Analyse et Programmation Orientées Objets / C++

Les classes

Classes et objets (1)

➡ **Définition d'une classe**

- un ensemble de données membres (**les attributs**)
- un ensemble de fonctions membres (**les méthodes**)
- les attributs sont privés
- les méthodes sont publiques
- nouvelle possibilité de décrire des fonctions privées

Classes et objets (2)

➡ **Langage C++**

- Toutes les classes doivent être déclarées
- Tous les attributs doivent être déclarés
- Toutes les classes possèdent un constructeur par défaut
- Toutes les méthodes doivent être déclarées
- Toutes les fonctions privées doivent être déclarées
- Toutes les méthodes doivent être définies
- Toutes les fonctions privées doivent être définies

Déclaration d'une classe C++

➡ **Fichier *Xxxxx.h***

```
class Xxxxx {
private :
---                    // Les attributs
---                    // Prototypes des fonctions privées
public :
---                    // Les prototypes méthodes
};
```

Portée et accessibilité

➡ **Synthèse des contraintes**

- Toutes les contraintes du langage C sont conservées
  - Portée d ’une variable limitée au bloc déclarant
- Les variables globales sont obsolètes
- Nouvelles contraintes nées de l’encapsulation

Déclaration des attributs

➡ **Exemple en langage C++**

```
class Point {
private :
double m_X;
double m_Y;
double m_Z;

public :

---
};
```

Déclaration des méthodes

➡ **Exemple en langage C++**

```
class Point {
---
---
public :
Point ();
Point (double, double, double);
double distance (Point, Point );
---
};
```

Classes et objets (3)

➡ **Description d'un objet**

- Instance d'une classe via un constructeur
- Valeur des attributs spécifiques à chaque objet
- Méthodes communes à toutes les instances
- Allocation sur la pile ou dans le data segment (**static**)
- Allocation dynamique (mots clés **new** et **delete**)

Déclaration des objets

➡ **Exemple en langage C++**

```
# include  "Point.h "
---
void main ( ) {
---
Point p1(1.0, 1.55, 3.7);
Point* pP2= new Point (3, 12.5, 4);


---
}
```

Invocation des méthodes

➡ **Langage C++**

```
# include  "Point.h "

void main ( ) {
Point p1(1.0, 1.55, 3.7), p2(-4, 7, 1.5);
double d;
---
      d= p1.distance (p2);
---
}
```

L'opérateur d'application (1)

➡ **Description de la construction**

- Construction purement syntaxique
- Aucune plus value sémantique
- Génération d'un code source intermédiaire
- Passage implicite par paramètre de l'adresse de la cible
- Usage du pointeur formel implicite this

L'opérateur d'application (2)

- **Traduction en langage C**

```
# include  "Point.h "

void main ( ) {
Point p1(1.0, 1.55, 3.7), p2(-4, 7, 1.5);
double d;
---

     d= p1.distance (p2);  // <=>  d= distance (&p1, p2)
---
}
```

Définition des méthodes (1)

➡ **Exemple en langage C++**

```
# include  "Point.h "

double Point::distance (Point p2) {
double d1= m_X – p2.m_X;
double d2= m_Y – p2.m_Y;
double d3= m_Z – p2.m_Z;

        return sqrt (d1*d1 + d2*d2 + d3*d3);
}
```

:: est appelé Opérateur de Résolution de Portée (ORP)

Définition des méthodes (2)

➡ **Usage explicite du pointeur this**

```
# include  "Point.h "

double Point::distance (Point p2) {
double d1= this->m_X – p2.m_X;
double d2= this-> m_Y – p2.m_Y;
double d3= this-> m_Z – p2.m_Z;

        return sqrt (d1*d1 + d2*d2 + d3*d3);
}
```

Le pointeur **this** est appelé "pointeur sur moi-même"