Le fichier séquentiel [fs]
Support de Cours

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 1er avril 2017

Table des matières

C++ -

Mots-Clés Fichier séquentiel ■
Requis Structuration de l'information ■
Difficulté • ○ ○ (1 h) ■

Introduction

Jusqu'à présent nous avons considéré que les données nécessaires au fonctionnement d'un programme étaient entrées au clavier. Cette façon de procéder se révèle rapidement inadéquate s'il y a une quantité importante de données à saisir ou s'il faut conserver les résultats de manière permanente : c'est là qu'intervient la notion de **fichier**.

Ce module présente le fichier séquentiel, décrit les primitives des fichiers séquentiels, donne des exemples d'utilisation et analyse le cas des fichiers vides.

1 Présentation

Fichier informatique
Unité informationnelle physiquement stockée sur un support de mémoire de masse permanent (disque dur, CD-ROM, clé USB par exemple).

Fichier séquentiel
Le qualificatif **séquentiel** signifie que l'on ne peut accéder à un composant qu'**après** avoir accédé à **tous** ceux qui le précèdent. (Il existe également des fichiers à *accès direct* ; cf. @[Les fichiers].)

Lecture ou Écriture
Nous allons considérer qu'un fichier peut être lu ou écrit mais pas les deux en même temps, même si certains systèmes et langages mettent à disposition des fichiers en lecture/écriture simultanées.

Élément courant
Prochain élément qui sera lu lors du traitement du fichier séquentiel.

Marque de fin de fichier
L'ensemble étant fini, il doit être possible d'en détecter la **fin** : on peut imaginer la présence d'une marque particulière caractérisant cette fin (**marque de fin de fichier**).

Nous utilisons la convention qu'une lecture infructueuse est nécessaire pour détecter la fin d'un fichier. (Il existe une autre convention de lecture que l'on rencontre dans certains langages et qui stipule qu'on peut détecter la fin de fichier juste après la dernière lecture utile.)

2 Exploitation d'un fichier à accès séquentiel

Opération d'entrée/sortie, canal d'entrée/sortie

Un programme qui lit des données ou écrit des résultats dans un fichier accomplit une **opération d'entrée/sortie**. Un tel transfert d'information est effectué via un **canal d'entrées/sorties**. L'information traitée via un canal d'entrées/sorties est présentée sous forme textuelle (c.-à-d. une séquence de caractères) dans le fichier visé.

Fichier en lecture

L'accès à un fichier en lecture (fichier existant non modifié par l'algorithme) répond à la logique :

- Une ouverture (voir `Ouvrir`)
- Des lectures (voir `Prendre` et `FinDeFichier`)
- Une fermeture (voir `Fermer`)

Fichier en écriture

Symétriquement, l'accès à un fichier en écriture (fichier créé par l'algorithme) répond à la logique :

- Une ouverture (voir `Ouvrir`)
- Des écritures (voir `Mettre`)
- Une fermeture (voir `Fermer`)

Modes d'ouverture

Trois modes sont supportés :

- `Lecture` : Permet de lire le contenu du document. Si le document est inexistant, l'exécution de l'algorithme est interrompue.
- `Écriture` : Permet d'écrire des résultats dans le document. Le contenu antérieur à l'ouverture du document est « détruit ». Si le document est un fichier inexistant, celui-ci est créé.
- `Ajout` : Permet d'écrire des résultats à la fin du document. Le contenu antérieur à l'ouverture du document est conservé. Si le document est un fichier inexistant, celui-ci est créé.

Seule l'instruction de lecture (`Prendre`) est autorisée sur un canal d'entrées/sorties associé à un document ouvert en mode `Lecture`. Similairement, seule l'instruction d'écriture (`Mettre`) est autorisée sur un canal d'entrées/sorties associé à un document ouvert en mode `Écriture` ou `Ajout`. Toute instruction de lecture ou d'écriture invalide sur un canal d'entrées/sorties provoque l'arrêt de l'exécution de l'algorithme avec affichage d'un message d'erreur.

3 Primitives

3.1 Déclaration de fichier

Déclaration de fichier

```
#include <fstream>
ifstream varFich; // fichier en lecture (Input)
ofstream varFich; // fichier en écriture (Ouput)
```

Explication

Déclare une variable `varFich` de type `Fichier`.

3.2 Ouverture d'un canal d'entrée/sortie

Ouverture d'un canal

```
varFich.open(nomFich);
```

Déclaration et Ouverture d'un canal

```
#include <fstream>
ifstream varFich(nomFich); // fichier en Lecture
ofstream varFich(nomFich); // fichier en Ecriture
```

Explication

Associe un canal d'entrées/sorties à un fichier. La variable `varFich` sera utilisée pour toutes les opérations sur ce fichier jusqu'à sa fermeture. Le `nomFich` est une chaîne de caractères contenant le nom du fichier à ouvrir avec éventuellement le chemin d'accès à savoir le nom du disque et le chemin relatif ou absolu permettant d'atteindre le fichier. A défaut, le fichier doit être dans le dossier courant (habituellement le dossier où est sauvegardé le projet en exécution).

