Chapitre 9 : Gestion des flux

Pour manipuler les flux d'entrées/sorties, le c++ met à notre disposition des opérateurs surchargés << et >> qui permettent respectivement d'afficher et de lire des données, ces opérateurs pourront parfaitement être redéfinis dans le cas de nos classes, ce qui permettra par exemple d'afficher le contenu des données membres d'une classe à l'écran, ou d'écrire ces mêmes variables dans un fichier.

1. Généralités sur les flux

Un flux représente un ensemble de données pouvant être manipulés à la fois en lecture ou à la fois en écriture.

Un flux, aussi appelé canal de données, offre une transparence vis-à-vis de la source ou de la destination des données :

- Ecran, fichier, mémoire pour les types de flux de sortie.

- Clavier, fichier, mémoire pour les types de flux d'entrée.

En C++, tous les flux sont symbolisés par des classes qui font partie de la librairie 'iostream.h'.

La classe de base est **iostream** et regroupe les caractéristiques communes aux flux d'E/S.

Les deux principales classes dérivées de la classe iostream sont **ostream** pour les flux de sortie, et **istream** pour les flux d'entrées.

Toute les classes de la librairie 'iostream.h' disposent des deux opérateurs surchargés **<<** et **>>**. L'opérande de gauche de l'opérateur << doit correspondre à un objet de la classe ostream, pour l'opérateur >> cet opérande doit être un objet de la classe istream.

Ces deux opérateurs ont été définis pour les types de données suivants : char, short, int, long, float, double, long double, char*, void*.

C++ fournie quatre flux prédéfinis pour afficher ou lire des informations.

cout **:** Flux de sortie standard (l'écran par défaut). Cette variable est un objet d'une classe dérivée de ostream.
cin **:** Flux d'entrée standard (le clavier par défaut). Cette variable est un objet d'une classe dérivée de istream.
cerr **:** Sortie erreur standard (l'écran par défaut). Cette variable est un objet d'une classe dérivée de ostream.
clog **:** Permet à la fois d'envoyer des messages d'erreur vers la sortie erreur standard (l'écran par défaut) et de remplir un fichier d'alerte. Cette variable est un objet de la classe 'ostream'.

2. Afficher sur l'écran avec 'cout'

Exemple

```
main()
{
      int e=37;
      float x=4,5;
      cout<<"debut \n";
      cout<<"Entier : "<<e<<endl;
      cout<<"Reel : "<<x<<endl;
}
```

Commentaire

- Ce programme utilise 3 fois l'objet 'cout' pour afficher du texte à l'écran.
- La librairie 'iostream.h' fournit un certain nombre de mots clés qui permettent de modifier les caractéristiques d'un flux. Ces commandes sont utilisées directement avec l'opérateur << en respectant la syntaxe : 'cout<<manipulateur ;'.

Manipulateur	Rôle
dec	Convertir en base décimale
hex	Convertir en base hexadécimale
oct	Convertir en base octale
ws	Supprimer les espaces
endl	Ajouter un saut de ligne en fin de flux (\n)
ends	Ajouter un caractère de fin de chaîne
flush	Vider un flux de sortie
setbase(int a)	Choisir la base (0, 8, 10 ou 16) . 0 est la valeur par défaut
setfill(int c)	Choisir le caractère de remplissage (padding)
setprecision(int c)	Indiquer le nombre de chiffres d'un nombre décimal
setw	Choisir la taille du champ (padding)

Exemple

```
int e=31;
cout<<e;
cout<<hex<<e;
```

Remarque

Le padding consiste à compléter généralement avec des espaces ou des zéros, un élément affiché à l'écran en précisant la taille d'un champ ou d'une colonne (set w). Tout élément affiché dont la taille est insuffisante sera complété par défaut par des espaces. De cette manière chaque élément sera affiché sur la même longueur de caractères.

En plus de ces manipulateurs, la classe iostream fournit un ensemble de méthodes.

