Les entrées sorties en C/C++

Dans ce chapitre, nous allons voir les fonctions d'entrée/sortie du langage C++, extension du langage C.

Écriture sur la sortie standard

Fonction printf et opérateur <<

Exemple 1

```
#include <iostream.h>   // indispensable pour utiliser cout

int main()
{
     cout << "bonjour";  // équivalent à printf("bonjour"); du langage C
     return 0;
}
```

- cout est un flot de sortie prédéfini associé à la sortie standard (stdout du C).
- << est un opérateur dont l'opérande de gauche (cout) est un flot et l'opérande de droite, une expression de type quelconque.

L'instruction "cout << …" précédente peut être interprétée comme ceci : le flot cout reçoit la valeur "bonjour".

Exemple 2

```
#include <iostream.h>   // indispensable pour utiliser cout

int main()
{
     int n = 25;

     cout << "valeur: ";
     cout << n;
     return 0;
}
```

Noter l'utilisation de // pour les commentaires de fin de ligne en C++

Le résultat serait :

valeur: 25

Exemple 3

Les instructions

```
cout << "valeur: ";
cout << n;
```

sont équivalentes à

```
cout << "valeur: " << n;
```

Exemple 4

```
#include <iostream.h>

int main()
{
     int n = 25;
     char c = 'a';
     char *ch = "bonjour";
     double x = 12.3456789;

     cout << "valeur de n : " << n << "\n";
     cout << "valeur de c : " << c << "\n";
     cout << "chaîne ch : " << ch << "\n";
     cout << "valeur de x : " << x << "\n";
     return 0;
}
```

L'exécution devrait donner :

```
valeur de n : 25
valeur de c : a
chaîne ch : bonjour
valeur de x : 12.3456789
```

Lecture sur l'entrée standard

Fonction scanf et opérateur >>

Lecture en C :

```
#include <stdio.h>
…
scanf ( Format, Liste des adresses des variables);
```

Lecture en C++ :

```
#include <iostream.h>
…
cin >> Var1 >> Var2 >> … >> VarN;
```

cin est le flot d'entrée connectée à l'entrée standard, le clavier. Il correspond au fichier prédéfini stdin du langage C.

Exemple
En C :

```
#include <stdio.h>

int main ()
{
     float valeur1, valeur2, valeur3;

     printf ("entrez 3 valeurs : ");
     scanf ("%f%f%f", &valeur1, &valeur2, &valeur3);
     return 0;
}
```

En C++ :

```
#include <iostream.h>

int main ()
{
     float valeur1, valeur2, valeur3;

     cout << "entrez 3 valeurs : ";
     cin >> valeur1 >> valeur2 >> valeur3;
     return 0;
}
```

Noter que contrairement à ce qui se passait pour scanf(), la lecture d'un caractère sur cin commence par sauter les séparateurs. Nous verrons plus tard comment lire directement ces caractères.

Exemples

```
#include <iostream.h>

int main ()
{
     int n;
     float x;
     char t[80+1];
```

```
      do
      {
            cout << "donner un entier, une chaîne et un flottant : ";
            cin >> n >> t >> x;
            cout << "merci pour " << n << ", " << t << " et " << x << "\n";
      }
      while (n);
      return 0;
}
```

Exécution:

```
donner un entier, une chaîne et un flottant : 15 bonjour 8.25
merci pour 15, bonjour et 8.25
donner un entier, une chaîne et un flottant : 15
                                    bonjour
8.25
merci pour 15, bonjour et 8.25
donner un entier, une chaîne et un flottant : 0 bye 0
merci pour 0, bye et 0
```

Noter l'usage des séparateurs.

```
#include <iostream.h>

int main ()
{
      char t[80+1];  // pour conserver les caractères lus sur cin
      int i = 0;     // position courante dans le tableau t

      cout << "entrez une suite de caractères terminée par un point.\n";
      do
            cin >> t[i];
      while ( t[i++] != '.');
      cout << "\n\nVoici les caractères effectivement lus :\n";

      i=0;
      do
            cout << t[i];
      while ( t[i++] != '.');
      return 0;
}
```

