Chapitre 14

LES FICHIERS

14.1 Généralités

Lorsqu'un utilisateur saisit des données, il est souvent utile de les conserver sur disque. Ainsi ces données peuvent être récupérées entre deux exécutions du programme ou deux démarrages de la machine. Les informations sont alors enregistrées dans un ou plusieurs fichiers.

Un fichier est un ensemble d'informations stockées sur une mémoire de masse (disque dur, SSD, bande magnétique, CD, carte SD ...). Un fichier est une suite d'octets. Les informations contenues dans le fichier ne sont pas forcément de même type (un char, un int, une structure ...).

Différents types de fichiers

Fichier texte :
les données sont une suite de carcatères qui forment des lignes terminées par un ou deux caractères de fin de ligne :

- Convention "Unix" : LF (Line Feed), code ASCII 10, noté `'\n'` en C,
- Convention "Mac" : CR (Carriage Return), code ASCII 13, noté `'\r'` en C,
- Convention "Dos" : CR LF.

Un fichier texte peut être créé par un éditeur de texte.

Fichier binaire :
les données sont une suite d'octets sans structure particulière.

Fichier structuré :
Ce sont des fichiers texte dans lesquels les données sont une suite de données structurées (pouvant contenir diverses informations) séparées par des délimiteurs. Certains formats sont bien connus : CSV (Comma-Separated Values) où chaque information est séparée par des virgules (ou des points virgules). Il existe d'autres formats plus élaborés comme XML et JSON.

Méthodes d'accès

La bibliothèque standard vous fournit différentes fonctions pour manipuler les fichiers, toutes déclarées dans l'entête `<stdio.h>`. Toutefois, celles-ci manipulent non pas des fichiers, mais des flux de données en provenance ou à destination de fichiers. L'utilisation d'un flux a deux intérêts. Le premier est de ne plus se soucier des différentes conventions des systèmes d'exploitation. En effet il nous suffira d'écrire `'\n'` dans notre code pour effectuer un retour à la ligne, quel que soit l'OS.
Le second, est de réduire le temps d'accès à la mémoire ROM (la plus lente de la machine), en plaçant un tampon intermédiaire. Un flux autorise deux formes d'accès :

Accès séquentiel : la lecture ou l'écriture de données s'effectue depuis le début du fichier, ligne à ligne, les unes après les autres dans l'ordre. Le tampon intermédaire manipule donc des lignes (temporisation/mémorisation par lignes).

accès direct (Random I/O) : où nous pouvons nous positionner sur une donnée en indiquent son numéro d'ordre. Ici le tampon intermédaire manipule un bloc de données d'une taille déterminée (temporisation/mémorisation par bloc).

14.2 Accès aux fichiers

Comment faire pour accéder à un fichier dans notre programme ? Comment relier le nom d'un fichier (identificateur externe au programme) à un identificateur interne à notre code source ?

La fonction fopen

```
1 FILE *fopen(char *chemin, char *mode);
```

La fonction `fopen()` permet d'ouvrir un flux. Celle-ci attend deux arguments :

- un chemin d'accès (relatif ou absolu) vers un fichier qui sera associé au flux,
- un mode qui détermine la nature du flux : binaire ou texte, ainsi que le type d'accès (lecture , écriture ou les deux).

Elle retourne un pointeur vers un flux en cas de succès ou un pointeur de type `NULL` en cas d'échec. Ce pointeur, appelé **pointeur de fichier**, pointe sur une structure contenant des informations sur le fichier : l'adresse du tampon, la position du caractère courant dans le tampon, des indicateurs permettant de savoir si le fichier est ouvert en lecture ou en écriture, si des erreurs sont intervenues ou si la fin du fichier est atteinte. Le mode d'ouverture du flux fait intervenir trois sortes d'indicateurs qu'il est possible de combiner.

TYPE D'INDICATEUR	VALEUR	SIGNIFICATION
Principal (obligatoire)	"r"	Lecture seule d'un fichier existant
	"w"	Ecriture seule dans un nouveau fichier ou écrasement d'un fichier existant
	"a"	Extension : ajout d'informations **en fin de fichier uniquement**. Si le fichier n'esxiste pas, il sera créé.
Mode d'ouverture (facultatif)	"b"	Ouverture d'un flux binaire.
Mise à jour (facultatif)	"+"	Autorise à la fois la lecture et l'écriture.

Tableau 14.1 – Les trois indicateurs du mode d'ouverture.

