



# Les fichiers

**Rajae El Ouazzani**



# Chapitre 5: Les fichiers



# Plan:

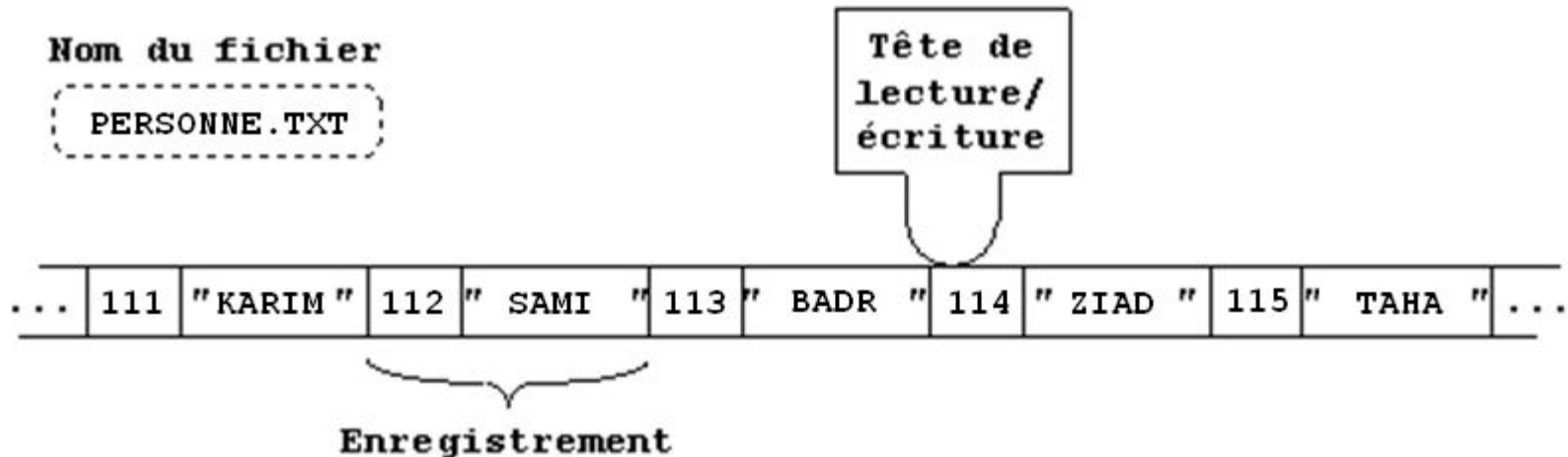
- Introduction
- Types d'accès aux fichiers
- Les entrées formatées/fichiers séquentiels
- Exemple d'application
- Les entrées non formatés/fichiers binaires
- Mise à jour des fichiers séquentiels
- Déplacements dans les fichiers



# Introduction

# Définition

- Un fichier est un support de stockage de données. Ces données sont stockées sous forme d'enregistrements.
- Ces enregistrements sont mémorisés consécutivement dans l'ordre de leur entrée.
- On peut imaginer le fichier séquentiel **PERSONNE.TXT** de champs *num* et *nom*, enregistré sur la mémoire de la manière suivante:



# Les types de fichiers

- Il existe 2 types de fichiers :
  - les fichiers textes: se sont des fichiers lisibles par un simple éditeur de texte (word, bloc notes, etc.).
  - les fichiers binaires: se sont des fichiers dont les données correspondent en général à une copie bit à bit du contenu de la RAM. Ils ne sont pas lisibles avec les éditeurs de texte.



# Types d'accès aux fichiers

- On distingue deux manières d'accès aux informations d'un fichier :
  - accès séquentiel (le plus courant) : on lit/écrit dans l'ordre dans lequel les informations apparaissent.
  - accès direct : on lit/écrit dans un endroit particulier du fichier; dont les enregistrements sont indexés.

accès aux données	séquentiel	direct
	lire/écrire dans l'ordre	enregistrements indexés

**NB:**

- Par défaut l'accès est séquentiel.
- L'accès direct se fait au moyen de fonctions fseek, ftell, etc.



# Accès aux données des fichiers

- Avant de lire ou d'écrire dans un fichier, il faut l'ouvrir avec la fonction `fopen`.
- A la fin des traitements sur le fichier, il faut le fermer avec la fonction `fclose`.
- Ainsi, le fichier est ouvert entre les commandes `fopen()` et `fclose()`.

# Ouvrir un fichier

```
p_fichier=fopen(nom_fichier,"mode");
```

- **fopen** est une fonction qui:
  - demande en entrée le nom du fichier **nom\_fichier** et le **mode** d'accès au fichier,
  - et retourne l'adresse mémoire du fichier. Cette adresse est stockée dans **p\_fichier**.
- Ensuite, on utilise **p\_fichier** au lieu de **nom\_fichier** pour référencer le fichier.
- **nom\_fichier** est une chaîne de caractères qui représente le nom du fichier avec son chemin d'accès.
- Le **mode** d'accès au fichier peut être:
  - "r": pour l'ouvrir en lecture (read), erreur si le fichier n'existe pas.
  - "w": pour l'ouvrir en écriture (write), le fichier est créé s'il n'existe pas, écrasé s'il existe.
  - etc.

# Exemple:

- Ouvrir un fichier en mode écriture.

```
char nom_fichier[30];
```

```
FILE *p_fichier;      /* variable de type pointeur sur un fichier */
```

```
/* Créer le fichier */
```

```
printf("Entrer le nom du fichier (avec l'extension) ");
```

```
scanf("%s", nom_fichier);
```

```
p_fichier = fopen(nom_fichier, "w");    /* "w"=write */
```

# Modes d'accès aux fichiers

Le C dispose de nombreuses options :

- **r** : ouvrir un fichier texte en lecture. Erreur si le fichier n'existe pas.
- **w** : ouvrir un fichier texte en écriture. Le fichier est créé s'il n'existe pas, écrasé s'il existe.
- **r+**, **"update"**: La différence entre r et r+ est que le fichier sera ouvert en lecture ET en écriture, le curseur est placé au début du fichier.
- **w+**, **"update"**: ouvre le fichier en écriture ET en lecture. Si le fichier n'existe pas, il sera créé, par contre s'il existe, son contenu sera écrasé. Le curseur est également positionné au début du fichier.
- **a**: **"append"**: le fichier sera ouvert en ajout (donc il écrira à la fin du fichier). Si le fichier n'existe pas, il sera créé. Le curseur est par contre positionné à la fin du fichier.
- **a+**: Le fichier sera ouvert en ajout (donc il écrira à la fin du fichier) ET en lecture. Si le fichier n'existait pas, la fonction le crée.

