# Notes sur le langage C++

Pascal Viot

September 11, 2016

# Avant-propos

- Ces notes sont une introduction sur le langage C++ et toujours orientées vers le calcul scientifique.

# Avant-propos

- Ces notes sont une introduction sur le langage C++ et toujours orientées vers le calcul scientifique.
- Historiquement developpé pour être un langage qui soit un surensemble du langage C, le C++ a acquis son autonomie et a introduit de nombreux concepts dont la notion d'objet est la plus connue.

# Avant-propos

- Ces notes sont une introduction sur le langage C++ et toujours orientées vers le calcul scientifique.
- Historiquement developpé pour être un langage qui soit un surensemble du langage C, le C++ a acquis son autonomie et a introduit de nombreux concepts dont la notion d'objet est la plus connue.
- Son interfaçage avec le langage C permet d'utiliser la quasi totalité des bibliothèques disponibles en C , quand celles-ci n'ont pas été réécrites directement dans un C++ natif.

# Avant-propos

- Ces notes sont une introduction sur le langage C++ et toujours orientées vers le calcul scientifique.
- Historiquement developpé pour être un langage qui soit un surensemble du langage C, le C++ a acquis son autonomie et a introduit de nombreux concepts dont la notion d'objet est la plus connue.
- Son interfaçage avec le langage C permet d'utiliser la quasi totalité des bibliothèques disponibles en C , quand celles-ci n'ont pas été réécrites directement dans un C++ natif.
- Le gros avantage du C++ est la possibilité de pouvoir écrire des programmes très sécurisés.

# Avant-propos

- Ces notes sont une introduction sur le langage C++ et toujours orientées vers le calcul scientifique.
- Historiquement developpé pour être un langage qui soit un surensemble du langage C, le C++ a acquis son autonomie et a introduit de nombreux concepts dont la notion d'objet est la plus connue.
- Son interfaçage avec le langage C permet d'utiliser la quasi totalité des bibliothèques disponibles en C , quand celles-ci n'ont pas été réécrites directement dans un C++ natif.
- Le gros avantage du C++ est la possibilité de pouvoir écrire des programmes très sécurisés.
- Le C++ est un langage qui évolue de manière très importante. Après la révision historique de 1998, il y a eu celle de 2011 ,plus récemment celle de 2014 et la prochaine est prévue en 2017.

# Avant-propos

- Ces notes sont une introduction sur le langage C++ et toujours orientées vers le calcul scientifique.
- Historiquement developpé pour être un langage qui soit un surensemble du langage C, le C++ a acquis son autonomie et a introduit de nombreux concepts dont la notion d'objet est la plus connue.
- Son interfaçage avec le langage C permet d'utiliser la quasi totalité des bibliothèques disponibles en C , quand celles-ci n'ont pas été réécrites directement dans un C++ natif.
- Le gros avantage du C++ est la possibilité de pouvoir écrire des programmes très sécurisés.
- Le C++ est un langage qui évolue de manière très importante. Après la révision historique de 1998, il y a eu celle de 2011 ,plus récemment celle de 2014 et la prochaine est prévue en 2017.
- Parallèlement, le compilateur Gnu évolue rapidement pour suivre ces nouveautés. La version 4.8 intégre la version 11 et la dernière version 6.2 intègre toutes les nouveautes de la version 14.

# Plan

- La structure de base

# Plan

- La structure de base
- Les déclarations de variables

# Plan

- La structure de base
- Les déclarations de variables
  - Types prédéfinis, Tableaux, Structure

# Plan

- La structure de base
- Les déclarations de variables
  - Types prédéfinis, Tableaux, Structure
- Les classes

# Plan

- La structure de base
- Les déclarations de variables
  - Types prédéfinis, Tableaux, Structure
- Les classes
  - Définition, surcharge de constructeur,

# Plan

- La structure de base
- Les déclarations de variables
  - Types prédéfinis, Tableaux, Structure
- Les classes
  - Définition, surcharge de constructeur,
  - Objet dynamique, notion d'héritage

# Plan

- La structure de base
- Les déclarations de variables
  - Types prédéfinis, Tableaux, Structure
- Les classes
  - Définition, surcharge de constructeur,
  - Objet dynamique, notion d'héritage
- Instructions itératives et conditionnelles

# Plan

- La structure de base
- Les déclarations de variables
  - Types prédéfinis, Tableaux, Structure
- Les classes
  - Définition, surcharge de constructeur,
  - Objet dynamique, notion d'héritage
- Instructions itératives et conditionnelles
- Opérateurs de comparaison et d'assignation

# Plan

- La structure de base
- Les déclarations de variables
  - Types prédéfinis, Tableaux, Structure
- Les classes
  - Définition, surcharge de constructeur,
  - Objet dynamique, notion d'héritage
- Instructions itératives et conditionnelles
- Opérateurs de comparaison et d'assignation
- Surcharge de fonctions

# Plan

- La structure de base
- Les déclarations de variables
  - Types prédéfinis, Tableaux, Structure
- Les classes
  - Définition, surcharge de constructeur,
  - Objet dynamique, notion d'héritage
- Instructions itératives et conditionnelles
- Opérateurs de comparaison et d'assignation
- Surcharge de fonctions
- Fichiers d'en-tête

# Plan

- La structure de base
- Les déclarations de variables
  - Types prédéfinis, Tableaux, Structure
- Les classes
  - Définition, surcharge de constructeur,
  - Objet dynamique, notion d'héritage
- Instructions itératives et conditionnelles
- Opérateurs de comparaison et d'assignation
- Surcharge de fonctions
- Fichiers d'en-tête
- Entrées, sorties, formats

# Plan

- La structure de base
- Les déclarations de variables
  - Types prédéfinis, Tableaux, Structure
- Les classes
  - Définition, surcharge de constructeur,
  - Objet dynamique, notion d'héritage
- Instructions itératives et conditionnelles
- Opérateurs de comparaison et d'assignation
- Surcharge de fonctions
- Fichiers d'en-tête
- Entrées, sorties, formats
- Allocation dynamique de mémoire, pointeurs

