

9- LANGUAGE C

FICHER

Loïc Cuvillon

l.cuvillon@unistra.fr

Sommaire chapitre 9

Fichier	380
• ouverture/fermeture	382
• lecture/écriture en mode texte	
• élément par élément (le plus utilisé)	387
• ligne par ligne	391
• caractère par caractère	392
• lecture/écriture en mode binaire	395

.

Fichiers

- Un impératif : stockage permanent de l'information

Fichiers : Ensembles structurés de données stockées sur un support externe (suite d'enregistrements)

- On peut distinguer 2 types de fichiers :
 - texte :
 - * organisé en lignes (séparées par '\n')
 - * succession de caractères (1 octet = 1 caractère lisible)
 - * human-readable (ex: loic, 30 ans, francais; john, 48 ans, belge;)
 - binaire :
 - * séquence d'octets avec une représentation binaire de l'information
 - * not human-readable
 - * adapté aux données hétérogènes et structures complexes
 - * fichier plus court mais risque d'incompatibilité entre machines utilisant un codage binaire différent pour les types de bases

Accès aux fichiers

- Le fichier est repéré par un nom externe sur le médium (disque dur)

`(/home/morane/fichiers/myfile.dat)`

Comment relier les entrées/sorties en C avec ce nom externe ?

■ On passe par une structure de type FILE

regroupe toutes les informations nécessaires au programme pour manipuler les données du fichier (voir `<stdio.h>`)

- adresse de la mémoire tampon
- position actuelle de la tête de lecture/écriture
- type d'accès au fichier : lecture, écriture, ...
- état du fichier
- ...

Ouverture/fermeture d'un fichier

- Une structure `FILE` doit être créée par `fopen(...)`

```
f= fopen("nom_fichier" , "mode_ouverture");
```

- négocie avec le système une structure `FILE`
 - alloue et initialise cette structure
 - rend un pointeur `f` sur cette structure (`NULL` si erreur d'ouverture)
-
- Exemple : `FILE *f ;` (déclaration pointeur)
`f = fopen("truc.txt", "w") ;` (ouverture)
-
- La commande `fclose(f)` annule cette liaison et ferme le fichier.

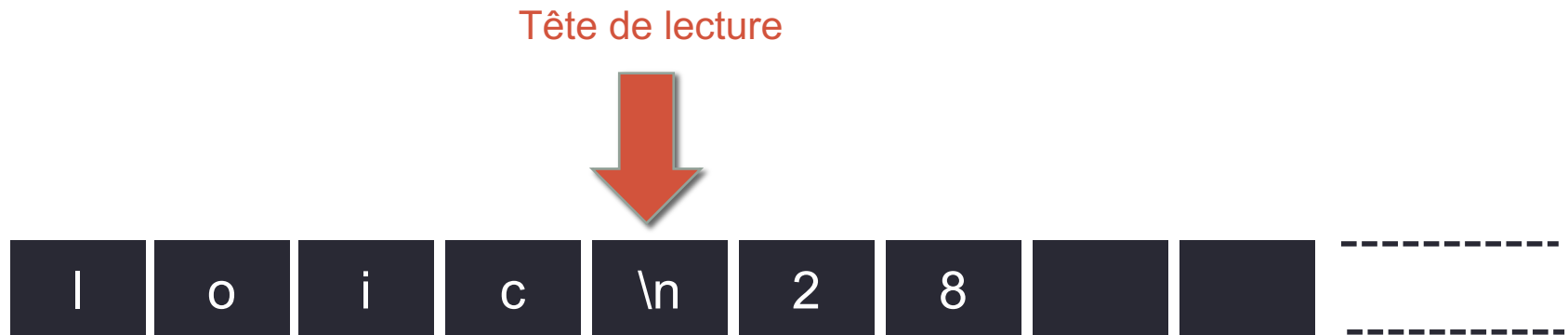
Modes d'ouverture des fichiers

	Mode d'ouverture	Position après ouverture	Si fichier non existant	Si fichier déjà existant
"r" (read)	lecture seule	début fichier	erreur	
"w" (write)	écriture seul	début fichier	création	écrasement !
"a" (append)	écriture seul	fin fichier	erreur	

Nom et chemin du fichier

- Pas d'accents, ni espaces dans les noms de fichiers !
- Chemin absolu : `“/home/etudiant/tp1/ex1.txt”`
- Chemin relatif (préférable) par rapport au répertoire ou est exécuté le programme :
 - `“ex1.txt”` ou `“./ex1.txt”` pour un fichier dans le répertoire courant
 - `“TP/ex.txt”` ou `“./TP/ex.txt”` pour un fichier dans sous-répertoire TP
 - `“../ex1.txt”` pour un fichier dans le répertoire supérieur

Lecture / écriture séquentielle dans le fichier



- On ne peut lire ou écrire que sous la tête de lecture
- La tête de lecture avance lors d'une instruction de lecture ou écriture

Sommaire chapitre 9

Fichier	380
• ouverture/fermeture	382
• lecture/écriture en mode texte	
• élément par élément (le plus utilisé)	387
• ligne par ligne	391
• caractère par caractère	392
• lecture/écriture en mode binaire	395

.

