CHAPITRE 6

Lecture/Écriture de fichiers

Entrées-sorties standard

- < stdio.h > :Fonctions et les structures associées décrites dans
- Tout programme désirant manipuler les E/S devra contenir la ligne : #include<stdio.h>

- Le fichier stdio.h contient :
  - les définitions de macro-expressions ;
  - les prototypes des fonctions ;
  - les définitions de constantes : EOF, stdin, stdout, . . . ;

Entrées-sorties standard

- En langage C, un fichier est une suite d’octets stockant des caractères. Le fichier n’a pas de structure propre qui lui est attachée.

- Le principe de la manipulation d’un fichier est le suivant :
  - ouverture du fichier ;
  - lecture, écriture, et déplacement dans le fichier ;
  - fermeture du fichier.

Entrées-sorties standards

- Trois **pseudo-fichiers** sont ouverts par l'environnement de programmation lorsque le programme commence à s'exécuter. Ces trois pseudo-fichiers permettent l'accès au terminal de l'utilisateur. Ce sont :
  - stdin
    - fichier standard d'entrée (souvent associé au clavier)
  - stdout
    - fichier standard de sortie (souvent l'écran). Les fonctions d'accès associées utilisent une technique de tampon (buffer) pour les écritures qui lui sont associées ;
  - stderr
    - fichier standard d'erreur. Ce fichier correspond lui aussi le plus souvent à l'écran mais les écritures sont non tamponnées (pas de buffer).
  - Ce sont les flux par défaut.

Entrées-sorties standards

- À ces fichiers standards sont associées des fonctions prédéfinies qui permettent de réaliser les opérations suivantes :
  - lecture et écriture caractère par caractère
  - lecture et écriture ligne par ligne
  - lecture et écriture formatées
- Ces fichiers peuvent être redirigés au niveau de l'interprète de commandes par l'utilisation des symboles > et < à l'appel du programme.
- Ils sont automatiquement redirigés lorsque les commandes sont enchaînées par des tubes en utilisant le symbole |. Exemples :
- prog > fichier : lorsque prog écrit, les octets sont dirigés vers le fichier fichier
- prog < fichier : prog lit dans le fichier fichier
- prog 2> fichier : prog écrit ses messages d'erreur dans le fichier fichier
- prog1 | prog2 : la sortie standart de prog1 est associée à l'entrée standard de prog2.

Fichiers

- À part les trois pseudo-fichiers dont nous venons de parler, tout fichier doit être ouvert avant de pouvoir accéder à son contenu en lecture, écriture ou modification.

- L’ouverture d’un fichier est l’association d’un objet extérieur (le fichier) au programme en cours d’exécution.

- L’ouverture d’un fichier est réalisée par la fonction fopen( )

Fichiers

- FILE *fopen(const char *, const char*) ;
  - ouverture d'un fichier référencé par le premier argument (nom du fichier dans le système de fichiers sous forme d'une chaîne de caractères) selon le mode d'ouverture décrit par le second argument (chaîne de caractères).
  - arguments :
    - La première chaîne de caractères contient le nom du fichier de manière à référencer le fichier dans l'arborescence. Le deuxième argument est lui aussi une chaîne de caractères. Il spécifie le type d'ouverture.
  - retour :
    - pointeur sur un objet de type FILE (type défini dans < stdio.h >) qui sera utilisé par les opérations de manipulation du fichier ouvert (lecture, écriture ou déplacement).
    - Ce « pointeur de fichier » doit être déclaré.
  - conditions d'erreur :
    - retourne le pointeur NULL ((void *)0) si le fichier n'a pas pu être ouvert (problème d'existence du fichier ou de droits d'accès).

Fichiers

- Sans précision, le fichier est considéré de type texte (c'est-à-dire ne contenant que des caractères ASCII). Le type d'ouverture de base peut être :
  - **"r"** : le fichier est ouvert en lecture. Si le fichier n'existe pas, la fonction ne le crée pas.
  - **"w"** : le fichier est ouvert en écriture. Si le fichier n'existe pas, la fonction le crée. Si le fichier existe la fonction le vide.
  - **"a"** : le fichier est ouvert en ajout. Si le fichier n'existe pas, la fonction le crée. Les écritures auront lieu à la fin du fichier.

