Fichiers

Définition

Les fichiers sont des structures ordonnées dont tous les éléments sont du même type et leur nombre est indéfini. Généralement ils ont stockés sur des supports externes.

Quelles que soient les particularités du système d'exploitation sous-jacent, les fonctions de la bibliothèque standard des entrrées-sorties font en sorte que les fichiers soient vus par le programmeur comme des *flots*, c'est-à-dire des suites d'octets qui représentent des données selon une des deux modalités suivantes :

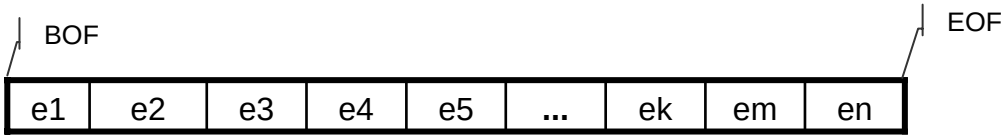

Fig. 10.1

Types de fichiers

- Classifiés par leur contenu
  - binaires – les données sont présentées comme elles sont stockées dans la mémoire machine. Ce sont des fichiers composés d'octets
  - texte – leurs éléments sont des caractères qui sont stockés en lignes dans le format approprié du système d'exploitation. Lecture et écriture se font par conversion dans le type de la variable.
- Classifiés par le mode d'accès
  - séquentiels – ils ont d'éléments du même type. L'accès se produit de premier au dernier élément sans retour. les fichiers texte sont toujours séquentiels
  - directs – les éléments sont accédés dans un ordre arbitraire.

On déclare des pointeurs vers la structure FILE qui est définie dans le fichier "stdio.h".

```
FILE *fich1, *fich2;
```

Dans le fichier "stdio.h" cette structure dans la bibliothèque de Turbo C est la suivante:

```
typedef struct  {
        int             level;          /* fill/empty level of buffer */
        unsigned        flags;          /* File status flags          */
        char            fd;             /* File descriptor            */
        unsigned char   hold;           /* Ungetc char if no buffer   */
        int             bsize;          /* Buffer size                */
        unsigned char   *buffer;        /* Data transfer buffer       */
        unsigned char   *curp;          /* Current active pointer     */
        unsigned        istemp;         /* Temporary file indicator   */
        short           token;          /* Used for validity checking */
}       FILE;                           /* This is the FILE object    */
```

Dans cette structure on peut voir des pointeurs vers le tampon et vers le curseur du fichier. Le type et l'état du fichier sont codés dans le champ *flags.* Chacun des compilateurs a sa propre réalisation de la bibliothèque et sa propre structure *FILE* mais la fonctionnalité est toujours pareille.

Opérateurs

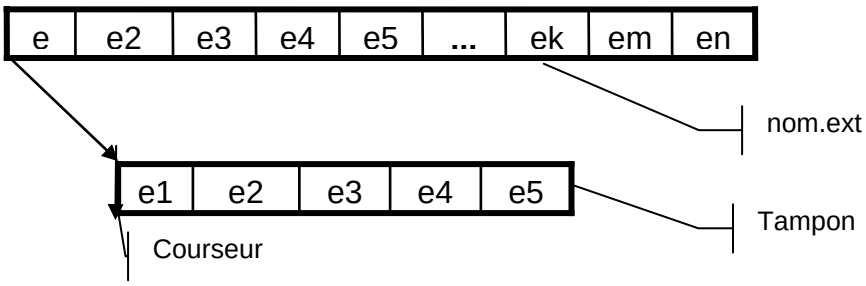

Fig. 10.2

Ouverture

L'objectif de l'ouverture est de prépare le fichier pour le traitement – soit lecture soit écriture soit les deux. Elle:

- Fait la correspondance entre le nom dans le SE est la variable du type fichier;
- Prépare les données pour le traitement
  - Le tampon est créé;
    - Si le fichier est ouvert pour lecture le tampon est rempli par les premiers données lues et le curseur est positionné avant le premier octet;
    - Si le fichier est ouvert pour écriture, le curseur est positionné à l'EOF.

