Chapitre 1

Les fichiers

En général, un fichier est une suite d'informations conservée sur un périphérique de stockage (disquette, disque dur, CDROM, bande). Les limitations technologiques de ces périphériques nécessitent une manipulation spéciale des informations qu'ils contiennent. En particulier, il est utile de faire apparaître la notion de tête de lecture/écriture.

Cette notion de fichier peut être étendue pour gérer d'autres flux d'informations (écran, clavier, ...) mais la manipulation des fichiers reste toujours la même.

On peut donc définir un fichier comme une séquence d'informations que l'on peut manipuler par l'intermédiaire d'une tête de lecture/écriture. Chaque fois que la tête lit ou écrit une information, elle passe automatiquement à l'information suivante pour être prête à la lire ou à l'écrire.

Chaque fichier a un début (la première information) et une fin (le vide après la dernière information). Entre ce début et cette fin se trouvent les informations que contient le fichier. Un fichier peut être vide, c'est à dire ne contenir aucune information. Il est possible sur certains fichiers de repositionner la tête de lecture sur n'importe quelle position entre le début et la fin du fichier (on parle de fichier à accès aléatoire). À l'inverse, certains fichiers permettent uniquement de se placer au début du fichier et de lire les informations, une par une, dans l'ordre, jusque la fin (on parle de fichier à accès séquentiel)

On peut accéder à un fichier soit en lecture seule, soit en écriture seule, soit enfin en lecture/écriture.

En C++, on peut utiliser les fichiers de C (type FILE *, utilise la notion de pointeur), ou les fichiers de C++ (types ofstream, ifstream, utilise la notion d'objet). On utilisera ici les fichiers de C++.

1.1 Les fichiers textes

Un fichier texte est un fichier qui contient des informations sous la forme de caractères (en général en utilisant le code ASCII). Le source d'un programme C++ est un exemple de fichier texte.

1.1.1 Avantages d'un fichier texte

- facile à manipuler
- facile à modifier (il suffit de prendre un éditeur de texte)
- facilement compréhensible par l'utilisateur (les données sont représentées sous leur forme textuelle)
- portable (transférable d'un ordinateur à un autre sans grande difficulté) (NB : des systèmes d'exploitation différents n'utilisent pas forcément le même codage des caractères (ASCII, EBCDIC, caractères accentués) ou utilisent des conventions différentes pour les sauts de ligne (LF sous UNIX, CR LF sous DOS/Windows, CR sur Macintosh) mais ces difficultés sont mineures)

1.1.2 Inconvénients d'un fichier texte

- perte de temps dans la transcription des informations en caractères
- une même information n'occupe pas toujours le même espace (ex : un entier peut avoir 1 ou plusieurs chiffres) ce qui implique un accès séquentiel aux informations

1.1.3 Fichiers textes standards

Tout programme C++ a automatiquement accès à trois fichiers textes (à condition toutefois d'inclure <iostream>)

– cout : (séquentiel, en écriture seulement)
  représente le périphérique d'affichage standard (normalement l'écran)
– cin : (séquentiel, en lecture seulement)
  représente le périphérique de saisie standard (normalement le clavier)
– cerr : (séquentiel, en écriture seulement)
  représente le périphérique d'affichage d'erreur standard (normalement l'écran)

Ces trois fichiers correspondent aux fichiers standards gérés par le système d'exploitation et qui peuvent être redirigés à volonté (cf cours de SE).

Le programmeur n'a pas besoin d'ouvrir ou de fermer ces fichiers.

1.1.4 Manipulation des fichiers textes

La manipulation s'effectue en plusieurs étapes.

Le cas d'un fichier en écriture seulement

1. déclaration d'une variable de type fichier (cette variable contiendra diverses informations, en particulier celles concernant la position de la tête d'écriture)

```
#include <iostream>
#include <fstream>

using namespace std;

ofstream f; // o pour output
```

2. désignation de l'emplacement où les données doivent être stockées (chemin du système d'exploitation) et choix du mode d'ouverture du fichier

```
f.open("/tmp/essai.txt"); // mode d'ouverture par défaut
// si le fichier n'existe pas sur le disque, il est créé,
// si il existe, son contenu est effacé !
```

3. test pour savoir si la désignation précédente s'est bien déroulée

```
if (!f.good())
{
  cerr << "Impossible d'écrire dans le fichier !\n";
  exit(1); // arrêt du programme avec un code d'erreur différent de 0
}
```

4. impression de données dans le fichier

```
for(int i=0;i<5;i++)
  f << i << "\n";
```

5. fermeture du fichier. Cette étape est indispensable pour ne pas perdre d'information. En effet, pour des raisons d'efficacité, les données ne sont pas transférées directement sur le disque mais restent en mémoire jusqu'à ce que l'on puisse transférer au moins un bloc. La fermeture force le transfert des dernières informations.

```
f.close(); // INDISPENSABLE DANS TOUS LES LANGAGES
// (mais C++ le fera automatiquement si on l'oublie)
```

