Cours Langage C/C++ - Les fichiers

Thierry Vaira

BTS IRIS Avignon

tvaira@free.fr v0.1

Introduction

- La notion d'enregistrement n'existant pas en C, un **fichier est vu comme une suite d'octets**.
- Ceci a pour incidence qu'il n'existe qu'une seule organisation de fichier : l'**organisation séquentielle**.
- L'accès direct reste cependant possible dans la mesure où le C permet un positionnement dans un fichier avec une précision d'un octet.
- Coupons court tout de suite aux fantasmes : il n'existe pas de fonction pour insérer des données au milieu d'un fichier. Il faut écrire (coder) ce mécanisme. La seule insertion possible supportée de base se fait en fin de fichier (mode ajout).

Les Entrées/Sorties

- Les Entrées/Sorties ne constituent pas une partie intrinsèque du langage.
- Pour utiliser la bibliothèque d'E/S standard du C, il faut inclure dans le programme source le contenu du fichier `stdio.h` et ajouter l'option `-lc` à l'édition de liens.
- Ce fichier contient un certain nombre de déclarations qui sont nécessaires pour utiliser les E/S standards.
- Tout programme C possède 3 voies standards de communication qui sont :

  `stdin` (pour valeur 0) généralement le clavier par défaut
  `stdout` (pour valeur 1) généralement l'écran par défaut
  `stderr` (pour valeur 2) généralement l'écran pour la sortie des erreurs

stdin, stdout et stderr

- `stdin`, `stdout` et `stderr` sont 3 constantes de type `FILE *` représentant les **descriptions internes de fichiers** associées aux fichiers standards entrée, sortie et erreur.
- Ces 3 constantes désignent des **entrées dans une table de descripteurs de fichiers** dont les numéros sont respectivement 0, 1 et 2. Le prochain descripteur "ouvert" par le programme aura le numéro 3 et ainsi de suite ...
- D'autres constantes et types sont aussi utilisés :

`EOF` est une valeur renvoyée par les routines d'entrée dès la rencontre d'une fin de fichier ou d'une erreur. Elle a généralement la valeur `-1`.

`NULL` est une valeur renvoyée par les fonctions dont le résultat est un pointeur pour indiquer qu'une erreur s'est produite.

`BUFSIZE` est la taille d'un tampon d'E/S, habituellement 512.

`FILE` est un type prédéfini pour désigner un *fichier*.

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Portabilité

- Les **appels conformes** à la **norme ANSI C (C89)** pour la gestion des E/S fichiers sont : `fopen`, `fread`, `fwrite`, `fclose`, ...
- Il existe aussi des **appels bas niveaux** propres aux systèmes d'exploitation (*Operating System*). On peut citer :

  OS POSIX (dont Unix/Linux) : les appels systèmes `open`, `read`, `write`, `close`, `ioctl`, `fcntl`, ...

  OS API Win32 (Windows) : les fonctions `CreateFile`, `ReadFile`, `WriteFile`, `CloseHandle`, ...
- Bien évidemment, des **bibliothèques (frameworks)** de développement fourniront des **services de haut niveau** comme `QFile` (pour **Qt**), `TFileStream` (pour **Builder**), `fstream` (**C++**), ...

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Ouverture de fichier

- L'**association entre nom logique et nom physique** s'effectue à l'**ouverture du fichier**.
- Le **nom logique du fichier est en fait un pointeur sur une structure de type FILE**.
- Le **nom physique du fichier est une chaîne de caractères contenant son nom** et éventuellement son chemin dans l'arborescence du système de fichiers géré par l'OS.
- La fonction permettant d'**ouvrir un fichier physique** `fichier` est :

```
FILE *fopen(const char *fichier, const char *mode) ;
```

- Elle renvoie un **pointeur (fichier logique) vers le fichier ouvert** et `NULL` en cas d'échec.