- Ce type d'ouverture peut être complété de deux caractères qui sont :
  - **"b"** : le fichier est considéré en mode binaire. Il peut donc contenir des données qui sont transférées sans interprétation par les fonctions de la bibliothèque.
  - **"+"** : le fichier est ouvert dans le mode complémentaire du mode de base. Par exemple s'il est ouvert dans le mode "r+" cela signifie qu'il est ouvert en mode lecture et plus, soit lecture et écriture.

Fichiers

- La combinaison des modes de base et des compléments donne les possibilités suivantes :
  - "r+"
  - "w+"
  - "a+" "rb"
  - "wb"
  - "ab"
  - "r+b"
  - "w+b"
  - "a+b"

- Si un fichier qui n’existe pas est ouvert en mode écriture ou ajout, il est créé par le système.

Fichiers

- Fermeture d’un fichier
- int fclose(FILE *) ;
  - Fonction inverse de fopen() ; elle détruit le lien entre la référence vers la structure FILE et le fichier physique. Si le fichier ouvert est en mode écriture, la fermeture provoque l’écriture physique des données du tampon.
  - arguments :
    - une référence de type FILE valide.
  - retour :
    - 0 dans le cas normal, EOF en cas d’erreur.

Fichiers

- Tout programme manipulant un fichier doit être encadré par les deux appels de fonctions fopen() et fclose() :

```
#include <stdio.h>
int main (int argc, char *argv[]) {
        FILE *MyFich;
        MyFich = fopen ("/etc/passwd", "r");
        /* ... */
        fclose (MyFich);
    return 0;
 }
```

Lecture/écriture caractère par caractère

- int getchar(void) ;
  - Permet de lire un caractère (unsigned char) sur stdin s'il y en a un ;
  - aucun argument
  - Retourne un entier contenant la valeur du caractère lu, ou EOF en fin de fichier

- int putchar(int) ;
  - Permet d'écrire un caractère sur stdout ;
  - Retourne la valeur du caractère écrit (considéré comme un entier).
  - En cas d'erreur la fonction retourne EOF.

Lecture/écriture caractère par caractère

- Programme reproduisant le « cat » du système unix en lisant le fichier d'entrée standard caractère par caractère et reproduisant les caractères de son fichier standard d'entrée sur son fichier standard de sortie

- ```
  #include <stdio.h>
   int main (int argc, char *argv[]) {
       int c;
       while ((c = getchar ()) != EOF)
           putchar (c);
       return 0;
   }
  ```
  ```

Lecture/écriture ligne par ligne

- La ligne est considérée comme une suite de caractères (char) terminée par un caractère de fin de ligne ou par la détection de la fin du fichier.

- Il existe une fonction char *gets(char *) ;
  MAIS
  - Cette fonction ne peut pas vérifier que la taille de la ligne lue est inférieure à la taille de la zone mémoire dans laquelle il lui est demandé de placer les caractères
  - -> ne pas l'utiliser, préférer : fgets()

- int puts(char *) ;
  - Écrit une chaîne de caractères, suivie d'un '\n' sur stdout.
  - argument : l'adresse d'une chaîne de caractères.
  - retour : une valeur entière non négative en cas de succès (attention !)
  - Elle retourne EOF en cas de problème.

Lecture/écriture ligne par ligne

- Programme reproduisant le « cat » du système unix avec lecture et réécriture ligne par ligne.

- On sur-dimensionne le tableau à 256 caractères pour tenter d'éviter un débordement par la fonction gets().
- Attention : si on passe une ligne de plus grande taille, le comportement du programme est non défini.

