

C : La bibliothèque `stdio`

Christophe Prieur

1 Les fichiers

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

Ouvre le fichier *nom* et renvoie un pointeur sur l'objet (`FILE`) permettant de le manipuler, ou `NULL` en cas d'erreur.

Valeurs possibles pour *mode* :

- `"r"` lecture ;
- `"w"` écriture, fichier écrasé s'il existe, créé sinon ;
- `"a"` *append*, écriture après la fin du fichier s'il existe (créé sinon) ;
- `"r+"` lecture/écriture (le fichier doit exister) ;
- `"w+"` lecture/écriture (fichier créé ou écrasé) ;
- `"a+"` lecture/écriture (l'écriture est faite en fin de fichier).

`int fclose(FILE *f) ;`

Ferme le fichier désigné par *f*.
Renvoie 0 si tout va bien, `EOF` sinon.

`FILE *stdin, *stdout, *stderr ;`

pointeurs macro-définis désignant l'entrée et les sorties standard.

`FILE *freopen(const char *nom, const char *mode, FILE *f) ;`

Ouvre (comme `fopen()`) le fichier *nom* et lui associe le pointeur *f*. Autrement dit, redirige *f* vers le fichier *nom*.

Par exemple, commencer le programme `prog` par l'instruction suivante :

```
freopen("toto", "w", stdout) ;
```

est équivalent à lancer le programme sous Unix par :

```
prog>toto
```

`void rewind(FILE *f)`

Remplace le curseur au début du fichier, vide le tampon d'écriture et réinitialise les indicateurs d'erreurs concernant le fichier.

2 Entrées/sorties directes

`int fgetc(FILE *f) ; int getc(FILE *f) ;`

Renvoie le caractère suivant dans le fichier (`unsigned char` converti en `int`) et le supprime du tampon de lecture.

Renvoie `EOF` si on est à la fin du fichier ou en cas d'erreur.

`fgetc()` est une fonction et en général `getc()` est une macro (plus rapide mais pas utilisable comme pointeur de fonction).

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

Écrit le caractère (`unsigned char`) *c* et retourne le caractère écrit ou `EOF` si erreur.

Même distinction qu'entre `fgetc()` et `getc()`.

`int getchar()` ;
Équivalent à `getc(stdin)`.

`int putchar(int c)` ;
Équivalent à `putc(c, stdout)`.

`int ungetc(int c, FILE *f)` ;
Remplace le caractère (`unsigned char`) `c` dans le tampon de lecture du fichier désigné par `f`.

`char *fgets(char *s, int n, FILE *f)` ;
Lit au plus `n - 1` caractères dans `f` et les place à partir de l'adresse `s`, en ajoutant à la fin le caractère `'\0'`.
La lecture s'arrête avant `n - 1` si la fin de fichier est atteinte ou si le caractère `'\n'` est lu. Dans ce dernier cas, le caractère est copié dans la chaîne.
Renvoie `s` ou `NULL` en cas d'erreur.

`int fputs(const char *s, FILE *f)` ;
Écrit le contenu de la chaîne `s` (sans le caractère `'\0'` final).
Renvoie `EOF` en cas d'erreur.

`int fread(void *ptr, size_t taille, size_t n, FILE *f)` ;
Lit au plus `n` objets de taille `taille` et les place à partir de l'adresse `ptr`.
Renvoie le nombre d'objets lus.

`int fwrite(void *ptr, size_t taille, size_t n, FILE *f)` ;
Écrit `n` objets de taille `taille` contenus à l'adresse `ptr`.
Renvoie le nombre d'objets écrits.

3 Entrées/sorties formatées

`int fscanf(FILE *f, const char *format, ...)` ;
La chaîne `format` est une suite de caractères ordinaires, et de spécifications de format de conversion de la forme :

`%[*][longueur][qualificatif]format`
(les `[]` indiquent un élément optionnel).

- Le caractère `*` indique que la valeur ne sera pas affectée ;
- l'entier `longueur` indique le nombre maximal de caractères correspondant ;
- les qualificatifs `h` `l` ou `L` spécifient la taille de l'objet correspondant (`short`, `long`, `long double`) ;
- les principaux formats de conversion sont les suivants :
 - `c` caractère
 - `s` suite de caractères (sans espaces)
 - `p` pointeur (hexadécimal non signé)
 - `d` entier sous forme décimale
 - `u` entier décimal non signé
 - `o` entier octal
 - `x` entier hexadécimal
 - `f` flottant
 - `n` nombre de caractères déjà lus (pas de conversion).

Les paramètres qui suivent `format` sont les adresses où seront placées les valeurs lues pour chaque format annoncé.
La fonction renvoie le nombre de conversions réalisées, ou `EOF` en fin de fichier ou en cas d'erreur dans une conversion.

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

Équivalent à `fscanf(stdin, format, ...)`.

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

Les spécifications de format sont de la forme :

```
%[-]/+[n1][n2][qualificatif]format
```

(les `[]` indiquent un élément optionnel).

- - spécifie une justification à gauche (par défaut, elle est à droite);
- + spécifie l’affichage systématique du signe pour les valeurs signées (un espace à la place du + demande qu’une valeur positive soit précédée d’un espace);
- n_1 indique le nombre de caractères de la chaîne que produira la conversion;
- n_2 indique le nombre de décimales;
- *qualificatif* `h` ou `L` comme pour `fscanf()`;
- principaux formats de conversion :
 - `d` entier en décimal
 - `u` entier non signé en décimal
 - `o` entier en octal
 - `x` entier en hexadécimal
 - `f` double en virgule fixe
 - `e` double en virgule flottante
 - `c` `unsigned char`
 - `s` chaîne de caractères
 - `p` pointeur (adresse affichée en hexadécimal)

La fonction renvoie le nombre de caractères écrits ou une valeur négative en cas d’erreur.

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

Équivalent à `fprintf(stdout, format, ...)`.

4 Positionnement dans un fichier

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

Positionne le pointeur de lecture/écriture du fichier à la position *offset* par rapport à la position *origine*, qui peut être `SEEK_SET` (début du fichier), `SEEK_CUR` (position courante) ou `SEEK_END` (fin).

5 Manipulation du tampon

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

Vide dans le fichier le contenu du tampon d’écriture.
Retourne 0, ou EOF si erreur.