# Suite

- **rb** : ouverture d'un fichier binaire en lecture. Erreur si le fichier n'existe pas.
- **wb** : ouverture d'un fichier binaire en écriture. Le fichier est créé s'il n'existe pas, écrasé s'il existe.
- **ab** : ouverture d'un fichier binaire en écriture à la fin. Si le fichier n'existe pas, il sera créé.
- **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.

# Le résultat de fopen

- Si le fichier est ouvert avec succès, **fopen** renvoie l'adresse mémoire d'une variable de type FILE.
- Sinon, **fopen** fournit la valeur 0 ou NULL.
- Les causes d'un résultat nul :
  - ☐ pas de disque,
  - ☐ essai d'ouvrir un fichier sans préciser le mode d'accès: 'r' ou 'w' ou 'a+', etc.
  - ☐ essai d'écrire sur un disque protégé contre l'écriture/lecture,
  - ☐ etc.

# Fermer un fichier

`fclose(p_fichier);`

- `fclose` est une fonction qui demande en entrée le pointeur sur le fichier: `p_fichier`.
- `fclose` provoque le contraire de `fopen`. La mémoire est libérée et la liaison entre `p_fichier` et `nom_fichier` est annulée.
- Après `fclose()`, le pointeur `p_fichier` devient invalide.

Exemple:

`fclose(p_fichier);`

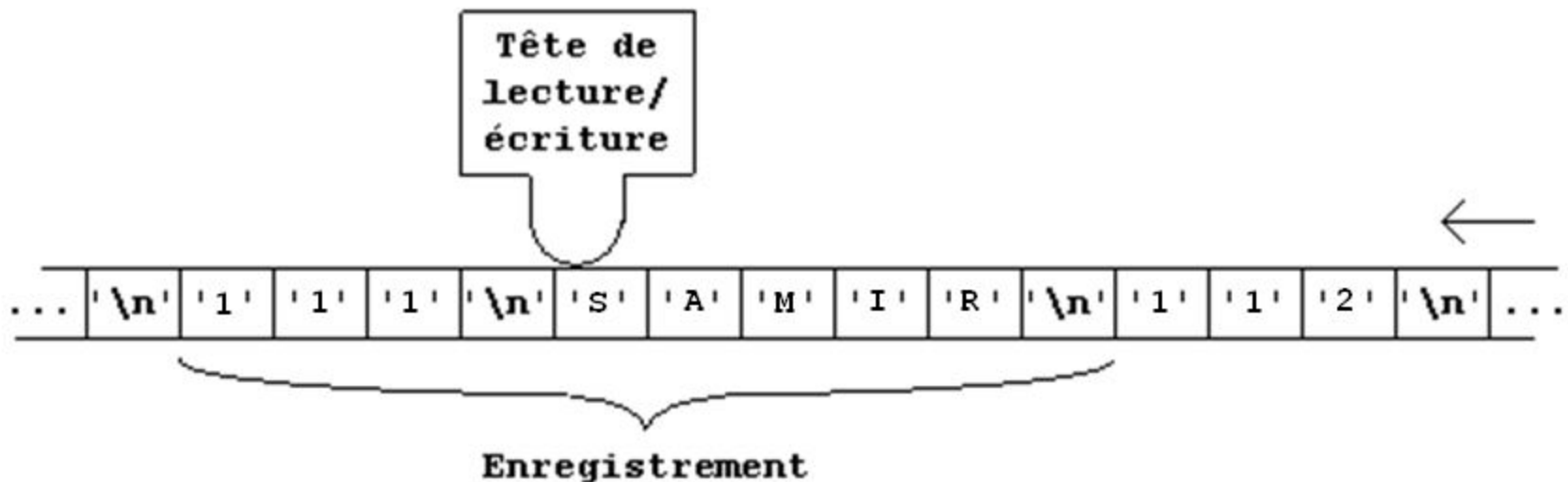


# Les entrées formatées/ fichiers séquentiels



# Organisation des données dans les fichiers

- Les fichiers textes sont généralement organisés en lignes, càd la fin d'une information dans le fichier est marquée par le symbole '\n':



- Pour lire/écrire des enregistrements dans un fichier séquentiel, on doit respecter l'ordre des champs à l'intérieur des enregistrements.
- Exemple: Commencer par le numéro puis le nom.

# Ecrire dans un fichier séquentiel : fprintf

```
fprintf( p_fichier , "Form1\n", Expr1);  
fprintf( p_fichier , "Form2\n", Expr2);  
...  
fprintf( p_fichier, "FormN\n", ExprN);
```

■ ou bien :

```
fprintf(p_fichier,"Form1\n Form2\n ...\nFormN\n", Expr1, Expr2,..., ExprN);
```

- Le pointeur `p_fichier` est relié au fichier avec l'instruction `fopen`.
- `Expr1, ..., ExprN` représentent les valeurs qu'on veut écrire dans le fichier. Elles correspondent aux champs d'un enregistrement.  
Exemple: les champs `numéro` et `nom`.
- `Form1, .., FormN` représentent les formats des champs.  
Exemple: les formats `%d` et `%s` pour les champs `numéro` et `nom`.

# Exemple:

- On souhaite créer un fichier de personnes (numéro et nom).

```
char nom_fichier[30];
```

```
FILE *p_fichier;      /* variable de type pointeur sur un fichier */
```

```
int num_pers; char nom_pers[30];
```

```
/* Créer le fichier */
```

```
printf("Entrer le nom du fichier (avec l'extension)");
```

```
scanf("%s", nom_fichier);
```

```
p_fichier = fopen(nom_fichier, "w");
```

```
printf("Entrez le numéro et le nom de la personne : ");
```

```
scanf("%d%s", &num_pers, nom_pers);
```

```
fprintf(p_fichier, "%d\n%s\n", num_pers, nom_pers);
```

# Lire à partir d'un fichier séquentiel : fscanf

```
fscanf( p_fichier, "Form1\n", Adr1);  
fscanf( p_fichier, "Form2\n", Adr2);  
...  
fscanf( p_fichier , "FormN\n", AdrN);
```

■ ou bien :

```
fscanf(p_fichier,"Form1\n Form2\n ...\n FormN\n", Adr1, Adr2, ..., AdrN);
```

- Le pointeur `p_fichier` est relié au fichier par l'instruction `fopen`.
- `Adr1, ..., AdrN` représentent les adresses mémoires des variables.
- `Form1, ... , FormN` représentent les formats des champs : `%d`, `%f`, `%s`, `%c`, etc.

