Algorithmique
Cours 5

IUT Informatique de Lens, 1ère Année
Université d'Artois

Frédéric Koriche
koriche@cril.fr
2011 - Semestre 1

Sommaire

L'objectif de ce cours est d'étudier les **énumérations et structures** en algorithmique et programmation C++.

Sommaire

L'objectif de ce cours est d'étudier les **énumérations et structures** en algorithmique et programmation C++.

Rappel des types simples

Un type simple est un domaine de valeurs, qui peuvent être des booléens, des caractères, des nombres entiers ou des nombres réels.

Nom	C++	Octets	Domaine
booléen	`bool`	1	0 et 1
caractère	`char`	1	de -128 à 127 (codage ASCII)
entier	`short`	2	de -2^{15} à $2^{15}-1$
	`int`	4	de -2^{31} à $2^{31}-1$
	`long long`	8	de -2^{63} à $2^{63}-1$
réel	`float`	4	$\pm 3.410^{\pm 38}$ (7 décimales)
	`double`	8	$\pm 1.710^{\pm 308}$ (15 décimales)

Rappel des types simples

Un type simple est un domaine de valeurs, qui peuvent être des booléens, des caractères, des nombres entiers ou des nombres réels.

Nom	C++	Octets	Domaine
booléen	`bool`	1	0 et 1
caractère	`char`	1	de -128 à 127 (codage ASCII)
entier	`short`	2	de -2^{15} à $2^{15}-1$
	`int`	4	de -2^{31} à $2^{31}-1$
	`long long`	8	de -2^{63} à $2^{63}-1$
réel	`float`	4	$\pm 3.410^{\pm 38}$ (7 décimales)
	`double`	8	$\pm 1.710^{\pm 308}$ (15 décimales)

Note

En C++, il est possible de construire des types non signés (nombres positifs) en utilisant le mot clé `unsigned` devant le type de données (ex : `unsigned int`).

Conversions

Une **conversion de type** est une transformation permettant d'effectuer des opérations sur des données de types différents.

- Conversion implicite : le compilateur réalise automatiquement de la conversion
- Conversion explicite : le programmeur doit intervenir pour réaliser la conversion

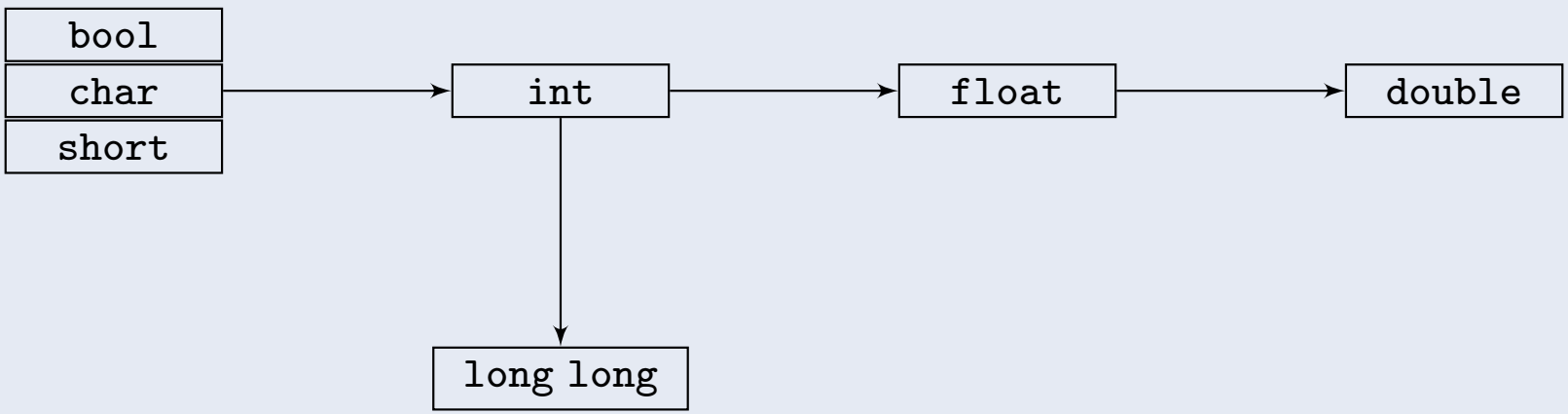
Conversions

Une **conversion de type** est une transformation permettant d'effectuer des opérations sur des données de types différents.

