Langage C
fichiers

Wieslaw Zielonka
zielonka@irif.fr

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

r	("r" et "r+")	le fichier doit exister
w	("w" et "w+")	le fichier écrasé à l'ouverture
+	("a+" "w+" "r+")	ouverture en lecture et écriture
a	("a" "a+")	mode append, pour effectuer une écriture la position courante passe à la fin du fichier

Si on mélange les lectures et écritures, il faut les séparer par `fflush() (ou rewind()ou fseek())`

fermeture d'un flot

```
int fclose(FILE *flot)
```

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

lecture caractère par caractère

```
int fgetc(FILE *flot)
```

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

`int getc(FILE *flot)` 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 la variable qui reçoit le résultat de ces fonctions.

réinjecter un caractère dans le flot

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

le caractère `c` est remis dans le flot, la lecture suivante lit ce caractère.

Le caractère `c` peut être différent du dernier caractère lu.

S'il y a une suite d'appels à `ungetc()` alors à un moment ces appels peuvent échoués et la fonction retournera `EOF`.

`ungetc()` ne remet pas physiquement le caractère dans le fichier mais le remet dans le tampon de lecture.

écriture caractère par caractère

```
int fputc(int c, FILE *flot)
```

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

`int putc(int c, FILE *flot)` 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 *flot)
```

lit au plus `n-1` caractères et les places 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

```
int fputs(const char *s,  FILE *flot)
```

écrit les caractères de la chaîne s dans le flot (sans caractère `'\0'`). Retourne un nombre non-négatif si OK et `EOF` en cas d'erreur.

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

puts() écrit `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).

vider les le tampon d'un flot d'écriture

Pour les flots ouvert en écriture

- la fermeture du flot par fclose() et
- la terminaison de processus par return dans main() ou par exit()

provoquent l'écriture du contenu du tampon vers le fichier.

Pour forcer l'écriture de tampon explicitement utilisez

```
int fflush(FILE *flot)
```

`flush(NULL)` force l'écriture des tampons de tous les flots ouverts en écriture

le tampon d'un flot

Chaque flot a un des trois modes de fonctionnement :

- non mémorisé (*unbuffered*) -- les octets sont transmis le plus tôt possible après chaque read/write. Le flot n'utilise pas de tampon (ce qui n'exclut pas que le système d'exploitation utilise des tampons)

- pleinement mémorisé (*buffered*) -- les octets sont transmis par le bloc de la taille du tampon.

- mémorisé par ligne (*line buffered*) -- les octets stockés dans les tampon sont transmis vers le disque quand les caractère `'\n'` est envoyé dans le flot (ou quand le tampon est plein).

les modes de fonctionnement des flots

Le flot `stderr` est unbuffered.

Tous les autres flots sont

- " line buffered " pour les flots qui correspondent à un terminal et
- " fully buffered " pour le flots qui correspondent à un fichier.

Pour les flots ouvert en écriture le tampon de FILE est écrit sur le disque quand :

- fclose(flot) ferme le flot,
- le programme termine.

contrôle de tampon d'un flot

Un flot qui n'est pas associé à une entrée/sortie interactive (terminal/clavier) s'ouvre par défaut en mode pleinement mémorisé (*fully buffered*).

Pour changer le mode :

```
int setvbuf(FILE *flot, char *buf, int mode, size_t t)
```

`buf` – si non NULL le tampon à utiliser, `t` – la taille du tampon

mode :

- `_IOFBF` fully buffered
- `_IOLBF` line buffered
- `_IONBF` unbuffered

`void setbuf(FILE *flot, char *buf)` forme simplifiée :

`setbuf(flot, NULL)` met le flot en mode unbuffered, par exemple

`setbuf(stdout, NULL)` met le flot de sortie standard en mode non mémorisé

tout ce qui est écrit dans stdout est envoyé immédiatement sur l'écran

gestion de la position courante dans un flot

contrôle de la position courante dans un fichier

Chaque lecture/écriture modifie la position courante.

```
long ftell(FILE *flot)
```

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

```
int rewind(FILE *flot)
```

ramène la position courante au début du fichier.

contrôle de la position courante dans un fichier

Pour changer la position courante :

```
int fseek(FILE *flot, 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

contrôle de position courante dans un fichier

Exemples :

`fseek(flot, 0L, SEEK_SET)` aller au début du fichier, même chose que `rewind(flot)`

fseek(flot, -10L, SEEK_END) aller 10 octets avant la fin du fichier

`fseek(flot, -1024L, SEEK_CUR)` revenir 1024 octets en arrière par rapport à la position courante

On ne peut pas aller à une position avant le début de fichier, mais on peut aller au delà de la fin de 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 ?

