6. Entrées, sorties, fichiers, en C++

Les entrées sont les données que l'utilisateur fournit à un programme durant l'exécution, par l'intermédiaire du clavier ou d'un fichier.
Les sorties sont les résultats que le programme fournit à l'utilisateur durant l'exécution, par l'intermédiaire de l'écran ou d'un fichier.
On utilise les instructions d'entrée-sortie du C++ parce qu'elles sont plus simples que celles du C. Un chapitre, hors programme, est consacré à celles du C.
On n'indique ici que le strict minimum.
Il faut, dans tous les cas, ajouter au début du fichier contenant le programme :

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

1 Afficher des résultats à l'écran

Pour afficher à l'écran la valeur d'une variable `x` :

```
cout << x << endl;
```

Le `endl` signifie que le programme devra aller à la ligne après la valeur de `x`.
Pour ajouter des commentaires, par exemple : *Valeur de x :*

```
cout << "Valeur de x : " << x << endl;
```

Pour afficher les valeurs de plusieurs variables `x`, `y` et `z` :

```
cout << x << " " << y << " " << z << endl;
```

Il faut au moins un commentaire blanc pour séparer les valeurs des variables.

2 Entrer des données à partir du clavier

Pour attribuer une valeur à une variable `x` à partir du clavier :

```
cin >> x;
```

Lors de l'exécution, la valeur entrée au clavier doit être validée par un *Entrée*.
Il faut faire précéder cette instruction d'une demande telle que :

```
cout << "Valeur de x ? ";
```

sinon le programme reste muet dans l'attente de la valeur et l'utilisateur ne comprend pas nécessairement que c'est à lui d'agir.
Pour lire plusieurs valeurs à la fois :

```
cout << "Valeur de x y z ? ";
cin >> x >> y >> z;
```

Les valeurs entrées au clavier doivent être séparées par des blancs ou des *Entrée* et la dernière (celle de `z` ici), suivie d'un *Entrée* de validation.
Tant qu'un programme n'est pas au point, il ne faut pas utiliser de `cin`, mais mettre les valeurs des données dans le programme sous la forme `x = 3.12`, etc. Sinon, à chaque exécution destinée à tester le programme, il faut taper les données au clavier, ce qui est une perte de temps. Ensuite, si le programme est destiné à être utilisé, on peut mettre des `cin`, mais à condition qu'ils soient en nombre très limité. Sinon, l'entrée des données au clavier est trop fastidieuse et comporte trop de risques d'erreur. Il vaut mieux placer d'abord les données dans un fichier qui leur est dédié et les faire lire par le programme dans ce fichier (voir ci-dessous l'emploi des fichiers).
Inversement, il n'est pas normal que l'utilisateur d'un programme considéré comme achevé soit contraint d'entrer ses données dans le fichier du programme lui-même.

3 Écrire dans un fichier

Pour écrire dans un fichier, il faut ajouter la directive :

```
#include<fstream>
```

et ouvrir le fichier en écriture par :

```
fstream fich;
fich.open("programme.res", ios::out);
```

Schématiquement on peut dire que le fichier dans lequel on veut écrire a deux noms, tous les deux choisis par l'utilisateur :

1. l'un pour *Linux* , dans cet exemple : *programme.res*
2. l'autre pour le programme C, dans cet exemple `fich`

Ensuite, on procède exactement de la même façon qu'avec l'affichage à l'écran, en remplaçant simplement partout `cout` par `fich`. Par exemple, pour écrire la valeur de la variable `x` dans le fichier connu de *Linux* sous le nom *programme.res* :

```
fich << x << endl;
```

Il faut fermer le fichier après la dernière instruction qui l'utilise par :

```
fich.close();
```

Les fichiers encore ouverts à la fin de l'exécution du programme sont fermés automatiquement.

Remarque

Tout comme la déclaration et l'initialisation d'une variable d'un autre type (comme par exemple un `int`), il est aussi possible de combiner la déclaration et l'initialisation d'une variable du type `fstream` dans une seule instruction. Ainsi on peut écrire `fstream fich("programme.res",ios::out);` au lieu de `fstream fich; fich.open("programme.res",ios::out);`

4 Lire dans un fichier

Pour lire dans un fichier, il faut, comme pour y écrire, ajouter la directive :

```
#include<fstream>
```

et ouvrir le fichier en lecture par :

```
fstream fich;
fich.open("programme.dat", ios::in);
```

Ensuite, on procède exactement de le même façon qu'avec l'entrée des données au clavier, en remplaçant simplement partout `cin` par `fich`. Par exemple, pour attribuer une valeur écrite dans le fichier connu de *Linux* sous le nom *programme.res*, à la variable `x` :

```
fich >> x;
```

