11. Les fichiers

Présentation des fichiers(1)

- En C, les communications d'un programme avec son environnement se font par l'intermédiaire de fichiers. Pour le programmeur, tous les périphériques, même le clavier et l'écran, sont des fichiers. Jusqu'ici, nos programmes ont lu leurs données dans le fichier d'entrée standard, (le clavier) et ils ont écrit leurs résultats dans le fichier de sortie standard (l'écran). Nous allons voir comment nous pouvons créer, lire et modifier nous-mêmes des fichiers sur les périphériques disponibles.
- Un ***fichier*** (angl.: *file*) est un ensemble structuré de données stocké en général sur un support externe (disquette, disque dur, disque optique, bande magnétique, ...). Un ***fichier structuré*** contient une suite *d'enregistrements* homogènes, qui regroupent le plus souvent plusieurs composantes appartenant à un ensemble *(champs)*.
- ***Fichier séquentiel:*** Dans des ***fichiers séquentiels***, les enregistrements sont mémorisés consécutivement dans l'ordre de leur entrée et peuvent seulement être lus dans cet ordre. Si on a besoin d'un enregistrement précis dans un fichier séquentiel, il faut lire tous les enregistrements qui le précèdent, en commençant par le premier.

Présentation des fichiers(2)

- **Types d'accès aux fichiers**
  - **Accès séquentiel** (surtout pour les bandes magnétiques)
    - Pas de cellule vide.
    - On accède à une cellule quelconque en se déplaçant depuis la cellule de départ.
    - On ne peut pas détruire une cellule.
    - On peut par contre tronquer la fin du fichier.
    - On peut ajouter une cellule à la fin.
  - **Accès direct** (disques, disquettes, CD-ROM où l'accès séquentiel est possible aussi).
    - Cellule vide possible.
    - On peut directement accéder à une cellule.
    - On peut modifier (voir détruire) n'importe quelle cellule.

Présentation des fichiers(3)

- **Codage**
  - En binaire : Fichier dit « binaire », Ce sont en général des fichiers de nombres. Ils ne sont pas listables.
  - En ASCII : Fichier dit « texte », les informations sont codées en ASCII. Ces fichiers sont listables. Le dernier octet de ces fichiers est EOF (caractère ASCII spécifique).
- **Fichiers standard**
  - Il existe deux fichiers spéciaux qui sont définis par défaut pour tous les programmes:
    - ***stdin*** : le fichier d'entrée standard, lié en général au clavier (nom interne = 0)
    - ***stdout*** : le fichier de sortie standard, lié à l'écran (nom interne = 1)
    - ***stderr*** : fichier de sortie erreur (nom interne = 2)

Présentation des fichiers(4)

- **La mémoire tampon**

  Pour des raisons d'efficacité, les accès à un fichier se font par l'intermédiaire d'une ***mémoire tampon*** (en anglais: ***buffer***). La mémoire tampon est une zone de la mémoire centrale de la machine réservée à un ou plusieurs enregistrements du fichier. L'utilisation de la mémoire tampon a l'effet de réduire le nombre d'accès à la périphérie d'une part et le nombre des mouvements de la tête de lecture/écriture d'autre part.

  Pour pouvoir manipuler un fichier, un programme a besoin d'un certain nombre d'informations : l'adresse de l'endroit de la mémoire-tampon où se trouve le fichier, la position de la tête de lecture, le mode d'accès au fichier (lecture ou écriture) ...Ces informations sont rassemblées dans une structure dont le type, **FILE ***, est défini dans **stdio.h**. Un objet de type FILE * est appelé flot de données (en anglais, **stream**).

Manipulation des fichiers(1)

- Les opérations sur les fichiers les plus courant sont: ***Créer*** - ***Ouvrir*** - ***Fermer*** - ***Lire*** - ***Ecrire*** - ***Détruire*** - ***Renommer***. Le langage **C** ne distingue pas les fichiers à accès séquentiel des fichiers à accès direct, certaines fonctions de la bibliothèque livrée avec le compilateur permettent l'accès direct. Les fonctions standard sont des fonctions d'accès séquentiel.
- **Déclaration :**

  FILE *fichier ; **/* majuscules obligatoires pour FILE */**

  Définit un pointeur qui va pointer vers une variable de type FILE qui est une structure (struct).