# Plan

- La structure de base
- Les déclarations de variables
  - Types prédéfinis, Tableaux, Structure
- Les classes
  - Définition, surcharge de constructeur,
  - Objet dynamique, notion d'héritage
- Instructions itératives et conditionnelles
- Opérateurs de comparaison et d'assignation
- Surcharge de fonctions
- Fichiers d'en-tête
- Entrées, sorties, formats
- Allocation dynamique de mémoire, pointeurs
- Passages de variables, tableaux et fonctions

# Plan

- La structure de base
- Les déclarations de variables
  - Types prédéfinis, Tableaux, Structure
- Les classes
  - Définition, surcharge de constructeur,
  - Objet dynamique, notion d'héritage
- Instructions itératives et conditionnelles
- Opérateurs de comparaison et d'assignation
- Surcharge de fonctions
- Fichiers d'en-tête
- Entrées, sorties, formats
- Allocation dynamique de mémoire, pointeurs
- Passages de variables, tableaux et fonctions
- STL, C++11 et C++14

# Plan

- La structure de base
- Les déclarations de variables
  - Types prédéfinis, Tableaux, Structure
- Les classes
  - Définition, surcharge de constructeur,
  - Objet dynamique, notion d'héritage
- Instructions itératives et conditionnelles
- Opérateurs de comparaison et d'assignation
- Surcharge de fonctions
- Fichiers d'en-tête
- Entrées, sorties, formats
- Allocation dynamique de mémoire, pointeurs
- Passages de variables, tableaux et fonctions
- STL, C++11 et C++14
- Conclusion et références

# La structure de base

- Le découpage d'un programme en classes" reste le premier travail à réaliser, c'est-à-dire associer les données et l'ensemble des méthodes s'y rapportant. Cette réflexion doit se faire en fonction de la nature du langage C++ orienté objet..

# La structure de base

- Le découpage d'un programme en classes" reste le premier travail à réaliser, c'est-à-dire associer les données et l'ensemble des méthodes s'y rapportant. Cette réflexion doit se faire en fonction de la nature du langage C++ orienté objet..
- Le langage C++ possède de quasiment toutes les vertus pour satisfaire les objectifs précédents. Son efficacité et son écriture restent les limitations principales de son utilisation. La version C++ 11 est la dernière version définie en août 2011 mais mise en place dans les compilateurs gcc dans la version 5 et la version 15 de icc.

# La structure de base

- Le découpage d'un programme en classes" reste le premier travail à réaliser, c'est-à-dire associer les données et l'ensemble des méthodes s'y rapportant. Cette réflexion doit se faire en fonction de la nature du langage C++ orienté objet..
- Le langage C++ possède de quasiment toutes les vertus pour satisfaire les objectifs précédents. Son efficacité et son écriture restent les limitations principales de son utilisation. La version C++ 11 est la dernière version définie en août 2011 mais mise en place dans les compilateurs gcc dans la version 5 et la version 15 de icc.
- Historiquement, une bonne partie de la syntaxe est héritée du langage C, mais le langage évolue rapidement

# La structure de base

- Le découpage d'un programme en classes" reste le premier travail à réaliser, c'est-à-dire associer les données et l'ensemble des méthodes s'y rapportant. Cette réflexion doit se faire en fonction de la nature du langage C++ orienté objet..
- Le langage C++ possède de quasiment toutes les vertus pour satisfaire les objectifs précédents. Son efficacité et son écriture restent les limitations principales de son utilisation. La version C++ 11 est la dernière version définie en août 2011 mais mise en place dans les compilateurs gcc dans la version 5 et la version 15 de icc.
- Historiquement, une bonne partie de la syntaxe est héritée du langage C, mais le langage évolue rapidement
- A noter que la syntaxe du C est acceptée par la quasi-totalité des compilateurs. Les fonctionnalités du C++ ne sont disponibles qu'en utilisant les concepts et la syntaxe spécifique de ce langage.

# Les déclarations de variables

- Types prédéfinis Ces types sont identiques à ceux du langage C.
- les types dérivés sont aussi identiques au langage. struct.
- Les tableaux
- L'initialisation d'un tableau peut être faite dans la déclaration, par exemple

```
double a[3]={0,1,2};
double b[10]={1,1,1};
// les 7 autres éléments sont mis à 0
double c[]={10,7,4};
```

- Notons que le commentaire en C++ commence par un double slash et se poursuit jusqu'à la fin de la ligne
- il est possible de définir des types plus élaborés et aussi de définir des types à l'aide de l'instruction typedef.

# Les déclarations de variables(2)

- On peut aussi définir des types nouveaux à travers des structures. Par exemple

```
typedef struct {
long i;
double x;
double y;
} num_pos_points;
```

- Ainsi défini dans l'en-tête du programme, on peut utiliser ce type comme un type habituel. Soit la déclaration de la structure cc

```
num_pos_points cc;
```

- La syntaxe pour l'affectation des membres de la structure sera la suivante

```
cc.i=5;
cc.x=0.6;
cc.y=0.7;
```

# Les déclarations de variables(3)

- Pour un tableau de structure, on aura la syntaxe suivante

```
num_pos_points cc[10];
```

pour la déclaration

```
 for(int i=0;i<10;i++)
{
cc[i].i=5*i;
cc[i].x=0.6*i;
cc[i].y=0.7*i;
}
```

# les classes

- La notion de classe est donc la généralisation de la notion de structure vue dans le langage C et aussi en fortran 90.
- La forme générale est la suivante:

```
class nom_classe {
  permission_etiquette_1:
    membre1;
  permission_etiquette_2:
    membre2;
  ...
  } ;
```

- Le nom_classe est le nom de la classe.
- Les permissions sont de trois types public, protected ou private.
- Les membres sont les fonctions ou les données de la classe.
- Les membres private accessibles que par les autres membres de la classe.

# les classes (2)

- Les membres protected accessibles que par les membres de la classe et des classes friend ou dérivées.

# les classes (2)

- Les membres protected accessibles que par les membres de la classe et des classes friend ou dérivées.
- Les membres public sont accessibles de partout où la classe est visible.