MODE	TYPE D'ACCES	POSITIONNEMENT DU POINTEUR DE FICHIER	FICHIER EFFACÉ	FICHIER CRÉÉ SI INEXISTANT
"r"	lecture seule, mode texte	début	non	non
"w"	écriture seule, mode texte	début	oui	oui
"a"	extension, mode texte	**fin**	non	oui
"r+"	lecture écriture, mode texte	début	non	non
"w+"	lecture écriture, mode texte	début	oui	oui
"a+"	extension, mode texte	**fin**	non	oui
"rb"	lecture seule, mode binaire	début	non	non
"wb"	écriture seule, mode binaire	début	oui	oui
"ab"	extension, mode binaire	**fin**	non	oui
"rb+"	lecture écriture, mode binaire	début	non	non
"wb+"	lecture écriture, mode binaire	début	oui	oui
"ab+"	extension, mode binaire	**fin**	non	oui

Tableau 14.2 – Les différents modes d'accès pour un fichier.

14.3 Particularités

Le choix du mode d'accès est déterminent pour atteindre notre objectif (création de fichier, mise à jour ...) et éviter de détériorer voir de supprimer son contenu.
En **mode extension** (`'a'` pour append), il est possible de déplacer le pointeur de fichier, notamment pour lire une information. Cependant, lorsqu'une instruction d'écriture est appelée, **le pointeur sera systématiquement replacé à la fin actuelle du fichier avant écriture**.

La fonction fclose

```
int fclose(FILE *flux);
```

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.

Exemple 14.1. : *Ouverture/fermeture d'un flux sur un fichier texte*

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

int main(void){
    char filename[] = "texte.txt";
    FILE *f = fopen(filename, "r");

    if (f == NULL){
        printf("Le fichier %s n'a pas pu être ouvert\n", filename);
        return EXIT_FAILURE;
    }

    printf("Le fichier %s existe\n", filename);

    // Traitement à réaliser ...

    if (fclose(f) == EOF){
        perror("Erreur lors de la fermeture du flux\n");
        return EXIT_FAILURE;
    }
    f = NULL;
    return EXIT_SUCCESS;
}
```

Comportement inattendu (unexpected behavior)

Si vous rencontrez un comportement inattendu de votre programme avec les fichiers, commencez par vérifier qu'à chaque appel à `fopen()` correspond bien une fermeture avec `fclose()`.

14.4 Lecture depuis un flux de texte

Lecture d'un carcatère

```
int getc(FILE *flux);
```

Lit 1 caractère, mais le retourne sous forme d'entier. Ces deux fonctions retournent EOF si la fin de fichier est rencontrée ou si une erreur est survenue.

Lecture de plusieurs carcatères

```
int fgetc(char *chaîne, int n, FILE *flux);
```

lit n-1 caractères à partir de la position du pointeur et les range dans chaîne en ajoutant le caractère nul `'\0'`.

Lecture d'une ligne

```
char *fgets(char *tampon, int taille, FILE *flux);
```

Lit n-1 caractères à partir de la position du pointeur et les range dans chaîne en ajoutant le caractère nul `'\0'`. Retourne un pointeur nul si la fin du fichier est atteinte ou si une erreur est survenue.
Étant donné que la norme nous garantit qu'une ligne peut contenir jusqu'à 254 caractères (caractère de fin de ligne inclus), nous utiliserons un tableau de 255 caractères pour les contenir (puisqu'il est nécessaire de prévoir un espace pour le caractère nul).

Lecture d'un entier

```
int getw(FILE *flux);
```

Equivalent à `getc` avec un entier ; le pointeur avance de la taille d'un entier.

La fonction fscanf

```
int fscanf(FILE *flux, char *format, ...);
```

Equivalent à `scanf` en utilisant un `flux`. Retourne le nombre de conversions réussies (voire zéro, si aucune n'est demandée ou n'a pu être réalisée) ou EOF si une erreur survient avant qu'une conversion n'ait eu lieu.

Fin de fichier

En langage C, le dernier caractère lu est EOF (End Of File) sa valeur dans la table ASCII est 26 en décimal (c'est un caractère de contrôle). Il indinque donc qu'aucun caractère supplémentaire ne peut être lu depuis le fichier ou le flux.

Exemple 14.2. : *Boucle de lecture d'un fichier texte caractère par caratère*

```
char c ='\n';
while(c != EOF){
    c = fgetc(f);
    printf("%c", c);
}
```

Exemple 14.3. : *Boucle de lecture d'un fichier texte ligne à ligne*

```
char buffer[SIZE]={0};    // SIZE est un symbole défini à 255 (voir : lecture d'une ligne)
while( fgets(buffer, SIZE, f) ){ // Rappel : NULL = 1er caractère ASCII = '\0' = 0
    printf("%s", buffer);
}
```

14.5 Ecriture vers un flux de texte

Ecrire un caractère

