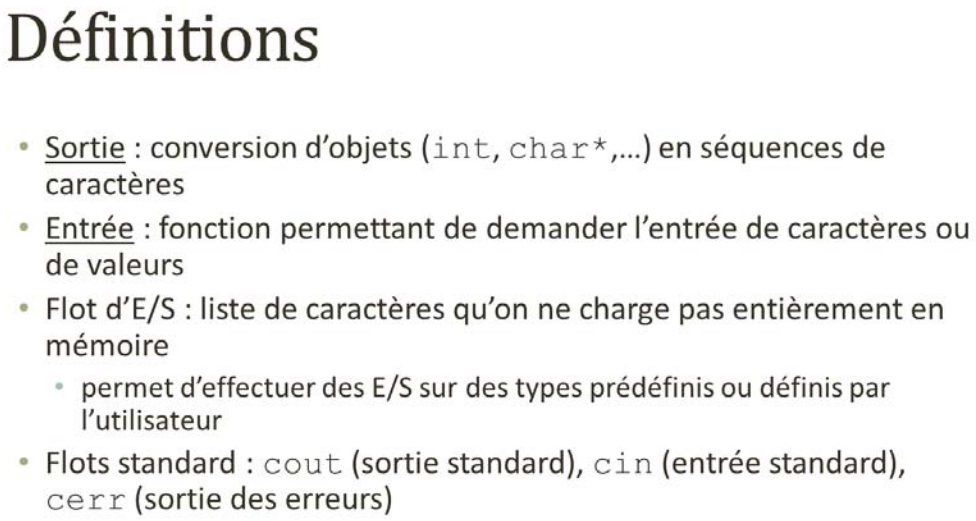
Chapitre 6. Les flots d'entrées/sorties.

Introduction

Les flots d'entrées/sorties en C++ permettent de formater les entrées et sorties standard et non standard aussi bien sur des types prédéfinis que sur des types définis par l'utilisateur.

On parlera indifféremment de flots ou bien de flux.

Définitions



Slide 2

Il existe un tampon pour chaque flot dans lequel on place les caractères en attente : ils ne sont donc pas tous chargés en mémoire.

Généralités sur les flots

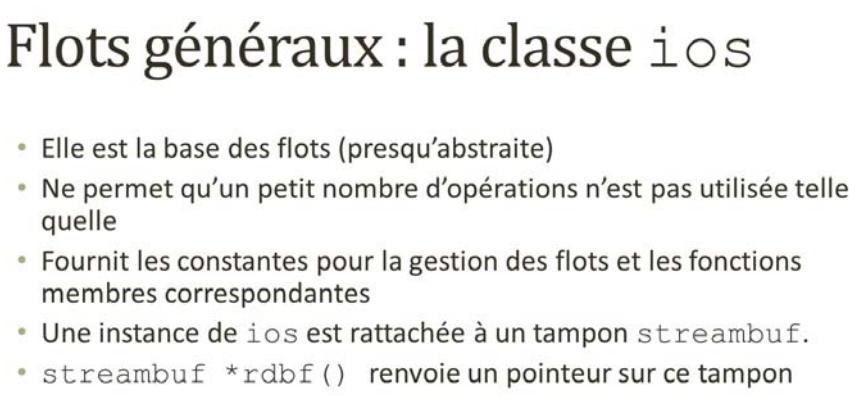
Slide 3

Les flots d'entrées/sorties sont de différentes natures, de différents niveaux de raffinement et constituent une hiérarchie d'héritage. Nous allons les examiner des plus généraux au plus spécifiques.

Les flots généraux

Les flots généraux sont contenus dans la classe ios (pour In Out Stream, *i.e.* flots d'entrées sorties).

La classe ios est presque une classe abstraite, c'est la base des flots et elle n'est généralement pas utilisée telle quelle.



Slide 4

Notion d'état d'un flot

L'état des flots est donné par un ensemble de valeurs, qui se réduisent parfois à un simple bit et dont on peut tester la valeur grâce à des méthodes accesseurs fournies par la classe. Les erreurs et conditions non standard peuvent être traités en définissant et en testant l'état du flot de façon appropriée.

