Gestion des Fichiers
en language C

Les fichiers orienté utilisateurs (texte)

Déclaration d'un fichier

le type d'un fichier est le type FILE et on demande toujours un pointeur sur ce type.
Nous déclarerons donc de cette façon :

```
FILE * flux;
```

REM : FILE est TOUJOURS en majuscule.
Nous utiliserons la variable flux dans la suite des exemples, on peux bien sur la nommer différemment...

Ouverture du fichier

```
flux = fopen ("nom_du_fichier.dat", "mode");
```

Les modes d'ouvertures des fichiers textes

MODE	*Explication*	*Le fichier*
r rt	Lecture du # texte	Existe : OK Existe pas : KO
w wt	Ecriture du # texte	Existe : Ecrase Existe pas : OK - Créé
a at	Ecriture à la fin	Existe : OK Existe pas : OK - Créé

Fermeture du Fichier

```
fclose( flux );
```

Ecriture

int fprintf(flux, "formatage", variable);

exple : fprintf(flux, "%d\n",compteur);

REM : La valeure de retour est le nombre de caractères imprimés. En plus du flux à afficher, la fonction ressemble très fortement à printf().

Lecture

int fscanf(flux,"formatage", &variable);

exple : fscanf(flux, "%d", &age);

REM : La valeure de retour est le nombre d'information lue. En plus du flux à afficher, la fonction ressemble très fortement à scanf().

int fgets(char * chaine, int longueur, FILE * flux);

fgets lit (longueur) caractères et ajoute le '\0' à la fin.
La valeure de retour est NULL si il y a un echec en lecture.
ATTENTION : à la rencontre de '\n', il y arrêt, maiscet '\n' est copié comme un autre caractère dans la chaine.

Exple :	char chaine[10]; fgets(chaine, 10, flux);	Si l'enregistrement est : « Ath\n » 'chaine' sera composé de : Ath\n\0????
Astuce :	on met l'\0 à la place de l'\n par une commande facile à comprendre. chaine [strlen(chaine) – 1] = '\0';	

Explication : strlen() (#include <string.h>) retourne la taille d'une chaine de caractère jusqu'à l'\0. Si l'on en soustrait 1, on pointe sur l' '\n' que l'on peut remplacer par un '\0'. Si je met l' '\0' à l'endroit voulu avec un égal c'est car il est composé de 1 caractère. Pour une chaine de caractère on utilisera la fonction strcpy( ) (#include <string.h>) ou la fonction strncpy( ) (#include <string.h>) qui ne s'impose pas ici...

Fin de Fichier

int feof(flux);

utilisation dans une boucle :

```
lecture initiale;
while( ! feof(flux) )
{
        lecture courante;
}
```

Exemple

```
#include <stdio.h>

void main()
{
    int donnee;
    int i = 0;
    FILE * flux_entree;
    FILE * flux_sortie;
    flux_entree = fopen("entree.dat", "r");   // entree.dat est composé d'un entier
                                              // par enregistrement
    flux_sortie = fopen("sortie.dat", "w");

    fscanf(flux_entree, "%d", &donnee);       // lecture initiale

    while ( ! feof(flux_entree) )
    {
        i++;
        // ecriture
        fprintf(flux_sortie, "%d ieme enregistrement : %d", i, donnee);
        // lecture courante
        fscanf(flux_entree, "%d", &donnee);
    } // end_while
    fclose(flux_entree);
    fclose(flux_sortie);
}
```

Les fichiers orienté machine (binaire)

Un fichier binaire est une copie conforme de la mémoire centrale. Les informations numériques sont érites en hexadécimal et le retour a la ligne prend une forme particulière (ce n'est pas 2 caractères comme en mode texte).

Déclaration d'un fichier

le type d'un fichier est le type FILE et on demande toujours un pointeur sur ce type. Nous déclarerons donc de cette façon :

```
FILE * flux;
```

REM : FILE est TOUJOURS en majuscule.
Nous utiliserons la variable flux dans la suite des exemples, on peux bien sur la nommer différemment...

Ouverture du fichier

```
flux = fopen ("nom_du_fichier.dat", "mode");
```

Les modes d'ouvertures des fichiers textes

MODE	*Explication*	*Le fichier*
rb	Lecture du # binaire	Existe : OK Existe pas : KO
wb	Ecriture du # binaire	Existe : ecrase Existe pas : OK - Créé
ab	Ecriture à la fin	Existe : OK Existe pas : OK - Créé

Pourquoi mettre un 'b' alors que les fonctions fprintf et fscanf fonctionnent?
Lors d'une écriture / lecture dans un fichier binaire, il n'y a pas de changement par rapport à la mémoire centrale. Tandis que dans un fichier texte, des changement sont effectué entre autre pour le retour chariot '\n'.