4.1 Lire dans un fichier contenant un nombre de lignes connu à l'avance

On suppose qu'on a un fichier de nom *Linux coordonnees.dat*, contenant un triplet de valeurs par ligne, comme dans l'exemple suivant :

1.76 3.45 7.81
9.87 4.65 7.33
6.78 0.67 6.23
...

et qu'on sait que ce fichier contient `n` lignes. Ces valeurs peuvent être lues par les instructions :

```
#include<iostream>
#include<fstream>
using namespace std;
int main() {
  double x, y, z;
  int i, n = ...;
  fstream fich;
  fich.open("coordonnees.dat", ios::in);
  for(i = 1; i <= n; i++) {
    fich >> x >> y >> z;
    cout << x << " " << y << " " << z << endl; // vérification de la lecture
    ...
```

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

4.2 Lire dans un fichier contenant un nombre de lignes non connu à l'avance

La seule différence avec le paragraphe précédent est qu'on ne connaît pas à priori le nombre de lignes du fichier. La condition d'arrêt de la boucle de lecture est donnée par la valeur de l'instruction `fich >> x >> y >> z`.

```
#include<iostream>
#include<fstream>
using namespace std;
int main() {
  double x, y, z;
  int i;
  fstream fich;
  fich.open("coordonnees.dat", ios::in);
  i = 0;
  while(fich >> x >> y >> z) {
    cout << x << " " << y << " " << z << endl; // vérification de la lecture
    i++;                                       // comptage du nombre de lignes
    ...
  }
  cout << "Le fichier coordonnees.dat a " << i << " lignes" << endl;
  fich.close();
  return 0;
}
```

5 Fermer puis ré-ouvrir un fichier dans un même programme pour y lire ou écrire

5.1 Fermer puis ré-ouvrir et lire

Le programme suivant écrit dans le fichier *toto.txt*, le ferme, puis le ré-ouvre et y lit à partir du début :

```
#include<iostream>
#include<fstream>
using namespace std;
int main() {
  int i = 1;
  fstream fich;
  fich.open("toto.txt", ios::out);
  fich << i+1 << endl;
  fich.close();
  fich.open("toto.txt", ios::in);
  fich >> i;
  fich.close();
  cout << "i=" << i << endl;
  return 0;
}
```

5.2 Fermer puis ré-ouvrir et écrire

Le programme suivant écrit dans le fichier *toto.txt*, le ferme, puis le ré-ouvre et y écrit à la suite :

```
#include<iostream>
#include<fstream>
using namespace std;
int main() {
```

```
  int i = 1;
  fstream fich;
  fich.open("toto.txt", ios::out);
  fich << i << endl;
  fich.close();
  fich.open("toto.txt", ios::out|ios::app);
  fich << i+1 << endl;
  fich.close();
  return 0;
}
```

Si, au lieu d'écrire à la suite, on veut écrire au début, donc effacer ce qui était précédemment écrit, on remplace la ligne :

```
fich.open("toto.txt", ios::out|ios::app);
```

par :

```
fich.open("toto.txt", ios::out|ios::trunc);
```

ou en fait par juste `ios::out` parce que `ios::trunc` est le mode par défaut.

6 Passer un nom de fichier en argument d'une fonction

Exemple :

```
...
int f(..., fstream &fich, ...) {
  ...
  fich << ... << endl;
  ...
  return 0;
}
int main() {
  ...
  fstream res("mon_fichier.res", ios::out);
  f(...,res,...);
  ...
  return 0;
}
```

C'est un passage par référence, propre au C++.

7 Présentation de l'affichage à l'écran et de l'écriture dans un fichier

Pour ce qui suit il faut ajouter la directive :

```
#include<iomanip>
```

7.1 Notation standard, fixe ou scientifique

```
cout << x << endl; // affiche en notation standard
cout << fixed; // passe en notation fixe pour les cout suivants
cout << scientific; // passe en notation scientifique pour les cout suivants
cout.setf(ios_base::floatfield); // revient en notation standard
```

7.2 Précision (nombre de chiffres affichés)

```
n = cout.precision(); // stocke la précision actuelle dans la variable entière n
cout << setprecision(15); // fixe la précision à 15 pour les cout suivants
cout << setprecision(n); // revient à la précision initiale
```

7.3 Largeur minimum consacrée à chaque affichage

```
cout << setw(10) << x << endl; // impose une largeur minimum, n'agit que sur
                               // l'élément suivant (largeur nulle par défaut)
```

Si on écrit dans un fichier au lieu d'afficher à l'écran, toutes ces instructions restent valides, il suffit de remplacer partout `cout` par `fich`.

