Chapitre1

Concepts de base de la programmation

Orienté Objets C++

Introduction

Les langages informatiques ont subit une évolution successive depuis la création de la discipline informatique. A ce jour on distingue trois familles (03) de langages de programmation

1- Langages de bas niveau tel que l'Assembleur
2- Langages procéduraux tel que Pascal, C
3- Les langages orientés objets (langages non procéduraux) tel que C++, Java)

Chacune de ces langages de programmation à sa philosophie (cycle de vie, structuration, complexité, etc.) mais sont complémentaires.

Nous nous intéressons aux langages de programmation orientés objets noté par L.O.O qui est un grand paradigme de programmation mathématiques et Informatique. C'est une nouvelle méthode de programmation qui tient à se rapprocher de notre manière naturelle d'appréhender le monde réel.

Principe

Un programme informatique comporte toujours des données et des traitements. Maintenant si la programmation structurée (programmation procédurale) s'intéresse aux traitements puis aux données alors la programmation orientée objets s'intéresse aux données puis aux traitements, d'où la question que nous posons dans ce cas est '' Sur quoi porte le programme'' et non '' que fait ce programme''.

La programmation orientée objets

Qu'est ce qu'un Objet ?

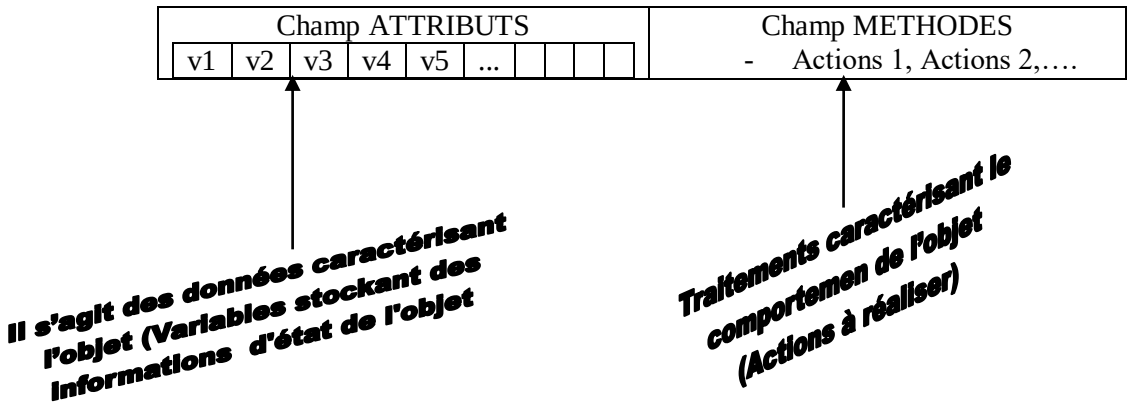
Un objet en programmation non procédurale (C++) est une entité informatique possédant une structure et un comportement et bien sûr son état interne. Il représente un concept, une idée, une fonction, une équation etc.

Mathématiquement parlant, un objet est modélisé sous forme d'une structure de donnée composée de deux grandes zones. La première zone contient la définition et la déclaration des données appelées ''ATTRIBUTS''. La deuxième zone contient les traitements appelés ''METHODES''.

Donc un objet est un support logique servant à stocker les données dans les zones attributs et à les gérer aux travers des méthodes.

Structure de donnée de l'objet :

Nom de l'objet : l'objet possède toujours son identifiant qui permet de le distinguer des autres.



Comparaison

Par rapport aux langages procéduraux (C, Pascal, …):

- le champ ''ATTRIBUT'' de l'objet sont ce que les ''variables '' sont à un programme simple.
- le champ ''METHODES'' de l'objet sont ce que les ''sous-programmes'' destinés à traiter les données

Exemple1

Soit l'objet $Z = a + i * b$ un nombre complexe dont *a* est la partie réelle et *b* la partie imaginaire.

Cet objet est caractérisé par les données *a et b.* Il est manipulé par des traitement ou procédures caractérisées par leurs noms suivi de paramètres réels et formels d'appels et de résultats qui sont par exemple ModuloZ(a, b), ARGZ(a, b) etc.

