Langage C++

2/2

Rémy Leconge- Rachid Jennane

Maths-info 2 semestre S2

Les tableaux

Les Tableaux (1)

Un tableau est une donnée structurée dont tous les éléments sont de même type.
Un tableau est caractérisé par 3 éléments :

- son identificateur,
- son type et
- sa dimension.

Il y a les tableaux unidimensionnels et les tableaux multidimensionnels.

Déclaration d'un tableau automatique

Syntaxe : type identificateur [dimension]; // tableau unidimensionnel
type identificateur [dim1] [dim2] [dim3] // multidimensionnel

Déclaration d'un tableau dynamique

Syntaxe : type* identificateur ; // tableau unidimensionnel
type** identificateur ; // multidimensionnel

Les Tableaux (2)

Remarques :

- le premier indice d'un tableau est **toujours** 0, que le tableau soit automatique ou dynamique.
- Il est possible de ne pas donner de dimension lors de la déclaration d'un tableau de définir la dimension par la suite avec l'opérateur new.
- La réservation mémoire d'un tableau unidimensionnel par new se fait de la manière suivante :

  new type **[**dimension**]**;

- La désallocation mémoire d'un tableau unidimensionnel par delete se fait de la manière suivante :

  delete [] identificateur ;

Les Tableaux (3)

Remarques (suite):

- La réservation mémoire d'un tableau bidimensionnel par new se fait de la manière suivante :

```
identificateur = new type* [dimension1];
for(int i=0;i<dimension1;i++)
{
        identificateur[i]= new type[dimension2];
}
```

- La désallocation mémoire d'un tableau bidimensionnel par delete se fait de la manière suivante :

```
for(int i=0;i<dimension1;i++)
{
        delete [] identificateur[i] ;
}
delete [] identificateur;
```

- Les tableaux multidimensionnels, qu'ils soient automatiques ou dynamiques, peuvent être vectorisés. La déclaration double a[3][6]; très utilisée en C++, conduit à a[i][j] ≡ a[i*6+j]; (6 groupes de 3 éléments chacun).

Les Tableaux (4)

Exemple

```
int ti[8];                          // tableau de 8 entiers
char  tc[80];                       // tableau de 80 caractères
double  td[80];                     // tableau de 80 doubles
unsigned char ima[512][512];        // tableau carré 512*512 (pixels)

for (int i=0; i<10; i++)            // initialisation de td
        td[i] = (double) i;

for (i=0; i<512; i++)               // initialisation de ima
      for (j=0; j<512; j++)
            ima[i][j] = 0;
```

Les Tableaux (5)

Relation entre tableaux et pointeurs

En C++, les pointeurs et les tableaux sont des variables de même nature. La seule différence est que s'il est possible d'incrémenter la valeur d'un pointeur (p++), cela est interdit pour un tableau.

int t[10];	ou	int *t = new int [10] ;
t	est identique à	&(t[0])
t	est l'adresse de	t[0]
t+1	est l'adresse de	t[1]
t+i	est l'adresse de	t[i]
*(t+i) = 1	est identique à	t[i] = 1

Surcharge d'opérateurs

Surcharge d'Opérateurs

- L'écriture utilisée pour travailler sur des instances d'objet est souvent lourde et peu lisible. En C++, les opérateurs classiques peuvent être redéfinis.

  Exemple

  Complex a;

  cout<<a<<endl; est moins déroutant que a.affiche (); dans ce sens que tout les autres affichages (entier, réel, chaine de caractère) sont réalisés avec l'instruction cout.

- Un opérateur surchargé est appelé fonction opérateur, Il est déclaré avec le préfixe operator.

Règles de Surcharge d'Opérateurs

Règles générales de surcharge d'opérateurs

Deux règles fondamentales régissent la surcharge des opérateurs unaires ou binaires :

- **si un opérateur est membre d'une classe, son premier opérande est toujours du type de la classe à laquelle il appartient**.

- **quand un opérateur n'est pas membre d'une classe, au moins un de ses opérandes doit être de type classe**.

Surcharge de l'Opérateur d'Affectation

- **Lorsque les données membres d'une classe contiennent des pointeurs sur des variables dynamiques, il est indispensable de redéfinir l'opérateur d'affectation =**
- Il est judicieux de passer l'unique paramètre en référence constante pour ne pas risquer de le modifier.

