Algorithmes et programmation II : La gestion des fichiers

Souheib Baarir[1]

[1]Université Paris Ouest Nanterre La Défense.
Laboratoire d'informatique de Paris 6.
Souheib.Baarir@u-paris10.fr

Licence Mia - 2010/2011

Grandes lignes du cours

Introduction sur les fichiers

Les fichiers et le language C

- La structure FILE
- Ouverture et fermeture d'un fichier
- Les entrées-sorties textes
- Les entrées-sorties binaires
- Les accès directes

Introduction sur les fichiers

Besoin des fichiers

- Jusqu'à maintenant nous n'avions vu que des *Entrées/Sorties (E/S)* sous la forme d'entrée standard (usuellement le clavier, `scanf`) et de sortie standard (usuellement l'écran, `printf`).

Besoin des fichiers

- Jusqu'à maintenant nous n'avions vu que des *Entrées/Sorties (E/S)* sous la forme d'entrée standard (usuellement le clavier, `scanf`) et de sortie standard (usuellement l'écran, `printf`).
- Ce type d'E/S atteint rapidement ses limites...

Besoin des fichiers

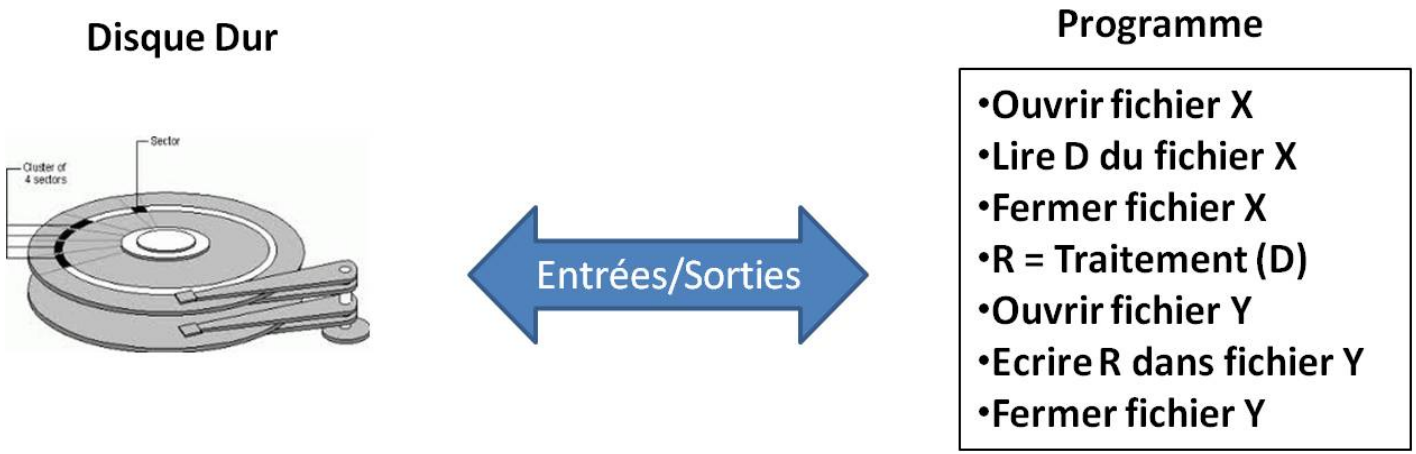
- Jusqu'à maintenant nous n'avions vu que des *Entrées/Sorties (E/S)* sous la forme d'entrée standard (usuellement le clavier, `scanf`) et de sortie standard (usuellement l'écran, `printf`).
- Ce type d'E/S atteint rapidement ses limites...

  Utilisation des fichiers : sauvegarde/restitution des données entre le programme et le *disque dur*.

Besoin des fichiers

- Jusqu'à maintenant nous n'avions vu que des *Entrées/Sorties (E/S)* sous la forme d'entrée standard (usuellement le clavier, `scanf`) et de sortie standard (usuellement l'écran, `printf`).
- Ce type d'E/S atteint rapidement ses limites...

Utilisation des fichiers : sauvegarde/restitution des données entre le programme et le *disque dur*.



Types de fichiers

Il existe deux types de fichiers (définitions Wikipédia) :

Types de fichiers

Il existe deux types de fichiers (définitions Wikipédia) :

- **Les fichiers *textes*** : sont les fichiers dont le contenu représente uniquement une suite de caractères imprimables, d'espaces et de retours à la ligne (.txt,...). Ils peuvent être lus directement par un *éditeur de texte*.