Slide 5

Tester l'état d'un flot (1/2)

- L'opérateur `!` (redéfini pour la classe `ios`) renvoie
  - 1 en cas de flot incorrect
  - 0 sinon
- L'opérateur `void*` renvoie
  - 0 en cas de flot incorrect
  - Un pointeur non nul sinon
- Utilisation :

```
iostream fl; // ...
if (fl) cout << "Tout va bien !\n"; // ...
if (!fl) cout << "Une erreur s'est produite.\n";
```

Slide 6

Extrait de l'en-tête de la classe ios :

```
Class ios {
        Public :
        //…
        operator void*();
        int operator !();
        //…
};
```

Les indicateurs de l'état d'un flot sont pour la plupart de simples bits positionnés à 0 ou 1.
La manipulation directe de ces indicateurs permet de gérer les problèmes de flots en cours d'exécution, et notamment de conserver un comportement stable au programme en cas d'incident lié aux flots d'entrées/sorties.

Tester l'état d'un flot (2/2)

- Un flux est représenté par un ensemble d'indicateurs (bits)
- Indicateurs (ou flags) définis dans `ios_base` :

Drapeaux	
`badbit`	Indique que le flot est corrompu
`eofbit`	Indique la fin de fichier
`failbit`	Indique que l'opération suivante va échouer
`goodbit`	Indique que l'opération suivante peut réussir

- Possibilité de manipuler directement les indicateurs d'état d'entrées/sorties (gestion des pbs de flots en cours d'exécution) :

```
//…
ios_base::iostate s = cin.rdstate();
// récupère l'ensemble de bits iostate pour le flux cin
if (s&ios_base::badbit) {/*traitement erreur */} //pb avec cin
```

Slide 7

Il existe également des indicateurs et des valeurs entières permettant de formater les entrées et sorties. Ils sont définis dans les classes basic_ios et ios_base.

Formatage d'un flot

- Les drapeaux permettent de modifier le format d'un flux
- Méthodes pour manipuler les drapeaux : dans les classes `basic_ios` et `ios_base`
- Méthodes manipulant les drapeaux gérant l'état du format : `flags(),setf(),unsetf(),copyfmt()`