La fonction *fopen*

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

- valeur rendue – un pointeur vers le fichier ouvert, ou NULL en cas d'erreur
- nom fichier – un pointeur vers la chaîne de caractères présentant le nom du fichier comme il est connu par le système d'exploitation;
- mode – une chaîne de deux éléments. Le premier le mode d'ouverture et le second – le type du fichier.

r	fichier ouvert pour lecture – il doit exister
w	fichier ouvert pour écriture – s'il existe il est détruit avant, il le crée sinon
a	fichier ouvert pour ajout à la fin. Si le fichier n'existe pas il est créé.
r+	fichier ouvert pour lecture et modification – il doit exister
w+	fichier ouvert pour écriture et modification – s'il existe il est détruit avant, il le crée sinon
a+	fichier ouvert pour ajout à la fin. Si le fichier n'existe pas il est créé.
t	fichier texte .
b	fichier binaire

Exemples

f = fopen("ftext.txt","rt");	ouvrir un fichier texte existant pour lecture
f1 = fopen("fich.dat","rb");	ouvrir un fichier binaire existant pour lecture
f2 = fopen("fic1.txt","wt");	ouvrir un fichier texte pour écriture
f1 = fopen("fich.dat","r+b"); f1= fopen("fich.dat","rb+");	ouvrir un fichier binaire existant pour lecture et modification
f= fopen("ftext.txt","at");	ouvrir un fichier texte existant ou pas existant pour ajouter
f2 = fopen("fic1.txt","a+b");	ouvrir un fichier binaire pour ajouter et modification

Fermeture

En fermant le fichier le système vide (en cas d'écriture il enregistre les données du tampon) et détruit le tampon. La liaison entre le nom du fichier et la variable est coupée.

La fonction *fclose*

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

- valeur rendue – 0 en cas de succès et EOF en cas d'erreur;
- action – stocke tous les données non stockées du tampon dans le fichier, libère les tampons et la variable fich peut être utilisée pour ouvrir un autre fichier

Exemple

```
fclose(fich1);
```

Lecture

La seule fonction qui lit depuis un fichier binaire est la fonction *fread*. Les autres lisent depuis un fichier texte.

La fonction *fread*

Fonction qui lit les données d'un fichier binaire. La lecture se fait par objets. On doit définir la taille de l'objet et le nombre d'objets qui doivent être lus. Le nombre d'octets lus est le produit de ces valeurs.

```
size_t fread(void *ptr, size_t taille, size_t n,FILE *fichier)
```

Paramètres

- *ptr* - pointeur vers la place de la mémoire ou on va stocker les données lues
- *taille* - la taille en octets d'un objet lu
- *n* - la nombre d objets qui doivent être lus
- *fichier* – pointeur vers la structure FILE

- Valeur rendue – le nombre d'objets réellement lus

Fonction *fgetc*

Elle lit un caractère d'un fichier texte.

```
int fgetc(FILE *fichier);
```

- rend le code du caractère lu ou *EOF* en cas d'erreur

Fonction *fgets*

Elle lit des caractères d'un fichier texte dans une chaîne jusqu'à la fin de ligne soit atteinte ou un nombre maximale de caractères soit lu.

```
char *fgets(char *s, int n, FILE * fichier);
```

Paramètres

- s – la chaîne de caractères qui doit recevoir les caractères.
- n – la longueur maximale de la chaîne. Maximum n-1 caractères soient lus.
- fichier – pointeur vers la structure FILE

- Valeur rendue – pointeur vers la chaîne s ou NULL en cas d'erreur ou *EOF*

Cette fonction est analogique à la fonction *gets* mais elle stock le caractère '\n' (nouvelle ligne).

Fonction fscanf

Elle a la même fonctionnalité comme scanf, mais le premier paramètre est un pointeur vers structure *FILE* ouvert comme texte

```
int fscanf( FILE *fich, char *format , adresse1 , adresse2 , ... adressen)
```

Ecriture

La seule fonction qui écrit dans un fichier binaire est la fonction *fread*. Les autres écrivent dans un fichier texte.

Fonction fwrite

Fonction qui écrit les données dans un fichier binaire. L'écriture se fait par objets. On doit définir la taille de l'objet et le nombre d'objets qui doivent être écrits. Le nombre d'octets écrits est le produit de ces valeurs.

```
size_t fwrite(void *ptr, size_t taille, size_t n,FILE *fichier)
```

Paramètres

- ptr - pointeur vers la place de la mémoire d'ou on prandre les valeurs pour les écrire

- taille - la taille en octets d'un objet
- n - la nombre d objets qui doivent être écrits
- fichier – pointeur vers la structure FILE

• Valeur rendue - le nombre d'objets réellement écrits

Fonction *fputc*

Ecrit le caractère *c* dans *fichier*

```
int fputc(int c,FILE *fichier);
```

• rend le code du caractère c ou EOF en cas d'erreur

Fonction *fputs*

Elle écrit la chaîne *s* dans le fichier désigné par le paramètre *fichier*. Elle est analogue de *puts* mais elle n'ajoute pas le caractère '\n'.

```
int *fputs(const char *s, FILE * fichier);
```