# les classes (2)

- Les membres protected accessibles que par les membres de la classe et des classes friend ou dérivées.
- Les membres public sont accessibles de partout où la classe est visible.
- Il est à noter que ces différents attributs sont assez délicats à manipuler et sont à l'origine de longs démélés avec le compilateur quand on est débutant

# les classes (2)

- Les membres protected accessibles que par les membres de la classe et des classes friend ou dérivées.
- Les membres public sont accessibles de partout où la classe est visible.
- Il est à noter que ces différents attributs sont assez délicats à manipuler et sont à l'origine de longs démélés avec le compilateur quand on est débutant
- On peut substituer le nom de class par struct.

# les classes (2)

- Les membres protected accessibles que par les membres de la classe et des classes friend ou dérivées.
- Les membres public sont accessibles de partout où la classe est visible.
- Il est à noter que ces différents attributs sont assez délicats à manipuler et sont à l'origine de longs démélés avec le compilateur quand on est débutant
- On peut substituer le nom de class par struct.
- Dans ce cas, la permission par défaut est public, et donc parfois laxiste.

# les classes (2)

- Les membres protected accessibles que par les membres de la classe et des classes friend ou dérivées.
- Les membres public sont accessibles de partout où la classe est visible.
- Il est à noter que ces différents attributs sont assez délicats à manipuler et sont à l'origine de longs démélés avec le compilateur quand on est débutant
- On peut substituer le nom de class par struct.
- Dans ce cas, la permission par défaut est public, et donc parfois laxiste.
- Si toutes les permissions sont spécifiées, il n'y a pas de différence entre struct et class.

# les classes (2)

- Les membres protected accessibles que par les membres de la classe et des classes friend ou dérivées.
- Les membres public sont accessibles de partout où la classe est visible.
- Il est à noter que ces différents attributs sont assez délicats à manipuler et sont à l'origine de longs démélés avec le compilateur quand on est débutant
- On peut substituer le nom de class par struct.
- Dans ce cas, la permission par défaut est public, et donc parfois laxiste.
- Si toutes les permissions sont spécifiées, il n'y a pas de différence entre struct et class.
- Les habitués du langage C trouveront avec cette syntaxe une généralisation du concept de structure vu en C.

# Les classes(3)

- Le programme suivant définit une classe Rectangle constituée de deux nombres réels (largeur et longueur) et les fonctions membres sont une fonction d'initialisation et une fonction qui calcule l'aire du rectangle

```
#include <iostream>
using namespace std;
class Rectangle {
    int x, y;
  public:
    void initial (int,int);
    int aire (void) const {return (x*y);}
};

void Rectangle::initial (int a, int b) {
  x = a;
  y = b;
}

int main () {
  Rectangle rect;
  rect.initial (3,4);
  cout << "aire: " << rect.aire()<<"\n";
  return 0;
}
```

# Les classes(4)

- Ce premier programme comprenant une seule classe illustre deux points:

# Les classes(4)

- Ce premier programme comprenant une seule classe illustre deux points:
  - le fait que pour un objet créé à partir d'une classe, l'appel aux fonctions membres, appelées aussi méthodes, se fait de manière analogue à l'écriture des membres d'une structure.

# Les classes(4)

- Ce premier programme comprenant une seule classe illustre deux points:
  - le fait que pour un objet créé à partir d'une classe, l'appel aux fonctions membres, appelées aussi méthodes, se fait de manière analogue à l'écriture des membres d'une structure.
  - Dans l'appel d'un objet, il est généralement nécessaire d'initialiser celui-ci avant d'appliquer des méthodes, sinon le risque de sortie du programme est grand.

# Les classes(4)

- Ce premier programme comprenant une seule classe illustre deux points:
  - le fait que pour un objet créé à partir d'une classe, l'appel aux fonctions membres, appelées aussi méthodes, se fait de manière analogue à l'écriture des membres d'une structure.
  - Dans l'appel d'un objet, il est généralement nécessaire d'initialiser celui-ci avant d'appliquer des méthodes, sinon le risque de sortie du programme est grand.
- Pour permettre une initialisation simple, le C++ fournit une méthode standard qui s'appelle le constructeur de la classe. Le constructeur est appelé lors de la création de l'objet qui est initialisé en même temps.

# Les classes(4)

- Ce premier programme comprenant une seule classe illustre deux points:
  - le fait que pour un objet créé à partir d'une classe, l'appel aux fonctions membres, appelées aussi méthodes, se fait de manière analogue à l'écriture des membres d'une structure.
  - Dans l'appel d'un objet, il est généralement nécessaire d'initialiser celui-ci avant d'appliquer des méthodes, sinon le risque de sortie du programme est grand.
- Pour permettre une initialisation simple, le C++ fournit une méthode standard qui s'appelle le constructeur de la classe. Le constructeur est appelé lors de la création de l'objet qui est initialisé en même temps.
- La fonction cout permet de réaliser une sortie à l'écran.

# Les classes(4)

- Ce premier programme comprenant une seule classe illustre deux points:
  - le fait que pour un objet créé à partir d'une classe, l'appel aux fonctions membres, appelées aussi méthodes, se fait de manière analogue à l'écriture des membres d'une structure.
  - Dans l'appel d'un objet, il est généralement nécessaire d'initialiser celui-ci avant d'appliquer des méthodes, sinon le risque de sortie du programme est grand.
- Pour permettre une initialisation simple, le C++ fournit une méthode standard qui s'appelle le constructeur de la classe. Le constructeur est appelé lors de la création de l'objet qui est initialisé en même temps.
- La fonction cout permet de réaliser une sortie à l'écran.
- Les méthodes peuvent être définies à l'intérieur de la classe comme la méthode aire,

# Les classes(4)

- Ce premier programme comprenant une seule classe illustre deux points:
  - le fait que pour un objet créé à partir d'une classe, l'appel aux fonctions membres, appelées aussi méthodes, se fait de manière analogue à l'écriture des membres d'une structure.
  - Dans l'appel d'un objet, il est généralement nécessaire d'initialiser celui-ci avant d'appliquer des méthodes, sinon le risque de sortie du programme est grand.
- Pour permettre une initialisation simple, le C++ fournit une méthode standard qui s'appelle le constructeur de la classe. Le constructeur est appelé lors de la création de l'objet qui est initialisé en même temps.
- La fonction cout permet de réaliser une sortie à l'écran.
- Les méthodes peuvent être définies à l'intérieur de la classe comme la méthode aire,
- ou à l'extérieur de la classe comme le constructeur Rectangle, à condition que celui-ci soit précédé du nom de la classe (en l'occurence, ici le nom de la classe est aussi Rectangle) et de deux ":" consécutifs.

