

## Chapitre 4

# Les fichiers

Jusqu'à maintenant tous les programmes que nous avons conçus travaillaient sur des données qui étaient perdues après chaque session de travail. On peut cependant, c'est bien naturel, désirer conserver des données pour plus tard : par exemple les notes des élèves pour faire la moyenne en fin de trimestre. Cette notion de *conservation des données* se concrétise sous la forme de *fichier* (au sens informatique).

Vous connaissez évidemment la notion de fichier, ne serait-ce que par les programmes source que vous conservez. Il s'agit ici de voir comment travailler avec les fichiers du point de vue de la programmation.

## 4.1 Notion générale de fichier

Définition.- *Un fichier (file en anglais) est une suite d'informations, enregistrée sur un support physique et repérée par un nom.*

Fichiers à un ou à plusieurs éléments.- Dans le cas d'un programme le fichier ne comporte intuitivement qu'un seul élément : ce programme. Dans le cas des notes, il comporte en général plusieurs notes. L'expérience montre que le cas des fichiers à plusieurs éléments est de beaucoup le plus fréquent. En informatique, la manipulation des fichiers sera presque toujours orientée vers ceux-ci, le premier cas, celui d'un fichier à un seul élément, pouvant se concevoir comme cas particulier. Le nom même retenu, « fichier », fait d'ailleurs tout de suite penser au cas de plusieurs éléments, un ensemble de « fiches » ayant un dénominateur commun.

Manipulations à effectuer sur un fichier.- Les manipulations concernant un fichier sont manifestement les suivantes :

- 1°) **créer** un fichier : le déclarer pour qu'il existe et l'initialiser ;
- 2°) **écrire** dans ce fichier ;
- 3°) **lire** ce qui y est écrit : sinon à quoi peut-il servir ?
- 4°) le **modifier** : nous avons bien vu à propos de l'écriture des programmes la nécessité de cette opération ;
- 5°) le **détruire** lorsqu'on n'en a plus besoin, sinon la mémoire risque vite d'être saturée.

## 4.2 Un premier exemple

Introduction.- Commençons par donner un exemple d'utilisation de fichier en langage C que nous commenterons au fur et à mesure après.

Le problème.- Établissons un répertoire constitué de noms et des numéros de téléphone correspondants.

Précisions.- Nous allons considérer un fichier dont chaque élément comprendra moralement deux éléments : le nom, qui sera un mot de moins de vingt-neuf lettres, et le numéro de téléphone correspondant, qui sera également considéré comme un mot (à cause des parenthèses et des tirets que certains placent) de moins de dix-neuf lettres. Commençons par initialiser ce répertoire par un certain nombre de données. Rentrons-les au clavier avec pour valeur sentinelle un mot commençant par le caractère '#' pour le nom.

Un programme.- Ceci nous conduit au programme suivant dont nous commenterons les instructions nouvelles après :

```
/* rep_1.c */

#include <stdio.h>

void main(void)
{
    char nom[30];
    char tel[20];
```

```

FILE *fich;
char nom_fich[20];
printf("\nNom du fichier repertoire : ");
gets(nom_fich);
fich = fopen(nom_fich, "w");
printf("Nom : ");
gets(nom);
while (nom[0] != '#')
{
    printf("Telephone : ");
    gets(tel);
    fprintf(fich,"%30s %20s", nom, tel);
    printf("Nom : ");
    gets(nom);
}
fclose(fich);
}

```

## 4.3 Déclaration des fichiers

En langage C on peut travailler sur les fichiers à deux **niveaux** : le **niveau bas**, utilisant directement les routines du système d'exploitation, et le **niveau haut**, indépendant du système d'exploitation. Nous n'étudierons que le niveau haut ici, le niveau bas se rencontrant dans le cours de programmation système.

### 4.3.1 Fichier logique et fichier physique

Il faut faire une différence entre le *fichier logique* et le *fichier physique*. Le **fichier logique** est un nouveau type de données. Le **fichier physique** est le fichier matériel placé en mémoire centrale ou, évidemment le plus souvent, en mémoire de masse ; le programmeur n'a pas à savoir comment il est constitué, il doit seulement en connaître le *nom physique*, qui est une chaîne de caractères, dont les règles de formation dépendent du système d'exploitation sur lequel on travaille.

#### 4.3.1.1 Fichier logique

Introduction.- Un fichier logique est vu comme constitué d'un ensemble d'éléments d'un même type, rangé chacun dans un composant du fichier.

En fait en langage C, contrairement à d'autres langages, le seul constituant est l'octet (ou caractère, ce qui revient au même).

Cependant on ne lit pas (et on n'écrit pas) octet par octet de façon physique : ceci serait trop long vu les temps d'accès aux mémoires de masse. On lit donc à travers un **tampon** (*buffer* en anglais) comprenant un certain nombre d'octets, ce nombre dépendant du système d'exploitation et du compilateur.

La manipulation de ce tampon se fait par l'intermédiaire d'une variable structurée du type prédéfini FILE. La structure FILE dépend du système d'exploitation et est définie dans le fichier en-tête `stdio.h`. Sa définition précise ne présente pas vraiment d'intérêt pour la programmation à haut niveau.

Remarque.- On notera bien la graphie de `FILE` avec des majuscules, ce qui est assez rare en langage C.

Pointeur tampon.- Pour accéder concrètement à un fichier il faut utiliser un pointeur vers une variable de type `FILE`. On déclare par exemple :

```
FILE *fich;
```

La variable `fich` jouera alors le rôle du *nom logique* du fichier.

