Lecture et Ecriture dans les fichiers en langage C

Ce cours présente les fonctions de la **bibliothèque standard** du langage C qui permettent de réaliser des lectures et/ou écritures dans les fichiers. L'utilisation de ces fonctions nécessite l'inclusion du fichier d'entête **<stdio.h>.**

1. Contenu du fichier <stdio.h>

On trouve notamment dans ce fichier d'entête :

- la macrodéfinition des constantes suivantes :
  - **NULL** : constante d'adresse nulle (adresse invalide, son déréférençage est interdit);
  - **EOF** : indicateur symbolique de fin de fichier (**E**nd **O**f **F**ile);
  - **BUFSIZ** : la taille par défaut des tampons d'entrées sorties;
- la définition des **types** :
  - **FILE** : **représente le type de la variable associée et dédiée à la manipulation d'un fichier**. Il ne faudra **jamais créer des variables de ce type** car ce sont les fonctions de la bibliothèque qui s'en chargent;
  - **size_t** : représente le type des longueurs de fichier (correspond au type *unsigned int*);
- la déclaration des variables :
  - **stdin** : de type FILE* et associée à l'entrée standard (par défaut le clavier);
  - **stdout** : de type FILE* et associée à la sortie standard (par défaut l'écran);
  - **stderr** : de type FILE* et associée à la sortie d'erreur standard (par défaut l'écran);

2. L'ouverture d'un fichier

La manipulation d'un fichier passe d'abord par la **demande d'ouverture** de ce fichier afin d'y réaliser des opérations (lectures et / ou écritures). Lorsqu'elle est obtenue, une telle ouverture garantit, sauf cas de force majeure, la réalisation des opérations demandées.

2.1 La fonction fopen()

Prototype : `FILE * fopen(const char *reference, const char *mode);`

En cas de succès, un appel à cette fonction permet d'**obtenir en valeur de retour l'adresse d'une variable de type FILE** (cette variable est créée par la fonction) **associée au fichier** de *référence* donnée en premier paramètre et autorisant les manipulations réclamées par le paramètre *mode*.
En cas d'échec, la fonction renvoie la constante d'adresse nulle (NULL). (C'est une convention toujours respectée par les fonctions renvoyant des adresses.)

A cette variable de type FILE, sont associés (entre autres) un **tampon** à travers lequel les lectures et/ou écritures sont effectuées, et une **position courante** qui repère dans le fichier la position de la prochaine opération d'entrée-sortie. Cette position courante n'est rien d'autre qu'un nombre précisant le rang du prochain octet à lire à ou écrire dans le fichier : on la désigne souvent en utilisant le terme de "**pointeur de fichier**" (même s'il de s'agit pas d'un véritable pointeur, ce n'est pas une adresse en mémoire).

Les modes d'ouverture autorisés sont :

`"r"` : *lecture* seulement ; le fichier doit exister.
`"w"` : *écriture* seulement. Si le fichier n'existe pas, il est créé. S'il existe, son (ancien) contenu est perdu (ou tronqué).
`"a"` : *écriture en fin de fichier seulement* (append). Si le fichier existe déjà, il sera étendu. S'il n'existe pas, il sera créé. Les écritures ont systématiquement lieu en fin de fichier.
`"r+"` : *mise à jour* (lecture et écriture). Le fichier doit exister. Il n'est pas tronqué.
`"w+"` : *création pour mise à jour* (lecture et écriture). Si le fichier existe, son (ancien) contenu sera détruit. S'il n'existe pas, il sera créé.
`"a+"` : *écriture en fin de fichier (+ lecture).* Si le fichier existe, son (ancien) contenu est conservé. S'il n'existe pas, il sera créé. Les écritures ont systématiquement lieu en fin de fichier.

Pour réaliser une opération sur un fichier, il faudra **systématiquement utiliser l'adresse de la variable de type FILE** (renvoyée par un appel réussi à la fonction fopen()) **associée à ce fichier**.
Dans la suite du cours, on va décrire les différentes opérations qu'on peut réaliser sur un fichier.

3. La fermeture d'un fichier

Après l'ouverture avec succès d'un fichier et la réalisation de toutes les opérations souhaitées, il ne faut pas oublier de fermer le fichier.

3.1 La fonction fclose()

Prototype : `int fclose(FILE *pFich);`
Cette fonction ferme le fichier associé au pointeur *pFich*. Les opérations effectuées sont :
- le transfert des données du tampon associé;
- la destruction de la variable de type FILE associé au fichier.