Slide 8

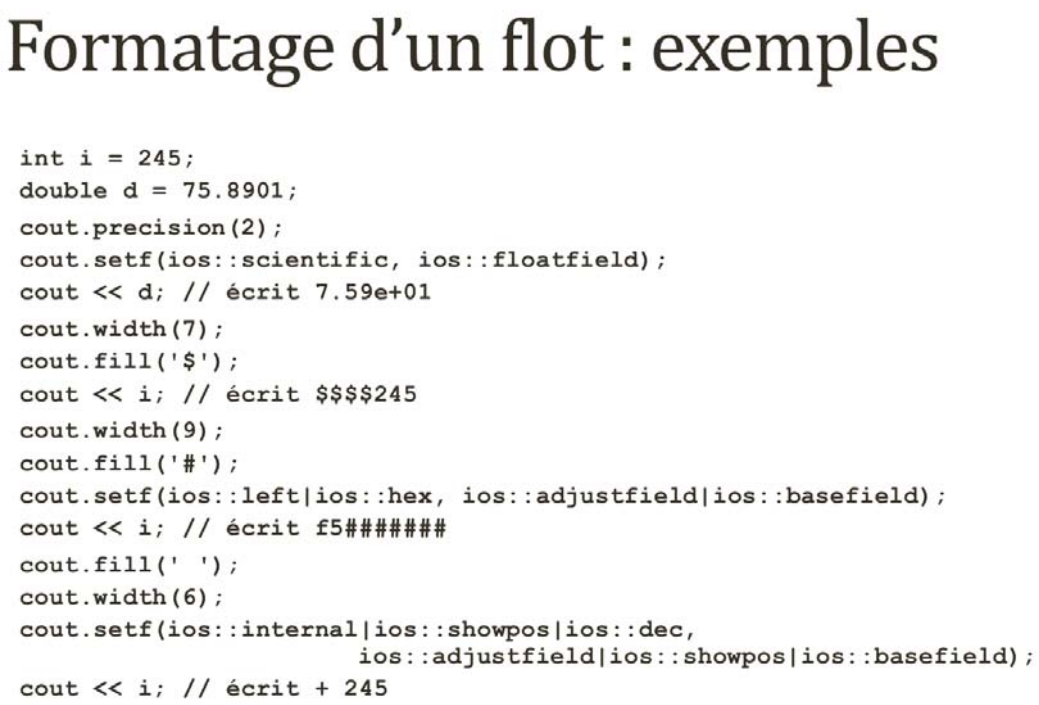
Drapeaux disponibles dans `ios_base`

Flag	Signification quand le champ est à 1
`ios::skipws`	Supprime les espaces en lecture. (1 par défaut)
`ios::left`	Ajustement à gauche en écriture.
`ios::right`	Ajustement à droite en écriture.
`ios::internal`	Remplissage après le signe + ou -, ou l'indicateur de base
`ios::dec`	Ecriture décimale (base 10).
`ios::oct`	Écriture en octal (base 8).
`ios::hex`	Ecriture en hexadécimal (base 16).
`ios::showbase`	En écriture, écrire un indicateur de base.
`ios::showpoint`	Ecrit le point décimal pour les nombres à virgule flottante, même si les décimales sont nulles.
`ios::uppercase`	Ecrit les lettres `A` à `F` en majuscules dans les chiffres hexadécimaux (minuscules sinon).
`ios::showpos`	Ecrit le signe pour tous les entiers, même positifs.
`ios::scientific`	Ecriture en notation scientifique (ex : `1.75e+01`).
`ios::fixed`	Ecriture en notation à virgule flottante (ex : `17.5`).
`ios::unitbuf`	Vide les tampons après écriture.
`ios::stdio`	Vide les tampons de sortie standard et de sortie d'erreur après insertion.

slide 9

on peut consulter ces drapeaux pour connaître le formatage du flot considéré ou modifier leur valeur pour changer le comportement du flot.

Exemples de formatage de flot



Slide 10

Chacune de ces méthodes est rattachée au flot cout.

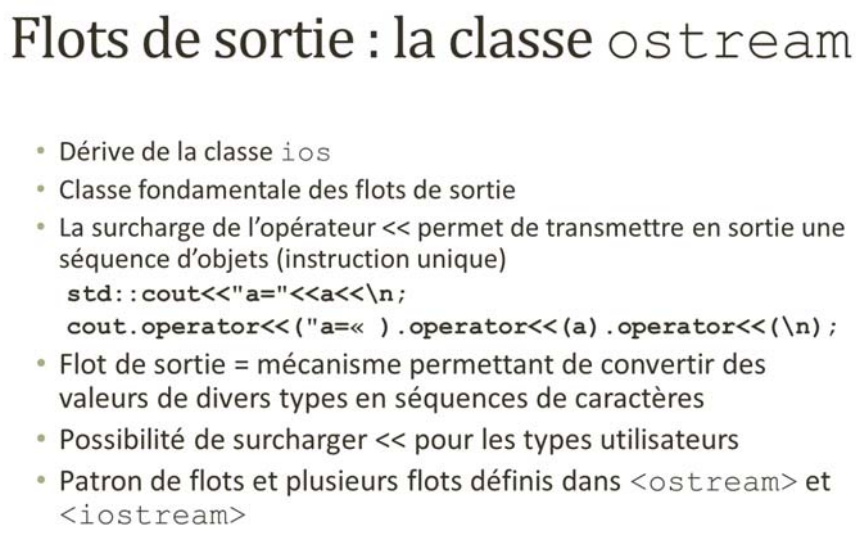
La méthode precision concerne le nombre maximal de décimales écrites dans les nombres à virgule flottante.
int precision() donne ce nombre
int precision(int) le modifie

Setf permet de choisir un format par défaut pour le flot.

La méthode width concerne la largeur (en caractères) sur laquelle seront écrits les caractères du flot
int width() la donne
int width(int) la modifie

La méthode fill concerne les caractères de remplissage (i.e. les caractères qui vont "remplir les blancs" si le nombre de caractères inscrits est inférieur à la largeur)
char fill() donne le caractère de remplissage actif
char fill(char) permet de le modifier.

Les flots de sortie et la classe ostream



Slide 11

La classe ostream est la classe fondamentale des flots de sortie. Elle dérive de manière publique et virtuelle de la classe ios.
Il s'agit d'un flot retardé : les données prennent place dans le tampon jusqu'à ce qu'il soit plein ou jusqu'à la fermeture du flot. Si l'on souhaite écrire immédiatement le contenu du tampon, il faut utiliser la méthode ostream & flush (void)

ostream & operator << (type) renvoie une référence sur le flot courant, ce qui permet de chaîner les instructions…
La ligne : `std::cout<<"a="<<a<<\n;`
Peut s'écrire : `cout.operator<<("a=").operator<<(a).operator<<(\n);`
Nous verrons un exemple de surcharge de << plus loin… ce qui est très utile pour les types utilisateur.