# Les classes(5)

- Surcharge de constructeur On peut créer plusieurs constructeurs pour une classe:

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

class Rectangle {
    int x, y;
  public:
    Rectangle();
    Rectangle(int,int);
    int aire (void) {return (x*y);}
};
Rectangle::Rectangle (int a, int b) {
  x = a;y = b;
}
Rectangle::Rectangle () {
  x = 0;y = 0;
}
int main () {
  Rectangle rect(3,4);
  Rectangle recto;
  cout << "aire: " << rect.aire()<<"\n";
  cout << "aire2: " << recto.aire()<<"\n";
  return 0;
}
```

# Les classes(6)

- On peut simplifier cette écriture en ne déclarant pas le constructeur par défaut que le C++ construit toujours.
- En définissant un constructeur qui par défaut initialise à zéro les paramètres et en utilisant la copie de classe, on peut réécrire le programme suivant

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

class Rectangle {
    int x, y;
  public:
  Rectangle (int a=0, int b=0): x(a),y(b){  };
 // ~Rectangle(){};

     int aire (void) const {return (x*y);}
};

int main () {
  Rectangle rect0;
  cout << "aire0: " << rect0.aire()<<"\n";
  Rectangle rect1(3,4);
  cout << "aire1: " << rect1.aire()<<"\n";
  Rectangle rect2=rect1;
  cout << "aire2: " << rect2.aire()<<"\n";
  return 0;
}
```

# Les classes(7)

- Le premier objet rect0 sans argument donne une aire égale à zéro, le second donne une aire égale à 12 et le troisième qui est une copie de rect1 donne aussi la meme valeur que ce dernier.

# Les classes(7)

- Le premier objet rect0 sans argument donne une aire égale à zéro, le second donne une aire égale à 12 et le troisième qui est une copie de rect1 donne aussi la meme valeur que ce dernier.
- Objet dynamique Par souci de symétrie (et surtout parce qu'un objet peut être créé dynamiquement, voir ci-dessous) le langage C++ fournit une méthode simple qui s'appelle le destructeur de la classe.

# Les classes(7)

- Le premier objet rect0 sans argument donne une aire égale à zéro, le second donne une aire égale à 12 et le troisième qui est une copie de rect1 donne aussi la meme valeur que ce dernier.
- Objet dynamique Par souci de symétrie (et surtout parce qu'un objet peut être créé dynamiquement, voir ci-dessous) le langage C++ fournit une méthode simple qui s'appelle le destructeur de la classe.
- Il est défini par un tilde suivi du nom de classe. Par défaut un constructeur est crée par le langage C++, ce qui explique que celui-ci est donc placé en commentaires ici car le destructeur est un destructeur vide. Si l'on souhaite modifier celui-ci il faut décommenter la ligne et remplir la partie entre accolades d'instructions. Il ne peut exister qu'un seul destructeur par classe.

# Les classes(7)

- Le premier objet rect0 sans argument donne une aire égale à zéro, le second donne une aire égale à 12 et le troisième qui est une copie de rect1 donne aussi la meme valeur que ce dernier.
- Objet dynamique Par souci de symétrie (et surtout parce qu'un objet peut être créé dynamiquement, voir ci-dessous) le langage C++ fournit une méthode simple qui s'appelle le destructeur de la classe.
- Il est défini par un tilde suivi du nom de classe. Par défaut un constructeur est crée par le langage C++, ce qui explique que celui-ci est donc placé en commentaires ici car le destructeur est un destructeur vide. Si l'on souhaite modifier celui-ci il faut décommenter la ligne et remplir la partie entre accolades d'instructions. Il ne peut exister qu'un seul destructeur par classe.
- De manière similaire à la création dynamique d'une structure, on peut créer dynamiquement un objet d'une classe donnée. Le programme suivant illustre ce concept.

# Les classes(8)

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

class Rectangle {
    int x, y;
  public:
  Rectangle (int a=0, int b=0): x(a),y(b){  };
 // ~Rectangle(){};

     int aire (void) const {return (x*y);}
};

