Cours: Programmation C++

Chapitre 2: Les classes C++

Réalisé par:
Dr. Sakka Rouis Taoufik
https://github.com/srtaoufik/coursCpp

1

I. Introduction

Le mécanisme de classe en C++ permet aux programmeurs de définir des nouveaux types propres à leurs applications.

La classe est une extension très forte à la notion de structure de données du langage C. En effet, au sien d'une classe, nous pouvons définie des données et des méthodes. Ces entités constituent les membres de la classe.

2

II. Définition d'une classe

En C++ la syntaxe d'une classe est la suivante :

```
[struct | class | union] NomClasse {
// Modificateur d'accès pour les données membres
//Données membres
// Modificateur d'accès pour les fonctions membres
//Fonctions membres
} ;
```

Rq. Le code d'une fonction membre peut se faire dans ou en dehors de la classe. Dans le second cas, le nom de la méthode sera préfixé par le nom de la classe suivit de l'opérateur **::**

3

II. Définition d'une classe

Exemple:

```
struct  Point {
private :
  // Déclaration de 2 données membres de type réel
            float abs ;
            float ord ;
public:
  // Déclaration d'une fonction définie à l'intérieur de la classe
             void initialise (float a, float b) {
                        abs=a;
                        ord=b;
             }
 // Déclaration d'une fonction définie à l'extérieur de la classe
            void deplacer (float dx, float dy) ;
} ;
void Point::deplacer (float dx, float dy) {
            abs+=dx;
            ord+=dy;
}
```

4

III. Objet et instance d'une classe

Un objet correspond à la localisation physique d'un exemplaire (ou instance) de la classe. Chaque instance de la classe (ou objet) possède ses propres exemplaires de données (non statiques) de la classe.
Les données membres (non statiques) d'une classe seront appelées généralement les composants d'un objet.

Exemple :

```
class Point {
        static int nb ;
        int abs ;
        int ord ;
/* nb est une donnée membre statique alors que abs et ord ne sont pas statique*/…};
```

Rq. Les fonctions membres (non statiques) d'un classe, quand-à elles, sont définies pour être appliquées sur des instances de la classe. Par exemple : Point P1; … P1.affiche() ;

III. Objet et instance d'une classe

La création d'une instance sous-entend un mécanisme d'allocation mémoire. À chaque allocation mémoire correspond une durée de vie particulière.

- **Durée de vie statique** : Les objets statiques sont ceux crées par une déclaration située en dehors de toute fonction.
- **Durée de vie automatique** : ces objets sont alloués dont la pile de la mémoire avec une durée de vie correspondant à l'exécution du bloc englobant. Les objets automatiques sont ceux crées par une déclaration dans une fonction ou dans un bloc.
- **Durée de vie temporelle** : les objets temporels sont créés pour le besoin de l'évaluation d'une instruction, ils sont détruits à la fin de l'évaluation de l'expression.
- **Durée de vie explicite** : ces objets sont créés dynamiquement par une requête explicite (en utilisant malloc et new). Leurs destruction est à la charge de programmeur.

III. Objet et instance d'une classe

Exemple:

```
Point M ;
//objet global (durée statique)

void main(){
        Point A, B ; //objet automatique ;
        Point *P= new Point ; //création dans la pile explicite
        static Point C ; //objet statique
        //....
        delete p ; //destruction  explicite
}
```


IV. Protection des membres d'une classe

Il est possible d'empêcher l'accès des champs ou de certaines méthodes à toute fonction autre que celles de la classe. Cette opération s'appelle l'encapsulation. Pour la réaliser, il faut utiliser les mots clés suivants :

- *public* : les accès sont libres ;
- *private* : les accès sont autorisés dans les fonctions de la classe seulement ;
- *protected* : les accès sont autorisés dans les fonctions de la classe et de ses descendantes (voir le chapitre d'héritage) seulement.

Rq. Le mot clé *protected* n'est utilisé que dans le cadre de l'héritage des classes.

IV. Protection des membres d'une classe

Exemple :

```
struct client {
private:   // Données privées :
    char Nom [20], Prenom [20];
    unsigned int Date_Entree;
    int Solde;
    // Il n'y a pas de méthode privée.
public:    // Les données et les méthodes publiques :
    // Il n'y a pas de donnée publique.
    bool dans_le_rouge (void);
    bool bon_client (void);
};
```


IV. Protection des membres d'une classe

RQ : Par défaut, les classes construites avec ***struct*** ont tous leurs membres publics. Il est possible de déclarer une classe dont tous les éléments sont par défaut privés. Pour cela, il suffit d'utiliser le mot clé ***class*** à la place du mot clé ***struct***.

Exemple :

```
class Client{
    // private est à présent inutile.
    char Nom [21], Prenom [21];
    unsigned int Date_Entree;
    int Solde;
public:    // Les données et les méthodes publiques.
    bool dans_le_rouge(void);
    bool bon_client(void);
};
//…
Client A, B ;
int s= A.Solde ;  //erreur car Solde est privé
bool b= A.dans_le_rouge() ; //ok car cette méthode est publique
```

V. Les fonctions d'objets

Les fonctions d'objets sont les fonctions membres d'une classe qui ne sont pas déclarées statiques.

