Introduction au langage C

14 janvier 2010

Table des matières

Chapitre 1

Les fichiers

1.1 Généralités sur les fichiers

1.1.1 Importance des fichiers

Les données stockées en mémoire sont perdues dès la sortie du programme. Les fichiers, sur supports magnétiques (bande, disque), constituent par contre des moyens de conservation à long terme des données produites par un programme.

1.1.2 Définition

Un fichier est un bloc d'informations binaires, c'est à dire une suite de 0 et de 1, portant un nom et conservé dans une mémoire de masse. Il existe plusieurs types de fichiers selon le format du contenu ou selon le mode d'accès au contenu :

1.1.2.1 Types de fichiers selon le format du contenu

1. les fichiers textes : un fichier texte est un fichier dont le contenu, la plupart du temps au format ASCII, peut être lu comme un texte ordinaire. Il est organisé en lignes de caractères, que l'on pourrait lire ou écrire avec un éditeur. Par opposition, un fichier binaire n'est pas lisible, ou l'est très difficilement, par un être humain ;
   - fichiers XML ;
   - code source d'un programme ;
   - fichiers de configuration d'un logiciel pouvant être lus et édités par l'utilisateur ;
   - fichiers destinés à être lus par l'utilisateur ;
   - fichiers `<stdin>` et `<stdout>`.
2. les fichiers binaires : leur contenu est uniquement constitué de 0 et 1. Par exemple, un nombre réel, dans un fichier binaire sera traduit par les 4 octets constituant son codage en virgule flottante ;
   - code objet obtenu après compilation ;

– fichier de base de données structuré en enregistrements de taille fixe ou variable ;
– document de traitement de texte ;
– fichiers multimédias : images, sons, vidéos.

De ce fait, ce type de lecture/écriture est plus rapide et engendre des fichiers plus petits. L'inconvénient est que ces fichiers ne sont pas consultables par un éditeur de texte.

1.1.2.2 Types de fichiers selon le mode d'accès

1. les fichiers à accès direct ;
2. les ficheirs à accès séquentiel : dans ces fichiers, à partir de l'ouverture du fichier, les données sont lues dans l'ordre dans lequel elles sont stockées ; il n'est pas possible d'accéder directement à une donnée particulière. Au fur et à mesure des lectures ou écritures, un pointeur de fichier avance automatiquement d'une donnée à la suivante..

1.1.2.3 Quelques caractéristiques des fichiers

Les fichiers sont enregistrés sur le disque dur sous la forme `identificateurDuFichier.ext`. `.ext` représente l'extension du fichier et constitue un moyen de reconnaître le type de programme avec lequel ce fichier peut être ouvert.

La longueur du nom, c'est à dire la longueur de la chaîne de caractères que constitue l'ensemble formé de l'identificateur du fichier et son extension, peut varier suivant le système d'exploitation :

– 8 caractères pour l'identificateur et 3 pour l'extension sous `DOS` et `Windows 3.1` ;
– 256 caractères pour l'identificateur et l'extension sous `Windows 9x` (95, 98, Me et NT) ;
– 256 sous les systèmes `UNIX` ;

Ainsi, sous `DOS` ou `Windows 3.1`, un fichier provenant de `Windows 9x` aura un nom tronqué comportant les 6 premiers caractères du nom suivi de `~x` où `x` représente un chiffre qui est incrémenté à chaque fois qu'un fichier porte le même nom. C'est-à-dire que si un fichier nommé `fichie~1` existe déjà il nommera le suivant `fichie~2`.

1.2 Ouverture et fermeture d'un fichier en langage C

1.2.1 Ouverture

Le langage C offre la possibilité de lire et d'écrire des données dans un fichier. Pour des raisons d'efficacité, les accès à un fichier se font par l'intermédiaire d'une mémoire-tampon (buffer), ce qui permet de réduire le nombre d'accès aux périphériques (disque...). Avant d'être accédé en mode lecture ou écriture, un fichier doit être ouvert par la fonction `fopen` définie dans la bibliothèque standard `stdio`. `fopen`

récupère le nom du fichier, prépare le système d'exploitation à la lecture du fichier et retourne un pointeur, le pointeur du fichier, chargé de parcourir le contenu du fichier. Le pointeur du fichier est une structure qui stockant des informations concernant le fichier, tel que la position du caractère courant, le mode de parcours (lecture ou écriture) du fichier, les éventuelles erreurs rencontrées ou le marqueur de fin de fichier.