Plusieurs flux de sortie standard sont définis dans la classe iostream, parmi lesquels, notamment, les flux associés à la sortie standard et aux messages d'erreur. Il est possible de créer d'autres flux en les faisant dériver de ces derniers, par exemple.

Slide 12

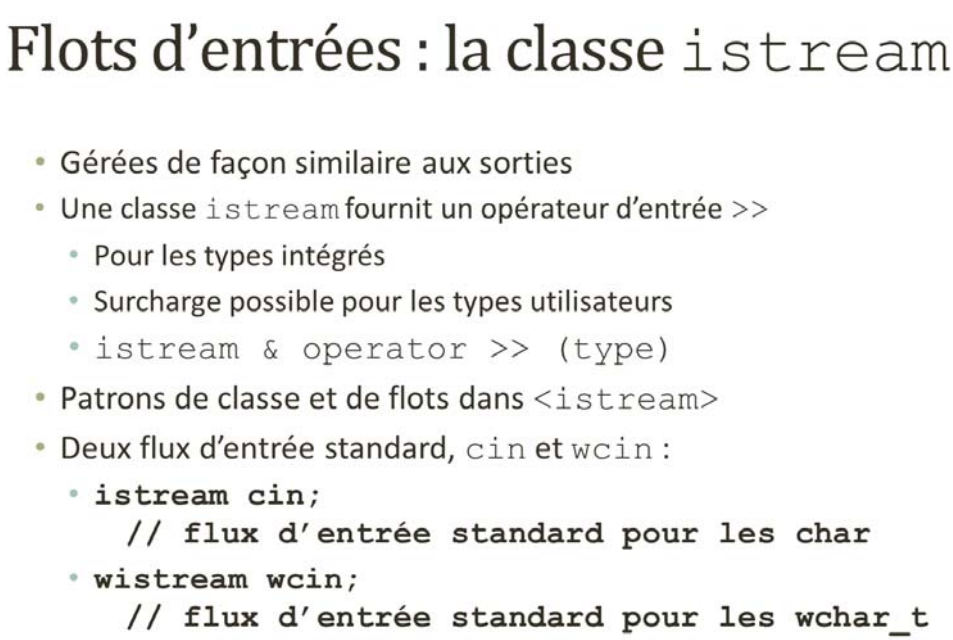
cout est l'équivalent de "stdout" en c.
cerr et clog sont l'équivalent de "stderr" en C et ont la même destination de sortie.
Il est possible de créer ses propres flux, en dérivant autant que possible des classes standard.

Les flots d'entrée : la classe istream

Les flots d'entrée sont, comme leur nom l'indique, destinés à récupérer des données en entrée, qu'elles soient issues de saisies clavier, de fichiers ou encore de capteurs…

Ces flots sont gérés de façon similaire aux flots de sortie

Les flots d'entrée sont définis dans les classes istream et iostream.



Slide 13

Fonctionnement de l'opérateur >>

Syntaxe de l'opérateur >> :
istream & operator >> (type)

Il fonctionne comme << et peut donc être chaîné.

cin est l'équivalent de stdin en C.
Le flux d'entrée wcin est défini dans la classe <iostream> et est destiné aux caractères étendus.

Sur un flux d'entrée, on peut choisir de lire les données de façon formatée ou non...

Flots d'entrées : lecture non formatée

- Plusieurs méthodes pour la lecture non formatée :
  - `istream &get(char*, int max, char = '\n');`
    // lit une série de caractères & les place dans un tableau. S'arrête quand le nombre maximal est dépassé ou quand le caractère de fin de ligne est rencontré. Ajout de \0.
  - `istream &read(char*, int max);`
    // lit un bloc de caractères de la longueur précisée sans aucun formatage
  - `istream &getline(char*, int max, char = '\n');`
    // même chose que get si le dernier argument n'est pas précisé. Sinon, arrêt quand on rencontre le caractère passé en paramètre OU \n

Slide 14

On peut récupérer un seul caractère ou plusieurs, de différentes manières.

