TD POO

BERTRAND Aurélien
L3 MIAGE
10 mars 2020

Sommaire

Introduction.

Ce rapport va servir de résumé aux différents TD de POO en L3 Miage de Bordeaux.
Il sert à montrer l'utilité du POO appliqué au C++.

La programmation orientée objet (POO), ou programmation par objet, est un paradigme de programmation informatique élaboré par les Norvégiens Ole-Johan Dahl et Kristen Nygaard au début des années 1960 et poursuivi par les travaux de l'Américain Alan Kay dans les années 1970. Il consiste en la définition et l'interaction de briques logicielles appelées objets ; un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre. Il possède une structure interne et un comportement, et il sait interagir avec ses pairs. Il s'agit donc de représenter ces objets et leurs relations ; l'interaction entre les objets via leurs relations permet de concevoir et réaliser les fonctionnalités attendues, de mieux résoudre le ou les problèmes. Dès lors, l'étape de modélisation revêt une importance majeure et nécessaire pour la POO. C'est elle qui permet de transcrire les éléments du réel sous forme virtuelle.

C++ est un langage de programmation compilé permettant la programmation sous de multiples paradigmes (comme la programmation procédurale, orientée objet ou générique). Ses bonnes performances, et sa compatibilité avec le C en font un des langages de programmation les plus utilisés dans les applications où la performance est critique.

Cahier des charges.

Les différents objectifs du TD sont les suivants :

- Savoir définir des constructeurs et destructeur dans une classe
- Comprendre le concept de surdéfinition
- Comprendre le fonctionnement et l'intérêt des fonctions en ligne
- Savoir définir une fonction amie

Le travail est aussi disponible sur Github : https://github.com/oreLINK/MIAGE-POO

Mise en oeuvre.

Exercice 1.

Créer une classe Personne représentée par un nom, un prénom et un âge. Le nom et le prénom seront représentés sous forme de tableau de 20 caractères et l'âge par un entier. On veut pouvoir créer des objets de type Personne soit en spécifiant le nom, le prénom et l'âge, soit en ne spécifiant rien, soit en spécifiant un objet Personne préexistant (constructeur de copie). Définir les constructeurs et destructeur associés. Afin de pouvoir tester la validité de votre classe, implémenter une fonction d'affichage : *void affiche() const.*

Dans un autre fichier. Faire une fonction `main` permettant de tester votre classe. Ce programme de test créera un tableau de Personne avec différentes initialisations, affichera l'ensemble des objets Personne à l'aide de la fonction *affiche()* et détruira le tableau.

Exercice 2.

Modifier la classe Personne en remplaçant les tableaux par des tableaux dynamiques. Modifier les constructeurs afin qu'ils allouent de la mémoire aux tableaux lors de la création d'un objet et le destructeur afin qu'il libère la mémoire allouée.

*Exercice 3.

Écrivez une classe nommée pile_entier permettant de gérer une pile d'entiers. Ces derniers seront conservés dans un tableau d'entiers alloué dynamiquement. La classe comportera les fonctions membres suivantes :

- *pile_entier (int n)* : constructeur allouant dynamiquement un emplacement de n entiers,
- *pile_entier ()* : constructeur allouant dynamiquement un emplacement de 20 entiers,
- *pile_entier (pile_entier &pile const)* : constructeur de copie créant une copie de la pile d'entier *pile*,
- *pile_entier ()* : destructeur
- *void empile (int p)* : empile l'entier p sur la pile,
- *int depile ()* : fournit la valeur de l'entier en haut de la pile et le supprime de la pile,
- *int pleine ()* : renvoie 1 si la pile est pleine (nombre d'entiers dans la pile = taille du tableau) et 0 sinon,
- *int vide ()* : renvoie 1 si la pile est vide et 0 sinon.

Exercice 4.

Quelles différences il-y-t-il entre les deux constantes MAX1 et MAX2 définies de la façon suivante ?

- #define MAX1 100
- static const int MAX2 = 100

Essayer d'accéder à l'adresse (pointeur) de MAX1 (int *p1=&MAX1) et à l'adresse de MAX2 (int *p2=&MAX2). Que se passe-t-il ?

Exercice 5.