Commentaire.- Le nom logique sera en langage C cette variable `fich`, qui est un identificateur, non utilisé pour autre chose comme d'habitude.

#### 4.3.1.2 Fichier physique

Introduction.- Le nom physique du fichier dépend du système d'exploitation. Chaque système d'exploitation comprend une partie permettant de gérer les mémoires de masse, c'est même la partie essentielle pour les micro-ordinateurs d'où le nom de **SED** (*Système d'Exploitation de Disquettes*; en anglais **DOS** pour *Disk Operating System*) qui est donnée à leurs systèmes d'exploitation.

Le programmeur doit pouvoir ignorer la technologie. Il a juste à connaître le minimum, c'est-à-dire comment manipuler les fichiers (physiques) à travers la partie du système d'exploitation qui lui est consacrée. Il appartient au concepteur du compilateur de s'arranger pour que cela fonctionne.

Désignation du fichier physique.- Le fichier physique est désigné par une expression chaîne de caractères suivant les règles propres au système d'exploitation. Pour MS-DOS, par exemple, un fichier sur disque dur sera, par exemple :

```
c :\tc\DONNEES\REPERT.DAT
```

Il ne faut pas oublier de l'écrire suivant les règles du langage C, c'est-à-dire :

```
''c :\\tc\\DONNEES\\REPERT.DAT''
```

si on l'écrit directement comme constante dans le corps du programme.

#### 4.3.2 Lien entre les deux désignations et ouverture

Introduction.- Le fichier logique et le fichier physique n'ont pas nécessairement le même nom. Une première raison est que le nom du fichier logique est un identificateur (au sens du langage C) alors que le nom du fichier physique peut ne pas l'être; c'est d'ailleurs le cas avec MS-DOS. Une deuxième raison est de permettre une meilleure portabilité : vous écrivez un programme en langage C pour tel système informatique puis, pour une raison ou une autre, vous en changez ; il n'est pas nécessaire de tout réécrire, seuls les noms des fichiers physiques auront à être changés dans notre cas.

La première opération avec les fichiers est donc de créer un lien entre le nom logique et le nom physique du fichier. Ceci se fait en langage C au moment de l'ouverture du fichier.

Syntaxe.- L'ouverture d'un fichier s'effectue grâce à la fonction `fopen()`, déclarée dans le fichier en-tête `<stdio.h>`, de prototype :

```
FILE *fopen(char *nom, char *mode);
```

où `nom` est une variable pointeur vers une chaîne de caractères, le *nom physique* du fichier, et `mode` un pointeur vers une autre chaîne de caractères, qui définit le **mode d'accès**.

Modes d'accès.- Les différents modes d'accès sont, pour l'instant, les suivants :

- "r" : ouverture du fichier en lecture ('r' pour *read*) seulement.  
La fonction `fopen()` retourne le pointeur `null` si le fichier n'existe pas (ou est introuvable).
- "w" : ouverture du fichier en écriture ('w' pour *write*) seulement.  
Le fichier est créé s'il n'existe pas encore. S'il existe déjà, le contenu est écrasé et perdu.
- "a" : ouverture du fichier pour ajouter ('a' pour *append*) des données, c'est-à-dire ouvrir en écriture à la fin du fichier.  
Le fichier est créé s'il n'existe pas.

Exemple.- Pour ouvrir en écriture le fichier `repert.dat` situé sur le lecteur de disquette `a :`, on écrira :

```
FILE *fich ;
char nom[20] ;
nom = "a :repert.dat" ;
fich = fopen(nom,"w") ;
```

Remarque.- Le nom logique d'un fichier n'a d'importance qu'à l'intérieur d'un programme donné. On pourra réutiliser le fichier physique dans un autre programme, avec un nom logique différent.

### 4.3.3 Fermeture des fichiers

Introduction.- La fermeture d'un fichier s'effectue par la fonction prédéfinie `fclose()`, déclarée dans le fichier en-tête `<stdio.h>`, de prototype :

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

où `fich` est un pointeur tampon.

La fonction `fclose()` retourne la valeur 0 si on a pu fermer le fichier rattaché à `fich`. La valeur de retour EOF indique une erreur.

La valeur EOF.- La valeur EOF (pour l'anglais *End Of File*) est définie comme valant -1 dans le fichier `stdio.h`. Ceci explique pourquoi le caractère est de type `int` et non pas `char`.

Exemple.- Pour fermer le fichier ci-dessus, on utilisera l'instruction :

```
fclose(fich) ;
```

Remarque.- On peut renoncer à utiliser la fonction `fclose()` dans les programmes puisque, dans un déroulement de programme normal et dépourvu d'erreurs, `fclose()` est appelée automatiquement à la fin du programme pour fermer tous les fichiers encore ouverts.

## 4.4 Fichier séquentiel

Dans les langages de programmation modernes, tel que le langage C, on n'est pas concerné par la structure physique des fichiers, et en particulier par le mode physique d'accès à ceux-ci : *accès séquentiel* ou *accès direct*. Tout fichier, quel que soit sa structure physique, peut être considéré à la fois comme un fichier séquentiel et comme un fichier à accès direct. Bien entendu ces accès sont émulés et donc le temps d'accès peut s'en ressentir, mais c'est un point que le programmeur ne prend pas en compte à ce niveau.

