Polycopié de Cours

Programmation Orientée Objets en C++

Plan Pédagogique du cours

Matière : Programmation Orientée Objet en C++

Filière : Electronique, Télécommunication, Génie Biomédical

Niveau : 1ere année Master (ESE, I, RT, IB)[1]

Volume Horaire : 45h Cours + Travaux Pratiques

Coefficient : 2

Crédits : 3

Evaluation : Contrôle continu : 40% ; Examen : 60%.

Objectif général du cours:

L'objectif général de ce cours est de permettre aux étudiants d'aborder les fondements de base de la programmation orientée objets ainsi que la maitrise des techniques de conception des programmes avancés en langage C++.

Les principaux points traités sont:

- Les structures de base du langage C++.
- L'allocation dynamique et la maitrise du fonctionnement des pointeurs.
- Le concept de classes et d'objets, les membres, fonctions membres, fonctions amies et le cas particulier très important des constructeurs et du destructeur.
- La notion d'héritage, simple puis multiple avec les notions de polymorphisme.
- La gestion des exceptions.

[1] ESE : Electronique pour les Systèmes Embarqués, I : Instrumentation, RT : Réseaux et télécommunications, IB : Instrumentation Biomédicale

SOMMAIRE

Chapitre I: Introduction à la Programmation Orientée Objets

I.1 Introduction

La conception par objet trouve ses fondements dans une réflexion menée autour de la vie du logiciel. D'une part, le développement de logiciels de plus en plus importants nécessite l'utilisation de règles permettant d'assurer une certaine qualité de réalisation. D'autre part, la réalisation même de logiciel composée de plusieurs phases, dont le développement ne constitue que la première partie. Elle est suivie dans la majorité des cas d'une phase dite de maintenance qui consiste à corriger le logiciel et à le faire évoluer. On estime que cette dernière phase représente 70 % du coût total d'un logiciel, ce qui exige plus encore que la phase de développement doit produire du logiciel de qualité.
La conception objet est issue des réflexions effectuées autour de cette qualité. Celle-ci peut-être atteinte à travers certains critères [6]:

- **La validité**: c'est-à-dire le fait qu'un logiciel effectue exactement les tâches pour lesquelles il a été conçu.
- **Extensibilité**: C'est-à-dire, la capacité à intégrer facilement de nouvelles spécifications (demandées par les utilisateurs ou imposées par un événement extérieur).
- **Réutilisabilité**: Les logiciels écrits doivent pouvoir être réutilisables, complètement ou en partie. Ceci impose lors de la conception une attention particulière à l'organisation du logiciel et à la définition de ses composantes.
- **Robustesse**: c'est-à-dire l'aptitude d'un logiciel à fonctionner même dans des conditions anormales.

I.2 Modularité

Les critères énoncés au paragraphe précédent influent sur la façon de concevoir un logiciel, et en particulier sur l'architecture logicielle. En effet, beaucoup de ces critères ne sont pas respectés lorsque l'architecture d'un logiciel est obscure. Dans ces conditions, le moindre changement de spécification peut avoir des répercutions très importantes sur le logiciel, imposant une lourde charge de travail pour effectuer les mises à jour.
On adopte généralement une architecture assez flexible pour parer à ce genre de problèmes, basée sur les modules. Ceux-ci sont des entités indépendantes intégrées dans une architecture pour produire un logiciel.

I.3 De la programmation classique vers la programmation orientée objet

Les premiers programmes informatiques étaient généralement constitués d'une suite d'instructions s'exécutant de façon linéaire (l'exécution commence de la première instruction du fichier source et se poursuivait ligne après ligne jusqu'à la dernière instruction du programme).

Cette approche, bien que simple à mettre en œuvre, a très rapidement montré ses limites. En effet, les programmes monolithiques de ce type:

- ne se prêtent guère à l'écriture de grosses applications
- et ne favorisent absolument pas la réutilisation du code.

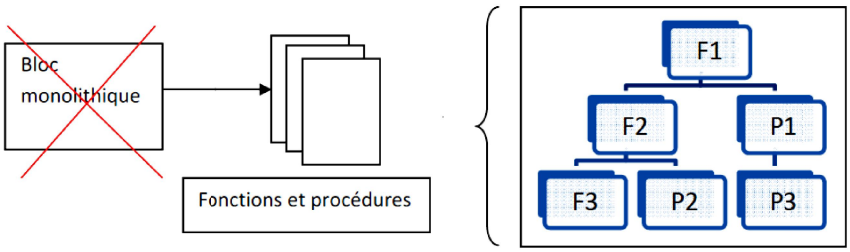
En conséquence, est apparue une autre approche radicalement différente: l'approche procédurale.

L'approche procédurale (classique) consiste à découper un programme en un ensemble de fonctions (ou procédures). Ces fonctions contiennent un certain nombre d'instructions qui ont pour but de réaliser un traitement particulier.

Exemples de traitements qui peuvent être symbolisés par des fonctions:

- Le calcul de la circonférence d'un cercle.
- L'impression d'un relevé de notes d'un étudiant.
- etc.

