

06B : Les fichiers et les flux en C

420-3A6-ST : Programmation système

Dany Gagnon

Cégep de Saint-Jean-sur-Richelieu

Mercredi 7 octobre 2026

Un fichier et un flux sont deux objets différents

- Un fichier conserve des données ; un chemin permet de le désigner.
- Un flux est l'interface de la bibliothèque C pour lire ou écrire une séquence de caractères.
- Un `FILE *` désigne un flux ouvert. Il donne accès à une position courante, à un tampon et à des indicateurs de fin et d'erreur.
- Les opérations de lecture et d'écriture font avancer la position du flux lorsque celui-ci est positionnable.

Le programme utilise les fonctions de `<stdio.h>` ; il ne manipule pas directement les champs de `FILE`.

Arguments et flux standards

- `argc` indique le nombre d'arguments ; `argv` contient les chaînes transmises au démarrage. Un chemin dans `argv[1]` est une chaîne, pas un fichier déjà ouvert.
- `stdin`, `stdout` et `stderr` sont des flux disponibles dès le début du programme.
- L'entrée standard fournit des données ; la sortie standard transporte le résultat ; la sortie d'erreur transporte les diagnostics.
- Une redirection du shell change la destination ou la source d'un flux standard. Elle n'ajoute pas d'argument.

Le statut renvoyé par `main` est distinct des données écrites : zéro indique la réussite, une valeur non nulle indique un échec selon la convention de la commande.

Ouvrir un flux : accès et effet sur le fichier

`fopen(chemin, mode)` retourne un `FILE *` en cas de réussite et `NULL` en cas d'échec.

Mode	Accès	Fichier absent	Fichier présent
"r"	lecture	échec	position au début
"w"	écriture	création	contenu tronqué
"a"	écriture	création	écritures à la fin

- Le mode exprime l'accès demandé ; les permissions peuvent empêcher l'ouverture.
- La troncature par "w" a lieu dès l'ouverture réussie, avant la première écriture.
- Le suffixe `b` demande le mode binaire : "rb", "wb". Il évite les transformations du mode texte sur les systèmes qui les distinguent.

Lire : une valeur de retour et un état du flux

- `fgetc(flux)` retourne le prochain caractère comme un `unsigned char` converti en `int`, ou `EOF`.
- Le résultat se conserve dans un `int` : `EOF` doit rester distinct de toutes les valeurs possibles d'un octet.
- `EOF` signale une fin de fichier ou une erreur. `feof(flux)` et `ferror(flux)` permettent de distinguer ces états.
- Ces indicateurs décrivent une opération déjà effectuée. `feof` ne prédit pas si la prochaine lecture réussira.

La lecture est pilotée par le résultat de la fonction ; l'état du flux explique ensuite pourquoi elle s'est arrêtée.

Lire du texte : caractères, morceaux et lignes

- `fgets(tampon, capacite, flux)` lit au plus `capacite - 1` caractères, puis ajoute le caractère nul `'\0'` lors d'une lecture réussie.
- La lecture s'arrête après un saut de ligne lu, lorsque le tampon est rempli, ou à la fin du fichier.
- Le saut de ligne est conservé s'il a été lu. Une ligne longue peut nécessiter plusieurs appels.
- La fonction retourne le tampon en cas de réussite et `NULL` si aucun caractère n'a été lu à la fin du fichier, ou en cas d'erreur.

Un morceau lu n'est donc pas nécessairement une ligne complète. `fputs` écrit une chaîne jusqu'au caractère nul, sans ajouter de saut de ligne.

Écrire : le tampon retarde certains effets

- `fputc` écrit un caractère ; `fputs` écrit une chaîne. Ces fonctions retournent `EOF` en cas d'échec.
- Le tampon est une zone de mémoire où la bibliothèque regroupe les données avant leur transmission au système. Une écriture réussie peut seulement avoir rempli ce tampon.
- `fflush(flux)` transmet les données de sortie en attente et retourne zéro en cas de réussite, `EOF` en cas d'échec.
- `fclose(flux)` vide les données de sortie en attente, ferme le flux et peut lui aussi signaler un échec par `EOF`.

Chaque étape a son propre résultat. Un disque plein peut être détecté seulement lors du vidage du tampon, après des écritures apparemment réussies.

Fermer les ressources et conserver le résultat

- Une ouverture réussie crée une ressource à fermer, même si une opération ultérieure échoue.
- Après `fclose`, le flux ne doit plus être utilisé, même si la fermeture a signalé une erreur.
- Le résultat global dépend de toutes les opérations nécessaires : une réussite antérieure ne compense pas un échec ultérieur.
- Sur POSIX, les fonctions d'entrée-sortie renseignent `errno` lors d'un échec. `perror` affiche le contexte fourni et le message correspondant à sa valeur courante.

Opération	Données en attente	Durée de vie du flux
<code>fflush</code>	transmission tentée	flux encore ouvert
<code>fclose</code>	transmission tentée	flux terminé

Le résultat d'une opération et la validité du flux sont deux informations distinctes.

Écrire un fichier : programme complet

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "Usage : %s FICHER\n", argv[0]);
        return 2;
    }
    FILE *sortie = fopen(argv[1], "w");
    if (sortie == NULL) {
        perror(argv[1]);
        return 1;
    }
    int resultat = 0;
    if (fputs("Mesure recue\n", sortie) == EOF) {
        perror("écriture");
        resultat = 1;
    }
    if (fclose(sortie) == EOF) {
        perror("fermeture");
        resultat = 1;
    }
    return resultat;
}
```

`ecrire.c` reçoit un chemin et remplace le contenu du fichier par une ligne.

L'ouverture en mode "w" crée ou tronque le fichier. Une erreur d'ouverture arrête le programme avant toute utilisation du flux.

Le statut `resultat` conserve un échec d'écriture ou de fermeture. Le flux est fermé même si `fputs` échoue.

`./ecrire mesures.txt`

Lire et afficher un fichier : programme complet

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "Usage : %s FICHIER\n",
            ↪ argv[0]);
        return 2;
    }
    FILE *entree = fopen(argv[1], "r");
    if (entree == NULL) {
        perror(argv[1]);
        return 1;
    }
    int resultat = 0;
    int octet;
    while ((octet = fgetc(entree)) != EOF) {
        if (fputc(octet, stdout) == EOF) {
            perror("sortie");
            resultat = 1;
            break;
        }
    }
}
```

```
    if (ferror(entree)) {
        perror("lecture");
        resultat = 1;
    }
    if (fclose(entree) == EOF) {
        perror("fermeture");
        resultat = 1;
    }
    if (fflush(stdout) == EOF) {
        perror("sortie");
        resultat = 1;
    }
    return resultat;
}
```

Le code se poursuit de la colonne gauche à la colonne droite.

```
./lire mesures.txt > copie.txt
```

`ferror` vérifie la lecture ; `fflush` détecte une erreur de sortie retardée. Le fichier d'entrée est fermé dans les deux cas.