ECOLE NORMALE SUPÉRIEURE DE L'ENSEIGNEMENT
TECHNIQUE DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA
المدرسة العليا لأساتذة التعليم التقني المحمدية

Département Mathématique et Informatique

- Cycle Ingénieur –

POO
Cours - 2 - résumé

Réaliser par :

KARROUM Houssam – BDCC 1

Contents

PROGRAMMATION ORIENTEE OBJET : NOTION DE CALSSE

1. INTRODUCTION :

La programmation structurée et la programmation orientée objet sont deux approches majeures en développement logiciel. Alors que la programmation structurée, présente dans des langages comme C et Pascal, se concentre sur la division du code en blocs logiques, la programmation orientée objet, représentée par des langages tels que C++, met l'accent sur les objets et leurs interactions. En examinant de plus près la programmation orientée objet, nous pouvons comprendre son influence sur la modularité et la gestion des données dans les logiciels modernes.

2. Les Structures en C++ :

Les structures en C++ permettent de regrouper des variables de types différents sous un seul nom. Elles sont déclarées à l'aide du mot-clé **struct**. Par exemple:

```
struct Person {
   string name;
   int age;
   double height;
};
```

Dans cet exemple, nous avons défini une structure **Person** qui contient trois membres : **name**, **age**, et **height**, chacun avec son propre type de données. Les membres d'une structure peuvent être accédés individuellement à l'aide de l'opérateur **.**

```
Person person1;
person1.name = "John";
person1.age = 30;
person1.height = 1.75;
```

3. La Création de Classes en C++ :

En termes simples, une classe en programmation est comme un modèle ou un plan pour créer des objets. Un objet est une entité qui contient à la fois des données et des actions qui peuvent être effectuées sur ces données. La classe définit la structure des données et des actions associées à ces objets. Lorsque nous créons un objet à partir d'une classe, nous appelons cela une instance de cette classe. Chaque fois qu'une instance est créée, de la mémoire est allouée pour stocker ses données, et un constructeur est utilisé pour initialiser ces données. De même, lorsque l'instance n'est plus nécessaire, un destructeur peut être utilisé pour libérer la mémoire associée à cet objet. En résumé, une classe est un moyen de définir la structure et le comportement des objets en programmation.

3.1. Déclaration d'une classe en C++

Pour déclarer une classe en C++, vous pouvez suivre les étapes suivantes :

```
class NomClasse
{
private:
    // partie accessible uniquement aux fonctions membres de la classe et aux fonctio
    // déclarations des membres de la classe

protected:
    // partie accessible aux membres de la classe ainsi qu'aux classes dérivées

public:
    // partie accessible à tout utilisateur d'une instance de la classe
    // déclarations des membres de la classe
};
```

Une classe peut avoir trois niveaux de portée : Private (par défaut), Protected ou Public. Les membres de la classe déclarés dans la section Private sont accessibles uniquement aux fonctions membres de la classe et aux fonctions amies. Les membres de la classe déclarés dans la section Protected sont accessibles aux membres de la classe ainsi qu'aux classes dérivées. Enfin,

les membres de la classe déclarés dans la section Public sont accessibles à tout utilisateur d'une instance de la classe.

Les fonctions membres d'une classe sont définies de la manière suivante :

```
TypeRetour NomClasse::NomFonction(ListeParamètres)
{
    // Déclarations
    // Actions
}
```

L'opérateur de résolution de portée :: est utilisé pour définir les fonctions membres de la classe en dehors de la déclaration de la classe.

3.2. Utilisation d'une classe en C++

a. Instanciation :

Pour créer une instance (objet) d'une classe donnée en C++, vous pouvez procéder comme suit :

```
NomClasse NomObjet; // Déclaration d'une variable objet
```

Pour créer un pointeur vers un objet, vous pouvez utiliser la syntaxe suivante :

```
NomClasse *PointeurObjet; // Déclaration d'un pointeur sur un objet
```

b. Accès aux membres d'une classe :

Pour accéder aux membres (fonctions ou variables) d'une classe pour une instance x de la classe X, vous pouvez appeler une fonction membre f de la manière suivante :

```
x.f(...);
```

Si **p** est un pointeur vers une instance de la classe **X**, vous pouvez accéder à une fonction membre **f** de la classe de la manière suivante :

```
p->f(...);
```

Dans ce cas, l'opérateur **->** est utilisé pour accéder aux membres de l'objet pointé par p.

