UNIVERSITÉ PARIS DESCARTES

SDA - Structures de Données et Algorithmes

Equipe pédagogique
Marie-José Caraty, Denis Poitrenaud, Julien Rossit,
Camille Kurtz, Jacques Alès-Bianchetti, Eloi Keita

T08 – Résumé E/S sur fichier
T10 – Lecture de caractère ou de chaîne
T11 et T13 – `read` et `write` de blocs
T12 et T13 – Accès direct en lecture et écriture
T14 – Etat de flot
T15 à T17 – [Code] Lecture, écriture et modification de fichier binaire
T19 – Principe de lecture d'un flot
T20 – [Code] Lecture robuste
T21 à T23 – Manipulateurs de flot et [Code]
T24 à T25 – [Code] Ecriture et lecture d'un fichier texte
T26 à T31 – Correspondance et [Code] en C

Cours n° 6
Les entrées-sorties

Bibliothèque C++ (**`iostream`**) – http://www.cplusplus.com/ref/iostream/
Bibliothèque standard C (**`stdio`**) – http://www.cppreference.com/stdio/

Sommaire

1. GENERALITES SUR LES ENTREES-SORTIES

Persistance des données

Durée de vie des données

limitée à l' **exécution du programme** qui les contient

Pour **sauvegarder** de façon **permanente** les données produites par programme (*i.e.* rendre disponible après la fin de l'exécution du programme)
une solution est d'utiliser les mémoires secondaires (disque dur, CD, clés, …) géré par le système de fichiers fourni par le système d'exploitation

Entrées-sorties

Lecture et **écriture** dans un **fichier** sont des opérations d'entrée-sortie
Le principe dans les entrées-sorties de haut niveau est de séparer
aspect logique (*i.e. primitives de manipulation des flots de données dans les programmes*)
aspect physique (*i.e. leur réalisation par le biais de périphériques particuliers*)

Entrées-sorties sur fichier : un flot connecté à un fichier

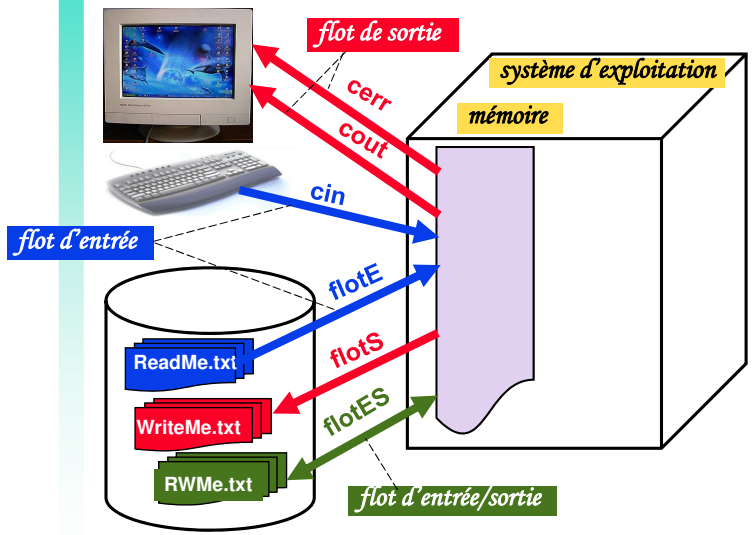
Flot entité **manipulable par programme** qui permet de **gérer les données** en **entrée** (du fichier vers la mémoire) et en **sortie** (de la mémoire vers le fichier)
Fichier **fichier physique** stocké sur une mémoire secondaire dont le nom physique est connu du système de fichiers

Sérialisation

Les fonctions de sérialisation sont dédiées à la sauvegarde/écriture et à la lecture de données d'un type donné sur un support non volatile (comme par exemple un fichier)

1. GENERALITES SUR LES ENTREES-SORTIES

Flots et périphériques

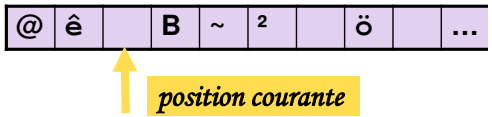



```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
  ifstream flotE;
  ofstream flotS;
  fstream flotES;

  flotE.open("ReadMe.txt");
  flotS.open("WriteMe.txt");
  flotES.open( "RWMe.txt ");

  float m, n, s;
  …
  …

  cin >> m >> n;
  …
  cerr << "Pb à l'ouverture";
  cout  << "Somme : "  << s;
  …
  …
}
```

Attributs de flot

Attributs d'un flot
- la **nature** de l'**entrée-sortie** (entrée, sortie, entrée et sortie)
- le **fichier** sur lequel elle porte
- la **mémoire tampon (zone mémoire)**
- la **position courante** dans le flot

tampon (d'octets)



