Entrées et sorties en C++, une histoire de flux

christophe.labourdette(at)cmla.ens-cachan.fr

Septembre 2016

Modèle

- Les systèmes d'exploitation d'aujourd'hui manipulent les entrées-sorties au travers de *driver* qui pilotent les périphériques et de librairies qui fournissent des API.
- Sur ce modèle les entrées-sortie peuvent être vues comme des **flux**, les **streams**.
- On distingue bien entendu les flux standards, d'entrées, **istream** et de sorties, **ostream**.
- On peut également manipuler des flux de *fichiers*, **ifstream** et **ofstream**, des flux de *string*, **isstream** et **osstream**.
- Les **stream** sont fournis par la librairie standart du **C++**.

iostream

- La librairie standard fournit les types
  - **istream** pour les entrées, l'entrée standard est **cin**
  - **ostream** pour les sorties, la sortie standard est **cout**
- Pour des raisons de performance les flux passent par des tampons (buffer).
- Les entrées-sorties ne se font donc pas en temps réel mais avec un délai,
- Il est nécessaire d'inclure le fichier d'entête, **#include** <iostream>
- Il ne faut pas oublier de préciser le *namespace* de la librairie, par exemple pour le flux de sortie standard
  std::cout << ''hello world'' << std::endl;

exemple d'ouverture

```
#include <iostream>
#include <string>
#include <fstream>
#include <cstdlib>
// .....
std::cout << "Entrez le nom du fichier: ";
std::string nom;
std::cin >> nom;
std::ifstream fichier(nom.c_str());
if (!fichier)
 {
  std::cout << "Erreur avec " << nom << "\n";
  exit(1);
 }
// .....
```

commentaires

- Les entêtes
  1. iostream définit les canaux standards
  2. string permet la manipulation des chaînes de caractères
  3. fstream définit les flux de fichiers et donc *ifstream*
  4. cstdlib permet l'utilisation de *exit()*
- On récupère le nom du fichier en manipulant les flux d'entrée et de sortie standard
- la définition et l'ouverturedu fichier se font avec :
  `std :: ifstream fichier (nom.c_str());`
- La dernière partie traite simplement le cas où le fichier ne s'ouvre pas, avec l'affichage d'un message et la sortie du programme.

ouverture et fermeture

- Dans le cas d'une ouverture d'un *ifstream*, d'un *ofstream* **fstream** comme dans l'exemple précédent, à l'aide d'une déclaration de variable, le fichier est fermé lorsque la variable associé quitte son champ d'application
- il est possible d'ouvrir ou de fermer explicitement un fichier avec les commandes open() et close()

Ouverture du fichier truc en lecture et fermeture

```
std::fstream fichier;
fichier.open("truc",ios_base::in);
// .....
fichier.close();
```

Lecture et écriture

Un fichier notes avec des lignes comme : 12.8 8.0 15.5 7.8

```
{
 float a, b, c, d;
 std::fstream fichier1, fichier2;
 fichier1.open("notesIn",ios_base::in);
 fichier2.open("notesOut",ios_base::out);
 while(fichier1 >> a >> b >> c >> d)
  {
    fichier2 <<d<<"\t"<<c<<"\t"<<b<<"\t"<<a<<"\n";
  }
}
```

Cet exemple reécrit, dans l'ordre inverse, les notes lues dans notesIn dans le fichier notesOut.

les entiers

il est possible d'utiliser les mots clés suivant pour des entiers

- **oct** utilisation d'une base octale
- **dec** utilisation d'une base décimale
- **hex** utilisation d'une base hexadécimale
- **showbase** on préfixe par 0 pour l'octal et 0x pour l'hexadécimal
- **noshowbase** on ne préfixe pas

exemple

le programme :

```
std::cout << 98765 << '\t' << std::hex << 98765 <<
std::cout << std::oct << 98765 << "\n";
std::cout << std::showbase;
std::cout << 98765 << '\t' << std::hex << 98765 <<
std::cout << std::oct << 98765 << "\n";
```

produit la sortie suivante :

```
98765   181cd   300715
0300715 0x181cd 0300715
```

les flottants

Pour les flottants, il existe deux modes d'affichage :

- **fixed** un affichage en virgule fixed
- **scientific** un affichage avec une mantisse et un exposant

Dans chaque mode on peut choisir, la précision et les champs utilisés.

Paramêtres

La précision est modifiée par la fonction **setprecision()**, alors que les champs sont fixés par la commande **setw()**. La précision n'a pas toujours la même définition selon le mode d'affichage :

- **fixed** : la précision représente le nombre de chiffre après la virgule
- **scientific** : la précision représente le nombre de chiffre après la virgule

Le champ est le nombre de caractères utilisé pour afficher un entier ou une chaîne de caractères. Il peut également être utilisé pour des flottants.

exemple

Le code :

```
double x1= 165543.09789655;
int i1=676878;
cout<<fixed<< x1 <<'\t'<<scientific<<x1<<'\n';
cout<<fixed<< setprecision(4)<<setw(15)<<x1<<'\t';
cout<<scientific<< setprecision(4)<<
cout<<setw(15)<<x1<<'\n';
```

produit la sortie suivante :

```
165543.097897 1.655431e+05
   165543.0979      1.6554e+05
```

Mode d'ouverture

Il est possible d'ouvrir un fichier dans de nombreux modes :

- **ios_base::app** append, pour ajouter à la fin du fichier
- **ios_base::ate** at the end, ouvre et se place à la fin
- **ios_base::binary** en mode binaire
- **ios_base::in** en mode lecture
- **ios_base::out** en mode écriture
- **ios_base::trunc** tronque le fichier à une 0

Le mode est spécifié de façon optionelle après le nom du fichier :

```
ofstream  fichier (nom, ios_base::app);
```

string

- Il est possible d'utiliser un **string** comme source d'un **istream** ou comme resultat d'un **ostream** ce sont des **istringstream** et des **ostringstram**.
- Ils sont utilisés pour extraire des valeurs numériques de chaînes de caractères ou inversement pour écrire des valeurs numériques comme chaînes de caractères.
- La fonction `getline(cin, ligne);` permet de lire une ligne complète et de la placer dans une **string**.
- La fonction `istream::get()` lit un simple caractère dans son argument
- Les fonctions toupper(c) et tolower(c) permettent de passer un caractère en majuscule ou en minuscule.