TD 6 — Fichiers

Jusqu'à maintenant, tous nos programmes manipulaient des variables stockées dans la mémoire, qui s'effaçaient à la fin de l'exécution. Dans ce TD, nous allons voir comment lire et écrire des données dans des fichiers.

Les fichiers sont des structures ordonnées dont tous les éléments sont du même type. Généralement, ils sont stockés sur des supports externes. Quelles que soient les particularités du système d'exploitation sous-jacent, les fonctions de la bibliothèque standard des entrées-sorties du langage C font en sorte que les fichiers soient vus par le programme comme des flots (ou des flux), c'est-à-dire des suites d'octets qui représentent de données selon une des deux modalités suivantes : flot de caractères ou flot binaire.

1 Fichiers binaires et fichiers textuels

Les fichiers sont de deux sortes.

Les fichiers binaires contiennent des données, présentées comme elles sont stockées en mémoire. Ce sont des fichiers composés d'octets, dans lesquels on peut par exemple enregistrer des valeurs de variables. Cest fichiers ne sont pas manipulables depuis un éditeur de texte… puisqu'ils ne contienent pas de texte !

Les fichiers textuels contiennent des chaines de caractères, qui sont écrites en lignes dans le format approprié du sytème d'exploitation. La lecture et l'écriture se font par conversion dans le type de la variable.

On peut accéder au contenu des fichiers de deux manières.

L'accès séquentiel pour des fichiers contenant des éléments du même type, avec une lecture du premier au dernier élément sans retour en arrière. *Tous les fichiers texte sont séquentiels.*

L'accès direct pour accéder aux éléments dans un ordre arbitraire (sauts vers l'avant, retours, etc.).

2 Accès à un fichier

Les fonctions permettant de manipuler les fichiers en C sont fournies par la bibliothèque `stdio.h`. Elle fournit un type spécifique de variable, le type `FILE`. Pour pouvoir accéder à un fichier, en lecture ou en écriture, on déclare un pointeur sur un objet de type `FILE` :

```c
#include <stdio.h>

int main(void) {
    FILE* fichier = NULL; // Pointeur vers le fichier que nous souhaitons manipuler

    /* Code … */

    return EXIT_SUCCESS;
}
```

2.1 Ouvrir un fichier

L'objectif de l'ouverture est de préparer le fichier pour le traitement, soit lecture, soit écriture ou les deux. Pour ouvrir un fichier, on fait appel à la fonction `fopen` (pour *file open*), dont le prototype est le suivant :

```c
FILE* fopen(const char* nomFichier, const char* mode);
```

- `const char* nom_fichier` est une chaine de caractères (ou un pointeur vers le premier caractère, ce qui revient au même, cf. le TD sur les pointeurs), indiquant le nom du fichier tels qu'il est présent sur la machine.
- `const char* mode` est une chaine de caractères, indiquant le mode d'ouverture (lecture seule ou lecture et écriture) et le type de fichier (textuel ou binaire).
- Le pointeur retourné prend la valeur `NULL` si l'ouverture du fichier a échoué.

Vérification de la valeur de retour

Comme pour les allocations dynamiques de mémoire, il faut vérifier systématiquement que l'ouverture si fichier a échoué :

```c
FILE* fichier = fopen("test.txt", "rt"); // Ouverture d'un fichier textuel
                                         // en mode lecture
if(fichier==NULL) {
    printf("Erreur d'ouverture…\n");
    return EXIT_FAILURE;
}
/* Suite du code si l'ouverture a réussi… */
```

Le tableau ci-dessous donne la liste des différents modes et types utilisés à l'ouverture d'un fichier.

`r`	Fichier ouvert en lecture seule. Si le fichier n'existe pas, l'ouverture échoue.
`w`	Fichier ouvert en écriture seule. Si le fichier existe, il est effacé. S'il n'existe pas, il est créé.
`a`	Fichier ouvert en écriture. Si le fichier existe, le contenu est ajouté à la suite. S'il n'existe pas, il est créé.
`w+`	Fichier ouvert en lecture et écriture. Si le fichier existe, il est effacé. S'il n'existe pas, il est créé.
`r+`	Fichier ouvert en lecture et écriture avec curseur en début de fichier. Si le fichier n'existe pas, l'ouverture échoue.
`a+`	Fichier ouvert en lecture et écriture avec curseur en fin de fichier. Si le fichier n'existe pas, il est créé.