Flots d'entrée : lecture formatée (1/2)

- Pour `short`, `int`, `long` (signés ou non) :
  - Espaces sautés
  - Lecture du signe et des chiffres jusqu'au prochain caractère autre que chiffre
  - Préfixes acceptés (lecture en octal, décimal, ...)
  - Quand le nombre de chiffres entrés est important, le résultat est obtenu modulo 65536
- Pour les nombres à virgule flottante :
  - Espaces initiaux sautés
  - Lecture conforme aux règles d'écriture d'un nombre à virgule flottante, jusqu'à ce qu'un caractère soit incorrect
  - Si la valeur entrée est supérieure à la valeur maximale, elle est remplacée par cette dernière dans la variable

Slide 15

Flots d'entrée : lecture formatée (2/2)

- Pour les caractères :
  - Espaces initiaux sautés (sauf si utilisation de get)
  - Lecture d'un seul caractère (pour cin, il faut un retour chariot pour finir l'entrée)
- Pour les chaînes de caractères `char*` :
  - Espaces initiaux sautés
  - Caractères suivants placés dans la chaîne jusqu'au prochain caractère d'espacement
  - Zéro final ajouté
- Dans tous les cas, si la fin de fichier est rencontrée en cours de lecture :
  - Rien n'est placé dans la variable
  - Le bit `failbit` est positionné

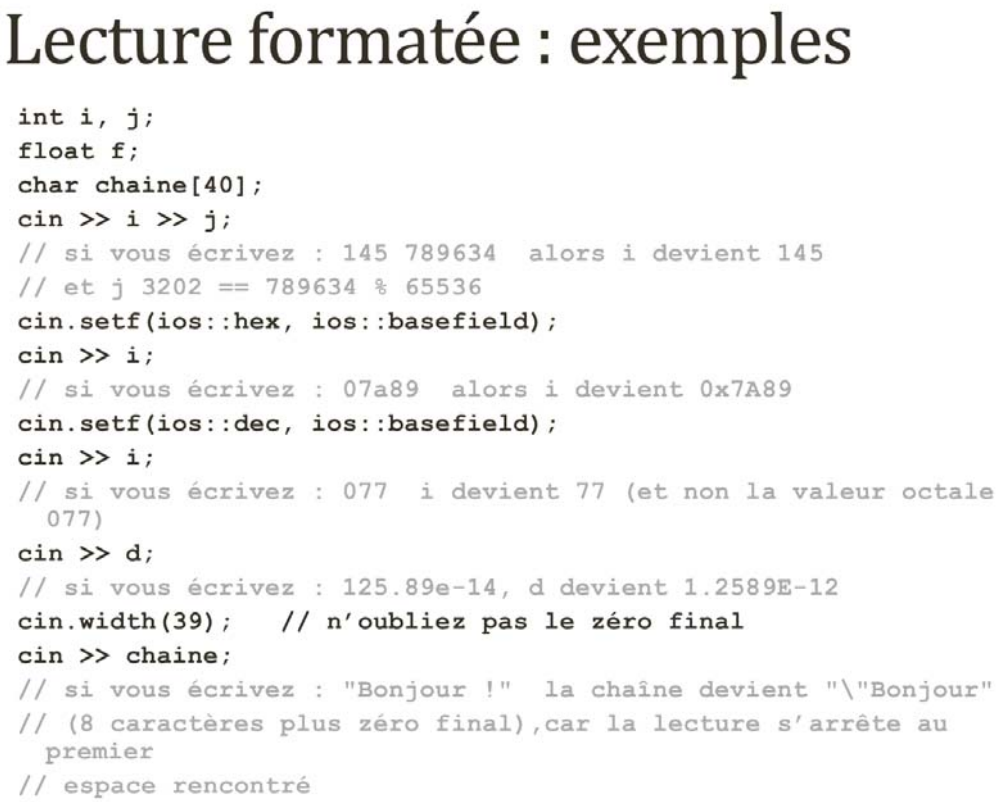
  => il est ainsi possible de gérer l'erreur en lecture

Slide 16

Pour lire un caractère unique, on utilise les méthodes suivantes :

```
istream & get (char&);
int get(void);
```

Exemples de lecture formatée



Slide 17

`cin.setf(ios::hex, ios::basefield);` `-> lecture en format hexadécimal`

`cin.setf(ios::dec, ios::basefield);` `-> lecture en format décimal, les 0 en début de ligne seront ignorés.`

En outre, il existe des flots acceptant à la fois des entrées et des sorties : on parle alors de flots mixtes.

