

**Langage C**  
**fichiers texte**

**Wieslaw Zielonka**  
**[zielonka@irif.fr](mailto:zielonka@irif.fr)**

## ouverture de fichier

```
FILE *fopen(const char *nom_fichier,  
            const char *mode)
```

l'ouverture de fichier ../file.txt en lecture

```
FILE *f1ot = fopen( "../file.txt", "r");
```

# ouverture de fichier

En ouvrant le fichier il faut décider :

- si le fichier est ouvert
  - en lecture seulement,
  - en écriture seulement,
  - en lecture et en écriture.
- pour le fichier ouvert en écriture ou lecture+écriture il faut indiquer
  - (a) est-ce que le contenu de fichier est effacé à l'ouverture (fichier vide au début) ou
  - (b) le contenu de fichier est préservé à l'ouverture
- pour le fichier ouvert en écriture ou lecture+écriture il faut indiquer
  - (c) si l'écriture se fait à la position courante dans le fichier ou
  - (d) si l'écriture se fait toujours à la fin du fichier, c'est-à-dire pour chaque opération d'écriture la position courante se déplace à la fin du fichier.














# ouverture d'un fichier

```
FILE *fopen(const char *nom_fichier, const char *mode)
```

Six modes possibles d'ouverture :

- "r" – ouvre le fichier en lecture, position initiale au début du fichier, erreur si le fichier n'existe pas
- "w" -- ouvre le fichier en mode écriture, si le fichier existait le contenu est écrasé à l'ouverture, si le fichier n'existait pas il est créé
- "a" -- ouvre le fichier en mode "append" (ajout), chaque écriture se fait à la fin de fichier, si le fichier n'existe pas il est créé
- "r+" - ouvre le fichier en mode lecture/écriture, la position initiale au début de fichier, si le fichier n'existe pas alors erreur
- "w+" - ouvre le fichier en mode lecture/écriture, si le fichier est non vide le contenu est écrasé, si le fichier n'existe pas alors erreur
- "a+" – ouvre le fichier en lecture/écriture, si le fichier n'existe pas alors il est créé, l'écriture se fait à la fin de fichier

## résumé de six modes d'ouverture

	r	w	a	r+	w+	a+
fichier doit déjà existé ?						
le contenu du fichier est écrasé à l'ouverture ?						
lecture du flot						
écriture dans le flot 0 partir de la position courante						
écriture uniquement à la fin du flot (mode append)						

## mode d'ouverture d'un fichier et la position courante

Juste après l'ouverture la position courante dans le fichier est la position 0 (juste avant le premier octet du fichier).

Chaque lecture et chaque écriture changent la position courante dans le fichier.

Si l'écriture ne se fait pas à la fin de fichier alors les octets écrits dans le fichier **effacent** les octets qui se trouvent dans le fichier.

# fermeture de flot

```
int fclose(FILE *f) {
```

`fclose()` retourne 0 si OK et EOF en cas d'erreur.

# Fichiers texte



# lecture caractère par caractère

`int fgetc(FILE *f)`

retourne le caractère lu. La fonction retourne EOF si la fin de fichier ou en cas d'erreur.

`int getc(FILE *f)` une macro-fonction, l'effet identique à `fgetc()`

`int getchar(void)` équivalent à `getc(stdin)`

Pour le traitement correcte de **EOF** et d'erreurs il faut déclarer comme `int` les variables qui reçoivent le résultats de ces fonctions.

```
int i;
```

```
while( ( i = fgetc( file ) ) != EOF ){
```

```
    /* traiter le caractère lu */
```

```
}
```

# remettre un caractère dans le flot

```
int ungetc(int c, FILE *floc)
```

remet le caractère c dans le flot. La lecture d'un caractère qui suit ungetc() retournera c.

ungetc() retourne c si l'opération réussit et EOF en cas d'échec

C garantit qu'une opération ungetc() doit réussir mais pas de garantie qu'une suite d'opérations ungetc() réussisse.