3.3. Notion de constructeur en C++ :

Un constructeur est une fonction membre spéciale dans une classe qui porte le même nom que la classe elle-même et qui ne retourne pas de valeur (même le type void). Il est systématiquement exécuté lors de la création d'une instance (ou d'un objet) de cette classe, c'est-à-dire lors de la déclaration d'un objet.

Par exemple, considérons une classe Point. Un constructeur dans cette classe pourrait ressembler à ceci :

```
class Point {
public:
    // Constructeur
    Point() {
        // Actions du constructeur
    }
};
```

Dans cet exemple, le constructeur **Point** est appelé automatiquement chaque fois qu'une instance de la classe **Point** est créée. Vous pouvez inclure des actions spécifiques à exécuter lors de la création de l'objet à l'intérieur du constructeur.

Il est important de noter que si aucun constructeur n'est défini dans la classe, le compilateur en génère un par défaut. Cependant, si vous définissez explicitement un constructeur dans votre classe, le constructeur par défaut n'est pas généré automatiquement.

3.4. Notion de destructeur en C++ :

Le destructeur est une fonction membre spéciale d'une classe qui est systématiquement exécutée lorsqu'une instance (ou un objet) de cette classe est détruite, c'est-à-dire à la fin de sa durée de vie.

Un destructeur porte le même nom que la classe précédé d'un tilde (~), et il n'a pas de paramètres ni de valeur retournée. Par exemple, pour une classe MaClasse, le destructeur serait défini comme suit :

```
class MaClasse {
public:
    // Constructeur
    MaClasse() {
        // Actions du constructeur
    }

    // Destructeur
    ~MaClasse() {
        // Actions du destructeur
    }
};
```

Le destructeur permet de spécifier toutes les actions nécessaires à exécuter avant la destruction de l'instance, telles que l'enregistrement de données, la fermeture de fichiers, ou d'autres opérations de nettoyage. Il est important de noter que si aucun destructeur n'est défini dans la classe, le compilateur en génère un par défaut. Cependant, il est souvent nécessaire de définir un destructeur explicite lorsque des ressources dynamiques ont été allouées dans le

constructeur de la classe, afin de libérer ces ressources correctement lors de la destruction de l'objet.

3.5. Allocation dynamique :

Lorsque les membres de données d'une classe sont des pointeurs, le constructeur est souvent utilisé pour allouer dynamiquement de la mémoire sur ces pointeurs, tandis que le destructeur est utilisé pour libérer cette mémoire allouée. Cela garantit que la mémoire est correctement gérée pendant toute la durée de vie de l'objet.

Par exemple, supposons une classe MaClasse avec un membre de données qui est un pointeur :

```
class MaClasse {
private:
    int* ptr;

public:
    // Constructeur
    MaClasse() {
        ptr = new int; // Allocation dynamique de mémoire
    }

    // Destructeur
    ~MaClasse() {
        delete ptr; // Libération de la mémoire allouée
    }
};
```

Dans cet exemple, le constructeur alloue dynamiquement de la mémoire pour **ptr**, et le destructeur libère cette mémoire lors de la destruction de l'objet.

3.6. Membres statiques en C++

Lorsque deux objets différents d'une même classe sont déclarés dans un même programme, chaque objet possède ses propres membres de données. Cependant, si un membre de données est déclaré comme statique, alors tous les objets de cette classe partagent la même copie de ce membre de données.

Par exemple, considérons une classe Compteur avec un membre de données statique nbInstances qui compte le nombre d'instances de la classe créées :

```
class Compteur {
private:
    static int nbInstances;

public:
    // Constructeur
    Compteur() {
        nbInstances++; // Incrémentation du compteur à chaque création d'objet
    }

    // Destructeur
    ~Compteur() {
        nbInstances--; // Décrémentation du compteur à chaque destruction d'objet
    }
};
```

Dans cet exemple, nbInstances est un membre de données statique qui est partagé entre toutes les instances de la classe Compteur. Chaque fois qu'un objet est créé, nbInstances est incrémenté, et chaque fois qu'un objet est détruit, nbInstances est décrémenté.