Paramètres

- s – la chaîne de caractères qui doit être écrite.
- fichier – pointeur vers la structure FILE

• Valeur rendue – entier non négatif ou EOF en cas d'erreur ou fin-de-fichier

Fonction *fprintf*

Elle a la même fonctionnalité comme *printf*, mais le premier paramètre est un pointeur vers structure *FILE* ouvert comme texte

```
int fprintf( FILE *fich, char *format , expr1 , expr2 , ... exprn)
```

Fonctions de positionnement et d'état

Fonction *feof*

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

• rend non-zero si on atteint la fin de fich et 0 sinon

Fonction *fflush*

Elle écrit tous les données non écrites du tampon dans le fichier

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

• rend 0 ou EOF en cas d'erreur

Fonction rewind

Elle positionne curseur du fichier *fich* au début.

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

Fonction fseek

Elle positionne le curseur du fichier *fich* après enregistrement du tampon au cas d'écriture.

```
int fseek(FILE *fich, long dist, int methode);
```

Paramètres

- fich – pointeur vers la structure FILE
- dist – déplacement relatif en octets (positif – vers la fin du fichier ou negatif – vers le début du ficier) à effectuer

- *methode* – relativement de quel point est le déplacement
  - SEEK_SET(0) – le début du fichier
  - SEEK_CUR(1) – la place courante
  - SEEK_END(2) – la fin du fichier

• Rend 0 en cas de succès et non-zéro en cas d'erreur

Quand on veut ecrire après lecture ou lire après écriture on doit obligatoirement passer par fseek

Exemples

Lire depuis stdin des nombres réels jusqu'à la fin du fichier et stocker les nombres dans deux fichiers – texte et binaire

```
void main (void) {
float nombre;
int m,i =0;
FILE  *nombres_txt, *nombres_bin;
  if ((nombres_txt =fopen("nombres.txt","wt")) != NULL &&
    (nombres_bin =fopen("nombres.dat","wb")) != NULL){
    printf ("tapez plusieurs nombres reels en finissant avec CTRL-Z\n");
     m=scanf("%f",&nombre);
     while(m!=EOF){
          fprintf(nombres_txt,"%8.3f ", nombre);
          if (++i % 4 ==0) fputc('\n',nombres_txt);
          fwrite(&nombre,sizeof(float),1,nombres_bin);
          m=scanf("%f",&nombre);
     }
     fclose (nombres_txt);
     fclose (nombres_bin);
   }
   else
     puts("erreur en ouvrant les fichiers \"nombre\".txt\n");
}
```

D'abord les on ouvre les deux fichiers *nombres.txt* et *nombres.dat* et si l'ouverture est une réussite on commence de lire depuis le clavier des nombres réels et d'écrire chaque nombre lu dans les deux fichiers. Dans le fichier texte on écrit 4 nombres par ligne.

Lire depuis *nombres.txt* des nombres réels jusqu'à la fin du fichier et stocker leurs carrés dans carres.txt

```
void main (void) {
float nombre;
int m,i =0;
FILE  *nombres_txt, *carres_txt;
  if ((nombres_txt =fopen("nombres.txt","rt")) != NULL &&
     (carres_txt =fopen("nombres.dat","wt")) != NULL){
      m=fscanf(nombres_txt,"%f",&nombre);
     while(m!=EOF){
          fprintf(carres_txt,"%8.3f ", nombre*nombre);
          if (++i % 4 ==0) fputc('\n',carres_txt);
          m=fscanf(nombres_txt,"%f",&nombre);
     }
   fclose (nombres_txt);
   fclose (carres_txt);
}
  else
     puts("erreur en ouvrant les fichiers \"nombre\".txt\n");
}
```

Ici on lit des nombres d'un fichier texte et les stocke dans un autre fichier texte. Les nombres sont stockés en lignes par 4.

Lire depuis nombres.dat des nombres réels jusqu'à la fin du fichier et stocker leurs carrés dans carres.dat

```
void main (void) {
float nombre,carre;
int m;
FILE  *nombres_bin, *carres_bin;
  if ((nombres_bin =fopen("nombres.dat","rb")) != NULL &&
     (carres_bin =fopen("carres.dat","wb")) != NULL){
     m = fread(&nombre,sizeof(float),1,nombres_bin);
     while(m>0){
           carre = nombre*nombre;
          fwrite(&carre,sizeof(float),1,carres_bin);
          m = fread(&nombre,sizeof(float),1,nombres_bin);
     }
      fclose (nombres_bin);
      fclose (carres_bin);
  }
  else
     puts("erreur en ouvrant les fichiers \n");
}
```

Ici on lit des nombres d'un fichier binaire et les stocke dans un autre fichier binaire.

