Chapitre 1
LES FICHIERS

I. INTRODUCTION

Nous avons déjà eu l'occasion d'étudier les « entrées-sorties conversationnelles », c'est-à-dire les fonctions permettant d'échanger des informations entre le programme et l'utilisateur.
Lors de l'exécution d'un programme le résultat envoyé est stocké d'une manière temporaire dans la mémoire vive (RAM), dés qu'on ferme la fenêtre d'exécution les résultats sont perdus.
Pour garder une trace d'exécution d'un programme, on propose ici d'étudier les fonctions permettant au programme d'échanger des informations avec des fichiers.

II. GENERALITE SUR LES FICHIERS

II.1 Définition

Un fichier désigne un ensemble d'informations situé sur une « mémoire de masse » telle que un disque dur ou une Clé USB, etc.

II.2 Caractéristiques d'un fichier :

- ✓ il possède un nom qui permet de l'identifier de manière unique (unicité),
- ✓ il est rangé sur un support permanent (disques),

II.3 Les techniques d'accès un fichier:

On distingue généralement deux types d'accès:

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

En fait, pour des fichiers disque (ou disquette), la distinction entre accès séquentiel et accès direct n'a plus véritablement de raison d'être. **Le langage C ne distingue pas les fichiers à accès séquentiel des fichiers à accès direct**

II.4 Les types de fichiers:

On distingue deux types de fichiers:
1-Un fichier texte:
Fichier dit « texte », les informations sont codées en ASCII. Ces fichiers sont listables. Le dernier octet de ces fichiers est EOF (caractère ASCII spécifique).
2- Un fichier binaire:
Fichier dit «binaire», les informations sont codées telles que. Ce sont en général des fichiers de nombres. Ils ne sont pas listables.

III. MANIPULATION DES FICHIERS

Opérations possibles avec les fichiers: Créer - Ouvrir - Fermer - Lire – Ecrire.... La plupart des fonctions permettant la manipulation des fichiers sont rangées dans la bibliothèque standard STDIO.H.
1 - Déclaration: **FILE *fichier; /* majuscules obligatoires pour FILE */**
Cette déclaration signifie que fichier est un pointeur sur un objet de type FILE . Ce nom désigne en fait un modèle de structure défini dans le fichier stdio.h (par une instruction typedef, ce qui explique l'absence du mot struct).
N'oubliez pas que cette déclaration ne réserve qu'un emplacement pour un pointeur. C'est la fonction fopen qui créera effectivement une telle structure et qui en fournira l'adresse en résultat.
2 - Ouverture: **FILE *fopen(char *nom, char *mode);**
La fonction fopen est ce que l'on nomme une fonction d'ouverture de fichier. Elle possède deux arguments :

- Le nom du fichier concerné, fourni sous forme d'une chaîne de caractères ; notez qu'en général ce nom pourra comporter une information (chemin, répertoire...) permettant de préciser l'endroit où se trouve le fichier.

- Le mode c'est une indication, fournie elle aussi sous forme d'une chaîne, précisant ce que l'on souhaite faire avec ce fichier.

Les différents modes d'accès sont les suivants :

mode (pour les fichiers TEXTES) :

« r » lecture seule
« w » écriture seule (destruction de l'ancienne version si elle existe)
« w+ » lecture/écriture (destruction ancienne version si elle existe)
« r+ » lecture/écriture d'un fichier existant (mise à jour), pas de création d'une nouvelle version.
« a+ » lecture/écriture d'un fichier existant (mise à jour), pas de création d'une nouvelle version, le pointeur est positionné à la fin du fichier.

mode (pour les fichiers BINAIRES) :

« rb » lecture seule
« wb » écriture seule (destruction de l'ancienne version si elle existe)
« wb+ »lecture/écriture (destruction ancienne version si elle existe)
« rb+ » lecture/écriture d'un fichier existant (mise à jour), pas de création d'une nouvelle version.
« ab+ »lecture/écriture d'un fichier existant (mise à jour), pas de création d'une nouvelle version, le pointeur est positionné à la fin du fichier.