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

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 *flot, fpos_t *position)
```

mémorise dans position la position courante du flot

```
int fsetpos(FILE *flot,
            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

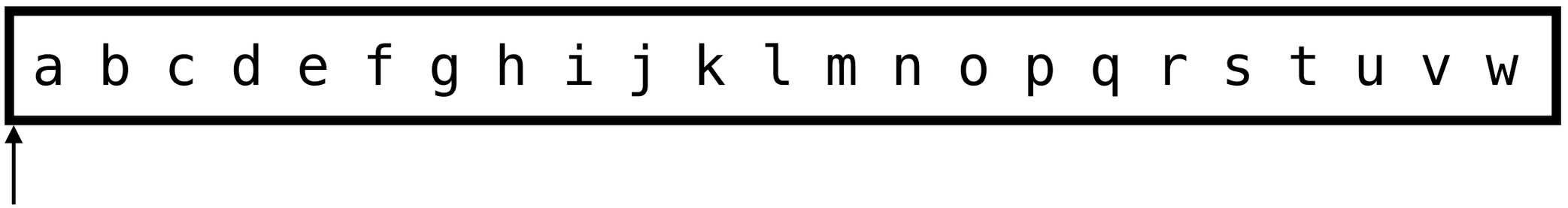
- si une écriture est suivi d'une lecture il faut insérer un appel à une des fonctions : `fflush(), fseek(), rewind()` entre l'écriture et la lecture,
- si une lecture est suivi d'une écriture il faut insérer un appel à une des fonctions : `fseek() ou rewind()` entre la lecture et l'écriture.

Si ces consignes ne sont pas respectées le comportement de votre application peut être imprévisible.

contrôle de la position courante - exemple

0uvrir 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).

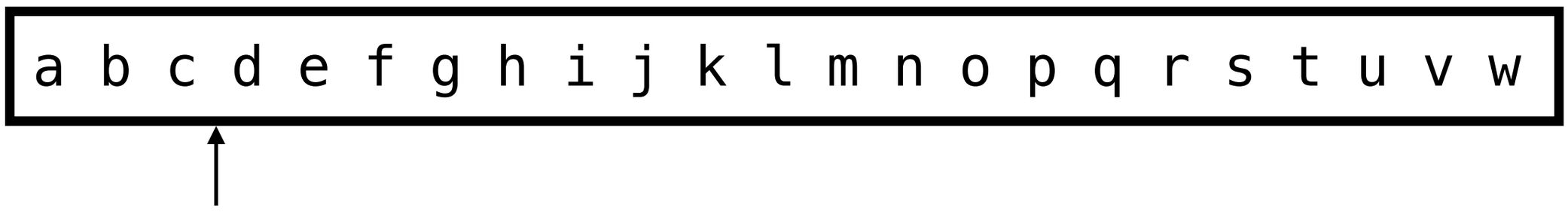
```
FILE *flot = fopen( argv[1], "r+" );
```



Pourquoi l'ouverture "r+" ?

Lire trois premiers caractères dans un tampon t.

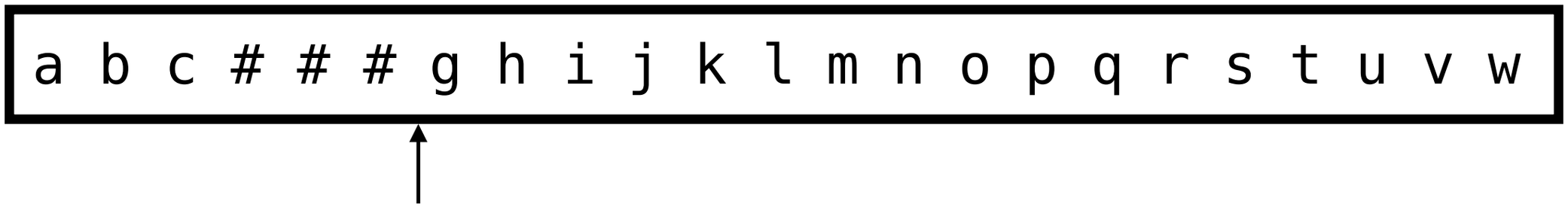
```
char t[10];
fgets( t, 4, flot);  /* fgets(. , 4, .) lit au plus 3 char
                      * et met '\0'  à la fin */
```