Types de fichiers

Il existe deux types de fichiers (définitions Wikipédia) :

- **Les fichiers *textes*** : sont les fichiers dont le contenu représente uniquement une suite de caractères imprimables, d'espaces et de retours à la ligne (.txt,...). Ils peuvent être lus directement par un *éditeur de texte*.
- **Les fichiers *binaires*** : sont les fichiers qui ne sont pas assimilables à des fichiers textes (.exe, .mp3, .png, .doc,...). Ils ne peuvent pas être lus directement par un éditeur de texte .

Mécanique de l'utilisation des fichiers (1/2)

- L'accès à un fichier d'un support de stockage de masse (disque dur, disque optique,...) est coûteux : temps des transfers, mécanique détériorable,...

Mécanique de l'utilisation des fichiers (1/2)

- L'accès à un fichier d'un support de stockage de masse (disque dur, disque optique,...) est coûteux : temps des transfers, mécanique détériorable,...
- Donc, il faut réduire le nombre d'accès

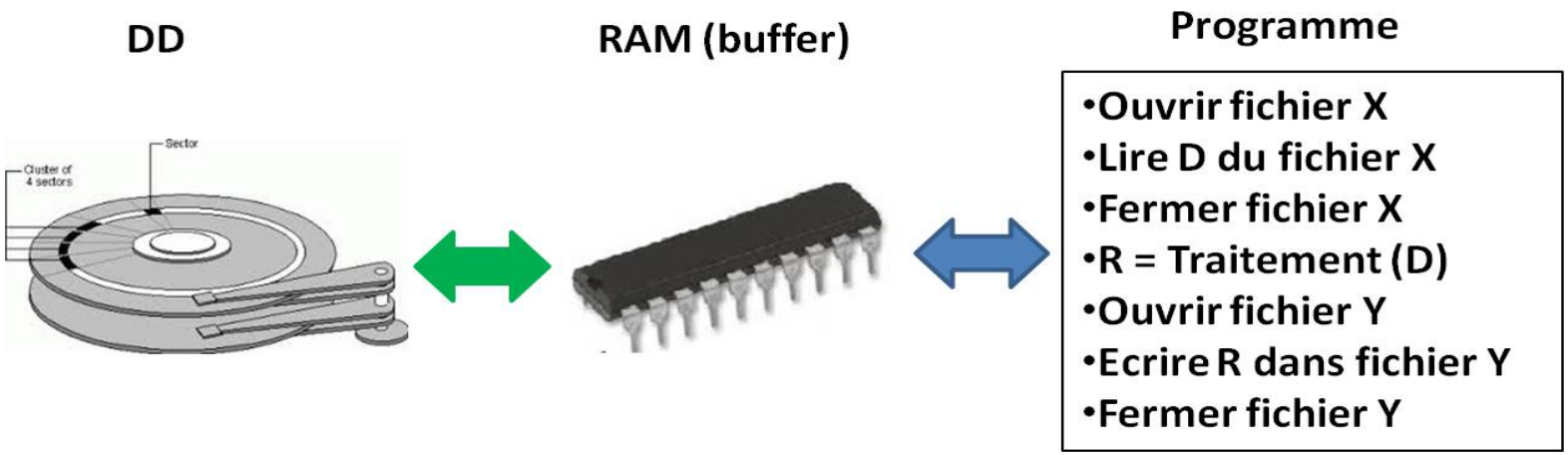
Mécanique de l'utilisation des fichiers (1/2)

- L'accès à un fichier d'un support de stockage de masse (disque dur, disque optique,...) est coûteux : temps des transfers, mécanique détériorable,...
- Donc, il faut réduire le nombre d'accès $\Rightarrow$ Utilisation d'une zone mémoire appelée : ***zone tampon ("buffer")***.