Modes d'ouverture

- La valeur de `mode` détermine le mode d'ouverture du fichier :
  - r : Ouvre le fichier en lecture. Le pointeur de flux est placé au début du fichier.
  - r+ : Ouvre le fichier en lecture et écriture. Le pointeur de flux est placé au début du fichier.
  - w : Ouvre le fichier en écriture. Le pointeur de flux est placé au début du fichier.
  - w+ : Ouvre le fichier en lecture et écriture. Le fichier est créé s'il n'existait pas. S'il existait déjà, sa longueur est ramenée à 0. Le pointeur de flux est placé au début du fichier.
  - a : Ouvre le fichier en ajout (écriture à la fin du fichier). Le fichier est créé s'il n'existait pas. Le pointeur de flux est placé à la fin du fichier.
  - a+ : Ouvre le fichier en lecture et ajout (écriture en fin de fichier). Le fichier est créé s'il n'existait pas. La tête de lecture initiale du fichier est placée au début du fichier mais la sortie est toujours ajoutée à la fin du fichier.

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Le mode binaire

- La valeur de `mode` peut également inclure la lettre "b". Ce mode (qui n'a probablement aucun effet) sert uniquement à assurer la compatibilité avec C89 car il existe deux façons de considérer le contenu d'un fichier : sous forme **binaire** (*data*) ou en **texte** (*text*) qui est le choix par défaut.
- *Remarque :* Le "b" est ignoré sur tous les systèmes compatibles POSIX, y compris Linux. Mais d'autres systèmes peuvent traiter les fichiers texte et les fichiers binaires différemment, et l'ajout du "b" peut être une bonne idée si vous faites des entrées-sorties binaires et que votre programme risque d'être porté sur un environnement non-Unix.

Fermeture de fichier

- Si `fopen` permet d'ouvrir un fichier, une fois vos traitements sur ce fichier terminés, il vous faudra **fermer l'accès à cette ressource ouverte** en utilisant la fonction : `int fclose(FILE *flux) ;`
- Cette fonction permet de **fermer un fichier logique** `flux`. Elle renvoie la valeur 0 si tout se passe bien et `EOF` sinon.

Tampon mémoire (*buffer*)

- La gestion de fichier entre votre programme et le système d'exploitation met en oeuvre des **tampons mémoires pour optimiser les accès disques physiques**.
- Vous pouvez intervenir sur les choix par défaut en appelant la fonction : `void setbuf(FILE *flux, char *tampon);`
- Cette fonction, si le paramètre tampon est `NULL`, supprime l'utilisation d'un tampon pour le flux concerné.
- Dans le cas contraire, elle permet de forcer le système à utiliser un tampon d'adresse `tampon`.
- *Remarque :* en C++, on utilisera un objet de type `filebuf` pour gérer le tampon mémoire associé à un fichier.

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Exemple simple d'ouverture/fermeture de fichier en C

```c
#include <stdio.h>
#include <stdlib.h> /* pour exit */

int main() {
  FILE *fi;
  fi = fopen("essai.txt", "r");
  // attitude obligatoire : tester si l'ouverture a réussie
  if (! fi) {
    printf("Ouverture en lecture du fichier impossible !\n");
    perror("fopen"); // affichons le message d'erreur système
    exit(-1); // cela ne sert à rien de continuer, non ?
  }
  else printf("Ouverture en lecture du fichier réussie !\n"); // message
       inutile !
  // etc ...
  fclose(fi); // on ferme avant de quitter
  return 0;
}
```

Plusieurs situations à envisager

- Le fichier n'existe pas :

```
$ ./a.out
Ouverture en lecture du fichier impossible !
fopen: No such file or directory
```

- Le fichier existe mais mon programme n'a pas les droits d'accès en lecture :

```
$ ls -l essai.txt
-rw-r----- 1 root root 0 2012-04-01 08:57 essai.txt
$ ./a.out
Ouverture en lecture du fichier impossible !
fopen: Permission denied
```