Flots mixtes : la classe `iostream`

- Patrons dans <iostream>
- Hérite à la fois de ostream et de istream
- Classe utilisée pour faire à la fois des entrées (lectures) et des sorties (écritures)
- << et >> sont disponibles

Slide 18

La classe iostream hérite à la fois de ostream et de istream (un des intérêts de l'héritage multiple)
Cette classe fournit également un ensemble de manipulateurs pour permettre de formater les entrées/sorties de façon simplifiée…

Les manipulateurs

- Un manipulateur s'insère entre des objets lus ou écrits et modifie l'état du flot (simplification du formatage)
- Dans : <ios>, <istream>, <ostream>, <iostream>, <iomanip>
- Liste des manipulateurs :

boolalpha/noboolalpha	right	flush
showbase/noshowbase	dec	ws (ignore les espaces)
showpoint/noshowpoint	oct	resetiosflags/setiosflags
showpos/noshowpos	hex	setbase(int)
skipws/noskipws	fixed	setfill(int)
uppercase/nouppercase	scientific	setprecision(int)
internal	endl ('\n' et flush)	setx(int)
left	ends ('\0' et flush)	

Slide 19

Les manipulateurs : exemple

```
#include <iostream>
#include <iomanip>

int main( )
{
  double d=1234.5678 ;
  std::cout<<std::setprecision(6)<<d<<std::endl;
  //1234.57
  std::cout<<std::setfill('#') <<std::setw(12) <<d<<
  std::endl; //# # # # 1234.57
  return 0;
}
```

Slide 20

Enfin, les flots peuvent permettre de gérer les données à récupérer ou entrer dans un fichier.

Flots et fichiers (1/2)

- Possibilité de définir des flux entre fichiers

```
#include <fstream>
#include <cstdlib>
int main(int argc, char*argv[]) {
  if (argc!=3) return 1 ;
  std::ifstream from(argv[1]);
       //ouvre le flux de fichier en entrée
  if (!from) return 1 ; //ouverture de fichier impossible
  std::ofstream to(argv[2]) ;//ouvre le flux de fichier en sortie
  if (!to) return 1 ; //ouverture de fichier impossible
  char ch ;
  while (from.get(ch)) to.put(ch) ;
  if (!from.eof() || !to) return 1; //incident en cours de copie
  return 0;
}
```

Slide 21

Le programme proposé ici copie un fichier dans un autre (les noms sont passés sous forme d'arguments de ligne de commande)

Ces flots sont définis dans la classe fstream (f pour file)

Flots et fichiers (2/2)

- **<fstream>** contient les définitions de :
  - **ifstream** (flot d'entrée à partir d'un fichier)
  - **ofstream** (flot de sortie vers un fichier)
  - **fstream** (possibilité de lecture et d'écriture)
- Ouverture d'un fichier en entrée : créer un objet de la classe **ifstream** avec le nom du fichier comme argument
- Ouverture d'un fichier en sortie : créer un objet de la classe **ofstream** avec le nom du fichier comme argument
- Mode d'ouverture passé en paramètre aux constructeurs

Drapeaux	
app	Ajout
ate	Ouvre et se place à la fin du fichier
binary	Mode binaire
in	Ouvre en lecture
out	Ouvre en écriture
trunc	Tronque le fichier à une taille nulle

slide 22

En plus des fichiers, un flot peut évidemment être rattaché à des chaînes de caractères issues de la classe string.

Flots et chaînes

- Les flots peuvent être attachés à des chaînes de caractères (**string**). Déclarés dans **<sstream>** :
  - **istringstream**
  - **ostringstream**
  - **stringstream**
- ... ou à des tableaux de caractères. Déclarés dans **<strstream>** :
  - **istrstream**
  - **ostrstream**
  - **strstream**
- Possibilité de définir des flots lisant et écrivant directement dans des tableaux de caractères

Slide 23

Surcharge des opérateurs d'entrée/sortie

Enfin, tout comme il est pertinent de surcharger les opérateurs classiques de type affectation, addition et autres pour les types construits, il est possible de surcharger les opérateurs << et >>.

slide 24

Les opérateurs << et >> sont déclarés comme des fonctions amies et accèdent donc aux attributs des flots ; ce ne sont pas des fonctions membres de la classe complexe. Ils renvoient une référence vers un flux pour permettre de chaîner les appels à l'opérateur (et donc avoir des écritures du type cout << i << j << k ;).

Lors de la surcharge des opérateurs << et >> ici, on a tout simplement besoin d'appeler ce même opérateur sur chacun des membres de la classe en le reliant au flux os ou is passé en paramètre.