# écriture caractère par caractère

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

écrit le caractère c dans le flot, retourne c si OK et EOF en cas d'erreur.

`int putc(int c, FILE *f)` même chose mais implémentée comme une macro-fonction

`int putchar(int c)` équivalent à `putc(c, stdout)`

# lecture d'une ligne

une ligne : une suite de caractères qui termine par le caractère '\n'

`char *fgets(char *s, int n, FILE *f, ...)`

lit au plus  $n-1$  caractères et les place à l'adresse `s`. La lecture s'arrête si la fonction rencontre le caractère '\n' qui sera aussi recopié dans `s`. La fonction place '\0' à la fin de la suite de caractères lus.

`fgets()` retourne `s` si tout est OK ou `NULL` si la fin de fichier ou en cas d'erreur.

# écriture de chaînes de caractères dans un fichier

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

s pointeur vers une chaîne de caractère (qui termine avec '\0').

fputs ( ) écrit les caractères de la chaîne pointée par s dans le flot (**sans jamais écrire le caractère '\0'** ).

fputs ( ) retourne un nombre non-négatif si OK et EOF en cas d'erreur.

**fputs ( ) n'est pas adapté pour écrire dans des fichiers binaires (qui peuvent contenir le caractère '\0' ).** fputs ( ) est à utiliser uniquement avec les fichiers texte.

```
int puts( const char *str )
```

puts écrit la chaîne pointée par str dans le flot stdout et écrit le caractère '\n' à la suite de str (contrairement à fputs() qui n'ajoute pas '\n' à suite de caractères écrits).

# Sorties formatées

# sorties formatées

```
int fprintf(FILE *fplot, const char *format, ...)
```

```
int printf(const char *format, ...)
```

**entrées formatées**



# entrées formatées

```
int fscanf(FILE *fplot, const char *format,...)
```

```
int scanf(const char *format, ...)
```

les fonctions lisent depuis un fichier texte, transforment le texte lu en une suite de valeurs selon le format.

`scanf( .... )` est équivalent à `fscanf( stdin, .....)`

les fonctions retournent le nombres de "match" (les nombres de variables sur la liste ... dont la valeur est lue) ou **EOF** si fin de flot ou erreur.

## trois flots standard

### Trois flots

- `stdin` – flot d'entrée standard
- `stdout` – flot sortie standard
- `stderr` – flot sortie d'erreurs standard

sont déjà ouverts au début de l'exécution du programme. (Il est possible de les fermer au lancement de programme).

`stdin`, `stdout`, `stderr` sont des objets de type `FILE *` donc les opérations lecture/écriture qu'on applique aux fichiers s'appliquent aussi à ces trois flots.

# sortie dans un fichier ou sur stdout ?

```
FL0T *f;

if( sortie standard ){
    f = stdout;
}else if( sortie dans fichier ){
    f = fopen( chemin, "w" );
    if( f == NULL ){
        perror( "open" );
        . . . .
    }
}

/* écriture sur le flot f */
```

# Fichiers binaires

# lecture/écriture dans un fichier binaire

Le fichier binaire : on stocke les données de n'importe quel type **sans les faire transformer en texte**. Une zone de mémoire est directement copiée dans le fichier.

En lecture : une suite d'octets est transférée depuis le fichier vers la mémoire à l'adresse indiquée.

# lecture/écriture binaires

```
size_t fwrite(const void *buf, size_t size,  
              size_t nitems, FILE *fptr)
```

`fwrite()` écrit `nitems` éléments d'un tableau à l'adresse `buf`, chaque élément est de taille `size`.

`fwrite()` retourne le nombre d'éléments écrits.

# exemple : fichier binaire - écrire dans un fichier

```
#define SIZE 100

double tab[ SIZE ];

// remplir le tableau

//écrire le tableau dans un fichier "nombres"

FILE *flot = fopen("nombres", "w");

if( flot == NULL ){ perror( "fopen" ); exit(1); }

/* écrire le vecteur tab dans le fichier binaire */

size_t n = fwrite( tab, sizeof( tab[0] ), SIZE, flot);

fclose(flot);
```

Le fichier sera de longueur de `sizeof(double) * 100` octets.