# Exemple

- On souhaite lire le **num** et le **nom** de la personne à partir du fichier et les afficher sur la console.

```
char nom_fichier[30];  
FILE *p_fichier;      /* variable de type pointeur sur un fichier */  
int num_pers; char nom_pers[30];  
  
/* Ouvrir le fichier en mode lecture */  
printf("Entrer le nom du fichier (avec l'extension)");  
scanf("%s", nom_fichier);  
p_fichier = fopen(nom_fichier, "r");  
  
fscanf(p_fichier, "%d\n%s\n", &num_pers, nom_pers);  
printf("num : %d et nom : %s\n", num_pers, nom_pers);
```

# Lire et écrire dans un fichier texte séquentiel

- Pour lire ou écrire dans un fichier, on utilise les fonctions standard `fscanf`, `fprintf`, `fgetc`, `fputc`, `fgets` et `fputs` qui correspondent à `scanf`, `printf`, `getchar`, `putchar`, `gets` et `puts` qu'on utilise dans les traitements normaux.
- `fgetc` / `fputc` : lit/écrit un seul caractère à partir d'un fichier.
- `fgets` / `fputs` : lit/écrit une chaîne de caractères à partir d'un fichier;
- `getchar` / `putchar` : lit/écrit un seul caractère à partir du clavier.
- `gets` et `puts` : lit/écrit une chaîne de caractères à partir du clavier.

# Syntaxes

`char *fgets(char *s, int size, FILE * p_fichier)`

- permet de lire `size` caractères dans la chaîne `s` à partir du flux `p_fichier`.
- La valeur de retour est l'adresse de la chaîne lue quand tout s'est bien passé, ou égale à `NULL` en cas d'erreur.

`int fputs(const char *s, FILE * p_fichier)`

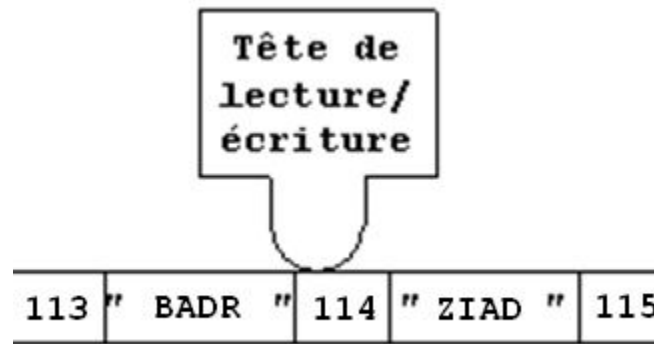
- permet d'écrire la chaîne `s` dans le flux `p_fichier`.
- La valeur de retour est différente de 0 quand tout s'est bien passé, ou égale à 0 en cas d'erreur.

# Propriétés des fichiers séquentiels

- Les fichiers ont les propriétés suivantes:

- A un moment donné, on peut uniquement accéder à un seul enregistrement; celui qui se trouve en face de la tête de lecture/écriture.

- Après chaque accès, la tête de lecture/écriture est déplacée sur l'enregistrement suivant.





# Détection de la fin d'un fichier séquentiel

- La fonction `feof` permet de détecter la fin du fichier.

`int feof(p_fichier);`

- Si `p_fichier` a atteint la fin du fichier, `feof` retourne une valeur différente de 0.
- Sinon, si `p_fichier` n'a pas encore atteint la fin du fichier, `feof` retourne 0.
- **NB:** Pour que la fonction `feof` détecte correctement la fin du fichier, il faut terminer la chaîne de format de `fscanf` par un retour à la ligne `'\n'` lors de la lecture du fichier.

# Exemple:

- Ce code lit les numéros et les noms de toutes les personnes qui se trouvent dans le fichier et les affiche sur la console.

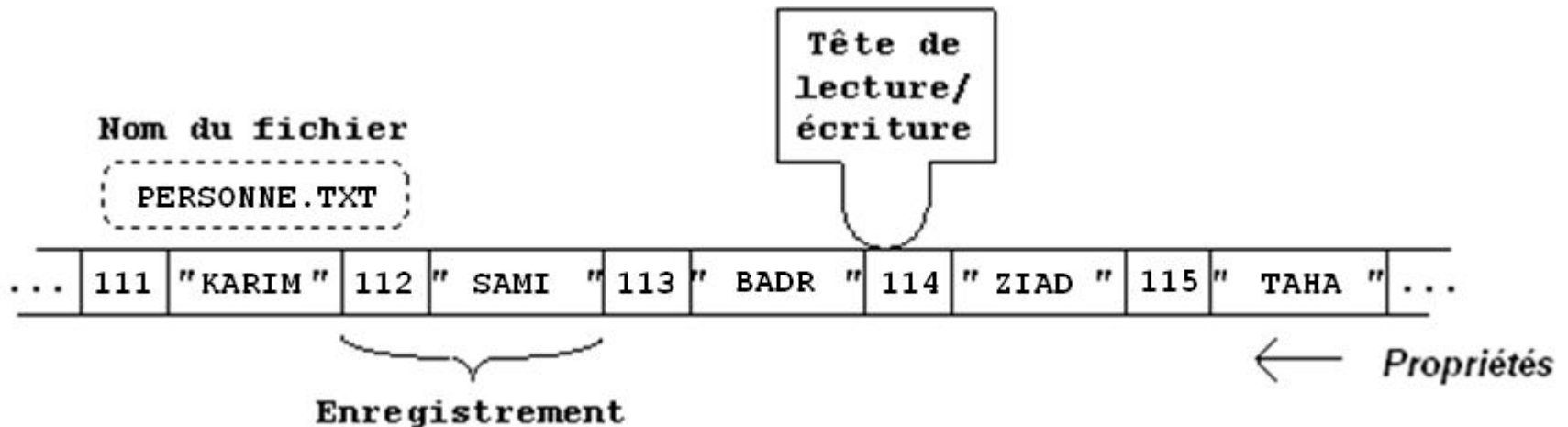
```
while (! feof (p_fichier) ) {  
    fscanf(p_fichier, "%d\n%s\n", &num_pers, nom_pers);  
    printf("num: %d et nom : %s\n", num_pers, nom_pers);  
}
```



# Exemple d'application

# Ecrire/lire avec un fichier séquentiel

- On va créer un fichier **personne.txt** qui contient les numéros et les noms de personnes.