Le cas d'un fichier en lecture seulement

1. déclaration d'une variable de type fichier (cette variable contiendra diverses informations, en particulier celles concernant la position de la tête de lecture)

```
#include <iostream>
#include <fstream>

using namespace std;

ifstream f; // i pour input
```

2. désignation de l'emplacement où les données doivent être lues (chemin du système d'exploitation) et choix du mode d'ouverture du fichier

```
f.open("/tmp/essai.txt"); // mode d'ouverture par défaut
```

3. test pour savoir si la désignation précédente s'est bien déroulée

```
if (!f.good())
{
  cerr << "Impossible de lire dans le fichier !\n";
  exit(1); // arrêt du programme avec un code d'erreur différent de 0
}
```

4. lecture de données à partir du fichier et détection de la fin de fichier

```
int i;
do
{
  f >> i; // les espaces sont automatiquement sautés, la lecture
          // de l'entier s'arrête au premier caractère qui ne peut
          // pas faire partie de la représentation de l'entier

  // ATTENTION ! : la détection de la fin de fichier ne s'effectue en
  // C/C++ que lorsque l'on essayé de lire une donnée et qu'il n'y a
  // plus rien à lire

  if (!f.eof())
    cout << i << " ";
}
while (!f.eof());
```

5. fermeture du fichier. Cette étape est indispensable.

```
f.close(); // INDISPENSABLE DANS TOUS LES LANGAGES
// (mais C++ le fera automatiquement si on l'oublie)
```

1.1.5 La fin de fichier

En C/C++, la fonction f.eof() ne renvoie vrai qu'à partir du moment où l'on a essayé de lire une information et où l'ordinateur s'est aperçu qu'il n'y avait plus rien à lire. Autrement elle renvoie faux, y compris au moment où l'on vient juste de lire le dernier caractère du fichier. C'est donc immédiatement APRÈS une tentative de lecture qu'il faut vérifier que l'on n'est pas arrivé à la fin du fichier.

1.1.6 Les modes d'ouvertures

Lors de l'ouverture d'un fichier, il est possible de préciser un deuxième paramètre qui est le mode d'ouverture (f.open(filename,mode)). Ce mode est constitué de une ou plusieurs des indications ci-dessous, séparées par un *ou bit à bit* (une seule barre verticale).

```
ios::app // ajout en fin de fichier (append)
ios::ate // at end (ouverture et positionnement à la fin du fichier)
ios::binary // fichier binaire et non pas texte
ios::in // ouverture en lecture
ios::out // ouverture en écriture
ios::trunc // vider le fichier lors de l'ouverture
           // (effacer le contenu mais pas le fichier)
```

Un ofstream a par défaut un mode d'ouverture `ios_base::out | ios_base::trunc` ce qui signifie ouverture en écriture et effacement du contenu du fichier. Un ifstream a par défaut un mode d'ouverture `ios_base::in` ce qui signifie lecture seule.

1.1.7 Lecture/écriture des types de base

L'écriture dans un fichier s'effectue comme l'affichage à l'écran avec l'opérateur $<<$, la lecture dans un fichier s'effectue comme la lecture au clavier avec l'opérateur $>>$ (le sens des flèches indique le sens de transfert des informations). Toute variable ou constante des types de base (bool, char, int, float, double, string,...) peut être écrite ou lue. Lorsqu'une variable est lue, le programme commence par sauter tous les caractères assimilables à un espace (espace, tabulation, fin de ligne,...) puis lit tous les caractères qui peuvent constituer une représentation ascii du type lu. La lecture s'arrête au premier espace ou au premier caractère qui ne peut pas faire partie de la représentation ascii du type lu (ce sera alors le prochain caractère à lire). Pour une string, la lecture s'arrête au premier espace.

Exemple :

En supposant que le fichier f contient `136n'est pas 136.01`

```
int i;
string s;
float r;

f >> i; // lit 136 et s'arrête au caractère 'n' qui ne peut pas
        // faire partie de la représentation de l'entier
f >> s; // lit "n'est" et s'arrête à l'espace
f >> s; // saute les espaces et lit "pas"
f >> r; // saute les espaces et lit 136.01
```

1.1.8 Lecture/écriture de types définis par l'utilisateur

Lorsque le programmeur définit ses propres types (structures ou tableaux), il a la possibilité de définir comment ils doivent être écrits à l'écran ou dans un fichier et lus à partir du clavier ou d'un fichier. Cela s'effectue en définissant des fonctions qui répondent exactement au modèle ci-dessous :

Pour l'écriture :

```
ostream &operator <<(ostream &s, MonType &v)
{
  // écriture du paramètre v dans le fichier s
  ...
  // return indispensable
  return s;
}
```