- Le fichier existe avec les droits suffisants et l'ouverture réussie :

```
$ ls -l essai.txt
-rw-r----- 1 tv tv 0 2012-04-01 08:57 essai.txt
$ ./a.out
Ouverture en lecture du fichier réussie !
```

Les fichiers en C++

- C++ fournit les **classes** suivantes pour gérer des E/S sur les fichiers :
  - `ofstream` : classe *stream* pour **écrire** sur les fichiers
  - `ifstream` : classe *stream* pour **lire** à partir de fichiers
  - `fstream` : classe *stream* à la fois **lire et écrire** à partir de / vers des fichiers.
- Ces classes sont dérivées directement ou indirectement des classes `istream` et `ostream`. On a déjà utilisé des objets dont les types étaient de ces classes : `cin` (un objet de la classe `istream`) et `cout` (un objet de classe `ostream`). On pourra donc utiliser les flux de fichiers de la même façon.

Ouverture de fichier en C++ (1/3)

- La première opération généralement effectuée sur un objet de l'une de ces classes est de l'associer à un **fichier physique**.
- Cette procédure est connue sous le nom d'"**ouverture de fichier**". Un fichier ouvert est représenté au sein d'un programme C++ par un **objet de flux** (une instance d'une de ces classes) et toute opération d'entrée ou de sortie effectuée sur cet objet flux sera appliquée au fichier physique qui lui est associé.
- Pour cela, on utilise :

```
void open(const char * filename, ios_base::openmode mode);
```

- *Important :* bien refermer le fichier dès que possible soit par la **destruction de l'instance** de `ifstream` ou de `ofstream` soit en appelant directement `close()`.

Ouverture de fichier en C++ (2/3)

- `mode` décrit le mode E/S demandé pour le fichier physique `filename`. Il s'agit d'un objet de type `ios_base::openmode`, lequel consiste en une combinaison (avec un **OU bit à bit** soit l'opérateur |) d'un ou plusieurs des **drapeaux** suivants, définis comme des constantes membres :

app (*append*) : définit l'indicateur de position du flux à la fin du fichier avant chaque opération de sortie.

ate (*at end*) : règle l'indicateur de position du flux à la fin du fichier à l'ouverture.

binary (*binary*) : considère le contenu du fichier sous forme **binaire** sinon en **texte** (*text*) qui est le choix par défaut.

in (*input*) : autorise les opération de lecture sur le flux (choix par défaut pour `ifstream`).

out (*output*) : autorise les opération d'écriture sur le flux (choix par défaut pour `ofstream`).

trunc (*truncate*) : perte du contenu courant en supposant une longueur de zéro lors de l'ouverture.

Ouverture de fichier en C++ (3/3)

- Il est conseillé avant toute opération sur le flux de **tester les indicateurs d'état en appelant une de ces méthodes** :
  - `bad()` : Renvoie `true` si une lecture ou d'écriture opération échoue. Par exemple dans le cas que nous essayons d'écrire dans un fichier qui n'est pas ouvert en écriture.
  - `fail()` : Retourne la valeur `true` dans les mêmes cas que `bad()`, mais aussi dans le cas où une erreur de format se produit, comme quand un caractère est extrait et on essaye en fait de lire un nombre entier.
  - `eof()` : Renvoie `true` si un fichier ouvert en lecture a atteint la fin.
  - `good()` : C'est le drapeau d'état le plus générique. Il retourne `false` lorsque les méthodes précédentes retourneraient `true`.
- Pour les opérations de lecture, on peut soit utiliser l'opérateur de flux » soit les fonctions `get()`, `getline()` et `read()` pour `ifstream`.
- Pour les opérations d'écriture, on peut soit utiliser l'opérateur de flux « soit les fonctions `put()` et `write()` pour `ofstream`.

