

Programmation avancée

Les Fichiers en C

Walter Rudametkin

Walter.Rudametkin@polytech-lille.fr
<https://rudametw.github.io/teaching/>

Bureau F011
Polytech Lille

CM5

1 / 20

Fichiers Binaires

- ▶ Les octets représentent la copie exacte des données en mémoire sur un système donné
- ▶ Non standard – Non éditables – Non imprimables
- ▶ Mais lecture / écriture plus rapides (pas d'analyse)
- ▶ En général, plus compacts
 - ▶ ex: 654875 = 6 octets (char), 2/4 octets (short/int)

Pas d'attribut *texte* ou *binaire* sur un fichier
(dépend de l'interprétation des octets)

- ▶ N'intervient pas à la déclaration
- ▶ Lié aux opérations applicables

3/20

Fichiers texte: ouverture

- ▶ Défini dans `<stdio.h>`
- ▶ `FILE * fopen(char *nom, char *mode)` où

$$\text{mode} = \begin{cases} \text{'r'} : & \text{lecture} \\ \text{'w'} : & \text{création/écriture} \\ \text{'a'} : & \text{allongement (ajout à la fin du fichier)} \end{cases}$$

lecture: F

A ₀	A ₁	A ₂				...	EOI
----------------	----------------	----------------	--	--	--	-----	-----

écriture: F

allongement: F

A ₀	A ₁	A ₂		...	A _{n-1}	A _n	EOI
----------------	----------------	----------------	--	-----	------------------	----------------	-----

5/20

Fichiers texte: exemple

```
1 #include <stdio.h>
2
3 int main(){
4     FILE *fp = fopen("toto.txt", "r");
5     if (fp == NULL) printf("toto.txt inaccessible: fp=%p\n", fp);
6     else printf("toto.txt accessible: fp=%p\n", fp);
7
8     if (fp != NULL){
9         printf("Fermer toto\n");
10        fclose(fp);
11    }
12 }
```

```
si toto.txt existe
toto.txt accessible:fp=0x1d12010
Fermer toto
```

```

    si toto.txt n'existe pas
    toto.txt inaccessible:
    fp=NULL

```

7/20

Les fichiers en C

Pas de fichiers de base dans le langage C

- ▶ Mais dans la bibliothèque standard `libc.so/libc.a` en incluant le fichier d'en-tête `<stdio.h>`

Un fichier C = une suite d'octets (= flot)

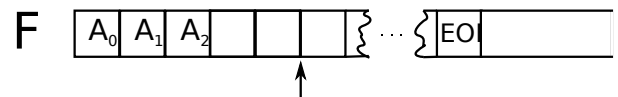
- ▶ Pas de types de fichiers (e.g. image, tableau ...)

Fichiers texte

- ▶ Les octets représentent des caractères codant les données (souvent le très limité ASCII, mais aussi le populaire et recommandé **UTF-8**)
- ▶ Standard – Éditables – Imprimables

2/20

Le type FILE



- ▶ Défini dans `<stdio.h>`
- ▶ Structure C contenant
 - ▶ Identification du fichier associé (descripteur)
 - ▶ Position du curseur dans le fichier
 - ▶ Tampon de lecture / écriture
 - ▶ Indication de mode d'ouverture ...
- ▶ Opérations sont effectuées sur un FILE * fourni à l'ouverture

4/20

Fichiers texte: ouverture/fermeture

Retour

- ▶ FILE * si tout va bien
- ▶ NULL si erreur (ex: fichier inexistant, pas les droits de lecture ou écriture, ...)

Fermeture

- ▶ `fclose(FILE *fp)`
- ▶ Déconnecte `fp` du fichier physique
- ▶ Libère la mémoire du programme associée au fichier
- ▶ Permet aux autres applications d'utiliser le fichier

6/20

Fichiers texte: utilisation

- ▶ Généralisation des manipulations effectuées sur l'entrée/sortie standard (`stdin`, `stdout`)
- ▶ Dans `<stdio.h>`
 - ▶ entrée standard : `FILE * stdin`
 - ▶ sortie standard : `FILE * stdout`
- ▶ Connexion à l'exécution aux entrées / sorties standard fournies par le système (console par défaut, redirigeables par `<`, `>` ou `|`)
- ▶ Lectures et écritures à partir de la position suivant le curseur

8/20

Fichiers texte: lecture

- ▶ `char getc (FILE *fp)`
 - ▶ `getchar()` \Leftrightarrow `getc(stdin)`
- ▶ `int fscanf(FILE *fp, char *format, ...)`
 - ▶ `scanf(...)` \Leftrightarrow `fscanf(stdin, ...)`
 - ▶ Retourne le nombre d'items lus, à vérifier à chaque utilisation
- ▶ `char * fgets(char *chaine, int taille, FILE *fp)`
- ▶ `int feof(FILE *fp)`
 - ▶ Retourne une valeur différent à zéro si la fin du fichier a été rencontrée lors d'une opération de lecture
- ▶ `int ferror(FILE *fp)`
 - ▶ Retourne une valeur différent à zéro si un erreur a été rencontrée lors d'une opération de lecture