A l'ouverture, le pointeur est positionné au début du fichier (sauf « a+ » et « ab+ »)

Exemple : **FILE *fichier ;**
fichier = fopen(« c :\toto.dat », « rb ») ;

3 - Fermeture: **int fclose(FILE *fichier);**

la fonction fclose réalise ce que l'on nomme une fermeture de fichier. Elle force l'écriture sur disque du tampon associé au fichier. Il retourne 0 si la fermeture s'est bien passée, EOF en cas d'erreur.

Exemple : **FILE *fichier ;**

```
fichier = fopen(« c :\toto.dat », « rb ») ;
/* Ici instructions de traitement */
fclose(fichier) ;
```

IV. Les entrées-sorties sur les fichiers textes

En langage C, il est également possible d'accompagner les transferts d'information entre la mémoire et un fichier avec d'opérations de formatage analogues à celles que réalisent printf ou scanf. Les fichiers concernés par ces opérations de formatage sont appelés "fichiers de type texte" ou encore des "fichiers textes"

Ce transfert d'information entre la mémoire RAM et fichier consiste à coder le contenu de la mémoire avant en utilisant le code ASCII avant de le transférer vers le fichier. Les octets qui figurent dans le fichier ne sont pas des copies conformes de ceux qui apparaissent en mémoire, mais plutôt sont des informations codées en ASCII. De plus, chaque octet dans le fichier représente un caractère. Généralement, on y trouve des caractères de fin de ligne (\n), de sorte qu'ils apparaissent comme une suite de lignes.

IV.1 La fonction d'écriture fprintf

La syntaxe de fprintf	Exemple

La fonction fprintf, analogue à printf, permet d'écrire des données dans un fichier. Sa syntaxe est :
fprintf(f,"chaîne de contrôle",expression-1, ..., expression-n)
où 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.
	fprintf(fichier,"%s","il fait beau");
	fprintf(fichier, "%d",n);
	fprintf(fichier,"%s%d","il fait beau",n);

Exemple : Création d'un fichier texte en utilisant fprintf

Programme 1

```
main()
{
char nomfich[21] ;
int i ;
FILE * sortie ;
printf ("nom du fichier à créer : ") ;
scanf ("%s",nomfich) ;
sortie=fopen(nomfich,"w") ;
For (i=0;i<=10;i++)
  fprintf(sortie, "%d\n",i);
fclose (sortie) ;
}
```

Résultat 1

```
0
1
2
3
4
5
6
7
8
9
10
```

Programme 2

```
main()
{
char nomfich[21]= ''TableMutli5'' ;
int n=5 ;
FILE * sortie ;
sortie=fopen(nomfich,"w") ;
for(i=0;i<=10;i++)
  fprintf(sortie, "%d*%d=%d\n",n,i,n*i);
fclose (sortie) ;
}
```

Résultat

```
5*0=0
5*1=5
5*2=10
5*3=15
5*4=20
5*5=25
5*6=30
5*7=35
5*8=40
5*9=45
5*10=50
```

IV.2 La fonction de lecture fscanf

La syntaxe de fscanf	**Exemple**

La fonction fscanf, analogue à scanf, permet de lire des données dans un fichier. Sa syntaxe est semblable à celle de scanf. Sa syntaxe est:
fscanf(FILE *f,"chaîne de contrôle",argument-1,...,argument-n)
où f est le descripteur de fichier retourné par fopen. Les spécifications de format sont ici les mêmes que celles de la fonction scanf**.**
	fscanf(fichier,"%s",ch);
	fscanf(fichier, "%d",&n);
	fscanf(fichier,"%s%d",ch,&n);