```
#include <stdio.h>
int  main (int argc, char *argv[]) {
        char BigBuf[256];
        while (gets (BigBuf) != NULL)
             puts (BigBuf);
        return 0;
 }
```

Lecture/écriture formatées

- int scanf(const char *, ...) ;
  - lecture formatée sur stdin.
  - liste d'arguments variable à partir du second argument.
    - le premier argument est une chaîne de caractères qui doit contenir la description des variables à saisir.
    - les autres arguments sont les adresses des variables (conformément à la description donnée dans le premier argument) qui sont affectées par la lecture.
  - retour : nombre de variables correctement saisies.
  - la valeur EOF est retournée en cas d'appel sur un flux d'entrée standard fermé.

Lecture/écriture formatées

- int printf (const char *, ...) ;
  - écriture formatée sur stdout
  - arguments :
    - chaîne de caractères contenant des commentaires et des descriptions d’arguments à écrire, suivie des valeurs des variables.
  - retour : nombre de caractères écrits.
  - la valeur EOF est retournée en cas d’appel sur un fichier de sortie standard fermé.

Accès au contenu d'un fichier

- Une fois un fichier ouvert, le langage C permet plusieurs types d'accès à un fichier :
  - par caractère,
  - par ligne,
  - par données formatées,
  - + par enregistrement

- Dans tous les cas, les fonctions d'accès au fichier (sauf les opérations de déplacement) ont un comportement séquentiel. L'appel de ces fonctions provoque le déplacement du pointeur courant relatif au fichier ouvert.

Accès caractère par caractère

fgetc, getc, ungetc, fputc, putc

- int fgetc(FILE *) ;
  - lecture d’un caractère (unsigned char) dans le fichier associé
  - argument : une référence de type FILE valide correspondant à un fichier ouvert en lecture
  - Retourne la valeur du caractère lu promue dans un entier
  - À la rencontre de la fin de fichier, la fonction retourne EOF et positionne les indicateurs associés

Accès caractère par caractère

- int ungetc(int, FILE *) ;
  - cette fonction permet de remettre un caractère dans le buffer de lecture associé à un flux d'entrée.

- int fputc(int , FILE *) ;
  - écrit dans le fichier associé décrit par le second argument un caractère spécifié dans le premier argument. Ce caractère est converti en un unsigned char
  - argument :
    - le premier argument contient le caractère à écrire et le second contient la référence de type FILE du fichier ouvert
  - retour :
    - la valeur du caractère écrit promue dans un entier sauf en cas d'erreur
  - conditions d'erreur :
    - en cas d'erreur d'écriture, la fonction retourne EOF et positionne les indicateurs associés

Accès caractère par caractère

- int getc(FILE *) ;
  - cette fonction est identique à fgetc() mais est réalisée par une macro définie dans < stdio.h >

- int putc(int , FILE *) ;
  - cette fonction est identique à fputc() mais est réalisée par une macro définie dans < stdio.h >

Étant donné qu'elle peut être implémentée comme une macro, la fonction getc() peut traiter incorrectement un argument de flux comportant des effets de bord. En particulier, getc(*f++) ne fonctionne pas nécessairement comme prévu.

Exemple

- Ecriture à l'écran du contenu d'un fichier dont le nom est donné en argument (identique à la commande cat UNIX avec comme argument un nom de fichier). Le fichier est ouvert en mode lecture et il est considéré comme étant en mode texte.

```
#include <stdio.h>
int  main (int argc, char *argv[]) {
    FILE *MyFic;
    int TheCar;
    if (argc != 2)
       return 1;
    MyFic = fopen (argv[1], "r");
    if (MyFic == NULL) {
         printf ("Impossible d ouvrir le fichier %s \n", argv[1]);
         return 2;
    }
    while ((TheCar = fgetc (MyFic)) != EOF)
        fputc (TheCar, stdout);
    fclose (MyFic);
    return 0;
}
```

Accès ligne par ligne (fgets, fputs)

- char *fgets(char *, int , FILE *) ;
  - lit une ligne de caractères ou au plus le nombre de caractères correspondant au deuxième argument moins un, dans le fichier associé au troisième argument.
  - Les caractères de la ligne sont rangés dans la mémoire à partir de l'adresse donnée en premier argument. Si le nombre de caractères lu est inférieur à la taille, le '\n' est lu et rangé en mémoire. Il est suivi par un caractère nul '\0' de manière à ce que la ligne une fois placée en mémoire, puisse être utilisée comme une chaîne de caractères.
  - arguments :
    - adresse de la zone de stockage des caractères en mémoire (doit être allouée)
    - nombre maximum de caractères (taille de la zone de stockage),
    - la référence de type FILE du fichier ouvert.
  - retour : adresse reçue en 1er agrument sauf en cas d'erreur ;
  - La lecture s'arrête quand
    - une ligne est lue (rencontre du retour chariot), OU la fin de fichier est rencontrée, OU le nombre de caractères lu est égal à la taille du buffer moins un.

À la rencontre de la fin de fichier, la fonction retourne NULL et positionne l'indicateurs de fin de fichier (accessible via int feof(FILE * stream)).

Accès ligne par ligne

- int fputs(const char *, FILE *) ;
  - Permet d'écrire une chaîne de caractères référencée par le premier argument dans le fichier décrit par le second argument.
  - le premier argument contient l'adresse de la zone mémoire contenant les caractères à écrire (et un caractère nul). Elle doit contenir un '\n' pour obtenir un passage à la ligne suivante.
  - le second argument contient la référence de type FILE du fichier ouvert dans lequel les caractères seront écrits.
  - retour :
    - une valeur positive si l'écriture s'est correctement déroulée, EOF sinon.

Exemple

- lecture du fichier correspondant au nom passé en argument sur la ligne de commande et l'écriture de ces lignes sur le fichier standard de sortie. Les opérations sont réalisées ligne par ligne.

