# Les fichiers en C.

**Patrick Poulingeas.**

- On utilise les fonctions de la bibliothèque standard du C pour gérer les fichiers (et non pas celle de C++).
  On ajoutera en début de programme :
  *#include <cstdio>*
  Signification du nom 'cstdio' :
  c : bibliothèque C employée dans un programme C++
  stdio : standard input/output

- Les fichiers binaires : ce sont des fichiers constitués d'une suite d'octets.
  Les fichiers textes : ils sont constitués d'une suite de caractères (codés en ASCII), certains caractères ayant une signification spéciale (notamment pour indiquer une fin de ligne). On peut lire ces fichiers dans des éditeurs de texte comme emacs ou Kwrite, les fournir en argument à de nombreuses commandes Unix (less, wc, etc.).
  On peut écrire un programme C/C++ qui traite un fichier texte comme un fichier binaire (la suite de caractères étant alors vue comme une simple suite d'octets). L'inverse est également possible mais risque grandement d'aboutir à une erreur fatale.

- Déclaration d'une variable de type fichier (binaire ou texte) :
  *FILE *fichier;*
  'fichier' est le nom de la variable.

Attention aux majuscules pour 'FILE'. Contrairement à Turbo Pascal, C et C++ font la distinction entre minuscules et majuscules (On dit qu'ils sont « sensibles à la casse »).

- Ouverture d'un fichier :
  *fichier = fopen(nom_fichier,mode) ;*

  nom_fichier est une variable de type chaîne de caractères.
  mode est une variable de type chaîne de caractères.

  Les différents modes possibles :
  - « r » : ouverture en lecture.
  - « r+ » : ouverture en lecture/écriture.
  - « w » : ouverture en écriture. Si le fichier existe déjà, son contenu est effacé. Si le fichier n'existe pas, il est créé.
  - « w+ » : comme « w », mais les opérations de lecture sont aussi permises.
  - « a » : ouverture en écriture pour des ajouts en fin de fichier. Si le fichier n'existe pas, il est créé.
  - « a+ » : comme « a », avec en plus la possibilité de faire des lectures.

S'il y a un problème à l'ouverture (par exemple on essaie d'ouvrir en lecture un fichier qui n'existe pas), la variable fichier vaut NULL (qui est une constante prédéfinie). Si l'opération d'ouverture se passe bien, on a : fichier ≠ NULL.

Quand on veut exploiter un fichier sous forme binaire, il faut rajouter la lettre 'b' à la fin de la chaîne décrivant le mode (Exemple : « rb », « w+b », etc.), sinon le fichier doit être traité comme un fichier texte. Sous un système d'exploitation Unix (comme Linux), on peut se passer de cela : on ouvre le fichier et on applique les fonctions correspondant au format sous lequel on souhaite traiter le fichier (format binaire ou format texte).

- Fermeture d'un fichier :
  *fclose (fichier) ;*

- Test de fin de fichier :
  *feof (fichier)*

Cette fonction renvoie une valeur non nulle si l'on a tenté une opération de lecture après avoir lu le dernier élément du fichier.

Notez qu'il ne suffit pas d'arriver en fin de fichier pour que feof(fichier) ≠ 0, il faut en plus procéder à une tentative de lecture (qui avortera, sans planter le programme) afin que feof renvoie une valeur non nulle signalant que l'on est à la fin du fichier.

- Lecture depuis un fichier binaire avec stockage du résultat dans une variable :
  *fread (&variable,sizeof(variable),1,fichier) ;*

&variable indique l'adresse d'une variable dans laquelle on va stocker une suite d'octets lus depuis le fichier.

sizeof(variable) (ou sizeof(type de la variable)) indique la taille en octets d'un élément que l'on va lire.

1 correspond au nombre d'éléments à lire.

Au total, on lit donc sizeof(variable)*1 octets depuis le fichier.

La fonction fread renvoie le nombre d'éléments effectivement lus (donc si l'on tente une lecture après le dernier élément du fichier, la fonction renverra 0).