Manipulation des fichiers(2)

- **Ouverture:**

  FILE *fopen(char *nom_fichier, char *mode_ouverture);

  nom_fichier est une chaîne de caractères représentant le nom du fichier à ouvrir.

  mode_ouverture est une chaîne représentant le mode d'ouverture du fichier. Elle peut être l'une des chaînes suivantes :

- **Mode** (pour les fichiers TEXTES) :

  "r" ouverture en lecture seule.

  "w" ouverture en écriture seule.

  "a" ouverture en écriture à la fin.

  "r+" ouverture en lecture/écriture.

  "w+" ouverture en lecture/écriture.

  "a+" ouverture en lecture/écriture à la fin.

Manipulation des fichiers(3)

- **Mode** (pour les fichiers BINAIRES) :

  "rb" ouverture en lecture seule.

  "wb" ouverture en écriture seule.

  "ab" ouverture en écriture à la fin.

  "r+b" ou "rb+" ouverture en lecture/écriture.

  "w+b" ou "wb+" ouverture en lecture/écriture.

  "a+b" ou "ab+" ouverture en lecture/écriture à la fin.
- A l'ouverture, le pointeur est positionné au début du fichier (sauf "a+" et "ab+")
- **Exemple :**

```
FILE *fichier ;
fichier = fopen( "c:\\fich.dat", "rb") ;
```

Manipulation des fichiers(4)

- **Valeur rendue**

  La fonction **fopen** retourne une valeur de type pointeur vers **FILE**, où **FILE** est un type prédéfini dans le fichier **stdio.h**.

  – Si l'ouverture a réussi, la valeur retournée permet de repérer le fichier, et devra être passée en paramètre à toutes les procédures d'entrées-sorties sur le fichier.

  – Si l'ouverture s'est avérée impossible, **fopen** rend la valeur **NULL**.

  Conditions particulières et cas d'erreur

  – **Si** le mode contient la lettre "**r**", le fichier doit exister, **sinon** c'est une erreur.

  – **Si** le mode contient la lettre "**w**", le fichier peut exister ou pas.
    - **Si** le fichier n'existe pas, il est créé ;
    - **Si** le fichier existe déjà, son ancien contenu est perdu.

  – **Si** le mode contient la lettre "**a**", le fichier peut exister ou pas.
    - **Si** le fichier n'existe pas, il est créé ;
    - **Si** le fichier existe déjà, son ancien contenu est conservé.

Manipulation des fichiers(5)

- **Exemple:**

```
FILE *fp;
if ( (fp = fopen("donnees.txt","r") ) == NULL)
{
    fprintf(stderr,"Impossible d'ouvrir le fichier
                données en lecture\n");
    exit(1);
}
else  fprintf(stderr,  "ouverture avec succès\n " ) ;
```

Manipulation des fichiers(6)

- **Fermeture:**

  int fclose(FILE *fichier);

  Retourne **0** si la fermeture s'est bien passée, **EOF** en cas d'erreur.

  Il faut toujours fermer un fichier à la fin d'une session.

- Exemple :

```
FILE *fichier ;
fichier = fopen( "c:\\fich.dat ", " rb") ;
/* Ici instructions de traitement */
fclose(fichier) ;
```

Manipulation des fichiers(7)

- **Destruction:**

  int remove(const char *nom_fichier);

  Permet de supprimer un fichier fermé. Retourne **0** en cas de succès.
  Exemple : remove ("c:\\fich.dat ");

- **Renommage:**

  int rename(char *AncienNom, char *NouvNom);

  Retourne **0** si la fonction s'est bien passée. La valeur **-1** Sinon.

- **Changement du mode d'accès:**

  int chmod (const char *nom_fichier, int mode);

  Modifie ou définie les droit d'accès à un fichier. « mode » est une constante (S_IREAD, S_IWRITE, S_IREAD | S_IWRITE) dans la bibliothèque **<sys\stat.h>**
  Retourne la valeur **0** en cas de succès. La valeur **-1** Sinon.

Manipulation des fichiers(8)

- **Positionnement du pointeur au début du fichier:**

  void rewind (FILE *fichier);

  Repositionne le pointeur du fichier sur le début du fichier.
- **Positionnement du pointeur dans un fichier**

  Int fseek (FILE *fichier, long nbOctet, int direction);

  fseek déplace le pointeur de fichier sur le **nbOctet+1** dans le sens indiqué par **direction**.