7.4 Exemple récapitulatif

Le programme suivant :

```
#include<iostream>
#include<iomanip>
#include<fstream>
using namespace std;
int main() {
  double x = 12345.6789012345;
  int n;
// Notation standard, fixe ou scientifique ------------------------------------------------------
  cout << "x=" << x << endl; // affiche en notation standard
  cout << fixed; // passe en notation fixe pour les cout suivants
  cout << "x=" << x << endl;
  cout << scientific; // passe en notation scientifique pour les cout suivants
  cout << "x=" << x << endl;
  cout.setf(ios_base::floatfield); // revient en notation standard
  cout << "x=" << x << endl << endl;
// Précision (nombre de chiffres affichés) ------------------------------------------------------
  n = cout.precision(); // stocke la précision actuelle dans la variable entière n
  cout << "La précision actuelle est : " << n << endl;
  cout << "x=" << x << endl;
  cout << setprecision(15); // fixe la précision à 15 pour les cout suivants
  cout << "x=" << x << endl;
  cout << setprecision(n); // revient à la précision initiale
  cout << "x=" << x << endl;
// Largeur minimum consacrée à chaque affichage -------------------------------------------------
  cout << "x=" << setw(10) << x << endl; // impose une largeur minimum, n'agit que sur
                                         // l'élément suivant (largeur nulle par défaut)
  return 0;
}
```

donne le résultat :

x=12345.7
x=12345.678901
x=1.234568e+04
x=12345.7

La précision actuelle est : 6
x=12345.7
x=12345.6789012345

x=12345.7
x= 12345.7

8 Créer une séquence de fichiers

8.1 Méthode C

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

```
#include<stdlib.h>
#include<iostream>
#include<fstream>
using namespace std;
int main() {
  char* s = (char*)malloc(50*sizeof(char)); // voir chapitre Tableaux dynamiques
  int i, n = 13;
  fstream fich;
  for(i = 0; i < n; i++) {
    // %03d pour avoir une séquence 000 001 002 etc.
    sprintf(s, "%s%03d%s", "fichier_", i, ".dat");
    cout << s << endl;
    fich.open(s, ios::out);
    fich << "Contenu du fichier" << " " << i << endl;
    fich.close();
  }
  return 0;
}
```

8.2 Méthode C++

```
// On ne peut pas écrire directement dans une string, il faut passer par un ostringstream.
#include<iostream>
#include<iomanip>
#include<fstream>
#include<stdlib.h>
#include<sstream>
#include<string>
using namespace std;
int main() {
  ostringstream y;
  int n = 13, i;
  fstream fich;
  for(i = 0; i < n; i++) {
    // setfill et setw pour avoir une séquence 000 001 002 etc.
    y << "fichier_" << setfill('0') << setw(3) << i << ".dat";
    fich.open((y.str()).c_str(), ios::out);
    fich << "Contenu du fichier " << i << endl;
    fich.close();
    y.str(""); // vide l'ostringstream
  }
  return 0;
}
```

9 Redirections d'entrée et de sortie

Il existe une autre possibilité pour écrire et lire dans un fichier : la « redirection » du résultat d'une commande *Linux*. Si on fait compiler et exécuter le programme contenu dans le fichier *jojo.cpp* par :

$ ccc jojo.cpp > lili

au lieu de simplement :

$ ccc jojo.cpp

tout ce qui aurait dû s'afficher à l'écran[1] est envoyé dans le fichier de nom *Linux lili*.

Remarque :

1. Donc toutes les sorties produites par `cout`.

Dans le cas où le fichier *lili* existe déjà :

$ ccc jojo.cpp >> lili

ajoute les sorties de `cout` au fichier *lili* sans effacer son contenu initial.

De même :

$ ccc jojo.cpp < dudu

tout ce qui aurait dû être entré au clavier [2] est envoyé par le fichier *dudu*.
Il faut bien remarquer qu'ici, c'est la commande *Linux* qui provoque l'écriture ou la lecture dans un fichier et non une instruction du C. C'est une méthode simple et efficace pour enregistrer des résultats ou fournir des données.

10 Remarque pour amateurs avertis

En plus de faire un retour à la ligne en ajoutant un caractère `\n`, `endl` purge le buffer de sortie et force ainsi son écriture en appelant `ostream::flush()` (cela a le même fonctionnement que la fonction `fflush()` du C). Les deux lignes de code suivantes sont donc équivalentes :

```
cout << "coucou" << endl;
cout << "coucou\n" << flush;
```

Il faut donc être prudent avec son utilisation, notamment avec les fichiers, car une opération de flush n'est pas gratuite. Son utilisation fréquente peut même sérieusement grever les performances en annulant tous les bénéfices d'une écriture bufférisée (*http ://cpp.developpez.com/faq/cpp/index.php ?page=SL#SL_ endl*).

2. Donc toutes les entrées produites par `cin`.