Méthodes	Rôle
fill()	Renvoie le caractère de remplissage
fill(char c)	modifie le caractère de remplissage

`précision()`	renvoie la précision pour le nombre décimal
`précision(int n)`	modifie la précision pour le nombre décimal
`setf(long flag)`	modifie une propriété de formatage (format)
`setf(long flag,long champ)`	Modifie une propriété de formatage d'un champ
`width()`	Renvoie la valeur d'affichage
`width(int n)`	Renvoie la valeur d'affichage

Exemple

```
main()
{
    int e=15;
    cout<<"conversion"<<endl;
    cout<<"Entier : "<<e<<endl;
    cout<<"Hexadecimal : "<<hex<<e<<endl;
    cout<<"Octal : "<<oct<<e<<endl;

    cout<<"----------------------------------------\n";

    cout<<"Largeur et Padding\n";
    float x=3.15;
    cout.setf(ios::right,ios::adjustfield);
    cout<<"Decimal : "<<setw(10)<<x<<endl;
    cout<<"Decimal : "<<setw(10)<<x+1000<<endl;
    cout<<"Decimal : "<<setw(10)<<setfill('#') <<x<<endl;

    cout<<"----------------------------------------\n";

    cout<<"Nombre de chiffres\n";
    double pi=3.14151617;
    cout<<"PI : "<<pi<<endl;
    cout<<"PI : "<<setprecision(1)<<pi<<endl;
    cout<<"PI : "<<setprecision(6)<<pi<<endl;

    cout<<"----------------------------------------\n";

    cout<<"Chaînes, Alignement et Padding\n";
    char texte[50];
    strcpy(texte,"Bonjour tout le monde");
    cout<<texte<<endl;
    cout.setf(ios::left,ios::adjustfield);
    cout.width(50);
    cout.fill('-');
    cout<<texte<<endl;
    cout.width(40);
    cout.fill('*');
    cout<<texte;
}
```

Exécution

```
conversion
Entier : 15
Hexadecimal : f
Octal : 17
----------------------------------------
Largeur et Padding
Decimal :      3.15
Decimal :    1003.15
Decimal : ######3.15
----------------------------------------
Nombre de chiffres
PI : 3.14152
PI : 3
PI : 3.14152
----------------------------------------
Chaînes, Alignement et Padding
Bonjour tout le monde
Bonjour tout le monde----------------------------
Bonjour tout le monde********************
```

3. Saisir au clavier avec 'cin'

Dans l'exemple suivant :

```
char chaine[10];
cout<<"saisir une chaine : ";
cin.width(sizeof(chaine));  cin>>chaine;
```

La fonction 'width' fixe le nombre de caractères à affecter à la variable 'chaine' sans débordement.

La classe istream contient la fonction membre **getline** qui permet de saisir des chaînes de caractères comprenant des espaces non acceptés par l'opérateur >>.

Le prototype de getline est :

```
istream &getline(char* chaine, int nombre, char fin= '\n');
```

Avec :

chaine : chaîne à saisir.

nombre : nombre maximum de caractères de la chaîne.

fin : caractère de fin de saisie (par défaut '\n').

