Gestion des fichiers

Fondements du langage C

L'utilisation des fichiers permet de lire, de traîter et de sauvegarder sur la mémoire de masse (disque dur, carte mémoire, etc.) des données qui doivent être conservées même après l'arrêt de la machine, ce qui n'est pas possible par l'utilisation des variables. Les opérations de gestion de fichiers concernent les traitements courants de lecture et d'écriture mais également de création, de renommage et de suppression. La plupart des fonctions liées à la gestion des fichiers sont définies dans le fichier d'entête `stdio.h` de la bibliothèque standard.

Fonctions `fopen()` et `fclose()` pour l'ouverture et la fermeture et différents modes d'accès

L'ouverture d'un fichier s'effectue au moyen de la fonction `fopen()`. Son prototype est le suivant :

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

Cette fonction renvoie un pointeur sur un type spécifique complexe nommé `FILE`. Il s'agit en fait d'une structure de données définie dans le fichier d'entête `stdio.h`.

Le paramètre `fichier` est une chaîne de caractères correspondant au nom du fichier à ouvrir et éventuellement au chemin d'accès complet jusqu'au fichier à ouvrir s'il ne se situe pas dans le répertoire courant. Le paramètre `mode` permet de préciser le mode d'ouverture spécifiant le type d'accès au contenu du fichier. Les principales valeurs sont listées dans le tableau 1. Elles s'appliquent à un accès en mode texte. C'est-à-dire que le contenu des fichiers est traîté sous forme de caractères et non pas comme une succession d'éléments binaires (0 ou 1).

Spécificateurs	Signification
"r"	Lecture seule
"w"	Écriture seule
"a"	Ajout à la fin du fichier
"r+"	Lecture et écriture
"w+"	Lecture et écriture avec suppression préalable du contenu
"a+"	Lecture et ajout à la fin du fichier

Tab. 1 – *Liste des spécificateurs de mode d'accès*

Il existe aussi les spécificateurs "rb", "wb", "ab", "rb+", "wb+" et "ab+". Ils sont destinés à accéder aux fichiers non plus en mode texte mais en mode binaire.

En cas d'échec d'ouverture du fichier, la fonction `fopen()` renvoie la valeur `NULL`. Il convient alors de tester la valeur renvoyée afin de s'assurer que l'ouverture du fichier s'est correctement effectuée.

La fermeture d'un fichier préalablement ouvert par la fonction `fopen()` s'effectue en utilisant la fonction `fclose()` qui ne prend en paramètre que le nom du pointeur renvoyé lors de l'ouverture du fichier. Le prototype est indiqué ci-dessous :

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

La fonction `fclose()` renvoie la valeur entière `0` lorsque la fermeture du fichier s'est correctement déroulée. Elle libère alors de la mémoire vive la variable de type `FILE` associée au préalablement fichier ouvert par la fonction `fopen()`. En cas d'échec, la fonction `fclose()` renvoie la valeur `EOF`.

Le code source présenté ci-dessous montre un exemple simplifié d'exploitation des fonctions `fopen()` et `fclose()` qui consiste à ouvrir un fichier en mode texte avec un accès en lecture et écriture puis de le refermer :

```
FILE *fic;

fic = fopen("fichier.txt", "r+");

if (fic != NULL)
{
  printf("Le fichier est ouvert !\n");
  fclose(fic);
}
else
  printf("L'ouverture du fichier a
          échoué !\n");
```

La valeur renvoyée par la fonction `fclose()` n'est ici pas évaluée. Il est en effet rare qu'une désallocation mémoire ne s'effectue pas correctement. Aussi et excepté dans le cas où le niveau de fiabilité et de robustesse du programme est critique, il n'est pas toujours nécessaire de considérer le cas d'erreur lors d'une fermeture de fichier.

Fonctions `fputc()`, `fputs()` et `fprintf()` pour l'écriture

Lorsqu'il s'agit d'insérer un simple caractère dans un fichier préalablement ouvert en écriture, il convient d'utiliser la fonction `fputc()` dont le prototype est indiqué ce-dessous :

```
int fputc(int caractere,
          FILE *fichier);
```

Cette fonction ne prend en compte que deux paramètres que sont le caractère à insérer et le fichier concerné par l'écriture. Un caractère est codé sur un octet, c'est-à-dire un entier sur huit bits. L'utilisation du type `int` permet ainsi de passer le caractère en paramètre.