Fermeture du Fichier

fclose(flux);

Ecriture

fwrite(&variable, sizeof(variable), 1, flux);

exple : fwrite(&structure, sizeof(type_de_structure), 1, flux);

Astuce pour écrire un tableau :
fwrite(&table, sizeof(&table[0]) * DIMENSION_TABLE, 1, flux);
OU de façon équivalente
fwrite(&table, sizeof(&table[0]), DIMENSION_TABLE, flux);

Lecture

fread(&variable, sizeof(type_de_variable), 1, flux);

exple : fread(&structure, sizeof(struct structure), 1, flux);
L'astuce pour la lecture est identique à l'écriture.

Fin de Fichier

int feof(flux);

utilisation dans une boucle :

```
lecture initiale;
while( ! feof(flux) )
{
        lecture courante;
}
```

Exemple

```
#include <stdio.h>

void main()
{
    int donnee;
    int i = 0;
    FILE * flux_entree;
    FILE * flux_sortie;
    flux_entree = fopen("entree.dat", "rb"); // entree.dat est composé d'un entier
                                             // par enregistrement
    flux_sortie = fopen("sortie.dat", "wb");

    fread(&donnee, sizeof(donnee), 1, flux); // lecture initiale

    while ( ! feof(flux_entree) )
    {
        i++;
        // ecriture
        fwrite(&donnee, sizeof(donnee), 1, flux);
        // lecture courante
        fread(&donnee, sizeof(donnee), 1, flux);
    } // end_while
    fclose(flux_entree);
    fclose(flux_sortie);
}
```

... Astuce de programmation ...

Nous pouvons aussi jouer sur les valeures de retour pour une programmation plus rapide.
Mais comme un dessin vaut mieux qu'un grand discours...

```
#include <stdio.h>
#define FICHIER "entree.dat"
struct record
            {
            char nom[30];
            char prenom[30];
            int age;
            }
void main( )
{
    FILE * flux;                    // déclaration du flux
    struct record entree;           // entree est de type structure record

    if ( ( flux = fopen(FICHIER, "r") ) == NULL )
    {
            printf("Erreur d'ouverture du fichier %s\n", FICHIER);
    }// end_if
    while ( fscanf(flux, "%s %s %d", entree.nom, entree.prenom, &entree.age) == 3 )
    {
            printf("NOM : %s\nPRENOM : %s\nAGE : %d\n\n", entree.nom,
                                                    entree.prenom, entree.age);
    }// end_while

    // verification d'erreur en lecture
    if ( ! feof(flux) )
    {
            printf("Erreur de lecture sur : %s\n", FICHIER);
    } // end_if

    if ( fclose(flux)  ) // fermeture du fichier
    {
            printf("Erreur de fermeture sur : %s\n", FICHIER);
    } // end_if
} // end_main
```

L'accès direct

Le principe de l'accès direct est proche des fichiers organisés relativement. Il agit sur les pointeurs de fichiers. Son avantage est que la rapidité augmente car le nombre de lecture diminue...

Le fichier doit être séquentiel (ce n'est pas un gros problème en C) et de préférence de longueur fixe.

Les fonctions utilisées :

ftell( ) : Le compteur

long int ftell(FILE * flux);

La valeure de retour (long int) est égal au nombre de bytes qui sépare le pointeur de fichier au début de celui-ci. Il prend une valeure de '-1' si une erreur a été constatée.

fseek( ) : Le positionneur

int fseek(flux, déplacement, origine);

La valeure de retour est un entier qui est NULL si le positionnement s'est bien effectué est une autre valeure si une erreur s'est produit. Malheureusement plusieurs compilateurs ne l'entendent pas comme cela... (fonctionne avec Visual C++ mais je ne sais pas actuellement avec Borland)

Les origines

SEEK_SET	Le pointeur compte à partir du début du fichier. Si le déplacement est null, il sera au début du fichier prêt à lire le premier enregistrement.
SEEK_END	Calcul le déplacement par rapport à la fin du fichier.
SEEK_SET	Calcul le déplacement par parrort à la position courante du pointeur de fichier

Les déplacements

Les déplacements obligent certaines vérifications pour que la programation soit propre (bien qu'il n'y ai aucune obligation).

Il faut vérifier que l'origine ajouté au déplacement correspondent bien au contenu du fichier.

0 origine + déplacement Taille du fichier

La taille du fichier :

```
fseek (flux, 0, SEEK_END);
long int Taille_du_fichier = ftell(flux);
```

Le pointeur de fichier se place à la toute fin du fichier.
On calcul le déplacement du pointeur de fichier par rapport au début du fichier.