Pour la lecture :

```
istream &operator >>(istream &s, MonType &v)
{
  // lecture du paramètre v à partir du fichier s
  ...
  // return indispensable
  return s;
}
```

Exemple :

```
#include <iostream>
#include <fstream>

using namespace std;

typedef
  struct
  {
    float re,im;
  } complexe;
```

```
ostream &operator <<(ostream &s, complexe &c)
{
  // utilisation d'une présentation simple
  s << c.re;

  if (c.im>=0)
    s << "+";
  s << c.im << "*i";

  return s;
}

istream &operator >>(istream &s, complexe &c)
{
  char tmp;
  bool error=false;

  s >> c.re; // la lecture s'arrête au premier caractère qui ne
     // peut pas faire partie du réel (donc au + ou au
              // - qui sépare la partie réelle de la partie imaginaire)

  s >> c.im;

  // lire le "*i" qui termine le complexe
  s >> tmp;
  if (tmp!='*')
    error=true;
  else
  {
    s >> tmp;
    if (tmp!='i')
      error=true;
  }

  if (error)
  {
    cerr << "Le complexe n'a pas été écrit dans le bon format !\n";
    exit(1); // arrêt du programme
    // note : il faut normalement gérer plus finement l'erreur
    // sans nécéssairement arrêter le programme
  }

  return s;
}

int main()
{
  complexe c;
  ofstream ecr;
  ifstream lect;

  cout << "Tapez un complexe sous la forme 1+6*i :";
  cin >> c;
  cout << "Vous avez tapé " << c << "\n";

  cout << "Ecriture du complexe dans le fichier\n";
  ecr.open("fic.txt");
  if (!ecr.good())
  {
```

```
    cerr << "Impossible de créer le fichier !\n";
    exit(1);
  }
  ecr <<  c;
  ecr.close();

  cout << "Relecture du complexe à partir du fichier\n";
  lect.open("fic.txt");
  if (!lect.good())
  {
    cerr << "Impossible de créer le fichier !\n";
    exit(1);
  }
  lect >>  c;
  lect.close();

  cout << "J'ai relu " << c << "\n";
}
```

1.1.9 Les manipulateurs

Un manipulateur permet de modifier le format d'affichage ou de lecture

manipulateur	effet
boolalpha	les booléens sont représentés par les mots "true" et "false"
noboolalpha	les booléens sont représentés par 0 et 1
showbase	les nombres en octal sont précédés par 0, les nombres hexadécimaux par 0x
noshowbase	pas de préfixe devant les nombres en octal et en hexadécimal
showpoint	afficher les zéros à la fin
noshowpoint	ne pas afficher les zéros à la fin
showpos	mettre un signe + devant les nombres positifs
noshowpos	ne pas mettre un signe + devant les nombres positifs
skipws	sauter les espaces
noskipws	ne pas sauter les espaces
uppercase	utiliser des majuscules pour les nombre hexadécimaux et le E de exposant
nouppercase	utiliser des minuscules pour les nombre hexadécimaux et le e de exposant
setw(n)	affichage de la prochaine donnée sur n caractères (en rajoutant des espaces si nécessaire)
setfill(c)	utiliser le caractère c au lieu de l'espace pour le remplissage
left	alignement à gauche (le remplissage se fait à droite)
right	alignement à droite (le remplissage se fait à gauche)
internal	alignement interne (le remplissage se fait entre le signe et le nombre)
dec	décimal
hex	hexadécimal
oct	octal
setbase(n)	affichage en base n
fixed	notation en virgule fixe
scientific	notation avec mantisse et exposant
flush	(écriture) forcer la sauvegarde dans le fichier des informations qui seraient encore en mémoire
endl	(écriture) afficher une fin de ligne '\n' et flush
ends	(écriture) afficher une fin de chaîne '\0' et flush
ws	(lecture) sauter les espaces

Exemple d'utilisation : `cout << showbase << hex << 64 << endl << flush;` affichera 0x20 (20 en hexadécimal), passera à la ligne et enverra l'affichage à l'écran.

1.2 Les fichiers binaires

Un fichier binaire est un fichier qui contient directement la représentation mémoire des informations. Un float sera donc stocké dans un fichier binaire sous la forme des 4 octets qui constituent sa représentation dans la norme IEEE. Un fichier binaire peut aussi être vu comme une séquence d'octets qui dans tous les cas possède une struc-

ture, même si elle n'est pas apparente. Un exemple de fichier binaire est le résultat de la compilation d'un programme C++ (programme exécutable).