### 4.4.1 Notion générale

L'idée que l'on peut se faire d'un **fichier séquentiel** est la suivante : on a un ruban sur lequel on place un composant, appelons-le le premier composant, puis un autre composant, le deuxième composant, puis un autre, le troisième, et ainsi de suite jusqu'à ce que l'on s'arrête.

l'exemple type est celui d'une bande magnétique, que ce soit la bande d'un magnétophone, d'un lecteur de bandes pour les gros ordinateurs ou d'une bande pour la sauvegarde des données.

### 4.4.2 Instructions fondamentales pour un fichier séquentiel

#### 4.4.2.1 Écriture dans un fichier séquentiel

Les instructions fondamentales pour écrire pour la première fois dans un fichier séquentiel (et non pour modifier un composant) sont au nombre de quatre :

- **ouverture** du fichier (en écriture) : cette instruction joue deux rôles différents suivant que le fichier physique existe déjà ou non ; s'il n'existe pas on le crée et on se prépare à écrire le premier composant, disons en plaçant un **index** devant celui-ci ; s'il existe déjà on le détruit et on commence également à écrire le premier élément (et non à en ajouter d'autres).
- **écriture** proprement dite, c'est-à-dire placer un nouveau composant dans le fichier physique et se préparer à en placer un autre, c'est-à-dire placer l'index devant l'élément suivant.
- **fermeture** : ceci indique que le fichier n'est plus accessible en écriture, mais surtout permet de déclencher un certain nombre d'actions liées à la technologie et au système d'exploitation (par exemple en conserver la taille dans un endroit approprié) pour pouvoir réutiliser ce fichier ensuite. Ne pas fermer un fichier dans une session peut le rendre inutilisable à la session suivante.
- **ajout** : cette instruction a pour rôle d'ouvrir un fichier existant et de se placer après le dernier composant existant pour se préparer à écrire le suivant (et non à recouvrir le premier élément).

#### 4.4.2.2 Lecture dans un fichier séquentiel

Les instructions fondamentales pour lire dans un fichier séquentiel sont au nombre de trois :

- **ouverture** (en lecture) : le fichier physique doit exister ; cette instruction se prépare à lire le premier composant en plaçant l'index devant celui-ci.
- **lecture** proprement dite, c'est-à-dire lire un nouveau composant dans le fichier physique et se préparer à lire le suivant s'il existe.
- **fermeture** : ceci indiquera que le fichier n'est plus accessible en lecture.

Il existe de plus une fonction fondamentale, celle de **détection de fin de fichier**. Elle est importante pour éviter d'essayer de lire un composant qui n'existe pas.

### 4.4.3 Opérations de lecture et d'écriture de caractères

#### 4.4.3.1 Syntaxe

Introduction.- Pour lire et écrire des données dans un fichier, en langage C on dispose de fonctions analogues à celles qui servent à saisir des données au clavier et à les afficher à l'écran. Ces opérations peuvent être effectuées caractère par caractère, par ligne ou avec formatage. Commençons par les opérations caractère par caractère, c'est-à-dire octet par octet.

Les fonctions.- On écrit et lit un caractère dans un fichier grâce aux fonctions respectives `fputc()` et `fgetc()` déclarées dans le fichier en-tête `stdio.h`. La fin d'un fichier est détectée par la fonction `feof()`. Les prototypes respectifs de ces fonctions sont :

```
int fputc(int caractere, FILE *fich);
int fgetc(FILE *fich);
int feof(FILE *fich);
```

où *caractere* et *fich* sont des identificateurs, désignant respectivement le caractère à écrire et le nom logique du fichier.

La valeur de retour de `fputc()` est le caractère écrit ou EOF en cas d'erreur. On ne s'en préoccupera donc pas en général.

La fonction `fgetc()` retourne le caractère lu sous la forme d'une valeur de type `int`. Si la valeur renvoyée est EOF, c'est que la fin du fichier a été atteinte ou qu'il y a eu une erreur.

La valeur de retour de `feof()` est non nulle si la fin du fichier a été atteinte et zéro sinon.

#### 4.4.3.2 Exemples

Exemple 1.- Écrivons un programme C qui demande le nom d'un fichier texte et affiche alors ce texte à l'écran (l'analogue de la commande `type` de MS-DOS ou `cat` d'UNIX). Rappelons que la fin d'un fichier texte est CTRL-Z de code ASCII 26 en MS-DOS.

```
/* fich_2.c */
#include <stdio.h>

void main(void)
{
    FILE *fich;
    char c, nom[30];
    printf("Nom du fichier a lire : ");
    gets(nom);
    fich = fopen(nom,"r");
    c = fgetc(fich);
    while ((!feof(fich)) && (c != '\26')) /* non portable */
    {
        if (c != '\26') putchar(c);    /* pour MS-DOS */
        c = fgetc(fich);
    }
    fclose(fich);
}
```

Exemple 2.- Écrivons un programme C qui demande le nom d'un fichier (texte), lit un texte au clavier et le place dans ce fichier. Le texte ne comportera pas le symbole '#' qui sera la valeur signal pour la fin du texte. Nous allons utiliser la fonction `getche()` des plate-formes MS-DOS, le programme ne sera donc pas portable.

```

/* fich_3.c */
#include <stdio.h>
#include <conio.h> /* non portable */

void main(void)
{
    FILE *fich;
    char c, nom[30];
    printf("\nNom du fichier de sauvegarde : ");
    gets(nom);
    fich = fopen(nom, "w");
    printf("Entrez votre texte :\n\n");
    c = getche(); /* non portable */
    while (c != '#')
    {
        if (c == '\r') /* non portable */
        {
            fputc('\n', fich);
            putchar('\n');
        }
        else fputc(c, fich);
        c = getche(); /* non portable */
    };
    fclose(fich);
    printf("\nVotre texte est enregistré.\n");
}

```

#### 4.4.4 Opérations de lecture et d'écriture de chaînes de caractères

Les fonctions.- Les fonctions `fgets()` et `fputs()`, toujours déclarées dans `stdio.h`, permettent de lire et d'écrire des chaînes de caractères dans un fichier. Les prototypes de ces fonctions sont :

```

char *fgets(char *tampon, int entier, FILE *fich) ;
char *fputs(char *tampon, FILE *fich) ;

```

où *tampon* est la variable chaîne de caractères qui permet de faire transiter les informations, *entier* la longueur (maximale) du contenu de cette variable et *fich* le nom logique du fichier.