Rôle de la mémoire tampon
Le **tampon** (zone mémoire) **optimise** (diminue) les **temps d'accès** aux périphériques (plus long, d'un facteur 1000, que l'accès en mémoire centrale)
L'écriture des données (envoi vers le périphérique) se fait lorsque le tampon est plein

Position courante dans le flot
Les flots sont à **accès séquentiel**
La **position courante** dans le flot indique la position à partir de laquelle s'effectuera
- l'insertion de données dans le flot (en cas d'écriture)
- l'extraction de données du flot (en cas de lecture)

A chaque **opération** d'écriture ou de lecture, la **position courante** est **incrémentée**

Fichier texte vs fichier binaire

123.458 à stocker dans un fichier
- *texte : stocké sur 7 octets*
- *binaire : stocké avec 4 octets*

`(float)` *ouvert sous éditeur, les octets sont interprétés suivant le code ASCII* `ëöB` *on lira 4 caractères :*

Fichier texte emploie les E/S formatées (<<, >>)
Stockage (écriture) originel dans le fichier de données sous le format ASCII

Avantage
- Ecriture et lecture par les outils classiques (more, type, wordpad, bloc-note)
- Lecture et écriture par programme

Désavantage
- Stockage volumineux (à associer avec un codeur compressif)
- Imprécision de données (e.g. réels)

Problème de portabilité
Le code de fin de ligne (utilisé dans les documents textes électroniques) n'est pas standard

		Code ASCII	
sous Windows	: \r\n	13 10	(CR LF)
sous Linux	: \n	10	(LF)
sous Mac	: \r	13	(CR)

Fichier binaire emploie les E/S non formatées (`read` et `write`)
Stockage (écriture) originel dans le fichier de données typées (type T prédéfini/utilisateur)
Les données sont interprétables par bloc d'octets (de taille sizeof(T))

Avantage
- Stockage compact
- Précision des données

Désavantage
- Nécessité d'écrire les programmes d'écriture et de lecture associée (illisible par les éditeurs classiques)

Etapes de mise en œuvre des E/S sur fichier

Utilisation des bibliothèques
iostream pour les entrées-sorties standards écran clavier
fstream pour les entrées-sorties fichier

1) Déclaration du flot
Déclaration d'une variable (logique) du type de flot (d'entrée ou de sortie) correspondant

2) Connexion-Ouverture du fichier
Connexion de la **variable** logique avec un **fichier physique** (dispositif d'entrée-sortie) géré par le système de fichiers

3) Traitement des entrées-sorties
Utilisation de la variable logique (liée au fichier) pour effectivement traiter les entrées-sorties (lecture ou écriture d'items formatés ou non, accès direct, …)

4) Déconnexion-Fermeture du fichier
Déconnexion de la variable logique au fichier physique

Etapes de mise en œuvre des E/S sur fichier

Inclusion des entêtes
```
#include <iostream
#include <fstream>
```

C++

1) Déclaration de flot
- d'entrée `ifstream streamName;`
- de sortie `ofstream streamName;`
- d'entrée/sortie `fstream streamName;`

Remarque :
```
istream cin;
ostream cout, cerr;
```

2) Connexion de flot-Ouverture de fichier
```
void open(const char* fileName, openmode mode);
```

3) Traitements des entrées-sorties (méthodes des classes ifstream et ofstream)
- entrée
  - `>>` lecture formatée
  - `get` lecture d'un caractère
  - `get, getline` lecture d'une chaîne de caractères
  - `read` opération binaire de lecture (lecture de bloc)
  - accès direct : `seekg` déplacement dans le fichier
  - `tellg` position courante dans le fichier
- sortie
  - `<<` lecture formatée
  - `put` écriture d'un caractère
  - `write` opération binaire d'écriture (écriture de bloc)
  - accès direct : `seekp` déplacement dans le fichier
  - `tellp` position courante dans le fichier

4) Fermeture du flot (pour tout mode d'ouverture)
```
void close();
```

Appel/invocation des méthodes `streamName.functionName(arg1, …);`

Inclusion des entêtes
```
#include <iostream
#include <fstream>
```

Ouverture de flot

C++

Mode d'ouverture d'un flot

flot d'entrée
flot de sortie
flot d'E/S

```
void open(const char* fileName, openmode mode=in);
void open(const char* filename, openmode mode=out|trunc);
void open(const char* filename, openmode mode=in|out);
```

openmode – vecteur d'état

`ios::in`	Fichier en lecture (flot d'entrée)
`ios::out`	Fichier en écriture (flot de sortie)
`ios::binary`	Fichier binaire (la fin de ligne est codée en fonction du système (`\r\n` sous Windows, \n sous Linux, \r sous Mac, ce qui pose le problème de portabilité)
`ios::app`	(append) Ouverture en ajout en fin de fichier (pour toute écriture)
`ios::ate`	(at end) Déplacement en fin de fichier (après ouverture)
`ios::trunc`	(truncate) Ecrase le fichier à l'ouverture (sans app et ate)