Exemple2

Soit l'équation EDP à 1D qui décrit le phénomène physique d'un polluant transporté par un courant d'eau avec une vitesse constante α suivante :

$$\begin{cases} \frac{\delta u(x,t)}{\delta t} + \alpha \frac{\delta u(x,t}{\delta x} = 0 \\ avec Cinit, C\lim et\ \alpha\ la\ vitesse de l'eau \end{cases}$$

Cet objet est caractérisé par les données *de* conditions initiales *''Cinit'',* de conditions aux limites *'''Clim''* et la vitesse constante de l'eau *''α''* ainsi que d'autres variables qui peuvent apparaitre lors de la résolution du problème comme le pas de discrétisation, la variable matrice etc. Cet objet à 1D est manipulé par des traitement ou procédures caractérisées par leurs noms suivi de paramètres réels et formels d'appels et de résultats qui sont par exemple

Classe d'objet

Avec la notion d'objet, il convient d'amener la notion de classe.

On appelle classe la structure d'un objet, c'est-à-dire la déclaration de l'ensemble des entités qui composent l'objet (en réalité on dit qu'un objet est une instanciation d'une classe).

Revenant maintenant à l'intérêt de la P.O.O :

Pour concevoir et développer des logiciels de haute qualité, on fait appel à la programmation orientée objets C++ car elle permet la bonne structuration de l'écriture des blocs actions de résolution, simplifier la réutilisation du code, améliorer la protection et la sureté des données.

Fondamentaux de la programmation orientée objets

Il existe trois (03) fondamentaux de la programmation orientée objets qui sont l'Encapsulation, l'Héritage et le polymorphisme. Bien sûr il existe d'autres mais on ne va pas les voir cette fois ci.

a- Encapsulation

Encapsuler $\Longleftrightarrow$ Empêchant (Interdire) l'accès aux données.
Encapsuler $\Longleftrightarrow$ Garantir l'intégrité des données (protéger les données).

En conséquence : l'encapsulation est un mécanisme qui consiste à rassembler l'entité objet (Les données et les traitements) au sein d'une structure en cachant l'implémentation de l'objet.
En effet, l'encapsulation permet de définir des niveaux de visibilités des éléments de la classe objet. Ces niveaux définissent les droits d'accès aux données. Trois niveaux de visibilités sont offerts par les langages orientés objet C++ :
(En C++ on utilise le mot réservé 'public').

- Niveau public : les fonctions (traitements) de toutes les classes peuvent y accéder aux données ou aux ''méthodes'' d'une classe.
- Niveau protégé : l'accès aux données est réservé aux fonctions de la classe héritières seulement. (le mot réservé utilisé est 'protected'.
- Niveau privé : l'accès aux données est limité seulement aux méthodes de la classe d'objet elle-même 'le mot réservé utilisé est 'private'.

b- Héritage

L'héritage est un principe propre à la programmation orienté objets (P.O .O) qui permet de créer une nouvelle classe d'objet à partir d'une classe existante (on appelle ça dérivation d'une classe).

Cette notion provient du fait que la classe dérivée contient les 'attributs' et les 'méthodes' de sa superclasse (la classe dont elle dérive) et peut même ajouter et définir ses propres 'attributs' et 'méthodes'.

Pourquoi créer une sous classe ?

La permission de la création d'une sous classe à partir d'une classe existante est autorisée pour éliminer la redondance d'informations (ne pas répéter la définition et la déclaration des

mêmes 'attributs' et la description des mêmes 'méthodes' qui existe déjà dans la superclasse) et c'st pour augmenter la finesse de la modélisation d'objet.

Syntaxe:

<Nom de la nouvelle classe> hérite de <nom de la superclasse>

{- définition du champ 'nouveau attributs' de la sous classe

_- définition du champ 'nouvelles méthodes' de la sous classe

} ;

c- Polymorphisme

Dans un programme orientée objets, on peut définir plusieurs objets, donc plusieurs classes, qui peuvent y également contenir des méthodes de mêmes noms mais dont la syntaxe est différente (c'est exactement les cas qui arrivent lors de l'héritage). Ceci pose un problème lors de l'appel à ces méthodes. Le polymorphisme consiste alors à appeler la bonne méthode d'après la syntaxe de l'objet implémenté en faisant recourir à l'utilisation des pointeurs par exemple.

