Cours n° 2

Les entrées-sorties

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

Sommaire

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 les **sorties** (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 et à la lecture de données d'un type donné sur un support non volatile (comme par exemple un fichier)

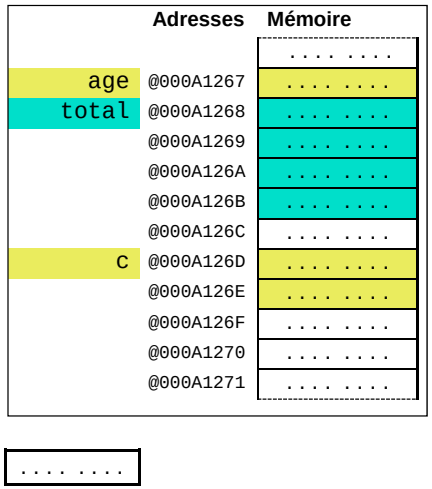
Notion de pointeur et adresse-mémoire d'une variable

Variable contenant une adresse mémoire
Début d'une zone mémoire
Adresse mémoire d'une variable
Structures de données avancées (arborescence)

Opérateurs
& Adresse mémoire d'une variable
* Valeur de la variable pointée
sizeof Taille mémoire d'une variable
calloc Réservation d'une zone mémoire)

Exemples

```
byte age; float total; char c;
byte* page  = &age;
float* ptotal  = &total;
char* pc  = &c;
float *vect = calloc(10, sizeof(float));
Total = vect[2];
```



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

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

Etapes de mise en œuvre des E/S sur fichier

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

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

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

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

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

Mode d'ouverture d'un flot

```
flot d'entrée   void open(const char* fileName, openmode mode=in);
flot de sortie  void open(const char* filename, openmode mode=out|trunc);
flot d'E/S      void open(const char* filename, openmode mode=in|out);
```

openmode – vecteur d'état

`ios_base::in`	Fichier en lecture (flot d'entrée)
`ios_base::out`	Fichier en écriture (flot de sortie)
`ios_base::binary`	Fichier binaire (versus fichier texte)
`ios_base::app`	(append) Ouverture en ajout en fin de fichier (pour toute écriture)
`ios_base::ate`	(at end) Déplacement en fin de fichier (après ouverture)
`ios_base::trunc`	(truncate) Ecrase le fichier à l'ouverture (sans app et ate)

Remarque : Le paramètre mode a une valeur par défaut (`mode=<valParDéfaut>`) 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_base::in);  fichier texte en lecture
flotS.open("XFile") //≡ flotS.open("XFile", ios_base::out);  fichier texte en écriture
flotE.open("XFile", ios_base::binary);                       fichier binaire en lecture
flotS.open("XFile", ios-base::out|ios_base::binary|ios_base::append);  …
```


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

Extrait des fonctions applicables aux flots d'entrée - ifstream
cf. Bibliothèque de iostream http://www.cplusplus.com/ref/iostream/

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

Lecture de chaîne de caractères

```
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)

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)

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 de 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-base::beg) déplace la position courante n octets après le début

```
streampos  tellg ( );
```

Retourne la position courante dans le flot d'entrée

Extrait des méthodes d'écriture – *ofstream*

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

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

Ecriture de chaîne de caractères
`<<`

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

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 caractères

```
/** @file EcritureFBinaire.cpp
 *  @brief Ecriture d'un fichier binaire de caracteres
 */

int main() {

   ofstream flotOut;
   flotOut.open("FileChar.bin", ofstream::out|ofstream::binary);

   if (flotOut.fail()) cerr << "Impossible d'écrire dans le fichier\n";
   else {
      char d;
      cout << "Saisie de caractères a enregistrer dans le fichier\n"
      cout << "nombre de caracteres ?" << endl;
      cin >> nCar;
      for (int i = 0;i < nCar;i++) {
         cout << "?" << endl; cin >> d;
         flotOut.write((char*) &d, sizeof(d));
      }

      while (d.annee != 0);

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

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


Lecture d'un fichier binaire de caractères

```
/** @file LectureFBinaire.cpp
 *  @brief Lecture d'un fichier binaire
 */
int main() {
   cout << "Lecture du fichier binaire" <<endl;

   ifstream flotIn;
   flotIn.open("FileChar.bin", ifstream::in|ifstream::binary);