int main () {
  Rectangle *b;
  b =new Rectangle(2,3);
  cout << "aire: " << b->aire()<<"\n";
  delete b;
  b =new Rectangle;
  cout << "aire2: " << b->aire()<<"\n";
  return 0;
}
```

- On voit que le premier objet a été à la fois créé avec l'opérateur new et initialisé avec les valeurs 2 et 3, tandis que le second a été crée et initialisé avec les valeurs par défaut de la classe.

# Notion d'héritage

- Une des propriétés les plus intéressantes du C++ est la notion d'héritage: elle permet de construire une classe à partir d'une classe déjà existante

# Notion d'héritage

- Une des propriétés les plus intéressantes du C++ est la notion d'héritage: elle permet de construire une classe à partir d'une classe déjà existante
- en ajoutant soit des fonctions membres soit des données, soit les deux.

# Notion d'héritage

- Une des propriétés les plus intéressantes du C++ est la notion d'héritage: elle permet de construire une classe à partir d'une classe déjà existante
- en ajoutant soit des fonctions membres soit des données, soit les deux.
- Nous allons créer une seconde classe qui permet de définir un parallépipède rectangle et dont on peut calculer le volume

# Notion d'héritage

- Une des propriétés les plus intéressantes du C++ est la notion d'héritage: elle permet de construire une classe à partir d'une classe déjà existante
- en ajoutant soit des fonctions membres soit des données, soit les deux.
- Nous allons créer une seconde classe qui permet de définir un parallépipède rectangle et dont on peut calculer le volume
- La classe Paralle_de est construite à partir de la classe Rectangle en introduisant une troisième donnée pour définir la hauteur après avoir défini le rectangle avec une largeur et une longueur.

# Notion d'héritage

- Une des propriétés les plus intéressantes du C++ est la notion d'héritage: elle permet de construire une classe à partir d'une classe déjà existante
- en ajoutant soit des fonctions membres soit des données, soit les deux.
- Nous allons créer une seconde classe qui permet de définir un parallépipède rectangle et dont on peut calculer le volume
- La classe Paralle_de est construite à partir de la classe Rectangle en introduisant une troisième donnée pour définir la hauteur après avoir défini le rectangle avec une largeur et une longueur.
- Le constructeur utilise une initialisation par défaut et la copie de classe en combinant celle de rectangle et celle de l'argument z.

# Notion d'héritage

- Une des propriétés les plus intéressantes du C++ est la notion d'héritage: elle permet de construire une classe à partir d'une classe déjà existante
- en ajoutant soit des fonctions membres soit des données, soit les deux.
- Nous allons créer une seconde classe qui permet de définir un parallépipède rectangle et dont on peut calculer le volume
- La classe Paralle_de est construite à partir de la classe Rectangle en introduisant une troisième donnée pour définir la hauteur après avoir défini le rectangle avec une largeur et une longueur.
- Le constructeur utilise une initialisation par défaut et la copie de classe en combinant celle de rectangle et celle de l'argument z.
- La fonction membre de la classe Paralle_de, volume, est construite à partir de la fonction membre de la classe Rectangle, aire.

# Notion d'héritage

- Une des propriétés les plus intéressantes du C++ est la notion d'héritage: elle permet de construire une classe à partir d'une classe déjà existante
- en ajoutant soit des fonctions membres soit des données, soit les deux.
- Nous allons créer une seconde classe qui permet de définir un parallépipède rectangle et dont on peut calculer le volume
- La classe Paralle_de est construite à partir de la classe Rectangle en introduisant une troisième donnée pour définir la hauteur après avoir défini le rectangle avec une largeur et une longueur.
- Le constructeur utilise une initialisation par défaut et la copie de classe en combinant celle de rectangle et celle de l'argument z.
- La fonction membre de la classe Paralle_de, volume, est construite à partir de la fonction membre de la classe Rectangle, aire.
- La fonction Paralle_de hérite de la fonction membre de Rectangle.

# Notion d'héritage(2)

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

class Rectangle {
  int x, y;
public:
  Rectangle (int a=0 ,int b=0): x(a), y(b) {};
   int aire (void) {return (x*y);}
};

class Paralle_de : public Rectangle
{
  int z;
public:
  Paralle_de(int a=0, int b=0, int c=0): Rectangle(a,b), z(c) {};
  int volume (void)  {return (aire()*z);}
};

int main () {
  Paralle_de Para;
  cout <<"volume "<<Para.volume()<<endl;
  Paralle_de Para2(3,4,5);
  cout <<"volume "<<Para2.volume()<<endl;
  Paralle_de Para3=Para2;
   cout <<"volume "<<Para3.volume()<<endl;
  return 0;
}
```

# Notion d'héritage(3)

- Dans le programme principal, le premier objet donne un volume car ses paramètres sont nuls par défaut.

# Notion d'héritage(3)

- Dans le programme principal, le premier objet donne un volume car ses paramètres sont nuls par défaut.
- Les objets suivants donnent la valeur 60 pour le volume produit des trois entiers. Le troisième élément est simplement la copie de l'objet numéro 2.

# Notion d'héritage(3)

- Dans le programme principal, le premier objet donne un volume car ses paramètres sont nuls par défaut.
- Les objets suivants donnent la valeur 60 pour le volume produit des trois entiers. Le troisième élément est simplement la copie de l'objet numéro 2.
- Pour être complet notons que le programme principal peut utiliser des objets créés dynamiquement.

# Notion d'héritage(3)

- Dans le programme principal, le premier objet donne un volume car ses paramètres sont nuls par défaut.
- Les objets suivants donnent la valeur 60 pour le volume produit des trois entiers. Le troisième élément est simplement la copie de l'objet numéro 2.
- Pour être complet notons que le programme principal peut utiliser des objets créés dynamiquement.
- La syntaxe est illustrée dans le programme où seul le programme principal est modifié, les autres lignes du programme n'étant pas modifiées, seul le programme principal est donné.

# Les instructions itératives

- Pour les boucles, on dispose des mêmes fonctionalités que celle du langage C; la seule différence est la possibilité de pouvoir déclarer l'indice de boucle à l'intérieur de la boucle, ce qui assure que la portée de cette variable reste à l'intérieur de cette boucle. Ainsi on peut écrire pour la boucle for

```
unsigned int m,n,step
for (unsigned int i=m;i<n;i+=step){
...
}
```

où step correspond au pas d'incrémentation. Si $n$ est strictement inférieur à $m$, aucune instruction de la boucle ne sera exécutée.

- Pour une valeur de step égale à 1, on peut écrire plus simplement

```
for (unsigned int i=m;i<n;i++){
...
}
```

# Les instructions itératives(2)

- Pour mémoire, on a les boucles while et do-while avec la syntaxe suivante:

```
 while(condition) {
...
}
```

- les instructions de la boucle ne sont exécutées que si la condition est vérifiée.

```
do{
...
}
 while(condition);
```

- Les instructions sont exécutées une première fois et la condition est alors testée.

# if, case, switch, break

- Pas de changement par rapport au langage C.

# Les fichiers d'en-tête

- Le langage C++ comme le langage C nécessite d'inclure des fichiers d'en-tête pour que le compilateur (préprocesseur pour être précis) puisse savoir le type de classes à utiliser. Voici la liste des fichiers d'en-tête que l'on peut utiliser en C++.
- `<algorithm> <bitset> <deque> <exception> <fstream> <functional><iomanip> <ios> <iosfwd> , <iostream> <istream> <iterator> <limits> <list> <locale> <map> <memory> <new> <numeric> <ostream> <queue> <set> <sstream> <stack> <stdexcept> <streambuf> <string> <typeinfo> <utility> <valarray> <vector>`
- parmi les fichiers d'en-têtes correspondant au langage C, on retrouve cmath.

