

[LC100]

Accès fichiers

Entrées/Sorties (E/S) sur flux

Un *flux*¹ (*stream* en anglais) est une *source* ou une *destination*² de données qui peut être associée à un fichier ou à un périphérique.

Un flux en *mode texte* est formé d'une suite de lignes, chaque ligne se terminant par une *fin de ligne*².

Un flux en *mode binaire* est constitué d'une suite d'octets non traités.

A chaque flux est associé une structure de données de type FILE. Cette structure est constituée des informations nécessaires à la gestion de ce flux. En fait, comme on accède toujours à un flux par l'intermédiaire d'un pointeur (de type FILE *) on associe souvent le flux à ce pointeur, plutôt qu'à la structure qu'il pointe.

L'ouverture d'un fichier provoque la création du flux associé et retourne donc un pointeur de type

FILE *.

Il existe aussi des flux prédéfinis qui sont ouverts automatiquement au début de l'exécution d'un programme. Il s'agit de :

- **stdin** /* flux d'entrée *std* */
- **stdout** /* flux de sortie *std* */
- **stderr** /* flux de *erreur* *std* */
- **stdprn** /* flux de *sortie imprimante*. (n'existe pas sous Unix) */

Ce sont donc, eux aussi, des objets de type FILE *.

Le fichier d'entête stdio.h

Le fichier d'entête *stdio.h* contient :

- la définition du type sous forme d'une structure,
- les macro-définitions de différentes constantes telles que :

| | | | |
|--------|--------|----------|----------|
| NULL | _IOFBF | _F_RDWR | SEEK_SET |
| EOF | _IOLBF | _F_READ | SEEK_CUR |
| BUFSIZ | _IONBF | _F_WRITE | SEEK_END |
| ... | | ... | |
- la déclaration d'un *tableau externe* d'objets de type FILE,
- les macro-définitions de *stdin*, *stdout*, *stderr* comme adresses des 3 premiers éléments du tableau précédent,
- les *déclarat°* ou *macro définit°* des fonctions d'Entrées/Sorties que nous allons voir.

Remarque :

La structure FILE ne doit pas être manipulée directement. Il s'agit d'une structure "privée" que l'on accède uniquement par les fonctions que nous allons décrire. Précisons cependant qu'elle contient entre autres (sous Unix) :

- un *descripteur* (c'est-à-dire en fait un indice vers la table des *descripteurs de fichiers* du processus),
- un *tampon* (de taille BUFSIZ),
- des informations pour la *gest°* de ce tampon.

¹ on dit aussi *flot*

² une telle *fin de ligne* peut être sujette à *conversion* lorsqu'elle passe dans le flux (voir page suivante)

Conversion des fins de lignes en mode texte

Exemple de fichier texte **LF**
au format **LF**
du système UNIX **LF**
LF
Chaque ligne est **LF**
terminée par un **LF**
saut ligne **LF**

format UNIX

Exemple de fichier **LF CR**
texte au format PC **LF CR**
(MS-DOS ou Win) **LF CR**
LF CR
Chaque ligne se termine **LF CR**
par un saut ligne(LF) **LF CR**
suivi d'un **LF CR**
retour chariot (CR) **LF CR**

format MS-DOS

Exemple de fichier texte **CR**
au format Mac Intosh **CR**
(système MacOS) : **CR**
CR
Chaque ligne est terminée **CR**
par un : **CR**
retour chariot **CR**

format MAC

En langage C : Une fin de ligne `\n` est représentée (en mémoire) par le caractère **LF** . Ceci impose de faire une conversion (sauf dans le cas d' **UNIX** bien entendu) lorsqu'on écrit ou qu'on lit du texte.

Par contre, en *binaire* , il ne faut pas faire de conversion !

Fonctions d'entrées/sorties sur flux

Les protocoles des fonctions qui suivent sont données dans le document : "La bibliothèque standard ANSI" avec éventuellement d'autres informations. Elles sont associées à un flux, soit par le fait qu'elles retournent un pointeur , soit parce que l'un de leurs arguments est de ce type. En outre il faut noter que certaines de ces fonctions sont en fait des macros.