```
int putc(char c, FILE *flux);
```

Ecrit la valeur de c à la position courante du pointeur, le pointeur avance d'une case mémoire.Retourne EOF en cas d'erreur.

Ecrire une ligne

```
int fputs(char *ligne, FILE *flux);
int puts(char *ligne);
```

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`.

La fonction fprintf

```
int fprintf(FILE *flux, char *format, ...);
```

La fonction `fprintf()` 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.

Exemple 14.4. : *Boucle d'écriture dans un fichier texte caractère par caratère*

```
for(int c='a'; c!='z'; c++){
    int writed = putc(c, f);
    if(writed == EOF){
        printf("Erreur d'écriture dans le fichier %s !\n", filename);
        return EXIT_FAILURE;
    }
}
```

Exemple 14.5. : *Boucle d'écriture dans un fichier texte ligne à ligne*

```
// LINES est un symbole défini à 4 et LENGHT à 255.

const char texte[LINES][LENGTH]={
    "Tout comme malloc() s'accompagne toujours de free(),\n",
    "la fonction fopen() s'accompagne toujours de fclose().\n",
    "Autre point commun : il faut impérativement vérifier le résultat\n",
    "d'une ouverture de fichier, tout comme celui d'une allocation dynamique.\n"
                    };

for(int i=0; i<LINES; i++){
    if( fputs(texte[i], f) == EOF){
        printf("Erreur d'écriture du fichier %s !\n", filename);
        return EXIT_FAILURE;
    }
}
```

Exemple 14.6. : *Ecriture puis lecture de données formatées*

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

#define N 6

int main(void){
    const char filename[]="exmp.txt";
    FILE *f = fopen(filename,"w+");
    if(f == NULL){
        printf("Erreur de création du fichier %s !\n", filename);
        return EXIT_FAILURE;
    }

    char course[N][255]= {"Méthodes", "C", "Java", "Math", "Réseau", "Anglais"};
    float exam_grade[N]= {10.0, 11.11, 12.22, 13.33, 14.44, 15.55};
    char separator = ' ';

    for(int i=0; i<N; i++){
        fprintf(f, "%s%c%f%c", course[i], separator, exam_grade[i], separator);
    }

    fseek(f, 0, SEEK_SET);

    char name[10]={0};
    float grade = -1;
    for(int i=0; i<N; i++){
        fscanf(f, "%s%*c%f", name, &grade);
        // Dans le spécificateur de format : %*c,
        // l'asterisque permet de lire un caractère sans le placer dans une variable.
        printf("%s : %.2f\n", name, grade);
    }

    if(fclose(f) == EOF){
        perror("Erreur lors de la fermeture du flux !\n");
        return EXIT_FAILURE;
    };
    f = NULL;

    return EXIT_SUCCESS;
}
```

NB : Ici les informations sont une suite continue de données respectant le même format, d'où le terme "données formatées". Les fichiers structurés (comme les CSV) sont différents puisqu'ils représentent souvent un tableau. Chaque ligne du fichier CSV est alors une ligne du tableau et chaque séparateur représente la délimitation d'une colonne. L'interprétation de fichiers structurés en langage C peut rapidement s'avérer fastidieuse sachant qu'il nous faut convertir les données enregistrées sous forme de caractères dans leur type d'origine. En langage C, il est bien plus pratique de traiter des données structurées avec des fichiers binaires !

Exercice 10. : *Premiers accès*

Créez un fichier texte avec l'éditeur de votre choix et placez-y quelques mots. Créez ensuite un programme qui :

- *1. ouvre ce fichier texte s'il existe, et dans le cas contraire signale son absence à l'utilisateur,*
- *2. affiche le premier caractère du fichier puis dénombre le nombre de fois où ce caractère est présent dans le fichier,*
- *3. affiche le nombre d'occurence(s) du premier caractère dans le fichier.*

Exercice 11. : *Lecture et ajout*

Créez avec un éditeur de texte les fichiers premier.txt et second.txt placez-y quelques lignes. Créez ensuite un programme qui lit le contenu du fichier permier.txt pour le recopier à la fin du fichier second.txt.

14.6 Lecture depuis un flux binaire

Lire un multiplet

```
int getc(FILE *flux);
int fgetc(FILE *flux);
```

Ici ces fonctions permettent de récupérer un multiplet (sous la forme d'un `unsigned char` converti en `int`) depuis un flux.

Lire une suite de multiplets

```
size_t fread(void *ptr, size_t taille, size_t nombre, FILE *flux);
```

La fonction `fread` lit le nombre d'éléments de taille multiplet depuis le `flux` et les stocke dans l'objet référencé par `ptr`. Elle retourne le nombre d'éléments lus en cas de succès ou une valeur inférieure en cas d'échec.