- Ecriture d'une variable dans un fichier binaire :
  *fwrite (&variable,sizeof(variable),1,fichier) ;*

&variable est l'adresse d'une variable dont veut écrire le contenu dans le fichier.

sizeof(variable) (ou sizeof(type de la variable)) indique la taille en octets d'un élément que l'on veut écrire dans le fichier.

1 correspond au nombre d'éléments que l'on désire écrire dans le fichier.

Au total, on écrit donc sizeof(variable)*1 octets dans le fichier.

La fonction renvoie le nombre d'éléments écrits (qui peut-être inférieur au nombre demandé en cas d'erreur – comme une saturation du disque dur par exemple).

- Exemple d'écriture d'un enregistrement dans un fichier binaire :

```
struct compte
{
        // ...
} ;
```

```
FILE *fichier ;
compte un_compte ;

fichier = fopen(« comptes.dat », « wb ») ;
if (fichier == NULL)
        cout << « Probleme : impossible d'ouvrir en ecriture le fichier comptes.dat »
             << endl ;
else
{
        fwrite(&un_compte,sizeof(compte),1,fichier) ;
        fclose(fichier) ;
}
```

- Positionnement dans un fichier binaire :
  *fseek(fichier,deplacement,origine) ;*

'deplacement' indique la longueur du déplacement à effectuer en octets (ce nombre peut être négatif).
'origine' indique à partir d'où doit s'effectuer le déplacement.

Valeur possibles pour 'origine' :
SEEK_SET : déplacement par rapport au début du fichier,
SEEK_CUR : déplacement par rapport à la position courante dans le fichier,
SEEK_END : déplacement par rapport à la fin de fichier.

Exemple d'utilisation : revenir au début du fichier
*fseek(fichier,0,SEEK_SET) ;*

- Lecture dans un fichier texte avec stockage dans des variables :
  *fscanf(fichier_texte,format,&variable1,&variable2,…) ;*

'format' est une chaîne de caractères indiquant le nombre et le type des données que l'on souhaite lire et ranger dans les variables 'variable1', 'variable2', …
Par exemple, si l'on veut lire un entier et un réel depuis un fichier texte, on aura :

```
int i ;
float x ;

fscanf(fichier_texte, « %d%f »,&i,&x) ;
```

Dans la chaîne de format, %d désigne un entier en notation décimale et %f un nombre réel.

Lecture d'un caractère depuis un fichier texte :
*fscanf(fichier_texte, « %c »,&caractere) ;*

Quand on arrive en fin de fichier ou qu'il y a eu un problème empêchant toute lecture, la fonction fscanf renvoie la valeur EOF. Sinon, elle retourne le nombre de variables lues avec succès.

Remarque : il existe une fonction scanf qui effectue des lectures depuis l'entrée standard (habituellement : le clavier). C'est cette fonction que l'on utilise, à la place de cin, quand on souhaite faire des lectures en C (cin est un apport de C++). Comme scanf est plus complexe à manipuler que cin, avec notamment sa chaîne de format dont nous n'avons pas vu toutes les possibilités (ou obscurités… ;-), c'est la raison pour laquelle on a choisi cin dans la feuille de traduction Algorithmique → C/C++.

- Ecriture de variables dans un fichier texte :
  *fprintf(fichier_texte,format,variable1,variable2,…) ;*

'format' est toujours une chaîne de caractère dans laquelle on peut insérer des éléments spéciaux (similaires à ceux que l'on vient de voir pour la lecture : %d, %f, %c, etc.) pour indiquer l'insertion de la valeur d'une variable.

Exemple :
On a une variable entière i.
On veut écrire dans un fichier texte la chaîne : « La variable vaut : », suivie de la valeur de la variable i et d'un saut de ligne.
*fprintf(fichier_texte,«La variable vaut : %d\n »,i) ;*
\n signifie 'saut de ligne'.

Ecriture d'un caractère dans un fichier texte :
*fprintf(fichier_texte, « %c »,caractere) ;*

Remarque : On dispose en C d'une fonction printf qui sert à faire des écritures sur la sortie standard (en général, l'écran). C'est cette fonction que l'on utilise en C pour des affichages à l'écran, et non pas cout qui est une extension apportée par C++.