Fonctions d'ouvertures et de fermetures de flux

Pour les ouvertures de fichiers, c'est la fonction `fopen` qui est principalement utilisée. L'un de ses arguments est le *mode* d'ouverture du fichier : On peut choisir entre *texte* ou *binaire*, mais aussi entre *lecture*, *écriture* ou *ajout.*, éventuellement combinés (voir document "La bibliothèque standard ANSI").

Mais il existe aussi *freopen* pour réouvrir un flux existant sur un nouveau fichier, et pour créer un fichier temporaire.

La fonction *fclose* sert pour les fermetures des fichiers.

Fonctions d'entrées/sorties sur flux (orientées texte)

Ces fonctions permettent de lire ou écrire dans des flux ouverts par les fonctions d'ouvertures précédentes. Il s'agit de :

fprintf écriture formatée,

fscanf lecture formatée,

fputc écriture d'un caractère

putc écriture d'un caractère,

fgetc lecture d'un caractère

getc lecture d'un caractère,

Les deux équivalences ci-dessus ne sont pas absolues, car **putc** et **getc** sont en fait (en général) des macros au lieu d'être des fonctions. Elles sont de ce fait plus rapides d'exécution, mais à manier avec les précautions usuelles pour les macros.

fputs écriture d'une chaîne,

fgets lecture d'une chaîne,
ungetc remise d'un caractère dans le tampon d'entrée.

Les fonctions d'E/S standard

Ce sont naturellement les plus connues. En fait on a les équivalences suivantes :

printf(...) \Leftrightarrow **fprintf(,...)**

scanf(...) \Leftrightarrow **fscanf(,...)**


putchar(c) \Leftrightarrow **putc(c,)**


fputchar(c) \Leftrightarrow **fputc(c,)**

getchar() \Leftrightarrow **getc()**

fgetchar() \Leftrightarrow **fgetc()**

Ici encore, certaines des fonctions précédentes sont en fait des macros. Il s'agit normalement (à vérifier sous Unix) de **fputchar**, **putchar** et **getchar**. Les fonctions qui suivent (**puts** et **fput**, et de même, **gets** et **fgets**) ne sont pas équivalentes deux à deux.

 **puts** remplace **\0** final par **\n**, alors que **fputs** ne le fait pas.

 **gets** remplace **\n** par **\0**, alors que **fgets** ne le fait pas et rajoute simplement un **\0** final. En outre **fgets** possède un argument supplémentaire qui doit être égal au nombre maximum de caractères que peut recevoir son premier argument³.

C'est pour cette raison qu'il vaut mieux faire une saisie de chaîne de caractères en utilisant **fgets(...,stdin)** plutôt que **gets**. Mais ne pas oublier dans ce cas que le dernier caractère saisi peut-être **\n**, et que, si on ne souhaite pas le conserver, il faudra le tester pour éventuellement l'annuler.

Exemple :

```
char chain[NBCAR]; unsigned lo;  
...  
fgets(chain,NBCAR,stdin);  
lo=strlen(chain); if (chain[lo-1]=='\n') chain[--lo]='\0';
```

Par ailleurs, la fonction **scanf** laisse la fin de ligne⁴ dans le tampon⁵ d'entrée. Il en est de même de la fonction **getchar**. Ceci peut être gênant si l'on utilise ensuite une fonction **gets** ou **fgets**. Dans ce cas il sera nécessaire de rajouter un **getchar()** juste avant l'appel de **gets** ou **fgets** afin de lire cette fin de ligne résiduelle.

Exemple :

```
scanf ("%d%lg", &nb, &rd);  
...  
getchar();  
fgets(chain,NBCAR,stdin);
```

Une autre solution consiste à faire toutes les saisies avec **fgets** suivi éventuellement de **sscanf** pour analyser la chaîne de caractères obtenue.

