Programmation avancée

Les Fichiers en C

Walter Rudametkin

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

Bureau F011
Polytech'Lille

CM5

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

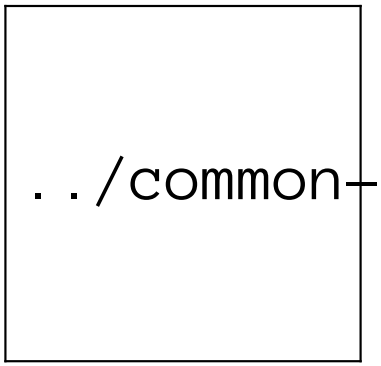
Fichiers Binaires

- 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

Le type `FILE`

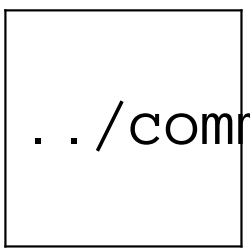


- Défini dans `<stdio.h>`
- Structure C contenant
  - Identification du fichier associé (descripteur)
  - Position du curseur dans le fichier
  - Tampon de lecture / écriture
  - Indication de mode d'ouverture ...
- Opérations sont effectuées sur un `FILE *` fourni à l'ouverture

Fichiers texte: ouverture

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

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



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

Fichiers texte: exemple

si toto.txt existe

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

si toto.txt n'existe pas

```
toto.txt inaccessible:
fp=NULL
```

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

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
- `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 (valeur lue indéterminée)

Fichiers texte: mode écriture/allongement

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

Fichiers texte: exemple

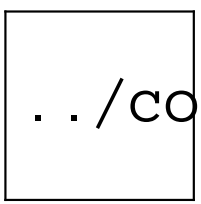
Fichiers binaires: ouverture

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

Ouverture

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

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



Fichiers binaires: fermeture/écriture

Fermeture (idem fichiers texte)

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

Écriture (mode création ou lecture/écriture)

- `int fwrite(void *pt, int taille, int 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`.
- Utilisation courante :

```
FILE *fp; <T> x;
fwrite(&x, sizeof(x), 1, fp);
```

Fichiers binaires: écriture

Mode lecture/écriture

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

Fichiers binaires: lecture

- `int fread(void *pt, int taille, int 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);`

Fichiers binaire: lecture

Fichiers binaires: accès direct

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

où `origine` $=$
$\begin{cases} \texttt{SEEK\_SET}: & \text{début} \\ \texttt{SEEK\_CUR}: & \text{position courant} \\ \texttt{SEEK\_END}: & \text{fin} \end{cases}$

- 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`

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 ?????