`t`	Fichier textuel
`b`	Fichier binaire

Exemple. Voici quelques exemples d'appel à la fonction `fopen` :

`f = fopen("ftext.txt", "rt")`	Ouverture d'un fichier textuel existant en lecture seule.
`f = fopen("fbin.bat", "rb")`	Ouverture d'un fichier binaire en lecture seule.
`f = fopen("ftext.log", "wt")`	Ouverture d'un fichier textuel en mode écriture seule.
`f = fopen("fbin.bat", "r+b")`	Ouverture d'un fichier binaire déjà existant en lecture et écriture.
`f = fopen("ftext.txt","a+b")`	Ouverture d'un fichier binaire en mode lecture et écriture à la fin.

2.2 Fermer un fichier

Lorsque l'on ferme le fichier, le système d'exploitation efface le tampon et la liaison entre le nom du fichier et le pointeur de type `FILE*` est coupée. On ferme un fichier avec la fonction `fclose` fournie par la bibliothèque `stdio.h` ; voici son prototype :

stdio.h
```
1 int fclose(FILE* fichier);
```

Exercices

Exercice 1. Écrire un programme qui ouvre un fichier textuel en mode lecture seule et le ferme aussitôt. Le fichier, nommé `exo1_test.txt`, devra être placé dans le même répertoire que l'exécutable. Il pourra éventuellement contenir du texte. On pensera à tester la valeur retournée par la fonction `fopen` pour vérifier que l'ouverture se déroule sans erreur.

Exercice 2. **1)** Reprendre le code précédent en ouvrant, cette fois, le fichier en mode textuel et en écriture (mode `w`), puis fermez-le. Que constatez-vous en visualisant le contenu du fichier ?

2) Ouvrez un fichier, nommé `exo2_test.txt`, inexistant sur le disque, avec le même mode. Que remarquez-vous ?

3 Lecture et écriture dans un fichier

Nous avons vu comment ouvrir et fermer un fichier à travers un programme écrit en C. Mais nous n'avons pas vu comment *lire son contenu* ou *écrire*. C'est l'objectif de cette partie.

Le traitement est différent selon que l'on travaille avec un fichier binaire ou avec un fichier textuel.

3.1 Fichiers binaires

3.1.1 Lecture de fichiers binaires

La lecture dans un fichier binaire se fait avec la fonction `fread`, dont le prototype est le suivant :

stdio.h

```
size_t fread(void* donneesLues, size_t tailleObjet,
             size_t nbObjetsLus, FILE* fichier);
```

La lecture se fait *objet par objet*. On doit préciser la taille de chaque objet et le nombre d'objets qui doivent être lus. Le nombre d'octets lus est le produit des deux valeurs.

`void* donneesLues` est un pointeur vers la zone mémoire où l'on souhaite stocker les données.

`size_t tailleObjet` est la taille, en octets, d'un objet lu. Si l'on lit un `int`, la taille sera `sizeof(int)`. Si l'on lit un tableau de 4 `float`, la taille sera `4*sizeof(float)`.

`size_t nbObjetsLus` est le nombre d'objets que l'on souhaite lire.

`FILE* fichier` est un pointeur vers le fichier à lire.

La valeur retournée est la taille du nombre d'objets réellement lus.

Le type `size_t`

Le type `size_t` en C est un type dérivé de `unsigned int` qui permet de stocker des *tailles* de variables. Son fonctionnement interne est hors de propos dans le cadre d'un cours introductif. Vous avez simplement à retenir qu'il améliore la lisibilité du code et sa portabilité, en fournissant un type standard pour les tailles de variables, de tableaux, etc. On peut considérer qu'une valeur de tyêpe `size_t` se comporte comme un entier non signé.

3.1.2 Écriture dans des fichiers binaires

On écrit dans un fichier binaire avec la fonction `fwrite` :

stdio.h

```
size_t fwrite(void* donneesEcrites, size_t tailleObjet,
              size_t nbObjetsEcrits, FILE* fichier);
```

La valeur de retour est la taille, en nombre d'octets, des objets réellement écrits.

3.2 Fichiers textuels

3.2.1 Lecture de fichiers textuels

La bibliothèque `stdio.h` fournit trois fonctions pour lire dans un fichier textuel : `fgetc`, `fgets` et `fscanf`. Voici leurs prototype respectifs :

stdio.h
```
int fgetc(FILE* fichier);
char* fgets(char* chaine, int longueurMax, FILE* fichier);
int fscanf(FILE* fichier, char* format, adresse1, …, adressen);
```

La fonction `fgetc` (*file get character*) permet de lire un seul caractère dans le fichier pointé par `FILE* fichier`. La valeur retournée est le code ASCII du caractère lu.

Type de la valeur retournée

La fonction `fgetc` renvoie une variable de type `int` mais il s'agit bien un caractère qui est lu !