La fonction `fputc()` retourne une valeur de type `int`. Celle-ci vaut `EOF` si l'écriture a échoué et une autre valeur entière correspondant au code du caractère en cas de réussite.

Le prototype et l'utilisation de la fonction `fputs()` sont très similaires à ceux de la fonction `fputc()`. La différence provient du fait que la fonction `fputs()` permet d'écrire dans un fichier une chaîne de caractères et non pas un seul caractère. Le prototype est le suivant :

```
int fputs(const char *chaine,
          FILE *fichier);
```

Le terme `const` dans le prototype indique que la chaîne passée en paramètre sera considérée comme une constante car il s'agit uniquement de lire la variable associée et en aucun cas de la modifier.

La fonction `fputs()` renvoie une valeur entière supérieure à zéro en cas d'exécution correcte et `EOF` en cas d'échec.

Le code source présenté ci-dessous montre un exemple simplifié d'exploitation des fonctions `fputc()` et `fputs()` :

```
FILE *fic;

fic = fopen("fichier.txt", "w");

if (fic != NULL)
{
  fputc('B', fic);
  fputs("onjou", fic);
  fputc('r', fic);
  fclose(fic);
}
```

Lorsqu'il est nécessaire d'écrire une chaîne formatée dans un fichier, il convient d'utiliser la fonction variadique `fprintf()` qui permet donc de prendre en compte des variables et d'en spécificier le format. Son prototype est indiqué ci-dessous :

```
int fprintf(FILE *fichier,
            const char *chaine, ...);
```

La fonction `fprintf()` retourne le nombre de caractères écrits si l'opération s'est bien déroulée ou une valeur négative en cas d'échec.

L'exemple de code ci-dessous présente une utilisation simplifiée de la fonction `fprintf()` pour l'écriture dans un fichier :

```
int nb = 1024;
FILE *fic;

fic = fopen("fichier.txt", "w");

if (fic != NULL)
{
  fprintf(fic, "Nombre : %d\n", nb);
  fclose(fic);
}
```

Fonctions `fgetc()`, `fgets()` et `fscanf()` pour la lecture

La fonction `fgetc()` permet de lire un caractère unique dans un fichier. Son prototype est le suivant :

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

En cas de réussite, la fonction `fgetc()` renvoie une valeur entière correspondant au caractère lu. Elle retourne la valeur `EOF` dans le cas contraire.

La fonction `fgetc()` lit le caractère courant du fichier sur lequel elle pointe. Au moment de la première exécution, elle lit donc le premier caractère et successivement les autres caractères lors des exécutions suivantes.

Le code simplifié ci-dessous montre un exemple d'utilisation de la fonction `fgetc()` :

```
FILE *fic;
int car;

fic = fopen("fichier.txt", "r");

if (fic != NULL)
{
  /* Lectures successives      */
  /* des caractères du fichier */
  do
  {
    /* Lecture et affichage */
    /* du caractère courant */
    car = fgetc(fic);
    printf("%c", car);
  } while (car != EOF);

  fclose(fic);
}
```

Pour réaliser une lecture ligne par ligne plutôt que caractère par caractère, il est possible d'utiliser la fonction `fgets()` dont le prototype est le suivant :

```
char *fgets(char *chaine,
            int nombre,
            FILE *fichier);
```

La fonction `fgets()` lit une ligne dans un fichier ouvert en lecture qu'elle stocke dans une chaîne pointée par le premier paramètre noté `chaine`. La lecture s'interrompt lorsque le nombre `nombre` de caractères à lire est atteint, lorsqu'un retour à la ligne (`'\n'`) est lu ou que la fin de fichier `EOF` est atteinte. Par ailleurs et afin de marquer la fin de chaîne, le caractère `NULL` (`'\0'`) est ajouté après le dernier caractère lu. La fonction `fgets()` lit donc au plus `nb - 1` caractères à partir du fichier.