9/20

Fichiers texte: mode écriture/allongement

- ▶ `int putc(char c, FILE *fp)`
 - ▶ `putc(c)` \Leftrightarrow `putc(c, stdout)`
- ▶ `int fprintf(FILE *fp, char *format, ...)`
- ▶ `int fputs(char *chaine, FILE *fp)`

10/20

Fichiers texte: exemple (MAUVAISE)

```
1  #include <stdio.h>
2
3  int main() {
4      FILE *fp1, *fp2;
5      if ((fp1 = fopen("titi.txt", "r")) != NULL) {
6          if ((fp2 = fopen("titibis.txt", "w")) != NULL) {
7              int c = getc(fp1);
8              while (!feof(fp1)){ //mauvais façon de faire!
9                  putc(c, fp2);
10                 c = getc(fp1);
11                 printf("%c", c);
12             }
13         }
14     }
15     if (fp1 != NULL) fclose(fp1);
16     if (fp2 != NULL) fclose(fp2);
17 }
```

11/20

Fichiers texte: exemple (BONNE)

```
1  #include <stdio.h>
2
3  int main() {
4      FILE *fp1, *fp2;
5      if ((fp1 = fopen("titi.txt", "r")) != NULL) {
6          if ((fp2 = fopen("titibis.txt", "w")) != NULL) {
7              int c = 0; //!= getc(fp1);
8              int p = 0;
9              while (c!=EOF && p!=EOF){ //BONNE FAÇON DE FAIRE !
10                 c = getc(fp1);
11                 p = putc(c, fp2);
12                 printf("%c:%c\n", p, c);
13                 printf("%d:%d\n", p, c);
14             }
15         }
16     }
17     if (fp1 != NULL) fclose(fp1);
18     if (fp2 != NULL) fclose(fp2);
19 }
```

12/20

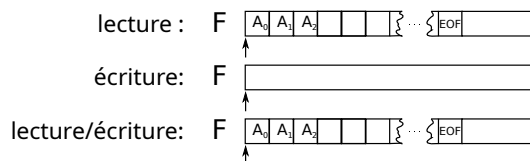
Fichiers binaires: ouverture

- ▶ Octets représentent la copie exacte du codage des données en mémoire

Ouverture

- ▶ `FILE *fopen (char *nom, char *mode)` où

mode = $\begin{cases} \text{rb:} & \text{lecture} \\ \text{wb:} & \text{création/écriture} \\ \text{ab:} & \text{allongement} \\ \text{rb+:} & \text{lecture/écriture} \\ \dots \end{cases}$



13/20

Fichiers binaires: fermeture/écriture

Fermeture (idem fichiers texte)

- ▶ `fclose(FILE *fp)`
- ▶ `int feof(FILE *fp)`

Écriture (mode création ou lecture/écriture)

- ▶ `size_t fwrite(const void *pt, size_t taille, size_t nb, FILE *fp);`

Écrit sur le fichier `fp`, à partir de la position suivant le curseur, `nb` objets, chacun de taille `taille`, qui se trouvent contiguëment dans la zone mémoire pointée par `pt`. Et la valeur de retour ?

- ▶ Utilisation courante :
`FILE *fp; <T> x;`
`fwrite(&x, sizeof(x), 1, fp);`

14/20

Fichiers binaires: écriture

Mode lecture/écriture

- ▶ `char x = '??';`
`fwrite(&x, 1, 1, fp);`

15/20

Fichiers binaires: lecture

- ▶ `size_t fread(void *pt, size_t taille, size_t nb, FILE *fp);`
Lire `nb` objets de taille `taille` et les copier dans l'espace pointé par `pt`.
- ▶ Utilisation courante :
`<T> x;`
`fread(&x, sizeof(<T>), 1, fp);`

16/20

Fichiers binaire: lecture (MAUVAISE)

```
1  #include <stdio.h>
2
3  typedef struct{
4      char nom[30];
5      int age;
6  } Personne;
7
8  int main(){
9      Personne P;
10     FILE * fich = fopen("personnes" ,"rb");
11     if(fich!=NULL){
12         fread(&P, sizeof(Personne), 1, fich);
13
14         while (!feof(fich)) { //MAUVAIS FAÇON DE TESTER!!!
15             printf("%s %d\n", P.nom, P.age);
16             fread(&P, sizeof(Personne), 1, fich);
17         }
18         fclose(fich);
19     }
20 }
```

17/20

Fichiers binaire: lecture (BONNE)

18/20

Fichiers binaires: accès direct

► `int fseek(FILE *fp, long déplacement, int origine)`

où $\text{origine} = \begin{cases} \text{SEEK_SET} : \text{début} \\ \text{SEEK_CUR} : \text{position courant} \\ \text{SEEK_END} : \text{fin} \end{cases}$

- Positionne le curseur pour la prochaine lecture ou écriture
- Position = déplacement + origine
- Usage courant :
`fseek(fp, i*sizeof(<T>), SEEK_SET);`
- `stdin` et `stdout` ne supportent pas `fseek`

19/20

Fichiers: conclusion

- *Texte* ou *binaire* n'est pas un attribut de fichier
- Un fichier texte peut être exploité en binaire comme simple suite d'octets
 - ex : `fseek(fp, i*sizeof(char), SEEK_SET);`
 - ex : utilisation de `fread` ou `fwrite` sur un fichier texte
- Exploitation d'un fichier binaire en texte ?????

20/20