Manipulation de fichiers

Table des matières

En informatique, un fichier est un ensemble d'informations stockées sur un support, réuni sous un même nom et manipulé comme une unité. Techniquement un fichier est une information numérique constituée d'une séquence d'octets, c'est-à-dire d'une séquence de nombres, permettant des usages divers.

1.Introduction

Qu'elles proviennent de capteurs ou d'informations d'utilisateurs, les données sont en fait absolument essentielles à n'importe quel projet. Plus encore, depuis quelques années, on s'intéresse au Big Data où, cette fois-ci, les données deviennent des marchandises que l'on peut s'échanger et qui possèdent une grande valeur.

Pourtant, traiter un grand volume de données n'a pas toujours été facile dans l'histoire de l'informatique. Aujourd'hui, à l'heure du tera-octet de données sur un ordinateur portable, on développe des jeux ou des programmes de plusieurs giga-octets. Pourtant, quelques années en arrière, on comptait encore en kilobits, kilo-octet ou autres unités qui nous semble maintenant désuètes. Il serait bon parfois de repenser à cet ancien temps où le travail du développeur était d'optimiser ses programmes et où cette dernière jouait en rôle absolument essentiel dans la qualité d'un programme.

L'idée de la manipulation de fichier, c'est de stocker des données sous une autre forme que celle d'une base de données par exemple. En fait, c'est d'être capable de les enregistrer dans un fichier texte de manière formatée et d'être ensuite capable de travailler sur ce fichier. Cela a pour avantage de ne pas nécessiter la mise en place d'un serveur de base de données et donc de faire gagner plus ou moins de temps dans la réalisation de nos « petits » projets personnels.

2. Langage C

2.1. Ouverture de fichier

La bibliothèque standard vous fourni différentes fonctions pour manipuler les fichiers, toutes déclarées dans l'en-tête <stdio.h>. Toutefois, celles-ci manipulent non pas des fichiers, mais des flux de données en provenance ou à destination de fichiers. Ces flux peuvent être de deux types :

- des flux de textes qui sont des suites de caractères terminées par un caractère de fin de ligne (\n) et formant ainsi des lignes ;
- des flux binaires qui sont des suites de multiplets.

La fonction fopen() permet d'ouvrir un flux. Celle-ci attend deux arguments : un chemin d'accès vers un fichier qui sera associé au flux et un mode qui détermine le type de flux (texte ou binaire) et la nature des opérations qui seront réalisées sur le fichier via le flux (lecture, écriture ou les deux). Elle retourne un pointeur vers un flux en cas de succès et un pointeur nul en cas d'échec.

Le mode est une chaîne de caractères composée d'une ou plusieurs lettres qui décrit le type du flux et la nature des opérations qu'il doit réaliser.

Cette chaîne commence obligatoirement par la seconde de ces informations. Il existe six possibilités reprises dans le tableau ci-dessous.

Mode	**Type(s) d'opération(s)**	**Effets**
r	Lecture	Néant
r+	Lecture et écriture	Néant
w	Écriture	Si le fichier n'existe pas, il est créé.

		Si le fichier existe, son contenu est effacé.
w+	Lecture et écriture	Idem
a	Écriture	Si le fichier n'existe pas, il est créé. Place les données à la fin du fichier
a+	Lecture et écriture	Idem

Par défaut, les flux sont des flux de textes. Pour obtenir un flux binaire, il suffit d'ajouter la lettre b à la fin de la chaîne décrivant le mode.

La fonction fclose() termine l'association entre un flux et un fichier. S'il reste des données temporisées, celles-ci sont écrites. La fonction retourne zéro en cas de succès et EOF en cas d'erreur.

