Les fichiers : Généralités

.Le C (comme d'autres langages) offre la possibilité de lire et d'écrire des données dans un fichier. Les accès à un fichier se font par l'intermédiaire d'une mémoire-tampon (*buffer*).

.Pour pouvoir manipuler un fichier, un programme a besoin d'un certain nombre d'informations :

.l'adresse de l'endroit de la mémoire-tampon où se trouve le fichier, la position de la tête de lecture, le mode d'accès au fichier (lecture ou écriture) Ces informations sont rassemblées dans une structure dont le type, `FILE*`, est défini dans `stdio.h`.

.Un objet de type `FILE *` est appelé *flot de données* (stream).

.Avant de lire ou d'écrire dans un fichier, on notifie son accès par la commande `fopen` qui prend comme argument le nom du fichier, négocie avec le système d'exploitation et initialise un flot de données, qui sera ensuite utilisé lors de l'écriture `fprintf/fputc/fwrite` ou de la lecture `fscanf/fgetc/fread` .

. Après les traitements, on annule la liaison entre le fichier et le flot de données grâce à la fonction `fclose`.

Les fichiers : Ouverture et Fermeture

. `FILE*`, est défini dans `stdio.h`.

. `FILE *fopen(const char *, const char *);`

. `int fclose(FILE *);`

```
#define nom_fic "le_nom"
#define mode "XXX" /* XXX voir page suivante */
…
file * f;
…
f = fopen(nom_fic ,mode);
/* Lecture ou/et écriture de f  */
fclose(f);
```


`"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+"`	
`"a+"`	ouverture d'un fichier texte en lecture/écriture à la fin
`"r+b"`	ouverture d'un fichier binaire en lecture/écriture
`"w+b"`	
`"a+b"`	ouverture d'un fichier binaire en lecture/écriture à la fin


```
#include <stdio.h>
#include <stdlib.h>
#define fic "fic.txt"
int main(){
   FILE *f;
   f=fopen(fic,"w");
   fprintf(f,"%s ,%d ; %f","bonjour", 1001, 17.52);
   fclose(f);
   if ((f=fopen(fic,"r"))!= NULL){
    char s[10];int n;float x;
    fscanf(f,"%s  ,%d;%f",s,&n,&x);
    printf("%s/%d/%f",s,n,x);
    fclose(f);
   }
  return EXIT_SUCCESS;
}
```

Contenu du fichier ***fic.txt*** :
bonjour , 1001 ; 17.520000
Ce fichier est écrasé à chaque exécution

Ecriture

Lecture

•Les mêmes symboles de séparation:
•1 espace au moins en écriture
•Pas forcément le même nombre d'espaces en écriture qu'en lecture(0)

Affichage :
Bonjour/1001/17.520000

fprintf et fscanf sont analogue à printf et scanf
Ils ont un paramètre supplémentaire de type `FILE *`
Trois flots standard peuvent être utilisés en 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).

```
#include <stdio.h>
#include <stdlib.h>

int main(void){
  printf("%s monsieur\n","bonjour");
  fprintf(stdout,"%s monsieur\n","bonjour");
  return(EXIT_SUCCESS);
}
```



```
#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);        }
```

Les fichiers : entrées-sorties binaires

.Les fonctions d'entrées-sorties binaires permettent de transférer des données dans un fichier sans transcodage.

.Elles sont donc plus efficaces que les fonctions d'entrée-sortie standard, mais les fichiers produits ne sont pas portables puisque le codage des données dépend des machines.

Elles sont notamment utiles pour manipuler des données de grande taille ou ayant un type composé

Leurs prototypes sont :
size_t fread(void *pointeur, size_t *taille*, size_t *nombre*, FILE *flot);
size_t fwrite(void *pointeur, size_t *taille*, size_t *nombre*, FILE *flot);

où *pointeur* est l'adresse du début des données à transférer, *taille* la taille des objets à transférer, *nombre* leur nombre.
Rappelons que le type size_t, défini dans stddef.h, correspond au type du résultat de l'évaluation de sizeof. Il s'agit du plus grand type entier non signé.


```c
#include <stdio.h>
#include <stdlib.h>

#define NB 50
#define F_SORTIE "sortie"

int main(void){
 FILE *f_in, *f_out;
 int *tab1, *tab2;
 int i;

 tab1 = (int*)malloc(NB * sizeof(int));
 tab2 = (int*)malloc(NB * sizeof(int));

 for (i = 0 ; i < NB; i++)     tab1[i] = i;

 f_out = fopen(F_SORTIE, "w"); /* il manque un test*/
 fwrite(tab1, NB * sizeof(int), 1, f_out);
 fclose(f_out);

 f_in = fopen(F_SORTIE, "r");  /* il manque un test*/
 fread(tab2, NB * sizeof(int), 1, f_in);
 fclose(f_in);

 for (i = 0 ; i < NB; i++)    printf("%4d",tab2[i]);
 printf("\n");

 return(EXIT_SUCCESS);    }
```