Remarque : Le paramètre mode a une valeur par défaut (`mode=<valParDefaut>`) à l'appel de l'ouverture d'un flot de sortie, sans paramètre effectif de mode, le mode d'ouverture est la sortie (`out`) et le fichier écrasé (`trunc`) s'il existe déjà

Exemple d'ouverture : flots d'entrée `flotE` et sortie `flotS` – Nom du fichier `"XFile"`

```
flotE.open("XFile"); //≡ flotE.open("XFile", ios::in);      fichier en lecture
flotS.open("XFile") //≡ flotS.open("XFile",) ios::out|ios::trunc);      fichier en écriture
flotS.open("XFile", ios::out|ios::app);                    ...
```


Extrait des méthodes de lecture – *ifstream* (1/3)

C++

Extrait des fonctions applicables aux flots d'entrée - ifstream

cf. Bibliothèque de iostream http://www.cplusplus.com/ref/iostream/

`>>` Opérateur d'extraction (donnée de type natif ou chaîne de caractères sans espace) du flot d'entrée

Lecture de caractères

```
int get();                          // char c=flotE.get()
istream& get ( char& c );           // char c; flotE.get(c);
istream& get ( char* s, streamsize n );
istream& get ( char* s, streamsize n, char delim='\n' );
```

Extrait les caractères du flot et les stocke dans le tableau de caractères pointé par s

Les caractères sont extraits jusqu'à ce que l'une des conditions soit satisfaite :
- (n-1) caractères sont extraits
- le **caractère délimiteur** (delim ou '\n') est trouvé et **n'est pas extrait du flot**
- la fin de fichier ou toute erreur de lecture du flux

à la fin, le caractère'\0' est automatiquement ajouté en fin de s

Retour : une référence au flot d'entrée concerné

```
istream& getline ( char* s, streamsize n, char delim='\n' );
```

Seule différence importante avec la spécification précédente:
- le **caractère délimiteur** (delim ou '\n') est trouvé et **extrait du flot**

Extrait des méthodes de lecture – *ifstream* (2/3)

C++

Extrait des fonctions applicables aux flots d'entrée - ifstream

cf. Bibliothèque de iostream http://www.cplusplus.com/ref/iostream/

Lecture de bloc

```
istream&  read (char* s, streamsize n );
```

Lit séquentiellement, à partir du flot d'entrée, un bloc de données de longueur `n` et le stocke dans un tableau de caractères pointé par `s`

***Remarque** : une conversion de type (char*) pourra être utilisée*

Les données sont extraites jusqu'à ce que l'une des conditions soit satisfaite :
- la longueur `n` est atteinte
- la fin de fichier est atteinte

Retour : une référence au flot d'entrée concerné

Extrait des méthodes de lecture – *ifstream* (3/3)

C++

Extrait des fonctions applicables aux flots d'entrée - ifstream

cf. Bibliothèque de iostream http://www.cplusplus.com/ref/iostream/

Positionnement - Accès direct

```
istream&  seekg ( streampos pos );
istream&  seekg ( streamoff off, ios::seekdir dir );
```

Déplace la position courante dans le flot d'entrée
***Rappel** : la position courante du flot d'entrée détermine la prochaine position à lire dans le buffer associé au flot*

1. à l'octet n° `pos`
2. d'une valeur (positive ou négative) `off` relative à la position `dir`

`dir` peut prendre les valeurs constantes
ios::beg (début de flot)
ios::cur (position courante du flot)
ios::end (fin de flot)

Retour : une référence au flot d'entrée concerné

Exemple : seekg(int n, ios::beg)
déplace la position courante de n octets après le début de flot

```
streampos  tellg ( );
```
Retourne la position courante dans le flot d'entrée

```
streamsize gcount ( );
```
Retourne le nombre d'octets lus

Extrait des méthodes d'écriture – *ofstream*

C++

Bibliothèque iostream http://www.cplusplus.com/ref/iostream/

<< Opérateur d'insertion (de donnée de type natif) dans un flot de sortie

Ecriture de caractères
```
ostream&  put ( char ch );
```

Ecriture de bloc
```
ostream&  write ( const char* str , streamsize n );
```
Ecrit séquentiellement, dans le flot de sortie, un bloc de données stocké dans le tableau (de caractères) pointé par **str** de taille **n**