14.7 Ecriture vers un flux binaire

Ecrire un multiplet

```
int putc(int ch, FILE *flux);
int fputc(int ch, FILE *flux);
```

Ici ces fonctions écrivent un multiplet (sous la forme d'un `int` converti en `unsigned char`) dans le flux spécifié.

Ecrire une suite de multiplets

```
size_t fwrite(void *ptr, size_t taille, size_t nombre, FILE *flux);
```

La fonction `fwrite()` écrit le tableau référencé par `ptr` composé de `nombre` éléments de taille multiplets dans le `flux`. Elle retourne une valeur égale a `nombre` d'éléments en cas de succès et une valeur inférieure en cas d'échec.

14.8 Déplacement du pointeur de fichier

Pour les fichiers en mode d'accès direct, la fonction `fseek` permet de déplacer le pointeur de fichier sur un octet quelconque.

```
int fseek(FILE *fichier,int offset,int direction);
```

Déplace le pointeur de `offset` cases à partir de `direction`. Valeurs possibles pour direction :

- `SEEK_SET` : à partir du début du fichier.
- `SEEK_CUR` : à partir de la position courante du pointeur.
- `SEEK_END` : en arrière, à partir de la fin du fichier.

Important : bien que les fonctions fseek() et ftell() semblent simple d'utilisation, nous vous recommandons de privilégier les fonctions **fgetpos()** et **fsetpos()** qui se révèlent plus fiable à l'utilisation et se réfèrent systématiquement au début du fichier (équivalent à `SEEK_SET`).

Exemple :

```
fpos_t position;
fgetpos(f, &position);      // Relève la position initiale.
fread(&front, size, 1, f);  // Lit l'enregistrement courant puis avance d'une position.
fsetpos(f, &position);      // Retour à la position initiale.
```

14.9 Particularités

Il est permis d'exécuter plusieurs lectures consécutives. Il en va de même pour les écritures consécutives. En revanche, **il n'est pas permis d'enchainer une instruction de lecture puis d'écriture (ou l'inverse) sans vider le tampon** ou effectuer un repositionnement *(qui videra le tampon)*.

Exemple :

```
1 fread(ptr, size, 1, f);
2 fflush(f);
3 fwrite(ptr, size, 1, f);
```

Exemple 14.7. : *Ecriture puis lecture dans un fichier binaire*

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

#define MAX 50

typedef struct{
    unsigned short bloc;
    char name[MAX];
    unsigned short credits;
}course;

void file_error(FILE *f, char message[]){
    printf("%s\n", message);
    if(f != NULL){
        fclose(f);
        f = NULL;
    }
    exit(EXIT_FAILURE);
}


int main(void){
    const char filename[]="courses.dat";
    FILE *f = fopen(filename,"wb+");
    if(f == NULL){
        file_error(f, strcat("Impossible de créer le fichier ", filename));
    }

    int number_of_courses = 3;
    course my_courses[number_of_courses];
    my_courses[0] = (course) {1, "Principe de programmation 1", 8};
    my_courses[1] = (course) {1, "Initiation aux langages de programmation", 5};
    my_courses[2] = (course) {1, "Architectures des ordinateurs", 4};

    int course_size = sizeof(course);
    size_t writed = fwrite(my_courses, course_size, number_of_courses, f);
    if(writed < number_of_courses){
        file_error(f, strcat("Erreur d'enregistrement dans le fichier ", filename));
    }

    fseek(f, 0, SEEK_SET);
    // Repositionnement nécéssaire pour :
    // 1) revenir au début du fichier,
    // 2) enchainer le dernier accès en écriture puis le premier accès en lecture.

    course readed_courses[number_of_courses];
    size_t readed = fread(readed_courses, course_size, number_of_courses, f);
    if(readed < number_of_courses){
        file_error(f, strcat("Erreur de lecture dans le fichier ", filename));
    }

    fclose(f);
    f = NULL;

    for(int i=0; i<number_of_courses; i++){
        printf("Bloc %d - %s : %d crédits\n",
                readed_courses[i].bloc,
                readed_courses[i].name,
                readed_courses[i].credits);
    }

    return EXIT_SUCCESS;
}
```

Exercice 12. : *Sauvegarde de magazines*
Créez un programme qui :

- *déclare une structure magazine,*
- *instancie* n *magazines,*
- *place les* n *magazines instanciés dans un tableau de structures,*
- *enregistre le tableau de strucutres en une seule instruction dans un fichier binaire*

Exercice 13. : *Lectures de magazines*
Créez un programme qui :

- *lit en une seule instruction un fichier binaire contenant un tableau de* n *structures de type magazine,*
- *affiche à l'écran l'ensemble des magazines*