contrôle de la position courante - exemple

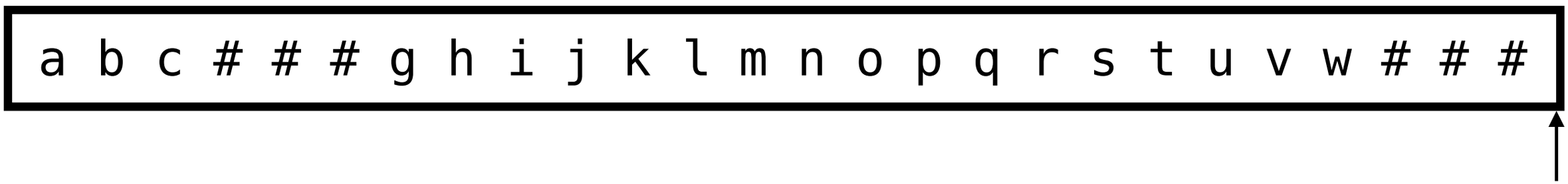
Remplacer 3 char suivants par "###" .

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



Ajouter "###" à la fin du flot:

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



lecture/écriture dans un fichier binaire

Le fichier binaire : on stocke les données de n'importe quel type **sans le faire transformation 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 fread(void *buf, size_t size,
             size_t nitems, FILE *flot)
```

`fread()` lit `nitems` éléments de taille `size` et les place à l'adresse `buf`. La fonction retourne le nombre d'éléments lus, ou 0 en cas d'erreur ou à la fin du fichier.

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

écrit nitems éléments d'un tableau à l'adresse buf, chaque élément est de taille size. Retourne le nombre d'éléments écrits.

exemple : fichier binaire - écrire dans un fichier

```
double tab[100];

// remplir le tableau



//ecrire le tableau dans un fichier "nombres"

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

```
size_t n = fwrite( tab, sizeof( tab[0] ), 100, flot);

fclose(flot);
```

Tout le tableau est écrit dans le fichier.

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

exemple : fichier binaire - lire depuis un fichier

```
/* lire i-eme double depuis le fichier qui contient un tableau de
doubles */

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

/* se déplacer juste après i-1 premiers doubles */

fseek(flot, sizeof(double) * (i-1) , SEEK_SET);

double a;

/* lire une valeur double et la mettre dans  a */

size_t n = fread( &a, sizeof( double ), 1, flot);

if( n == 0 ){ /*lecture a échouée */   }

fclose(flot);
```

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 héxa 00 00 00 01

est stocké en mémoire

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

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

exemple : copier un fichier

```
FILE *source = fopen(nom_ficher_source,"r");
FILE *dest = fopen(nom_ficheir_dest,"w");
```

```
/* copier caractère par caractère */
int c;
while(  ( c = fgetc( source ) ) != EOF ){

    if( fputc(c, dest)  ==  EOF ){
       /* traiter erreur de fputc() */
    }

}
```

```
/* 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  */
   }
}
```

copier un fichier

taille de fichier 2351369 :

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

real0m0.024s
user0m0.002s
sys 0m0.012s
```

tampon de 1024 octets

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

real0m0.020s
user0m0.002s
sys 0m0.007s
```

tampon de 4096 octets

Détecter erreur pour une opération lecture/écriture

Détecter fin de flot en lecture

l'indicateur de fin de flot et indicateur d'erreur

Souvent les fonctions d'entrée/sortie retournent la même valeur

- à la fin du flot (par exemple pour signaler la fin de fichier en lecture) et
- en cas d'erreur de lecture/écriture.

Pour distinguer fin de flot et erreur le flot possède deux indicateurs : indicateur de fin de fichier et indicateur d'erreur.

```
int feof(FILE *flot)
```

```
int ferror(FILE *flot)
```

retournent une valeur différente de 0 si l'indicateur correspondant est positionné.

Une fois activé l'indicateur reste dans l'état activé.

Pour mettre les deux indicateurs à 0 on utilise :

```
void clearerr(FILE *flot)
```

l'indicateur de fin de flot et indicateur d'erreur

```
FILE *flot = fopen(nom_fichier, "r");
#define T 124
char in[ T ];
while( fgets(in, T, flot) != NULL ){
/* faire le traitement de caractères lus */
}
// vérifier si fin de flot ou  erreur
if( ferror( flot ) ){
   /* traiter l'erreur de la lecture */
}
if( feof( flot ) ){
   // traiter fin du fichier
}
clearerr( flot );
```