Exécution:

```
entrez une suite de caractères terminée par un point.
Voyons ce que
fait C++
                          à la lecture
d'une "suite de caractères".

Voici les caractères effectivement lus :
VoyonscequefaitC++àlalectured'une"suitedecaractères".
```

Fonction getline() et gets()

Ces fonctions permettent de lire des chaînes de caractères qui contiennent plusieurs mots.
En C :

```
#include <stdio.h>
…
     gets (Var_chaine);
```

En C++ :

```
#include <iostream.h>
…
cin.getline (Var_chaine, Nb_max, Car_special);
```

En C, la fonction gets() prend un argument de type chaîne de caractères. La lecture se poursuit jusqu'à la rencontre des caractères de fin de ligne ou de fin de fichier. Le caractère de fin de ligne est ignoré et le caractère '\0' est ajouté à la fin de la chaîne.

La fonction getline() prend trois arguments : Var_chaine est une variable de type chaîne de caractères. Nb_max est le nombre maximum de caractères à lire moins 1 (Nb_max – 1). Ce nombre correspond au nombre maximal de caractères que la chaîne peut contenir. Car_special, le troisième argument qui est facultatif, permet de spécifier un caractère dont la rencontre va interrompre la lecture. Lorsque cet argument est absent c'est le caractère de fin de ligne qui sera le caractère de fin de lecture.

Exemple :

```
char chaine[81];
cin.getline( chaine , 81);

char chaine[81];
cin.getline(chaine, 81, '%');
```

Les manipulateurs

Pour préciser les formats des données, des *manipulateurs* peuvent être insérés dans les instructions d'extraction et d'insertion dans un flot. Ces manipulateurs conservent leur état jusqu'au prochain changement, sauf pour setw, qui revient à 0 après chaque opération. Pour utiliser les manipulateurs, il faut inclure les fichiers <iostream.h> et <iomanip.h>.

Variables numériques
setw (i) : i est la largeur minimale de la prochaine donnée à afficher.

```
int var = 12;
cout << setw(7) << var;
```

cette instruction permet d'inscrire 5 espaces devant la valeur 12

```
cout << setw(7) << -12;
```

provoque l'ajout de 4 espaces devant le nombre –12.

setiosflags (ios :: mode1 | ios :: mode2 | ios :: mode3) : précise un ou des modes d'affichage

L'affichage d'un réel peut se faire en mode point flottant, *fixed* (ex : 3.1415) ou en notation scientifique, *scientific* (ex : 3.1415E+00). On précise le mode d'affichage d'un réel à l'aide de la fonction setiosflags(). ios :: fixed, comme argument de la fonction, précise le mode point flottant. La notation scientifique est obtenue en utilsant ios :: scientific dans l'argument. On peut préciser plus d'un argument en les séparant par l'opérateur | (ou binaire).

setprecision (i) : nombre de chiffres significatifs pour float et double

Exemples :

Instructions	Affichage
`cout << setiosflags (ios::fixed);`	
`cout << 3.14;`	`3.14`
`cout << setw(4) << setprecision(1) << 3.14159;`	`3.1`
`cout << setw(2) << setprecision(6) << 3.14159;`	`3.14159`

Si on veut par exemple afficher six décimales après le point même si le nombre à afficher n'en comporte pas autant dans sa partie fractionnaire, on utilise l'expression ios :: showpoint.

```
cout << setiosflags(ios::showpoint|ios::fixed);
cout << 3.14;                                    3.140000
cout << setw(3) << setprecision(1) << 3.14159;   3.1
cout << setw(2) << setprecision(6) << 3.14159;   3.141590
```

hex : nombres en base 16
dec : nombres en base 10
oct : nombre en base 8

exemple :

```
#include <iostream.h>
#include <iomanip.h>

int main ()
{
     int n = 12000;
     cout << "par défaut : " << n << "\n";
     cout << "en hexadécimal : " << hex << n << "\n";
     cout << "en décimal : " << dec << n << "\n";
     cout << "en octal : " << oct << n << "\n";
     cout << "en ???? : " << n << "\n";
     return 0;
}
```

exécution :

```
par défaut : 12000
en hexadécimal : 2ee0
en décimal : 12000
en octal : 27340
en ???? : 27340
```