##### Exemple.- (Copie de fichier texte)

*Écrivons un programme C qui demande le nom (physique) d'un fichier texte source (existant) et le nom d'un fichier texte but, et qui copie alors le contenu du fichier source dans le fichier but.*

```

/* copie.c */
#include <stdio.h>

void main(void)
{
    FILE *source, *but;
    char nom1[30], nom2[30], ligne[81];
/* initialisation */
    printf("Nom du fichier a copier : ");

```

```

    gets(nom1);
    source = fopen(nom1, "r");
    printf("Nom du fichier but : ");
    gets(nom2);
    but = fopen(nom2, "w");
/* copie */
    fgets(ligne, 81, source);
    while (!feof(source))
    {
        fputs(ligne, but);
        fgets(ligne, 81, source);
    }
/* fermeture des fichiers */
    fclose(source);
    fclose(but);
    printf("\nLa copie est terminée.\n");
}

```

Exercice.- Réécrire les exemples de la section précédente en manipulant des chaînes de caractères au lieu de caractères.

#### 4.4.5 Opérations de lecture et d'écriture formatées

Les fonctions.- Les fonctions `fprintf()` et `fscanf()`, déclarées dans `stdio.h`, permettent d'écrire et de lire des données formatées de façon à passer, par exemple, des structures champ par champ. Les prototypes de ces fonctions sont :

```

int fprintf(FILE *fich, char *format, ...);
int fscanf(FILE *fich, char *format, ...);

```

où *fich* est le nom logique du fichier, *format* l'indication des formats des données avec les '%', suivi du nombre d'argument qu'il faut.

La syntaxe et la sémantique de ces fonctions se déduisent de celles indiquées pour les fonctions `printf()` et `scanf()`.

Remarque.- Pour la fonction `scanf()`, nous avons fortement insisté sur le fait qu'il ne faut entrer qu'une donnée à la fois, précédée d'un prompteur pour que l'utilisateur sache ce qu'il doit faire. Ce n'est pas le cas pour `fscanf()` car l'ordinateur n'a pas besoin de prompteur (*sic*).

Exemple.- Nous avons donné en premier exemple l'initialisation d'un répertoire téléphonique. Considérons le programme suivant qui permet d'afficher le fichier physique créé par ce programme :

```

/* rep_2.c */
#include <stdio.h>

void main(void)
{
    FILE *repert;
    char fich[30], nom[30], tel[20];
    printf("Nom du fichier repertoire : ");
    gets(fich);
}

```

```

repert = fopen(fich, "r");
printf("Nom                Telephone");
printf("\n\n");
fscanf(repert,"%30s %20s", nom, tel);
while (!feof(repert))
{
    printf("%30s : %20s \n", nom, tel);
    fscanf(repert,"%30s %20s", nom, tel);
}
fclose(repert);
}

```

Remarque.- Rappelons qu'avec la fonction `scanf()` (et donc aussi `fscanf()`) un espace termine la lecture d'une chaîne de caractères. On ne doit donc pas utiliser d'espaces dans les données ci-dessus, ni pour mettre NOM et PRÉNOM à la place du nom, ni pour séparer les chiffres du numéro de téléphone de deux en deux.

Autre façon.- L'inconvénient pédagogique de l'exemple précédent est qu'on n'utilise que des chaînes de caractères, donc les entrées-sorties formatées ne se justifient pas pleinement. Réécrivons ces programmes en déclarant le numéro de téléphone comme un entier. Pour terminer l'initialisation on entrera un nom commençant par le caractère '#'.  
 Pour l'initialisation on a le programme suivant :

```

/* Programme rep_3.c */
#include <stdio.h>

void main(void)
{
    FILE *repert;
    char fich[30], nom[30];
    long tel;
    printf("\nNom du fichier pour le repertoire : ");
    gets(fich);
    repert = fopen(fich, "w");
    printf("Nom          : ");
    gets(nom);
    while (nom[0] != '#')
    {
        printf("Telephone : ");
        scanf("%ld", &tel);
        fprintf(repert,"%30s %ld", nom, tel);
        printf("Nom : ");
        scanf("%s", nom);
    }
    fclose(repert);
    printf("Votre fichier est enregistre\n");
}

```

Remarquons que l'on ne peut pas saisir un numéro de téléphone qui commence par '0'.