```
#include <stdio.h>
int main (int argc, char *argv[]) {
  FILE *TheFic;
  char BigBuf[256];

  if (argc != 2)
     return 1;
  TheFic = fopen (argv[1], "r");
  if (TheFic == NULL)
    {
       printf ("Impossible d ouvrir le fichier %s \n", argv[1]);
       return 2;
    }
  while (fgets (BigBuf, sizeof BigBuf, TheFic) != NULL)
     fputs (BigBuf, stdout);
  fclose (TheFic);
  return 0;
}
```

Entrées-sorties formatées

- Les lectures formatées nécessitent :
  - le format de description des lectures à faire
  - une adresse pour chaque variable simple ou pour un tableau de caractères.
- Les écritures formatées nécessitent :
  - le format de description des écritures à faire
  - les valeurs des variables simples à écrire. Comme dans le cas de la lecture, l'écriture d'un ensemble de caractères est une opération particulière qui peut se faire à partir de l'adresse du premier caractère.
- Les formats de conversion servent à décrire les types internes et externes des données à lire.

Entrées-sorties formatées : lecture

- Les formats peuvent contenir :
  - des caractères "blancs" (espace, tabulation).
  - des caractères ordinaires (ni blanc, ni %). Ces caractères devront être rencontrés à la lecture ;
  - des spécifications de conversion, commençant par le caractère %.

- Format : %[*][width][length]specifier
- Une conversion consiste en :
  - un caractère de pourcentage (%) ;
  - un caractère (optionnel) d'effacement (*) ; dans ce cas la donnée lue est mise à la poubelle ;
  - un champ (optionnel) définissant la taille de la donnée à lire exprimée par une valeur entière en base dix ;

Entrées-sorties formatées : lecture

- Une conversion consiste en (continuation):
  - un caractère (optionnel) de précision de taille qui peut être : l, ll, h, hh ou L. Ces caractères agissent sur les modes de spécification de la manière suivante :
    - si le format initial est du type d ou i ou n, les caractères l et ll précisent respectivement que la donnée est du type entier long (long int) ou long long, h et hh un entier court (short int) ou un char plutôt qu'un int.
    - si le format initial est du type o, x ou u, les caractères l et h précisent respectivement que la donnée est du type entier long non signé (unsigned long int) ou entier court non signé (unsigned short int) plutôt qu'entier non signé (unsigned int).
    - si le format initial est du type e, f ou g, les caractères l et L précisent respectivement que la donnée est du type nombre avec point décimal de grande précision double) ou nombre avec point décimal de très grande précision (long double) plutôt que du type nombre avec point décimal (float).
    - dans tous les autres cas, le comportement est indéfini.

Entrées-sorties formatées : lecture

Les codes de conversion pour scanf() sont :

- % : lit un %
- d : entier signé exprimé en base décimale
- u : entier non signé exprimé en base décimale
- i : entier signé exprimé en base décimale ou octal ou hexadécimal
- o : entier non signé exprimé en base octale u entier non signé exprimé en base décimale
- x : entier non signé exprimé en hexadécimal
- e f g : nombre avec partie décimale en notation point décimal ou exponentielle
- c : caractère
- s : mots ou chaîne de caractères sans blanc
- [spécification] : chaîne de caractères parmi un alphabet
- [^spécification] : chaîne de caractères sans un alphabet
- p : adresse, pour faire l'opération inverse de l'écriture avec %p
- n : permet d'obtenir le nombre d'octets lus dans cet appel

La spécification entre les crochets définit un alphabet de caractères. La donnée lue doit être conforme à cette spécification.

Entrées-sorties formatées : écriture

- Les formats peuvent contenir :
  - des caractères qui sont recopiés dans la chaîne engendrée par l'écriture ;
  - et des spécifications de conversion.
- Format :

  %[flags][width][.precision][length]specifier

- Une spécification de conversion consiste en :
  - un caractère de pourcentage (%) ;
  - des drapeaux (flags) qui modifient la signification de la conversion (-, 0…)
  - la taille minimum du champ dans lequel est insérée l'écriture de la donnée
  - un point suivi de la précision. La précision définit le nombre de chiffres significatifs pour une donnée de type entier, ou le nombre de chiffres après la virgule pour une donnée de type flottant. Elle indique le nombre de caractères pour une chaîne ;

Entrées-sorties formatées : écriture

- Une spécification de conversion consiste en (continuation) :
- un h ou un l ou un L signifiant court ou long et permettant de préciser :
  - dans le cas d'une conversion d'un entier (format d, i, o, u, x, ou X) que l'entier à écrire est un entier court (h) ou long (l) ;
  - dans le cas d'une conversion d'un nombre avec partie décimale (format e, f, g, E, ou G) que le nombre à écrire est un nombre avec point décimal de très grande précision (long double).

Entrées-sorties formatées : écriture

- Les champs taille et précision peuvent contenir une *. Dans ce cas la taille doit être passée dans un argument à [sf]printf.
- Par exemple les lignes suivantes d'appel à printf() sont équivalentes :

