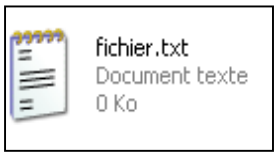
CHAPITRE 5 : LES FICHIERS DE DONNÉES

I. Introduction :



Nous avons déjà eu l'occasion d'étudier les « entrées-sorties conversationnelles » *(Clavier et Ecran)*, c'est-à-dire les fonctions permettant d'échanger des informations entre le programme et l'utilisateur. Dans ce chapitre on va étudier les fonctions permettant au programme d'échanger des informations avec des **fichiers**.

II. Généralités sur les fichiers

II-1. Importance des fichiers :

Les données stockées en mémoire en utilisant les variables, sont perdues dès la sortie du programme. Les fichiers, sur mémoire de masse (disque dur, USB,…), constituent par contre des moyens de conservation à long terme des données produites par un programme.

II-2. Types de fichiers selon le format du contenu :

- *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. (*Exemple :* code source d'un programme)

- *les fichiers binaires :* leur contenu est uniquement constitué de 0 et 1 (Fichiers multimédias : images, sons, vidéos).

II-3. Types de fichiers selon le mode d'accès :

- *Les fichiers à accès direct* **:** l'accès direct consiste à se placer immédiatement sur l'information souhaitée, sans avoir à parcourir celles qui la précèdent.
- *Les fichiers à accès séquentiel* **:** l'accès séquentiel consiste à traiter les informations séquentiellement, c'est-à-dire dans l'ordre où elles apparaissent (ou apparaîtront) dans le fichier ;

III. MANIPULATION DES FICHIERS :

Les opérations possibles avec les fichiers: Créer un fichier – Ouvrir un fichier – Fermer un fichier – Lire un fichier – Ecrire dans un fichier – Détruire un fichier – Renommer un fichier.

La plupart des fonctions permettant la manipulation des fichiers sont rangées dans la bibliothèque standard *stdio.h*, certaines dans la bibliothèque *io.h*.

III-1. Ouverture et fermeture d'un fichier :

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.h.

La fonction *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 stocke 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 syntaxe :

```
FILE *p;    /* majuscule obligatoire pour FILE */
```

FILE est une structure définie dans la bibliothèque **stdio.h**

Voici ce qu'il faut faire à chaque fois dans l'ordre quand on veut ouvrir un fichier (que ce soit pour lire ou pour écrire dedans) :

- On déclare un pointeur de fichier.
- On appelle la fonction **d'ouverture de fichier** *fopen* qui nous renvoie un pointeur sur le fichier.
- **On vérifie si l'ouverture a réussi** (c'est-à-dire si le fichier existait) en testant la valeur du pointeur qu'on a reçu. Si le pointeur vaut NULL, c'est que l'ouverture du fichier n'a pas marché, dans ce cas on ne peut pas continuer (il faut afficher un message d'erreur).
- Si l'ouverture a marché (si le pointeur est différent de NULL donc), alors on peut **lire et écrire dans le fichier**
- Une fois qu'on a **terminé de travailler sur le fichier**, il faut penser à le "fermer" avec la fonction *fclose*.

a. Ouverture du fichier : *La fonction fopen*

Voyons le prototype de la fonction fopen :

```
FILE *fopen(char * nomDuFichier, char * modeOuverture)
```

Cette fonction attend 2 paramètres :

- Une chaîne de caractères contenant le nom du fichier à ouvrir.
- Une chaîne de caractères indiquant le mode d'ouverture du fichier, c'est-à-dire une indication qui dit si vous voulez juste écrire dans le fichier, juste lire dans le fichier, ou les deux à la fois.

Voici les modes d'ouvertures possibles pour les fichiers textes :

- "r" : lecture seule. Vous pourrez lire le contenu du fichier, mais pas écrire dedans. Le fichier doit avoir été créé au préalable.
- "w" : écriture seule. Vous pourrez écrire dans le fichier, mais pas lire son contenu. Si le fichier n'existe pas, il sera créé.
- "a" : mode d'ajout. Vous écrirez dans le fichier, en partant de la fin du fichier. Vous rajouterez donc du texte à la fin du fichier. Si le fichier n'existe pas, il sera créé.
- "r+" : lecture et écriture. Vous pourrez lire et écrire dans le fichier. Le fichier doit avoir été créé au préalable.
- "w+" : lecture et écriture, avec suppression du contenu au préalable. Le fichier est donc d'abord vidé de son contenu, et vous écrivez et lisez ensuite dedans. Si le fichier n'existe pas, il sera créé.
- "a+" : ajout en lecture / écriture à la fin. Vous écrivez et lisez du texte à partir de la fin du fichier. Si le fichier n'existe pas, il sera créé.

Exemple :

Le code suivant ouvre le fichier test.txt en mode "r+" (lecture / écriture) :

```
#include <stdio.h>
int main()
{   FILE* p = NULL;
   fichier = fopen("test.txt", "r+");
   return 0;
}
```

b. Tester l'ouverture du fichier :

En cas d'erreur (chemin invalide, fichier inexistant ouvert en mode lecture, ...), la fonction *fopen* retourne le pointeur *NULL*.

```
#include <stdio.h>
int main()
{   FILE* p = NULL;
   p = fopen("test.txt", "r+");

   if(p= = NULL)  /* Il y a eu un problème */
    {
        printf("Erreur d\' ouverture\n") ;
    }
  else
     {
       /* Si l'ouverture a pu se faire */
    }

}
```

c. Fermeture du fichier : La fonction fclose

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.