Exemple simple d'ouverture/fermeture de fichier en C++

```
#include <iostream>
#include <fstream> /* pour ifstream */
using namespace std;

int main() {
  ifstream ifile("essai.txt"); // équivalent à :
  //ifstream ifile("essai.txt", ifstream::in);
  if (! ifile.is_open()) {
    cout << "Ouverture en lecture du fichier impossible !\n";
    return -1;
  }
  else cout << "Ouverture en lecture du fichier réussie !\n";
  // etc ...
  ifile.close();
  return 0;
}
```

Fonctions d'E/S en C (1/2)

- Les E/S caractères :

```
int fgetc(FILE *flux) // Cette fonction lit un caractère du flux indiqué.
char *fgets(char *ch, int n, FILE *flux) // Cette fonction lit une chaîne
     de caractère du flux indiqué.
int fputc(int c, FILE *flux) // Cette fonction écrit un caractère sur le
     flux indiqué.
int fputs(const char *ch, FILE *flux) // Cette fonction écrit une chaîne de
      caractère sur le flux indiqué.
```

- Les E/S formatées :

```
int fscanf(FILE *flux, const char *format, arg1, ..., argn) // Cette
     fonction est identique à scanf, mais substitue à stdin le flux indiqué

int fprintf(FILE *flux, const char *format, arg1, ..., argn) // Cette
     fonction est identique à printf, mais substitue à stdout le flux
     indiqué
```

Fonctions d'E/S en C (2/2)

- Les E/S "objets" :

```
size_t fread(void *zone, size_t n, size_t nb, FILE *flux) // Cette fonction
     lit, sur le flux indiqué, au maximum nb blocs (enregistrement) de n
    octets chacun et les range en mémoire à partir de l'adresse zone. Elle
     retourne le nombre de blocs effectivement lus.

size_t fwrite(const void *zone, size_t n, size_t nb, FILE *flux) // Cette
    fonction écrit, sur le flux indiqué nb blocs (enregistrement) de n
    octets à partir de l'adresse zone. Elle retourne le nombre de blocs
    effectivement écrits.
```

Exemple simple d'écriture dans un fichier en C

```c
#include <stdio.h>

int main() {
  FILE *fo;
  char msg[16] = "Hello world !";

  fo = fopen("essai.txt", "w");
  if (! fo) {
    printf("Ouverture en écriture du fichier impossible !\n");
    perror("fopen");
    return -1;
  }
  else printf("Ouverture en écriture du fichier réussie !\n");

  fprintf(fo, "%s\n", msg);

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

Résultat de l'écriture dans un fichier en C

- Ce qui donne :

```
$ ./a.out
Ouverture en écriture du fichier réussie !

$ ls -l essai.txt
-rw-r----- 1 tv tv 14 2012-04-01 09:29 essai.txt

$ cat essai.txt
Hello world !
```

- Supposons maintenant que le fichier `essai.txt` contienne la chaîne de caractère de description d'un joueur :

```
$ cat essai.txt
Michael Jordan 23 1.98
```

Exemple de lecture de données formatées en C

```c
#include <stdio.h>
int main() {
  FILE *fi;
  char nom[16]; char prenom[16]; int numero; float taille;
  fi = fopen("essai.txt", "r");
  if (! fi) {
    printf("Ouverture en lecture du fichier impossible !\n");
    perror("fopen");
    return -1;
  }
  fscanf(fi, "%s %s %d %f\n", prenom, nom, &numero, &taille);
  printf("Données lues dans le fichier :\n");
  printf("Joueur -> %s %s\n", nom, prenom); // Affiche : Joueur ->
      Jordan Michael
  printf("Numéro -> %d\n", numero); // Affiche : Numéro -> 23
  printf("Taille -> %.2f m\n", taille); // Affiche : Taille -> 1.98 m
  fclose(fi);
  return 0;
}
```

Fonctions d'E/S en C++ (1/2)

- Les E/S caractères :

```
bool istream::get(char& c); // Lit un seul caractère et le placer dans c
istream::getline(char *s, int nmax); // Lit une ligne de moins de (nmax-1)
    lettres, et la place dans s
// Si la ligne est plus longue, ce qui dépasse est perdu (mais ne provoque
    pas de faute de mémoire).
ostream::put(const char&); // Écrit le caractère c
isteam::putback(const char&); // Remet un caractère c dans le flux de façon
     à ce qu'il soit le prochain caractère lu
```

- Les E/S "objets" :