La déclaration d'un pointeur de fichier est donnée par la synstaxe :

```
FILE *fp;
FILE *fopen(char *nomfic, char *mode):
```

1. `FILE` est un type de donnée au même titre que les types de données `int` ou `char`.
2. `name` est une chaîne de caratère contenant le nom du fichier ;
3. `mode` : est une chaîne de caratère indiquant l'intention du programmeur de lire, ou écrire, ou ajouter des données dans le fichier ;

Les différentes valeurs que peuvent prendre `mode` sont :

Mode	**Description**
r	ouverture d'un fichier texte en lecture
w	ouverture d'un fichier texte en écriture
a	ouverture d'un fichier texte en écriture à la fin
rb	ouverture d'un fichier binaire en lecture
wb	ouverture d'un fichier binaire en écriture
ab	ouverture d'un fichier binaire en écriture à la fin
r+	ouverture d'un fichier texte en lecture/écriture
w+	ouverture d'un fichier texte en lecture/écriture
a+	ouverture d'un fichier texte en lecture/écriture à la fin
r+b	ouverture d'un fichier binaire en lecture/écriture
w+b	ouverture d'un fichier binaire en lecture/écriture
a+b	ouverture d'un fichier binaire en lecture/écriture à la fin

Ces modes d'accès ont pour particularités :

– Si le mode contient la lettre `r` (*read*, c'est à dire *lecture seule*), le fichier doit nécessairement exister ;
– Si le mode contient la lettre `w` (*write*, c'est à dire *écriture*), le fichier peut ne pas exister. Dans ce cas, il sera créé. Si le fichier existe déjà, son ancien contenu sera vidé ;
– Si le mode contient la lettre `a` (append, c'est à dire ajout), le fichier peut ne pas exister. Dans ce cas, il sera créé. Si le fichier existe déjà, les nouvelles données seront ajoutées à la suite du contenu actuel.

La valeur retournée par `fopen` est un `flot de données`. Si l'exécution de cette fonction ne se déroule pas normalement, la valeur retournée est le pointeur `NULL`. Il est

donc recommandé de toujours tester si la valeur renvoyée par la fonction `fopen` est égale à `NULL` afin de détecter les erreurs (lecture d'un fichier inexistant...).

Trois flots standard peuvent être utilisés en langage C sans qu'il soit nécessaire de les ouvrir ou de les fermer :

- `stdin` (standard input) : unité d'entrée (par défaut, le clavier) ;
- `stdout` (standard output) : unité de sortie (par défaut, l'écran) ;
- `stderr` (standard error) : unité d'affichage des messages d'erreur (par défaut, l'écran).

En cas d'erreur (chemin invalide, fichier inexistant ouvert en mode lecture, ...), le pointeur `NULL` est retourné, le type d'erreur est donné dans une variable `errno`, détaillée dans `errno.h`. La fonction `void perror(char *s mess)` affichera le message correspondant à l'erreur, en général on lui donne le nom du fichier. Il est fortement conseillé d'afficher systématiquement les messages d'erreur sur `stderr` afin que ces messages apparaissent à l'écran même lorsque la sortie standard est redirigée.
Exemple :

```
#include <stdio.h>
int main()
{
     FILE * fp;

     f = fopen("monFichier.txt", "w");
     if (f != NULL)
     {
  fprintf(fp, "Bonjour les amis!\n");
  fclose(fp);
     }
     else
  perror("monFichier.txt");

     return 0;
}
```

1.2.2 Fermeture

L'inverse de la fonction `fopen` est la fonction `fclose`. `fclose` interrompt la connection, établie par `fopen` entre le pointeur du fichier et le nom physique du fichier, libérant ainsi le pointeur du fichier. Si des données sont encore en attente d'écriture sur disque, le tampon est vidé et les données sont écrites physiquement sur le fichier disque.

La syntaxe de déclaration de la fontion `fclose` est la suivante :

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

La fonction `fclose` retourne un entier qui vaut zéro si l'opération s'est déroulée normalement (et une valeur non nulle en cas d'erreur).