Exemple

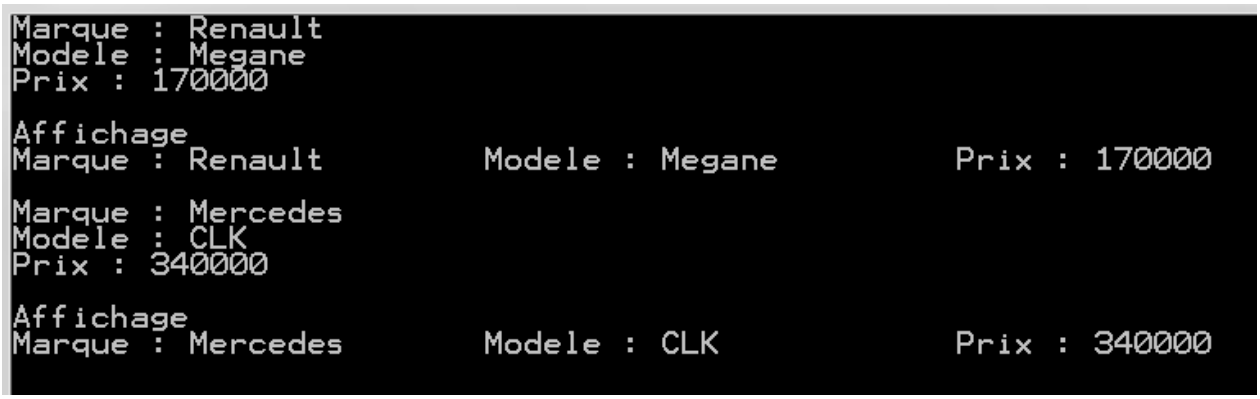
```
class voiture
{
       char marque[20];
       char modele[20];
       char prix[20];
     public:
       voiture(char *ma="",char *mo="",char *pr="")
       {
          strcpy(marque,ma);
          strcpy(modele,mo);
          strcpy(prix,pr);
       }
       void saisie();
       void affiche();
};

//-------------------------------------------------------------
void voiture::saisie()
{
     cout<<"Marque : ";  cin.getline(marque,sizeof(marque));
     cout<<"Modele : ";  cin.getline(modele,sizeof(modele));
     cout<<"Prix : ";    cin.getline(prix,sizeof(prix));
}
//-------------------------------------------------------------
void voiture::affiche()
{
     cout.setf(ios::left,ios::adjustfield);
     cout<<"Marque : "; cout.width(15); cout<<marque;
     cout<<"Modele : "; cout.width(15); cout<<modele;
     cout<<"Prix : ";   cout.width(15); cout<<prix; cout<<endl;
}
//-------------------------------------------------------------
main()
{
     voiture v1;
     v1.saisie();
     cout<<"\nAffichage \n";
     v1.affiche();

     cout<<endl;

     voiture v2;
     v2.saisie();
     cout<<"\nAffichage \n";
     v2.affiche();
}
```

Exécution



4. Redéfinir les opérateurs de flux

Pour redéfinir les opérateurs de flux, on doit respecter une syntaxe telle qu'elle est définie dans l'exemple suivant :

Exemple

```
class demo
{
     public :
       friend ostream& operator<<(osream&,demo&);
       friend istream& operator>>(isream&,demo&);
};
Ostream & operator<<(osream& …,demo& …)
{ …  }
Istream & operator>>(isream& …,demo& …)
{ …  }
```

Commentaire

- Les deux opérateurs << et >> sont surdéfinis en tant que fonctions amies de la classe 'demo', de cette manière, ils pourront accéder à tous ses membres.
- Dès que l'opérateur << sera utilisé avec comme opérande de gauche un objet de type 'ostream' (ou dérivé) et comme opérande de droite, un objet de type 'demo' ce sont les instructions de la fonction **operator <<** qui seront exécutées.
- L'opérateur >> est utilisé d'une façon analogue à celle de l'opérateur <<.
- Les opérateurs << et >> peuvent être surdéfinis pour des objets dynamiques.

Exemple

```
class voiture
{
       char marque[20];
       char modele[20];
       char prix[20];
     public:
       voiture(char *ma="",char *mo="",char *pr="")
       {
          strcpy(marque,ma);
          strcpy(modele,mo);
          strcpy(prix,pr);
       }
       friend ostream& operator<<(ostream&,voiture&);
       friend istream& operator>>(istream&,voiture&);
};
//-----------------------------------------------------------
ostream & operator<<(ostream& out,voiture& v)
{
     out.setf(ios::left,ios::adjustfield);
     out<<"Marque : "<<v.marque;
     out<<"\tModèle : "<<v.modele;
     out<<"\tPrix : "<<v.prix<<endl;
     return out;
}
//-----------------------------------------------------------
istream & operator>>(istream& in,voiture& v)
{
     cout<<"Marque : "; in.getline(v.marque,sizeof(v.marque));
     cout<<"Modele : "; in.getline(v.modele,sizeof(v.modele));
     cout<<"Prix : ";     in.getline(v.prix,sizeof(v.prix));
     return in;
}
//-----------------------------------------------------------
main()
{
  voiture v1("Mercedes","CLK 500","19900DH");
  cout<<v1;
  cout<<"Saisir la marque, le modele et le prix de voiture\n";
  cin>>v1;
  cout<<"\nAffichage"<<endl;
  cout<<v1;
  voiture v2;
  cout<<"Saisir la marque, le modele et le prix de voiture\n";
  cin>>v2;
  cout<<"\nAffichage"<<endl;
  cout<<v2;
  cout<<v1<<v2;
}
```