La fonction `fgets()` retourne le même pointeur que celui pointé par l'argument `chaine` si la fonction s'est exécutée sans erreur. Elle retourne la valeur `NULL` si une erreur s'est produite ou lorsque la fin de fichier est atteinte. L'évaluation de la valeur retournée permet donc de savoir si l'exécution s'est correctement déroulée ou non mais aussi de détecter la fin de fichier.

Un exemple simplifié d'utilisation de la fonction `fgets()` pour la lecture d'un fichier texte est présenté ci-dessous :

```
FILE *fic;
char lig[51];

fic = fopen("fichier.txt", "r");

if (fic != NULL)
{
  /* Lectures successives  */
  /* des lignes du fichier */
  while(fgets(lig, 50, fic) != NULL)
    printf("%s", lig);

  fclose(fic);
}
```

La fonction `fscanf()` est de type variadique. Elle est utilisée pour lire dans un fichier contenant une organisation spécifique et connue du texte. Son prototype est précisé ci-dessous :

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

Le code ci-dessous présente un exemple simplifié d'utilisation de la fonction `fscanf()` :

```
FILE *fic;
char nom[20];
int age;

fic = fopen("fichier.txt", "r");

if (fic != NULL)
{
  fscanf(fic, "%s %d", nom, &age);
  printf("Ligne : %s, %d", nom, age);
  fclose(fic);
}
```

Dans cet exemple, il s'agit de lire une ligne composée de deux champs séparés par un espace et d'afficher le résultat de la lecture. Le premier champ est une chaîne de caractères et le second est un entier.

Fonctions `ftell()`, `fseek()` et `rewind()` pour le positionnement

À l'ouverture d'un fichier, un curseur est initialisé pour consigner la position dans le fichier qui est donnée en nombre d'octets. Il s'agit donc du nombre de caractères par rapport au début du fichier.

La fonction `ftell()` permet de connaître la position courante dans le fichier. Elle renvoie la position du curseur dans le fichier sous la forme d'une variable de type `long` ou la valeur `-1` en cas d'erreur. Son prototype est le suivant :

```
long ftell(FILE *fichier);
```

La fonction `fseek()` permet de déplacer le curseur d'un certain nombre de caractères à partir d'une position d'origine. Elle retourne la valeur `0` en cas de réussite et une autre valeur en cas d'échec. Son prototype est indiqué ci-dessous :

```
int fseek(FILE *fichier,
          long deplacement,
          int origine);
```

La valeur du deplacement peut être donnée par un nombre positif ou négatif selon que le déplacement doit être opéré en avant ou en arrière. Elle peut également être nulle s'il s'agit de déplacer le curseur à l'origine pasée en paramètre. L'origine est indiquée par l'une des trois constantes suivantes :

- SEEK_SET
- SEEK_CUR
- SEEK_END

Ces trois constantes correspondent respectivement au début du fichier, à la position courante du curseur et à la fin du fichier. À titre d'exemple, le code ci-dessous place le curseur à la fin du fichier :

```
FILE* fic;

fic = fopen("fichier.txt", "r");

if (fic != NULL)
{
  fseek(fic, 0, SEEK_END);
  fclose(fic);
}
```

La fonction `rewind()` force le renvoi du curseur au début du fichier. Elle ne retourne rien et son prototype est le suivant :

```
void rewind(FILE *fichier);
```

Fonction `rename()` pour le renommage

Le renommage d'un fichier s'effectue au moyen de la fonction `rename()` dont le prototype est indiqué ci-dessous :

```
int rename(const char *anciennom,
           const char *nouveaunom);
```

La fonction `rename()` renvoie la valeur `0` si le renommage du fichier s'est correctement déroulé ou une valeur différente en cas d'erreur.

Fonction `remove()` pour la suppression

La suppression d'un fichier est réalisée par la fonction `remove()`. Son prototype est le suivant :

```
int remove(const char *fichier);
```

La fonction `remove()` renvoie la valeur `0` si la suppression du fichier s'est correctement déroulée ou `-1` en cas d'erreur.

Il convient d'être prudent dans l'utilisation de la fonction `remove()`. Elle supprime le fichier dont le nom est passé en paramètre sans émettre de message d'avertissement ou de confirmation.