1.3 Les entrées et les sorties formatées

1.3.1 La fonction fgets()

Syntaxe d'utilisation :

```
char *fgets(char *ligne, int longueurLigne, FILE *fp)
```

Le fichier est lu ligne par ligne. La fonction `fgets` lit une ligne d'au plus `longueurLigne-1` caractères du fichier `fp` dans la chaîne `ligne`.

1. la chaîne doit avoir été déclarée pour contenir au moins `longeurLigne` caractères ;
2. normalement, `fgets` retourne `ligne`. Mais lorsque la fin du fichier est atteinte, ou en cas d'erreur, la fonction `fgets()` retourne la valeur `NULL`. Dans ce cas, la chaîne destination n'est pas modifiée.

Exemple :

Pour lire un fichier du début à la fin, on utilisera en général la méthode suivante. Le programme ci-dessous affiche le nombre de lignes du fichier **monFichier.txt** :

```
int main(void)
{
     int n = 0; /* nombre de lignes lues */
     char ligne[256];
     FILE *fp = fopen( "monFichier.txt", "r" ); /* ouverture en mode lecture */
     if (fp == NULL)
     {
         printf("erreur d'ouverture\n");
         return 1;        /* sortie du programme (code erreur 1) */
     }
     while (fgets( ligne, 256, fp ) != NULL)
     { /* essai lecture ligne */
         n++;
     }
     printf("%d lignes lues\n", n);    /* affiche */
     fclose(fp);  /* fermeture du fichier */
     return 0;
}
```

Attention :

– si le fichier est vide, ce programme affichera bien 0 ;
– si certaines lignes comptes plus de 256 caractères, ou bien si des erreurs de lecture se produisent, le résultat affiché sera faux.

1.3.2 La fonction fputs()

```
int fputs(char *lignes, FILE *fp)
```

1. `fputs` retourne `EOF` en cas d'erreur ou, ou une valeur positive en cas de succès ;
2. normalement, `fgets` retourne `ligne`. Mais lorsque la fin du fichier est atteinte, ou en cas d'erreur, la fonction `fgets()` retourne la valeur `NULL`. Dans ce cas, la chaîne destination n'est pas modifiée.

1.3.3 La fonction d'écriture fprintf()

La fonction `fprintf`, analogue à `printf`, permet d'écrire des données dans un fichier.

Sa syntaxe est :

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

`flot` est le flot de données retourné par la fonction `fopen`. Les spécifications de format utilisées pour la fonction fprintf sont les mêmes que pour `printf`. En cas de succès, le nombre total de caractères imprimé est retourné. En cas d'échec, un nombre négatif est retourné.

Exemple :

```
/* Exemple d'utilisation de fprintf*/
#include <stdio.h>

int main()
{
    FILE * fp;
    int n;
    char nom[100];

    fp = fopen ("monFichier.txt","w");
    for (n=0 ; n<3 ; n++)
    {
        puts("Veuillez entrer un nom: ");
        gets(nom);
        fprintf (fp, "Nom %d [%-10.10s]\n",n,nom);
```

```
    }
   fclose (fp);

   return 0;
}
```

Cet exemple sollicite 3 noms, de l'utilisateur, et les impriment dans le fichier `monFichier.txt`. Chacun des noms est inséré dans une ligne de longeur fixe (19 caractères + caractère retour chariot).

Supposons que les 3 noms entrés sont Nabi, Millogo Moussa, et Sissoko, le fichier `monFichier` contiendra :

```
Nom 1 [Nabi      ]
Nom 2 [Millogo Mo]
Nom 3 [Sissoko   ]
```

1.3.4 La fonction de saisie fscanf()

La fonction `fscanf`, analogue à `scanf`, permet de lire des données d'un fichier. Sa syntaxe est semblable à celle de `scanf` :

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

`flot` est le flot de données retourné par `fopen`. Les spécifications de format sont ici les mêmes que celles de la fonction `scanf`.

Exemple :

```
/* fscanf example */
#include <stdio.h>

int main()
{
    char chaine[80];
    float f;
    FILE * fp;

    fp = fopen ("monFicher.txt","w+");
    fprintf (fp, "%f %s", 3.1416, "PI");
    rewind (fp);
    fscanf (fp, "%f", &f);
    fscanf (fp, "%s", chaine);
    fclose (fp);
    printf ("J'ai lu: %f et %s \n",f, chaine);
```

```
    return 0;
}
```

Cet exemple crée un fichier nommé `monFichier.txt` et y imprime un nombre réel et une chaîne de caractères. Ensuite, l'indicateur de position se place au début du flot (rembobinage du flot) et les valeurs sont lues avec la fonction `fscanf`.