```
istream& read(char* s, streamsize n); // Cette fonction lit sur le flux
    ouvert de n octets les range en mémoire à partir de l'adresse s.

ostream& write(const char* s , streamsize n); // Cette fonction écrit sur
    le flux ouvert n octets à partir de l'adresse s.
```

Fonctions d'E/S en C++ (2/2)

- Les E/S formatées :

```
ostream& operator<< (ostream& out, char c); // existe aussi pour signed et
    unsigned
ostream& operator<< (ostream& out, const char* s); // existe aussi pour
    signed et unsigned
ostream& operator<< (bool val);
ostream& operator<< (int val); // existe aussi pour short et long (signed
    et unsigned)
ostream& operator<< (float val); // existe aussi pour double et long double
// et beaucoup d'autres encore...

stream& operator>> (istream& is, char& ch); // existe aussi pour signed et
    unsigned
// etc ...
```

Exemple simple d'écriture dans un fichier en C++

```
#include <iostream>
#include <fstream> /* pour ofstream */
using namespace std;

int main() {
   ofstream ofile("decompte.txt");

   for(int i=10;i>0;i--)
      ofile << i << endl;
   ofile << "boum !" << endl;

   return 0;
} // fermeture du fichier lors de la destruction de ofile.
```

Notion d'enregistrement

- La notion d'enregistrement n'existant pas en C, elle peut néanmoins être mise en oeuvre à l'aide des fonctions `fread` et `fwrite`.
- On aura l'habitude de **définir un enregistrement à partir d'un type structuré**.
- Prenons comme exemple la structure suivante :

```
// Ceci représentera un "enregistrement" :
struct compte {
   char nom[16];
   int uid; // identifiant d'utilisateur
   int gid; // identifiant de groupe
};
```

Exemple simple d'écriture d'enregistrement en C

```
int main() {
  FILE *fo;
  struct compte toto = { "toto", 500, 500 };
  int ret;

  fo = fopen("essai.dat", "w");
  if (! fo) { perror("fopen"); exit(-1); }

  printf("Taille de l'enregistrement : %d octets\n", sizeof(struct
      compte));

  ret = fwrite(&toto, sizeof(struct compte), 1, fo);
  if(ret != 1) { perror("fwrite"); fclose(fo); exit(-2); }
  else printf("Écriture de l'enregistrement réussie.\n");

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

Exemple simple d'écriture d'enregistrement en C++

```
int main() { ofstream ofile("essai.dat");
  struct compte toto = { "toto", 500, 500 };
  if (! ofile.is_open()) {
    cout << "Ouverture en écriture du fichier impossible !\n";
    return -1;
  }
  cout << "Taille de l'enregistrement : " << sizeof(struct compte) << "
      octets\n";
  ofile.write((char *)&toto, (streamsize)sizeof(struct compte));
  if(ofile.fail()) {
     cout << "Erreur lors de l'écriture de l'enregistrement dans le
         fichier !\n";
     ofile.close();
     return -2;
  }
  ofile.close();
  return 0;
}
```

Résultat pour l'écriture d'un enregistrement en C/C++

- Analysons le résultat de cette exécution :

```
$ ./a.out
Taille de l'enregistrement : 24 octets
Écriture de l'enregistrement réussie.

$ ls -l essai.dat
-rw-r--r-- 1 tv tv 24 2012-04-01 09:48 essai.dat

$ hexdump -C essai.dat
00000000 74 6f 74 6f 00 00 00 00 00 00 00 00 00 00 00 00 |toto............|
00000010 f4 01 00 00 f4 01 00 00
```

- En examinant le contenu du fichier, on trouve :
  - les **16 octets** du champ **nom** (`char [16]`) qui contient la chaîne ASCII "toto"
  - les **4 octets** pour le champ **uid** (`int`) codé en *little-endian* ici : `f4 01 00 00` soit 0x000001f4 donc 500 en décimal
  - les **4 octets** pour le champ **gid** (`int`) codé en *little-endian* ici : `f4 01 00 00` soit 0x000001f4 donc 500 en décimal
- Le fichier `essai.dat` est donc considéré comme un **fichier "binaire"**.