Lecture / écriture en mode texte

■ élément par élément (fichiers texte)

- `int fprintf (FILE *F, "format", expr1, expr2, ...);`
 - écrit les variables/expressions listées selon le `format` précisé dans le fichier `F`.
 - retourne le nombre de caractères écrits ou une valeur `<0` si erreur

Note: `fprintf(stdout, ..)` écrit sur la sortie standard → équivalent à `printf`
- `int fscanf (FILE *F, "format", &expr1, &expr2, ...);`
 - lit les données d'après le `format` annoncé dans le fichier `F` et les stocke aux adresses des variables indiquées.
 - retourne le nombre de données lues avec succès ou `EOF` (End of File) si fin de fichier ou erreur.
- `int feof(FILE *F)` Test si `EOF` est effectivement dû à une fin de fichier et non une erreur : retourne une valeur `≠ 0` pour VRAI.

Ecriture en mode texte

```
int main()
{
    FILE *F_out;
    char name[] = "loic";
    int age = 18, poids = 70;

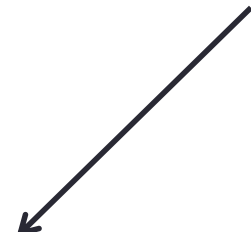
    F_out=fopen("liste.txt", "w");

    if (F_out ==NULL)
        {printf("Erreur"); return(-2); }

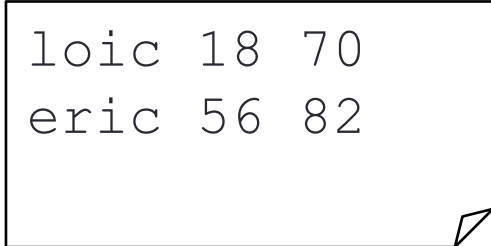
    fprintf(F_out, "%s %i %i\n", name, age, poids);
    fprintf(F_out, "%s %i %i\n", "eric", 56, 82);

    fclose(F_out);
    return(0); }
```

Sortie du programme
si échec ouverture du
fichier.



liste.txt:



```
loic 18 70
eric 56 82
```

Lecture en mode texte

```
int main()
{
    FILE *F_in;
    char name[100];
    int age, poids;

    F_in = fopen("liste.txt", "r");
    if (F_in == NULL)
        { printf("error open"); return(-2); }

    while ( fscanf(F_in, "%s %i %i",
                    name, &age, &poids) != EOF )
    {
        printf("%s %i %i", name, age, poids);
    }
    fclose(F_in);
    return(0);
}
```

liste.txt:

```
loic 18 70
eric 56 82
```

Tant que le code de retour de `fscanf` est différent de `EOF`

Affichage à l'exécution:

```
loic 18 70
eric 56 82
```

Lecture en mode texte d'un tableau

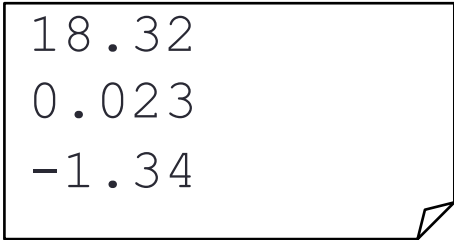
```
int main()
{
    FILE *F_in;
    int i=0;
    double tmp, tab[100];

    F_in = fopen("data.txt", "r");
    if (F_in == NULL)
        { printf("error"); return(-2); }

    while ( fscanf(F_in, "%lf", &tmp) != EOF )
    {
        tab[i]=tmp;
        printf("%f\n", tab[i]);
        i++;
    }

    fclose(F_in);
    return(0);
}
```

data.txt:



```
18.32
0.023
-1.34
```

Tant que le code de retour de `fscanf` est différent de `EOF`

Affichage à l'exécution:

```
18.32
0.023
-1.34
```

Lecture / écriture en mode texte

- ligne par ligne avec (fichiers texte)
- `char* fgets (char *s, int num, FILE *F);`
 - avec `num` la taille du tableau `s`, lit jusqu'au `'\n'` (une ligne) ou au maximum `num-1` caractères dans le fichier `F`.
 - la lecture (dont l'éventuel `'\n'`) est mise dans la chaîne pointée par `s`.
 - retourne le pointeur `s` en cas de succès ou `NULL` en cas d'erreur/fin de fichier.
Rappel: peut s'utiliser pour lire un ligne au clavier avec `fgets(, , stdin);`
- `int fputs (char *s, FILE *F);`
 - écrit la chaîne `s` dans le fichier `F`.
 - retourne 0 ou valeur positive en cas de succès, `EOF` en cas d'erreur/fin de fichier.