1. Écrire un fonction affichant un entier : void affiche(const int& n). Que signifie le const dans le prototype de la fonction affiche ? Que se passe-t-il s'il ont tente de modifier l'entier n à l'intérieur de la fonction affiche ?
2. Rajouter dans la classe Personne une fonction membre d'affichage : "void affiche() const". Que se passe-t-il ont tente de modifier l'objet courant à l'intérieur de la fonction membre affiche (par exemple en utilisant this->age =0) ?

*Exercice 6.

Définir un pointeur sur un entier, nommé p (int *p), un pointeur sur un entier constant, nommé q (const int* q), un pointeur constant sur un entier, nommé r (int* const r). Faites des allocations de mémoire avec l'opérateur new. Que constatez-vous ? Définir un tableau de dix pointeurs sur des entiers. Faites une allocation de l'ensemble et une désallocation de mémoire. Vérifiez l'état de la mémoire. Définir une référence sur un entier, nommée s (int& s) puis une référence constante vers un entier, nommée t (const int& t). Essayez toutes les affectations entre p, q, r, s et t, et expliquez les résultats obtenus.

Exercice 7.

Quel est la différence entre les deux "fonctions" suivantes ?

- #define copie1(source,dest) source=dest ;
- inline void copie2(int source, int dest) { source=dest ; }

Exercice 8.

On souhaite donner le même nom à trois fonctions : "somme". La première additionne deux entiers (type int), la deuxième deux réels (type float) et la troisième deux tableaux de dix entiers. Donner le protoype de ces fonctions. Que se passe-t-il lorsqu'un appel est fait avec comme arguments deux short. Est-il possible de créer une fonction somme prenant 3 paramètres de même type ? Est-il possible de définir une fonction somme ayant deux paramètres de types différents (par exemple un int et un float) ?

Exercice 9.

Ecrivez une classe Vecteur3D comportant :

- trois données membre de type double x,y,z (privées)
- deux fonctions membres d'affichage :
  - *void affiche()* affichant le vecteur.
  - *void affiche(const char* string)* affichant string avant l'affichage du vecteur.
- deux constructeurs
  - l'un sans argument, initialisant chaque composante à 0.
  - l'autre, avec trois arguments correspondant aux coordonnées du vecteur.

Modifiez ensuite le code pour que les constructeurs soit des fonctions en ligne (inline).

Exercice 10.

Rajouter les fonctions suivantes à la classe Vecteur3D :

- Des fonctions permettant d'accéder au coordonnées d'un vecteur :
  - *int abscisse()* pour x,
  - *int ordonnée()* pour y et
  - *int cote()* pour z.
- Des fonctions permettant de modifier au coordonnées d'un vecteur :
  - *void fixer_ abscisse(int nouvelle_ abscisse)* pour x,
  - *void fixer_ ordonnée(int nouvelle_ ordonnée)* pour y et
  - *void fixer_ cote(int nouvelle_ ordonnée)* pour z.
- *bool coincide(Vecteur3D v)* qui renvoie true si v et l'objet courant ont les mêmes coordonnées, false sinon.

*Exercice 11.

Rajouter les fonctions suivantes à la classe Vecteur3D :

- double produit_scalaire(Vecteur3D v) qui calcule le produit scalaire de v avec l'objet courant.
- Vecteur3D somme(Vecteur3D v) qui calcule le vecteur somme de v avec l'objet courant.

Exercice 12.

Changer la fonction *coincide* le l'exercice 10 pour quelle soit symétrique. Utiliser le prototype suivant : *friend int coincide(Vecteur3D v1, Vecteur3D v2)*. Que signifie le mot-clé friend ?

Exercice 13.

On veut réaliser le produit d'une matrice avec un vecteur. Les classes Matrice et Vecteur sont définies de la manière suivante :

```
#define TAILLE 3

class Vecteur
{
  private:
    double vect[TAILLE];
  public:
    Vecteur(double t[TAILLE]){
    for(i=0;i<TAILLE;i++)
      vect[i]=t[i];
    }
}
```

```
class Matrice
{
  private:
    double mat[TAILLE][TAILLE];
  public:
    Matrice(double t[TAILLE][TAILLE]){
      int i,j;
      for(i=0;i<TAILLE;i++)
        for(j=0;j<TAILLE;j++)
          mat[i][j]=t[i][j];
    }
}
```

— Pourquoi les constructeurs sont-il implémentés dans la classe ? Quelle est la différence avec des constructeurs implémentés à l'extérieur de la classe ?

