République Algérienne démocratique et Populaire

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université des Sciences et de la Technologies d'Oran « Mohamed Boudiaf »

FACULTE DE GENIE ELECTRIQUE

Département d'Automatique

TP n°2 de Programmation orientée objet

Réalisé par :

- RAHMOUN LOKMANE NOUR EL ISLEM.
- KABOUCHE BACIM.

Année universitaire 2019/2020

Préambule :

Notion général sur la programmation orienté objet :

Définition d'un objet :

Il s'agit d'un mélange de plusieurs variables et fonctions. Par exemple si on a créé un programme qui permet de résoudre des équations différentielles ordinaires : il peut afficher nos solutions, calculer des ordres de convergence, comparer deux méthodes ... Le code est complexe : il aura besoin de plusieurs fonctions qui s'appellent entre elles, ainsi que de variables pour mémoriser la solution au cours du temps, la méthode utilisée, le pas de temps choisi ... Au final, notre code est composé de plusieurs fonctions et variables. Il sera difficilement accessible par quelqu'un qui n'est pas un expert du sujet : Quelle fonction il faut appeler en premier ? Quelles valeurs doit-on envoyer à quelle fonction pour afficher la solution ? etc ... la solution est de concevoir un code de manière orientée objet. Ce qui signifie que nous placerons tout notre code dans une grande boîte. Cette boîte c'est ce qu'on appelle un objet. L'objet contient toutes les fonctions et variables mais elles sont masquées pour l'utilisateur. Seulement quelques outils sont proposés à l'utilisateur comme par exemple : définir mon pas de temps, mon intervalle de calcul et ma méthode, calculer l'ordre de la méthode, afficher la solution ...

En résumé:

- La programmation orientée objet est une façon de concevoir un code dans laquelle on manipule des objets.
- Les objets peuvent être complexes mais leur utilisation est simplifiée. C'est un des avantages de la programmation orientée objet.
- Un objet est constitué d'attributs et de méthodes, c'est-à-dire de variables et de fonctions membres.
- On appelle les méthodes de ces objets pour les modifier ou obtenir des informations.

Pour créer un objet, il faut d'abord créer une classe. Créer une classe consiste à définir les plans de l'objet. Une fois que la classe est faite (le plan), il est possible de créer autant d'objets du même type. Vocabulaire : on dit qu'un objet est une instance d'une classe.

But du TP :

L'objectif de ce TP est de découvrir les bases de la programmation orientée objet avec c++. Nous nous attacherons à construire et à manipuler une classe très simple pour mettre en pratique les notions de classe, d'objet, d'attribut, de méthode, de rétention d'information (encapsulation en anglais), de constructeur, ...

Notion de visibilité :

- L'encapsulation est l'idée de protéger les variables contenues dans un objet et de ne proposer que des méthodes pour les manipuler. L'objet est ainsi vu de l'extérieur comme une "boîte noire" possédant certaines propriétés et ayant un comportement spécifié.
- Le C++ permet de préciser le type d'accès des membres (attributs et méthodes) d'un objet. Cette opération s'effectue au sein des classes de ces objets :
  - public : les membres publics peuvent être utilisés dans et par n'importe quelle partie du programme.
  - privé (private) : les membres privés d'une classe ne sont accessibles que par l'objet lui-même.
  - Protégé (protected) : les membres sont accessibles par les classe qui hérite de cette classe.

Préprocesseur et directives de compilation:

Avant de compiler le programme, il est possible d'effectuer certaines modifications sur le code source. Le programme effectuant ces modifications s'appelle le préprocesseur. Les commandes destinées au préprocesseur commencent toutes par # en début de ligne.

1-Inclusion de fichiers:

Pour inclure un fichier à un certain endroit dans le fichier source, on écrit : #include "nom_du_fichier" Le contenu du fichier nom_du_fichier est alors inséré dans le fichier source. Le nom du fichier peut être écrit entre guillemets "nom_du_fichier" ou entre chevrons. Dans le premier cas, cela signifie que le fichier se trouve dans le même dossier que le fichier source, tandis que dans le deuxième cas, il s'agit d'un fichier se situant dans un endroit différent (ce fichier pouvant être fournit par le compilateur ou une librairie externe par exemple).