³ y compris l'octet nul **\0** de fin de chaîne

⁴ si vous lui faites lire des données sur plusieurs lignes, elle ne laissera que la dernière fin de ligne

⁵ buffer en anglais

Exemple :

```
fgets(chain,NBCAR,stdin);
sscanf(chain,"%d %s %g",&n,ch,&r);
```

Il faut enfin citer d'autres fonctions, utilisables sous PC uniquement. Ce sont par exemple **getch** et **getche** qui permette des saisies de caractères isolés (sans utiliser la touche <entrée>) ou encore la fonction **putch** effectuant une sortie directe à l'écran... Ces fonctions nécessitent le fichier d'entête *conio.h* qui n'existe pas sous Unix.

```

/**** exemple 1 : Lecture et analyse d'un fichier texte ****/

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

typedef enum {NO,OUI} bool;
#define SZBUF 128 /* taille d'un buffer */
#define ERREUR(N,MESS) \
{printf("Fichier %s ligne %d - Erreur n° %d : %s\n",nfich,li,N,MESS);\ exit(N);}

FILE * fich;

char car,buffli[SZBUF+1];
int li; /* n° ligne */
extern int AmbR,AmbV,AmbB; /* couleur d'ambiance en RVB */
extern float oeilx,oeily,oeilz; /* coordonnees de l'oeil */

int LITOBJET(char * nfich) /* Fonction de lecture du fichier d'objets */
{
fich=fopen(nfich,"r"); /* ouverture et test */
if (fich==NULL) ERREUR(2,"Ouverture impossible");
li=1;

/* lecture ligne A (couleur ambiante en RVB) et test */
while ((car=getc(fich))!=EOF && car!='A')
{ li++; if (car != '\n') fgets(buffli,SZBUF,fich); }
if (car==EOF) ERREUR(14,"Ligne A (couleur ambiante en RVB) manquante");
if (fscanf(fich,"%d %d %d",&AmbR,&AmbV,&AmbB)!=3)
ERREUR(15,"Ligne A erronée");
li++; fgets(buffli,SZBUF,fich);

/* lecture ligne E (position de l'oeil) et test */
while ((car=getc(fich))!=EOF && car!='E')
{ li++; if (car != '\n') fgets(buffli,SZBUF,fich); }
if (car==EOF) ERREUR(14,"Ligne E (position de l'oeil) manquante");
if (fscanf(fich,"%lf %lf %lf",&oeilx,&oeily,&oeilz)!=3)
ERREUR(15,"Ligne E erronée");
li++; fgets(buffli,SZBUF,fich);

/* ... (suite du traitement) ... */

/* fermeture du fichier */
fclose(fich);
return(0);
}

```

Fonctions d'entrées/sorties sur flux (orientées binaires)

Ces fonctions sont destinées à des lectures ou écritures d'enregistrements (définis comme des structures ou simplement des tableaux) depuis ou vers des fichiers binaires. Il s'agit de :

fwrite écriture

fread lecture

Il ne faut pas les confondre avec **write** et **read** qui sont (au moins sous Unix) des primitives système, et qui sont de ce fait moins portables. Il ne faut pas les confondre non plus avec les fonctionbs **_write** et **_read** qui sont des primitives système sous PC et qui ne sont pas non plus portables.

```
/**** exemple 2 :   Ecriture d'une image en format TGA   *****/

int save(char *nom,unsigned EX,unsigned EY)
{
    static unsigned char * buffer;
    static FILE * ftga;
    unsigned header[9]={0,2,0,0,0,0,0xffff,0xffff,0x001B};
    int i,j,k,nb;
    int coul,Rouge,Vert,Bleu;

    ftga=fopen(nom,"r"); /* ouverture fich. en lecture pour tester existence */
    if (ftga!=NULL)
        { printf("ATTENTION ! Le fichier %s existe déjà \n",nom); return -2;}

    if (!(ftga=fopen(nom,"wb"))) /* OUVERTURE du fichier en ECRITURE */
    {
        printf("Création du fichier %s impossible\n",nom); return -3;    }
    header[6]=EX;
    header[7]=EY;
    k=fwrite(header,sizeof(unsigned),9,ftga);
    if (k!=9) {printf("Erreur d'écriture fichier TGA"); return 1;}

    if (!(buffer=(unsigned char *)malloc(3*EX*sizeof(char))))
        { printf("Plus de mémoire !"); return(-1); }

    for (i=EY-1;i>=0;i--) /* boucle d'écriture des lignes de l'image */
    {
        for(j=0;j<EX;j++) /* boucle de préparation des pixels d'une ligne */
        {
            coul=getpixel(j,i);
            getpal(coul,&Rouge,&Vert,&Bleu);
            buffer[3*j]=(unsigned char) Bleu<<2;
            buffer[3*j+1]=(unsigned char) Vert<<2;
            buffer[3*j+2]=(unsigned char) Rouge<<2;
        }
        k=fwrite(buffer,sizeof(char),nb=3*EX,ftga);
        if (k!=nb) {printf("Erreur d'écriture fichier TGA"); return 1;}
    }
    free(buffer);

    k=fclose(ftga);
    if (k==0) {printf("Erreur en fermeture du fichier TGA"); return 1;}
    return(0);
}
```