# Etapes à suivre


- 1) Créer un fichier **PERSONNE.TXT** et l'ouvrir en mode **écriture "w" (fopen)**.
- 2) Entrer le nombre de personnes à ajouter **nb\_enreg**.
- 3) Entrer les numéros **num\_pers** et les noms **nom\_pers** des personnes un par un et les écrire dans le fichier (**fprintf**).
- 4) Fermer le fichier ouvert en mode écriture (**fclose**).
- 5) Ouvrir le même fichier **PERSONNE.TXT** en mode **lecture "r" (fopen)**.
- 6) Lire son contenu (**fscanf**) et afficher (**printf**) les personnes avec leurs numéros et leurs noms sur la console.
- 7) Fermer le fichier ouvert en mode lecture (**fclose**).

# Solution

```
#include <stdio.h>

main(){
    char nom_fichier[30];
    FILE *p_fichier;      /* variable de type pointeur sur un fichier */
    int num_pers; char nom_pers[30];
    int nb_enreg;
    int i;      /* i sert de compteur */

    /* Première partie : Créer et remplir le fichier */
    printf("le nom du fichier (avec l'extension)?? ");
    scanf("%s", nom_fichier);
    printf("Nombre d'enregistrements à créer : ");
    scanf("%d", &nb_enreg);
```



```
p_fichier = fopen(nom_fichier, "w");    /* write */
i=0;
while (i<nb_enreg) {
    printf("Entrez le numéro et le nom de la personne : ");
    scanf("%d%s", &num_pers, nom_pers);
    fprintf(p_fichier, "%d\n%s\n", num_pers, nom_pers);
    i++;
}
fclose(p_fichier);
```

**/\* Deuxième partie : Lire et afficher le contenu du fichier \*/**

```
p_fichier = fopen(nom_fichier, "r");    // Ouverture du fichier en mode lecture
while (!feof(p_fichier)) {
    fscanf(p_fichier, "%d\n%s\n", &num_pers, nom_pers);
    printf("num : %d et nom : %s\n", num_pers, nom_pers);
}
fclose(p_fichier);
return 0;
}
```



# Exercice

- Réécrire la deuxième partie du programme en utilisant la variable `nb_enreg` à la place de `feof()`.

# Solution

```
/* Deuxième partie : Lire et afficher le contenu du fichier avec nb_enreg*/  
i=0;  
while (i<nb_enreg) {  
    fscanf(p_fichier, "%d\n%s\n", &num_pers, nom_pers);  
    printf("nun : %d et nom : %s\n", num_pers, nom_pers);  
    i++;  
}
```



# **Les entrées non formatés/ fichiers binaires**

# Lecture avec fread

- Prototype :

`size_t fread(void * ptr, size_t taille, size_t nbloc, FILE * p_fichier);`

- La fonction `fread` :

- ☐ lit `nbloc` (le plus souvent 1) de taille `taille` dans `p_fichier` et les lit à partir de l'adresse `ptr`.
- ☐ retourne le nombre de blocs effectivement lu.

Il y'a une erreur si ce nombre retourné est différent de `nbloc`.  
Cette erreur peut être la rencontre de la fin du fichier.

# Ecriture avec fwrite

- Prototype :

`size_t fwrite(const void *ptr, size_t taille, size_t nbloc, FILE * p_fichier);`

- La fonction `fwrite` :

- ☐ écrit `nbloc` (le plus souvent 1) de taille `taille` situés à l'adresse `ptr` dans le flux `p_fichier`.
- ☐ retourne le nombre de blocs effectivement écrits.

Il y'a une erreur si ce nombre est différent de `nbloc`.

# Exemple

- le programme suivant écrit un tableau d'entiers (contenant les 20 premiers entiers: 0,..,19) avec **fwrite** dans le fichier **sortie.bin**, puis lit ce fichier avec **fread** et imprime les éléments du tableau sur la console.

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

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

```
#define NB 20
```


```
int main() {
```

```
    char nom_fichier[30]= "sortie.bin";
```

```
    FILE *p_fichier;
```

```
    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;
```

```
/* écriture du tableau dans le fichier */
```

```
if ((p_fichier = fopen(nom_fichier, "wb")) == NULL){
    printf("\nImpossible d'ecrire dans le fichier %s\n", nom_fichier);
    return(EXIT_FAILURE);
}
```

```
fwrite(tab1, NB * sizeof(int), 1, p_fichier);
```

```
// fwrite(tab1, sizeof(int), NB, p_fichier);
```

```
fclose(p_fichier);
```

```
/* lecture du fichier */
```

```
if ((p_fichier = fopen(nom_fichier, "rb")) == NULL) {  
    printf("\nImpossible de lire dans le fichier %s\n", nom_fichier);  
    return(EXIT_FAILURE);  
}
```

```
fread(tab2, NB * sizeof(int), 1, p_fichier);
```

```
// fread(tab2, sizeof(int), NB, p_fichier);
```

```
fclose(p_fichier);
```

```
for (i = 0 ; i < NB; i++)
```

```
    printf("%d\t", tab2[i]);
```

```
system("pause");
```

```
return(EXIT_SUCCESS);
```

```
}
```

Résultat d'exécution:

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19





# Mise à jour des fichiers séquentiels

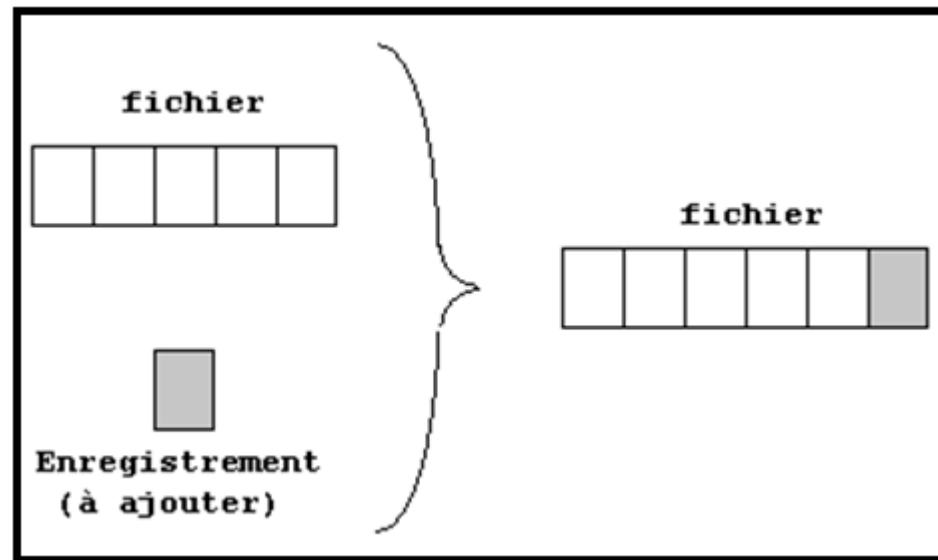


# Mise à jour d'un fichier séquentiel

- Ajouter un enregistrement dans un fichier (au début, à la fin ou au milieu).
- Supprimer un enregistrement d'un fichier.
- Modifier un enregistrement dans un fichier.