2.2. Écriture dans un fichier

La fonction fputs() écrit une ligne dans le flux. La fonction retourne un nombre positif ou nul en cas de succès et EOF en cas d'erreurs. La fonction puts() est identique si ce n'est qu'elle ajoute automatiquement un caractère de fin de ligne et qu'elle écrit sur le flux stdout.

```
#include <stdio.h>

int main(void)
{
   FILE *fp = fopen("texte.txt", "w");

   if ( fp == NULL )
   {
      fputs("Le fichier texte.txt n'a pas pu être ouvert\n", stderr);
      return -1;
   }

   if ( fputs("Premier test d'écriture dans un fichier en langage C", fp) == EOF )
   {
      fputs("Erreur lors de l'écriture d'une ligne\n", stderr);
      return -2;
   }

   if ( fclose(fp) == EOF )
   {
      fputs("Erreur lors de la fermeture du flux\n", stderr);
      return -3;
   }

   return 0;
}
```

Remarques :

1. La fonction int fprintf(FILE *flux, char *format, …) est la même que la fonction printf() si ce n'est qu'il est possible de lui spécifier sur quel flux écrire (au lieu de stdout pour printf()). Elle retourne le nombre de caractères écrits ou une valeur négative en cas d'échec.
2. La fonction size_t fwrite(void *ptr, size_t taille, size_t nombre, FILE *flux) permet d'écrire des valeurs binaires référencées par ptr, composées de nombre éléments de taille multiplets dans le flux. Elle retourne une valeur égale à nombre en cas de succès et une valeur inférieure en cas d'échec.

2.3. Lecture d'un fichier

La fonction size_t fread(void *ptr, size_t taille, size_t nombre, FILE *flux) est l'inverse de la fonction fwrite() : elle lit nombre éléments de taille multiplets depuis le flux et les stocke dans l'objet référencé par ptr. Elle retourne une valeur égale à nombre en cas de succès ou une valeur inférieure en cas d'échec.

```
#include <stdio.h>

int main(void)
{
   FILE *fp = fopen("texte.txt", "r");

   if ( fp == NULL )
   {
      fputs("Le fichier texte.txt n'a pas pu être ouvert\n", stderr);
      return -1;
   }

   int     n;
   char    string[81];
   while ( (n = fread(&string, sizeof(char), 80, fp)) > 0 )
   {
      string[n] = 0;
      printf("%s", string);
   }

   if ( fclose(fp) == EOF )
   {
      fputs("Erreur lors de la fermeture du flux\n", stderr);
      return -3;
   }

   return 0;
}
```

3. Langage C++

3.1. Ouverture de fichier

Afin d'ouvrir des fichiers, que ce soit en lecture ou en écriture, il faut utiliser la classe ifstream pour la lecture et ofstream pour l'écriture. Ces deux classes sont des dérivées de std::ios_base qui est la classe de base pour les flux.

Pour pouvoir les utiliser, il faut inclure l'entête fstream.

3.2. Écriture dans un fichier

Pour ouvrir le fichier en écriture, il y a différents modes d'ouverture :

Mode	**Type(s) d'opération(s)**
ios::out (output)	spécifie qu'on ouvre le fichier en écriture. Par défaut quand on utilise un objet ofstream.
ios::app (append = ajouter à la suite)	lorsqu'on ouvre le fichier en écriture, on se trouve à la fin pour écrire des données à la suite du fichier (sans effacer le contenu, s'il y en a un). Avec ce mode d'ouverture, à chaque écriture, on est placé à la fin du fichier, même si on se déplace dans celui-ci avant.
ios::trunc (truncate = tronquer)	lorsqu'on ouvre le fichier en écriture, spécifie qu'il doit être effacé s'il existe déjà, pour laisser un fichier vide.
ios::ate (at end)	ouvre le fichier en écriture et positionne le curseur à la fin de celui-ci. La différence avec ios::app est que si on se repositionne dans le fichier, l'écriture ne se fera pas forcément à la fin du fichier, contrairement à ios::app.

```
#include <iostream>
#include <fstream>

using namespace std;

int main(void)
{
   ofstream flux("texte.txt");

   if ( !flux )
   {
      cerr << "Le fichier texte.txt n'a pas pu être ouvert";
      return -1;
   }

   flux << "Premier test d'écriture dans un fichier en langage C++";
```

```
   if ( flux.good() == false )
   {
      cerr << "Erreur lors de l'écriture d'une ligne";
      return -2;
   }

   flux.close();

   return 0;
}
```