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

      /* calcul du nombre d'éléments dans le fichier */
      flotIn.seekg(0, ifstream::end);
      int nbChar = flotIn.tellg() / sizeof(char);
      flotIn.seekg(0);
      cout << "Le fichier contient " << nbChar << " caracteres\n"

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

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


Modification d'un fichier binaire de caractères

```
/** @file ModificationFBinaireDates.cpp
 *  @brief Modification d'un fichier binaire de dat
 */
int main() {
   cout << "Modification d'un fichier binaire\n(dé
   fstream flotInOut;
   flotInOut.open("FileChar.bin", ios::in|ios::out|ios::binary);
   if (flotInOut.fail()) { cerr << "Impossible de lire le fichier\n";
                           exit(1); }
   char d;
   /* calcul du nombre d'éléments dans le fichier */
   flotInOut.seekg(0, fstream::end);
   int nbChar = flotInOut.tellg() / sizeof(char);
   if (nbChar>0) {
      /* modification de l'élément médian (incréméntation) */
      flotInOut.seekg ((nbChar/2)*sizeof(char), ios::beg);
      flotInOut.read((char*) &d, sizeof(char));
      d +=1;
      flotInOut.seekg ((nbChar/2)*sizeof(char), ios::beg);
      flotInOut.write((char*) &d, sizeofchar));
      flotInOut.flush();
   }
   /** lecture et affichage des caracteres du fichier */
   flotInOut.seekg(0, ios::beg);
   for (int i = 0; i < nbChar; ++i) {
      flotInOut.read((char*)&d, sizeof(char)); cout << d;
    }
   flotInOut.close(); return 0;
}
```

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


Lecture robuste (1/2)

Problème

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

Exemple :

```
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**

Lecture robuste – principe de lecture d'un flot

Exemple : `cin << i << j << k;`

Caractères séparateurs de données dans un flot
l'espace, la tabulation, la fin de ligne, le retour chariot

à la première requête : cin << i

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

Lecture robuste (2/2)

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* msgSaisie) {
   unsigned int n;
   cout << msgSaisie << flush;
   cin >> n;
   if (cin.fail()) { // teste l'état de cin
      cout << "l'entrée n'est pas valide..." << endl;
      cin.clear(); // remets cin dans un état lisible
      cin.ignore(numeric_limits<streamsize>::max(), '\n'); // ignore
toute la ligne de données
   }
   return n;
}
int main() {
   unsigned int nb;
   char msg[]="Saisir un nombre entre 1 et 10 : ";
   do {
      nb=saisirNombreNaturel(msg);
   } while ((nb < 1) || (nb > 10));
   cout <<"Nombre lu : " << nb << endl;
   return 0;
}
```

```
Entrez un nombre entre 1 et 10 : 0
Entrez un nombre entre 1 et 10 : A
l'entrée n'est pas valide…
Entrez un nombre entre 1 et 10 : -3
l'entrée n'est pas valide…
```

```
Entrez un nombre entre 1 et 10 : 7
Nombre lu : 7
```


Manipulateurs et options de configuration

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 << ...
```

Utilisation des options de format

```
setf(ios::option)        // activer l'option
unsetf(ios::option)      // désactiver l'option
```

Manipulateurs généraux (manipulateur – persistance – effet)

flush	non	écrit 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 (comportement par défaut)

Manipulateurs et options de configuration

Formatage des nombres entiers (manipulateur – persistance – effet)

dec	oui	Injecte / extrait les nombres sous forme décimale
oct	oui	Injecte / extrait les nombres sous forme octale
hex	oui	Injecte / extrait les nombres sous forme héxadécimale
uppercase/ nouppercase	oui	Affiche les lettres de la représentation héxadécimale en majuscule / annule l'effet d'uppercase

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 de boolalpha

Manipulateurs et configuration

```
/**
 * @file EcritureFTexte.cpp
 * @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
   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

```
/** @file EcritureFTexte.cpp
 *  @brief Ecriture d'un fichier texte avec numérotation des  lignes
 */
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();
   }
}
```

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

```
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

```
/** @file LectureFTexte.cpp
 *  @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 : .
```