# exemple : fichier binaire - écrire dans un fichier

```
#define SIZE 100

double tab[ SIZE ];

// remplir le tableau

//écrire le tableau dans un fichier "nombres"

FILE *flot = fopen("nombres", "w");

if( flot == NULL ){ perror( "fopen" ); exit(1); }

/* écrire le vecteur tab dans le fichier binaire */

size_t n = fwrite( tab, sizeof( tab[0] ), SIZE, flot);

fclose(flot);
```

Le fichier sera de longueur de `sizeof(double) * 100` octets.



# lecture d'un fichier binaire

```
size_t fread(void *buf, size_t size,  
             size_t nitems, FILE *fptr)
```

`fread()` lit `nitems` éléments de taille `size` et les place à l'adresse `buf`.

`fread()` retourne le nombre d'éléments lus, ou 0 en cas d'erreur ou à la fin du fichier.

# exemple : fichier binaire - lire un fichier

```
#define SIZE 1024
```

```
double tab[ SIZE ];
```

```
FILE *fnot = fopen("nombres", "r");
```

```
if( fnot == NULL ){ perror( "fopen" ); exit(1); }
```

```
/* lire le vecteur tab dans un fichier binaire */
```

```
size_t n = fread( tab, sizeof( tab[0] ), SIZE, fnot);
```

```
/* n le nombre d'éléments lus, peut être inférieur
```

```
à SIZE si le fichier contient moins de nombres */
```

# lecture/écriture binaires

Attention à la portabilité de fichier binaire.

Un int sur deux machines différentes peut avoir une représentation binaire différente (le nombre d'octets différent ou l'ordre d'octets différents - little endian ou big endian).

Le fichier binaire n'est pas portable d'une machine à l'autre.

```
int i = 1; // en hexa 00 00 00 01
```

est stocké en mémoire

- sur une machine petit-boutiste (little-endian) dans l'ordre  
01 00 00 00

c'est-dire les octets rangés dans la mémoire dans l'ordre croissant de poids, l'octet de poids faible le premier

- sur une machine gros-boutiste (big-endian) dans l'ordre  
00 00 00 01

c'est-dire les octets rangés dans la mémoire dans l'ordre inverse de poids, l'octet de poids faible le dernier

## exemple : copier un fichier caractère pas caractère

```
FILE *source = fopen(nom_fichier_source, "r");  
FILE *dest = fopen(nom_fichier_dest, "w");
```

```
/* copier caractère par caractère */
```

```
int c;
```

```
while( ( c = fgetc( source ) ) != EOF ){
```

```
    if( fputc(c, dest) == EOF ){
```

```
        /* traiter erreur de fputc() */
```

```
    }
```

```
}
```

# exemple : copier un fichier bloc par bloc

Le code suivant convient aussi bien pour les fichiers binaires que pour les fichiers texte

```
FILE *source = fopen(nom_fichier_source, "r");
FILE *dest = fopen(nom_fichier_dest, "w");

/* copier bloc par bloc */
size_t s, u;

#define LEN 1024
char buf[LEN];

while( ( s = fread( buf, 1, LEN, source ) ) > 0 ){
    if( ( u = fwrite(buf, 1, s, dest) ) < s ){
        /* traiter erreur de fwrite */
    }
}
```

noter que le nombre d'octets écrits dans le fichier dest doit être le même que le nombre d'octets lus dans le fichier source

# copier un fichier - temps d'exécution

taille de fichier 2351369 octets :

caractère par caractère ( fgetc() -> fputc() )

```
time ./fcp qmbook.pdf qmbook2.pdf
```

```
real 0m0.355s
user 0m0.330s
sys 0m0.013s
```

---

Par le blocks de caractères ( fread() -> fwrite() )

```
time ./fcp_tampon_fwrite qmbook.pdf qmbook2.pdf
```

```
real 0m0.024s
user 0m0.002s
sys 0m0.012s
```

tampon de 1024 octets

```
time ./fcp_tampon_fwrite qmbook.pdf qmbook2.pdf 4096
```

```
real 0m0.020s
user 0m0.002s
sys 0m0.007s
```

tampon de 4096 octets

**Position courante  
dans le fichier**

# contrôle de la position courante dans un fichier

Chaque lecture/écriture modifie la position courante.