***Remarque** : Aucune vérification du caractère de fin de chaîne*

Les données sont insérées dans le flot jusqu'à ce que l'une des conditions soit satisfaite :
- la longueur **n** est atteinte
- une erreur d'écriture dans le flot

Retour : une référence au flot de sortie concerné

Accès direct
```
ostream&  seekp ( streampos pos );
ostream&  seekp ( streamoff off, ios::seekdir dir );
streampos tellp ( );
```
cf. Spécifications de **seekg** et **tellg**. Substituer « flux de sortie » à « flux d'entrée »

Etat de flot – statut d'erreur

C++

Etat de flot composé de 4 bits dont 3 bits d'erreur

eofbit	activé si la fin de fichier est atteinte
failbit	activé si la dernière opération d'E/S a échoué
badbit	activé si la dernière opération d'E/S est invalide
hardfail	activé si si le flux est en état d'erreur

bits d'erreur

Fonctions booléennes permettant d'accéder aux bits d'erreur

eof()	retourne **true** si la fin de fichier est atteinte, **false** sinon
fail()	retourne **true** si la dernière opération d'E/S a échoué, **false** sinon
good() :	retourne vrai s'il aucun bit de l'état de flot n'est actif, faux sinon

Fonction permettant de positionner les bits de l'état de flot

clear() : désactive tous les bits de l'état de flot

Test de l'état d'un flot f (dont **cin**)

- **if (f.good())** **le flot est en état d'être lu**
- **if (f.fail())** **une erreur est détectée, le flot ne peut plus être lu, toute autre opération d'E/S échouera (format de donnée invalide, dépassement de capacité, …)**

Ecriture d'un fichier binaire de dates

C++

```
/** @file EcritureFBinaireDates.cpp - coursCpp/coursCpp2
 *  @author l'équipe pédagogique
 *  @brief Ecriture d'un fichier binaire de dates
 */

int main() {

   ofstream flotOut;
   flotOut.open("FileDates.bin", ios::out);

   if (flotOut.fail()) cerr << "Impossible d'écrire dans le fichier\n";
   else {
      Date d;
      cout << "Saisie de dates à enregistrer dans le fichier\n";
      cout << "(jusqu'à une année nulle non enregistrée)\n";
      do {
         d=saisir();
         if (d.annee!= 0)
            flotOut.write((char*) &d, sizeof(d));
      } while (d.annee != 0);

      flotOut.close();
      return 0;
   }
}
```

```
#include <iostream>
#include <fstream>
#include "Date.h"
using namespace std;
```

```
Saisie de dates à enregistrer dans le fichier
(jusqu'à une année nulle non enregistrée) :
Date (jour? mois? annee?) ? 14 7 1789
Date (jour? mois? annee?) ? 4 7 1783
Date (jour? mois? annee?) ? 1 8 2006
Date (jour? mois? annee?) ? 1 1 0
```


Lecture d'un fichier binaire de dates

C++

```
/** @file LectureFBinaireDates.cpp - coursCpp/CoursCpp3
 *  @brief Lecture d'un fichier binaire de dates
 */
int main() {
   cout << "Lecture du fichier binaire" <<endl;

   ifstream flotIn;
   flotIn.open("FileDates.bin", ios::in);

   if (flotIn.fail()) cerr << "Impossible de lire le fichier\n";
   else {
      Date d;

      /* calcul du nombre d'éléments de type Date dans le fichier */
      flotIn.seekg(0, ios::end); // déplacement en fin de flot
      int nbDates = flotIn.tellg()/sizeof(Date); // calcul nb de dates

      flotIn.seekg(0); // déplacement en debut de flot pour affichage
      cout << "Le fichier contient " << nbDates << " dates" << endl;

      /** lecture et affichage des dates du fichiers */
      for (int i = 0; i < nbDates; ++i) {
         flotIn.read((char*)&d, sizeof(Date));
         afficher(d);
      }
      cout << "\nFin de lecture" << endl;
      flotIn.close();
      return 0;
   }
}
```

```
#include <iostream>
#include <fstream>
#include "Date.h"
using namespace std;
```

```
Lecture du fichier binaire
Le fichier contient 3 dates
14/7/1789  4/7/1776  1/5/2006
Fin de lecture
```