Remarques sur les enregistrements

- *Remarque n° 1 :* évidemment le programme de lecture de ce fichier DEVRA connaître la structure de l'enregistrement. Ce sera le cas pour tous les fichiers "binaires" qui possèdent un format (mp3, bmp, etc ...). Pour un programmeur, un **format de fichier** c'est connaître sa structure pour pouvoir le manipuler.
- *Remarque n°2 :* notre fichier ne contient (pour l'instant) qu'un SEUL enregistrement. Lorsqu'il aura plusieurs enregistrements, il sera nécessaire pour le manipuler de posséder des **fonctions de positionnement** !
- Les fonctions vues précédemment sont associées à un **pointeur de flux qui contient la position courante** dans le fichier. Celui-ci est initialisé à l'ouverture du fichier selon le mode choisi. Puis, il est positionné automatiquement en fonctions des opérations effectuées sur ce fichier.

Fonctions de positionnement en C

- Il est possible de manipuler le positionnement dans le fichier avec les fonctions suivantes :

```
int fgetpos(FILE *flux, fpos_t *position) // Cette fonction range à l'
    adresse position, la valeur courante du pointeur de fichier relatif au
     flux spécifié. Elle retourne 0 en cas de succès et une valeur
    négative en cas d'échec.
int fsetpos(FILE *flux, const fpos_t *position) // Cette fonction place le
    pointeur de fichier flux à l'adresse définie par position. Elle
    retourne 0 en cas de succès et une valeur négative en cas d'échec.
int fseek(FILE *flux, long deplacement, int origine) // Cette fonction
    positionne le pointeur de fichier flux à l'endroit indiqué par
    deplacement par rapport à origine. Le déplacement est exprimé en
    octets et origine peut avoir l'une des valeurs suivantes : SEEK_SET (
    correspond au début du fichier), SEEK_CUR (la position actuelle du
    pointeur de fichier) ou SEEK_END (la fin du fichier). fseek retourne 0
     en cas de succès et une valeur négative en cas d'échec.
long ftell(FILE *flux) // Cette fonction renvoie la position courante, en
    octets, du flux indiqué. Elle retourne -1 en cas d'erreur.
void rewind(FILE *flux) // Cette fonction place le pointeur de fichier flux
     à son début.
```

Fonctions de positionnement en C++

- Il est possible de manipuler le positionnement dans le fichier avec les fonctions suivantes :

```
istream& seekg(streampos pos) // Cette fonction positionne le pointeur de
    flux à la nouvelle position pos. seekg retourne *this.

istream& seekg(streamoff off, ios_base::seekdir dir) // Cette fonction
    positionne le pointeur de flux à l'endroit indiqué par off par rapport
     à dir. Le déplacement (offset) est exprimé en octets et dir peut
    avoir l'une des valeurs suivantes : ios::beg (correspond au début du
    fichier), ios::cur (la position actuelle du pointeur de fichier) ou
    ios::end (la fin du fichier). seekg retourne *this.

streamsize tellg() // Cette fonction renvoie la position courante, en
    octets, du flux indiqué. Elle retourne -1 en cas d'erreur.
```