En cas de succès, la fonction renvoie 0. En cas d'erreur, elle renvoie la constante EOF.

4. Les opérations d'écriture

Ces opérations se font toutes par l'intermédiaire d'un tampon. De plus, elles entraînent toutes la modification de la valeur de la **position courante** dans le fichier, position courante qui est initialisée à 0 lors de l'ouverture du fichier (sauf pour le mode "a"). La modification correspond exactement au nombre de caractères écrits.

4.1 L'écriture d'un caractère : la fonction fputc()

Prototype : `int fputc( int caractere, FILE *pFich);`

Un appel à cette fonction permet d'écrire le *caractere* dans le fichier associé à *pFich* à la position courante, qui est ensuite automatiquement incrémentée.
Elle renvoie le caractère écrit en cas de succès, et la constante EOF sinon.

Remarque : on peut très bien appeler cette fonction en lui transmettant en 1er paramètre une expression de type *char*.

4.2 L'écriture d'une chaîne de caractères : la fonction fputs()

Prototype : `int fputs( const char* str, FILE *pFich);`

Cette fonction écrit la chaîne de caractères pointée par *str* dans le fichier associé à *pFich* à partir de la position courante, qui est ensuite automatiquement augmentée du nombre de caractères écrits.
Le caractère nul ('\0') qui termine la chaîne de caractères pointée par *str* n'est pas écrit dans le fichier.
Elle renvoie un entier non négatif en cas de succès, et la constante EOF sinon.

4.3 L'écriture des éléments d'un tableau : la fonction fwrite()

Prototype : `int fwrite(void *adr, size_t size, size_t nb, FILE *pFich);`

L'adresse dans la mémoire du 1er élément à écrire (*adr*) est transmise en premier paramètre.
Cette fonction écrit un tableau de *nb* éléments, chaque élément ayant une taille égale à *size* octets, dans le fichier associé à *pFich* à partir de la position courante, qui est ensuite automatiquement augmentée du nombre d'octets écrits.
Cette fonction renvoie le nombre d'éléments qui ont été écrits avec succès dans le fichier. Si tout s'est bien passé, elle doit renvoyer la valeur *nb*.

En interne, cette fonction interprète le bloc pointée par *adr* comme si c'était un tableau de (*size* nb*) éléments de type `unsigned char`, et les écrit les uns après les autres dans le fichier comme si la fonction `fputc()` était appelée pour chaque octet.

4.4 L'écriture formatée : la fonction fprintf()

Prototype : **`int fprintf(FILE *pFich, const char *format, ...);`**

Son fonctionnement est similaire à celui de la fonction `printf()`. Au lieu d'écrire ses données sur la sortie standard (comme le fait `printf()`), cette fonction les écrit dans le fichier associé à *pFich* à partir de la position courante, qui est ensuite automatiquement augmentée du nombre de caractères écrits.

Remarque : la fonction `printf()` n'est que la définition d'un appel à `fprintf()` sur `stdout`.

Par exemple, l'expression :
`printf("hello")`
joue le même rôle que :
`fprintf (stdout, "hello")`

4.5 Remarque

Il est possible de forcer le vidage du tampon associé à un fichier ouvert en écriture en utilisant la fonction `fflush()`.

Prototype : **`int fflush(FILE *pFich);`**

4.6 Exemple

Ecrire un programme qui demande à l'utilisateur le nom du fichier à créer, puis :
- demande à l'utilisateur de saisir un entier;
- l'écrit **sans formatage des données** dans le fichier;
- recommence jusqu'à ce que l'utilisateur saisisse la valeur 0.

```
#include <stdio.h>

int main()
{
    char nomfich[21] ;
    int n ;
    FILE * pfic ; // déclaration d'un pointeur sur un FILE

    printf ("nom du fichier à créer (max 20 caractères) : ") ;
    scanf ("%20s", nomfich) ;       // saisie sécurisée
    pfic = fopen (nomfich, "w") ;
    // on teste si l'ouverture s'est bien passée (TOUJOURS le faire !)
    if (pfic == NULL)
    {
            return -1;  // fin de la fonction main() avec code de retour -1
    }
    do
    {
      printf ("donnez un entier (0 pour sortir) : ") ;
      scanf ("%d", &n) ;
      if (n != 0)
      {
            fwrite (&n, sizeof(int), 1, pfic) ;
      }
    }
    while (n != 0) ;
    fclose (pfic) ; // A ne pas oublier
    return 0;
}
```

Analyse des résultats de l'exécution du programme précédent :
- on ne peut pas lire le fichier créé avec un simple éditeur de texte comme gedit. Pour voir son contenu, il faut utiliser un éditeur hexadécimal (ghex par exemple).
- pour chaque entier saisi de valeur non nulle, on écrit, dans le fichier créé, 4 octets qui correspondent à son code (CPL2 sur 32 bits) stocké en mémoire dans la variable n. La fonction `fwrite()` permet donc d'écrire dans le fichier les données telles sont représentées dans la mémoire du système : elle ne réalise pas de formatage des données (elle ne modifie pas le format des données).