Pour la lecture on a le programme suivant :

```
/* Programme rep_4.c */
#include <stdio.h>

void main(void)
{
    FILE *repert;
    char fich[30], nom[20];
    long tel;
    printf("\nNom du fichier repertoire : ");
    gets(fich);
    repert = fopen(fich, "r");
    printf("\nNom                : Telephone");
    printf("\n\n");
    while (!feof(repert))
    {
        fscanf(repert,"%30s %ld", nom, &tel);
        printf("%30s %10ld \n\n", nom, tel);
    }
    fclose(repert);
}
```

#### 4.4.6 Recherche séquentielle ou consultation

Introduction.- Une des opérations fondamentales à propos des fichiers en est la **consultation**. On peut, bien sûr, vouloir voir défiler l'ensemble des composants du fichier mais ceci est assez rare. En général on s'intéresse à un composant que l'on repère grâce à une **clé**.

Exemple.- Considérons notre fichier *repertoire*. Au début, lorsqu'il n'est pas très important, il peut être agréable de faire défiler l'ensemble des composants à chaque fois que l'on recherche un numéro de téléphone, en pensant à chacune des personnes répertoriées. Mais lorsqu'il commence à comporter quelques centaines d'éléments, cela devient moins amusant et risque de ne pas être très rapide. On va donc chercher à connaître soit le numéro de téléphone d'une personne donnée (le cas le plus fréquent), soit le nom connaissant le numéro de téléphone (par exemple après un message du genre « rappeler au numéro suivant », la situation pouvant être assez délicate si on ne sait pas d'avance à qui on téléphone).

Il faut donc commencer par choisir une clé : le nom ou le numéro de téléphone. On pourrait laisser le choix, mais nous déciderons ici qu'il s'agit du nom. Le problème revient alors, lorsqu'on a entré un nom, à afficher le numéro de téléphone correspondant en s'aidant du fichier en question.

Il ne faut pas oublier que l'on peut demander par mégarde une personne qui n'est pas répertoriée; ce cas doit donc être prévu.

L'algorithme va être très simple. On passe en revue chacun des composants, on compare le nom correspondant à celui qui est demandé; dès qu'on le rencontre on donne le numéro de téléphone correspondant et on s'arrête (on ne tiendra pas compte des **doublons** ici); si on ne l'a pas trouvé en arrivant à la fin du fichier il faut l'indiquer.

Le programme.- On peut utiliser le programme suivant :

```
/* Programme rep_5.c */
#include <stdio.h>
#include <string.h>

void main(void)
{
    char fich[20], nom[30], name[30];
    char tel[20];
    FILE *repert;
    int  trouve;
    printf("\nNom du fichier repertoire : ");
    gets(fich);
    repert = fopen(fich, "r");
    printf("Quel est le nom recherche : ");
    scanf("%s", name);
    trouve = 0;
    while ((!feof(repert)) && (!trouve))
    {
        fscanf(repert, "%30s %20s", nom, tel);
        if (! strcmp(nom, name))
        {
            trouve = 1;
            printf("Son numero de telephone est : %s\n", tel);
        }
    }
    if (! trouve) printf("Non repertorie. \n");
    fclose(repert);
}
```

#### 4.4.7 Ajout d'éléments

Introduction.- L'ajout d'un ou de plusieurs éléments dans notre fichier répertoire est facile. Il suffit pratiquement de reprendre le même programme que le programme d'initialisation en remplaçant le mode d'accès "w" par le mode d'accès "a".

Exercice.- (Ajout)

*Écrire un programme C d'ajout de notre fichier répertoire ci-dessus. On ajoutera des noms et le numéro de téléphone correspondant à la suite du fichier déjà existant.*

### 4.4.8 Modification et suppression d'éléments

L'accès séquentiel ne permet ni de modifier ni de supprimer un élément de façon simple. Si on veut vraiment le faire par des opérations séquentielles pures il faut créer un fichier auxiliaire, que l'on pourra renommer avec le nom de l'ancien fichier à un certain moment. Nous renvoyons aux exercices pour cela.

#### Exercice 1.- (Modification)

*Écrire un programme C de modification d'un élément de notre fichier répertoire ci-dessus. L'utilisateur doit entrer un nom, le programme doit alors afficher l'ancien numéro de téléphone correspondant (ou « inexistant » s'il n'apparaissait pas dans le fichier), demander le nouveau numéro de téléphone et l'enregistrer.*

[ On utilisera un fichier auxiliaire. ]

#### Exercice 2.- (Suppression)

*Écrire un programme C de suppression d'un élément de notre fichier répertoire ci-dessus. L'utilisateur doit entrer un nom, le programme doit alors supprimer tous les éléments du fichier dont le premier champ est ce nom et terminer en indiquant que la suppression a été effectuée.*

## 4.5 Fichier à accès direct

### 4.5.1 Notion générale

Introduction.- L'idée générale que l'on peut se faire d'un **fichier à accès direct** est la suivante : on a une **clé** principale et tout composant est repéré grâce à cette clé principale ; pour accéder à un composant donné il suffit d'indiquer le contenu de sa clé principale. L'intérêt est qu'on n'a pas besoin de passer en revue les composants un par un pour accéder à un composant donné.

On peut émuler cet accès direct à un fichier pour un fichier séquentiel. Cependant certains supports de fichiers physiques permettent réellement cet accès direct : c'est le cas des mémoires vives à tore de ferrite puis à semi-conducteurs, des mémoires de masse que sont les tambours puis les disquettes puis les disques durs.

Cas du langage C.- La possibilité de l'accès direct est en fait assez rudimentaire dans le cas du langage C. En effet on n'a pas le choix de la nature de la clé principale. Chaque composant d'un fichier, un octet rappelons-le, est systématiquement repéré par un numéro, la numérotation commençant à 0 (et non à 1) et c'est cette seule clé qui sera utilisée.

On peut bien entendu émuler d'autres clés, à condition que cela revienne à utiliser celle-ci *in fine*.