# Ajouter un enregistrement à la fin du fichier

- Par défaut, lorsqu'on ajoute un enregistrement dans un fichier, il s'ajoute à la fin.
- On ouvre le fichier en mode "a".
- On ajoute le nouveau enregistrement.



# Exemple:

```
#include <stdio.h>
#include <stdlib.h>
main() {
    char nom_fichier[30];
    FILE *p_fichier;
    int num_ajout; char nom_ajout[30];
    /* ouverture du fichier en mode ajout */
    do {
        printf("nom du fichier : ");
        scanf("%s", nom_fichier);
        p_fichier = fopen(nom_fichier, "a");
        if (! p_fichier)
            printf("\a erreur: impossible d'ouvrir le fichier: %s.\n", nom_fichier);
    }
    while (! p_fichier);    // ou: while (p_fichier == 0)
```



```
printf("Enregistrement à ajouter: ");  
scanf("%d%s", &num_ajout, nom_ajout);
```

```
/* écriture du nouvel enregistrement, */
```

```
fprintf(p_fichier, "%d\n%s\n", num_ajout, nom_ajout);
```

```
/* fermeture du fichier */
```

```
fclose(p_fichier);
```

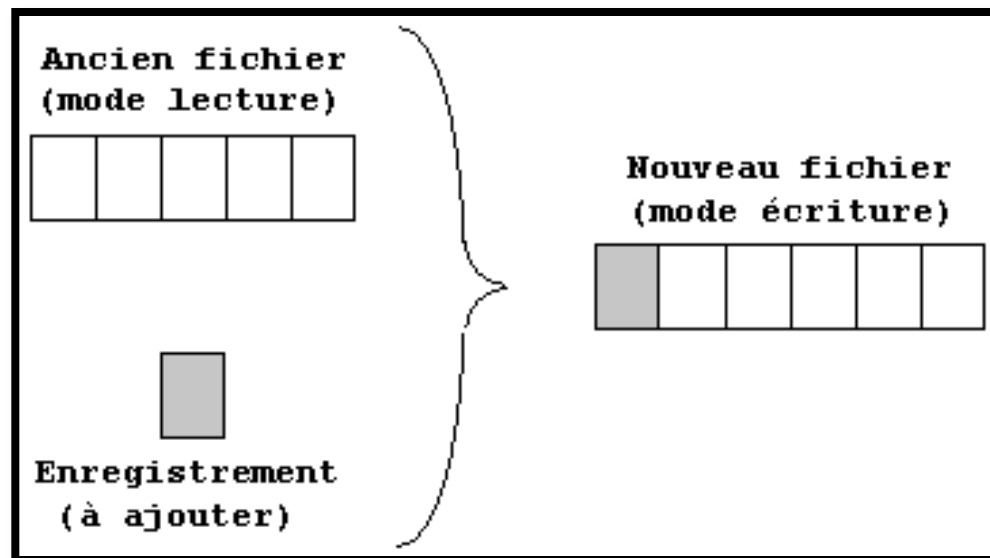
```
system("PAUSE");
```

```
return 0;
```

```
}
```

# Ajouter un enregistrement au début du fichier

- On utilise un nouveau fichier qui va contenir l'enregistrement à ajouter et les enregistrements de l'ancien fichier.
- Donc, il faut:
  - 1) Ouvrir l'ancien fichier en mode lecture.
  - 2) Ouvrir le nouveau fichier en mode écriture.
  - 3) Ecrire le nouveau enregistrement dans le nouveau fichier.
  - 4) Copier le contenu de l'ancien fichier dans le nouveau fichier: Lire à partir de l'ancien fichier et les écrire dans le nouveau fichier.



# Exemple:

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

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

```
main() {
```

```
    char ancien[30], nouveau[30];
```

```
    FILE *p_fichier_ancien, * p_fichier_nouveau;
```

```
    char nom_pers[30]; int num_pers;  // pour les enreg de l'ancien fichier
```

```
    char nom_ajout[30]; int num_ajout; // pour l'enreg à ajouter
```

```
    /* 1- ouverture de l'ancien fichier en mode lecture */
```

```
    do {
```

```
        printf("nom de l'ancien fichier : ");
```

```
        scanf("%s", ancien);
```

```
        p_fichier_ancien = fopen(ancien, "r");
```

```
        if (! p_fichier_ancien)
```

```
            printf("\a erreur: impossible d'ouvrir le fichier: %s.\n", ancien);
```

```
    } while (! p_fichier_ancien);    // tant que (p_fichier_ancien == 0)
```



```
/* 2- ouverture du nouveau fichier en écriture */
```

```
do {
```

```
    printf("nom du nouveau fichier : ");
```

```
    scanf("%s", nouveau);
```

```
    p_fichier_nouveau = fopen(nouveau, "w");
```

```
    if (!p_fichier_nouveau)
```

```
        printf("Impossible d'ouvrir le fichier: %s.\n", nouveau);
```

```
}
```

```
while (!p_fichier_nouveau);
```

```
printf("Enregistrement à ajouter: ");
```

```
scanf("%d%s", &num_ajout, nom_ajout);
```

```
/* 3- ecriture du nouvel enregistrement, */
```

```
fprintf(p_fichier_nouveau, "%d\n%s\n", num_ajout, nom_ajout);
```