Fichier binaire / fichier texte : définitions
Un **fichier texte** est un fichier binaire dans lequel on ne stocke que des caractères imprimables (ainsi que des sauts de ligne, des tabulations, ...) et qui peut être lu par un **simple éditeur de texte**.
Un **fichier binaire** est un **fichier informatique** contenant des données sous forme d'**octets** pouvant prendre n'importe quelle valeur et qui n'ont donc de sens que pour le **logiciel** qui les utilise *(et non pour les utilisateurs finaux)*. Si par stricte définition tout fichier est binaire, l'usage veut que l'on qualifie un fichier de binaire pour indiquer qu'il ne s'agit pas d'un fichier texte. Les fichiers binaires peuvent être ouverts par des **éditeurs de texte** *(déconseillé aux utilisateurs non expérimentés)*, mais les données y seront mal représentées et surtout, incompréhensibles.
Par exemple, les fichiers contenant du langage machine, du son, de la vidéo, des images, sont des fichiers binaires.
Pour décrire l'organisation des données dans un fichier binaire, on parle de format de fichier (= format des données stockées dans un fichier).

Le fichier créé par le programme est un fichier binaire.
Dans le programme précédent, quelle instruction faut-il modifier pour créer un fichier texte ?
Il faut remplacer l'appel à la fonction `fwrite()` par un appel à la fonction `fprintf()` :
```
fprintf (pfic, "%d\n", n) ;   //on écrit un entier par ligne
```

Autre solution qui permet de comprendre le formatage des données réalisée par la fonction `fprintf()` : après avoir inclus le fichier d'entête <string.h> (utilisation de la fonction `strlen()`), on remplace l'instruction précédente par les suivantes :
```
char chn[20]; // déclaration d'un tableau de caractères
sprintf (chn, "%d\n", n) ; //si n=12, on place dans chn la chaîne "12\n"
fwrite(chn, 1, strlen(chn), pfic);
```

5. Les opérations de lecture

Ces opérations se font toutes par l'intermédiaire d'un tampon. De plus, elles entraînent toutes la modification de la valeur de la **position courante** dans le fichier, position courante qui est initialisée à 0 lors de l'ouverture du fichier. La modification correspond exactement au nombre de caractères lus.

5.1 La lecture d'un caractère : la fonction fgetc()

Prototype : **`int fgetc(FILE *pFich);`**

Un appel à cette fonction renvoie, sous forme entière, le caractère situé à la position courante dans le fichier associé à *pFich*. Suite à cet appel, la position courante est incrémentée.
En cas d'erreur, ou lorsqu'il n'y a plus de caractère à lire (fin de fichier atteinte), la fonction renvoie la constante EOF.

Remarque : on peut très bien appeler cette fonction en affectant sa valeur de retour dans une variable de type *char*.

5.2 La lecture d'une ligne (suite de caractères terminée par un '\n') : la fonction fgets()

Prototype : **`char* fgets(char *pchaine, int taille, FILE *pFich);`**

Cette fonction lit des caractères dans le fichier associé à *pFich*, à partir de la position courante, et les range dans la mémoire système à partir de l'adresse *pchaine*.
Le paramètre *taille* fixe le nombre maximum de caractères que la fonction peut lire dans le fichier; ce nombre est égal à *taille -1*.
Cette fonction lit les caractères un par un. Si elle lit un caractère '\n', elle le range dans le tableau pointée par *pchaine* et **s'arrête.**

Ainsi, l'arrêt de la lecture est toujours provoquée par une des 3 causes suivantes (celle qui apparaît en premier) :
- le nombre de caractères lus dans le fichier a atteint son maximum (égal à *taille -1*);
- un caractère '\n' vient d'être lu;
- la fin de fichier est atteinte.

Un caractère nul ('\0') est toujours ajouté dans le tableau pointé par *pchaine*, après le dernier caractère lu.
Si la fonction réussit à lire au moins un caractère, elle renvoie *pchaine*. Sinon, elle renvoie la constante NULL.

