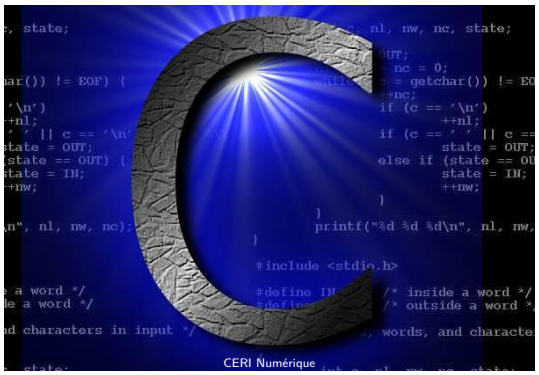
# Algorithmique et langage C

— I/O : file.h, ... —




Luc Fabresse
luc.fabresse@imt-nord-europe.fr

version 1.2

## Plan





## Manipulation de fichiers

### <stdio.h>

- De nombreuses fonctions pour la manipulation de fichiers
- Utilisent un "buffer" (bloc en mémoire) géré automatiquement
- Aucune écriture physique sur le disque mais dans le buffer qui est écrit d'un coup quand il est plein ou qu'on le purge

### Fonctions de <stdio.h>

- le type `FILE`
- `FILE *fopen(char *name, char *mode)`. Différents modes :
  - "r" ouvre le fichier en lecture
  - "w" ouvre le fichier en écriture. S'il n'existe pas, il est créé. S'il existe, son contenu est effacé.
  - "a" ouvre le fichier en écriture pour ajout (append). Conserve le contenu du fichier s'il existe.
- `int fclose(FILE*)` retourne 0 si ok et `EOF` sinon

## Principe de la manipulation de fichiers

### Principe général

1. Ouvrir le fichier
2. lire / écrire dans le fichier
3. fermer le fichier

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

int main(void){
   char fileName[] = "donnees.txt";
   FILE* f;

   f = fopen(fileName,"r");

   // traiter le contenu du fichier ici

   fclose(f);

   return EXIT_SUCCESS;
}
```



## Principe de la manipulation de fichiers

### Principe général

1. Ouvrir le fichier
2. lire / écrire dans le fichier
3. fermer le fichier

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

int main(void){
   char fileName[] = "donnees.txt";
   FILE* f;

   f = fopen(fileName,"r");

   // traiter le contenu du fichier ici

   fclose(f);

   return EXIT_SUCCESS;
}
```

### Attentions aux erreurs !

Toujours tester les codes d'erreurs



## Principe de la manipulation de fichiers avec gestion des erreurs

```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

int main(void){
   char fileName[] = "donnees1.txt";
   FILE* f;

   f = fopen(fileName,"r");

   if(f==NULL) {
      printf ("Code de l'erreur : %d\n", errno);

      if (errno == ENOENT)
         printf ("Le fichier n'existe pas !\n");
      else
         printf ("Erreur inconnue\n");

      return EXIT_FAILURE;
   }

   // traiter le contenu du fichier ici

   fclose(f);

   return EXIT_SUCCESS;
}
```



## Les fonctions de lecture

### Lecture formatées

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

### Lecture par caractères

`char fgetc(FILE *id)` ou `int getc(FILE *fp)` retourne le prochain caractère du fichier (`EOF` pour end of file)

### Lecture par chaînes

`char *fgets(char *line, int maxline, FILE *fp)` : lit la prochaine ligne dans `fp` (dont le caractère `\n`) et place le contenu dans `line`. Au plus, `maxline-1` caractères seront lus. La chaîne résultante sera automatique terminée par `\0`. Normalement, `fgets` retourne `line` et NULL en cas d'erreur ou fin de fichier.

### Lecture par blocs

`int fread(void *bloc, int taille, int nb, FILE *id)` : lit nb éléments dont on donne la taille unitaire en octets, dans le fichier désigné par id, le résultat étant stocké à l'adresse bloc. La fonction rend le nombre d'éléments lus (<nb si fin de fichier), 0 si erreur

### Détecter la fin de fichier

`int feof(FILE *id)` indique si on est en fin de fichier ou non (0).



## Les fonctions d'écriture

### Ecritures formatées

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

### Par blocs, caractères, ...

- `int fwrite(void *bloc, int taille, int nb, FILE *id)` : écriture du bloc sur fichier, si le nombre rendu est différent de nb, il y a eu erreur (tester ferror ou errno).
- `int putc(int c, FILE *fp)` écrit c dans le fichier et retourne c ou `EOF` en cas d'erreur

### Gestion du buffer système

`int fflush(FILE *id)` : transfère effectivement le reste du buffer sur disque, sans fermer le fichier (à appeler par exemple avant une instruction qui risque de créer un "plantage").

**`int fseek(FILE *id, long combien, int mode)`**

déplace le pointeur de fichier de combien octets, à partir de : début du fichier (mode=0), position actuelle (mode=1) ou fin du fichier (mode=2). Retourne 0 si tout c'est bien passé. Cette fonction n'est utilisable que si l'on connaît la taille des données dans le fichier (impossible d'aller directement à une ligne donnée d'un texte si on ne connaît pas la longueur de chaque ligne).





```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

int main(void){
   char fileName[] = "data.txt";
   FILE* f;

   if( (f=fopen(fileName,"r")) == NULL) {
      printf ("Code de l'erreur : %d\n", errno);
      return EXIT_FAILURE; }

   // traiter le contenu du fichier ici
   char buffer[10];

   while( fgets(buffer,10,f) != NULL ) {
      printf(" %s",buffer);
   }

   fclose(f);
   return EXIT_SUCCESS;
}
```

```
$ cat data.txt
10
-7
piklpilu
azdjhaoidhiopahdiohpodazljdbaz
qvcwx$
```

```
$ ./prog data.txt
 10
 -7
 piklpilu
 azdjhaoid hiopahdio hpodazljd baz
 qvcwx$
```