Exemple :

```
class CPlx
{
  ....
  public:
        CPlx operator = (const CPlx &);
}

 CPlx CPlx ::operator = (const CPlx & x)
{
         re = x.re; im = x.im;
         return (*this);
}
void main (void)
{
        CPlx x1, x2, x3, x4(4,4);
         ...
         x1 = x2 = x3 = x4;
}
```

Surcharge de l'Opérateur []

Exemple :

```
class CTab
{
        private:
            double t[TMAX];
        public:
            double& operator [] (int);
} ;
double& CTab::operator [] (int i)
{
        if ( (i < 0) || (i > TMAX-1) )
            {cerr << "Débordement de tableau" << endl; return; }
        return t[i];
}
void main (void)
{
        CTab t;
        t[4] = t[3];
}
```

Remarque :

L'utilisation de **operator ( )** est tout à fait similaire à celle de l'opérateur [ ].

Surcharge de l'Opérateur d'insertion

- **Cet opérateur est déjà utilisé pour tous les affichage de types prédéfinis (entier, réel, chaine de caractère). En le surchargeant on peut gérer l'affichage de n'importe quelle classe.**
- Sa syntaxe est assez particulière par rapport aux autres opérateurs :

Exemple :

```
#include <iostream>
using namespace std;
class CTab
{
 private:
        int T[TMAX];
public:
        CTab();
        friend ostream& operator <<(ostream& os, CTab& tb);
} ;
CTab::CTab()
{
     for(int i=0;i<TMAX;i++)
     { T[i]=i;}
}
```

```
ostream& operator <<(ostream& os, CTab& tb)
{
    for(int i=0;i<TMAX;i++)
     { os<<tb.T[i]<<endl;}
    return os;
}


void main (void)
{
         CTab t;
        cout<<t<<endl;
}
```

Surcharge de l'Opérateur d'extraction

- **Cet opérateur est utilisé pour tous les saisies de types prédéfinis (entier, réel, chaine de caractère). En le surchargeant on peut gérer la saisie d'objet de n'importe quelle classe.**
- Comme la surcharge de l'opérateur d'insertion sa syntaxe est assez particulière par rapport aux autres opérateurs :

Exemple :

```
#include <iostream>
using namespace std;
class CTab
{
 private:
        int T[TMAX];
public:
        CTab();
        friend istream& operator >>(istream& is, CTab& tb);
} ;
CTab::CTab()
{
     for(int i=0;i<TMAX;i++)
     { T[i]=i;}
}
```

```
istream& operator >>(istream& is, CTab& tb)
{
    for(int i=0;i<TMAX;i++)
     { is>>tb.T[i];}
    return is;
}


void main (void)
{
         CTab t;
        cout<<t<<endl;
}
```

Flux

Flux

Les fonctions d'entrées/sorties sont implémentées dans une libraire standard C++ appelée I/O stream [Input/Output stream ≡ flots d'entrées/sorties]. En C++, les entrées/sorties sont des échanges de séquences de caractères ou octets, appelées flots de données. Un flux est un périphérique, un fichier ou une zone mémoire, qui reçoit (flux d'entrée) ou au contraire qui fournit (flux de sortie) un flot de données.