**/\* 4- copier le contenu de l'ancien fichier dans le nouveau fichier \*/**

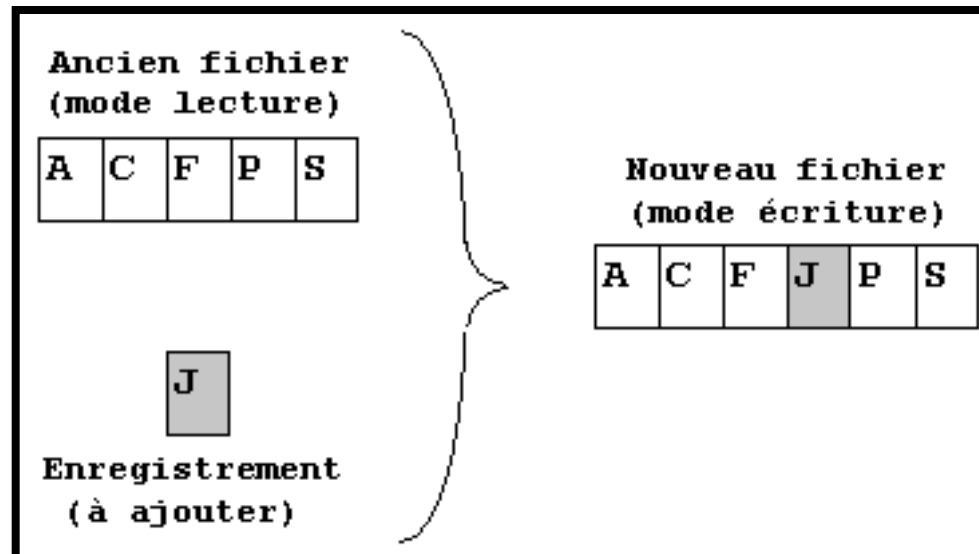
```
while(!feof(p_fichier_ancien)){  
    fscanf(p_fichier_ancien, "%d\n%s\n", &num_pers, nom_pers); // lecture  
    fprintf(p_fichier_nouveau, "%d\n%s\n", num_pers, nom_pers); // écriture  
}
```

**/\* fermeture des fichiers \*/**

```
fclose(p_fichier_nouveau);  
fclose(p_fichier_ancien);  
system("PAUSE");  
return 0;  
}
```


# Ajouter un enregistrement dans un fichier trié

- On utilise un nouveau fichier.
- On procède comme suit:
  - 1) Ouvrir l'ancien fichier en mode lecture,
  - 2) Ouvrir le nouveau fichier en mode écriture,
  - 3) Copier les enregistrements de l'ancien fichier qui précèdent l'enregistrement à ajouter (faire des comparaisons),
  - 4) Ecrire l'enregistrement à ajouter dans le nouveau fichier,
  - 5) Copier le reste des enregistrements de l'ancien fichier au nouveau fichier.



# Exemple

- Ce programme ajoute un enregistrement dans un fichier trié selon le champ `nom_pers`. On compare les noms des personnes pour retrouver la bonne position pour ajouter le nouveau l'enregistrement.



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

```
#include <string.h>
```

```
main() {
```

```
    char ancien[30], nouveau[30];
```

```
    FILE *p_fichier_ancien, * p_fichier_nouveau;
```

```
    char nom_pers[30]; int num_pers;  // pour les enreg de l'ancien fichier
```

```
    char nom_ajout[30]; int num_ajout; // pour l'enreg à ajouter
```

```
    int trouve;
```

```
    /* 1- ouverture de l'ancien fichier en lecture */
```

```
    do {
```

```
        printf("nom de l'ancien fichier : ");
```

```
        scanf("%s", ancien);
```

```
        p_fichier_ancien = fopen(ancien, "r");
```

```
        if (! p_fichier_ancien)
```

```
            printf("\a erreur: impossible d'ouvrir le fichier: %s.\n", ancien);
```

```
    } while (! p_fichier_ancien);    // tant que (p_fichier_ancien == 0)
```



```
/* 2- ouverture du nouveau fichier en écriture */
```

```
do {  
    printf("nom du nouveau fichier : ");  
    scanf("%s", nouveau);  
    p_fichier_nouveau = fopen(nouveau, "w");  
    if (!p_fichier_nouveau)  
        printf("Impossible d'ouvrir le fichier: %s.\n", nouveau);  
} while (! p_fichier_nouveau);
```

```
printf("Enregistrement à insérer : ");  
scanf("%d%s", &num_ajout, nom_ajout);
```



```
trouve = 0;
```

```
/* 3- copier les enregistrements dont le nom précède celui à insérer.*/
```

```
/* sortir de la boucle lorsqu'on trouve un nom plus grand que nom_ajout */
```

```
while( ! feof(p_fichier_ancien) && ! trouve ){           // trouve==0
```

```
    fscanf(p_fichier_ancien, "%d\n%s\n", &num_pers, nom_pers); // lecture
```

```
    if (strcmp(nom_pers, nom_ajout) > 0) { //strcmp est défini dans <string.h>
```

```
        trouve=1;
```

```
/* 4- ecriture du nouvel enregistrement */
```

```
    fprintf(p_fichier_nouveau, "%d\n%s\n", num_ajout, nom_ajout);
```

```
/*Ecriture de l'enregistrement immédiatement supérieur au nouvel enregistrement*/
```


```
    fprintf(p_fichier_nouveau, "%d\n%s\n", num_pers, nom_pers);
```

```
}
```

```
else
```

```
    fprintf(p_fichier_nouveau, "%d\n%s\n", num_pers, nom_pers);
```

```
}
```

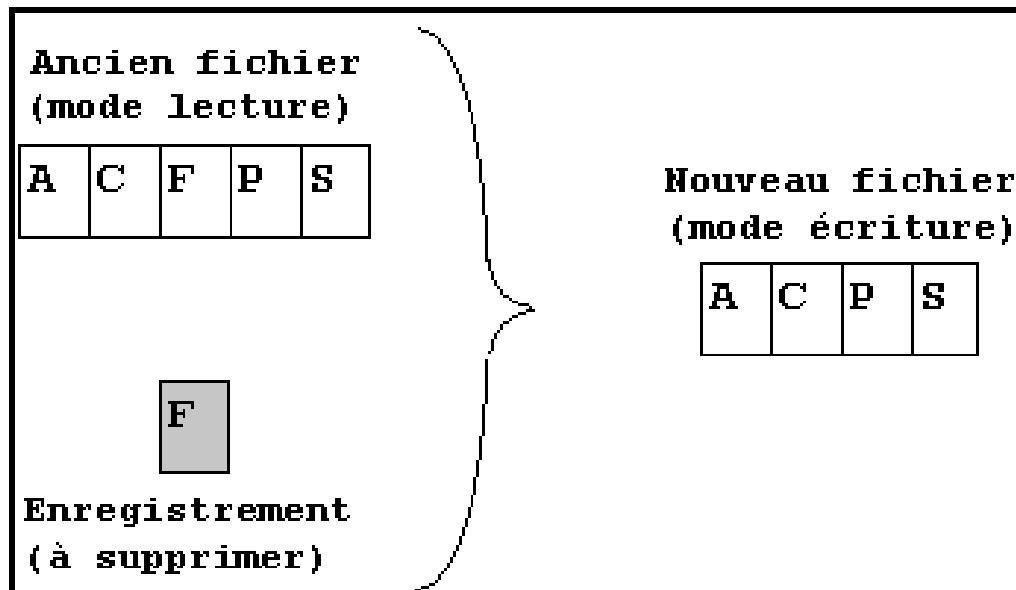


```
/* 5- copie du reste des enregistrements */  
while (!feof(p_fichier_ancien)) {  
    fscanf(p_fichier_ancien, "%d\n%s\n", & num_pers, nom_pers);  
    fprintf(p_fichier_nouveau, "%d\n%s\n", num_pers, nom_pers);  
}
```

```
/* fermeture des fichiers */  
fclose(p_fichier_nouveau);  
fclose(p_fichier_ancien);  
return 0;  
}
```