2-#define, #undef:

La directive #define permet de remplacer toutes les occurrences d'un certain mot par un autre. Par exemple : #define N 1143 Sur cet exemple toutes les occurrences de N seront remplacées par 1143. Cela est parfois utilisé pour définir des constantes. On préférera toutefois utiliser le mot-clé const. On peut très bien ne pas fixer de valeur et écrire : #define PLATEFORME_INTEL La variable de compilation PLATEFORME_INTEL est ici définie. Combiné à #ifdef, on pourra compiler ou non certaines parties du code à certains endroits du programme.

De la même façon que l'on peut définir une variable, on peut arrêter une définition en utilisant #undef. Son utilisation est rare, mais peut servir à ne plus définir une variable de compilation.

3-#ifdef, #ifndef, #if, #endif et #else:

Toutes ces directives permettent la compilation conditionnelle. C'est à dire que la partie du code comprise entre la directive conditionnelle (#ifdef, #ifndef ou #if) et la fin du bloc signalée par la directive #endif n'est compilée que si la condition est remplie. • La directive #ifdef permet de compiler toute une série de lignes du programme si une variable de compilation a précédemment été définie (par la directive #define). La directive #endif indique la fin de la partie de code conditionnelle. La partie du programme compilée sera toute la partie comprise entre le #ifdef et le prochain #endif.

4-#ifdef, #ifndef, #if, #endif et #else:

La directive #ifndef permet de compiler un bout de programme si une variable de compilation n'est pas définie. C'est donc l'inverse de #ifdef. La fin de la partie à inclure est déterminée également par #endif. •La directive #if permet de tester qu'une expression est vraie. Cette expression ne peut utiliser que des constantes (éventuellement définies par une directive #define), et la fonction defined permettant de tester si une variable de compilation existe.

La notion de constructeur

Le constructeur est la fonction membre appelée automatiquement lors de la création d'un objet (en statique ou en dynamique). Cette fonction membre est la première fonction membre à

Etre exécutée il s'agit donc d'une fonction permettant l'initialisation des variables.

Le constructeur d'un objet porte le même nom que la classe et ne possède aucune valeur de retour (même pas *void*).

- un constructeur porte le même nom que la classe dans laquelle il est défini.
- un constructeur n'a pas de type de retour (même pas *void*).
- un constructeur peut avoir des arguments.

La définition de cette fonction membre spéciale n'est pas obligatoire (si vous ne souhaitez pas initialiser les données membres par exemple) dans la mesure où un *constructeur par défaut* (appelé parfois *constructeur sans argument*) est défini par le compilateur C++ si la classe n'en possède pas.

L'appel du constructeur se fait lors de la création de l'objet. De ce fait, l'appel du constructeur est différent selon que l'objet est créé de façon statique ou dynamique :

- en statique : le constructeur est appelé grâce à une instruction constituée du nom de la classe, suivie par le nom que l'on donne à l'objet, et les paramètres entre parenthèses.

- en dynamique : le constructeur est appelé en définissant un pointeur vers un objet du type désiré puis en lui affectant la valeur retournée par l'opérateur *new*.

Il est possible d'utiliser des valeurs par défaut pour les arguments, afin d'éviter à avoir à entrer de façon répétitive un ou plusieurs paramètres portant généralement la même valeur.

Les destructeurs

Il s'agit d'une fonction membre qui intervient automatiquement lors de la destruction d'un objet. Il permet ainsi d'une certaine façon d'exaucer ses dernières volontés…

Le destructeur est une fonction membre dont la définition ressemble étrangement à celle du constructeur, hormis le fait que le nom du destructeur est précédé d'un *tilde* (~), et qu'il ne possède aucun arguent.

- un destructeur porte le même nom que la classe dans laquelle il est défini et est précédé d'un tilde.
- un destructeur n'a pas de type de retour (même pas *void*).
- un destructeur ne peut pas avoir d'argument.
- la définition d'un destructeur n'est pas obligatoire lorsque celui-ci n'est pas nécessaire.

Les destructeurs ont en général beaucoup moins besoin d'être définis que les constructeurs, c'est donc le destructeur par défaut qui est appelé le cas échéant. Toutefois, lorsque les objets sont chaînés dynamiquement grâce à des pointeurs (lorsqu'une fonction membre d'un objet est un pointeur vers un objet de même type par exemple), ou dans d'autres cas particuliers la définition d'un destructeur permettant de « nettoyer » l'ensemble des objets peut être indispensable
Le destructeur, comme dans le cas du constructeur, est appelé différemment selon que l'objet auquel appartient a été créé de façon statique ou dynamique.