- Conversion implicite : le compilateur réalise automatiquement de la conversion
- Conversion explicite : le programmeur doit intervenir pour réaliser la conversion

Conversions implicites

Les types simples sont convertis implicitement en obéissant à la règle de **promotion** :



Conversions

Une **conversion de type** est une transformation permettant d'effectuer des opérations sur des données de types différents.

- Conversion implicite : le compilateur réalise automatiquement de la conversion
- Conversion explicite : le programmeur doit intervenir pour réaliser la conversion

Exemple : quelques conversions implicites

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


int main()
{
bool b = true;
int n = 1000;
float f = 1.0/3.0;

f = n;

n = b;
}
```

Initialisation d'un booléen

Initialisation d'un entier

Initialisation d'un flottant

Code C++

Conversions

Une **conversion de type** est une transformation permettant d'effectuer des opérations sur des données de types différents.

- Conversion implicite : le compilateur réalise automatiquement de la conversion
- Conversion explicite : le programmeur doit intervenir pour réaliser la conversion

Exemple : quelques conversions implicites

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


int main()
{
bool b = true;
int n = 1000;
float f = 1.0/3.0;

f = n;

n = b;
}
```

Code C++

Initialisation d'un booléen

Initialisation d'un entier

Initialisation d'un flottant

Conversion implicite d'un entier en flottant ; `f` prend la valeur 1000

Conversion implicite d'un booléen en entier ; `n` prend la valeur 1

Conversions

Une **conversion de type** est une transformation permettant d'effectuer des opérations sur des données de types différents.

- Conversion implicite : le compilateur réalise automatiquement de la conversion
- Conversion explicite : le programmeur doit intervenir pour réaliser la conversion

Conversions explicites

La conversion d'un type en un autre peut être réalisée explicitement à l'aide d'un opérateur de **cast** ; le type est explicité entre parenthèses.

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


int main()
{
int a = 4;
int b = 5;
float f;

f = a / b;

f = (float)a / b;
}
```

Initialisation de deux entiers

Initialisation d'un flottant

Code C++

Conversions

Une **conversion de type** est une transformation permettant d'effectuer des opérations sur des données de types différents.

- Conversion implicite : le compilateur réalise automatiquement de la conversion
- Conversion explicite : le programmeur doit intervenir pour réaliser la conversion

Conversions explicites

La conversion d'un type en un autre peut être réalisée explicitement à l'aide d'un opérateur de **cast** ; le type est explicité entre parenthèses.

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


int main()
{
int a = 4;
int b = 5;
float f;

f = a / b;

f = (float)a / b;
}
```

Code C++

Initialisation de deux entiers

Initialisation d'un flottant

A cause de la division entière, la valeur de `f` est 0

Par conversion explicite du numérateur en flottant, la valeur de `f` est 0.8

Enumérations

Une **énumération** (ou type énuméré) est un type dont le domaine de valeurs est défini par le programmeur.

Pseudo-code

Pseudo-code

énumération Nom { valeur_1, valeur_2,$\cdots$, valeur_n }

variables
  ...

début
  ...
fin

Enumérations

Une **énumération** (ou type énuméré) est un type dont le domaine de valeurs est défini par le programmeur.

Pseudo-code

Pseudo-code

énumération Nom { valeur_1, valeur_2,$\cdots$, valeur_n }

variables
  | ...

début
  | ...
fin

Une énumération est un type et figure donc avant la déclaration de variables

Enumérations

Une **énumération** (ou type énuméré) est un type dont le domaine de valeurs est défini par le programmeur.