3.3. Lecture d'un fichier

Quand on ouvre un fichier en lecture, le mode d'ouverture ios::in est positionné par défaut.

```
#include <iostream>
#include <fstream>

using namespace std;

int main(void)
{
   ifstream flux("texte.txt");

   if ( !flux )
   {
      cerr << "Le fichier texte.txt n'a pas pu être ouvert";
      return -1;
   }

   string ligne;
   while ( getline(flux, ligne) )
   {
      cout << ligne;
   }

   flux.close();

   return 0;
}
```

3.4. Gestion par exception

Une exception est l'interruption de l'exécution du programme à la suite d'un événement particulier. Le but des exceptions est de réaliser des traitements spécifiques aux événements qui en sont la cause.

```
#include <iostream>
#include <fstream>
```

```
using namespace std;

int main(void)
{
   ofstream flux("texte.txt");

   try
   {
    ofstream flux("texte.txt", ios_base::app);
    flux.exceptions(ofstream::failbit | ofstream::badbit);

    if ( flux.is_open() )
    {
      flux << "Premier test d'écriture dans un fichier en langage C++";
      flux.close();
    }
   }
   catch(ios_base::failure &ex)
   {
      cerr << "Erreur lors de l'écriture d'une ligne";
      return -1;
   }

   return 0;
}
```

4. Langage Python

4.1. Ouverture de fichier

Sous Python, l'accès aux fichiers est assuré par l'intermédiaire d'un « objet-fichier » que l'on crée à l'aide de la fonction interne open(). Après avoir appelé cette fonction, vous pouvez lire et écrire dans le fichier en utilisant les méthodes spécifiques de cet objet-fichier.

La fonction open est disponible sans avoir besoin de rien importer. Elle prend en paramètre :

- le chemin (absolu ou relatif) menant au fichier à ouvrir ;
- le mode d'ouverture.

Le mode est donné sous la forme d'une chaîne de caractères. Voici les principaux modes :

- 'r' : ouverture en lecture (Read).
- 'w' : ouverture en écriture (Write). Le contenu du fichier est écrasé. Si le fichier n'existe pas, il est créé.
- 'a' : ouverture en écriture en mode ajout (Append). On écrit à la fin du fichier sans écraser l'ancien contenu du fichier. Si le fichier n'existe pas, il est créé.

Remarque : on peut ajouter à tous ces modes le signe b pour ouvrir le fichier en mode binaire.

```
>>> mon_fichier = open("fichier.txt", "r")
```

Remarque : si d'autres applications, ou d'autres morceaux du code, souhaitent accéder à ce fichier, ils ne pourront pas car le fichier sera déjà ouvert. La méthode à utiliser est close :

```
>>> mon_fichier.close()
```

4.2. Écriture dans un fichier

Vous pouvez utiliser le mode w ou le mode a. Le premier écrase le contenu éventuel du fichier, alors que le second ajoute ce que l'on écrit à la fin du fichier. Dans tous les cas, ces deux modes créent le fichier s'il n'existe pas.

Pour écrire dans un fichier, on utilise la méthode write en lui passant en paramètre la chaîne à écrire dans le fichier. Elle renvoie le nombre de caractères qui ont été écrits. Chaque nouvel appel de **write()** continue l'écriture à la suite de ce qui est déjà enregistré.

```
fd = open("fichier.txt", "w")

nbcar = fd.write("Premier test d'écriture dans un fichier via Python")
if nbcar == 0:
      print "erreur d'écriture"

fd.close()
```

Remarque : la méthode write n'accepte en paramètre que des chaînes de caractères. Si vous voulez écrire dans votre fichier des nombres, des scores par exemple, il vous faudra les convertir en chaîne avant de les écrire et les convertir en entier après les avoir lus.

