cesi ÉCOLE D'INGÉNIEURS

PROGRAMMATION ORIENTEE OBJET

1--CORBEILLE

---

1 INTRODUCTION

Cette corbeille d'exercices a pour but de vous initier à l'utilisation des notions de base de la programmation orientée objet avec langage C++.

La première partie de cette corbeille vous accompagne dans la compréhension et création de classes, méthodes, attributs, constructeur et instanciation de classe statique et dynamique. La seconde partie sera consacrée à la réalisation de quelques exercices.

Table des matières

2 PARTIE 1 : CONCEPTS DE BASE

Jusqu'à maintenant vous avez utilisé des approches orientées « structures & fonctions » avec un découpage des applications en un ensemble de structures de données et de fonctions pour les manipuler.

L'approche objet considère comme indissociable les données et les méthodes pour les manipuler, Un ***objet peut être imaginé/représenté comme une boite noire implémentant une fonctionnalité et rendant un service.***

Classe

La classe décrit le modèle structurel d'un objet :

- Ensemble des **attributs** (ou champs ou données membres) décrivant sa structure,
- Ensemble des **méthodes** (ou opérations ou fonctions membres) qui lui sont applicables.

Une classe en C++ est une structure qui contient : des Fonctions et des Données/Variables.

Exemple :

```
class Voiture {
public :
   // Méthodes publiques
   Voiture ( int _annee );
   void demarrer ( int code );
   int lireVitesse ();
   void accelerer ();
   void freiner ();

private :
   // Attributs privées
   char immatriculation[6];
   char *type;
   int annee;
   float poids;

   // Méthode privée
   void erreur(char *message);
};
```

Exemple : Décomposition fichiers « .cpp » et « .h »

Fichier : « Nombre.h »

```
class Nombre{

 double valeur;

 Nombre(int _valeur) ;

 int getValue();

 void setValue(int _v);

};
```

Fichier : « Nombre.cpp »

```
#include "Nombre.h"

Nombre::Nombre(int _valeur) {
      valeur = (double)_valeur;
}

int Nombre::getValue(){
      return (int)valeur;
}

void Nombre::setValue(int _v){
      valeur = (double)_v;
}
```

Exercice : Classe ListData

Nous souhaitons développer une classe nommée « ListData » simplifiant les calculs liés aux vecteurs de données et assurant les services suivants :

1. Ajout de nouvelles données par l'utilisateur de la classe,
2. Lecture du nombre de données fournies par l'utilisateur,
3. Calcul de la somme des données fournies par l'utilisateur,
4. Calcul de la moyenne des données fournies par l'utilisateur,
5. Calcul de la valeur minimum contenue dans les données fournies, Calcul de la valeur maximum contenue dans les données fournies,
6. Connaitre l'espace de variation maximum entre les données fournies, cette méthode sera nommée variationMax( ) et retournera un entier

Hypothèse : L'utilisateur ne peut pas fournir plus de 1024 données. Chaque donnée est rentrée une par une dans le tableau (ajoutée à la fin).

Objet

Création d'instances à partir des classes :

- De façon similaire à une « struct » ou à une « union », le nom de la classe représente un nouveau type de donnée,
- On peut donc définir des variables de ce nouveau type; on dit alors ***que l'on crée des objets ou des instances de cette classe.***

Exemple :

```
Voiture car_1;  // une instance simple (statique)
Voiture *car_2; // un pointeur (non initialisé)
car_2 = new Voiture(2005);      // création (dynamique) d'une
                                // instance avec initialisation
Voiture compagnie_location[10]; // un tableau d'instances
```

Instanciation d'objets – Durée de vie

1. Déclaration sous forme de variable locale (statique) :
    a. L'objet a une durée de vie associée à la structure dans laquelle il est déclaré (identique à une variable classique).

```
void MaFonction( … ){
     Objet obj( );
     … … … … …
     … … … … …
} // Mort de l'objet
```

    b. Accès aux méthodes et aux attributs à l'aide du séparateur « . »

```
void MaFonction( … ){
     Objet obj( );
     obj.Methode( valeur );
     obj.attribut = 10;
}
```