Dans le cas de l'approche procédurale, un programme correspond à l'assemblage de plusieurs fonctions qui s'appellent entre elles.



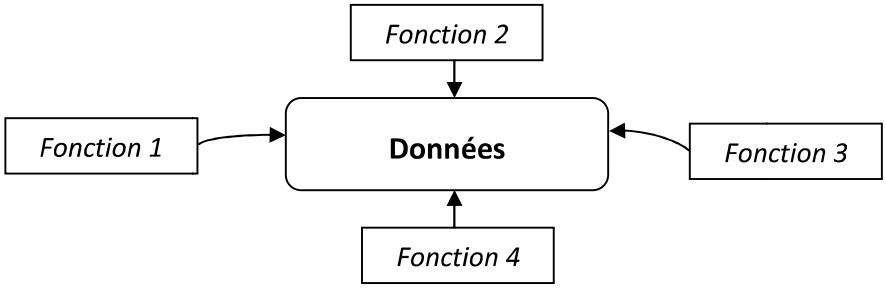
Exemple de langages de programmation procédurale: C, Pascal, Fortran, etc. [13]

L'approche procédurale favorise:

- La création d'un code plus modulaire et structuré.
- La possibilité de réutiliser le même code à différents emplacements dans le programme sans avoir à le retaper.

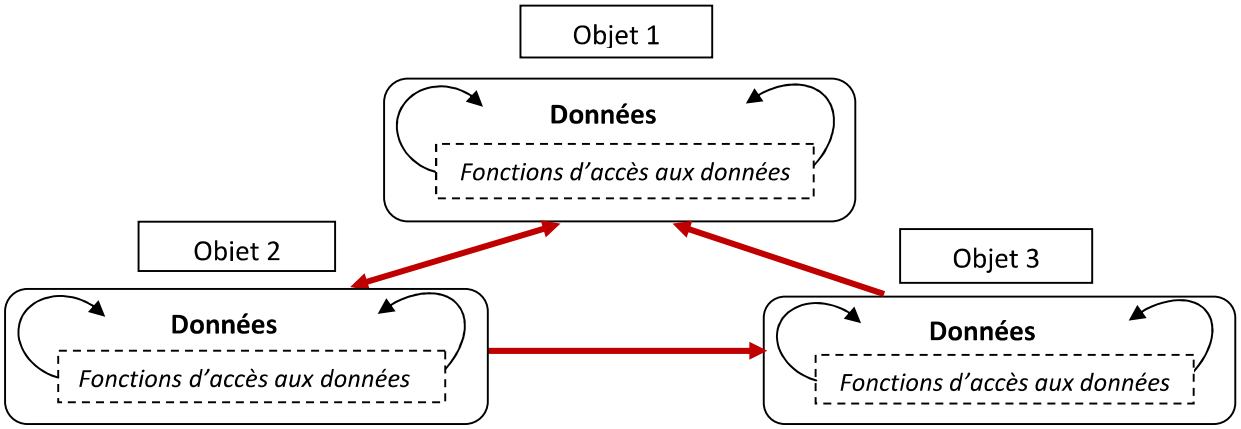
Malgré ses avantages, l'approche procédurale présente également des inconvénients:

- Les fonctions, procédures accèdent à une zone où sont stockées les données. Il y a donc une dissociation entre les données et les fonctions ce qui pose des difficultés lorsque l'on désire changer les structures de données.



- Dans les langages procéduraux, les procédures s'appellent entre elles et peuvent donc agir sur les mêmes données. Il ya donc un risque de partage de données (écriture en même temps dans le même fichier).

De ces problèmes est issus une autre manière de programmer c'est la programmation par objet ou bien *L'approche orientée objet (Début des années 80)*. Selon cette approche, un programme est vu comme un ensemble d'entités (ou objets). Au cours de son exécution, ces entités collaborent en s'envoyant des messages dans un but commun [13].



Nous avons dans ce schéma un lien fort entre les données et les fonctions qui y accèdent.
Mais qu'appelle-t-on un objet ? Que représente un objet ?

I.4 Conceptions par objets

Dans la conception basée sur les données, une réflexion autour des données conduit à :

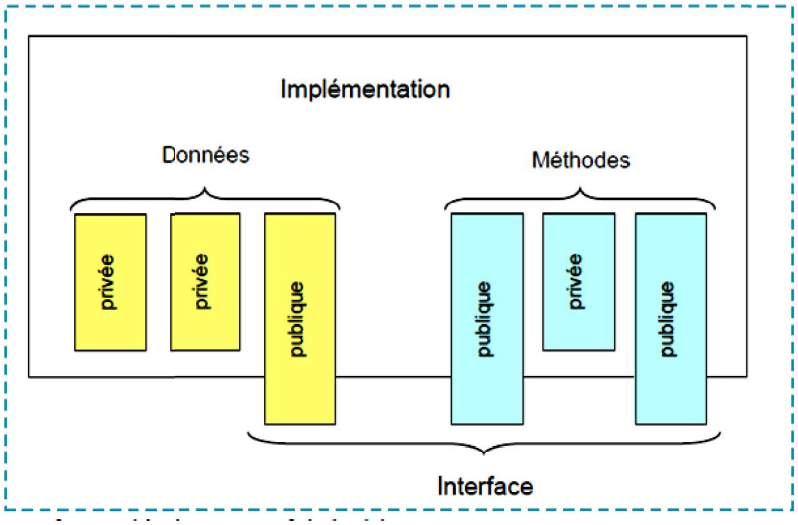
- Déterminer les données à manipuler.
- Réaliser, pour chaque type de données, les fonctions qui permettent de les manipuler.