Code C++

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


enum Nom {valeur_1, valeur_2,···, valeur_n};


int main()
{
...
}
```

Code C++

Enumérations

Une **énumération** (ou type énuméré) est un type dont le domaine de valeurs est défini par le programmeur.

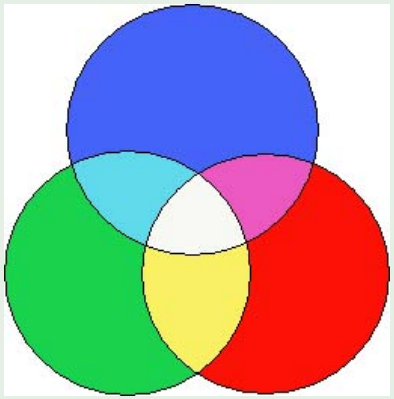
Code C++

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


enum Nom {valeur_1, valeur_2,···, valeur_n};


int main()
{
...
}
```

Code C++

Ne pas oublier le point virgule !

Exemple : couleurs



Pseudo-code

```
// Déclaration du type Couleur
```

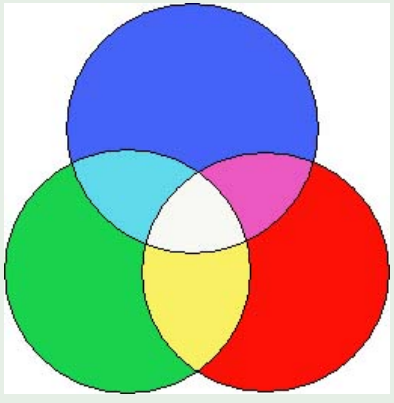
énumération Couleur {bleu, vert, rouge, jaune, cyan, magenta, blanc}

Exemple : couleurs

Couleurs en pseudo-code

```
// Déclaration du type Couleur
énumération Couleur {bleu, vert, rouge, jaune, cyan, magenta, blanc}

variables
    // Déclaration de deux variables de type Couleur
    Couleur x, y


début
    // Affectation de x à bleu
    x ← bleu

    // Si x est bleu alors y devient rouge
    si x = bleu alors
        y ← rouge
    fin
fin
```

Exemple : couleurs



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

// Déclaration du type Couleur
enum Couleur
    {
    bleu,
    vert,
    rouge,
    jaune,
    cyan,
    magenta,
    blanc
    };
```

Code C++

Exemple : couleurs

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

// Déclaration du type Couleur
enum Couleur {bleu, vert, rouge, jaune, cyan, magenta, blanc};

int main()
{
// Déclaration de deux variables de type Couleur
Couleur x,y;

// Affectation de x à bleu
x = bleu;

// Si x est bleu alors y devient rouge
if(x == bleu)
   y = rouge;
}
```

Code C++

Exemple : jours de la semaine

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

// Déclaration du type Jour
enum Jour
    {
    lundi,
    mardi,
    mercredi,
    jeudi,
    vendredi,
    samedi,
    dimanche
    };

int main()
{
// Déclaration d'une variable de type Jour
Jour x;

// Affectation de x à mardi
x = mardi;
}
```

Code C++

Stockage des énumérations

En C+, les valeurs des énumérations sont traitées comme des entiers, numérotés de gauche à droite en démarrant à 0.

Exemple : jours de la semaine

```
enum Jour
    {
    lundi,          // 0
    mardi,          // 1
    mercredi,       // 2
    jeudi,          // 3
    vendredi,       // 4
    samedi,         // 5
    dimanche        // 6
    };
```

Code C++

Opérateurs de comparaison

Les comparaisons sur les énumérations s'effectuent par conversion implicite `enum` en `int`

Nom	Code C++	Exemple
égal	`==`	`jour == mardi`
différent	`!=`	`jour != mardi`
plus petit	`<`	`jour1 < jour2`
plus grand	`>`	`jour > mercredi`

Opérateurs de comparaison

Les comparaisons sur les énumérations s'effectuent par conversion implicite `enum` en `int`

Nom	Code C++	Exemple
égal	`==`	`jour == mardi`
différent	`!=`	`jour != mardi`
plus petit	`<`	`jour1 < jour2`
plus grand	`>`	`jour > mercredi`

Opérateurs arithmétiques

La conversion implicite `int` en `enum` n'étant pas autorisée, les manipulations arithmétiques doivent se faire par conversion explicite