Remarque : Avec la fonction `scanf()`, quand on utilise le code de conversion **s**, on ne peut pas lire de chaîne de caractères contenant des espaces. La fonction fgets() peut être utilisée pour **lire une ligne sur l'entrée standard** `stdin`.
Exemple : (`^` désigne un espace et `@` un saut de ligne)

```
char Line[8];     // déclaration d'une variable servant pour la saisie
char *ret;        // on y affecte la valeur de retour de fgets()
ret = fgets(Line, 8, stdin);   // saisie sécurisée, pas de risque de
                                  //débordement de tableau
```

`ab@`	Line = ab@\0 , ret = Line après la saisie, tampon = vide
`ab^12@`	Line = ab^12@\0 , ret = Line après la saisie, tampon = vide
`ab^123456@`	Line = ab^1234@\0 , ret = Line après la saisie, tampon = `56@`
`@`	Line = @\0 , ret = Line après la saisie, tampon = vide

5.3 La lecture des éléments d'un tableau : la fonction fread()

Prototype : **`int fread(void *adr, size_t size, size_t nb, FILE *pFich);`**

Le paramètre *adr* (transmis en premier paramètre) correspond à l'adresse mémoire à partir de laquelle on souhaite ranger les éléments à lire.
Cette fonction lit un tableau de *nb* éléments, chaque élément ayant une taille égale à *size* octets, dans le fichier associé à *pFich* à partir de la position courante, qui est ensuite automatiquement augmentée du nombre d'octets lus.
Cette fonction renvoie le nombre d'éléments qui ont été lus avec succès dans le fichier. Si la valeur renvoyée est inférieure à *nb*, cela signifie que la fin de fichier a été atteinte (cette fonction ne renvoie jamais la constante EOF).

5.4 La lecture formatée : la fonction fscanf()

Prototype : **`int fscanf(FILE *pFich, const char *format, ...);`**

Son fonctionnement est similaire à celui de la fonction `scanf()`. Au lieu de lire ses données sur l'entrée standard (comme le fait `scanf()`), cette fonction les lit dans le fichier associé à *pFich* à partir de la position courante. Après l'appel de `fscanf()`, la position courante repère le 1er caractère non consommé.
La fonction renvoie le nombre de variables convenablement lues (qui peut être nul en cas de tentative de lecture d'un caractère invalide), ou la constante EOF si la fin de fichier est atteinte.

Remarque : la fonction `scanf()` n'est que la définition d'un appel à `fscanf()` sur `stdin`.
Par exemple avec `car` une variable de type `char`, l'expression :
`scanf("%c", &car)`

joue le même rôle que :
`fscanf (stdin, "%c", &car)`

5.5 La fonction feof()

Prototype : **`int feof(FILE *pFich);`**

Cette fonction renvoie une valeur non nulle lorsqu'une fin de fichier a été détectée, et la valeur 0 dans le cas contraire.

La fin de fichier n'est détectée que lorsque l'on cherche à lire un caractère alors qu'il n'y en a plus de disponible, et non pas, dès que l'on a lu le dernier caractère.

Remarque : comme toutes les fonctions de lecture disposent d'un mécanisme permettant de savoir si une fin de fichier a été détectée, on peut bien souvent se passer d'utiliser la fonction `feof()`.

5.6 Exemple

Ecrire un programme qui lit tous les entiers écrits dans le fichier binaire créé précédemment, et les affiche à l'écran.

```
#include <stdio.h>

int main()
{
    int n , res, i = 1;
    FILE * entree ;     // déclaration d'un pointeur sur un FILE

    entree = fopen ("sauvbin", "r") ;
    // on teste si l'ouverture s'est bien passée (TOUJOURS le faire !)
    if (entree == NULL)
    {
        printf("Fichier à lire inexistant\n");
        return -1;      // fin de la fonction main() avec code de retour -1
    }
    do
    {
        //res = fscanf(entree,"%d", &n);
        //if (res != EOF) printf ("Nombre %d : %d\n", i, n) ;
        res = fread(&n, sizeof(int), 1, entree);
        if (res != 0)
        {
            printf ("Nombre %d : %d\n", i, n) ;
        }
        i++;
    }
    while(res != 0);
    //while(res != EOF);
    fclose (entree) ;   // A ne pas oublier
    return 0;
}
```

Dans le programme précédent, quelle instruction faut-il modifier pour lire le fichier texte ?
Il faut remplacer l'appel à la fonction `fread()` par un appel à la fonction `fscanf()` :
Attention, la fonction `fscanf()` renvoie la constante EOF si la fin de fichier est atteinte.