Les fichiers : entrées-sorties binaires

Fichier ***sortie*** (Hexedit)

```
00000000    00 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00    ................
00000010    04 00 00 00 05 00 00 00 06 00 00 00 07 00 00 00    ................
00000020    08 00 00 00 09 00 00 00 0A 00 00 00 0B 00 00 00    ................
00000030    0C 00 00 00 0D 00 00 00 0E 00 00 00 0F 00 00 00    ................
00000040    10 00 00 00 11 00 00 00 12 00 00 00 13 00 00 00    ................
00000050    14 00 00 00 15 00 00 00 16 00 00 00 17 00 00 00    ................
00000060    18 00 00 00 19 00 00 00 1A 00 00 00 1B 00 00 00    ................
00000070    1C 00 00 00 1D 00 00 00 1E 00 00 00 1F 00 00 00    ................
00000080    20 00 00 00 21 00 00 00 22 00 00 00 23 00 00 00     ...!...".../#...
00000090    24 00 00 00 25 00 00 00 26 00 00 00 27 00 00 00    $...%...&...'...
000000A0    28 00 00 00 29 00 00 00 2A 00 00 00 2B 00 00 00    (...)...*...+...
000000B0    2C 00 00 00 2D 00 00 00 2E 00 00 00 2F 00 00 00    ,...-......./...
000000C0    30 00 00 00 31 00 00 00 20                         0...1...
```


Positionnement dans un fichier

.Les différentes fonctions d'entrées-sorties permettent d'accéder à un fichier en *mode séquentiel* : les données du fichier sont lues ou écrites les unes à la suite des autres. Il est également possible d'accéder à un fichier en *mode direct*, c'est-à-dire que l'on peut se positionner à n'importe quel endroit du fichier. La fonction fseek permet de se positionner à un endroit précis ; elle a pour prototype : int fseek(FILE *flot, long *deplacement*, int *origine*); La variable *deplacement* détermine la nouvelle position dans le fichier. Il s'agit d'un déplacement relatif par rapport à l'origine ; il est compté en nombre d'octets. La variable *origine* peut prendre trois valeurs :

.SEEK_SET (égale à 0) : début du fichier ;

.SEEK_CUR (égale à 1) : position courante ;

.SEEK_END (égale à 2) : fin du fichier.

.La fonction int rewind(FILE *flot); permet de se positionner au début du fichier. Elle est équivalente à
fseek(*flot*, 0, SEEK_SET);

La fonction long ftell(FILE *flot); retourne la position courante dans le fichier (en nombre d'octets depuis l'origine).


```c
#include <stdio.h>
#include <stdlib.h>

#define NB 50
#define F_SORTIE "sortie"

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

 tab = (int*)malloc(NB * sizeof(int));
 for (i = 0 ; i < NB; i++)
   tab[i] = i;

if ((f_out = fopen(F_SORTIE, "w")) == NULL)    {
    fprintf(stderr, "\nImpossible d'ecrire dans %s\n", F_SORTIE);
    return(EXIT_FAILURE);    }
 fwrite(tab, NB * sizeof(int), 1, f_out);
 fclose(f_out);

if ((f_in = fopen(F_SORTIE, "r")) == NULL)    {
    fprintf(stderr, "\nImpossible de lire dans %s\n", F_SORTIE);
    return(EXIT_FAILURE);
  } /* … */
```



```c
/* … */

/* on se positionne a la fin du fichier */
 fseek(f_in, 0, SEEK_END);
 printf("\n position %ld", ftell(f_in));
 /* deplacement de 10 int en arriere */
 fseek(f_in, -10 * sizeof(int), SEEK_END);
 printf("\n position %ld", ftell(f_in));
 fread(&i, sizeof(i), 1, f_in);
 printf("\t i = %d", i);
 /* retour au debut du fichier */
 rewind(f_in);
 printf("\n position %ld", ftell(f_in));
 fread(&i, sizeof(i), 1, f_in);
 printf("\t i = %d", i);
 /* deplacement de 5 int en avant */
 fseek(f_in, 5 * sizeof(int), SEEK_CUR);
 printf("\n position %ld", ftell(f_in));
 fread(&i, sizeof(i), 1, f_in);
 printf("\t i = %d\n", i);
 fclose(f_in);
 return(EXIT_SUCCESS);
}
```

.L'exécution de ce programme affiche à l'écran :

position 200
position 160 i = 40
position 0 i = 0
position 24 i = 6

On constate en particulier que l'emploi de la fonction fread provoque un déplacement correspondant à la taille de l'objet lu à partir de la position courante.