`long ftell(FILE *f)`

retourne la position courante (-1 en cas d'erreur).

`int rewind(FILE *f)`

ramène la position courante au début du fichier (à la position 0)



# contrôle de la position courante dans un fichier

Pour changer la position courante :

```
int fseek(FILE *f, long offset,  
          int origine)
```

origine :

- SEEK\_SET à partir du début du fichier
- SEEK\_CUR à partir de la position courante
- SEEK\_END à partir de la fin du fichier

fseek() retourne 0 si OK et -1 si échec

# changer la position courante : exemples

positionner sizeof(int) octets avant la fin du fichier:

```
fseek( file, -sizeof( int ), SEEK_END );
```

reculer sizeof( double ) octets par rapport à la position courante :

```
fseek( file, -sizeof( int ), SEEK_CUR );
```

revenir au début du fichier :

```
fseek( file, 0 , SEEK_SET );
```

# contrôle de position courante dans un fichier

On ne peut pas aller à une position avant le début de fichier :

```
fseek( file , -10, SEEK_SET );
```

```
/* 10 octets avant le début du fichier :  
incorrect */
```

# contrôle de position courante dans un fichier

```
FILE * file= fopen("toto.txt", "w");
```

```
fputs("debut", file);
```

```
fseek(file, 1000000, SEEK_END);
```

```
fputs("fin", file);
```

```
fclose(file);
```

Qu'est-ce contient le fichier toto.txt ?

- le fichier commence par les 5 caractères :       debut
- suivis de 1000000 caractères nul:       '\0'
- et à la fin le fichier termine avec trois caractères :   fin

Conclusion : les "trous" dans le fichier sont remplis avec le caractère nul '\0'

# contrôle de position courante dans un fichier

Pour lire et changer la position courante dans un fichier dont la longueur est plus longue que LONG\_MAX il faut utiliser les fonctions

```
int fgetpos(FILE *f, fpos_t *position)
```

mémoirise dans position la position courante du flot

```
int fsetpos(FILE *f,  
            const fpos_t *position)
```

le flot revient à la position sauvegardée dans position.

# alterner les lectures et écritures sur le même flot

Si le flot est ouvert en lecture ET écriture ( les modes "r+" "w+" "a+" ) alors

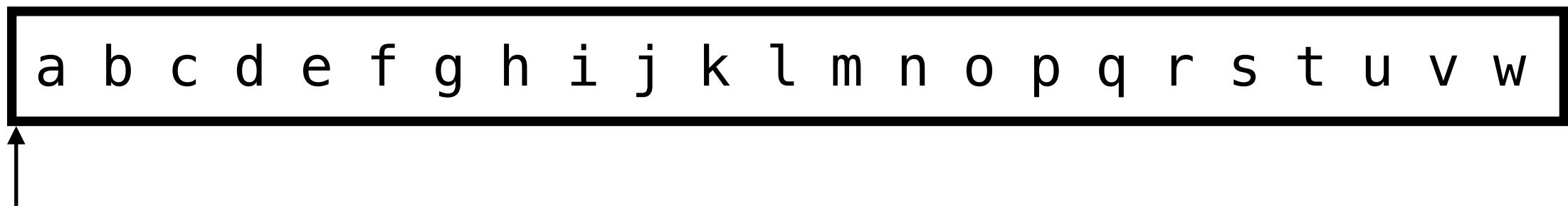
- si une écriture est suivi d'une lecture il faut insérer un appel à une des fonctions : `fflush()`, `fseek()`, `rewind()` entre l'écriture et la lecture,
- si une lecture est suivi d'une écriture il faut insérer un appel à une des fonctions : `fseek()` ou `rewind()` entre la lecture et l'écriture.