Exemple

Nous avons un fichier contenant tous les noms des étudiants de l'IRAM. Il est trié alphabétiquement. Nous faisons une recherche dichotomique dans ce fichier (vu en fichier en première avec Mr Pingaut !). Il est bien entendu que c'est un fichier avec des enregistrements à longueurs fixes (50 caractères + '\n')...

```
#include <stdio.h>      // toutes les entrées sorties
#include <string.h>     // Pour les fonctions : strlen( ) , strncmp()
#include FICHIER    "fichier.dat"
void main ( )
{
    FILE * flux;
    char nom;
    char nom_lu[51];      // Le nom recherché
    int longueur;         // longueur du nom recherché
    int borne_inf;
    int borne_sup;
    int borne_med;        // borne médiane
    int nbr_record;       // nombre total d'enregistrements
    int deplacement;
    // Ouverture du fichier
    //-----------------------------
        flux = fopen( FICHIER, "rt");
```

```
    // Initialisation des variables
    //---------------------------------
    // Calcul du nombre d'enregistrements
    fseek ( flux, 0, SEEK_END );                    // positionnement à la fin
    nbr_record = ftell ( flux ) / sizeof(nom_lu);  // Taille totale / taille particulière

    borne_inf = 0 ;
    borne_sup = nbr_record – 1 ;

    printf ( "Nom recherché : ");
    scanf( "%s", nom_lu);
    longueur = strlen ( nom_lu );

    while (borne_inf <= borne_sup )
    {
          borne_med = (borne_inf + borne_sup) / 2;
          deplacement =  sizeof(nom_lu) * borne_med;  // calcul du déplacement
          fseek ( flux, deplacement, SEEK_SET);         // positionnement
          fscanf(flux, "%s", nom);
          if ( strncmp(nom_lu, nom, longueur) >= 0)
          {
                borne_sup = borne_med – 1;
          }
          else
          {
                borne_inf = borne_med + 1;
          } // end_if
    } // end_while

    // Affichage des noms
    //------------------------
    while ( fscanf ( flux, "%s", nom) == 1 &&  strncmp ( nom, nom_lu, longueur ) == 0  )
    {
           printf(" %s \n", nom);
    }// end_while

    fclose (flux);       // fermeture du fichier

} // end_main
```

Remarquer la présence de la mémoire tampon

Mettez une disquette dans le lecteur, recopier ce programme qui s'appelle 'afac.c' et executez... Regardez bien la lampe d'écriture/lecture du lecteur de disquette et tapez sur <ENTER> tous les 15 caractères.

Le programme

```
#include <stdio.h>
void main()
{    FILE * flux_entree;
     FILE * flux_sortie;
     char c[80];
     flux_sortie = fopen("A:/text.c", "wt");    // sortie sur une disquette !
     flux_entree = fopen("afac.c", "rt");
     while( fscanf(flux_entree, "%15c", c) == 1)
     {
              fprintf(flux_sortie,"%s", c);
              printf("%s", c);
              fgetc(stdin);
     }// end_while
     fclose(flux_sortie);
     fclose(flux_entree);
} // end _main
```

Explications

En réalité, le programme écrit dans le tampon et lorsqu'il est plein ou lorsque l'on ferme le fichier, il écrit réellement sur le support !
Selon les machines et les compilateurs, cette mémoire tampon a une taille différente. Chez moi, l'ecriture se fait à la fin (fermeture du fichier) car le fichier lu n'est pas assez grand que pour être écrit sur la disquette en plusieurs fois... ou c'est ma mémoire tampon qui est trop grande ? ; o)

!!!! C'est bête, mais il fallait y penser !!!!
l'idée n'est malheureusement pas de moi !

Les fichiers prédéfinis

Lorsque l'on affiche à l'écran, il s'agit en réalité d'un fichier prédéfini.

Entree par défaut : stdin

L'entree par défaut est généralement le clavier et est reconnu par le programme comme un fichier. C'est le système d'exploitation (OS) qui gère la lecture au clavier càd que l'on a plus en tant que programmeur à s'en occuper. (Pour info, c'est semblable en Fortran)
Les deux phrases suivantes sont parfaitement équivalente.

```
scanf("%d", &valeur);
fscanf(stdin, "%d", &valeur);
```

Sortie par défaut : stdout

La sortie par défaut est généralement l'écran et est reconnu par le programme comme un fichier. C'est le système d'exploitation (OS) qui execute l'affichage à l'écran.
Les deux phrases suivantes sont parfaitement équivalente.

```
printf("%d", valeur);
fprintf(stdout, "%d", valeur);
```

Sortie imprimante : stdprt ou stdprn

Imprimer un document c'est comme écrire dans un fichier. Sauf que c'est un fichier particulier. C'est le fichier dont le nom est stdprt (cours de Tonton) ou stdprn (MicroApp.). Mais ce n'est pas une sortie standard. Cela dépend des compilateurs. (Visual C++ 97 ne le connait pas chez moi. Si cela fonctionne chez vous, pouvez-vous me le dire. Merci !)

```
fprintf(stdprn, "%d", valeur);
```