  Pour les fichiers d'en-tête correspondant à des bibliothèques appartenant à l'ordinateur, la syntaxe est la suivante. Par exemple

  ```
  #include <iostream>
  ```

# Les fichiers d'en-tête(2)

- <cassert> (ANSI C++) == <assert.h> (ANSI C)
  <cctype> (ANSI C++) == <ctype.h> (ANSI C)
  <cerrno> (ANSI C++) == <cerrno.h> (ANSI C)
  <cfloat> (ANSI C++) == <float.h> (ANSI C)
  <ciso646> (ANSI C++) == <ciso646.h> (ANSI C)
  <climits> (ANSI C++) == <limits.h> (ANSI C)
  <clocale> (ANSI C++) == <locale.h> (ANSI C)
  <cmath> (ANSI C++) == <math.h> (ANSI C)
  <csetjmp> (ANSI C++) == <setjmp.h> (ANSI C)
  <csignal> (ANSI C++) == <signal.h> (ANSI C)
  <cstdarg> (ANSI C++) == <stdarg.h> (ANSI C)
  <cstddef> (ANSI C++) == <stddef.h> (ANSI C)
  <cstdio> (ANSI C++) == <stdio.h> (ANSI C)
  <cstdlib> (ANSI C++) == <stdlib.h> (ANSI C)
  <cstring> (ANSI C++) == <string.h> (ANSI C)
  <ctime> (ANSI C++) == <time.h> (ANSI C)
  <cwchar> (ANSI C++) == <wchar.h> (ANSI C)
  <cwtype> (ANSI C++) == <wtype.h> (ANSI C)

# Les fichiers d'en-tête(3)

- Pour permettre au compilateur de vérifier votre programme, il est recommandé de placer les définitions de classes soit dans un fichier d'en-tête personnel, soit en début du programme.
- On appelle cela le prototypage
- Si le fichier d'en-tête se trouve dans le répertoire courant, la syntaxe est

```
#include "monfichier.h"
```

# Les fichiers d'en-tête(4)

- iostream. Ce fichier définit les classes de gestion des entrées et sorties. Les procédures et fonctions de classes sont appelées des fonctions membres. Comme on peut le voir sur l'exemple ci-dessous

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

int main(){
  unsigned int n=100;
  double x[n];
  double sum=0;
  {
    for (unsigned int i=1;i<n;++i){
      x[i]=cos( i *M_PI*0.01);
      sum+=x[i];
    }
  }
  cout<<"somme "<<sum<<endl;

  return 0;
}
```

De plus, on voit qu'il est nécessaire d'avoir la ligne

```
using namespace std;
```

# Les fichiers d'en-tête(5)

- L'instruction namespace indique que les fonctions membres cin et cout à utiliser sont ceux de la classe std.
- fstream correspond à la définition des classe pour la gestion des entrées-sorties sur disque dur.
- cmath Le fichier définit à la fois des constantes mathématiques usuelles ($\pi$,$e$) et les fonctions mathématiques.

# Les entrées et les sorties

- On voit qu'une partie des fichiers d'en-tête du langage C peuvent être utilisées dans le langage C++.
- Pour les paresseux (ou les conservateurs) en incluant le fichier cstdio, on peut utiliser les fonctions printf et scanf du langage C sans changer ses habitudes.
- Pour lire des données formatées sur l'entréee standard (clavier), on utilise la fonction membre cin.

```
double a;
cin>>a;
```

# Les entrées et les sorties(2)

- Pour écrire les données formatées sur la sortie standard (écran), on utilise la fonction membre cout

```
double a;
cout<<a;
```

- Il existe aussi la sortie erreur qui permet d'afficher des messages à l'écran quand la sortie standard a été redirigée. L'instruction est alors cerr.
- Cette sortie est généralement utilisée pour envoyer des messages signalant un comportement anormal du programme.

```
int n;
if (n<0) cerr<<n;
```

# Les entrées et les sorties(3)

- Pour lire les données formatées sur un fichier, on fait appel à la classe des entrées sorties définies dans fstream.
- Dans l'exemple qui suit, on définit un objet fichier initialisé par le fichier exemple.txt, la méthode is_open vérifie que la création du fichier s'est bien passée. La méthode close ferme le fichier.

```
#include <fstream>
using namespace std;
int main () {
  double a=2.0004;
  ofstream fichier ("exemple.txt");
  if (fichier.is_open())
  {
    fichier << "Début d'écriture\n'';
    fichier << "a= "<<a<<endl;
    fichier << "Déjà fini!endl;
    fichier.close();
  }
  return 0;
}
```

# Les entrées et les sorties(4)

- Pour lire sur un fichier, il faut préalablement ouvrir ce fichier donc définir un descripteur de fichier, ce qui explique la déclaration du programme précédent

```
ofstream fichier ("exemple.txt");
```

- Une fois l'ensemble des ordres d'écriture effectués, il faut penser à fermer le fichier, ce qui est fait avec la fonction membre close.