  Valeurs possibles pour **direction**:
  - **0** -> à partir du début du fichier.
  - **1** -> à partir de la position courante du pointeur.
  - **2** -> en arrière, à partir de la fin du fichier.
- **Détection de la fin d'un fichier séquentiel:**

  int feof(FILE *fichier) ;

  Retourne 0 si la fin du fichier n'est pas atteinte

Lecture et écriture dans les fichiers séquentiels(1)

- Les fichiers que nous employons ici sont des fichiers texte, c-à-d. toutes les informations dans les fichiers sont mémorisées sous forme de chaînes de caractères et sont organisées en lignes. Même les valeurs numériques (types **int**, **float**, **double**, ...) sont stockées comme chaînes de caractères.
- Pour l'écriture et la lecture des fichiers, nous allons utiliser les fonctions standard **fprintf**, **fscanf**, **fputc** et **fgetc** qui correspondent à **printf**, **scanf**, **putchar** et **getchar** si nous indiquons *stdout* respectivement *stdin* comme fichiers de sortie ou d'entrée.

Lecture et écriture dans les fichiers séquentiels(2)

- **Lecture par caractère :**

  int fgetc(FILE *fichier);

  Lit 1 caractère, mais retourne **un entier n** en cas de succès et retourne **EOF** si erreur ou fin de fichier; (le pointeur avance d'un octet pour exécuter l'opération de lecture).

- **Exemple :**

```
FILE *fichier ; char c ;
fichier = fopen( "c:\\ fich.dat ", " rb") ;
while ( (c = fgetc(fichier)) != EOF)
{
   ...  /*   utilisation de c   */
}
```

Lecture et écriture dans les fichiers séquentiels(3)

- **Ecriture par caractère :**

  int fputc(int c, FILE *fichier);

  Ecrit la valeur de **c** à la position courante du pointeur, le pointeur avance d'un octet. Retourne **EOF** en cas d'erreur et le **caractère injecté** en cas de succès.

- **Exemple : copier un fichier dans un autre**

```
FILE *fichier_src, *fichier_dest;
char c ;
Fichier_src = fopen("c:\\autoexec.bat","r");
Fichier_dest = fopen("c:\\copie_autoexec.bat","w");
while ((c = fgetc(fichier_src)) != EOF)
  fputc(c,fichier_dest);
```

Lecture et écriture dans les fichiers séquentiels(4)

- **Lecture par chaîne de caractère :**

  char *fgets (char * chaine, int n,FILE *fichier);

  Lit **n-1** caractères à partir de la position du pointeur et les range dans **chaine** en ajoutant **'\0'**. Retourne **chaine** en cas de succès ou la valeur **NULL** sinon.

- **Exemple :**

```
char ligne[20];
FILE *fichier;
Fichier= fopen (a :\\ fich.dat ", " rb") ;
while (fgets(ligne,20,fichier) != NULL)   /*  stop sur fin de fichier ou erreur  */
{
  ...  /*   utilisation de ligne   */
}
```

Lecture et écriture dans les fichiers séquentiels(5)

- **Ecriture par chaîne de caractère :**

  int fputs(char * chaine , FILE *fichier);

  fputs écrit sur le fichier le contenu de **chaine** ('\0' n'est pas rangé dans le fichier). Le pointeur avance de la longueur de la chaîne.

  Retourne : **une valeur non négative** si l'écriture se passe sans erreur, et **EOF** en cas d'erreur

- **Exemple :**

```
char ligne[20];
FILE *fichier;
Fichier= fopen ("a :\\ fich.dat ", " a+") ;
fputs ("un texte",fichier) ;
```

Lecture et écriture dans les fichiers séquentiels(6)

- **Lecture formatée à partir d'un fichier:**

  int fscanf (FILE *fichier, const char *format, liste d'adresses);

  fscanf lit une suite de caractères du fichier défini par ***fichier*** en vérifiant que cette suite est conforme à la description qui en est fait dans ***format***. Cette vérification s'accompagne d'un effet de bord qui consiste à affecter des valeurs aux variables pointées par les différents *paramètres* spécifiés par ***la liste d'adresse***.

  fscanf retourne : le **nombre** de ***parami*** affectés en cas de sucées. **S**'il y a eu rencontre de **fin de fichier** ou **erreur d'entrée-sortie**, avant toute affectation à un *parami*, **fscanf** retourne **EOF**.

  La lecture d'un paramètre s'arrête quand on rencontre un caractère **blanc** (*espace*, *tab*, *new line*).

Lecture et écriture dans les fichiers séquentiels(7)

- **Lecture formatée à partir d'une chaine:**

  int sscanf (const char *buffer, const char *format, liste d'adresses);

  La fonction **sscanf** réalise le même traitement que la fonction **fscanf**, avec la différence que les caractères lus par **sscanf** ne sont pas lus depuis un fichier, mais du tableau de caractères ***chaîne***. La rencontre du **'\0'** terminal de ***chaîne*** pour **sscanf** est équivalente à la rencontre de fin de fichier pour **fscanf**.