On parle alors d'**OBJETS**
Un objet est une association de données et des fonctions (méthodes) opérant sur ces données.

Objet = Données + Méthodes

I.4.1 Concepts fondamentaux des objets [6]

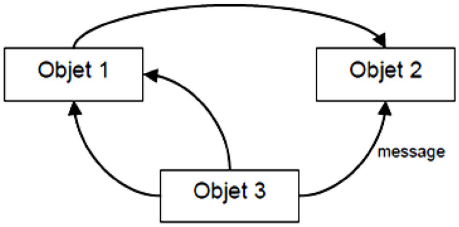
- ***Encapsulation des données*** : consiste à faire une distinction entre l'interface de l'objet et son implémentation.



⇒ Interface : décrit ce que fait l'objet.
⇒ Implémentation : définit comment réaliser l'interface.

Le principe de l'encapsulation est qu'on ne peut agir que sur les propriétés publiques d'un objet: les données sont toutes privées, leur manipulation se fait à travers les méthodes publiques.

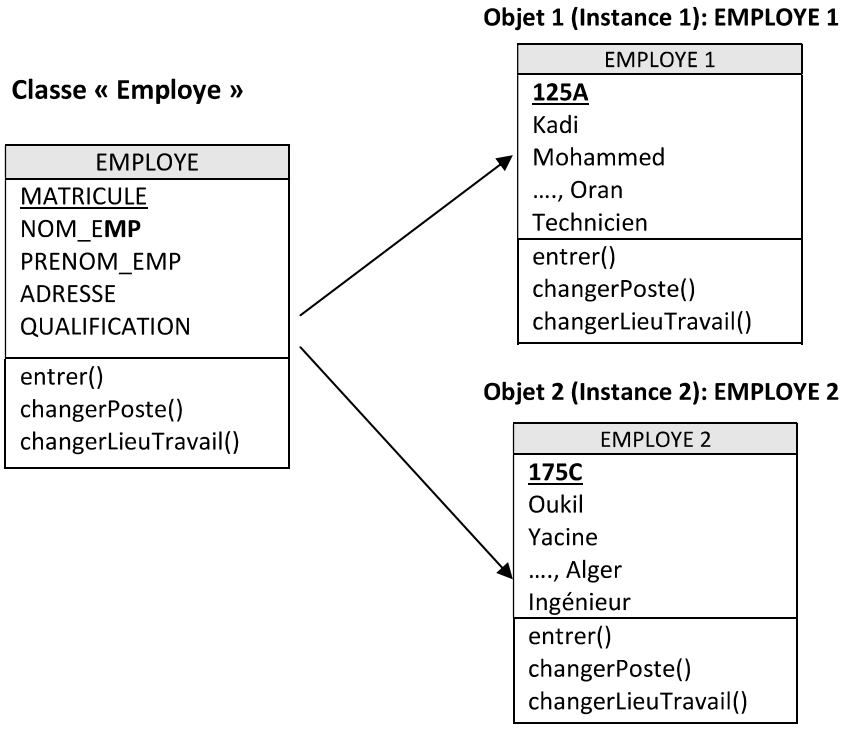
- ***Communication par messages :*** Les objets communiquent entre eux par des messages (un objet demande à un autre objet un service).



- ***Identité et classification***: consiste à regrouper les objets ayant le même comportement pour former un même ensemble, appelé **CLASSE (**cette notion n'est autre que la généralisation de la notion de *type*).

  Un objet d'une classe s'appelle **INSTANCE** de cette classe.

***Exemple*:**



« EMPLOYE 1 » et « EMPLOYE 2 » sont caractérisés par les mêmes propriétés (matricule, nom, prénom, qualification) mais associés à des valeurs différentes. Ils ont le même comportement (entrer/ changerposte,…) mais ont des identités différentes. Et il en serait de même pour tous les employés.

⇒ Tous les employés obéissent à un même schéma

- ***Héritage***: consiste à définir une nouvelle classe à partir d'une classe existante, à laquelle on ajoute des nouvelles propriétés (données ou méthodes).

- ***Polymorphisme***: possibilité à divers objets de classes dérivées d'une même classe de répondre au même message. Autrement dit, un même nom peut designer des propriétés de classes différentes.
- ***Généricité***: consiste à définir des classes paramétrées. Une classe générique n'est pas directement utilisable, mais permet de créer des classes dérivées qui peuvent être manipulées.

- ***Modularisation***: Les modules sont construits autour des classes. Un module contiendra l'implémentation d'une classe ou d'un ensemble de classes liées.