```
fichier.close();
```

# Surcharge de fonctions

- Une fonctionnalité très intéressante dans le langage C++ est de pouvoir définir une fonction plusieurs fois avec le même nom mais avec des arguments de types différents.

# Surcharge de fonctions

- Une fonctionnalité très intéressante dans le langage C++ est de pouvoir définir une fonction plusieurs fois avec le même nom mais avec des arguments de types différents.
- Seuls les arguments de la fonction peuvent être différents.

# Surcharge de fonctions

- Une fonctionnalité très intéressante dans le langage C++ est de pouvoir définir une fonction plusieurs fois avec le même nom mais avec des arguments de types différents.
- Seuls les arguments de la fonction peuvent être différents.
- Le type de la fonction doit rester identique.

# Surcharge de fonctions

- Une fonctionnalité très intéressante dans le langage C++ est de pouvoir définir une fonction plusieurs fois avec le même nom mais avec des arguments de types différents.
- Seuls les arguments de la fonction peuvent être différents.
- Le type de la fonction doit rester identique.
- Dans l'exemple qui suit, le compilateur choisit la fonction affiche selon les arguments passés. On vérifie que l'affichage des huit premiers élémentsd du tableau $x$ donne bien ceux de l'initialisation pour les sept premiers élement et le huitième a été initialisé à 0.

# Surcharge de fonctions(2)

```
#include <iostream>
using namespace std;
void affiche(const int n);
void affiche(const float n);
void affiche(const double  *x,const unsigned  n);
int main(){
  double x[10]={1e0,0.7,0.6,1.0,4.5,8,0.9};
  int a=67;
  float c=0.8;
  affiche(x,8);
  affiche(a);
  affiche(c);
}

void affiche(const int n)
{
  cout<<n<<endl;
}
void affiche(const double *x,const unsigned n)
{
  for(unsigned i=0;i<n;++i){
    cout<<i<<' '<<x[i]<<endl;
  }
}

void affiche(const float n)
{
  cout<<n<<endl;
}
```

# Alloc. dynamique, pointeurs

- Il est bien entendu possible en C++ de faire de l'allocation dynamique de mémoire comme nous l'avons vu dans la section sur les classes. Nous allons voir plus spécifiquement ici le cas des tableaux.
- L'allocation dynamique de mémoire se fait de la manière suivante: on déclare un pointeur avec le type correspondant à la variable, au tableau ou à la structure que l'on désire créer.

```
double * a=NULL;
```

- En initialisant le pointeur à l'adresse NULL, on évite de récupérer une adresse incorrecte et de créer des bugs certains. Dans un second temps, on utilise la fonction **new** en précisant le type du tableau que l'on souhaite créer. La syntaxe est la suivante:

```
a= new double [10];
```

# Alloc. dynamique, pointeurs(2)

- Si la machine ne dispose pas de la mémoire demandée en cours d'exécution du programme, l'allocation ne se fait pas.
- Une manière simple de le savoir est de vérifier une fois l'allocation dynamique effectuée que le pointeur *a* possède la valeur d'une adresse différente de la valeur assignée au départ.
- l'instruction suivante placée après l'instruction **new**, conduit à une interruption du programme si l'allocation dynamique n'a pas fonctionnée, ce qui évite un arrêt dans l'exécution à un endroit du programme où l'on fait appel à un tableau qui n'a pas pu être créé.

```
if (a ==  NULL)
  {cerr<<'ça ne marche pas \n";
   return(EXIT_FAILURE);}
```

- Une fois créés, les éléments du tableau sont accessibles à partir d'une syntaxe tout à fait identique à celle des tableaux statiques.
- Avant de quitter le programme il est nécessaire de libérer la mémoire qui a été demandée par l'utilisateur. L'instruction correspondante est alors

# Alloc. dynamique, pointeurs(3)

- En C++, si la mémoire est déjà libérée, le programme ne se plantera pas car la fonction delete vérifie si le pointeur est libre ou non.

# Alloc. dynamique, pointeurs(3)

- En C++, si la mémoire est déjà libérée, le programme ne se plantera pas car la fonction delete vérifie si le pointeur est libre ou non.
- Ainsi la séquence à respecter est la suivante,

# Alloc. dynamique, pointeurs(3)

- En C++, si la mémoire est déjà libérée, le programme ne se plantera pas car la fonction delete vérifie si le pointeur est libre ou non.
- Ainsi la séquence à respecter est la suivante,
  - Déclaration d'un pointeur (avec son type)

# Alloc. dynamique, pointeurs(3)

- En C++, si la mémoire est déjà libérée, le programme ne se plantera pas car la fonction delete vérifie si le pointeur est libre ou non.
- Ainsi la séquence à respecter est la suivante,
  - Déclaration d'un pointeur (avec son type)
  - Appel de la fonction new

# Alloc. dynamique, pointeurs(3)

- En C++, si la mémoire est déjà libérée, le programme ne se plantera pas car la fonction delete vérifie si le pointeur est libre ou non.
- Ainsi la séquence à respecter est la suivante,
  - Déclaration d'un pointeur (avec son type)
  - Appel de la fonction new
  - Vérification que la mémoire est bien allouée

# Alloc. dynamique, pointeurs(3)

- En C++, si la mémoire est déjà libérée, le programme ne se plantera pas car la fonction delete vérifie si le pointeur est libre ou non.
- Ainsi la séquence à respecter est la suivante,
  - Déclaration d'un pointeur (avec son type)
  - Appel de la fonction new
  - Vérification que la mémoire est bien allouée
  - Libération de la mémoire avec la fonction delete dès que l'on n'a plus besoin des données.

# Opérateurs surchargés

- Pour un tableau créé dynamiquement, la syntaxe du coeur du programme reste identique. Le mini-programme ci-dessous en est l'illustration.

```
#include<iostream>
using namespace std;
typedef struct {
  int i;
  double x;
  double y;
} num_pos_points;

ostream & operator << (ostream & io, const num_pos_points & p)
{io<<p.i<<" "<<p.x<<" "<<p.y;return io;};
int main(){
  num_pos_points *cc=NULL;

  cc= new  num_pos_points [10];
  if (cc ==  NULL)
    {cerr<<" l'allocation dynamique n'a pas fonctionnée \n";
  return(EXIT_FAILURE);}
  for(unsigned int i=0;i<10;i++)
    {
      cc[i].i=i;
      cc[i].x=0.6*i;
      cc[i].y=0.7*i;
      cout<<cc[i].i<<" "<<cc[i].x<<" "<<cc[i].y<<"\n";
      cout<<cc[i]<<endl;
    }

delete [] (cc);
 return 0;}
```

# Opérateurs surchargés(2)

- A noter que la fonction $<<$ a été redéfinie pour la structure num_pos_points afin que l'écriture de la structure reste simple. Le prix à payer est la définition d'une surcharge d'opérateurs. avec une syntaxe de déclaration un peu délicate!

# Variables, des tableaux et des fonctions

- Une référence est définie à partir de l'opérateur &.

