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 ?

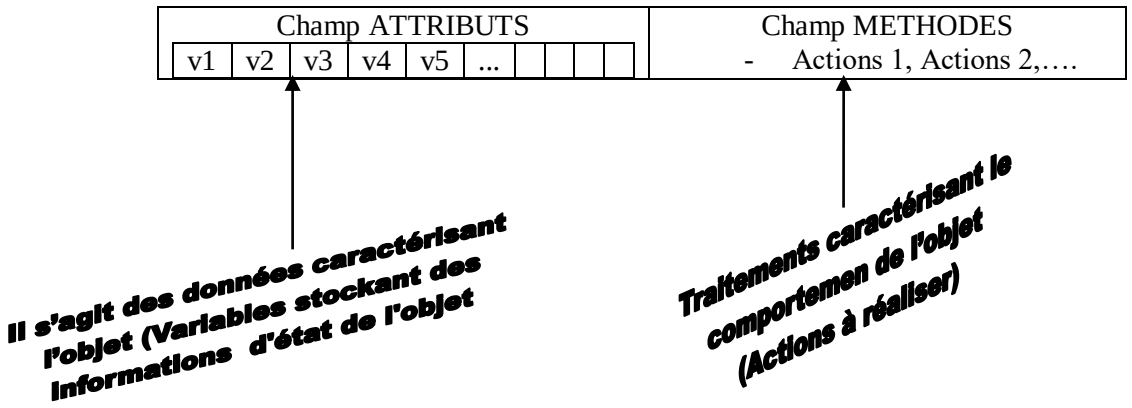
Un objet en programmation non procédurale (C++) est une entité informatique possédant une structure et un comportement et bien sûr son état interne. Il représente un concept, une idée, une fonction, une équation etc.

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 :

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

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

Vocabulaire

Le langage de programmation orienté objets C++ met à notre disposition plusieurs formes de symboles, on distingue : caractères numériques 0 ..9, des caractères alphabétiques a..z et A..Z et aussi des caractères spéciaux tels que opérateurs arithmétiques, opérateurs de relations et opérateurs de comparaisons, des séparateurs, caractère de monnaie. L'utilisation et la manipulation de ces symboles exige de nous, de connaitre leurs natures et leurs syntaxe (forme d'écriture).

Notion de type

Dans la majorité des langages de programmation, la notion de type à pour rôle d'explicite le fait que les données n'ont pas la même nature pour réaliser des opérations d'affectation et de stockage. Le langage de programmation orienté objets C++ dispose de plusieurs formes de types pour les variables et qui doivent obligatoirement identifier et déclarer au préalable avant de s'en servir dans les traitements. Les types les plus employés sont int, float, string, char et bool.

Syntaxe : <type> < (variables)> ;

Exemple soit x une variable de type entier, sa déclaration est : int(x) ;

X une chaine de caractères, sa déclaration est : string(X) ;

Notion de variables

En C++, une variable est un identificateur, composé de caractères alphanumériques dont le 1er caractère est alphabétique. Il peut aussi comporter seulement le caractère spécial ''_ '' (celui de la touche du chiffre 8 du clavier). Cet identificateur est une quantité qui possède un type et peut varie en fonction du temps. On distingue deux sortes de variables :

- variable simple qui ne peut contenir qu'une seule valeur à un moment donné.

- variable indicée qui peut contenir plusieurs valeurs à un moment donné. Autrement dit, la variable indicée est considérée comme structure de données statiques ou bien une suite de cases contiguës qui permet de stocker de façon ordonnée des données de même type. Chaque case est repérée par un ou plusieurs indices de type entier. Exemple un vecteur, une matrice

Remarques importantes :

- toutes variables utilisées dans un programme en C++ doit être déclarer.

Une variable simple z se crée et se déclare par son type suivi de son nom. Exemple float z ;

Une variable indicée se crée et se déclare comme suit :

typedef <type> <nom de la variable indicée> [taille1][taille2]

#define [taille1] < valeur correspondante à la variable taille1>

#define [taille2] < correspondante à la variable taille1>

Exemple

Soit A (20, 30) une matrice bande de valeurs correspondant à l'équation aux dérivées partielles elliptique à 2D et soit Tab(10) un tableau ou vecteur de donné entier correspondant à une équation aux dérivées partielle de transport à 1D (D: espace ou direction spatiale). Définir A et Tab.