Fonctions de gestion du tampon (buffer)

- A chaque flux⁶ est associé un *tampon* de taille BUFSIZ⁷.
- On peut redéfinir ce tampon (juste après l'ouverture du flux) grâce aux fonctions `setvbuf` et `setbuf` :

```
int setvbuf(FILE * fl, char * tampon, int mode, size_t taille);
```

| valeur de mode | critère de vidage |
|---------------------|------------------------------|
| <code>_IOFBF</code> | Le tampon est plein |
| <code>_IOLBF</code> | Le tampon contient une ligne |
| <code>_IONBF</code> | Systématique (pas de tampon) |

Si tampon = NULL

→ *alloué par malloc*

→ la valeur retournée est : 0 si OK $\neq 0$ { si mode ou taille incorrecte
ou
si pas assez de mémoire pour allocation

```
void setbuf(FILE * fl, char * tampon);
```

Si tampon = NULL

→ mode *IONBF*
(pas de tampon)

sinon mode *IOFBF* avec
tampon de taille BUFSIZ

Erreur fréquente : allouer tampon comme variable locale et oublier de fermer le fichier avant de quitter la fonction où elle a été déclarée.

- On peut vider le tampon de sortie en utilisant la fonction `fflush` :

→ En écriture : Le tampon est *évacué* dans le fichier

→ En lecture : Son effet est indéfini, non imposé par la norme.
Elle est cependant souvent utilisable pour l'entrée standard (*stdin*) pour simplement vider le tampon, mais cet effet n'est pas garanti. A éviter donc !

→ la valeur retournée est : 0 si OK EOF si erreur

Fonctions donnant des informations sur les fichiers

Ces fonctions ne nécessitent pas d'avoir ouvert un flux

```
#include <sys/stat.h>
```

```
int stat(char * chemin, struct stat * ptrstat);
```

```
int fstat(int desc, struct stat * ptrstat);
```

```
#include <unistd.h>            /* sous Unix */
```

```
#include <ioh>                /* sous PC */
```

```
int access(char * chemin, int mode_d'accès); /* vérifier si un fichier a les  
droits d'accès donnés en deuxième argument */
```

⁶ sauf les flux d'E/S standard (*stdin*, *stdout*, *stderr*) non redirigés

⁷ constante définie (rappelons le) dans *stdio.h*

taille = (int) statistique. off - t

taille = (int) statistique. off - 1

```
dev_t      st_dev;      /* identification de disque logique du fichier */
ino_t      st_ino;      /* numéro du fichier (i-nœud) sur ce disque */
mode_t     st_mode;     /* type du fichier et droits d'accès des utilisateurs */
nlink_t    st_nlink;    /* nombre de liens physiques */
uid_t      st_uid;      /* n° du propriétaire du fichier */
gid_t      st_gid;      /* n° du groupe propriétaire du fichier */
off_t      st_size;     /* taille du fichier ( si cela a un sens. ) */
time_t     st_atime;     /* date de dernier accès */
time_t     st_mtime;     /* date de dernier modification */
time_t     st_ctime;     /* date de dernier modification du nœud */
}
```