Mécanique de l'utilisation des fichiers (1/2)

- L'accès à un fichier d'un support de stockage de masse (disque dur, disque optique,...) est coûteux : temps des transfers, mécanique détériorable,...
- Donc, il faut réduire le nombre d'accès $\Rightarrow$ Utilisation d'une zone mémoire appelée : ***zone tampon ("buffer").***
- Ainsi, une instruction d'écriture (resp. lecture) dans le programme ne se traduira pas immédiatement par une écriture (resp. lecture) sur le disque mais ***par une "écriture (resp. lecture)" dans le buffer, avec écriture (resp. lecture) sur disque quand le buffer est plein (resp. vide).***

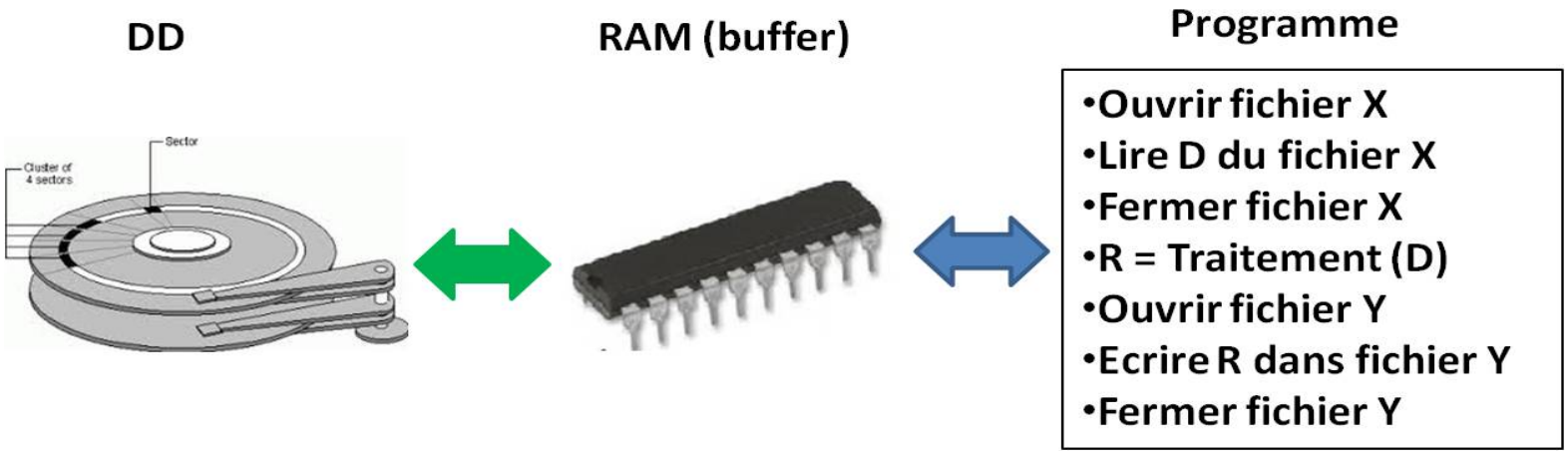
Mécanique de l'utilisation des fichiers (2/2)



Dans mon programme :

- Ouvrir un fichier.
- Lire/écrire dans le fichier ouvert.
- Fermer le fichier.

Mécanique de l'utilisation des fichiers (2/2)



Dans mon programme , le système d'exploitation fait :

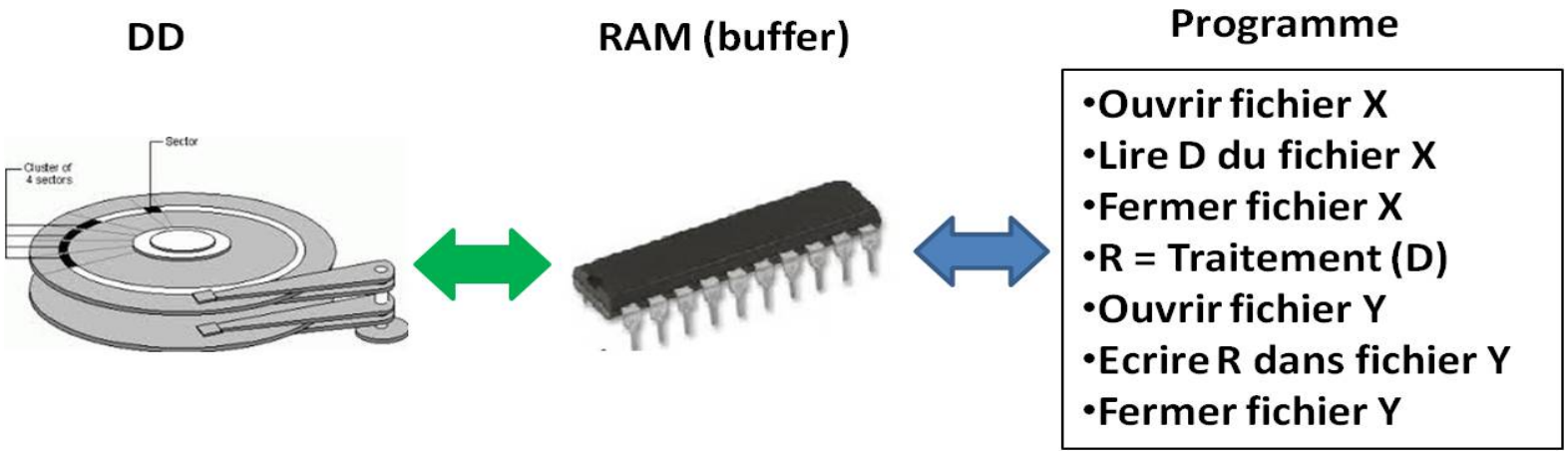
- Ouvrir un fichier.