2. Déclaration sous forme de pointeur (dynamique) :
    a. Durée de vie illimitée, il faut donc penser à la détruire
    b. Dans le cas contraire, des fuites mémoire vont apparaître.

```
Objet *obj;
obj = new
Objet(parametres);
… … … … …
delete obj; //Destruction
```

c. Accès aux méthodes et aux attributs à l'aide du séparateur « → »

```
Objet *obj;
obj = new
obj->Methode( valeur );
obj->attribut = 10;
delete obj; //Destruction
```

Exercice : Classe ListData (suite)

1. Utilisation statique de la classe :
   a. Écrivez un programme « main » permettant d'instancier statiquement votre objet.
   b. Ajouter 4 données à la liste (3,9,13, -1).
   c. Calculer la moyenne des données.
   d. Ajouter la donnée (0).
   e. Récupérez la valeur minimum présente dans la liste.

2. Utilisation dynamique de la classe :
   a. Écrivez un programme « main » permettant d'instancier dynamiquement votre objet.
   b. Ajouter 4 données à la liste (3,9,13, -1).
   c. Calculer la moyenne des données.
   d. Ajouter la donnée (0).
   Récupérez la valeur minimum présente dans la liste.

3. Utilisation multiple de la classe :
   a. Créez 2 objets de type « ListeData » (de manière dynamique)
   b. Ajouter 4 données à la liste « 1 » (3,9,13, -1).
   c. Ajouter 3 données à la liste « 2 » (-5,-2,-7).
   d. Calculer la moyenne des données des 2 listes.
   e. Comparez ces données et affichez à l'écran si elles sont identiques ou non ?

Constructeur

Pourquoi un constructeur ?

a. Les données membres d'une classe ne peuvent pas être initialisées à l'instanciation de l'objet,

b. Il faut donc prévoir une méthode d'initialisation des données qui sera invoquée lors de la création de l'objet en mémoire,

Il est nécessaire d'écrire un ou plusieurs constructeurs afin de permettre une initialisation correcte de l'objet (on peut/doit considérer plusieurs cas, par exemple : avec et sans paramètres).

Exemple :

```
class Somme{
      double valeur;

      Somme( ){
            valeur = 0;
      }

      void ajouter(double _valeur){
            valeur += _valeur;
      }

      double lireResultat( ){
            return valeur;
      }
};

int main( void ){
      Somme *s = new Somme();
      s->ajouter( 10 );
      delete s;
      return 1;
}
```

Exercice : Classe ListData (suite)

1. Écrivez le constructeur manquant de la classe ListData. Ce constructeur ne prend aucun paramètre (le tableau est dynamique : int* liste)
2. Écrivez un second constructeur à la classe, ce dernier prend comme paramètre la première valeur de la liste.

3 PARTIE 2 : APPLICATION

Exercice 1

1. Créez une classe ‘CLa’
2. Cette classe doit comporter une méthode publique ‘void afficher(void)
3. La méthode doit afficher ‘bonjour’ sur la sortie standard
4. Dans le « main » créez un objet o1 et un objet dynamique o2 ;
5. Allouez de la mémoire pour o2
6. Appelez afficher de o1
7. Appelez afficher de o2
8. Affichez l’adresse de o1
9. Affichez l’adresse de o2
10. Affichez l’adresse pointée par de o2
11. Affichez la taille en octets du type CLa
12. Affichez la taille en octets de o1
13. Affichez la taille en octets de o2
14. Libérez la mémoire pour o2