Modification d'un fichier binaire de dates

C++

```
/** @file ModificationFBinaireDates.cpp – coursCpp/CoursCpp4
 * @brief Modification d'un fichier binaire de date
 */
int main() {
   cout << "Modification d'un fichier binaire\n(déjà
   fstream flotInOut;
   flotInOut.open("FileDates.bin", ios::in|ios::out);
   if (flotInOut.fail()) { cerr << "Impossible de lire le fichier\n";
                           exit(1); }
   Date d;
   /* calcul du nombre d'éléments de type Date dans le fichier */
   flotInOut.seekg(0, ios::end);
   int nbDates = flotInOut.tellg() / sizeof(Date);
   if (nbDates>0) {
      /* modification de l'élément médian (incréméntation de l'année) */
      flotInOut.seekg ((nbDates/2)*sizeof(Date), ios::beg);
      flotInOut.read((char*) &d, sizeof(Date));
      d.annee +=1;
      flotInOut.seekp ((nbDates/2)*sizeof(Date), ios::beg);
      flotInOut.write((char*) &d, sizeof(Date));
      flotInOut.flush();
   }
   /** lecture et affichage des dates du fichier */
   flotInOut.seekg(0, ios::beg);
   for (int i = 0; i < nbDates; ++i) {
      flotInOut.read((char*)&d, sizeof(Date)); afficher(d);
    }
   flotInOut.close(); return 0;
}
```

```
#include <iostream>
#include <fstream>
#include "Date.h"
using namespace std;
```

Positionnement à l'élément médian pour sa modification

```
Modification d'un fichier binaire
(déjà créé) de dates
14/7/1789  4/7/1777  1/5/2006
```


Lecture robuste (1/2)

C++

Problème

Lecture des données (avec le flot standard **cin**)
lorsque format du type attendu n'est pas valide

Exemple : Lire un nombre entier compris entre 1 et 10

```
main() {
   unsigned int nbLu;
   do {
      cout << "Entrez un nombre entre 1 et 10 : " << flush;
      cin >> nbLu;
   } while ((nbLu < 1) || (nbLu > 10));
   return 0;
}
```

ou un nombre négatif (non valide en raison du type `unsigned int`*)*

Si vous tapez un caractère ou un nombre négatif : une boucle infinie s'exécute

```
Entrez un nombre entre 1 et 10 : 0
Entrez un nombre entre 1 et 10 : f
Entrez un nombre entre 1 et 10 : Entrez un nombre entre 1 et 10 : .....
```

Principe de lecture robuste

qui permet d'éviter ce dysfonctionnement

- **contrôler l'état** du flot **cin**
- **éliminer** du flot **cin** les **données invalides**

Principe de lecture d'un flot

C++

Exemple : `cin >> i >> r;` // i (entier), r (réel)

Caractères séparateurs de données dans un flot (*whitespace*)
{ espace, tabulation, fin de ligne, retour chariot }

à la première requête : cin >> i

déplacement de la position courante après le(s) caractère(s) séparateur(s)
boucle (de lecture de la donnée)
| lecture du caractère courant **c**
| ***si*** (**c** n'est pas autorisé dans le format) ***alors***
| | remise du caractère **c** dans le flot
| | positionne un bit d'erreur (cin.fail() renvoie **true**)
| ***sinon si*** (**c** != caractère_séparateur) alors
| | mémorise **c**
| ***finsi***
finsi
tant que (**c** !=caractère_séparateur)
convertit la donnée (suite des caractères mémorisés) en sa valeur **v**
si (**v** est en dépassement de capacité) ***alors***
| positionne un bit d'erreur (cin.fail() renvoie **true**)
sinon affecte **v** à **i**
retourne le flot

Lecture robuste (2/2)

C++

Ecrire les fonctions de saisie robuste associées à vos entrées de programme. Les grouper dans un composant de lecture robuste

```
/**
 * @brief Saisie robuste d'un nombre naturel
 * @param[in] le message d'aide à la saisie
 * @return le nombre naturel saisi
 */
unsigned int saisirNombreNaturel(const char* msg) {
   unsigned int n;
   do {
      cin >> n;
      if (cin.fail()) {          // teste l'état de cin
         cin.clear();            // remets cin dans un état lisible
         cin.ignore(INT_MAX, '\n'); // ignore toute la ligne de données
         cout << msg;
         cin >> n;
      }
   } while (!cin);
   return n;
}

int main() {
   unsigned int nb;
   char msg[]= "Entrez un naturel : ";
   do {
      cout << "Nombre entre 1 et 10 ? " << flush;
      nb=saisirNombreNaturel(msg);
   } while ((nb < 1) || (nb > 10));
   cout << "Nombre lu : " << nb << endl;
   return 0;
}
```

```
Nombre entre 1 et 10 ? 0
Nombre entre 1 et 10 ? abcd
Entrez un naturel : -3 12
Entrez un naturel : 13
Nombre entre 1 et 10 ? 7
Nombre lu : 7
```

Manipulateurs et options de configuration (1/2)

Des formats de sortie peuvent être explicitement spécifiés :
– soit par des **manipulateurs** appliqués à l'instruction d'écriture dans le flot `<<`
– soit par des **options de configuration** pour une variable de type `ofstream` (dont `cout`)