Lire depuis un fichier binaire des nombres réels et les remplacer par leurs carrées !

```
void main (void) {
float nombre,carre;
int m,i=0;
FILE  *ncarres_bin;
  if ((ncarres_bin =fopen("ncarres.dat","r+b")) != NULL){
     m = fread(&nombre,sizeof(float),1,ncarres_bin);
     while(m>0){
          carre = nombre*nombre;
          fseek(ncarres_bin,i*sizeof(float),SEEK_SET);
          fwrite(&carre,sizeof(float),1,ncarres_bin);
          i++;
          fseek(ncarres_bin,i*sizeof(float),SEEK_SET);
          m = fread(&nombre,sizeof(float),1,ncarres_bin);
     }
     rewind(ncarres_bin);
     m = fread(&carre,sizeof(float),1,ncarres_bin);
     while(m>0){
          printf(" %10.5f\n", carre);
          m=fread(&carre,sizeof(float),1,ncarres_bin);
     }
     fclose (ncarres_bin);
  }
  else
     puts("erreur en ouvrant les fichier \n");
}
```

Quand écrit à la même place on doit appeler la fonction *fseek* entre chaque *fread* et chaque *fwrite* et entre chaque *fwrite* et chaque *fread*. Chaque *fread* lit un nombre réel et si la lecture est une réussite elle rend 1 comme résultat qui est stocké dans *m*. Dans le moment quand *m* devient 0 (ça denote qu'il n'y a plus de nombres), la fin du fichier est atteint et la boucle est terminée. La deuxième boucle affiche le fichier après le traitement.

Ecrire les données des étudiants dans un fichier binaire

Les structures sont des objets et il n'y a aucune différence. On peut écrire deux variants. Dans les deux variants tous les structures sont écrites par un appel mais dans le deuxième on écrit le nombre d'étudiants d'abord et de cette façon on peut lire le nombre en avant pour que on puisse savoir combien d'étudiants sont enregistrés et de les lire plus tard.

Variant 1

```
void enregistrer(ETUDIANT *g, int n, FILE *f){
     fwrite(g,sizeof(ETUDIANT),n,f);
 }
```

Variant 2

```
void enregistrer(ETUDIANT *g, int n, FILE *f){
        fwrite(&n,sizeof(int),1,f);
     fwrite(g,sizeof(ETUDIANT),n,f);
 }
```

Conversion depuis et vers chaînes

Même que il y a des fonction spéciales pour conversion, il y a deux fonctions analogique à *fprintf* et *fscanf* qui ont comme premier paramètre une chaîne de caractères au lieu d'un pointeur vers fichier. L'effet de ces fonctions et pareil et la seule différence que les caractères ne sont pas dans un fichier mais dans le string.

Fonction *sprintf*

Elle a la même fonctionnalité comme *printf*, mais le premier paramètre est un pointeur vers une chaîne de caractères

```
int sprintf(char *s,char format,expr1,expr2,...,exprn)
```

- rend le nombre d'octets écrits dans s sans le '\0' final

Fonction *sscanf*

Elle a la même fonctionnalité comme scanf, mais le premier paramètre est un pointeur vers une chaîne de caractères

```
int sscanf(char *s, char *format,adr1, adr2,...,adrn)
```

- rend le nombre d'objets convertis ou EOF en cas d'un essaie de lire après la fin de la chaîne

Paramètres du programme

On peut spécifier des arguments du programme comme des paramètres de la fonction main. Dans ce cas on doit taper les les paramètres après le nom du programme dans la fenêtre DOS.

Les paramètres sont deux:

- un entier positif – le nombre d'arguments (n)
- un tableau de strings dont le nombre d'éléments est n. Le premier string est toujours le nom du programme lui-même. Les strings sont séparés par intervalles

```
void main (int count, char argv[])
```

Exemples

Faire un programme qui affiche ses arguments

```
#include <stdio.h>
void main (int count, char *args[]) {
  int i;
  if (count > 1){
      for(i=0;i<count;i++)
          puts(args[i]);
  }
}
```

Faire un programme qui calcule la somme des entiers dans intervalle défini par se premiers deux arguments

```
#include <stdio.h>
void main (int argc, char *args[]) {
     int i,n=0,m=0,s=0;
     if (argc > 0) sscanf(args[1],"%d",&m);
     if (argc > 1) sscanf(args[2],"%d",&n);
     for (i=m; i<=n; i++) s+=i;
     printf("%d\n",s);
}
```