```
int a;
int &b=a;
// En d'autres termes, on peut dire que b est un alias de a. On a
//deux noms pour la meme case mémoire
a=3;
b++;
```

a vaut maintenant 4.

# Variables, des tableaux et des fonctions(2)

- On peut écrire en C++ le passage par référence pour l'argument de la fonction cela donne

```
/* déclaration du prototype de la fonction */
#include <iostream>
using namespace std;

void sqr_v3 ( double &);

int main()
{double a2,a=2.0;

  cout<<"a="<<a<<endl;
  sqr_v3(a);
  cout<<"a="<<a<<endl;
  exit(0);
}
void  sqr_v3 ( double &a)
{a *=a;
return ;}
```

- L'avantage réside dans l'absence de la recopie de la variable.

# Variables, des tableaux et des fonctions(3)

- Pour les tableaux, le fait de recopier l'ensemble des valeurs d'un tableau peut être à la fois très pénalisant en terme de temps de calcul; ainsi on a coutume d'utiliser les tableaux situés dans la région mémoire du programme appelant.
- En exécutant l'exemple ci-dessous, on peut vérifier que le tableau du programme principal a été modifié par la fonction inittab. Voici la version C++ du programme donné dans le chapitre sur le langage C. La version reste procédurale, mais le langage C++ le permet

# Variables, des tableaux et des fonctions(4)

```
#include <iostream>
#include <cmath>
using namespace std;
void inittab(const double *,int );
int main(){
  double x[10];
  inittab(x,10);
  for(unsigned int i=0;i<10;i++)
    { cout <<i<<" "<<x[i]<<"\n";
    }
  return 0;
}

void inittab(const double x[],int n)
// autre déclaration possible void inittab(const double *x,int n)
{

  for (unsigned int i=0;i<n;i++){
    x[i]=cos((double) i *M_PI/(double) n);
  }
}
```

# La spécification de type: const

- Si la fonction ne doit pas modifier les valeurs du tableau transmis dans les arguments de la fonction, une disposition provenant du C++ a été introduite en C pour sécuriser l'écriture des programmes.

# La spécification de type: const

- Si la fonction ne doit pas modifier les valeurs du tableau transmis dans les arguments de la fonction, une disposition provenant du C++ a été introduite en C pour sécuriser l'écriture des programmes.
- En placant la spécification de type const devant le type du tableau on oblige le compilateur à vérifier que le tableau ne sera pas modifié dans le sous-programme. Ainsi si on modifie le programme précédent, en ajoutant la fonction affichage on obtient:

# STL, C+11 et C++14

- La bibliothèque permet de sortir du cadre classique de la programmation d'un langage de bas niveau. En définissant un tableau à partir de la STL, on définit un objet donc un ensemble de fonctions membres accessibles. Les bornes sont définies à partir des fonctions membres.

```
#include<iostream>
#include<vector>
using namespace std;
 main(){
vector<int> a;
vector<int>::iterator cii;
// Add some elements to myIntVector
a.push_back(1);
a.push_back(4);
a.push_back(8);

for(cii = a.begin(); cii != a.end(); cii++)
{cout<<*cii<<" ";}
}
```

- On a une boucle qui s'adapte en fonction du nombre d'éléments du tableau. On a utilisé un itérateur à la place d'un indice de boucle.

# STL, C+11 et C++14 (2)

- On peut aller plus loin pour une écriture plus compacte avec les spécificités du C++11 On a alors l'écriture suivante

```
#include<iostream>
#include<vector>
using namespace std;
 main(){
vector<int> a;
// Add some elements to myIntVector
a.push_back(1);
a.push_back(4);
a.push_back(8);
for(int& cii :a )
{  cout<<cii<<" ";}
}
```

- On a une boucle qui s'adapte aussi en fonction du nombre d'éléments du tableau. On a utilisé une réference sur le tableau a qui sert d'indice de boucle. Cela donne une écriture élégante et compacte. A tester au niveau de l'efficacité pour le temps d'execution.

# STL, C+11 et C++14 (3)

- Pour créer des tableaux multidimensionnels, on peut utiliser le “template” vector de la manière suivante

```
#include <iostream>
#include <vector>
using namespace std;
main()
        {
vector< vector<int> > Matrice(3, vector<int>(2,0));
   Matrice[0][0] = 0;
   Matrice[0][1] = 1;
   Matrice[1][0] = 10;
   Matrice[1][1] = 11;
   Matrice[2][0] = 20;
   Matrice[2][1] = 21;
           cout << "boucle par indice :" << endl;
           for(int ii=0; ii < 3; ii++)
           {
              for(int jj=0; jj < 2; jj++) cout << Matrice[ii][jj]<< "\t";
              cout <<  endl;
           }
        }
```

# Conclusion et références

- Passer de la programmation procédurale à la programmation orientée objet est fascinant, mais nécessite un apprentissage assez long. Il est possible pour des habitués du langage C de faire cela en douceur.

# Conclusion et références

- Passer de la programmation procédurale à la programmation orientée objet est fascinant, mais nécessite un apprentissage assez long. Il est possible pour des habitués du langage C de faire cela en douceur.
- Comme nous l'avons vu dans ce chapitre une grande partie de la syntaxe et des concepts sont identiques. Nous avons délibérément choisi de présenter le C++ comme un surensemble du C mais il peut être utilisé avec ses caractéristiques propres, c'est-à-dire sa spécificité de langage objet.

# Conclusion et références

- Passer de la programmation procédurale à la programmation orientée objet est fascinant, mais nécessite un apprentissage assez long. Il est possible pour des habitués du langage C de faire cela en douceur.
- Comme nous l'avons vu dans ce chapitre une grande partie de la syntaxe et des concepts sont identiques. Nous avons délibérément choisi de présenter le C++ comme un surensemble du C mais il peut être utilisé avec ses caractéristiques propres, c'est-à-dire sa spécificité de langage objet.
- Pour avoir une documentation en ligne sur le C++, on peut consulter par exemples les sites,
  http://www.cplusplus.com/doc/tutorial/