Variables de type chaîne de caractères

```
cout << setw(i) << chaîne_de_caractères;
```

Si le nombre de caractères (i) spécifié est insuffisant pour que s'inscrivent tous les caractères, l'opérateur << ignore ce nombre et affiche la chaîne au complet. Si le nombre i dépasse la longueur de la chaîne, la chaîne est complétée avec des blancs par la gauche.

Autres manipulateurs :
endl : insère une fin de ligne

```
cout << "merci" << endl; équivalente à   cout << "merci" << "\n";
```

setfill(c) : Fixe le caractère de remplissage (blanc par défaut)
exemple :

```
cout << setfill('*') << setw(6) << 12;
```

L'affichage sera :

```
****12
```

On peut écrire nous-mêmes des manipulateurs, il suffit de faire des fonctions qui prennent un flot en entrée et qui retournent un flot en sortie, les deux par référence (nous reviendrons sur ce point lors de l'étude des fonctions).

Entrées /sorties sur fichiers

En C++, l'accès à des fichiers en entrée et en sortie se fait à l'aide des flots de fichiers d'entrée (*ifstream*), de sortie (*ofstream*), d'entrée/sortie (*fstream*).

Exemples

```
#include <fstream.h>
…
```

Déclaration de fichiers :

```
ifstream entree;              // fichier d'entrée
ofstream sortie;              // fichier de sortie
fstream entree_sortie;        // fichier d'entrée/sortie
```

Ouverture des fichiers : fonction open()

```
entree.open("donnees");
sortie.open ("resultats");
entree_sortie.open ("Fichier", ios::in|ios::out);
```

On peut tester si l'ouverture a réussi à l'aide de la fonction fail()

```
if ( entree.fail() )
     cout << "problème d'ouverture";

if ( entree.eof() )
     cout << "fin de fichier atteinte";
```

À l'ouverture, on peut spécifier comme deuxième paramètre certains modes particuliers. Voici les modes d'entrée-sortie possibles :

`ios::app`	écriture à la fin du fichier
`ios::in`	lecture au début
`ios::out`	écriture au début
`ios::nocreate`	le fichier doit exister à l'ouverture
`ios::noreplace`	le fichier ne doit pas exister
`ios::trunc`	efface un fichier existant.

```
#include <iostream.h>
#include <fstream.h>

int main ()
{
     fstream sortie;
     char nom[10];

     cout << "nom du fichier : ";
     cin >> nom;
     sortie.open(nom,ios::out|ios::noreplace);
```

```
     if (!sortie)
          cout << "le fichier existe déjà!" << endl;
     return 0;
}
```

Noter que `if (!sortie)` est équivalente à `if (sortie.fail())`

Lecture/écriture de fichiers
Les opérations d'entrée se font comme du clavier à l'aide de l'opérateur (>>)
Les opérations de sortie se font comme à l'écran à l'aide de l'opérateur (<<)

```
entree >> i >> x;                  // lecture
sortie << setw(4) << x << endl;    // écriture
entree_sortie >> j;
entree_sortie << x;                // Lecture suivie d'écriture
```

Fermeture des fichiers :
```
sortie.close();
entree.close();
```

Autres procédures d'entrées-sorties
Il existe un grand nombre de procédures d'entrées/sorties. Comme pour les fonctions open() et close(), l'appel se fait en préfixant le nom de la fonction appelée par le nom du flot concerné, les deux séparées par un ".".

`char get()` : lit le prochain caractère.
`put (char c)` : écrit le caractère c.
`read ((char*) &buf, int nb)` : lit nb caractères et les place à l'adresse buf en mémoire
`put(char c)` : Écrit le caractère c.
`putback (char c)` : recule d'un et remplace le caractère par c.
`write ((char *)&buf, int nb)` : écrit nb octets à partir de l'adresse buf en mémoire.