1.2.1 Avantages des fichiers binaires

- plus rapide (pas de conversion en ascii (cf procédure d'affichage d'un entier))
- accès direct aux données car longueur fixe des informations (pour accéder à la i-ème information, il suffit d'atteindre le i*taille(information) ème octet (temps presque constant))

1.2.2 Inconvénients d'un fichier binaire

- non éditable par l'utilisateur
- illisible pour l'utilisateur
- donnée non directement transférables d'un ordinateur à l'autre (problème du choix de représentation des entiers de plusieurs octets : poids faible en premier : little-endian (80x86,pentium), poids fort en premier : big-endian (MC68000,réseau IP,...), autre ordre...). Il faut alors prendre des précautions qui ne sont pas si simples.)
  Une toute petite procédure qui permet de déterminer le sexe de la machine (little/big endian) :

```
#include <string.h> // pour strcmp

void endian()
{
  static long int str[2] = { 0x41424344,0x0 }; /* ASCII "ABCD" */

  if (strcmp("DCBA", (char *) str) == 0)
    cout << "little-endian";
  else
    if (strcmp("ABCD",(char *)str) == 0)
      cout << "big-endian";
    else
      if (strcmp("BADC",(char *)str) == 0)
        cout << "PDP-endian";
}
```

1.2.3 Manipulation d'un fichier binaire

La manipulation d'un fichier binaire s'effectue presque comme celle d'un fichier texte. Les seules étapes qui diffèrent sont les étapes 2 et 4 (désignation du fichier et écriture ou lecture de données). On n'utilise plus $<<$ et $>>$ mais f.write(...) et f.read(...).

Le cas d'un fichier binaire en écriture seulement

1. déclaration d'une variable de type fichier

```
#include <iostream>
#include <fstream>

using namespace std;

ofstream f; // o pour output
```

2. désignation de l'emplacement où les données doivent être stockées (chemin du système d'exploitation) et choix du mode d'ouverture du fichier

```
f.open("/tmp/essai.txt",ios::out|ios::binary|ios::trunc);
// si le fichier n'existe pas sur le disque, il est créé,
// si il existe, son contenu est effacé !
```

3. test pour savoir si la désignation précédente s'est bien déroulée

```
if (!f.good())
{
  cerr << "Impossible d'écrire dans le fichier !\n";
  exit(1); // arrêt du programme avec un code d'erreur différent de 0
}
```

4. écriture d'une variable v (quel que soit son type)

```
f.write((char *)&v,sizeof(v));
```

5. fermeture du fichier. Cette étape est indispensable pour ne pas perdre d'information. En effet, pour des raisons d'efficacité, les données ne sont pas transférées directement sur le disque mais restent en mémoire jusqu'à ce que l'on puisse transférer au moins un bloc. La fermeture force le transfert des dernières informations.

```
f.close(); // INDISPENSABLE DANS TOUS LES LANGAGES
// (mais C++ le fera automatiquement si on l'oublie)
```

Le cas d'un fichier en lecture seulement

1. déclaration d'une variable de type fichier

```
#include <iostream>
#include <fstream>

using namespace std;

ifstream f; // i pour input
```

2. désignation de l'emplacement où les données doivent être lues (chemin du système d'exploitation) et choix du mode d'ouverture du fichier

```
f.open("/tmp/essai.txt",ios::in|ios::binary);
```

3. test pour savoir si la désignation précédente s'est bien déroulée

```
if (!f.good())
{
  cerr << "Impossible de lire dans le fichier !\n";
  exit(1); // arrêt du programme avec un code d'erreur différent de 0
}
```

4. lecture d'une variable v (quel que soit son type)

```
f.read((char *)&v,sizeof(v));
```

5. fermeture du fichier. Cette étape est indispensable.

```
f.close(); // INDISPENSABLE DANS TOUS LES LANGAGES
// (mais C++ le fera automatiquement si on l'oublie)
```

1.2.4 Accès aléatoire

Il est possible de positionner la tête de lecture/écriture à une position arbitraire du fichier. Cela est particulièrement utile dans un fichier binaire où les données ont en général toutes la même taille. En C++, il y a deux têtes : une tête de lecture (used to **g**et characters) et une tête d'écriture (used to **p**ut characters).

Les opérations suivantes permettent de manipuler ces têtes :

f.tellp()	fonction qui renvoie la position de la tête d'écriture (exprimée en nombre d'octets depuis le début du fichier, le premier octet ayant le numéro 0)
f.tellg()	fonction qui renvoie la position de la tête de lecture
f.seekp(déplacement,origine)	procédure qui change la position de la tête d'écriture
f.seekg(déplacement,origine)	procédure qui change la position de la tête de lecture

Pour les procédures, seekp et seekg, le déplacement représente un nombre d'octets à parcourir à partir de l'origine, qui peut valoir l'une valeurs suivantes :

ios : :beg	déplacement par rapport au début du fichier (le déplacement doit être positif ou nul)
ios : :end	déplacement par rapport à la fin du fichier (le déplacement doit être négatif ou nul)
ios : :cur	déplacement par rapport à la position courante