Lecture / écriture en mode texte

■ caractère par caractère (fichiers texte)

- `int fgetc (FILE *F) ;`

- lit un caractère du fichier F.

- retourne le caractère lu ou EOF si fin de fichier ou erreur.

- `int fputc (int c, FILE *F) ;`

- écrit le caractère c dans le fichier F.

- retourne le caractère écrit ou EOF si fin de fichier ou erreur.

- `int feof(FILE *F)` Test : retourne vrai si fin de fichier atteinte.

Copie d'un fichier texte

```
int main()
{
    FILE *F_in, *F_out;
    char c;

    F_in=fopen("liste.txt","r");
    F_out=fopen("copy.txt", "w");
    if (F_in == NULL || F_out== NULL)
        { printf("erreur"); return(-2); }

    while ( (c=fgetc(F_in)) != EOF )
        { fputc(c,F_out); }

    fclose(F_in);
    fclose(F_out);
    return(0); }
```

liste.txt:

```
loic 18 70
eric 56 82
```

Tant que `c` (le caractère lu) est différent de `EOF`.
[Attention au parenthèses]

liste.txt:

```
loic 18 70
eric 56 82
```

copy.txt:

```
loic 18 70
eric 56 82
```


Sommaire chapitre 9

Fichier	380
• ouverture/fermeture	382
• lecture/écriture en mode texte	
• élément par élément (le plus utilisé)	387
• ligne par ligne	391
• caractère par caractère	392
• lecture/écriture en mode binaire	395

.

Lecture / écriture en mode binaire

- Fichier binaire: efficace pour les gros tableaux et structures
- `int fwrite (type *bloc, int size, int nombre, FILE *F);`
 - écrit `nombre` éléments de taille `size` (octets) à partir de l'emplacement mémoire pointé par `bloc` dans le fichier `F`.
 - retourne le nombre d'éléments écrits (inférieur ou 0 si fin de fichier/erreur).
- `int fread (type *bloc, int size, int nombre, FILE *F);`
 - lit `nombre` éléments de taille `size` (octets) dans le fichier `F`. Le résultat est stocké à l'emplacement pointé par `bloc`.
 - retourne le nombre d'éléments lus (peut être $< \text{nombre}$ ou 0 si fin de fichier/erreur)
- `int feof(FILE* F)` Test : retourne vrai si fin de fichier atteinte.

Écriture en mode binaire

```
#include <stdio.h>

int main()
{
    FILE *OUT;
    int tab[8] = {7, 6, 5, 4, 3, 2, 1, -1};

    OUT = fopen("tableau.dat", "w");

    if (OUT == NULL)
    {printf ("erreur"); return(-2);}

    fwrite(tab, sizeof(int), 8, OUT);

    fclose(OUT);
    return(0);
}
```

tableau.dat: (ouvert par un éditeur de texte)

```
? @ de$ .ù  % 44
^`
```

tableau.dat: (ouvert par un éditeur hexadécimal)

```
07 00 00 00 06 00 00 00
05 00 00 00 04 00 00 00
03 00 00 00 02 00 00 00
01 00 00 00 FF FF FF FF
```

Lecture en mode binaire

```
int main()
{   FILE *IN;
    int i=0, n=10, *tab;
    tab=(int*)malloc(n*sizeof(int));

    IN = fopen("tableau.dat","r");
    /*a ajouter: test succes ouverture*/

    while ( fread(tab+i,sizeof(int),1,IN)==1)
        {   i++;   }

    printf("tab[0]: %i",tab[0]);

    fclose(IN);
    return(0);
}
```

tableau.dat: (ouvert par
un éditeur hexadécimal)

07	00	00	00	06	00	00	00
05	00	00	00	04	00	00	00
03	00	00	00	02	00	00	00
01	00	00	00	FF	FF	FF	FF

↖ tant qu'1 élément est lu
(test code retour fread)

Affiche à l'exécution:

tab[0] : 7

Lecture en mode binaire

```
int main()
{   FILE *IN;
    int i=0, n=10, *tab;
    tab=(int*)malloc(n*sizeof(int));

    IN = fopen("tableau.dat", "r");
    /*a ajouter: test succes ouverture*/

    fread(tab+i, sizeof(int), 8, IN); ←
    printf("tab[0]: %i", tab[0]);

    fclose(IN);
    return(0);
}
```

tableau.dat: (ouvert par un éditeur hexadécimal)

```
07 00 00 00 06 00 00 00
05 00 00 00 04 00 00 00
03 00 00 00 02 00 00 00
01 00 00 00 FF FF FF FF
```

Cas où nombre d'éléments (8) à lire est connu à l'avance.

[A ajouter test code retour de `fread()`]

Affiche à l'exécution:

```
tab[0] : 7
```