La foncion `fgets` (*file get string*) lit les caractères dans un fichier texte jusqu'à *la fin de la ligne* ou si `longueurMax-1` caractères ont été lus. La valeur retournée est un pointeur vers la chaine de caractères, qui vaut `NULL` en cas d'erreur ou si la fin du fichier est atteinte.

Le caractère de fin de fichier `EOF`

La fin d'un fichier textuel est indiquée par le caractère spécial, noté `EOF` (sans guillemets ni apostrophes), qui signifie *end of file*. C'est un caractère *non imprimable* : il ne s'affiche pas à l'écran, mais il indique que le fichier est terminé.

Enfin, la fonction `fscanf` fonctionne comme la fonction `scanf` : elle lit une chaine de caractères formatée, et le premier argument est un pointeur vers le fichier à lire. La valeur de retour est `EOF` en cas d'erreur, et le nombre de caractères lus en cas de succès.

3.2.2 Écriture dans des fichiers textuels

Comme pour la lecture, l'écriture dans des fichiers est fournie par trois fonctions définies dans la bibliothèque `stdio.h` :

stdio.h
```
// Retourne le code du caractère écrit ou EOF en cas d'erreur
int fputc(int caractere, FILE* fichier);

// Retourne un entier positif ou EOF en cas d'erreur
char* fputs(const char* chaine, FILE* fichier);

// Retourne le nombre d'octets imprimés ou une valeur négative en cas d'erreur
int fprintf(FILE* fichier, char* format, expr1, expr2, …, exprn);
```

4 Exercices d'application

Exercice 3. On considère le code source suivant :

main.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    float nombre;
    int m;
    FILE *nombresTxt, *nombreBin;

    nombresTxt = fopen("nombres.txt","wt");
    nombresBin = fopen("nombres.dat","wb");

    if(nombresTxt ==NULL || nombreBin == NULL)  {
        return EXIT_FAILURE;
    }
    else {
        printf("Saisir des nombres en finissant avec CTRL-Z\n");
        m=scanf("%f", &nombre);
        // CTRL-Z code EOF (End-Of-File) sous Windows
        while (m != EOF){
            fprintf(nombresTxt, "%8.3f", nombre);
            fputc('\n', nombresTxt);
            fwrite(&nombre, sizeof(float), 1, nombreBin);
            m=scanf("%f", &nombre);
        }
        fclose(nombresTxt);
        fclose(nombreBin);
    }
    return EXIT_SUCCESS;
}
```

1) Tester le programme et analyser le code pour comprendre son fonctionnement. Ajouter les commentaires pertinents aux bons endroits pour améliorer sa lisibilité.

2) Après avoir saisi plusieurs nombres au clavier, ouvrir les fichiers `nombresTxt.txt` et `nombresBin.txt` avec un éditeur de texte. Que remarque-t-on ?

3) Modifier le code pour obtenir, dans le fichier `nombresTxt.txt`, le numéro d'ordre de la saisie en plus de sa valeur, comme suit :

nombresTxt.txt

```
1:nombre1;2:nombre2;3:nombre3;…;10:nombre10;…
```

4) Procéder de même pour enregistrer dans un fichier binaire le numéro de chaque saisie (qui sera donc de type **`unsigned int`**) et la valeur saisie (de type **`float`**).

Exercice 4. Reprendre le code de l'exercice précédent et le modifier pour obtenir la présentation suivante dans le fichier texte (sur quatre colonnes) :

nombresTxt.txt

```
nombre1 nombre2 nombre3 nombre4
nombre5 nombre6 nombre7 nombre8
…
```

Exercice 5. Proposer un algorithme qui permet de relire et d'afficher dans la console le fichier créé à l'exercice 4.
Indication — On pourra utiliser la fonction `fscanf` avec le format `%8.3f` pour la lecture des flottants.

Exercice 6. Un fichier binaire, nommé `donnees.bin`, est mis à votre disposition. Il renferme des données structurées de la manière suivante :

valeur1	nom1	annne1	valeur2	nom2	annne2	…	valeurN	nomN	anneeN

où :

- `valeurk` est une variable de type `float` ;
- `nomk` est une chaine de caractères de taille 10, i.e. de type `char[10]` ;
- `anneek` est de type `int`.

Le nombre `N` de triplets (`anneek`, `nomk`, `valeurk`) n'est pas connu. Proposez un programme qui permet de lire le contenu du fichier dans sa totalité et affiche les valeurs des variables lues dans la console, avec un triplet par ligne.

Exercice 7. **1)** Créez un fichier binaire `nombres.dat` contenant des variables de type `int`.

2) Créez un second fichier, nommé `carres.dat`, contenant les carrés des valeurs écrites dans le fichier précédent.

3) Reprenez le fichier `nombres.dat` et *remplacez* dans ce fichier chaque valeur lue par son cube.