```
printf("Valeur de l'entier Indice : %*d\n",6,Indice);
printf("Valeur de l'entier Indice : %6d\n",Indice);
```

Entrées-sorties formatées : écriture

codes de conversion

- % : écrit un %
- d : entier signé exprimé en base décimale
- i : entier signé exprimé en base décimale
- o : entier non signé exprimé en base octale
- u : entier non signé exprimé en base décimale
- x, X : entier non signé exprimé en hexadécimal
- e, E : nombre avec partie décimale en notation exponentielle
- f,F : nombre avec partie décimale en notation point décimal
- g, G : nombre avec partie décimale, plus petit en taille des formats f ou e
- c : caractère
- s : chaîne de caractères
- p : la valeur passée est une adresse
- n : permet d'obtenir le nombre d'octets écrits

Entrées-sorties formatées : écriture

- La différence entre x et X vient de la forme d'écriture des valeurs décimales entre 10 et 15.

  Dans le premier cas, elles sont écrites en minuscule (a-f), dans le second cas, elles sont écrites en majuscule (A-F).
- De même, le caractère E de la notation exponentielle est mis en minuscule par les formats e et g. Il est mis en majuscule par les formats E et G.
- Selon la norme [ISO89], le nombre maximum de caractères qui peuvent être construits dans un appel aux fonctions de type fprintf() ne doit pas dépasser 509.

Entrées-sorties formatées : écriture

drapeaux

- - : la donnée convertie est cadrée à gauche
- + : si la donnée est positive le signe + est mis
- blanc : si le résultat de la conversion ne commence pas par un signe, un blanc est ajouté
- 0 : remplissage avec des 0 devant plutôt que des blancs
- ’ : pour les conversions décimales groupement des chiffres par 3
- # :
  - pour format o augmente la précision de manière à forcer un 0 devant la donnée
  - pour format x et X force 0x devant la donnée
  - pour format e, E, f, g, et G force le point décimal
  - pour format g et G les zéros après le point décimal sont conservés

Conversion sur les entrées-sorties standards

- Nous avons déjà étudié les deux fonctions d’entrées-sorties standards formatées qui sont :
  - int printf(const char *format, ...) ;
  - int scanf(const char *format, ...) ;

Conversion en mémoire

- sprintf, sscanf

- int sprintf(char *string, const char *format, ...) ;
  - conversion de données en mémoire par transformation en chaîne de caractères.
  - arguments :
    - zone dans laquelle les caractères sont stockés ;
    - format d'écriture des données ;
    - valeurs des données.
  - retour :
    - nombre de caractères stockés.

sprintf() convertit les paramètres arguments passés à partir du 3ème paramètre selon le format demandé et met le résultat dans la chaîne de caractères passée en premier paramètre.

```
int x=2, y=3;
sprintf(chaine, « La somme de %d et %d vaut %d», x, y, x+y);
```

Conversion en mémoire

- Inversement, sscanf extrait d'une chaîne de caractères passée en premier paramètre des valeurs qui sont stockées dans des variables suivant le format de contrôle.

- int sscanf(char *string, const char *format, ...) ;
  - lecture formatée à partir d'une zone mémoire.
  - arguments :
    - zone dans laquelle les caractères sont acquis ;
    - format de lecture des données ;
    - adresse des variables à affecter à partir des données.
  - retour :
    - nombre de variables saisies.
  - conditions d'erreur :
    - la valeur EOF est retournée en cas d'erreur empêchant toute lecture.

Exemple

- Faire un programme C qui fait la somme de deux arguments numériques passés en ligne de commande.