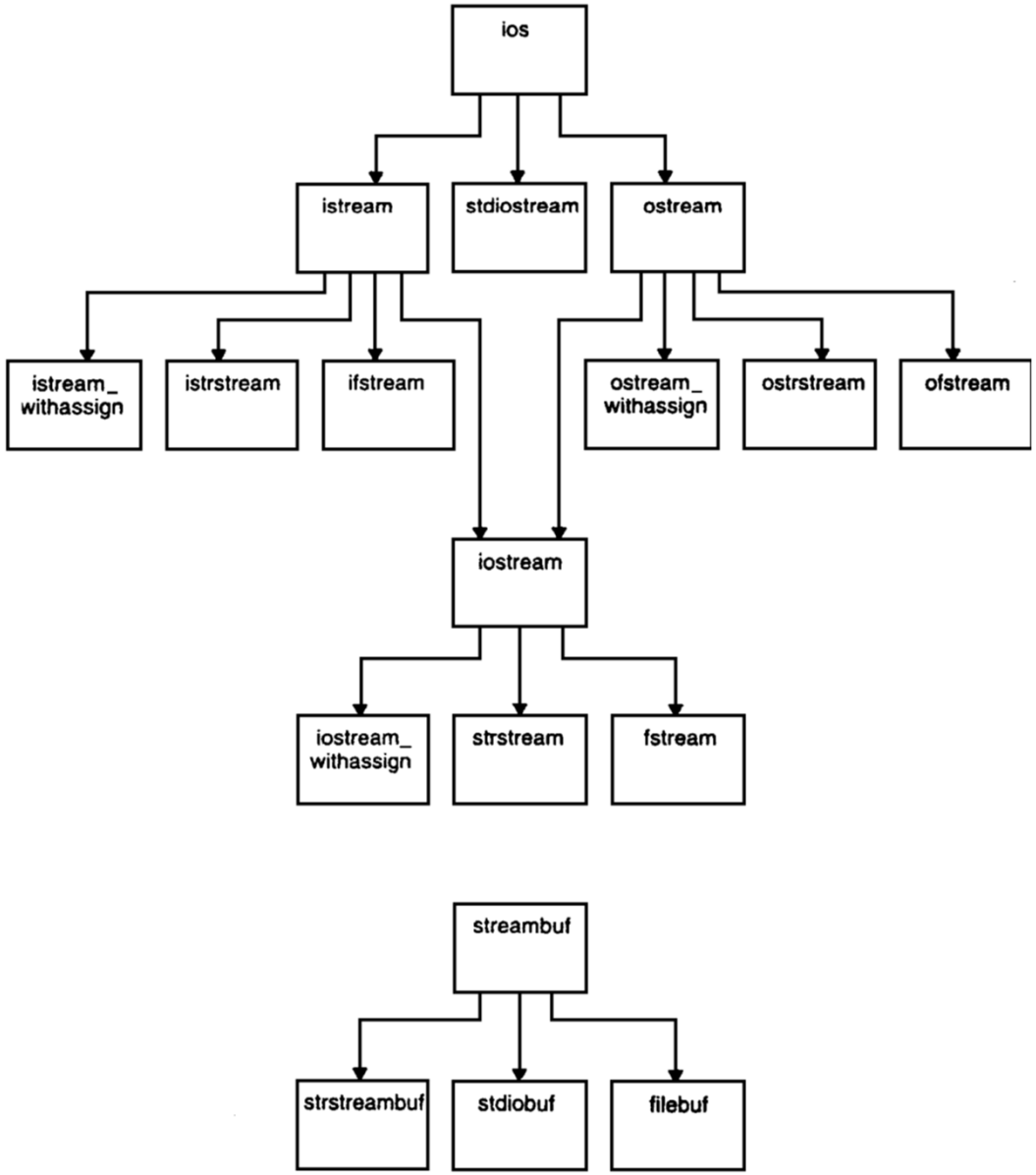
Librairie I/O stream

La libraire I/O stream contient deux classes de base, ios et streambuf, et plusieurs classes dérivées, (voir figure suivante).

- La classe streambuf et ses classes dérivées définissent des tampons pour les flots. Les tampons sont des zones mémoires qui servent de relais entre les variables du programme et optimisent les échanges en réduisant le nombre des appels systèmes qui sont souvent bien longs.

- La classe ios et ses classes dérivées définissent les opérations sur ces flots.

Librairie I/O stream



Opérateurs << , >>

Entrées-Sorties standards

- Pour lire ou écrire des données dans un flux, les deux opérateurs d'insertion >> et d'extraction << sont utilisés avec les deux instances prédéfinies cin et cout des classes istream (flux d'entrée) et ostream (flux de sortie).

- cin est un flux associé à l'entrée par défaut (clavier), tandis que cout est un flux associé à la sortie par défaut (l'écran).

- << est défini dans la classe ostream pour les types de base du langage.

- >> est défini dans la classe istream pour les types de base du langage.

- Il est possible d'adapter ces opérateurs à d'autres types de données.

Exemples de surcharge de << , >>

Exemple :

```
class CExemple
{
        char nom[20];
        int  valeur;
        public:
            friend ostream &operator<<(ostream &os, const CExemple &ex);
            friend istream &operator>>(ostream &is, CExemple &ex);
};

ostream &operator<<(ostream &os, const CExemple &ex)
 {
            os << ex.nom << " " << ex.valeur << endl;
            return os;
}
istream &operator>>(istream &is, CExemple &ex)
 {
             is >> ex.nom >> ex.valeur;
            return is;
}

void main()
{
            CExemple e1;
            cout << "Entrez un nom et une valeur : ";
            cin >> e1;
            cout << "Exemple = " << e1 << endl;
}
```

Format des sorties (1/2)

Etats de Formats

Pour formater les sorties par défaut, on utilise des manipulateurs simples ou on appelle directement des fonctions membres de la classe de base ios. Les manipulateurs simples s'emploient sous la forme :

flux << manipulateur ou flux >> manipulateur

(voir polycopié)