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Exemple pour déterminer la taille d'un fichier en C

```c
int main() { FILE *fi; long taille;
  // Ouvre le fichier en lecture. Le pointeur de flux est placé au début du
       fichier.
  fi = fopen("/etc/passwd", "r");
  if (! fi) { perror("fopen"); exit(-1); }

  // Le pointeur de flux est donc au début du fichier
  // Alors, plaçons nous à la fin du fichier (sans connaître sa taille !)
  // Donc, il faut faire un déplacement de 0 octets à partir de la fin :
  fseek(fi, 0, SEEK_END);

  // Puis on appelle ftell qui renvoie la position courante, en octets donc sa
       taille
  taille = ftell(fi);
  printf("Taille du fichier : %ld octets\n", taille);

  // Attention le pointeur de flux est maintenant positionné en fin de fichier
  // Il faudrait peut-être le remettre au début :
  rewind(fi); // ou : fseek(fi, 0, SEEK_SET);
  fclose(fi);
  return 0;
}
```

Exemple pour déterminer la taille d'un fichier en C++

```
int main() {
  ifstream ifile("/etc/passwd"); long taille;
  if (! ifile.is_open()) {
    cout << "Ouverture en lecture du fichier impossible !\n";
    return -1;
  }
  // Donc Le pointeur de flux est placé au début du fichier
  // Plaçons nous à la fin du fichier (mais on ne connaît pas sa taille !)
  // Donc, il faut faire un déplacement de 0 octets à partir de la fin
  ifile.seekg(0, ios::end);
  // Puis on appelle tellg qui renvoie la position courante, en octets
  // donc sa taille !
  taille = ifile.tellg();
  cout << "Taille du fichier : " << taille << " octets\n";
  // Attention le pointeur de flux est maintenant positionné en fin de fichier
  // Il faudrait peut-être le remettre au début :
  ifile.seekg(0, ios::beg);
  // ...
  ifile.close();
  return 0;
}
```

Fin de fichier en C

- C'est la question que l'on se pose le plus souvent lorsqu'on manipule un fichier en lecture : **ai-je atteint la fin du fichier ?**
- La bibliothèque d'Entrée/Sortie standard fournit la fonction : `int feof(FILE *flux)`
- Cette fonction teste une fin de fichier. Elle ne peut s'appliquer qu'à un fichier déjà ouvert. Elle renvoie une valeur différente de zéro (pas obligatoirement `EOF`) si une fin de fichier s'est produite et `0` sinon. Son résultat n'est valide que **si le fichier a été ouvert en lecture et que la dernière opération effectuée sur ce fichier était une lecture**.
- L'indicateur de fin de fichier ne peut être réinitialisé que par la fonction `clearerr()`.
- *Attention :* `fread()` traite la fin du fichier comme une erreur, il faut donc lire APRES la fin du fichier pour provoquer cette erreur et détecter la fin du fichier.

Exemple pour tester la fin de fichier en C

```
int main() { FILE *fi; struct compte user; int ret = 0; int nb = 0;
  ...
  // Il y a peut-être d'autres enregistrements, continuons à lire
  // tant que l'on a pas atteint la fin du fichier
  while(!feof(fi))
  {
      ret = fread(&user, sizeof(struct compte), 1, fi);
      // a-t-on lu un enregistrement ?
      if(ret == 1) {
         printf("Enregistrement lu :\n");
         printf("Nom : %s - UID : %d - GID : %d\n",user.nom,user.uid,user.gid);
         nb++;
      }
  }

  printf("Nombre d'enregistrements lus : %d\n", nb);
  fclose(fi);
  return 0;
}
```

Exemple pour tester la fin de fichier en C++

```
int main() { ifstream ifile("essai.dat"); struct compte user; int ret = 0;
  int nb = 0;
  ...
  // Il y a peut-être d'autres enregistrements, continuons à lire
  // tant que l'on a pas atteint la fin du fichier
  while(!ifile.eof()) {
      ifile.read ((char *)&user, (streamsize)sizeof(struct compte));
      // a-t-on lu un enregistrement ?
      if(!ifile.fail()) {
         cout << "Enregistrement lu :\n";
         cout << "Nom : " << user.nom << " - UID : " << user.uid << " - GID : "
              << user.gid << endl;
         nb++;
      }
  }

  cout << "Nombre d'enregistrements lus : " << nb << endl;
  ifile.close();
  return 0;
}
```