Alors, on voit bien que la matrice A et le vecteur Tab sont des variables indicées qui chacun d'eux contient plusieurs données de même type. Leurs définitions et déclarations est :

typedef float A[taille1][taille2]

#define taille1 20

#define taille2 30

typedef int Tab[taille3]

#define taille3 10

- Faites attention, le C++ fait une distinction entre les lettres minuscules et les lettres majuscules
- L'initialisation des indices associés aux variables indicées se commencent toujours à partir de la valeur zéro
- Le C++ utilise aussi le qualificatif ''const'' pour déclarer des variables indicées (Si on reprend l'exemple ci-dessus, alors on déclare comme suit :
  const taille1=20, taille2=30, taille3=10 ;
  float A[taille1][taille2];
  int Tab[taille3] ;

Affectation

Comme tout langage de programmation, une affectation est une assignation d'un résultat à une variable en utilisant le symbole d'égalité ''='' . Sa syntaxe est :

<Nom de variable> = <expression arithmétique ou valeur> ;

En plus de symbole''='', le C++ autorise l'utilisation d'une autre forme d'assignation en utilisant les symboles de parenthèses '' ( )''.

Exemple

1) si x est une variable qui reçoit la valeur 100 alors on écrit :

int x=100 ; ou bien int x(100) ;

2) soit les variables y=1, z=6 et t=0.
Commentez et donnez les résultats des écritures bool x(y=z) ; puis bool x(y=t) ;
Pour bool x(y=z) , nous avons x une variable booléenne qui reçoit la valeur qui est mis entre parenthèse après son exécution bien sûr. C'est-à-dire on assigne à la variable y la valeur de la variable z, donc y=6 et sa valeur ancienne est écrasée, ensuite cette nouvelle valeur de y sera affectée à la variable logique x et comme le type logique ne peut prendre que 0 ou 1, donc le résultat final de x est bien 1.
A vous maintenant de commentez la deuxième !

Opérateurs de comparaison et de relation

Les opérateurs de comparaison et les opérateurs de relation sont des opérateurs logiques qui renvoient toujours un et un seul résultat vrai ou faux. Généralement, ces opérateurs on les retrouve dans la syntaxe d'écritures des primitives conditionnelles d'actions qu'on décrira par la suite.

Les symboles autorisés par le C++ sont

symbole	Désignation
<	Inférieur
>	Supérieur
>=	Supérieur ou égale
<=	inférieur ou égale
!=	Non égale (différent)
==	Egale (égalité)
&&	Le 'et' logique
||	Le 'ou' logique
!	La négation
?:	Opérateur ternaire des expressions alternatives

Les Entrées /sorties

Le langage orienté objets non procédural C++ dispose de plusieurs instruction d'ordre traitement et d'ordre technique ou organisationnelle. Plus spécialement celles qui permettent l'introduction des données et l'impression des résultats. Ce genre d'instructions s'appellent les entrées et les sorties.

On peut utiliser les instructions classiques d'entrée et sorties du langage procédural ou impératif C, qui exigent des formatages des données (le spécification de format), cependant,

le C++ offre de nouvelles possibilités d'utiliser des instructions ne nécessitant pas de formatages et qui sont très simples à apprendre et à insérer dans des programmes, on distingue

- Instruction de lecture (ou introduction des données) : cin>>variables ;
- Instruction de sortie (ou d'impression des résultats) : cout<<variables ;

Nouvelles Instructions	Instructions classiques	Equivalence d'écriture (pour int n; par exemple)
Cin	scanf	cin>>n ; équivalent à scanf(''%d'',&n) ;
Cout	puts, printf	cout<<n ; équivalente à printf(''%d'',n) ; Et l'instruction 'puts' pour imprimer les messages parce qu'il n'y a pas de formatages [puts('ça marche') ;]

Ces instructions sont classées et rangées dans des fichiers appelés bibliothèque ou bien librairies. Pour pouvoir utiliser ces instruction en C++, il faut déclarer leurs bibliothèques tout d'abord à l'entête du programme en utilisant le mot prédéfini 'include' dont la syntaxe est : #include<nom de bibliothèque**.h**>

D'ailleurs, le langage C++ comme le langage impératif C recommande la déclaration de toutes les bibliothèques des différentes instructions utilisées dans un programme afin de les identifier et faire la différence avec les variables qui sont écrites dans ce même programme.