Le destructeur d'un objet créé de façon statique est appelé de façon implicite dès que le programme quitte la portée dans lequel l'objet existe.

Le destructeur d'un objet créé de façon dynamique doit être appelé grâce au mot clé *delete*, qui permet de libérer la mémoire occupée par l'objet.
Bien évidemment un destructeur ne peut être surchargé, ni avoir de valeur par défaut pour ses arguments.

<cstdlib> (stdlib.h) :

Bibliothèque d'utilitaires généraux standard C

Cet en-tête définit plusieurs fonctions à usage général, y compris la gestion dynamique de la mémoire, la génération de nombres aléatoires, la communication avec l'environnement, l'arithmétique d'entiers, la recherche, le tri et la conversion.

Srand :

Le générateur de nombres pseudo-aléatoires est initialisé à l'aide de l'argument passé comme valeur de départ.

Pour chaque valeur de départ différente utilisée dans un appel à srand, on peut s'attendre à ce que le générateur de nombres pseudo-aléatoires génère une succession différente de résultats dans les appels suivants à rand.

Deux initialisations différentes avec la même graine généreront la même succession de résultats dans les appels ultérieurs à rand.

Si seed est mis à 1, le générateur est réinitialisé à sa valeur initiale et produit les mêmes valeurs qu'avant tout appel à rand ou srand.

Afin de générer des nombres aléatoires, srand est généralement initialisé à une valeur d'exécution distincte, comme la valeur renvoyée par la fonction time (déclarée dans l'en-tête <ctime>). Ceci est assez distinctif pour la plupart des besoins de randomisation triviaux.

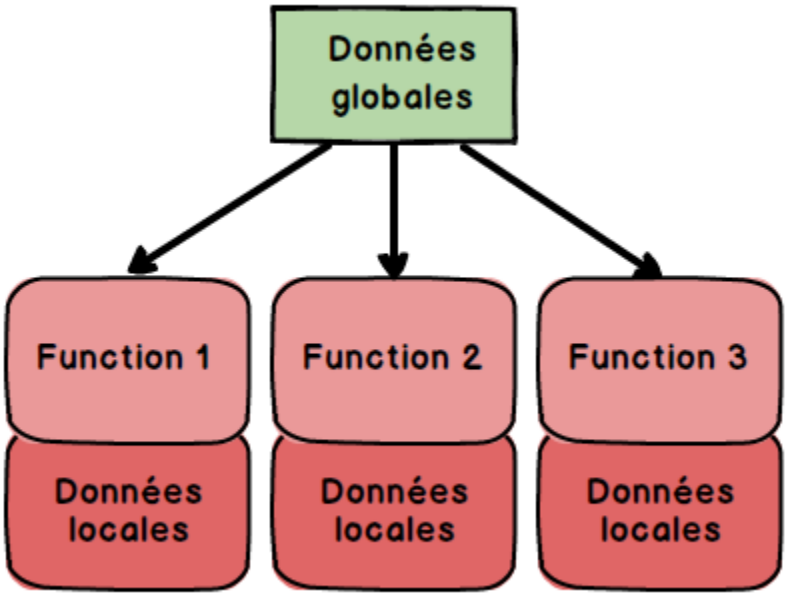
<Ctime>:

This header file contains definitions of functions to get and manipulate date and time information.

Programmation procédurale et programmation orienté objet :

