Manipuler des fichiers en C

Toutes ces fonctions sont déclarées dans `<stdio.h>`. Les prototypes ci-dessous sont présentés sans les qualificatifs `restrict`.

Ouverture et fermeture

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

Ouvre un fichier et renvoie un pointeur sur le flot, ou `NULL` en cas d'échec.

Mode principaux	Utilisation
`"r"`	Lecture : le fichier doit exister.
`"w"`	Écriture : crée le fichier ou efface son contenu existant.
`"a"`	Ajout en fin de fichier : crée le fichier s'il n'existe pas.

```
int fclose(FILE *stream);
```

Ferme le flot et transmet les données d'écriture encore en attente. Renvoie `0` en cas de succès, `EOF` en cas d'erreur.

Écriture

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

Écrit des données selon un format, comme `printf`. Renvoie le nombre de caractères écrits, ou une valeur négative en cas d'erreur.

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

Écrit une chaîne sans son `'\0'` et sans ajouter de saut de ligne. Renvoie une valeur non négative en cas de succès, `EOF` en cas d'erreur.

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

Écrit un caractère. Renvoie le caractère écrit (converti en `unsigned char` puis en `int`), ou `EOF` en cas d'erreur.

Précaution : une chaîne provenant d'une source extérieure ne doit pas être utilisée directement comme format de `printf` ou de `fprintf`. Pour afficher son contenu, utiliser `fputs(chaine, stdout)` ou `printf("%s", chaine)`, plutôt que `printf(chaine)`.

Lecture

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

Lit des données selon un format : utile pour les fichiers texte structurés (entiers, réels, etc.). Renvoie le nombre de valeurs affectées ; `EOF` si une fin de fichier ou une erreur de lecture empêche la première conversion.

```
int fgetc(FILE *stream);
```

Lit le caractère suivant, sans imposer de format. Renvoie sa valeur comme `unsigned char` converti en `int`, ou `EOF` à la fin du fichier ou en cas d'erreur. **Stocker le résultat dans un `int`.**

Principaux spécificateurs de format

La chaîne de format indique comment écrire ou lire chaque donnée. Les spécificateurs suivants s'utilisent avec `fprintf` et `fscanf` :

Format	Écriture (`fprintf`)	Lecture (`fscanf`)
`%s`	Chaîne de caractères	Tableau de caractères
`%c`	Caractère	Adresse d'un `char`
`%d`	Entier de type `int`	Adresse d'un `int`
`%f`	Réel (`float` ou `double`)	Adresse d'un `float`
`%lf`	Identique à `%f`	Adresse d'un `double`

Attention : pour `fscanf`, on fournit les adresses des variables à modifier. La distinction entre `%f` et `%lf` est alors indispensable. Pour `fprintf`, un `float` est automatiquement converti en `double` ; on utilise généralement `%f`.

Avec `fscanf`, `%s` lit un mot et ajoute le caractère final `'\0'` : limiter la lecture pour éviter un débordement, par exemple `"%99s"` pour un tableau de 100 caractères.

Avec `fscanf`, le spécificateur `%c` lit un seul caractère, même si celui-ci est un espace, une tabulation ou un saut de ligne. Pour ignorer les caractères blancs avant cette lecture, on place un espace devant `%c` dans la chaîne de format : `" %c"`. Dans les deux cas, seul le caractère lu est stocké : aucun caractère `'\0'` n'est ajouté.

```
int n;
double x;
char mot[100];
if (fscanf(f, "%d %lf %99s", &n, &x, mot) == 3)
    fprintf(stdout, "%d %f %s\n", n, x, mot);
```