On présente quelques bibliothèques correspondantes aux différentes instructions de C++ les plus souvent utilisé:

Bibliothèques	Instructions
stdio.h	Instruction d'entrées et sorties classiques: printf, puts, scanf
iostream.h	Les nouvelles instructions d'entrées et de sorties : cin, cout
math.h	Toutes les fonctions standards de mathématiques
conio.h	Instructions d'ordre techniques et organisationnelles : clscr, getch(), …

Opérateurs d'incrémentation et de décrémentation

Les opérations d'incrémentations et de décrémentations des variables peuvent être effectuée respectivement par des opérateurs unaires '++' et '--'. L'avantage de l'utilisation de ces opérateurs est qu'ils rendent des programmes plus court, bien structurés et moins de risque de faire des erreurs lors de la l'écriture des programmes.

On passe par un exemple pour mieux comprendre :

Soit la séquence de programme

```
int i=10 ;

int x=i++ ;
```

int y=i-- ;

Tout d'abord, cette forme est dite post-incrémentation/ post-décrémentation car l'opérateur ++/-- est situé après la variable. Le traitement de cette séquence se fait dans l'ordre suivant:

1- On affecte à la variable simple de type entier i la valeur 10
1- On affecte à la variable simple entière x la valeur de i (donc x reçoit 10)
2- On incrémente la variable i de 1(i reçoit de nouveau i+1 et donc i prend la valeur 11)
3- On affecte maintenant à la variable entière simple y la valeur de i qui est 11, (y=11)
4- Et enfin on décrémente la variable i de 1 et on aura i=10

De même on peut rencontrer des opérateurs pré-incrémentations et pré-décrémentations. Pour bien comprendre, reprenons l'exemple ci-dessus tout en changeant la position des opérateurs :

int i=10 ;

int x=++i ;

int y=--i ;

1- On affecte à la variable simple de type entier i la valeur 10
2- Avant d'affecter à la variable simple entière x la valeur de i, il faut incrémenter d'abord la variable i de1 (donc i reçoit 11)
3- puis affecter cette nouvelle valeur de i à la variable x et on aura x=11
4- de la même manière, effectuer la décrémentation de la variable i de 1 et donc on aura i=10
5- puis affecter cette nouvelle valeur de i à la variable y et on aura y=10

Remarque:

- Les opérateurs unaires ne s'appliquent pas à des expressions arithmétiques.
- Les opérateurs unaires sont prioritaires dans l'exécution par rapport aux opérateurs binaires seulement
- On peut aussi rencontrer une simplification d'écritures des expressions arithmétiques mais avec cette fois ci des opérateurs binaires de la manière suivante :
  X+=12 ; // cette expression veut dire qu'on ajoute à la variable X la valeur 12 et le résultat sera affecté a X. Elle est équivalente à écrire X=X+12 ;
  Même chose pour les autres opérateurs binaires.

Primitives conditionnelles

Les primitives conditionnelles d'actions en langage de programmation orienté objets que ce soit simple, à choix multiples ou répétitives restent inchangés et dans la forme d'écritures (syntaxe) comme dans la sémantique par rapport aux langages procéduraux.

Seulement, la primitive conditionnelle alternative (if - else - if), malgré sa syntaxe qui est idem à celle des autres langages classiques et que le C++ l'accepte vivement, alors dans

certain cas, l'effet d'une exécution alternative peut être obtenu plus directement en utilisant l'opérateur ternaire notée par les symboles '? :' sans écrire l'instruction (if – else –if). Cet opérateur implique trois opérandes dont le premier est obligatoirement de type booléen, alors que les deux autres peuvent être d'un type quelconque.

La forme d'écriture de cette primitive conditionnelle alternative est :

<Variable résultat>=<opérande1> **?** <opérande2> **:** <opérande3> **;**

Le mode opératoire de cette primitives est que, si l'opérande1 après son évaluation est vrai alors le résultat de cette primitive est Opérande2 et sera affecté à la Variable résultat. Dans le cas contraire, l'opérande3 sera affecté à la Variable résultat.