4.3. Lecture d'un fichier

La méthode read renvoie tout ou une partie du contenu du fichier, que l'on capture dans une chaîne de caractères. Le paramètre de la méthode indique le nombre de caractères à lire. S'il est omis, tout le fichier est lu.

```
fd = open("fichier.txt", "r")

end = False
while not end:
     text = fs.read(50)
     if text == "":
              end = True
     else:
              print text

fd.close()
```

4.4. Le mot-clé with

Il peut se produire des erreurs quand on lit, quand on écrit… et si l'on n'y prend garde, le fichier restera ouvert.

Il existe un mot-clé qui permet d'éviter cette situation :

```
with open(fichier, mode_ouverture) as variable:
      # Opérations sur le fichier
```

Si une exception se produit, le fichier sera tout de même fermé à la fin du bloc.

Le mot-clé with permet de créer un "context manager" (gestionnaire de contexte) qui vérifie que le

fichier est ouvert et fermé, même si des erreurs se produisent pendant le bloc.

Il est inutile, par conséquent, de fermer le fichier à la fin du bloc with. Python va le faire tout seul, qu'une exception soit levée ou non.

5. Langage PHP

5.1. Ouverture de fichier

Pour créer un nouveau fichier en PHP, nous allons à nouveau utiliser la fonction fopen().

En effet, cette fonction permet à la fois d'ouvrir un fichier ou du créer un fichier si on précise un mode adapté. En mentionnant simplement cela, le fichier sera par défaut créé dans le même dossier que notre page PHP.

5.2. Écriture dans un fichier

Pour écrire dans un fichier en PHP, nous allons utiliser la fonction fwrite(). Cette fonction va prendre en arguments l'information renvoyée par fopen() ainsi que le texte à écrire.

```
$fp = @fopen("texte.txt", "w");

if ( !$fp )
{
        echo "Le fichier texte.txt n'a pas pu être ouvert";
        return -1;
}

if ( !fwrite($fp, "Premier test d'écriture dans un fichier en langage PHP") )
{
        echo "Erreur lors de l'écriture d'une ligne";
        return -2;
}

if ( !fclose($fp) )
{
        echo "Erreur lors de la fermeture du flux";
        return -3;
}
```

5.3. Lecture d'un fichier

Pour lire un fichier en PHP, nous allons utiliser la fonction fread(), abréviation de « file read ».

Cette fonction va prendre en arguments le résultat renvoyé par fopen() et le nombre de Bytes maximum qui doivent être lus.

Remarque : pour être sûr de toujours lire l'entièreté de notre fichier, nous pouvons utiliser la fonction filesize() qui va justement renvoyer la taille d'un fichier. En lui passant notre fichier en argument, filesize() va donc renvoyer exactement sa taille qu'on va pouvoir passer à fread() (en supposant bien entendu qu'on commence à lire le fichier à partir du début).

```
$fp = @fopen("texte.txt", "r");

if ( $fp === false )
{
	echo "Le fichier texte.txt n'a pas pu être ouvert";
	return -1;
}

while ( !feof($fp) )
{
	$string = fread($fp, 512);
	echo $string;
}

if ( !fclose($fp) )
{
	echo "Erreur lors de la fermeture du flux";
	return -3;
}
```

5.4. Les exceptions

Le PHP 5 a introduit une nouvelle façon de gérer certaines erreurs en utilisant l'orienté objet.

Très simplement, on appelle ce type d'erreurs qu'on va pouvoir gérer en utilisant l'orienté objet des exceptions.

Les exceptions correspondent donc à un type précis d'erreurs qu'on va pouvoir lancer (throw) et attraper (catch).

```
try
{
	$fp = @fopen("texte.txt", "w");

	if ( $fp )
	{
		fwrite($fp, "Premier test d'écriture dans un fichier en langage PHP");
		fclose($fp);
	}
}
catch (Exception $e)
{
	echo $e->getMessage();
}
```