne doit figurer pour ce type de sélection ; les noms des villes figurent nécessairement pour chaque trajet.

### *Sélection selon un intervalle*

Pour ce type de sélection, il peut être utile de connaître en avance le nombre de trajets : ainsi, si l’intervalle demandé n’est pas adapté au cas de figure, nous pouvons le redimensionner pour éviter des erreurs.

Ainsi, si des nombres négatifs sont entrés, ils seront remplacés par des zéro. Si des nombres supérieurs au nombre total de trajets sont entrés, alors ils seront remplacés par ce dernier.

### *Présentation du format du fichier*

Tous ces critères nous ont donc amené à choisir un format permettant de distinguer un trajet simple d’un trajet composé et figurant toutes les informations nécessaires à la création d’un trajet par le biais des constructeurs des classes TrajetSimple et TrajetCompose.

Un nouveau trajet sera identifié par le caractère ‘#’ en début de ligne, suivit du caractère ‘S’ pour un trajet simple ou ‘C’ pour un trajet compose. Toutes les informations seront séparées par une virgule car nous considérons que ce caractère ne peut pas être présent dans le nom d’une ville, ou d’un mode de transport. Pour les trajets composés, une information s’ajoute à ses attributs – le nombre d’étapes.

À noter que l’utilité du caractère ‘#’ était de permettre à une étape d’un trajet composé de s’appelé S ou C sans que cela ne pose de problème d’interprétation pour le programme. Or, nous avons du rajouter l’identifiant ‘S’/’C’ en début de ligne pour définir les étapes d’un trajet composé pour permettre l’ajout d’un trajet composé lui-même composé de trajets composé. Le caractère ‘#’ perd donc son utilité, met rend plus lisible le fichier par l’œil humain.

Afin d’expliciter notre format, voici les deux différents cas de figure.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| # | S | , | villeDepart | , | villeArrivee | , | ModeTransport | \n |

*Exemple du format pour un trajet simple*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| # | C | , | villeDepart | , | villeArrivee | , | nbrEtapes | \n |
| S | , | vD1 | , | vA1 | , | mT1 | \n |  |
| S | , | vD2 | , | vA2 | , | mT2 | \n |  |
| … | … | … | … | … | … |  |  |  |

*Exemple du format pour un trajet composé*

## 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 de l’ensemble, nous avons donc adapté l’ajout de trajet 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, saisir à la suite du fichier le contenu souhaité (le nombre de trajets sera adapté) ou abandonner la démarche. 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 sous 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.

Cela pose un problème lorsque l’utilisateur essaie de rentrer le nom d’un fichier présent autre part que dans le repertoire courant.

La longueur maximale du nom sous Linux est limitée à 255 caractères, cependant, nous limitons l’utilisateur à des saisies de moins de 100 caractères.

En cas de saisie nulle, le nom est rejetté.

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 chaque entrée de données, nous faisons appel à une vérification de l’état du stream : nous vérifions si la fin de ce dernier est atteinte, si le failbit est activé, et pour les cas où nous souhaitons éviter les entrées nulles, nous vérifions la longueur de cette dernière. Pour éviter de devoir créer plusieurs méthodes de ce type pour gérer les différents types d’entrées attendus, nous avons décidé de passer, pour chaque entrée d’entier, par une chaîne de caractère. Cela réduit les possibilités d’erreurs à l’entrée, et nous nous chargeons de vérifier la possibilité de conversion par la suite.

En effet, 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 étant 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.

# **Contenu du fichier demo.txt**

Pour l’exemple donné, nous obtenons donc le fichier demo.txt suivant :

2-1

#S,Lyon,Bordeaux,Train

#S,Lyon,Paris,Auto

#C,Lyon,Paris,2

S,Lyon,Marseille,Bateau

S,Marseille,Paris,Avion

Le premier nombre est celui du nombre de trajets simples, le deuxième celui du nombre de trajets composés. Ils sont séparés par un ‘-‘ afin de faciliter l’accès aux données grâce au getline.

# **Conclusion**

## Problèmes rencontrés

Lors de l’élaboration de ce TP, plusieurs problèmes autours des entrées-sorties nous ont occupés.

Premièrement, l’ajout des nombres de trajets en tête de fichier fut source de nombreux problèmes. En effet, ces informations ne sont accessibles qu’à la fin de l’ajout de tous les trajets. Pourtant, nous avions décidé des les insérer en début de fichier. Cela supposa alors d’initialiser, en début d’écriture, ces nombres à leurs valeurs maximales afin qu’aucune information ne soit accidentellement effacée par la fin de notre méthode.

Deuxièmement, l’option de rajouter à la suite d’un fichier de sauvegarde déjà existant le contenu du catalogue actuel nous a longtemps occupé. Les différents modes d’accès à un fichier ont des conséquences sur l’emplacement du curseur initial, mais aussi sur l’accès aux données du fichier : en effet, l’accès à un fichier en mode « append » permet effectivement d’écrire à la suite du fichier, mais ne permet pas d’accéder aux données présentent avant l’ouverture du fichier existant. Or, l’ajout du nombre de trajets en haut du fichier est compliqué à gérer dans ce cas là car ces données ne peuvent être modifiés qu’une fois les trajets effectivement ajoutés au fichier. Nous avons finalement compris qu’en ouvrant le fichier en out et in, le contenu n’est non seulement pas effacé, il est en plus totalement accessible.

Finalement, les difficultés rencontrées reposaient souvent dans la compréhension des streams et les méthodes associées.

## Axes d’évolution et d’amélioration

Pour le moment, nous n’acceptons pas de fichiers stockés en dehors du répertoire courant. Il pourrait être intéressant d’ajouter cette option.

Pour l’instant, notre format de fichier est fait pour faciliter sa lecture. Nous avons essayé, par le biais du nombre de trajets en début de fichier, d’optimiser le temps d’exécution en lisant avant tout ces informations pour pouvoir « quitter » en cas de solution nulle, sans parcourir l’intégralité du fichier.

Il y a donc énormement de rendondances d’informations dans ce dernier, et il pourrait être intéressant d’essayer de développer une version plus allégée permettant de gagner en mémoire tout en essayant de garder une certaine optimisation en temps d’éxécution.

Le code est aussi très volumineux. Il pourrait être intéressant d’essayer de regrouper certaines fonctionalités se ressemblant en fonctions afin de réduire le plus possible le nombre de lignes.