Page programme C++

Le squelette d'un programme écrit en langage non procédural C++doit obéir à la structure suivante :

Une page programme C++ est divisée en trois grandes parties

1- En tête du programme : il contient l'inclusion des bibliothèques des instructions à utiliser et les déclarations globales des structures de données
2- Création et définition des classes d'objets
3- Programme principale 'main'

Autrement dit, la page programme C++ à la forme suivante :

```
#include<iostream>

#include>stdio.h>

#include<math.h>

.

.

.

class= <nom de la classe>

{  //attributs

//les méthodes

} ;

int main( )

{ //déclaration globales
```

//appels aux méthodes de classes

return( ) ;

}

Exercices

Enoncé 1

Soit la suite définie par les termes suivants

U(0)=3, U(1)=2, U(n)=nU(n-1)+(n+1)U(n-2)+n

Ecrire la classe d'objet qui Calculer U(n)

Enoncé 2

Soit x une variable de type entier, Ecrire la classe d'objet qui calcule son carré

Enoncé 3

Pourquoi on écrit ''using namespace std'' dans un programme en C++ et qu'est ce que ça veut dire (quelle est son rôle)

Enoncé 4

Soit le système d'équation linéaire de la forme AX=B

$$\begin{cases} 3x_1 - x_2 + 2x_3 = -3 \\ x_1 + x_2 + x_3 = -4 \\ 2x_1 + x_2 - x_3 = -3 \end{cases}$$

Ecrire la classe d'objet qui permet de décrire les méthodes (procédures et fonctions) suivantes :

- lire la matrice A et le vecteur B associées au système linéaire donné
- calcule et imprime la solution du système d'équations linéaire Ax=B
- calcule la trace de la matrice A
- les valeurs propres de A ou bien le déterminant de la matrice A

Enoncé 5

Soit à intégrer numériquement l'équation différentielle du 1er ordre suivante

$\frac{dy}{dx} = \sqrt[3]{2(2x^3 - y^3) + 2.525e^{-x-y}}$ Avec les conditions initiales $x = 1 et\ y = 2$

Ecrire un programme en C++ (vous pouvez l'écrire en C) qui donne la solution (utiliser la méthode de RUNGE KUTTA vue en 3ème Année Licence).

Enoncé 6

Exécuter la séquence programme suivante s'il n y a pas d'erreurs et affiché le résultat final pour chaque variable

```
#include <math.h>

#include <iostream>

 int main( )

{int x=45 ;float y=x+10 ; if (y !=0) {int z=x/y ; float exp=z--+x*y ;

 float f=10*sqrt(z)+x++-y/z) ; cout<<y<<z<<f <<exp;} }
```

Solutions

Solution1

```
#include <iostream>

  class suite

{       private:

            int N, u, i, v, w; i=0;

         public:

        int calcul (paramètres formels utilisées pour faire les calculs de la suite)

        {  cout<< "donner la valeur de N'' ; cin>>N ; u=3 ; v=2 ;

        if (N==0) w=u ;

        else if (N==1) w=v ;

        else

                for (i=2;i<=N; i++)   {w=i*v + (i+1)*u + i; u=v; v=w;}

cout<<" u("<<N<<")= "<<w<<endl;

return(résultats);}  };
```

Solution 2

```
class carrée
{       private:
          float z;
         public:
        float lecarréedez ( float z)
        {float resultat = z*z;  return(resultat);}  };
```

Solution3

Pour ce qui est de la directive ''using namespace std''**, normalement vous l'avez vu au cours de l'enseignement de la matière :Outils de programmation langage C en 2ème année.**

Mais voilà brièvement ce que veut ça veut dire. Vous pourriez écrire du programme en C++ qui a une fonction par exemple appelée ***pgm ( )*** et il y a une autre bibliothèque disponible qui a également le même fonction ***pgm ( )***. Maintenant, le compilateur n'a aucun moyen de savoir à quelle version de la fonction ***pgm ( )*** vous faites référence dans votre programme, c'est pour cela qu'on écrit au début du programme ''using namespace std'' pour faire la distinction entre les noms des fonctions utilisées (celles que vous définissez), les noms de classe objets etc… avec les mêmes noms disponibles dans différentes bibliothèques.