V.1 Déclaration d'une fonction membre

La déclaration d'une fonction d'objet est identique à la définition usuelle, cependant, déclarée dans le bloc de déclaration de la classe, elle est liée à cette classe particulière.

Exemple :

```
class Compte {
        int montant ;
        public :
        void init (int) ; // fonction d'objet
        int solde() ; // fonction d'objet
        void depot (int) ; // fonction d'objet
} ;
void affiche (Compte) ; //fonction usuelle
```


V. Les fonctions d'objets

La différence entre une fonction d'objet et une fonction usuelle est que la fonction membre possède un **argument implicite** qui est l'objet sur lequel elle est appliquée. La fonction sera appliquée à un objet, instance de classe.

L'utilisation de ces fonctions se fait avec la notation classique « . » ou «→» suivant que la fonction est appliquée à un objet ou à un pointeur sur un objet.

Exemple :

```
void main() {
        Compte C1 ;
        Compte *P=&C1 ;
        C1.initialise (30) ;
        P→ depot (500) ;  // C1.depot(500) ;
}
```

V. Les fonctions d'objets

V.2 Définition d'une fonction membre

La définition d'une fonction d'objet peut se faire en dehors du bloc de définition de son classe d'appartenance.
Lors de la définition d'une fonction d'objet, son nom est préfixé par le nom de la classe d'appartenance suivie de l'opérateur de résolution de portée **::**

Exemple :

```
void  Compte :: initialise (int m ) {
        montant =m ;
}

void Point :: affiche(){
        cout<< "ce point est de coordonnées "<<x<< "et " <<y <<endl ;
}
```

13

V. Les fonctions d'objets

V.3 Définition d'une fonction membre en ligne

Le mécanisme de fonctions en ligne permet de supprimer le coût de l'appel d'une fonction en substitution le code de cette fonction au niveau de l'instruction d'appel de cette fonction. De même une fonction membre pourra être définie en ligne.

Exemples :

Soit d'une façon explicite à l'aide du mot-clé inline	Soit d'une façon implicite dans le corps de la définition de la classe

class Compte {
  int montant ;
public :
  int solde ( ) ;
} ;
inline int Compte :: solde ( ) {
  return montant ;
}
	class Compte {
	  int montant ;
	public :
	  int solde ( ) {
	  return montant ;}
	} ;

14

VI. Les données et fonctions membres statiques d'une classe

Nous allons voir dans ce paragraphe l'emploi du mot clé ***static*** dans les classes. Ce mot clé intervient pour caractériser :

- les données membres statiques des classes,
- les fonctions membres statiques des classes,
- les données statiques des fonctions membres.

VI.Les données et fonctions membres statiques d'une classe

VI.1. Données membres statiques

Une donnée membre statique (variable de classe) est une variable globale de la classe. Elle n'est pas un composant d'objet.
La définition d'une variable de classe est utilisée pour conserver une information sur l'ensemble des instances de la classe.

Exemple :

```
class Point {
        //x et y sont 2 données membre privées non statique
        int x, y ;
        //nb est une donnée membre statique
        static int nb ;
} ;
int Point :: nb=0 ; //initialisation explicite
```

Rq. La variable Point :: nb sera partagée par tous les objets de classe Point, et sa valeur initiale est zéro.

VI.Les données et fonctions membres statiques d'une classe

VI.1. Données membres statiques

Donnée non statique : liée à un objet

Donnée statique : liée à la classe (fournie une information sur la classe)

Rq : La définition des données membres statiques suit les mêmes règles que la définition des variables globales. Autrement dit, elles se comportent comme des variables déclarées externes.

Les variables statiques des fonctions membres doivent être initialisées à l'intérieur des fonctions membres. Elles appartiennent également à la classe, et non pas aux objets. De plus, leur portée est réduite à celle du bloc dans lequel elles ont été déclarées.

17

Chapitre 2 : Les classes C++

VI.Les données et fonctions membres statiques d'une classe

VI.1. Données membres statiques

Exemple :

```
#include <stdio.h>
class Test {
public:
   int fn (void);
};
int Test::fn (void){
   static int compte=0;   // variable statique
   return compte++;
}
int main (void){
   Test objet1, objet2;
   printf ("%d ", objet1.fn() );   // Affiche 0
   printf ("%d \n ", objet2.fn() );  // Affiche 1
   return 0;
}
```

Affichera *0* et *1*, parce que la variable statique *compte* est la même pour les deux objets.

18

VI.Les données et fonctions membres statiques d'une classe

VI.2. Fonctions membres statiques

Les classes peuvent également contenir des fonctions membres statiques.

Cela peut surprendre à première vue, puisque les fonctions membres appartiennent déjà à la classe, c'est-à-dire à tous les objets.