```
int main(int argc, char *argv[]) {
      int i,j;
    sscanf(argv[1],"%d",&i);
    sscanf(argv[2],"%d",&j);
    printf("%d\n",i+j);
    return 0;
}
```

Conversion dans les fichiers

- Deux fonctions fprintf et fscanf permettent de réaliser le même travail que printf et scanf sur des fichiers ouverts en mode texte

- **int fprintf(FILE *,const char *,...);**
  - écriture formatée sur un fichier ouvert en mode texte.
  - **arguments :**
    - référence vers la structure décrivant le fichier ouvert dans lequel les caractères sont rangés ;
    - format d'écriture des données ;
    - valeurs des données.
  - **retour :**
    - nombre de caractères écrits.
  - **conditions d'erreur :**
    - une valeur négative est retournée en cas d'erreur d'écriture.

Conversion dans les fichiers

- **int fscanf(FILE *,const char *,...);**
  - lecture formatée dans un fichier ouvert en mode texte.
  - **arguments :**
    - référence vers la structure décrivant le fichier ouvert dans lequel les caractères sont lus ;
    - format de lecture des données ;
    - adresse des variables à affecter à partir des données.
  - **retour :**
    - nombre de variables saisies.
  - **conditions d'erreur :**
    - la valeur EOF est retournée en cas d'appel sur un fichier standard d'entrée fermé.

Accès enregistrement par enregistrement

- L'accès par enregistrement permet de lire et d'écrire des objets structurés dans un fichier (représentés en mémoire par des structures).
- Pour ce type d'accès, le fichier doit être ouvert en mode binaire. Les données échangées ne sont pas traitées comme des caractères.

- L'accès par enregistrement se fait grâce aux fonctions :
  - size_t fread(void *Zone, size_t Taille, size_t Nbr, FILE *fp) ;
  - size_t fwrite(void *Zone, size_t Taille, size_t Nbr, FILE *fp) ;
- Ces deux fonctions retournent le nombre d'enregistrements échangés.

Accès enregistrement par enregistrement

- size_t fread(void *Zone, size_t Taille, size_t Nbr, FILE *fp) ;

- le premier argument (Zone) est l'adresse de l'espace mémoire à partir duquel l'échange avec le fichier est fait. L'espace mémoire correspondant reçoit les enregistrements lus, ou fournit les données à écrire dans les enregistrements. Il faut que l'espace mémoire correspondant à l'adresse soit de taille suffisante pour supporter le transfert des données, c'est-à-dire d'une taille au moins égale à (Taille x Nbr).
- le deuxième argument (Taille) est la taille d'un enregistrement en nombre d'octets.
- le troisième argument (Nbr) est le nombre d'enregistrements que l'on désire échanger.
- le dernier argument (fp) est une référence vers une structure de type FILE correspondant à un fichier ouvert dans un mode de transfert binaire.

Accès enregistrement par enregistrement

Exemple :

- Une structure automobile contenant : age, couleur, numéro d'immatriculation, type, marque d'une voiture.
- On veut lire les informations sur 20 voitures dans un fichier et les stocker dans un tableau contenant 20 voitures.

Accès enregistrement par enregistrement

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

 struct automobile {
        int age;
        char couleur[20],
        numero[10],
        type[10],
        marque[10];

        } ParcAuto[20];
```

```
int  main (int argc, char *argv[]) {
   FILE *TheFic;
   int i;
   size_t fait;
   TheFic = fopen ("FicParcAuto", "rb+");
   if (TheFic == NULL)  {
        printf ("Impossible d ouvrir le fichier FicParcAuto\n");
        return 1;
   }
   fait = fread(ParcAuto,sizeof(struct automobile),
  20,TheFic);
   if(fait != 20){
      printf ("Erreur lecture fichier parcauto \n");
      return 2;
   }
  fclose (TheFic);
  return 0;
  }
```

Accès enregistrement par enregistrement

- Se méfier de la portabilité !
- Sur ce modèle, un fichier écrit sur une machine n’est pas forcément lisible simplement sur une autre machine.