# Supprimer un enregistrement d'un fichier

- On utilise un nouveau fichier.
- On procède comme suit:
  - 1) Ouvrir l'ancien fichier en mode lecture,
  - 2) Ouvrir le nouveau fichier en mode écriture,
  - 3) Copier tous les enregistrements de l'ancien fichier qui sont différents de l'enregistrement à supprimer.





```
#include <stdio.h>
#include <string.h>
main() {
    /* déclarations des noms des fichiers et des pointeurs sur les fichiers */
    char ancien[30], nouveau[30];
    FILE *p_fichier_ancien, *p_fichier_nouveau;
    char nom_suppr[30];
    int num_pers; char nom_pers[30];
    /* 1- ouverture de l'ancien fichier en lecture */
    do {
        printf("nom de l'ancien fichier : ");
        scanf("%s", ancien);
        p_fichier_ancien = fopen(ancien, "r");
        if (!p_fichier_ancien)
            printf("\a erreur: impossible d'ouvrir le fichier: %s.\n", ancien);
    }
    while (!p_fichier_ancien);
```

*/\* 2- ouverture du nouveau fichier en écriture \*/*

```
do {  
    printf("Nom du nouveau fichier : ");  
    scanf("%s", nouveau);  
    p_fichier_nouveau = fopen(nouveau, "w");  
    if (!p_fichier_nouveau) printf("\a Impossible d'ouvrir le fichier:%s.\n",  
        nouveau);  
}  
while (!p_fichier_nouveau);
```

*/\* saisie de l'enregistrement à supprimer \*/*

```
printf("Enregistrement à supprimer : ");  
scanf("%s", nom_suppr);
```

*/\* 3- copie de tous les enregistrements à l'exception de celui à supprimer. \*/*

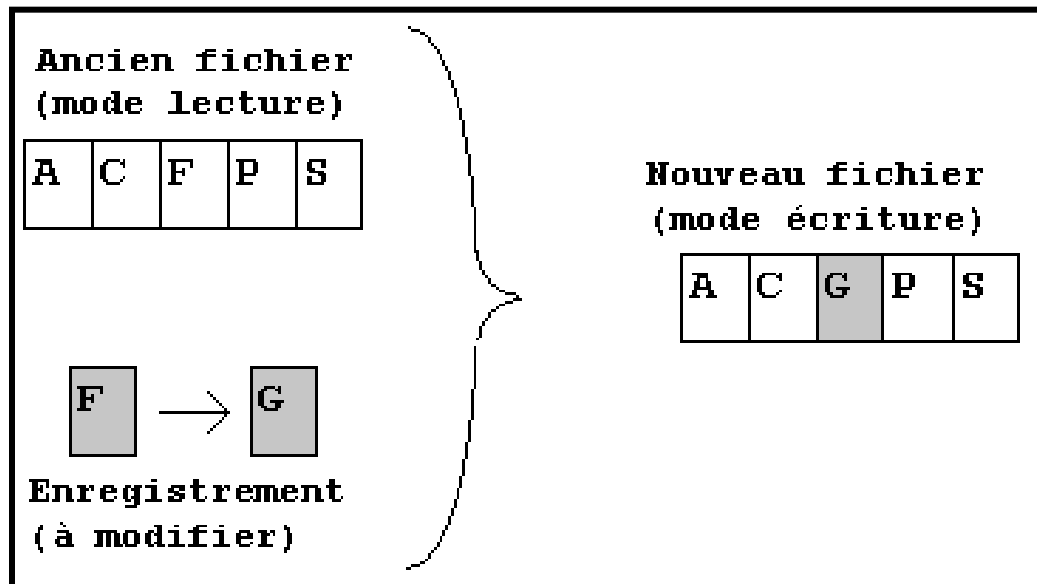
```
while ( !feof (p_fichier_ancien) ) {  
    fscanf(p_fichier_ancien, "%d\n%s\n", &num_pers, nom_pers);  
    if (strcmp(nom_pers, nom_suppr) != 0)  
        fprintf(p_fichier_nouveau, "%d\n%s\n", num_pers, nom_pers);  
}
```


*/\* fermeture des fichiers \*/*

```
fclose(p_fichier_nouveau);  
fclose(p_fichier_ancien);  
return 0;  
}
```

# Modifier un enregistrement dans un fichier

- On utilise un nouveau fichier.
- On procède comme suit:
  - 1) Ouvrir l'ancien fichier en mode lecture,
  - 2) Ouvrir le nouveau fichier en mode écriture,
  - 3) Copier tous les enregistrements de l'ancien fichier qui sont différents de l'enregistrement à modifier,
  - 4) Ecrire l'enregistrement modifié.