Utilisation des manipulateurs

```
#include <iomanip>
cout << manipulateur << expression  …
cin << ws; // déplace la position courante dans le flot après les white spaces
```

Utilisation des options de format

```
setf(ios::option)          // activer l'option
setiosflags(ios::option)   // activer l'option d'entrée-sortie
unsetf(ios::option)        // désactiver l'option
```

Manipulateurs généraux (manipulateur – persistance – effet)

flush	non	Ecrit le buffer du flot (vide le buffer)
endl	non	Envoie une fin de ligne ('\n') ainsi qu'un flush
setw(n)	non	Spécifie que la prochaine sortie s'effectuera sur n caractères
setfill(c)	oui	Indique le caractère de remplissage (c) pour setw
left	non	Aligne la sortie à gauche lors de l'utilisation de setw
right	non	Aligne la sortie à droite lors de l'utilisation de setw (par défaut)

Manipulateurs et options de configuration (2/2)

Sécurisation d'entrée des chaînes de caractères
Pour une chaîne `nom` de longueur `LG_MAX`
`is.width(LG_MAX); is >> nom;`

(manipulateur – persistance – effet)

Formatage des nombres entiers (manipulateur – persistance – effet)

dec	oui	Insère/extrait les nombres sous forme décimale
oct	oui	Insère/extrait les nombres sous forme octale
hex	oui	Insère/extrait les nombres sous forme hexadécimale
uppercase/ nouppercase	oui	Affiche les lettres en majuscule/annule l'effet majuscule

Formatage des nombres flottants (manipulateur – persistance – effet)

setprecision(n)	oui	Spécifie le nombre de chiffres après la virgule affichés pour les nombres flottants non entiers (6 par défaut)
fixed	oui	Affiche les nombres flottants en notation décimale
scientific	oui	Affiche les nombres flottants en notation scientifique
showpoint/noshowpoint	oui	Affiche le point/annule l'effet
showpos/noshowpos	oui	Affiche le signe/annule l'effet

Formatage des booléens (manipulateur – persistance – effet)

boolalpha/noboolalpha	Affiche les booléens sous forme alphabétique ("true" et "false" au lieu de "0" et "1")/annule l'effet

Manipulateurs et configuration

C++

```
/**
 * @file EcritureFTexte.cpp – coursC/coursCpp1
 * @brief Test de manipulateurs
 */
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
   double val[] = {1234.5678, -234.7, 3.};

   // sauvegarde de la configuration courante (par défaut)
   ios::fmtflags old = cout.flags();
   cout << setprecision(3);
   cout << showpos;
   cout << right << setfill('.');
   cout << fixed;
   cout << "Format défini par manipulateurs :" << endl;
   for (int i=0; i<3; i++) {
        cout << setw(10) << val[i] << endl;
   }
   cout << noshowpos;
   // restauration de la configuration initiale
   cout << "Format par défaut :" << setiosflags(old)<< endl;
   for (int i=0; i<3; i++) {
        cout << val[i] << endl;
   }
}
```

```
Format défini par manipulateurs :
.+1234.568
..-234.700
....+3.000
Format par défaut :
1234.568
-234.700
3.000
```


Ecriture dans un fichier texte

C++

```
/** @file EcritureFTexte.cpp – coursCpp/CoursCpp1
 *  @brief Ecriture d'un fichier texte avec numérotation des  lignes
 */
#include <iostream>
#include <fstream>
using namespace std;
main() {
    char nomFichier[255];
    cout << "Nom du fichier à écrire : ";
    cin.getline(nomFichier, 255);  // extraction du rc
    ofstream flotOut;
    flotOut.open(nomFichier, ios::out);
    if (flotOut.fail()) {
        cerr << "Erreur : impossible d'écrire dans le fichier "
             << nomFichier << endl;
    }
    else {
        int noL=0;
        char phrase[1000];
        cout << "Entrez votre texte (pour terminer,\n";
        cout << "'.' en début de ligne) :\n";
        do {
            //cout << "Entrez une phrase :" << endl;
            cin.getline(phrase, 1000); // extraction du rc
            flotOut << "L" << ++noL <<" : " << phrase << endl;
        } while (phrase[0]!='.');
        flotOut.close();
    }
}
```

```
Nom du fichier à écrire : Mauriac.txt
Entrez votre texte (pour terminer,
'.' en début de ligne) :
Rien ne dérange davantage
une vie que l'amour.
.
```

Fichier texte créé Mauriac.txt

```
L 1 : Rien ne dérange davantage
L 2 : une vie que l'amour.
L 3 : .
```