Si ces consignes ne sont pas respectées le comportement de votre programme est imprévisible.

# contrôle de la position courante - exemple

Ouvrir en fichier en lecture et écriture sans écrasement de contenu, avec l'écriture possible à l'intérieur de fichier. On supposera que c'est un fichier texte (sans caractère '\0' à l'intérieur).

```
FILE *flot = fopen( argv[1], "r+" );
```



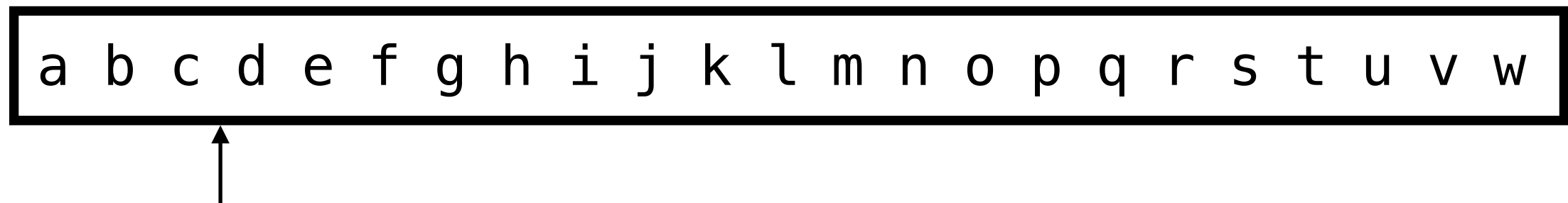
a b c d e f g h i j k l m n o p q r s t u v w

An upward-pointing arrow is positioned at the left edge of the first character 'a' in the buffer.

Pourquoi l'ouverture "r+" ?

Lire trois premiers caractères dans un tampon t.

```
char t[10];  
fgets( t, 4, flot);  /* fgets(. , 4, .) lit au plus 3 char  
                     * et met ensuite '\0' dans le tampon t */
```



a b c d e f g h i j k l m n o p q r s t u v w

An upward-pointing arrow is positioned at the left edge of the third character 'c' in the buffer.

# contrôle de la position courante - exemple

Remplacer 3 char suivants par "###" .

```
fseek(flott, 0L, SEEK_CUR);    /* fseek() pour séparer lecture-écriture */  
char *s="###";  
fputs(s, flott);
```

a b c # # # g h i j k l m n o p q r s t u v w



Ajouter "###" à la fin du flott:

```
fseek(flott, 0L, SEEK_END);  
fputs(s, flott);
```

a b c # # # g h i j k l m n o p q r s t u v w # # #





# contrôle de la position courante - exemple

Remplacer 3 char suivants par "###" .

```
fseek(flot, 0L, SEEK_CUR);    /* fseek() pour séparer lecture-écriture */  
char *s="###";  
fputs(s, flot);
```

a b c # # # g h i j k l m n o p q r s t u v w



Ajouter "###" à la fin du flot:

```
fseek(flot, 0L, SEEK_END);  
fputs(s, flot);
```

a b c # # # g h i j k l m n o p q r s t u v w # # #



## modifier le contenu de fichier

On suppose que le fichier contient un vecteur de int. La fonction suivante ajoute d à l'ième int stocké dans le fichier (en comptant à partir de 0).

```
int ajouter( const char *nom, int d, int i){
    FILE * f = fopen( nom, "r+" );
    if( f == NULL ) return -1;

    /* se positionner juste avant i-ème int */
    fseek( f, i * sizeof( int ), SEEK_SET );
    int k;

    /* lire un int et le mettre dans k */
    size_t v = fread( &k, sizeof( int ), 1, f );
    if( v == 0 ) return -1;
    k += d;

    /* reculer un int */
    fseek( f, -sizeof(int), 1, SEEK_CUR );
    /* écrire la nouvelle valeur */
    fwrite( &k, sizeof( int ), f );
    fclose(f);
}
```