  $\Rightarrow$ créer un *buffer (b)* dans la RAM.

- Lire/écrire dans le fichier ouvert.

- Fermer le fichier.

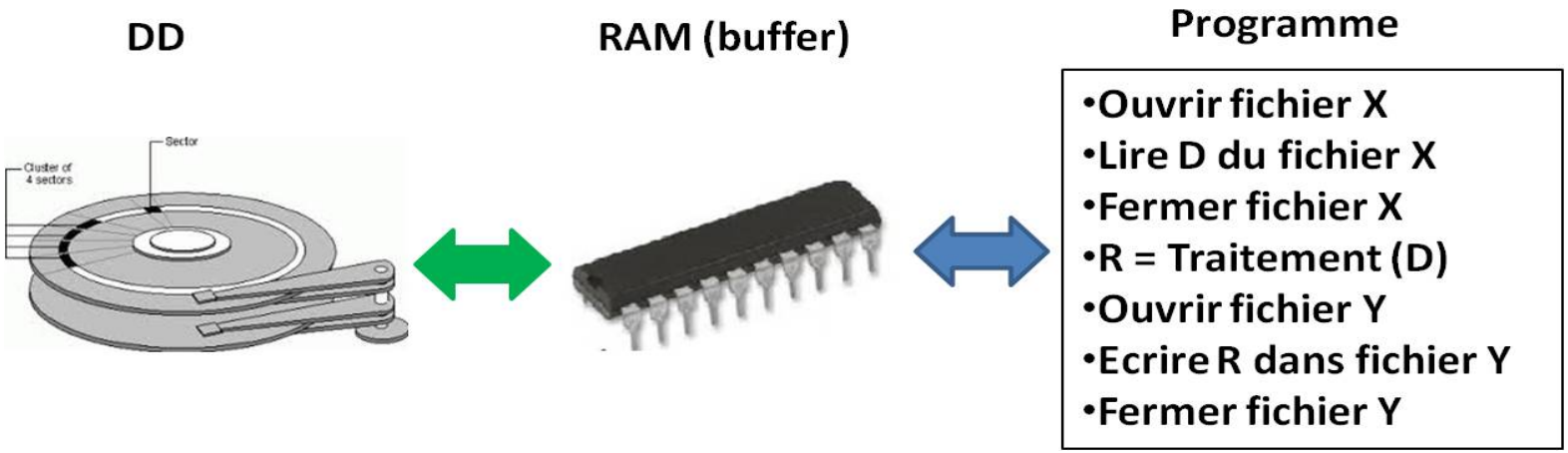
Mécanique de l'utilisation des fichiers (2/2)



Dans mon programme , le système d'exploitation fait :

- Ouvrir un fichier.
  $\Rightarrow$ créer un *buffer (b)* dans la RAM.
- Lire/écrire dans le fichier ouvert.
  $\Rightarrow$ lire/écrire dans *b*.
- Fermer le fichier.

Mécanique de l'utilisation des fichiers (2/2)



Dans mon programme , le système d'exploitation fait :

- Ouvrir un fichier.
  $\Rightarrow$ créer un *buffer (b)* dans la RAM.
- Lire/écrire dans le fichier ouvert.
  $\Rightarrow$ lire/écrire dans b.
- Fermer le fichier.
  $\Rightarrow$ "flusher" le contenu de b, libérer b,...