Lecture d'un fichier texte par programme

C++

```
/** @file LectureFTexte.cpp - coursCpp/CoursCpp1
 *  @brief Lecture d'un fichier texte
 */
#include <iostream>
#include <fstream>
using namespace std;

main() {

    char nomFichier[255];
    cout << "Nom du fichier texte à lire\n";
    cout << "(donnez le chemin du projet au fichier) :\n" ;
    cin.getline(nomFichier, 255);  // extraction du rc

    ifstream fEntree;
    fEntree.open(nomFichier);

    if (fEntree.fail()) {
       cerr << "Impossible de lire le fichier " << nomFichier << endl;
    }
    else {
       char phrase[1000];
       while (!fEntree.eof()) {
          fEntree.getline(phrase, 1000);
          cout << phrase << endl;
       }
       fEntree.close();
    }
}
```

```
Nom du fichier texte à lire
(donnez le chemin du projet au fichier) :
Mauriac.txt
L1 : Rien ne dérange davantage
L2 : une vie que l'amour.*
L3 : .
```


Inclusion d'entête
`#include <stdio.h>`

Correspondance des méthodes

C

cf. Bibliothèque standard C (**stdio**) – http://www.cppreference.com/stdio/

1) **Création d'un flot**
   – d'entrée ou de sortie **FILE*** *streamName* ;
2) **Ouverture d'un flot** (pour tout mode d'ouverture)
   FILE* fopen(const char* *fileName*, **const char*** *type*);
3) **Réalisation des entrées-sorties**

– entrée	fgetc	lecture d'un caractère
	fgets	lecture d'une chaîne de caractères
	fread	lecture de bloc
	fscanf	lecture formatée
– sortie	fputc	écriture d'un caractère
	<<	écriture d'une chaîne de caractères
	fwrite	écriture de bloc
	fprintf	écriture formatée
	fprintf	écriture du tampon
– accès direct	fseek	déplacement dans le fichier
	ftell	position dans le fichier

4) **Fermeture du flot** (pour tout mode d'ouverture)
   int fclose(const FILE* *streamName*);

Appel des fonctions `functionName(arg1, …);`

Ecriture d'un fichier binaire de dates

C

```
/**
 * @file EcritureFBinaireDates.c - coursC/coursC2
 * @author l'équipe pédagogique
 * @brief Ecriture d'un fichier binaire de dates
 */
int main() {

   FILE *flotOut;// fOut : le fichier de sortie
   flotOut = fopen("FileDates.bin", "wb");

   if (flotOut == NULL)
   fprintf(stderr, "Impossible d'écrire dans le fichier\n");
   else {
      Date d;
      printf("Saisie de dates à enregistrer dans le fichier\n");
      printf("(jusqu'à une année nulle non enregistrée)\n");
      fflush(stdout);
      do {
         d=saisir();
         if (d.annee!= 0)
         fwrite((const void*) &d, sizeof(d), 1, flotOut);
      } while (d.annee != 0);

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

```
#include <stdio.h>
#include "Date.h"
```

```
Saisie de dates à enregistrer dans le fichier
(jusqu'à une année nulle non enregistrée)
Date (jour? mois? annee?) ? 14 7 1789
Date (jour? mois? annee?) ? 4 7 1776
Date (jour? mois? annee?) ? 1 5 2006
Date (jour? mois? annee?) ? 1 8 0
```


Lecture d'un fichier binaire de dates

C

```
/**
 * @file LectureFBinaireDates.c - coursC/coursC3
 * @author l'équipe pédagogique
 * @brief Lecture d'un fichier binaire de dates
 */
int main() {
   printf("Lecture du fichier binaire\n"); fflush(stdout);
   FILE *flotIn;
   flotIn = fopen("FileDates.bin", "rb");
   if (flotIn==NULL) fprintf(stderr, "Impossible de lire le fichier\n");
   else {
      Date d;
      /* calcul du nombre d'éléments de type Date dans le fichier */
      fseek(flotIn, 0, SEEK_END);
      int nbDates = ftell(flotIn) / sizeof(Date);
      fseek(flotIn, 0, SEEK_SET);
      printf("Le fichier contient %d dates\n", nbDates); fflush(stdout);
      /** lecture et affichage des dates du fichiers */
      int i;
      for (i = 0; i < nbDates; ++i) {
         fread((void*)&d, sizeof(Date), 1, flotIn); afficher(d);
      }
      printf("Fin de lecture\n");
      fflush(stdout);
      fclose(flotIn);
      return 0;
   }
}
```

```
#include <stdio.h>
#include "Date.h"
```

```
Lecture du fichier binaire
Le fichier contient 3 dates
14/7/1789
4/7/1776
1/5/2006
Fin de lecture
```

Modification d'un fichier binaire de dates

C