### 4.5.2 Instructions fondamentales pour un fichier à accès direct

#### 4.5.2.1 Lecture dans un fichier à accès direct

Les instructions fondamentales pour lire dans un fichier à accès direct sont au nombre de quatre :

- ouverture en lecture du fichier ;
- **positionnement** devant le composant recherché ;
- lecture proprement dite du composant ;
- fermeture en lecture du fichier.

Il doit de plus exister une **fonction de détection** pour indiquer que, éventuellement, le contenu de la clé principale ne correspond pas à un composant existant.

#### 4.5.2.2 Écriture dans un fichier à accès direct

L'écriture dans un fichier à accès direct doit pouvoir se faire à n'importe quel numéro de composant, que le précédent existe déjà ou non. Ceci permet de modifier et de compléter un fichier.

Les instructions fondamentales pour écrire dans un fichier à accès direct sont au nombre de quatre :

- ouverture en écriture du fichier ;
- positionnement devant le composant sur lequel on va écrire ;
- écriture proprement dite du composant ;
- fermeture en écriture du fichier.

### 4.5.2.3 Accès direct et fichiers texte

L'accès direct pour les fichiers texte est peu recommandé à cause des problèmes dus aux représentations différentes des fins de lignes dans le tampon (caractère de saut de ligne) et dans le fichier lui-même (combinaison des caractères de saut de ligne et de retour-chariot, par exemple en MS-DOS).

## 4.5.3 Manipulation d'un fichier à accès direct

### 4.5.3.1 Déclaration, ouverture et fermeture, lecture et écriture

Déclaration.- La déclaration d'un fichier, qu'il soit en accès séquentiel ou en accès direct, se fait de la même façon, en utilisant le type FILE.

Ouverture.- L'ouverture d'un fichier s'effectue de la même façon, que ce soit en accès séquentiel ou en accès direct, grâce à la fonction :

```
FILE *fopen(char *nom, char *mode) ;
```

mais on peut rajouter les trois modes d'accès suivants (déclarables en accès séquentiel, mais peu manipulables alors, car on voit mal comment lire et écrire à la fois en séquentiel) :

- "r+" : ouverture du fichier en lecture et en écriture.  
La fonction `fopen()` retourne le pointeur `null` si le fichier n'existe pas (ou est introuvable).
- "w+" : ouverture du fichier en lecture et en écriture.  
Le fichier est créé s'il n'existe pas encore. S'il existe déjà, le contenu est écrasé et perdu.
- "a+" : Ouverture du fichier en lecture et en ajout. Le fichier est créé s'il n'existe pas.

Fermeture.- On utilise la même fonction que pour les fichiers séquentiels, à savoir `fclose()`.

Opérations de lecture et d'écriture.- On utilise les mêmes fonctions que pour l'accès séquentiel, à savoir les fonctions `fputc()`, `fgetc()`, `feof()`, `fgets()`, `fputs()`, `fprintf()` et `fscanf()`.

### 4.5.3.2 Positionnement

Positionnement.- On se positionne devant le *i*-ième octet, en commençant par l'octet composant d'indice 0.

La fonction de positionnement devant un octet donné est la fonction de prototype suivant :

```
int fseek(FILE *fich, long index, int base) ;
```

où *fich* est le nom logique du fichier, *index* le nombre d'octets dont il faut se déplacer (vers la fin du fichier s'il est positif, vers le début du fichier s'il est négatif) et *base* le numéro de l'octet à partir duquel il faut se déplacer.

Il y a trois valeurs possibles pour la base :

Signification	Valeur de la base	Constante symbolique
En début de fichier	0	SEEK_SET
Position actuelle dans le fichier	1	SEEK_CUR
En fin de fichier	2	SEEK_END

Les constantes symboliques sont définies dans le fichier en-tête `stdio.h`.

La fonction `fseek()` retourne la valeur 0 si le positionnement souhaité a pu se faire. En cas d'erreur elle renvoie un entier non nul.

Retour en début de fichier.- Pour revenir au début du fichier, disons de nom logique `fich`, on peut utiliser l'instruction suivante :

```
fseek(fich, 0L, 0);
```

(en remarquant bien la façon d'indiquer une constante de type `long` avec le `L` final). Il existe aussi une fonction prédéfinie `rewind()` (c'est-à-dire *rembobiner*) de prototype :

```
void rewind(FILE *fich);
```

ce qui nous permet donc aussi d'utiliser :

```
rewind(fich);
```

Détermination de la position de l'index.- La fonction `ftell()` (*dire*, sous-entendu *où en est le fichier*) permet de connaître la position de l'index. Son prototype est :

```
long ftell(FILE *fich);
```

En cas d'erreur cette fonction renvoie la valeur `-1L`.