Lecture et écriture dans les fichiers séquentiels(8)

- **Ecriture formatée dans un fichier:**

  int fprintf(FILE *fichier, char *format, liste d'expressions);

  fprintf permet d'écrire dans le fichier la liste des expression selon les format spécifiés. Elle retourne :
  - le **nombre** de **caractères écrits** en cas de succès
  - une valeur **négative** s'il y a eu une erreur d'E/S.

- **Ecriture formatée dans une chaîne :**

  int sprintf (const char *buffer, const char *format, liste d'adresses);

  La fonction **sprintf** réalise le même traitement que la fonction **fprintf**, avec la différence que les caractères émis par **sprintf** n'est pas écrite dans un fichier, mais dans le tableau de caractères ***chaîne***. Un **'\0'** est écrit dans ***chaîne*** en fin de traitement.

Lecture et écriture dans les fichiers séquentiels(9)

Exemple : copie d'un fichier dans un autre. Le premier fichier contient deux colonnes (nom, moyenne)

```
#include <stdio.h>
#include <process.h>
void main()
{
 FILE * fichier_src, *fichier_dest;
 char nom[20];
 float moyenne;
 if( (fichier_src=fopen("c:\\test.txt","r"))==NULL)
 {
      printf("erreur d'ouverture du fichier source\n");
      exit (0);
 }
 if( (fichier_dest=fopen("c:\\test2.txt","w"))==NULL)
 {
      printf("erreur d'ouverture du fichier
      destination\n");
      exit (0);
 }
 while (!feof(fichier_src))
 {
      fscanf(fichier_src,"%s%f",&nom,&moyenne);
      fprintf(fichier_dest,"%s%f\n",nom,moyenne);
 }
}
```

Lecture et écriture dans les fichiers séquentiels(10)

- **Autre fonctions:**

  int fwrite(void *p,int taille_bloc, int nb_bloc, FILE *fichier);

  p de type pointeur, écrit à partir de la position courante du pointeur fichier **nb_bloc** X **taille_bloc octets** lus à partir de l'adresse **p**. Le pointeur fichier avance d'autant.

  Le pointeur p est vu comme une adresse, son type est sans importance.

  Retourne le nombre de blocs écrits.

- **Exemple**: taille_bloc=4 (taille d'un entier en C), nb_bloc=3, écriture de 3 entiers

```
int tab[10] ;
fwrite(tab,4,3,fichier) ;
```

  int fread(void *p, int taille_bloc, int nb_bloc, FILE *fichier);

  Analogue à **fwrite** en lecture.

  Retourne le nombre de blocs lus, et 0 à la fin du fichier.

Résumé sur les fichiers

	Langage algorithmique	C
Ouverture en écriture	`ouvrir <Nom> en écriture`	`<FP> = fopen(<Nom>,"w");`
Ouverture en lecture	`ouvrir <Nom> en lecture`	`<FP> = fopen(<Nom>,"r");`
Fermeture	`fermer <Nom>`	`fclose(<FP>);`
Fonction fin de fichier	`finfichier(<Nom>)`	`feof(<FP>)`
Ecriture	`écrire <Nom>:<Exp>`	`fprintf(<FP>,"...",<Adr>);` `fputc(<C>, <FP>);`
Lecture	`lire <Nom>:<Var>`	`fscanf(<FP>,"...",<Adr>);` `<C> = fgetc(<FP>);`

Exercice

Lire un fichier texte, avec un contrôle d'erreur : L'utilisateur saisit le nom du fichier, la machine retourne le listing du fichier s'il existe et un message d'erreur s'il n'existe pas.

Corrigé

```
#include <stdio.h>
#include <conio.h>
void main()
{
FILE *fichier;
char c,nom[10];
printf("NOM DU FICHIER A LISTER: ");
gets(nom);
if((fichier = fopen(nom,"r"))==NULL)
	printf("\nERREUR A L'OUVERTURE, CE FICHIER N'EXISTE PAS\n");
else
	{
	printf("\n\t\t\tLISTING DU FICHIER\n");
	printf("\t\t\t-----------------\n\n");
	while((c=getc(fichier))!=EOF)printf("%c",c);
	}
fclose(fichier);
printf("\n\nPOUR SORTIR FRAPPER UNE TOUCHE ");
getch();
}
```