Interprétation du bit

| Nom symbolique du bit | | Interprétation du bit | |
|-----------------------|---------|-----------------------|--|
| | | } | type du fichier |
| | | | |
| | | | |
| | S_ISUID | | <i>set-uid bit</i> |
| | S_ISGID | | <i>set-gid bit</i> |
| | | | <i>sticky bit</i> |
| S_IRWXU { | S_IRUSR | | lecture par le propriétaire |
| | S_IWUSR | | écriture par le propriétaire |
| | S_IXUSR | | exécution par le propriétaire |
| S_IRWXG { | S_IRGRP | | lecture par les membres du groupe propriétaire |
| | S_IWGRP | | écriture par les membres du groupe propriétaire |
| | S_IXGRP | | exécution par les membres du groupe propriétaire |
| S_IRWXO { | S_IROTH | | lecture par les autres utilisateurs |
| | S_IWOTH | | écriture par les autres utilisateurs |
| | S_IXOTH | | exécution par les autres utilisateurs |

Fonctions permettant de faire des opérations sur les fichiers

Ces fonctions ne nécessitent pas non plus d'ouvrir un flux

```
#include <unistd.h>          ou      <stdio.h>
```

```
int unlink(char * chemin);  
int remove(char * chemin);
```

```
int rename(char * ancien_chemin, char * nouveau_chemin);
```

Attention : En fait cette fonction *rename* change le lien. Donc, si les répertoires d'accès sont différents, la fonction effectue un déplacement du fichier. Par ailleurs si la destination (nouveau chemin) est un fichier existant, il sera écrasé. Enfin si les deux chemins correspondent à des fichiers existants, ils doivent être de même type ; c'est-à-dire que si l'un est un répertoire, l'autre doit être aussi un répertoire.

```
#include <unistd.h>          /* Unix */
```

```
int link(char * chemin, char * nouveau_chemin);
```

```
#include <sys/stat.h>
```

```
int chmod(char * chemin,      mode_t  
                                int      mode_d_accès);
```

```
int fchmod(int * desc,      mode_t  
                                int      mode_d_accès);
```

Fonctions de pseudo entrées/sorties sur chaînes

Voir le document "[La bibliothèque standard ANSI](#)" §1.4

Fonctions de positionnement dans les fichiers

Voir le document "[La bibliothèque standard ANSI](#)" §1.5

Fonctions de gestion des erreurs (E/S ou autres)

Voir le document "[La bibliothèque standard ANSI](#)" §1.6

Ajoutons y également la fonction *strerror* :

```
char * strerror (int n)
```

qui retourne un pointeur sur la chaîne correspondant à l'erreur n° n (cf à ce sujet la variable globale *errno*).

*...Et lorsqu'un fichier ouvriras
Par exemple t'assureras :
Pour le lire
Qu'il existe déjà ;
Pour l'écrire
Qu'il n'existe pas !*

*Pour finir prendras bien soin,
Quand tu n'en auras plus besoin,
De bien refermer
Tous les fichiers.*

Quelques constantes représentant des valeurs possibles pour errno

| | | |
|-----------|---|---|
| E2BIG | Liste d'arguments trop longue | /* Arg list too long */ |
| EACCES | Accès refusé | /* Permission denied */ |
| EBADF | Descripteur de fichier incorrect | /* Bad file number */ |
| EDEADLK | Interblocage | /* A deadlock would occur */ |
| EEXIST | Fichier existe déjà | /* File exists */ |
| EILSEQ | Séquence illégale d'octets | /* Illegal byte sequence */ |
| EINVAL | Argument invalide | /* Invalid argument */ |
| EMFILE | Trop de fichiers ouverts | /* Too many open files */ |
| ENODEV | Périphérique inexistant | /* No such device */ |
| ENOENT | Fichier ou répertoire inexistant | /* No such file or directory */ |
| ENOEXEC | Format incorrect lors d'un appel à exec | /* Exec format error */ |
| ENOLCK | Table des verrous pleine | /* System record lock table was full */ |
| ENOMEM | Manque de mémoire | /* Not enough core */ |
| ENOTEMPTY | Répertoire non vide | /* Directory not empty */ |
| EXDEV | Liens entre disques différents | /* Cross-device link */ |