#### 4.5.4 Exemples

Exemple 1.- Le programme suivant, se servant du fichier répertoire créé ci-dessus, demande un numéro de composant et en indique le contenu s'il existe.

```
/* rep_6.c */
#include <stdio.h>

void main(void)
{
    FILE *repert;
    char fich[30], nom[20], tel[20];
    long numero;
    printf("\nNom du fichier pour le repertoire : ");
    gets(fich);
    repert = fopen(fich, "r");
    printf("Numero de composant : ");
    scanf("%ld", &numero);
    fseek(repert, 51*(numero - 1L), 0);
    fscanf(repert, "%30s %20s", nom, tel);
    printf("Nom : %s, telephone : %s.\n", nom, tel);
    fclose(repert);
}
```

Exemple 2.- (**Taille d'un fichier**)

Pour déterminer la taille d'un fichier (en octets) il suffit de se placer en fin de fichier et de récupérer le numéro de l'index. C'est ce qu'on fait dans le programme suivant.

```
/* rep_7.c */
#include <stdio.h>

void main(void)
{
    FILE *repert;
    char fich[30];
```

```

long taille;
printf("Nom du fichier : ");
gets(fich);
repert = fopen(fich, "r");
fseek(repert, 0L, 2);
taille = ftell(repert);
printf("La taille du fichier est de %ld octets.\n", taille);
fclose(repert);
}

```

**Exemple 3 (Modification de composant)**

Le programme suivant demande le nom du composant à modifier et permet de modifier le numéro de téléphone correspondant :

```

/* rep_8.c */
#include <stdio.h>
#include <string.h>

void main(void)
{
    FILE *repert;
    char fich[30], nom[30], tel[20], name[30];
    long numero;
    int trouve;

    printf("\nNom du fichier pour le repertoire : ");
    gets(fich);
    repert = fopen(fich, "r+");

    printf("Nom : ");
    scanf("%s", name);
    numero = -1L;
    trouve = 0;
    while ((!trouve) && (!feof(repert)))
    {
        numero++;
        fscanf(repert, "%30s %20s", nom, tel);
        if (!strcmp(nom, name))
        {
            trouve = 1;
            printf("\nAncien numero : %s", tel);
            printf("\nNouveau numero : ");
            scanf("%s", tel);
            fseek(repert, 51*(numero - 1L), 0);
            fprintf(repert, "%30s %20s", nom, tel);
            printf("La modification est effectuee.\n");
        }
    }
    if (!trouve) printf("\nNom non repertorie.\n");
    fclose(repert);
}

```

```

}
```

#### Exemple 4 (Lecture hors du fichier)

Reprenons notre fichier déjà créé. Écrivons un nouveau composant en dehors des limites (par exemple de numéro *taille* + 2) et essayons de lire un composant non initialisé (par exemple le numéro *taille* + 1, pour l'ancienne valeur de *taille* bien sûr) :

```

/* rep_9.c */
#include <stdio.h>

void main(void)
{
    FILE *repert;
    char fich[30], nom[30], tel[20];
    long numero;
    printf("\nNom du fichier pour le repertoire : ");
    gets(fich);
    repert = fopen(fich, "r+");
    fseek(repert, 0L, 2);
    numero = ftell(repert);
    printf("Nom : ");
    scanf("%s", nom);
    printf("Telephone : ");
    scanf("%s", tel);
    fseek(repert, numero + 2*51, 0);
    fprintf(repert, "%30s %20s", nom, tel);
    fseek(repert, numero + 51, 0);
    fscanf(repert, "%30s %20s", nom, tel);
    printf("Le composant non initialise est : \n");
    printf("%s : %s.\n", nom, tel);
    fclose(repert);
}

```

### 4.5.5 Suppression de composants

Introduction.- Il n'existe pas réellement de manière simple de supprimer un composant dans un fichier.

La première idée, naturelle, est que ce composant ne doit plus occuper de place dans le fichier physique. Il est alors nécessaire de compresser le fichier :

- on peut créer un nouveau fichier dans lequel on recopie l'ancien, sauf le composant concerné. Ceci se fait de façon purement séquentielle.
- on peut déplacer d'une position vers l'arrière tous les composants situés après celui à supprimer.

Ces méthodes ont l'inconvénient d'être assez lentes.

La deuxième idée est d'accepter qu'un composant, supposé ne plus exister, continue à occuper une place dans le fichier physique. Encore faut-il pouvoir identifier ces composants inexistants. On peut choisir, par exemple :

- l'écriture d'une valeur spéciale dans celui-ci (par exemple 'XXXXX' dans le champ nom, ce qui a peu de chance de nous gêner);

- la gestion d'une table des composants inexistants, cette table devant être conservée dans le fichier lui-même.

Exercice 1.- *Écrire un programme C permettant de supprimer un ou plusieurs composants en utilisant un fichier auxiliaire.*

Exercice 2.- *Écrire un programme C permettant de supprimer un ou plusieurs composants en déplaçant les composants.*

Exercice 3.- *Écrire un programme permettant de supprimer un ou plusieurs composants en utilisant une valeur spéciale.*

## 4.6 Fichiers texte

### 4.6.1 Notion de fichier texte

La plupart des systèmes d'exploitation possèdent un type de fichiers particulier, les **fichiers texte**. Comme son l'indique, un tel fichier ne contient *a priori* qu'un seul composant qui est un *texte*, c'est-à-dire qu'il est formé de caractères affichables à l'écran (on dit aussi *imprimables*). On peut manipuler de tels fichiers en particulier grâce à n'importe quel éditeur de textes. Les programmes source du langage C, par exemple, sont des fichiers texte. Les fichiers texte s'opposent quelquefois à tous les autres fichiers, dénommés alors **fichiers binaires** car ils semblent constitués d'une suite de 0 et de 1, qui ne donnent rien de compréhensible lorsqu'on essaie de les lire grâce à un éditeur de textes. Ceci est dû bien entendu à la façon de coder les données : un éditeur de textes décode correctement les caractères mais non les autres types de données.

Les fichiers texte peuvent être vus comme constitués soit d'un seul composant, soit de plusieurs composants, chaque composant étant réduit à un caractère. En fait dans ce dernier cas certains caractères particuliers (dits **caractères non affichables**) permettent de lui conférer une structure de *ligne* de longueur variable.