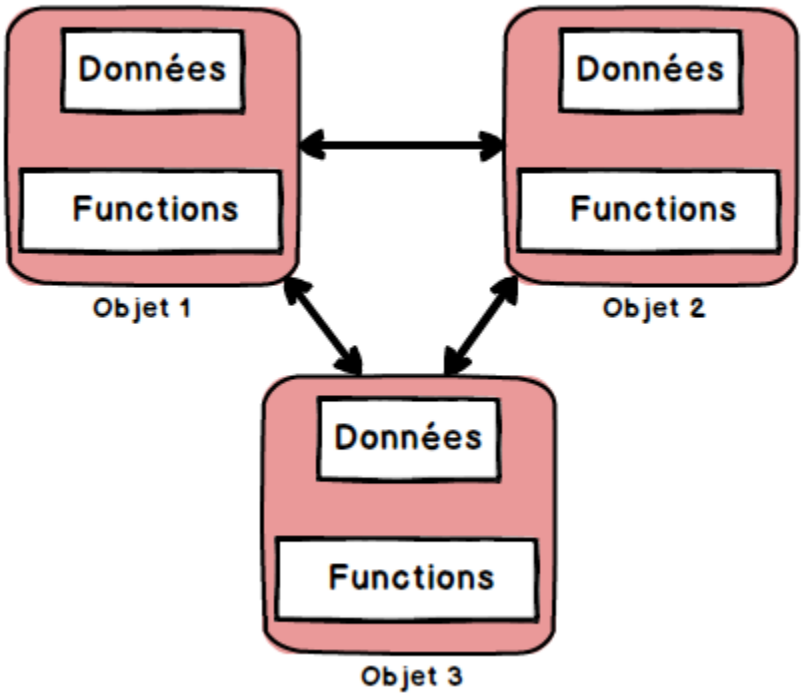
La différence entre la programmation procédurale et la programmation orientée objet (POO) réside dans le fait que dans la programmation procédurale, les programmes sont basés sur des fonctions, et les données peuvent être facilement accessibles et modifiables, alors qu’en programmation orientée objet, chaque programme est constitué d’entités appelées objets, qui ne sont pas facilement accessibles et modifiables.

-la programmation procedural:



Dans la programmation procédurale le programme est divisé en petites parties appelées procédures ou fonctions. Comme son nom l'indique, la programmation procédurale contient une procédure étape par étape à exécuter. Ici, les problèmes sont décomposés en petites parties et ensuite, pour résoudre chaque partie, une ou plusieurs fonctions sont utilisées.

-la Programmation Orientée Objet (POO) :



Dans la programmation orientée objet le programme est divisé en parties appelées objets.
La programmation orientée objet est un concept de programmation qui se concentre sur l'objet plutôt que sur les actions et les données plutôt que sur la logique.

Table de comparison

	Programmation Procédurale	Programmation Orientée Objet
Programmes	Le programme principal est divisé en petites parties selon les fonctions.	Le programme principal est divisé en petit objet en fonction du problème.
Les données	Chaque fonction contient des données différentes.	Les données et les fonctions de chaque objet individuel agissent comme une seule unité.
Permission	Pour ajouter de nouvelles données au programme, l'utilisateur doit s'assurer que la fonction le permet.	Le passage de message garantit l'autorisation d'accéder au membre d'un objet à partir d'un autre objet.
Exemples	Pascal, Fortran	PHP5, C ++, Java.
Accès	Aucun spécificateur d'accès n'est utilisé.	Les spécificateurs d'accès public, private, et protected sont utilisés.
La communication	Les fonctions communiquent avec d'autres fonctions en gardant les règles habituelles.	Un objet communique entre eux via des messages.
Contrôle des données	La plupart des fonctions utilisent des données globales.	Chaque objet contrôle ses propres données.
Importance	Les fonctions ou les algorithmes ont plus d'importance que les données dans le programme.	Les données prennent plus d'importance que les fonctions du programme.
Masquage des données	Il n'y a pas de moyen idéal pour masquer les données.	Le masquage des données est possible, ce qui empêche l'accès illégal de la fonction depuis l'extérieur.

Conclusion

Les failles de la programmation procédurale posent le besoin de la programmation orientée objet.
La programmation orientée objet corrige les défauts de la programmation procédurale en introduisant le concept «objet» et «classe». Il améliore la sécurité des données.
La programmation orientée objet permet de créer plusieurs instances de l'objet sans aucune interférence.

La programmation orientée objet (POO) est une méthode que nous utilisons pour écrire du code afin de faciliter le codage.

La programmation orientée objet n'est donc qu'une méthode de travail et rien de plus, et elle n'est pas spécifique au C ++ car elle est appliquée dans d'autres langages de programmation.

L'idée de la programmation orientée objet en général est de préparer la forme dans laquelle les informations seront sauvegardées, ce qui rend l'accès et la modification très faciles.