/* Spécifiques à UNIX */

| | | |
|---------|--------------------------------------|--|
| EAGAIN | Création de processus impossible | /* UNIX: No more processes */ |
| EBUSY | | /* UNIX: Mount device busy */ |
| ECHILD | Pas de processus fils | /* UNIX: No children */ |
| EFAULT | Mauvaise adresse | /* UNIX: Bad address */ |
| EBIG | Fichier trop gros | /* UNIX: File too large */ |
| EINTR | Appel système interrompu | /* UNIX: interrupted system call */ |
| EIO | Erreur d'entrée/sortie | /* UNIX: I/O error */ |
| EISDIR | Est un répertoire | /* UNIX: Is a directory */ |
| EMLINK | Trop de liens | /* UNIX: Too many links */ |
| ENFILE | Débordement table descr. de fichiers | /* UNIX: File table overflow */ |
| ENOSPC | Manque de place sur périphérique | /* UNIX: No space left on device */ |
| ENOTBLK | Nécessite un périph. orienté bloc | /* UNIX: Block device required */ |
| ENOTDIR | N'est pas un répertoire | /* UNIX: Not a directory */ |
| ENOTTY | N'est pas un terminal type télétype | /* UNIX: Not a typewriter */ |
| ENXIO | Adresse ou périph. inexistant | /* UNIX: No such device or address */ |
| EPERM | Accès interdit (non ROOT) | /* UNIX: Not super-user */ |
| EPIPE | Tube rompu (tube sans lecteur) | /* UNIX: Broken pipe */ |
| EROFS | Disque accessible unikt en lecture | /* UNIX: Read only file system */ |
| ESPIPE | Essai illégal positionnemt sur tube | /* UNIX: Illegal seek on pipe */ |
| ESRCH | Processus inexistant | /* UNIX: No such process */ |
| ETXTBSY | Fichier texte occupé | /* UNIX: Text file busy */ |
| EUNATCH | | /* UNIX: Protocol driver not attached */ |

/* Spécifiques à HPUNIX */

| | |
|---------|---|
| EBADMSG | /* HPUNIX: trying to read unreadable message */ |
| ECHNRG | /* HPUNIX: Channel number out of range */ |
| ECOMM | /* HPUNIX: Communication error on send */ |
| EIDRM | /* HPUNIX: Identifier removed */ |
| ELNRNG | /* HPUNIX: Link number out of range */ |
| ENOMSG | /* HPUNIX: No message of desired type */ |
| ENONET | /* HPUNIX: Machine is not on the network */ |
| ENOPKG | /* HPUNIX: Package not installed */ |
| ENOSTR | /* HPUNIX: Device not a stream */ |
| EPROTO | /* HPUNIX: Protocol error */ |
| ETIME | /* HPUNIX: timer expired */ |

/* Spécifiques aux compatibles PC (Borland) */

| | |
|---------|--|
| ECONTR | /* PC: Memory blocks destroyed */ |
| ECURDIR | /* PC: Attempt to remove CurDir */ |
| EDOM | /* PC: Math argument */ |
| EINVACC | /* PC: Invalid access code */ |
| EINVDAT | /* PC: Invalid data */ |
| EINVDRV | /* PC: Invalid drive specified */ |
| EINVENV | /* PC: Invalid environment */ |
| EINVFMT | /* PC: Invalid format */ |
| EINVFNC | /* PC: Invalid function number */ |
| EINVMEM | /* PC: Invalid memory block address */ |
| ENMFILE | /* PC: No more files */ |
| ENOFILE | /* PC: File not found */ |
| ENOPATH | /* PC: Path not found */ |
| ENOTSAM | /* PC: Not same device */ |
| ERANGE | /* PC: Result too large */ |
| EZERO | /* PC: Error 0 */ |