### 4.6.2 Les fichiers texte en MS-DOS

Introduction.- La définition des fichiers texte ne dépend pas du langage de programmation mais du système d'exploitation, puisqu'ils doivent pouvoir être lus par n'importe quel éditeur de textes conçu pour celui-ci. Nous insistons, dans ce cours, sur les notions générales mais, à chaque fois qu'il est nécessaire de choisir un exemple, nous faisons référence au système informatique défini au début. Puisque notre système d'exploitation est MS-DOS (rappelons que *Windows* n'est au départ qu'une extension graphique), il faut donc savoir comment les fichiers texte sont définis en MS-DOS.

En fait il n'est pas réellement besoin d'entrer dans les détails. Il suffit juste de faire la liaison entre la manipulation des fichiers en langage C et les fichiers texte en MS-DOS. Seul le concepteur du compilateur a besoin d'en savoir plus.

Structure.- 1<sup>o</sup>) Les fichiers texte de MS-DOS sont ceux qui peuvent être visualisés grâce aux commandes **type** ou **print** du système d'exploitation. Chaque octet d'un tel fichier représente un caractère codé en ASCII.

2<sup>o</sup>) La fin d'une ligne est indiquée par les deux caractères (non affichables) consécutifs :

- celui de retour-chariot, noté **CR** (pour *Carriage Return* en anglais), de code ASCII 13,
- celui de saut de ligne, noté **LF** (pour *Line Feed* en anglais), de code ASCII 10.

La succession de ces deux caractères provoquent, en MS-DOS, un retour en début de ligne suivi d'un saut de ligne, ce qui correspond bien à passage à la ligne suivante.

3<sup>o</sup>) La fin d'un fichier texte doit être indiquée. MS-DOS, comme tout système d'exploitation, lit unité d'allocation par unité d'allocation et il n'y a aucune raison que le fichier se termine juste à la fin d'une unité d'allocation. Si la fin du fichier texte n'est pas indiquée d'une façon ou d'une autre, la lecture se ferait jusqu'au dernier caractère de la dernière unité d'allocation du fichier, ce qui donne lieu à des caractères non voulus et éventuellement même incompréhensibles (puisque'ils ne correspondront pas nécessairement à des caractères affichables).

En MS-DOS la fin d'un fichier est connue à la fois grâce à sa taille mais aussi, pour des raisons de compatibilité avec le système d'exploitation dont il s'inspire (CP/M), par le caractère (non affichable) CTRL-Z de code ASCII 26.

**EXERCICES**

Exercice 1.- 1°) *Écrire un programme C permettant d'enregistrer dans un fichier un nombre quelconque de valeurs entières positives fournies au clavier. Les composants du fichier seront des entiers.*

2°) *Écrire un programme C permettant de retrouver, dans le fichier précédent, une valeur de rang donné. On fournira un message dans le cas où le rang demandé dépasse la taille du fichier.*

Exercice 2.- *Écrire un programme C qui affiche un fichier texte en numérotant les lignes. Le nom du fichier sera lu en données.*

Exercice 3.- (**Copie de fichier**)

*Écrire une fonction C qui copie un fichier donné dans un autre fichier de même type.*

[ On demandera le nom des deux fichiers et si le second fichier a déjà été créé ou non. ]

Exercice 4.- (**Modification d'un fichier séquentiel**)

*Écrire un programme C permettant de modifier un fichier séquentiel en s'aidant d'un fichier auxiliaire.*

[ On recopiera dans un fichier auxiliaire tous les composants du premier fichier sauf, éventuellement celui à modifier que l'on remplacera par un nouveau composant. Puis on recopiera ce fichier auxiliaire dans le fichier de départ. ]

Exercice 5.- (**Ajout dans un fichier séquentiel**)

*Écrire un programme C permettant de compléter un fichier séquentiel, en s'aidant d'un fichier auxiliaire.*

Exercice 6.- (**Concaténation de fichiers**)

*Écrire un programme C qui copie dans un fichier le concaténaté de deux autres fichiers.*

Exercice 7.- (**Complétion de fichier**)

*Écrire un programme C qui ajoute le contenu d'un fichier texte à la fin d'un autre fichier texte. Le contenu du premier fichier ne doit pas être modifié.*

Exercice 8.- (**Fusion de fichiers**)

*Écrire un programme C qui prend deux listes d'entiers, chacune triée du plus petit au plus grand, et les fusionne en une seule liste contenant tous ces nombres dans l'ordre croissant. Les trois listes sont dans des fichiers d'entiers.*

Exercice 9.- (**Tri interne de fichiers**)

*Écrire un programme C qui lit 10 valeurs entières d'un fichier d'entiers, les place dans un tableau, trie le tableau dans l'ordre croissant et place ce tableau trié dans le même fichier à la place des 10 valeurs de départ.*

Exercice 10.- (**Tri externe de fichiers**)

*Écrire un programme C qui trie un fichier d'entiers dans l'ordre croissant.*

[ Le résultat du fichier trié doit être le même fichier physique que le fichier initial. Le programme doit être valable quelle que soit la taille du fichier, on ne peut donc pas utiliser la méthode de l'exercice précédent. Une façon de faire est d'utiliser un autre fichier dans lequel on place le plus petit élément, puis le plus petit des éléments restants et ainsi de suite. On copie alors le fichier obtenu dans le fichier initial. Il existe des méthodes plus efficaces pour faire cela, que nous verrons plus tard. ]