Exemple :

```
fclose(p) ;
```

III-2. Lecture et écriture séquentielles dans un fichier texte :

Une fois le fichier ouvert, le langage C permet plusieurs types d'accès à un fichier :

- par données formatées : **fscanf** ,**fprintf**.
- par caractère : **fgetc**,**fputc**.
- par ligne : **fgets** ,**fputs**.

a. Ecrire dans le fichier :

Il existe plusieurs fonctions capables d'écrire dans un fichier. Ce sera à vous de choisir celle qui est la plus adaptée à votre cas.

Voici les 3 fonctions que nous allons étudier :

- **fputc :** écrit un caractère dans le fichier (UN SEUL caractère à la fois).
- **fputs :** écrit une chaîne dans le fichier
- **fprintf :** écrit une chaîne "formatée" dans le fichier, fonctionnement quasi-identique à printf

fputc :

Cette fonction transfère le caractère spécifié dans le fichier représenté par le pointeur de fichier passé en paramètre

Exemple :

```
#include <stdio.h>
 int main()
{   FILE* p;
    p = fopen("test.txt", "w");
    if (p != NULL)
   {  fputc('A', p); // Ecriture du caractère A
      fclose(fichier);
   }

}
```

Le code suivant écrit la lettre 'A' dans test.txt (si le fichier existe, il est remplacé ; si il n'existe pas, il est créé). Il y a tout dans ce code : ouverture, test de l'ouverture, écriture et fermeture.

fputs :

Cette fonction écrit une chaîne de caractères dans le flot spécifié (elle n'écrit pas le caractère '\0'),

Exemple :

Testons l'écriture d'une chaîne dans le fichier :

```
#include <stdio.h>
int main()
{   FILE* p;
    p = fopen("test.txt", "w");
    if (p != NULL)
   {  fputs("Salut mes eleves \nComment allez-vous ?", p);
      fclose(p);
   }
}
```

fprintf :

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

Exemple :

Ce code demande l'âge de l'utilisateur et l'écrit dans le fichier :

```
#include <stdio.h>
int main()
{   FILE* p;
   int age;
    p = fopen("test.txt", "w");
    if (p != NULL)
   {       // On demande l'âge
      printf("Quel age avez-vous ? ");
      scanf("%d", &age);
       // On l'écrit dans le fichier
      fprintf(p, "Le Monsieur qui utilise le programme, il a %d ans", age);
      fclose(p);
   }
}
```

b. Lecture :

Nous pouvons utiliser quasiment les mêmes fonctions que pour l'écriture, le nom change juste un petit peu :

- **fgetc :** lit un caractère
- **fgets :** lit une chaîne de caractères.
- **fscanf :** lit une chaîne formatée

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.

Exemple :

On va écrire un code qui lit tous les caractères d'un fichier un à un, et qui les affiche à chaque fois à l'écran. La boucle s'arrête quand fgetc renvoie EOF (qui signifie End Of File, c'est-à-dire "fin du fichier").

```
#include <stdio.h>
int main()
{   FILE* p ;
   int caractereActuel;
    p = fopen("test.txt", "r");
    if (p != NULL)
   {  caractereActuel = fgetc(p); // On initialise caractereActuel
                                  // Boucle de lecture des caractères un à un
                                  // On continue tant que fgetc n'a pas retourné EOF (fin de fichier)
        while (caractereActuel != EOF)
        {   printf("%c", caractereActuel); // On affiche le caractère stocké dans caractereActuel
            caractereActuel = fgetc(p); // On lit le caractère suivant
        }
        fclose(p);
   }}
```

fgets :

Cette fonction lit une chaîne à partir d'un fichier. Ca vous évite d'avoir à lire tous les caractères un par un. La fonction **lit au maximum une ligne** (elle s'arrête au premier \n qu'elle rencontre). Si vous voulez lire plusieurs lignes, il faudra faire une boucle.
Le prototype de la fonction est :

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

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

- la chaîne doit avoir été déclarée pour contenir au moins longeurLigne caractères
- 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 1: *Lire une ligne avec fgets*

```
#include <stdio.h>
int main( )
{   FILE* p;
   char chaine[30] ;
    p = fopen("test.txt", "r");
    if (p != NULL)
   {  fgets(chaine,30 , p); // On lit maximum 30 caractères du fichier, on stocke le tout dans "chaine"
      printf("%s", chaine); // On affiche la chaîne
       fclose(p);
   }
}
```

Exemple 2: *Lire tout le fichier avec fgets*

On n'a plus qu'à faire un while pour boucler tant que fgets ne renvoit pas NULL!

```
int main()
{   FILE*p;
   char chaine[30];
    p = fopen("test.txt", "r");
    if (p != NULL)
   { while (fgets(chaine, 30, p) != NULL) // On lit le fichier tant qu'on ne reçoit pas d'erreur (NULL)
      {  printf("%s", chaine); // On affiche la chaîne qu'on vient de lire
      }
       fclose(p);
   }
}
```

fscanf :

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

Exemple :

```
#include <stdio.h>
int main()
{
    char chaine[80];
    float f;
    FILE * p;
     p = fopen("test.txt", "w+");

      if (fichier != NULL)
    {
            fprintf (p, "%f %s", 3.1416, "PI");
            rewind (p);   // placer l'indicateur de position au début du fichier
            fscanf (p, "%f", &f);
            fscanf (p, "%s", chaine);
            fclose (p);
            printf ("J'ai lu: %f et %s \n",f, chaine);
      }
}
```

Cet exemple crée un fichier nommé *test.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.

III-3. Se déplacer dans un fichier :

rewind : remet le curseur au début du fichier.

III-4. Renommer un fichier : `rename("test.txt", "test_renomme.txt");`

III-5. supprimer un fichier : `remove("test.txt");`