Outline

La structure FILE

- En langage C, les informations nécessaires à maintenir l'association **programme** $\Leftrightarrow$ **buffer** $\Leftrightarrow$ **disque dur** sont décrites dans une structure `FILE` (`stdio.h`).

La structure FILE

- En langage C, les informations nécessaires à maintenir l'association **programme** $\Leftrightarrow$ **buffer** $\Leftrightarrow$ **disque dur** sont décrites dans une structure `FILE` (`stdio.h`).
- Parmi les informations stockées dans la structure `FILE` , on trouve :
  - *le N° du fichier à ouvrir,*
  - *le type d'ouverture (lecture/écriture),*
  - *l'adresse du **buffer** associé,*
  - *la position du curseur de lecture,*
  - *la position du curseur d'écriture,*
  - ...

La structure FILE

- En langage C, les informations nécessaires à maintenir l'association **programme** $\Leftrightarrow$ **buffer** $\Leftrightarrow$ **disque dur** sont décrites dans une structure `FILE` (`stdio.h`).
- Parmi les informations stockées dans la structure `FILE` , on trouve :
  - *le N° du fichier à ouvrir,*
  - *le type d'ouverture (lecture/écriture),*
  - *l'adresse du **buffer** associé,*
  - *la position du curseur de lecture,*
  - *la position du curseur d'écriture,*
  - ...
- Pour utiliser un fichier, il faut donc commencer par déclarer une variable de type `FILE`, ou plus exactement un pointeur sur `FILE` (`FILE *`), qu'on appel aussi *flux de données* :

```
FILE * nomPointeurFichier;
```

Outline

Ouverture et fermeture d'un fichier

fopen et *fclose* : definition

Le language C offre deux fonctions pour l'ouverture et la fermeture d'un fichier :

- La fonction `fopen` : permet d'ouvrir un fichier, suivant un *mode*, et retourne un flux (pointeur sur `FILE`).

  ```
  FILE * fopen(char* nomFichier, char* mode)
  ```

  La fonction retourne `NULL` si l'ouverture n'est pas possible

- La fonction `fclose` : permet de fermer un fichier (un flux) ouvert.

  ```
  void fclose(FILE * pf)
  ```

fopen et *fclose* : exemple

```c
// Déclaration du flux
FILE * fp ;

// Ouvrir le fichier ./test.text en écriture
// et association au flux
if ((fp=fopen("./test.text","w"))==NULL){
     printf("Impossible d'ouvrir le fichier \n");
     return -1;
}

...

// Fermeture du flux (du fichier)
fclose(fp);
```

Les cas *stdin*, *stdout*, et *stderr*

Trois flux standards peuvent être utilisés en C sans ouverture, ni fermeture :

- `stdin` (standard input) : par défaut le clavier ;
- `stdout` (standard output) : par défaut l'écran ;
- `stderr` (standard error) : par défaut l'écran ;

Les modes d'ouvertures

Les différents modes d'ouvertures d'un fichier sont :

Mode	Signification
"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+"	ouverture d'un fichier texte en lecture/écriture
"a+"	ouverture d'un fichier texte en lecture/écriture à la fin
"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

Outline

getc, putc, fscanf et fprintf : definition (1/2)

Plusieurs fonctions pour la lecture/écriture depuis/dans les fichiers textes existent :

- `int getc(FILE * pf)` : retourne le caractère suivant du flux `pf`. Elle retourne la constante `EOF` si elle rencontre la fin du fichier ou en cas d'erreur.

getc, putc, fscanf et fprintf : definition (1/2)

Plusieurs fonctions pour la lecture/écriture depuis/dans les fichiers textes existent :

- `int getc(FILE * pf)` : retourne le caractère suivant du flux `pf`. Elle retourne la constante `EOF` si elle rencontre la fin du fichier ou en cas d'erreur.
- `int putc(int c, FILE * pf)` : écrit le caractère c dans le fichier associé à`pf`. Retourne le caractère écrit ou `EOF` en cas d'erreur.

getc, putc, fscanf et fprintf : definition (1/2)

Plusieurs fonctions pour la lecture/écriture depuis/dans les fichiers textes existent :

- `int getc(FILE * pf)` : retourne le caractère suivant du flux `pf`. Elle retourne la constante `EOF` si elle rencontre la fin du fichier ou en cas d'erreur.
- `int putc(int c, FILE * pf)` : écrit le caractère c dans le fichier associé à`pf`. Retourne le caractère écrit ou `EOF` en cas d'erreur.