Remarque

Chaque objet à ses méthodes propres, cependant deux parmi elles existent toujours a savoir 'constructeur' et destructeur' qui sont utilisées pour la gestion mémoire qu'on ne va pas étudier dans ce cours.

Chapitre2

Création de Classes et Objets

1- Introduction

Dans la programmation orientée objets, nous allons apprendre à créer des objets donc de classes.

Pour créer un objet, il faut d'abord créer une classe ! Je m'explique, pour implémenter un programme il nous faut une machine '' micro-ordinateur ''. Donc pour nous la classe d'objet est le micro-ordinateur et que l'objet est bien le programme à exécuter. Une fois nous avons cette machine alors nous pouvons écrire et exécuté autant de programme que nous voulons.

Rappel

Une classe est une structure de donnée (entité informatique) constituée d'Attributs (variables) et méthodes (procédures et fonctions). Chaque Attribut est défini par son nom et son type et chaque méthode est décrite par une séquence de lignes d'actions de résolution.

2- Création de classe

La syntaxe ou le code associé à une classe est donné par le mot réservé '' class'' en lettres minuscules.

On écrit :

Code 1.1

```
 class <nom de la classe objet>

{-action1

  -action2

  - ..

  - action n

} ;
```

Les actions écrites entre le début '{' et la fin '}' représente la définition de l'objet qui appartient à la classe qui a pour nom <nom de l'objet> à choisir aléatoirement. Ces actions ne sont que des attributs et des méthodes.

La fin de la classe objet est toujours marqué par le symbole ' **;**'.

2.1- Insertion de méthodes et d'attributs

Il faut prendre beaucoup de temps à réfléchir pour remplir le corps de la classe d'objets. Il ne faut pas coder comme des ignorants dés le début. On commence généralement par la déclaration et la définition des variables associées à la résolution du problème posé connues par le prédicat 'attribut'. Pour cela il faut toujours analyser le problème et voir qu'est ce qui caractérise la classe. Ensuite décrire les codes des méthodes (connues par procédures et fonctions), qui ont une syntaxe un peu particulière par rapport au langage de programmation C, sans oublier de bien réfléchir aux passages paramètres et comment on l'utilise lors des appels à partir du programme principale.

D'après mon expérience, ceci ne peut se comprendre que si on passe par un exemple simple et clair.

Exemple

Reprenons l'exemple 2 du premier chapitre .Soit l'équation aux dérivées partielles de dimension1 suivante :

$$\begin{cases} \frac{\delta u(x,t)}{\delta t} + \alpha \frac{\delta u(x,t}{\delta x} = 0 \ , t \geq 0 \ et \ x \in R \\ à \ l'instant t = 0, U(x,0) = U_0(x) : condition initiale \\ j = 0 \ ou \ j = N+1 : condition limites \ au \ bord \ du \ domaine \\ \alpha \ vitesse constante \end{cases}$$

Ecrire un programme qui Calcule la solution approchée U_j^{n+1}.

Pour le moment on ne s'intéresse qu'à la création et la définition de l'objet et non à sa résolution complète.

Notre objet est une Equation aux dérivées partielles linéaire à une seule direction spatiale *x* (EDP à 1D), elle est identifiée par exemple par le nom choisi arbitrairement (mais il vaut mieux choisir un nom qui correspond directement à la nature du problème) : EQTRANS

L'objet EQTRANS est caractérisé par les attributs : variable de dépendance *U,* variables indépendantes *t* et *x,* condition initiale donnée $U(x,0) = U_0(x)$, conditions aux limites données et la vitesse de propagation constante α et c'est possible d'ajouter d'autres variables secondaires intervenant lors de résolution.

Les méthodes associées à l'objet EQTRANS sont par exemple : introduction des données, opération de discrétisation, solution, résultats.