Exemple : Afficher le contenu d'un fichier texte avec fscanf

Programme 3

```
/*on affiche le contenu de fichier crée dans le programme 1*/
main()
{char nomfich[21] ;
int n ;
FILE * sortie ;
printf ("nom du fichier à ouvrir : ") ;
scanf ("%s",nomfich) ;
sortie=fopen(nomfich,"r") ;
while(!feof(sortie))
{ fscanf(sortie,"%d",&n);
   printf("%d\n",n);
  }
fclose(sortie);
}
```

Résultat

```
0
1
2
3
4
5
6
7
8
9
10
```

Programme 4

```
/*on affiche le contenu de fichier crée dans le programme 2*/
#include<stdio.h>
#include<conio.h>
main()
{
char nomfich[21]= ''TableMutli5'' ;
int n,i,r ;
Char c1,c2;
FILE * sortie ;
sortie=fopen(nomfich,"r") ;
while(!feof(sortie))
{
   fscanf(sortie,"%d%c%d%c%d\n",&n,&c1,&i,&c2,&r);
   printf("%d%c%d%c%d\n",&n,&c1,&i,&c2,&r);
  }
fclose(sortie);
}
```

Résultat

```
5*0=0
5*1=5
5*2=10
5*3=15
5*4=20
5*5=25
5*6=30
5*7=35
5*8=40
5*9=45
5*10=50
```

IV.3 La fonction de lecture fgets

La syntaxe de fgets	Exemple
La fonction fgets permet de lire du fichier texte f, et la met dans la chaîne de caractères ligne, max est le nombre maximum de caractères de la ligne. Fgets retourne l'adresse de la ligne lue ou NULL à la fin de fichier **char fgets(char *ligne,int Max, FILE *f)**	fscanf(ch, 80,f);

Exemple : on affiche le contenu du fichier texte crée avec le programme 2

Programme 5

```
/*on affiche le contenu de fichier crée dans le programme 2*/
#include<stdio.h>
#include<conio.h>
main()
{
char nomfich[21]= ''TableMutli5'' ;
char ligne[100];
FILE * sortie ;
sortie=fopen(nomfich,"r") ;
while(!feof(sortie))
{
   fgets(ligne,80,sortie);
   printf("%s",ligne);
  }
fclose(sortie);
}
```

IV.4 Les fonctions d'écriture et de lecture de caractères

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. Son prototype est

int fgetc(FILE* *flot*);

où 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.

La fonction fputc écrit caractère dans le flot de données :

int fputc(int caractere, FILE *flot)

Elle retourne l'entier correspondant au caractère lu (ou la constante EOF en cas d'erreur).
Il existe également deux versions optimisées des fonctions fgetc et fputc qui sont implémentées par des macros. Il s'agit respectivement de getc et putc. Leur syntaxe est similaire à celle de fgetc et fputc:

int getc(FILE* flot);

int putc(int caractere, FILE *flot)

Exemple: 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);
}
```

IV.5 Relecture d'un caractère

Il est possible de replacer un caractère dans un flot au moyen de la fonction ungetc :

int ungetc(int caractere, FILE *flot);

Cette fonction place le caractère caractere (converti en unsigned char) dans le flot flot. En particulier, si caractere est égal au dernier caractère lu dans le flot, elle annule le déplacement provoqué par la lecture précédente. Toutefois, ungetc peut être utilisée avec n'importe quel caractère (sauf EOF). Par exemple, l'exécution du programme suivant

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

int main(void)
{
  FILE *f_in;
  int c;
    if ((f_in = fopen(ENTREE,"r")) == NULL)
```

```
    {
      fprintf(stderr, "\nErreur: Impossible de lire le fichier %s\n",ENTREE);
      return(EXIT_FAILURE);
    }

  while ((c = fgetc(f_in)) != EOF)
    {
      if (c == '0')
        ungetc('.',f_in);
      putchar(c);
    }
  fclose(f_in);
  return(EXIT_SUCCESS);
}
```

sur le fichier `entree.txt` dont le contenu est `097023` affiche à l'écran `0.970.23`

V- Gestion des erreurs:

fopen retourne le pointeur NULL si erreur (Exemple: impossibilité d'ouvrir le fichier).
fgets retourne le pointeur NULL en cas d'erreur ou si la fin du fichier est atteinte.
la fonction **int feof(FILE *fichier)** retourne 0 tant que la fin du fichier n'est pas atteinte.
la fonction **int ferror(FILE *fichier)** retourne 1 si une erreur est apparue lors d'une manipulation de fichier, 0 dans le cas contraire.