Exécution

```
Marque : Mercedes       Modele : CLK 500        Prix : 19900DH
Saisir la marque, le modele et le prix de voiture
Marque : Peugeot
Modele : P308
Prix : 165000

Affichage
Marque : Peugeot        Modele : P308   Prix : 165000
Saisir la marque, le modele et le prix de voiture
Marque : Volkswagen
Modele : Polo
Prix : 175000

Affichage
Marque : Volkswagen     Modele : Polo   Prix : 175000
Marque : Peugeot        Modele : P308   Prix : 165000
Marque : Volkswagen     Modele : Polo   Prix : 175000
```

Remarque

Dans le cas d'un pointeur vers la classe voiture, il faut redéfinir les opérateurs << et >> comme par exemple :

```
ostream& operator<<(ostream& out,voiture *v)
{
     out.setf(ios::left,ios::adjustfield);
     out<<"\tMarque : "<<v->marque;
     out<<"\tModèle : "<<v->modele;
     out<<"\tPrix : "<<v-> prix<<endl;
     return out;
}
```

5. Lire à partir d'un fichier ou écrire dans un fichier

Il suffit de créer un objet de type **ofstream** pour écrire dans un fichier et un objet de type **ifstream** pour lire un fichier. Ces deux classes sont déclarées dans le fichier 'fstream.h' et elles dérivent respectivement des classes ostream et istream

Exemple 1

Extrait d'un programme qui permet d'écrire dans un fichier.

```
ofstream fs("c:\\demo.txt");
if(!fs)
{
     cout<<"Erreur d'écriture dans ce fichier ! \a\a";
     return 1;
}
fs<<"valeur de la donnée: "; fs<<17.12; fs<<"Fin";
fs.close();  …
```

Commentaire

- Un objet ‘fs’ de type ostream est créé pour accéder en écriture au fichier “c:\\demo.txt“.
- En cas de succès de l’ouverture du fichier ‘demo.txt’, les données sont écrites vers le flux de fichier ‘fs’.

Exemple 2

Extrait d’un programme permettant de lire un fichier

```
ifstream fe("c:\\demo.txt");
if(!fe)
{
     cout<<"Erreurs d'ouverture ! \a\a";
     return 1;
}
char texte1[30];
double x;
char texte2[30];
fe>>texte1; fe>>x; fe>>texte2;
cout<<texte1<<x<<texte2;
fe.close();   …
```

Commentaire

- Un objet ‘fe’ de type ‘ifstream’ est créé pour accéder en lecture au fichier ‘demo.txt’.
- Le flux ‘fe’ est utilisé pour récupérer les données qui sont affichées ensuite sur l’écran.

Remarque

La fonction **close**, qui permet de fermer un fichier, est appelée automatiquement par le destructeur des classes ofstream et ifstream.

Exercice

Ecrire un programme permettant de rassembler les deux extraits.

Le programme :

```
main()
{
    // Ecriture dans le fichier
    ofstream fs("c:\\Exemple\\demo.txt");
    if(!fs)
    {
          cout<<"Erreur d'écriture dans le fichier ! \a\a";
          getch();
          return 1;
    }
    else
    { cout<<"Fichier cree...\n"; }
    fs<<"valeur-de-la-donnée-:*"<<endl;
    fs<<17.12<<endl;
    fs<<"*Fin";
    fs.close();
    // Lecture du fichier
    ifstream fe("c:\\Exemple\\demo.txt");
    if(!fe)
    {
          cout<<"Erreur de lecture du fichier ! \a\a";
          getch();
          return 1;
    }
    else
    { cout<<"Fichier ouvert...\n"<<endl; }
    char texte1[30];
    char texte2[30];
    double x;
    fe>>texte1;
    fe>>x;
    fe>>texte2;
    cout<<texte1<<x<<texte2;
    fe.close();

    getch();
}
```

Exécution