Exemple :

```
int     i = 512; double d = 123.4;
cout << oct << i << endl;              // conversion en octal, affiche 1000
cout << hex << i << endl;              // conversion en hexadécimal, affiche 200
cout << dec << i << endl;              // conversion en décimal, affiche 512
cout << scientific << d << endl;    // notation scientifique, affiche 1.234000E2
cout << fixed << d << endl;          // virgule fixe, affiche 123.4
```

Format des sorties (2/2)

Contrôle du format des entrées-sorties

Plusieurs fonctions membres de la classe ios modifient les données membres de cette classe pour contrôler les formats d'entrée et/ou de sortie des flots. (voir polycopié)

Exemple :

```
double pi = 3.14159265358979323385;
cout << pi;                                  // affiche 3.141592
cout.precision (8);
cout << pi                                   // affiche 3.14159265
cout.setf (ios::showpoint,ios::floatfield); // setf pour options exclusives
cout << 10.0;                                //affiche 10.00000000
cout.setf (ios::fixed,ios::floatfield);      // notation scientifique
cout.width (6);                              // affichage sur 6 caractères
cout.fill ('@');                             // caractère de remplissage @
cout << 12.5;                                // affiche @@12.5
cout.setf (ios::showpos);                    // affiche + pour les nombres > 0
cout.setf (ios::left, ios::adjustfield);     // justification à gauche
cout << 12.5;                                // affiche +12.5@
```

Utilisation des flux (1/2)

Manipulation de flux

Toutes les opérations d'entrée/sortie sur un flux sont réalisées dans un tampon. C'est pourquoi les classes ios, istream, ostream et iostream reçoivent un tampon comme unique paramètre pour leurs constructeurs.

Afin de réduire cette contrainte, les trois classes ofstream, ifstream, et fstream dérivées respectivement de ostream, istream et iostream, permettent de manipuler très simplement des flux en créant automatiquement un tampon associé au flux. On utilise principalement quatre fonctions membres de ces classes.

(i) [i/o]fstream ( );

(ii) [i/o]fstream (int d);

(iii) [i/o]fstream(const char*fname,int mode,int prot=filebuf::openprot);

(iv) [i/o]fstream (int d, char *p, int len);

Utilisation des flux (2/2)

Explication :

- La première version n'a pas de paramètre et construit un objet [i/o]fstream qui n'est pas ouvert.

- La seconde version reçoit un unique paramètre et construit un objet [i/o]fstream relié à un descripteur de flux d.

- La troisième version a trois arguments. Elle permet d'ouvrir un flux fname, avec le mode d'ouverture spécifié par mode et le mode de protection spécifié par prot. Le tableau suivant précise les différents modes d'ouverture possibles, qui peuvent être combinés entre eux à l'aide de l'opérateur ou (|).

- La quatrième version reçoit elle aussi trois paramètres. Le premier est un objet associé à un descripteur de flux d. Ce constructeur permet de construire un objet tampon streambuf, de taille len caractères et qui commence à l'adresse p. Si p vaut NULL ou que len est égale à zéro, l'objet associé filebuf n'est pas tamponné.

Modes d'ouverture d'un flux

Mode d'ouverture	Action
ios::in	ouverture en lecture seule (obligatoire pour la classe ifstream)
ios::out	ouverture en écriture seule (obligatoire pour ofstream)
ios::binary	ouverture en mode binaire
ios::app	ouverture en ajout de données (écriture en fin de flux)
ios::ate	déplacement en fin de flux après ouverture
ios::trunc	si le flux existe, son contenu est effacé (obligatoire si ios::out est activé sans ios::ate ni ios::app)
ios::nocreate	le flux doit exister préalablement à l'ouverture
ios::noreplace	le flux ne doit pas exister préalablement à l'ouverture (sauf si ios::ate ou ios::app est activé)

Ouverture/Fermetture de flux

- Ouverture de flux (open())

  La fonction **[i/o]fstream::open (const char* fname, int mode, int prot=filebuf::openprot) ;** permet d'ouvrir un flux fname, avec le mode d'ouverture spécifié par mode et le mode de protection spécifié par prot. Le tableau précédent précise les différents modes d'ouverture possibles, qui peuvent être combinés entre eux à l'aide de l'opérateur ou (|).

- Fermeture de flux (close())

  La fonction **fstreambase::close ( )** ferme le tampon attaché à l'objet de la classe [i/o]fstream en supprimant la connexion entre l'objet et le descripteur de flux.

- Flux en écriture

  La fonction **ostream& ostream::put (char c)** insère le caractère c dans le tampon associé à l'objet de la classe [i/o]fstream. Pour insérer plusieurs caractères simultanément, il faut utiliser la fonction **ostream& ostream::write (char* cp, int n)** qui insère n caractères stockés à l'adresse pointée par cp.