VII. Les fichiers prédéfinis

Un certain nombre de fichiers sont connus du langage C, sans qu'il soit nécessaire ni de les ouvrir ni de les fermer :

- stdin : unité d'entrée (par défaut, le clavier) ;
- stdout : unité de sortie (par défaut, l'écran) ;
- stderr : unité d'affichage des messages d'erreurs (par défaut, l'écran).

Remarques:
L'instruction **fscanf(stdin, "%d\n", &N);** est identique à **scanf("%d\n", &N);**
L'instruction **fprintf(stdout, "Bonjour\n");** est identique à **printf("\Bonjour\n");**
L'instruction **fputc('a', stdout);** est identique à **putchar('a') ;**

VIII. Les entrées-sorties sur les fichiers binaires

VIII.1 Création séquentielle d'un fichier

Voici un programme qui se contente d'enregistrer séquentiellement dans un fichier une suite de nombres entiers saisis au clavier.

```
#include <stdio.h>
main()
{
char nomfich[21] ;
int n ;
FILE * sortie ;
printf ("nom du fichier à créer : ") ;
scanf ("%20s", nomfich) ;
sortie = fopen (nomfich, "w") ;
do { printf ("donnez un entier : ") ;
scanf ("%d", &n) ;
if (n) fwrite (&n, sizeof(int), 1, sortie) ;
}
while (n) ;
fclose (sortie) ;
}
```

Nous avons déclaré un tableau de caractères nomfich destiné à contenir, sous forme d'une chaîne, le nom du fichier que l'on souhaite créer.

a)*Ecriture dans le fichier:*
Le remplissage du fichier est réalisé par la répétition de l'appel :

fwrite (&n, sizeof(int), 1, sortie) ;

La fonction fwrite possède quatre arguments précisant :

- l'adresse d'un bloc d'informations (ici &n) ;
- la taille d'un bloc, en octets : ici sizeof(int) ; notez l'emploi de l'opérateur sizeof qui assure la portabilité du programme ;
- le nombre de blocs de cette taille que l'on souhaite transférer dans le fichier (içi 1) ;
- l'adresse de la structure décrivant le fichier (sortie).

Notez que, d'une manière générale, fwrite permet de transférer plusieurs blocs consécutifs de même taille à partir d'une adresse donnée

VIII.2 Liste séquentielle d'un fichier

Voici maintenant un programme qui permet de lister le contenu d'un fichier quelconque tel qu'il a pu être créé par le programme précédent.

```
#include <stdio.h>
main()
{
char nomfich[21] ;
int n ;
FILE * entree ;
printf ("nom du fichier à lister : ") ;
scanf ("%20s", nomfich) ;
entree = fopen (nomfich, "r") ;
while ( fread (&n, sizeof(int), 1, entree), ! feof(entree) )
printf ("\n%d", n) ;
fclose (entree) ;
}
```

Les déclarations sont identiques à celles du programme précédent. En revanche, on trouve cette fois, dans l'ouverture du fichier, l'indication r (abréviation de read). Elle précise que le fichier en question ne sera utilisé qu'en lecture. Il est donc nécessaire qu'il existe déjà (nous verrons un peu plus loin comment traiter convenablement le cas où il n'existe pas).

a) Lecture du fichier:
La lecture dans le fichier se fait par un appel de la fonction fread :

fread (&n, sizeof(int), 1, entree)

dont les arguments sont comparables à ceux de fwrite . Mais, cette fois, la condition d'arrêt de la boucle est :
feof (entree)
Celle-ci prend la valeur vrai (c'est-à-dire 1) lorsque la fin du fichier a été rencontrée. Notez bien qu'il n'est pas suffisant d'avoir lu le dernier octet du fichier pour que cette condition prenne la valeur vrai.