Remarques :

- `getchar()` $\Leftrightarrow$ `getc(stdin)`

getc, putc, fscanf et fprintf : definition (1/2)

Plusieurs fonctions pour la lecture/écriture depuis/dans les fichiers textes existent :

- `int getc(FILE * pf)` : retourne le caractère suivant du flux `pf`. Elle retourne la constante `EOF` si elle rencontre la fin du fichier ou en cas d'erreur.
- `int putc(int c, FILE * pf)` : écrit le caractère c dans le fichier associé à`pf`. Retourne le caractère écrit ou `EOF` en cas d'erreur.

Remarques :

- `getchar()` $\Leftrightarrow$ `getc(stdin)`
- `putchar(c)` $\Leftrightarrow$ `putc(c,stdout)`

getc, putc, fscanf et fprintf : definition (2/2)

- `int fgetf(FILE * pf, char * format, arg1,...)` : lit les caractères sur le flux `pf`, les interprète selon la spécifications incluses dans `format`. Retourne le nombre d'éléments correctement lus ou `EOF` en cas d'erreur (avant toutes les lectures).

getc, putc, fscanf et fprintf : definition (2/2)

- `int fgetf(FILE * pf, char * format, arg1,...)` : lit les caractères sur le flux `pf`, les interprète selon la spécifications incluses dans `format`. Retourne le nombre d'éléments correctement lus ou `EOF` en cas d'erreur (avant toutes les lectures).
- `int fprintf(FILE * pf, char * format, arg1,...)` : convertit, met en forme (selon `format`) et imprime ses arguments dans le flux `pf`. Retourne le nombre de caractères imprimés ou `EOF` en cas d'erreur.

getc, putc, fscanf et fprintf : definition (2/2)

- `int fgetf(FILE * pf, char * format, arg1,...)` : lit les caractères sur le flux `pf`, les interprète selon la spécifications incluses dans `format`. Retourne le nombre d'éléments correctement lus ou `EOF` en cas d'erreur (avant toutes les lectures).
- `int fprintf(FILE * pf, char * format, arg1,...)` : convertit, met en forme (selon `format`) et imprime ses arguments dans le flux `pf`. Retourne le nombre de caractères imprimés ou `EOF` en cas d'erreur.

Remarques :

- `scanf(char * format,...)` $\Leftrightarrow$ `fscanf(stdin, char * format,...)`

getc, putc, fscanf et fprintf : definition (2/2)

- `int fgetf(FILE * pf, char * format, arg1,...)` : lit les caractères sur le flux `pf`, les interprète selon la spécifications incluses dans `format`. Retourne le nombre d'éléments correctement lus ou `EOF` en cas d'erreur (avant toutes les lectures).

- `int fprintf(FILE * pf, char * format, arg1,...)` : convertit, met en forme (selon `format`) et imprime ses arguments dans le flux `pf`. Retourne le nombre de caractères imprimés ou `EOF` en cas d'erreur.

Remarques :

- `scanf(char * format,...)` $\Leftrightarrow$ `fscanf(stdin, char * format,...)`

- `printf(char * format,...)` $\Leftrightarrow$ `fprintf(stdout, char * format,...)`

getc, putc, fscanf et fprintf : exemple

```c
int main(int arv, char * arg[]) {
    FILE *in, *out;
    int c;
    if ((in = fopen("entree.txt","r")) == NULL) {
        fprintf(stderr,"\nErreur: Impossible de lire");
        return(-1);
    }

    if ((out = fopen("sortie.txt","w")) == NULL) {
        fprintf(stderr,"\nErreur: Impossible d'écrire");
        return(-1);
    }

    while ((c = getc(in)) != EOF)
        putc(c, out);

    fclose(in); fclose(out);
    return(0);
}
```

Outline

fread, fwrite : definition

Plusieurs fonctions pour la lecture/écriture depuis/dans les fichiers binaires existent :

- `size_t fread(void * pointeur, size_t taille, size_t nombre, FILE * pf)` :
  lit un tableau de `nombre` éléments, chacun de taille `taille`, depuis le flux `pf`, et le stocke dans la zone pointée par `pointeur`. Le nombre total d'éléments lus sont retournés.

fread, fwrite : definition

Plusieurs fonctions pour la lecture/écriture depuis/dans les fichiers binaires existent :

- `size_t fread(void * pointeur, size_t taille, size_t nombre, FILE * pf)` :
  lit un tableau de `nombre` éléments, chacun de taille `taille`, depuis le flux `pf`, et le stocke dans la zone pointée par `pointeur`. Le nombre total d'éléments lus sont retournés.