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

```
#include <string.h>
```

```
main() {
```

```
    /* déclarations des noms des fichiers et des pointeurs sur les fichiers */
```

```
    char ancien[30], nouveau[30];
```

```
    FILE *p_fichier_ancien, *p_fichier_nouveau;
```

```
    int num_pers;  char nom_pers[30];
```

```
    char nom_modif[30];
```

```
    int num_nouv;  char nom_nouv[30];
```

```
    /* 1- ouverture de l'ancien fichier en lecture */
```

```
    do {
```

```
        printf("Nom de l'ancien fichier : ");
```

```
        scanf("%s", ancien);
```

```
        p_fichier_ancien = fopen(ancien, "r");
```

```
        if (!p_fichier_ancien)
```

```
            printf("\a erreur: impossible d'ouvrir le fichier: %s.\n", ancien);
```

```
    } while (!p_fichier_ancien);
```

*/\* 2- ouverture du nouveau fichier en écriture \*/*

```
do {  
    printf("Nom du nouveau fichier : ");  
    scanf("%s", nouveau);  
    p_fichier_nouveau = fopen(nouveau, "w");  
    if (!p_fichier_nouveau)  
        printf("\a erreur: impossible d'ouvrir le fichier: %s.\n", nouveau);  
}  
while (!p_fichier_nouveau);
```

*/\* saisie de l'enregistrement à modifier et de sa nouvelle valeur\*/*

```
printf("Enregistrement à modifier : ");  
scanf("%s", nom_modif);  
printf("Nouveau enregistrement : num et nom");  
scanf("%d%s", &num_nouv, nom_nouv);
```

*/\* 3 et 4- copie de tous les enregistrements avec le remplacement \*/*

```
while ( ! feof (p_fichier_ancien) ) {  
    fscanf(p_fichier_ancien, "%d\n%s\n", &num_pers, nom_pers);  
    if( strcmp (nom_pers, nom_modif) == 0)  
        fprintf(p_fichier_nouveau, "%d\n%s\n", num_nouv, nom_nouv);  
    else  
        fprintf(p_fichier_nouveau, "%d\n%s\n", num_pers, nom_pers);  
}
```

*/\* fermeture des fichiers \*/*

```
fclose(p_fichier_nouveau);  
fclose(p_fichier_ancien);  
return 0;  
}
```



# Déplacements dans les fichiers



# Présentation

- Chaque fois que vous ouvrez un fichier, il existe un curseur qui indique votre position dans le fichier (comme le curseur de votre éditeur de texte). Il indique où vous êtes dans le fichier, et donc où vous allez écrire.
- En résumé, le système de curseur vous permet de lire et d'écrire à une position précise dans le fichier.
- Il faut savoir les trois fonctions suivantes:
  - **ftell** : indique à quelle position vous êtes actuellement dans le fichier ;
  - **fseek** : positionne le curseur à un endroit précis ;
  - **rewind** : remet le curseur au début du fichier (c'est équivalent à demander à la fonction **fseek** de positionner le curseur au début).

# ftell : position dans le fichier

- Cette fonction est très simple à utiliser. Elle renvoie la position actuelle du curseur sous la forme d'un long (nombre d'octets):

`long ftell(FILE* p_fichier);`

- Le nombre renvoyé indique donc la position du curseur dans le fichier.

# Exemple

- Lire le contenu d'un fichier et afficher la position du curseur:  

```
p_fichier = fopen (nom_fichier, "r"); // Ouverture du fichier pour lire
while ( ! feof (p_fichier)) {
    long pos = ftell(p_fichier);
    printf("La position est: %ld", pos); // Afficher la position du curseur
    fscanf(p_fichier, "%d\n%s\n", &num_pers, nom_pers);
    printf("num : %d et nom : %s\n", num_pers, nom_pers);
}
fclose(p_fichier);
```

# fseek : se positionner dans le fichier

- Le prototype de `fseek` est le suivant :

```
int fseek ( FILE* p_fichier, long déplacement, int origine );
```

- La fonction `fseek` permet de déplacer le curseur d'un certain nombre de caractères (indiqué par `déplacement`) à partir de la position indiquée par `origine`.
- Le nombre `déplacement` peut être un nombre positif (pour se déplacer en avant), nul (= 0) ou négatif (pour se déplacer en arrière).

# fseek (Suite)

- Le nombre **origine** peut prendre une des 3 constantes suivantes:
  - **SEEK\_SET** : indique le début du fichier ;
  - **SEEK\_CUR** : indique la position actuelle du curseur ;
  - **SEEK\_END** : indique la fin du fichier .

## Exemples :

- Le code suivant place le curseur deux octets **après** le début :  
`fseek(p_fichier, 2, SEEK_SET);`
- Le code suivant place le curseur quatre octets **avant** la position courante (déplacement négatif ):  
`fseek(p_fichier, -4, SEEK_CUR);`
- Le code suivant place le curseur à la fin du fichier :  
`fseek(p_fichier, 0, SEEK_END);`

# Remarques

- Si vous écrivez après avoir fait un **fseek** qui positionne le curseur à la fin, alors le texte se place à la fin du fichier.
- En revanche, si vous placez le curseur au début et vous écrivez, cela écrasera le contenu du fichier.

# Exemple:

- Ajouter un enregistrement à la fin du fichier: **fseek**

```
p_fichier = fopen("personne.txt", "r+"); // ouvrir le fichier en lecture et en écriture
printf("Enregistrement à insérer : ");
scanf("%d%s", &num_ajout, nom_ajout);
printf("La position est: %ld", ftell(p_fichier)); // Afficher la position courante:0
fseek(p_fichier, 0, SEEK_END); // se placer à la fin du fichier
printf("La position est: %ld", ftell(p_fichier)); //Afficher la position courante: fin
/* écriture du nouvel enregistrement */
fprintf(p_fichier, "%d\n%s\n", num_ajout, nom_ajout);
fclose(p_fichier);
```

# rewind : retour au début

- Cette fonction est équivalente à utiliser `fseek` pour se placer au début du fichier.

`void rewind (FILE* p_fichier)`

- Exemple:

`rewind(p_fichier);`  `fseek(p_fichier, 0, SEEK_SET)`



# Renommer et supprimer un fichier

- **rename** : renomme un fichier.

Syntaxe:

```
int rename(const char* ancienNom, const char* nouveauNom);
```

- **remove** : supprime un fichier.

Syntaxe:

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

NB:

- Les 2 fonctions retourne 0 en cas de succès et une valeur différente de 0 en cas d'échec.
- Il ne faut pas préalablement ouvrir avec **fopen** le fichier à supprimer !

# Exemple

```
int main() {  
    char ancien[30], nouveau[30];  
    printf("nom de l'ancien fichier : ");  
    scanf("%s", ancien);  
    printf("nom du nouveau fichier : ");  
    scanf("%s", nouveau);  
    // traitements  
    int resultSup=remove(ancien);  
    if(resultSup!=0) printf("Impossible de supprimer le fichier %s", ancien);  
    else printf("Le fichier %s a ete supprime !", ancien);  
  
    int resultRen=rename(nouveau, ancien);  
    if(resultRen!=0) printf("Renommage impossible de %s\n", nouveau);  
    else printf("Le fichier %s est renommé en %s\n", nouveau, ancien);  
    return 0;  
}
```