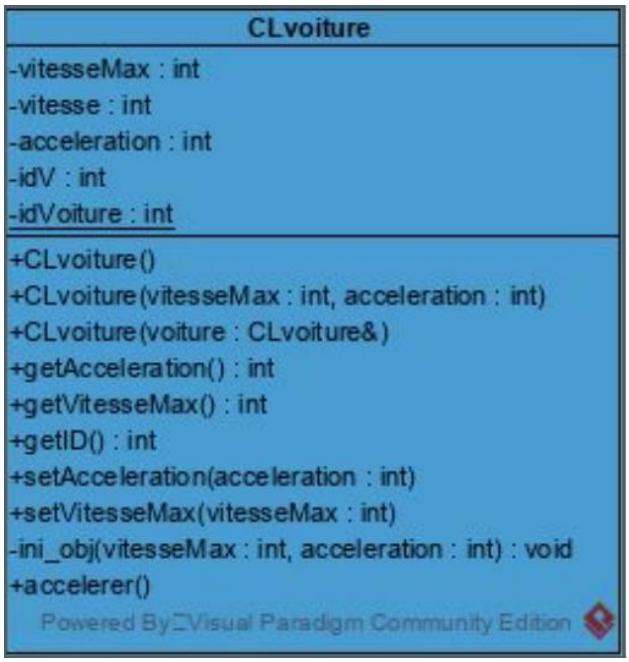
Exercice 2

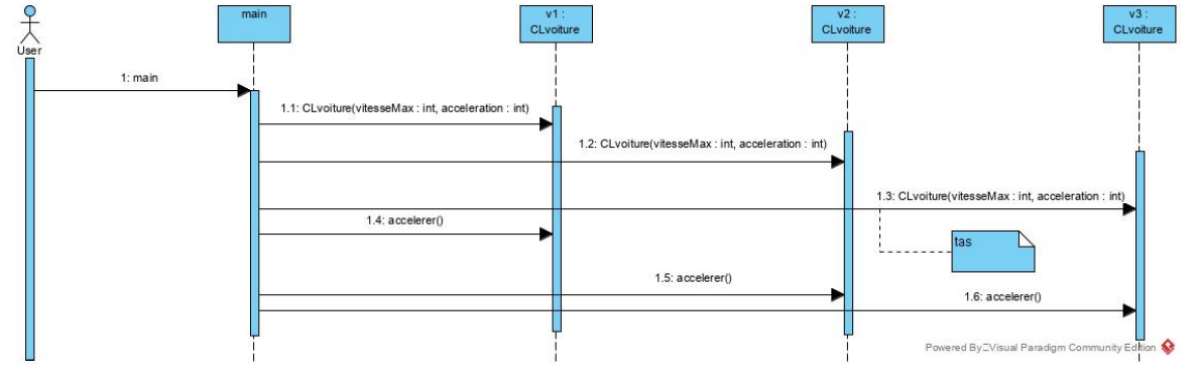
1. Créez une classe ‘CLobjet’
2. Créez les membres privés de type double : masse, vitesse, ec (énergie cinétique)
3. Créez les accesseurs, seul « ec » doit être en lecture seule.
4. La masse ne peut pas être nulle
5. La vitesse ne peut pas être nulle
6. Créez une méthode void calculer(void)
7. Cette méthode doit calculer l’énergie cinétique d’un objet
8. Le main doit comporter une méthode void comparer(CLobjet,CLobjet) qui doit comparer l’énergie cinétique de deux objets et qui déterminera et affichera quel est l’objet qui a la plus grande valeur cinétique
9. Utilisez le « main » pour créer des objets et comparer leurs valeurs.

Exercice 3

1. Reprenez l'exercice 2
2. Ajouter à la classe 'CLobjet' : un constructeur par défaut, un constructeur paramétré, un constructeur de recopie et une méthode privée d'initialisation de l'objet.
3. Dans le « main », créez une objet o1 sans argument. Créez une objet o2(2,2). Créez un objet dynamique o3. Créez un objet dynamique o4. Allouez de la mémoire pour o3(4,32). Allouez de la mémoire pour o4 en construisant cet objet avec l'objet o3. Calculez l'énergie de o1. Calculez l'énergie de o2. Comparez o1 et o2. Calculez l'énergie de o3. Calculez l'énergie de o4. Comparez o3 et o4. Libérez la mémoire pour o3. Affichez la masse et la vitesse de o4.

Exercice 4





PROGRAMMATION ORIENTEE OBJET

1 – CORBEILLE

1. Veuillez prendre connaissance du diagramme de classe et de séquence.
2. Vous devez coder le programme qui répond à ces deux diagrammes
3. Indications supplémentaires :
    a. Chaque voiture comporte un ID. Cet ID est généré automatiquement par une méthode statique.
    b. L'accélération et la vitesse ne peuvent être nulles.
    c. L'accélération au départ arrêtée. Le véhicule va accélérer jusqu'à atteindre sa vitesse maximale. L'affichage de la vitesse aura lieu tout au long de l'accélération. Lorsque le véhicule aura atteint sa vitesse maximale un coefficient de performance sera affiché (Vitesse maximale/Accélération).
4. Pour le « main » :
    a. Créez une voiture v1(20,1)
    b. Créez une voiture v2(300,10)
    c. Déclarez une voiture dynamique v3
    d. Allouez de la mémoire pour v3(160,2)
    e. Faites accélérer les voitures.