- `size_t fwrite(void *pointeur, size_t taille, size_t nombre, FILE * pf)` :
  écrit un tableau de `nombre` éléments, chacun de taille `taille`, depuis la zone pointée par `pointeur` vers le fichier associé au flux `pf`. Le nombre total de bytes écrits sont retournés.

fread, fwrite : exemple

```c
#include <stdio.h>
#include <stdlib.h>
#define NB 50
int main(int arv, char * arg[]) {
    FILE *in, *out;
    int tab1[NB], tab2[NB];
    int i;

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

    // écriture du tableau dans sortie.bin
    if ((out = fopen("./sortie.bin", "wb")) == NULL) {
        fprintf(stderr, "\nImpossible d'écrire");
        return(-1);
    }

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

fread, fwrite : exemple (suite et fin)

```
    // lecture dans sortie.bin
    if ((in = fopen(./sortie.bin, "rb")) == NULL) {
        fprintf(stderr, "\nImpossible de lire");
        return(-1);
    }

    fread(tab2, NB * sizeof(int), 1, in);
    fclose(in);

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

    printf("\n");

    return(0);
}
```

Outline

Les accès directes

ftell, fseek : définition

Le langage C permet un accès directe au données d'un fichier.

- `int fseek (FILE * pf, long int offset, int origine)` : affecte l'indicateur de position associé à `pf`, par la position `offset + origine`.
  - `pf` : pointeur sur `FILE` identifiant le flux.
  - `offset` : nombre de bytes à partir de `origine`.
  - `origine` : position à partir de laquelle `offset` est ajouté. Peut-être spécifié par l'une des constantes suivantes,
    - SEEK_SET Début du fichier
    - SEEK_CUR Position courante du pointeur
    - SEEK_END Fin du fichier.
- `long int ftell ( FILE * pf )` : retourne la valeur actuelle de l'indicateur de position.

ftell, fseek : exemple

```c
#include <stdio.h>

int main (int arv, char * arg[]) {
  FILE * fp;
  long size;

  if ((fp = fopen ("monFichier.txt","rb"))==NULL) {
    fprintf(stderr,"\nImpossible d'ouvrir le fichier");
    return(-1);
  } else {
    fseek(fp, 0, SEEK_END);
    size = ftell(fp);
    fclose (fp);
    printf ("La taille de monFichier.txt: %ld bytes.\n",
              size);
  }
  return 0;
}
```

Exercice : encodage/décodage d'un fichier texte (1/2)

Considérons le fichier texte "FichierAvecSauts.txt" contenant une liste de lignes (chacune se terminant par '\n').

- Écrire une fonction qui permet de créer, à partir de «FichierAvecSauts.txt», deux fichiers :
  - un fichier constitué des lignes de «FichierSansSauts.txt», mis les unes à la suite des autres en supprimant le caractère '\n' qui les sépare,
  - un fichier d'index «FichierIndex.bin» dans lequel seront rangées les positions ainsi que les longueurs des lignes du fichier précédent.

- Écrire une fonction qui permet, à partir des deux fichiers «FichierSansSauts.txt» et «FichierIndex.bin», de restituer le fichier original.

Exercice : encodage/décodage d'un fichier texte (2/2)

Trois nouvelles fonctions sont utiles :

- `char * fgets (char * str, int num, FILE * pf)` : Lit une chaîne de caractères dans le flux `pf` et s'arrête après n-1 caractères lus, ou quand `'\n'` est lu, ou quand la fin de fichier est rencontrée. Si le caractère `'\n'` est lu, il figure dans str.

- `int fputs (const char * str, FILE * pf)` : Écrit la chaine str dans le flux pf.

- `int feof (FILE * pf)` : détecte une fin de fichier dans un flux pf.

La solution peut être trouvée sur le lien suivant : `./exemple.c`