Solution5

Rappel

Soit une équation différentielle du 1er ordre de la forme $\frac{dy}{dx} = f(x,y)$ avec les conditions initiales $x = x_0 \; et \; y = y_0$ .La solution générale de l'équation différentielle est obtenue par une méthode approchée, par exemple, la méthode de RUNGE KUTTA. Dans ce cas, la solution se présente sous une forme discrète : $y_{i+1} = y_i + \frac{(k_1 + 2k_2 + 2k_3 + k_4)}{6}$ avec

$$\begin{cases} k_1 = h * f(x_i, y_i) \\ k_2 = h * f(x_i + h/2, y_i + k_1/2) \\ k_3 = h * f(x_i + h/2, y_i + k_2/2) \\ k_4 = h * f(x_i + h, y_i + k_3) \end{cases}$$

Et h : le pas d'intégration constant

J'essaye d'écrire le programme en C, parce que si je l'écris en C++ j'en suis très sûr que vous n'allez pas comprendre la description de la classe objet.

Dans notre énoncé, on connait f(x,y), x_0,=1, y_0= 2, h=0.05 et supposons xFinal=10

```
#include<math.h>
#include<iostream>
#include<stdio.h>
#include<stlib.h>
 using namespace std;
int  f (float x, float y)
{ y=sqrt(2*(2 *x**3-y**3 ))+2.525*exp(-x-y) ;
Return (y) ; }
 int main( )
 {
float x0, y0, h, xFinal ;
{cout<<  ''methode Runge Kutta '' ;
 cin>>x0, y0, h, xFinal ;
 cout<<'' x0=''<< x0<<''y0=''<<y0<<''h=''<<h<<''xFinal=''<<xFinal ;
y= y0 ; x=x0 ; cout<<''x=''<<x<<''y=''<<y ;
 while (x < xfinal)
```

$$\{k_1 = h * f(x,y),\ k_2 = h * f\left(x + h/2,\ y + k_1/2\right);\ k_3 = h * f\left(x + h/2,\ y + k_2/2\right);$$

$$k_4 = h * f\left(x + h,\ y + k_3\right);$$

$$y = y + \frac{(k_1 + 2k_2 + 2k_3 + k_4)}{6};$$

```
x=x + h; cout<<''x=''<<x<<''y=''<<y ; }
 cout<<''fin de résolution'';
}
```

Solution 6

```
{int x=45 ; int y=x+10 ; if (y !=0) {int z=y/x ; float exp=z--+x*y ;
 float f=10*sqrt(z+4)+x++-y/(++z) ; cout<<x<<y<<z <<exp<<f;} }
```

Int x	Int y	Int z	Float exp	Float f
45	55	1	exp=z--+x*y= 1+45*55=2476.	10*2+x++-y/++z= 10*2+45-55/1=10
		z-- ; z=0		
x++=45+1		++z=1		
x=46	y=55	z=1	exp=2476.	f=10.

Corrigé type et barème de l'interrogation écrite

Exercice1

Soient les expressions arithmétiques suivantes :

$F1 = ++sqrt(x) - x; \quad F2 = x - (--y * x)++; \quad F3 = ++x--; \quad F4 = y--y; \quad F5 = (-y * x++) + x;$

Evaluer en C++ les expressions pour *x=25* et *y=10,* dans le cas ou une expression est fausse affecter la valeur 0 aux variables ; sinon afficher les résultats de $F_{i(i=1,2..,5}$, *x, y*

Solution1 (14 points)

Expression *i*	Variable *x*	Variable *y*	Variable F_i
i=1 F1	0	0	0
i=2 F2	0	0	0
i=3 F3	25		26
i=4 F4	0	0	0
i=5 F5	26	10	-225

Exercice2

Comment rendre une classe objet non protégée ?

Solution2 (3points)

Soit on écrit le mot réservé ' public' pour les Attributs et pour les Méthodes.

class nom **{public:** //on déclare les attributs

public: // on écrit les procédures et les fonctions**} ;**

Soit on écrit à la place du mot 'class' le mot réservé 'struct'

struct nom **{** //on déclare les attributs

// puis on écrit les procédures et les fonctions **} ;**

Exercice 3

Citer 03 choses qui existent en C++ et n'existe pas en d'autres langages de programmation non orientés objets (Pascal, C, ….)

Solution (3 points)

Parmi les choses les plus remarquables (vue en cours) que le langage orienté objets C++ se distingue des langages de programmation non objets sont :

- La protection des données (encapsulation)
- Héritage
- Polymorphisme
- Non procédural
- Etc.

Remarque : Tous le monde à eu 2 points en plus gratuit