Erreur à l'ouverture

Le système peut être dans l'impossibilité d'ouvrir le fichier spécifié pour une ou l'autre des raisons suivantes :

- Tentative d'ouvrir un fichier inexistant en mode lecture.
- Tentative d'ouvrir un fichier qui est déjà ouvert.
- Tentative d'ouvrir un fichier sur un canal d'entrées/sorties invalide.
- Le nom du fichier est invalide : ceci peut être dû au dossier inexistant, au nom du fichier contenant des caractères interdits par le système d'exploitation ou à l'unité de stockage défectueuse ou non disponible.

Dans ce cas le système provoque l'interruption du programme et affiche un message d'erreur précisant la cause de l'erreur.

Fichier en mode écriture
L'ouverture efface automatiquement son contenu s'il existe déjà.

3.3 Fermeture d'un canal d'entrées/sorties

Fermeture d'un canal

```
varFich.close();
```

Explication
Ferme le canal d'entrées/sorties `varFich` et purge toutes les mémoires tampon. Dans le cas où le fichier a été ouvert en écriture, cette primitive place la marque spéciale de fin de fichier dans l'élément courant. Une fois le fichier fermé, il n'est plus permis de l'utiliser.

Remarque
Un fichier créé et non refermé risque de contenir des données aléatoires et invalides.

3.4 Lecture depuis un canal d'entrée

Lecture depuis un canal d'entrée

```
varFich >> nomVar1 >> nomVar2...;
```

Explication
Lit dans le fichier référencé par la variable `varFich`, ouvert en lecture, des valeurs qui seront affectées aux variables `nomVarI`. Cette opération peut échouer si la fin de fichier est atteinte. Ceci est décelable grâce à la fonction `FinDeFichier`.

Remarque
Le canal d'entrées/sorties doit obligatoirement être associé à un fichier ouvert en mode `Lecture`. Toute tentative de lecture visant un canal d'entrées/sorties associé à un fichier ouvert en mode `Ecriture` ou `Ajout` cause l'arrêt d'exécution de l'algorithme.

3.5 Écriture sur un canal de sortie

Écriture sur un canal de sortie

```
varFich << expr1 << expr2...;
```

Explication
Rajoute des données dans le fichier référencé par la variable `varFich`, ouvert en écriture, les valeurs des expressions `exprI`. Cette opération peut échouer si le support utilisé pour le fichier est plein.

Remarque
Le canal d'entrées/sorties doit obligatoirement être associé à un document ouvert en mode `Ecriture` ou `Ajout`. Toute tentative d'écriture visant un canal d'entrées/sorties associé à un document ouvert en mode `Lecture` provoque l'arrêt d'exécution de l'algorithme.

3.6 Détection de fin de contenu

Détection de fin de contenu

```
varFich.eof()
```

« eof » signifie « EndOfFile ».

Explication
Renvoie la valeur `Vrai` si le pointeur de lecture référencé par `varFich` détecte la **fin de fichier**, `Faux` sinon.

La primitive n'est applicable qu'aux canaux associés en mode `Lecture`. Toute invocation de la primitive sur un canal associé à un document ouvert en mode `Ecriture` ou `Ajout` cause l'arrêt d'exécution de l'algorithme.

4 Exemples d'utilisation

4.1 Algorithme de base

L'algorithme type dont la structure se retrouve dans la majorité des algorithmes exploitant un fichier est le suivant :

4.2 Exemple : Copie d'un fichier

L'algorithme recopie les données d'un fichier (en lecture) dans un autre (en écriture).

4.3 Le cas des fichiers vides

Dans tout algorithme de manipulation de fichiers, il faut se demander ce qu'il advient du cas d'un fichier vide (rappelons que c'est la première lecture qui permet d'identifier ce cas). Il y a trois cas de figure :

1. C'est un cas particulier qui est traité correctement par l'algorithme général.
2. Il demande un traitement spécial qui n'est pas couvert par le cas général.
3. Il dénote un problème mal posé qui ne peut être résolu.

Exemple
Si le problème est de calculer la somme, la moyenne ou encore le maximum de nombres contenus dans un fichier, alors la solution ne peut être donnée dans le cas d'un fichier vide (cas 3) à moins que l'analyse n'ait clairement indiqué la réponse à donner dans ce cas (cas 2).

Remarque
Il faut donc traiter le cas particulier de fichier vide juste après la première lecture. Réécrivons alors l'algorithme général de traitement d'un fichier séquentiel.

Remarque
Lorsque le « traitement spécial du fichier vide » consiste à générer une « ERREUR » alors le `Si` peut être changé en un `Si-Alors` suivi du traitement normal.

4.4 Exemple : Calcul de la moyenne

La fonction calcule et renvoie la moyenne d'une série de réels contenus dans un fichier.

5 Références générales

Comprend [] ■