En fait, cela signifie que ces fonctions membres ne recevront pas le pointeur sur l'objet *this*, comme c'est le cas pour les autres fonctions membres.

➔ elles ne pourront accéder qu'aux données statiques de l'objet.

VI.Les données et fonctions membres statiques d'une classe

VI.2. Fonctions membres statiques

Exemple :

```
class Entier {
    int i ;
    static int j ;
public:
    static int get_value (void) ;
} ;
int Entier::j=0 ;
int Entier::get_value (void) {
    j=1;        // Légal.
    return i;   // ERREUR ! get_value ne peut pas accéder à i.
}
```

La fonction get_value de l'exemple ci-dessus ne peut pas accéder à la donnée membre non statique i, parce qu'elle ne travaille sur aucun objet. Son champ d'action est uniquement la classe Entier. En revanche, elle peut modifier la variable statique j, puisque celle-ci appartient à la classe Entier et non aux objets de cette classe.

VI.Les données et fonctions membres statiques d'une classe

VI.2. Fonctions membres statiques

L'appel des fonctions membre statiques se fait exactement comme celui des fonctions membres non statiques, en spécifiant l'identificateur d'un des objets de la classe et le nom de la fonction membre, séparés par un point.

Cependant, comme les fonctions membres ne travaillent pas sur les objets des classes mais plutôt sur les classes elles-mêmes, la présence de l'objet lors de l'appel est facultatif. On peut donc se contenter d'appeler une fonction statique en qualifiant son nom du nom de la classe à laquelle elle appartient à l'aide de l'opérateur de résolution de portée **::**

VI.Les données et fonctions membres statiques d'une classe

VI.2. Fonctions membres statiques

Exemple :

```
class Entier {
    static int i;
public:
    static int get_value(void);
};
int Entier::i=3;
int Entier::get_value(void){
    return i;
}
void main(){
    // Appelle la fonction statique get_value
    int resultat;
    resultat = Entier::get_value();
}
```

VII. Fonctions et classes amies

Il est parfois nécessaire d'avoir des fonctions qui ont un accès illimité aux champs d'une classe. En général, l'emploi de telles fonctions traduit un manque d'analyse dans la hiérarchie des classes, mais pas toujours. Elles restent donc nécessaires malgré tout.

De telles fonctions sont appelées des fonctions amies. Pour qu'une fonction soit amie d'une classe, il faut qu'elle soit déclarée dans la classe avec le mot clé *friend*.

Il est également possible de faire une classe amie d'une autre classe, mais dans ce cas, cette classe devrait peut-être être une classe fille. L'utilisation des classes amies peut traduire un défaut de conception.

VII. Fonctions et classes amies

VII.1. Fonctions amies

Les fonctions amies se déclarent en faisant précéder la déclaration classique de la fonction du mot clé *friend* à l'intérieur de la déclaration de la classe cible. Les fonctions amies ne sont pas des méthodes de la classe cependant (cela n'aurait pas de sens puisque les méthodes ont déjà accès aux membres de la classe).

Exemple :

```
class Point {
   int x,y ;                  // Une donnée privée.
   friend void init (int, int); // La fonction init est une fonction amie.
};
Point A;
void init (int abs, int ord) {
            A.x=abs ; // accède à des données privées à l'objet A
            A.y=ord ;
}
```

VII. Fonctions et classes amies

Il est possible de déclarer amie une fonction d'une autre classe, en précisant son nom complet à l'aide de l'opérateur de résolution de portée.

Exemple :

```
class Point {
   int x, y ;                    // Une donnée privée.
   friend void Segment::init (int, int, init, int);   // Une fonction amie.
};
class Segment{
            Point G , D ;
            public :
            void init (int abs1, int ord1, int abs2, int ord2 ){
                        G.x=abs1 ;// accède à des données privées à l'objet A
                        G.y=ord1 ;
                        D.x=abs2 ;// accède à des données privées à l'objet B
                        D.y=ord2 ;
            }
} ;// end class B
```

➔ La méthode init de la classe Segment peut utiliser les membres privés de n'importe quel objet instance de la classe Point.

VII. Fonctions et classes amies

VII.2. Classes amies

Pour rendre toutes les méthodes d'une classe amies d'une autre classe, il suffit de déclarer la classe complète comme étant amie. Pour cela, il faut encore une fois utiliser le mot clé *friend* avant la déclaration de la classe, à l'intérieur de la classe cible.

Attention: Cette fois encore, la classe amie déclarée ne sera pas une sous-classe de la classe cible, mais bien une classe de portée globale.

VII. Fonctions et classes amies

VII.2. Classes amies

Exemple :

```
#include <iostream.h>
class A {
   friend class B;  // Toutes les méthodes de B sont amies.
   int i;              // Donnée privée de la classe A.
public:
   void A (void) {
      i=0;
     }
};
A  A1;
class B {
public:
   void print_A  ( ) {
      cout<< A1.i <<endl; // Accède à la donnée privée de A.
      }
};
void main(void){
   B b1;
   b1.print_A ( );
  }
```

27