Ecrivant maintenant la modélisation ou l'entité informatique correspondante:

Soit le code 2.2

class EQTRANS

//définition des attributs associés à l'objet EQTRANS

{ private : //protection

float *U,x,t,alpha* ;

int *j,N* ;

…. ;

public : //les méthodes sont grosso modo les actions élémentaires de résolution

void initialisation (paramètres réelles associées)

{ //décrire la procédure ou la fonction qui permet de faire l'entrée des données comme dans le cas d'un programme simple en langage de programmation C

}

void résolution (paramètres réelles associées)

{ // la même chose, écrire le programme correspondant au nom de la méthode ''résolution '' }

void résultat (paramètres réelles associées)

{ // la même chose, écrire le programme correspondant a l'impression des résultats }

} **;** // fin de la création de la classe d'objet

3- Utilisation de l'objet (extérieur de l'objet)

Utiliser un objet revient à faire exécuter ou fonctionner un objet en faisant appel aux différentes méthodes (procédures et fonctions) associées. Cela se traduit par l'écriture du programme principal suivant dont la syntaxe est :

int main (paramètres du système d'exploitation du micro-ordinateur utilisé ou bien c'est vide)

{ //On ne trouve dans cette séquence que les noms de méthodes à appeler suivi de leurs paramètres réels. Rarement ou écrit des algorithmes dans ce corps main//

<[Déclarations facultatives]> ;

<Nom de l'objet>**.** <Nom de méthode> (paramètres réels) **;**

return(0) ; }

Remarques

- Pour la dernière ligne du programme main, on écrit soit return(0) ou bien return ; ou bien return() ; tout dépend du type de programme principal main.
- Faites attention à la protection (Encapsulation) au moment des appels.
- Par défaut, si on n'utilise pas les niveaux de visibilités tout le contenu de la classe d'objet est ''private'' donc protégé et on ne peut pas réaliser les appels.

Reprenons l'exemple 2 et écrivant le tout (soit le code 3.3)

```
class EQTRANS
//définition des attributs  (données) associés à l'objet EQTRANS
{ private :     //protection
  float U,x,t,alpha ;
int j,Taille ; …. ;
  public :       //les méthodes sont grosso modo les actions élémentaires de résolution et ne sont pas protégé grâce au mot prédéfini ''public ;''
int Taille()
{//les actions élémentaire de lecture de la taille
 return(n) ;}
void discrétisation(paramètres formels associés)
{// Ecrire le programme correspondant a cette méthode  }
 void initialisation (paramètres réelles associées)
{    //décrire la procédure ou la fonction qui permet de faire l'entrée des données comme dans le cas d'un programme simple en langage de programmation C
}
  void résolution (paramètres formels associées)
{ // la même chose, écrire le programme correspondant au nom de la méthode ''résolution '' }
 void résultat (paramètres formels associées)
{ // la même chose, écrire le programme correspondant a l'impression des résultats  }
} ;  // Fin de la création de la classe d'objet
    int main ( )
```

```
{ //On ne trouve dans cette séquence que les noms de méthodes à appeler suivi de leurs paramètres réels. Rarement ou écrit des algorithmes dans ce corps main//

EQTRANS EDP  ;  //EDP de type EQTRANS

Int N,

EDP.dicrétisation(paramètres réels d'appel) ;

N=EDP.Taille( );

EDP.initialisation( les paramètres d'appels) ;

EDP.résolution(les paramètres d'appels ) ;

EDP.résultat (les paramètres d'appels) ;

return() ;

 }
```

Ici je n'ai pas explicité clairement les paramètres d'appels et leurs modes, on va étudier plus en détail au fur et à mesure qu'on avance dans les cours.

A ce stade, un seul fichier principal d'extension cpp (main.cpp) dans lequel figure-la classe EQTRANS.

Du moment que la programmation orientée objets est une nouvelle technique de programmation, on s'attend alors surtout à la finesse de la modélisation, à la qualité des logiciels et à l'organisation. Est bien oui, la Programmation orientée objets nous offre cette possibilité de décrire le code de la création de la classe d'objet plus modulaire en utilisant la technique séparer prototypes. C'est-à-dire séparer les procédures et les fonctions (les méthodes) en prototypes et définitions dans deux fichiers différents au lieu d'un seul comme c'est présenté dans le code 2.2. (voir Travaux pratiques).

Chapitre 3

Présentation générale de C++

Squelette et fonctionnement