A l'affichage à l'écran, l'on aura :

```
J'ai lu: 3.141600 and PI
```

1.4 Impression et lecture de caractères

1.4.1 fgetc()

Similaires aux fonctions `getchar` et `putchar`, les fonctions `fgetc` et `fputc` permettent respectivement de lire et d'écrire un caractère dans un fichier. La fonction `fgetc`, de type `int`, retourne le caractère lu dans le fichier. Elle retourne la constante `EOF` lorsqu'elle détecte la fin du fichier.

Syntaxe :

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

`flot` est le flot de type `FILE*` retourné par la fonction `fopen`. Comme pour la fonction `getchar`, il est conseillé de déclarer de type `int` la variable destinée à recevoir la valeur de retour de `fgetc` pour pouvoir détecter correctement la fin de fichier.

1.4.2 fputc()

La fonction `fputc` écrit caractere dans le flot de données :

Syntaxe :

```
int fputc(int caractere, FILE *flot)
```

Elle retourne l'entier correspondant au caractère lu (ou la constante `EOF` en cas d'erreur).

Exemple :

Ainsi, le programme suivant lit le contenu du fichier texte entree, et le recopie caractère par caractère dans le fichier sortie :

```
#include <stdio.h>
#include <stdlib.h>
#define ENTREE "entree.txt"
#define SORTIE "sortie.txt"

int main(void)
{
    FILE *f_in, *f_out;
    int c;

    if ((f_in = fopen(ENTREE,"r")) == NULL)
    {
         fprintf(stderr, "\nErreur: Impossible de lire le
                                  fichier %s\n",ENTREE);
         return(EXIT_FAILURE);
    }
    if ((f_out = fopen(SORTIE,"w")) == NULL)
    {
         fprintf(stderr, "\nErreur: Impossible d'ecrire dans le
                                 fichier %s\n", SORTIE);
         return(EXIT_FAILURE);
    }
    while ((c = fgetc(f_in)) != EOF)
         fputc(c, f_out);
    fclose(f_in);
    fclose(f_out);
    return(EXIT_SUCCESS);
}
```

1.4.3 renommer un fichier

La fonction `rename` permet de renommer un fichier lorsque cela est possible. Si la fonction réussit, 0 est retourné :

Syntaxe :

```
int rename(const char * oldname, const char * newname);
```

1.4.4 supprimer un fichier

La fonction `remove` permet de supprimer un fichier. Si la fonction réussit, 0 est retourné.

Syntaxe :

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

1.5 Redigirer un flux d'entrée ou de sortie

La fonction suivante `freopen` ferme le fichier associé au flot `fp`, ouvre le fichier dont le nom est spécifié par `filename` selon le mode spécifié par `mode` en lui associant le flot représenté par `fp` puis retourne `fp` ou `NULL` si une erreur s'est produite. Dans le programme suivant, la sortie standard est redirigée vers le fichier `monFichier.txt` :

```
FILE * freopen(const char * filename, const char * mode, FILE * fp);
```

Exemple :

```
#include <stdio.h>

int main()
{
    if (freopen("out.txt", "w", stdout) != NULL)
    {
         printf("Bonjour les amis\n");
         fclose(stdout);
    }
    else
         perror("monFichier.txt");

    return 0;
}
```

Exercices :

1. *agenda* : modifier l'agenda de l'exercice tel en permettant de sauver les données sur disque. On utilisera un fichier binaire à accès direct. On pourra ensuite apporter les améliorations suivantes : recherche rapide par dichotomie sans lire tout le fichier (en le supposant classé par ordre alphabétique), création de fichiers index classés alphabétiquement sur les noms, département et ville pour accès rapide par dichotomie, les autres se faisant par recherche séquentielle, avec possibilité d'ajout, suppression, édition du fichier.
2. *fic_formaté* : modifier le programme de produit de matrices en lui permettant de donner sur la ligne de commande trois noms de fichiers : les deux premiers contiendront la description des matrices à multiplier, le dernier sera créé et contiendra le résultat. Les fichiers seront formatés, contiendront en première ligne le nombre de lignes puis le nombre de colonnes, puis chaque ligne du fichier contiendra une ligne de la matrice.