Déplacement dans un flux

Pour déplacer le pointeur d'élément dans le flux, il faut utiliser la fonction **ostream& ostream::seekp (streamoff n, ios::seek_dir dir)** qui positionne le pointeur à n octets de la position dir. dir peut prendre l'une des valeurs du tableau suivant.

Position dans le flux	**Action**
ios::beg	Début du flux
ios::cur	position courante du flux
ios::end	fin du flux

Pour connaître la valeur courante du pointeur d'élément, on utilise la fonction **streampos ostream::tellp ( )**.

Lecture dans un flux

- Flux en lecture

  La fonction **istream& istream::get (char c)** extrait un caractère du tampon associé à l'objet de la classe [i/o]fstream et le place dans c. Pour extraire plusieurs caractères simultanément, il faut utiliser la fonction **istream& istream::read (char* cp, int n)** qui extrait n caractères du tampon et les stocke à l'adresse pointée par cp.

  Pour déplacer le pointeur d'élément dans le flux, il faut utiliser la fonction **istream& istream::seekg (streamoff n, ios::seek_dir dir)** qui positionne le pointeur à n octets de la position dir (les différentes valeurs de dir sont regroupées dans le tableau précédent).

  Pour connaître la valeur courante du pointeur d'élément, on utilise la fonction **streampos istream::tellg ( )**.

Exemple : Ecriture dans un flux texte

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>
#define N   16

void main (void)
{
        int i;
        double s[N];
        char * flux_name = "data.txt";

        for (i=0;i<N;i++)
                s[i] = sin (2 * M_PI * i / (double) N);
        ofstream fr (flux_name, ios::out);
        if (!fr)
        {
                cerr << "Erreur d'ouverture de : " << flux_name << endl;
                exit (1);
        }
        for (i=0;i<N;i++)
                fr << s[i];
        fr.close ( );
}
```

Exemple : Lecture dans un flux texte

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>
#define N   16

void main (void)
{
        int i;
        double t[N];
        char * flux_name = "data.txt";

        ifstream fd (flux_name, ios::in);
        if (!fd)
        {
                cerr << "Erreur d'ouverture de : " << flux_name << endl;
                exit (1);
        }
        for (i=0;i<N;i++)
        fd >> t[i];
        fd.close ( );
}
```

Exemple : Ecriture dans un flux binaire

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>
#define N   16

void main (void)
{
        int i;
        double s[N];
        char * flux_name = "data.txt";

        for (i=0;i<N;i++)
                s[i] = sin (2 * M_PI * i / (double) N);
        ofstream fr (flux_name, ios::out);
        if (!fr)
        {
                cerr << "Erreur d'ouverture de : " << flux_name << endl;
                exit (1);
        }

        fr.write((char*)s,N*sizeof(double));
        fr.close ( );
}
```

Exemple : Lecture dans un flux binaire

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>
#define N   16

void main (void)
{
        int i;
        double t[N];
        char * flux_name = "data.txt";

        ifstream fd (flux_name, ios::in);
        if (!fd)
        {
                cerr << "Erreur d'ouverture de : " << flux_name << endl;
                exit (1);
        }

        fd.read((char*)t,N*sizeof(double));
        fd.close ( );
}
```

Etats d’erreurs (1/2)

Les états d'erreurs permettent de garder trace des erreurs qui peuvent éventuellement survenir pendant la manipulation des flux, de la classe ios et de toutes ses classes dérivées.

États d'erreur	Signification
ios::goodbit	aucun bit d'erreur n'est activé
ios::eofbit	marque fin de flux rencontrée pendant une extraction
ios::failbit	erreur d'allocation mémoire pendant l'utilisation d'un flux
ios::badbit	erreur fatale sur le tampon (streambuf) associé au flux

Etats d’erreurs (2/2)

Pour préserver l'encapsulation des données, on accède aux valeurs des différents états d'erreurs au moyen de fonctions membres publiques dont la liste est donnée dans le tableau suivant

Fonctions membres	**Rôle**
int bad ( ) const	retourne une valeur non nulle si ios::badbit est activé
void clear (int)	efface les états d'erreur spécifiés (l'opérateur de bit | peut spécifier plus d'un état) ou tous les états si le paramètre est nul
int eof ( ) const	retourne une valeur non nulle si ios::eofbit est activé
int fail ( ) const	retourne une valeur non nulle si ios::badbit ou ios::failbit sont activés
int good ( ) const	renvoie une valeur non nulle si aucun bit d'erreur est activé
streambuf* rdbuf ( )	retourne un pointeur sur l'objet streambuf associé au flux