```
Jour j = lundi;
int x;
x = j;
x++;
j = (Jour) x;
```

Code C++

Opérateurs de comparaison

Les comparaisons sur les énumérations s'effectuent par conversion implicite `enum` en `int`

Nom	Code C++	Exemple
égal	`==`	`jour == mardi`
différent	`!=`	`jour != mardi`
plus petit	`<`	`jour1 < jour2`
plus grand	`>`	`jour > mercredi`

Opérateurs arithmétiques

La conversion implicite `int` en `enum` n'étant pas autorisée, les manipulations arithmétiques doivent se faire par conversion explicite

```
Jour j = lundi;
int x;
x = j;
x++;
j = (Jour) x;
```

Code C++

Conversion implicite, la variable `x` prend la valeur 0

Conversion explicite, la variable `j` prend la valeur `mardi`

Structures

Une **structure** est un objet composite formé par plusieurs types groupés ensembles

Déclaration de type

Pseudo-code

structure Nom
  Type_1 nom_1
  Type_2 nom_2
  ...
  Type_k nom_k

variables
  ...

début
  ...
fin

Structures

Une **structure** est un objet composite formé par plusieurs types groupés ensembles

Déclaration de type

Pseudo-code

structure Nom
| Type_1 nom_1
| Type_2 nom_2
| ...
| Type_k nom_k

variables
| ...

début
| ...
fin

Une structure est un type et figure donc avant la déclaration de variables

Structures

Une **structure** est un objet composite formé par plusieurs types groupés ensembles

Déclaration de type

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

struct Nom
    {
    type_1 nom_1;
    type_2 nom_2;
    ...
    type_k nom_k;
    };

int main()
{
...
}
```

Code C++

Structures

Une **structure** est un objet composite formé par plusieurs types groupés ensembles

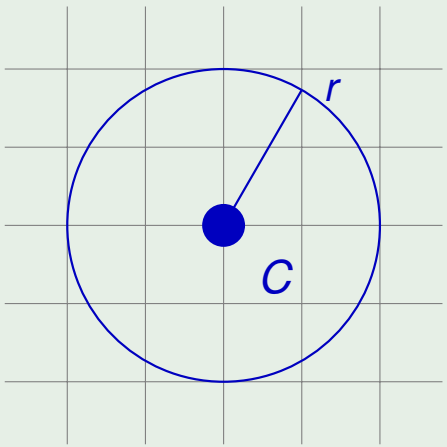
Déclaration de type

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

struct Nom
    {
    type_1 nom_1;
    type_2 nom_2;
    ...
    type_k nom_k;
    };

int main()
{
...
}
```

Code C++

Ne pas oublier le point virgule !

Exemple : Cercles



Pseudo-code

structure Cercle
- **réel** absCentre
- **réel** ordCentre
- **réel** rayon

Exemple : Cercles

Pseudo-code

```
structure Cercle
    réel absCentre
    réel ordCentre
    réel rayon

variables
    // Déclaration d'une variable de type cercle
    Cercle monCercle

début
    // Initialisation du centre
    monCercle.absCentre ← 0
    monCercle.ordCentre ← 0

    // Initialisation du rayon
    monCercle.rayon ← 1
fin
```

Exemple : Cercles

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

// Déclaration du type Cercle

struct Cercle
    {
    float absCentre;
    float ordCentre;
    float rayon;
    };

int main()
{
// Déclaration d'une variable Cercle
Cercle monCercle;

// Initialisation des valeurs de monCercle
monCercle.absCentre = 0;
monCercle.ordCentre = 0;
monCercle.rayon = 1;
}
```

Code C++

Cartes à Jouer (jeu de 32)

Pseudo-code

```
// Couleur de la carte
```

énumération Couleur {trêfle, carreau, pique, coeur}

```
// Face de la carte
```

énumération Face {sept, huit, neuf, dix, valet, dame, roi, as}

```
// Déclaration du type Carte
```

structure Carte

> Couleur couleur
> Face face

Cartes à Jouer (jeu de 32)


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

// Couleur de la carte
enum Couleur {trefle, carreau, pique,
coeur};