b) Remarques:

1) On pourrait remplacer la boucle while par la construction (moins concise) suivante :

```
do
{ fread (&n, sizeof(int), 1, entree) ;
if ( !feof(entree) ) printf ("\n%d", n) ;
}
while ( !feof(entree) ) ;
```

2) N'oubliez pas que le premier argument des fonctions fwrite et fread est une adresse. Ainsi, lorsque vous aurez affaire à un tableau, il faudra utiliser simplement son nom (sans le faire précéder de &), tandis qu'avec une structure il faudra effectivement utiliser l'opérateur & pour en obtenir l'adresse. Dans ce dernier cas, même si l'on ne cherche pas à rendre son programme portable, il sera préférable

d'utiliser l'opérateur sizeof pour déterminer avec certitude la taille des blocs correspondants.

3) fread fournit le nombre de blocs effectivement lus (et non pas le nombre d'octets lus). Ce résultat peut être inférieur au nombre de blocs demandés soit lorsque l'on a rencontré une fin de fichier, soit lorsqu'une erreur de lecture est apparue. Dans notre précédent exemple d'exécution, fread fournit toujours 1, sauf la dernière fois où elle fournit 0.

VIII.3 L'accès direct

Les fonctions fread et fwrite lisent ou écrivent un certain nombre d'octets dans un fichier, à partir d'une position courante. Cette dernière n'est rien d'autre qu'un « pointeur » dans le fichier, c'est-à-dire un nombre précisant le rang du prochain octet à lire ou à écrire. (Le terme de « pointeur » n'a pas exactement le même sens que celui de pointeur tel qu'il apparaît en langage C. En effet, il ne désigne pas, à proprement parler, une adresse en mémoire, mais un emplacement dans un fichier. Pour éviter des confusions, nous parlerons de « pointeur de fichier »).

Mais il est également possible d'agir directement sur ce pointeur de fichier à l'aide de la fonction fseek . Cela permet ainsi de réaliser des lectures ou des écritures en n'importe quel point du fichier, sans avoir besoin de parcourir toutes les informations qui précèdent. On peut ainsi réaliser ce que l'on nomme généralement un « accès direct ».

a) Accès direct en lecture sur un fichier existant

```
#include <stdio.h>
main()
{ char nomfich[21] ;
int n ;
long num ;
FILE * entree ;
printf ("nom du fichier à consulter : ") ;
scanf ("%20s", nomfich) ;
entree = fopen (nomfich, "r") ;
while(printf (" numéro de l'entier recherché : "),scanf ("%ld", &num), num )
{ fseek (entree, sizeof(int)*(num-1), SEEK_SET) ;
fread (&n, sizeof(int), 1, entree) ;
printf (" valeur : %d \n", n) ;
}
fclose (entree) ;
}
```

La principale nouveauté de ce code réside essentiellement dans l'appel de la fonction fseek :

fseek (entree, sizeof(int)*(num-1), SEEK_SET) ;

Cette dernière possède trois arguments :

- le fichier concerné (désigné par le pointeur sur une structure de type FILE , tel qu'il a été fourni par fopen) ;
- un entier de type long spécifiant la valeur que l'on souhaite donner au pointeur de fichier.

Il faut noter que l'on dispose de trois manières d'agir effectivement sur le pointeur, le choix entre les trois étant fait par l'argument suivant ;

- le choix du mode d'action sur le pointeur de fichier : il est défini par une constante entière.

Les valeurs suivantes sont prédéfinies dans <stdio.h> :
SEEK_SET (égale à 0): il désigne un déplacement (en octets) depuis le début du fichier
SEEK_CUR (égale à 1): il désigne un déplacement exprimé à partir de la position courante
SEEK_END (égale à 2): il désigne un déplacement depuis la fin du fichier.