```c
/** @file ModificationFBinaireDates.c-coursC/coursC4
 * @brief Modification d'un fichier binaire de dates
 */
#include <stdio.h>
#include "Date.h"
int main() {
    printf("Modification d'un fichier binaire (déjà créé) de dates\n");
    fflush(stdout);
    FILE *flotInOut;
    flotInOut = fopen("FileDates.bin", "rb+");
    if (flotInOut == NULL) fprintf(stderr, "Impossible de lire le
       fichier\n");
    else {
        Date d;
        /* calcul du nombre d'éléments de type Date dans le fichier */
        fseek(flotInOut, 0, SEEK_END);
        int nbDates = ftell(flotInOut) / sizeof(Date);
        if (nbDates>0) {
            /* modification de l'élément médian (année incrémentée) */
            fseek(flotInOut, (nbDates/2)*sizeof(Date), SEEK_SET);
            fread((void*)&d, sizeof(Date), 1, flotInOut);
            d.annee++;
            fseek(flotInOut, -sizeof(Date), SEEK_CUR);
            fwrite((const void*) &d, sizeof(d), 1, flotInOut);
            fflush(flotInOut);
        }
        /** lecture et affichage des dates du fichiers */
        fseek(flotInOut, 0, SEEK_SET);
        int i;
        for (i = 0; i < nbDates; ++i) {
            fread((void*)&d, sizeof(Date), 1, flotInOut);
            afficher(d);
        }
        printf("Fin de lecture\n");
        fflush(stdout);
        fclose(flotInOut);
        return 0;
    }
}
```

```
Modification d'un fichier
binaire (déjà créé) de dates
14/7/1789
4/7/1777
1/5/2006
Fin de lecture
```


Ecriture d'un fichier texte

C

```c
/** @file EcritureFTexte.c - coursC/coursC1
 *  @brief Ecriture d'un fichier texte avec numérotation des  lignes
 */
#include <stdio.h>
main() {
   char nomFichier[255];
   printf("Nom du fichier à écrire : "); fflush(stdout);
   scanf("%s", nomFichier);
   getchar();// extraction du RC
   FILE *flotOut;
 flotOut=fopen(nomFichier, "w");
   if (flotOut == NULL) {
      fprintf(stderr,"Erreur :
      impossible d'écrire dans le fichier %s\n", nomFichier);
   }
   else {
      int noL=0;
      char phrase[1000];
      printf("Entrez votre texte (pour terminer,\n");
      printf("'.' en début de ligne) :\n"); fflush(stdout);
      do {
         fgets(phrase, 1000, stdin); // extraction du rc
         fprintf(flotOut,"L %d : %s",++noL, phrase); fflush(flotOut);
      } while (phrase[0]!='.');

      fclose(flotOut);
   }
}
```

```
Nom du fichier à écrire : Mauriac.txt
Entrez votre texte (pour terminer,
'.' en début de ligne) :
Rien ne dérange davantage
une vie que l'amour.
.
```

Fichier texte créé `Mauriac.txt`

```
L 1 : Rien ne dérange davantage
L 2 : une vie que l'amour.
L 3 : .
```


Lecture d'un fichier texte

C

```c
/**
 * @file LectureFTexte.c - coursC/coursC1
 * @author l'équipe pédagogique
 * @brief Lecture d'un fichier texte
*/

#include <stdio.h>
main() {
   char nomFichier[255];
   printf("Nom du fichier texte à lire\n");
   printf("(donnez le chemin du projet au fichier) : ");
   fflush(stdout);
   //fgets(nomFichier, 255, stdin);  // extraction du rc
   scanf("%s", nomFichier);
   getchar();// extraction du rc

   FILE *fEntree;
   fEntree = fopen(nomFichier,"r");

   if (fEntree == NULL) {
      fprintf(stderr,"Erreur : impossible de lire dans le fichier
      %s\n", nomFichier);}
   else {
      char phrase[1000];
      while (fgets(phrase, 1000, fEntree) != NULL)
         printf("%s", phrase); fflush(stdout);
      fclose(fEntree);
   }
}
```

```
Nom du fichier texte à lire
(donnez le chemin du projet au fichier) :
Mauriac.txt
L 1 : Rien ne dérange davantage
L 2 : une vie que l'amour.
L 3 : .
```


Ce que vous avez appris aujourd'hui…

Les principes des entrées-sorties de haut niveau

- à rendre **persistantes** les données de programmes au moyen des **fichiers**
- à **sauvegarder** et à **relire** les données sur fichier binaire et sur fichier texte

- à **formater** les **données** en sortie
- à rendre **robuste** les **données** acquises par l'entrée standard et plus généralement par toute entrée de haut niveau