// Face de la carte
enum Face {sept, huit, neuf, dix,
valet, dame, roi, as};

// Déclaration du type Carte
struct Carte
    {
    Couleur couleur;
    Face face;
    };
```

Code C++

Cartes à jouer (jeu de 32)

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

// Couleur de la carte
enum Couleur {trefle, carreau, pique, coeur};

// Face de la carte
enum Face {sept, huit, neuf, dix, valet, dame, roi, as};

// Déclaration du type Carte
struct Carte {Couleur couleur; Face face;};

int main()
{
// Initialisation de deux cartes
Carte carte1 = {pique,as};
Carte carte2 = {coeur,valet};

// Comparaison des faces
if(carte1.face >= carte2.face)
   cout << "Le joueur 1 a gagne!" << endl;

}
```

Code C++

Films

Pseudo-code

```
// Genre du film
énumération Genre {action, animation, comédie, drame, sci-fi}

// Déclaration du type Film
structure Film
    caractère titre[50]
    caractère metteurEnScène[50]
    entier durée
    Genre genre
```

Films

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

// Genre du film
enum Genre {action, animation,
comedie, drame, scifi, thriller};

// Déclaration du type Film
struct Film
    {
    char titre[50];
    char metteurEnScene[50];
    short duree;
    Genre genre;
    };
```

Code C++

Films

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

enum Genre {action, animation, comedie, drame, scifi, thriller};

struct Film
    {
    char titre[50];
    char metteurEnScene[50];
    short duree;
    Genre genre;
    };

int main()
{
// Déclaration d'un film
Film monFilm;
monFilm.titre = "Memento";
monFilm.metteurEnScene = "Christopher Nolan";
monFilm.duree = 113;
monFilm.genre = thriller;
}
```

Code C++

Films

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

enum Genre {action, animation, comedie, drame, scifi, thriller};

struct Film
    {
    char titre[50];
    char metteurEnScene[50];
    short duree;
    Genre genre;
    };

int main()
{
// Autre forme de déclaration
Film film1 = {"Memento", "Christopher Nolan", 113, thriller};
Film film2 = {"Démineurs", "Kathryn Bigelow", 131, action};
}
```

Code C++

Films

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

enum Genre {action, animation, comedie, drame, scifi, thriller};

struct Film
    {
    char titre[50];
    char metteurEnScene[50];
    short duree;
    Genre genre;
    };

int main()
{
// Déclaration d'une bibliothèque de 50 films
Film bibliothèque[50];
}
```

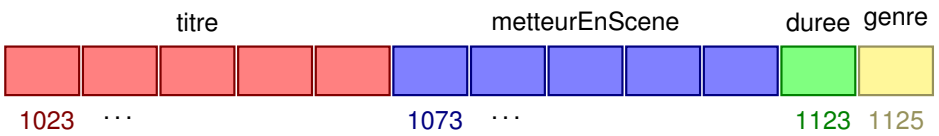
Code C++

Stockage des structures

Une structure est représentée en mémoire comme un tableau de données, chacune correspondant à un élément de la structure.

```
struct Film
    {
    char titre[50];
    char metteurEnScene[50];
    short duree;
    Genre genre;
    };
```

Une structure en C++



Stockage de la structure en mémoire

Opérateurs sur les structures

Comme les structures ne peuvent pas être converties en types simples, la seule opération autorisée est l'affectation (ex : initialisation).

Opérateurs sur les structures

Comme les structures ne peuvent pas être converties en types simples, la seule opération autorisée est l'affectation (ex : initialisation).

Programmation structurée

La plupart des données informatiques peuvent être représentées par des structures, des tableaux, ou des tableaux de structures (ex : bases de données).

Opérateurs sur les structures

Comme les structures ne peuvent pas être converties en types simples, la seule opération autorisée est l'affectation (ex : initialisation).

Programmation structurée

La plupart des données informatiques peuvent être représentées par des structures, des tableaux, ou des tableaux de structures (ex : bases de données).

Programmation orientée objets

Les classes, étudiées l'année prochaine, sont des structures associées avec des opérateurs (ex : comparer deux films) et des fonctions (ex : retourner le metteur en scène du film).