— Où peut-on déclarer la fonction *Vecteur produit(Matrice mat, Vecteur vect)* faisant le produit de mat par vect ? Comment faire pour que la fonction *produit* puisse accéder au données membres des objets mat et vect ?

— On souhaite implémenter une fonction *Vecteur produit(Vecteur vect)* dans la classe matrice. Comment faire pour que la fonction produit puisse accéder au données membres de l'objet vect ?

Tests et résultats.

Global.

La compilation de l'ensemble des codes se fait via le Makefile ci-dessous. En écrivant make dans le terminal du dossier concerné (chaque exercice à son propre dossier), on obtient un exécutable sutup. Il ne manque plus qu'à l'exécuter avec `./sutup` et le résultat s'affiche. Les tests ont été effectués avec des cout pour s'assurer de la bonne exécution du code.

```
1  CXX=g++
2  CXXFLAGS=-Wall -Wextra -g
3  LDFLAGS=
4  EXEC=sutup
5  SRC=$(wildcard *.cpp)
6  OBJ=$(SRC:.cpp=.o)
7
8  all: $(EXEC)
9
10 sutup: $(OBJ)
11     $(CXX) -o $@  $^ $(LDFLAGS)
12
13 .PHONY: clean mrproper
14
15 clean:
16     rm -rf *.o
17     rm -rf $(EXEC)
18
19 # mrproper: clean
20 #    rm -rf $(EXEC)
```

Exercice 4.

Ici, MAX1 est une macro et va être donc substitué partout où elle apparaîtra dans le code, les macros ne sont pas allouées dans la mémoire. A l'inverse MAX2 est une variable comme une autre que l'on définit ici à 100, allouée dans la mémoire.
Donc nous ne pouvons pas accéder à l'adresse de MAX1 étant donné que les macros ne sont pas alloués dans la mémoire :

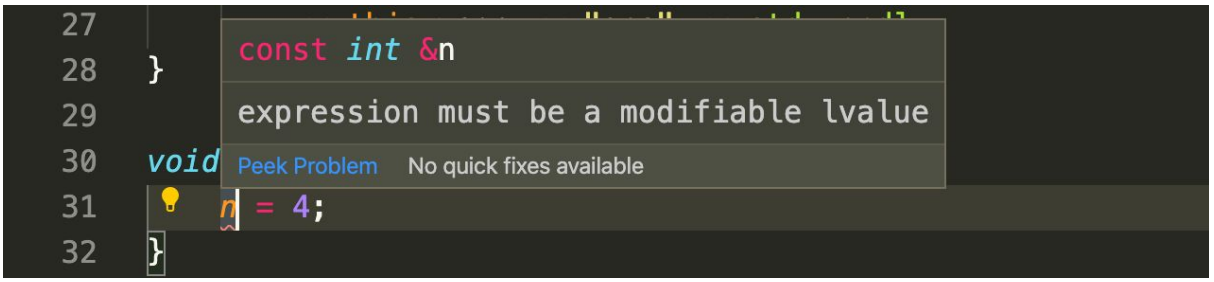
```
Main.cpp:7:15: error: cannot take the address of an rvalue of type 'int'
    int *p1 = &MAX1;
              ^~~~~
```

En revanche, en indiquant le `int` comme `const`, on obtient l'adresse de MAX2 :

```
→  MIAGE-POO/TD_3-4/Ex4 master ✗ ./sutup
Adresse MAX2 : 0x103113f4c
```

Exercice 5.

1. Le const écrit dans le prototype de la fonction implique que l'élément ainsi qualifié, ici n, ne doit pas être modifié pendant l'exécution du programme. Si on tente de le modifier, on reçoit un avertissement de l'IDE nous informant que le terme const est établi pour cet élément, et est donc, non modifiable.



Message d'avertissement de l'IDE VS Code.

Si cet avertissement n'est pas suivi, lors de la compilation du code avec le Makefile, on obtient une erreur comme quoi l'élément n'est pas modifiable.

```
Person.cpp:31:7: error: cannot assign to variable 'n' with const-qualified type 'const int &'
    n = 4;
    ~ ^
Person.cpp:30:25: note: variable 'n' declared const here
void affiche(const int& n) {
             ~~~~~~~~~~~^
```

Erreur lors de la compilation du code.

2. Comme on a placé le mot-clé const devant la fonction, le compilateur va automatiquement envoyé une erreur quand on tentera de modifier un élément dans ladite fonction. La même message d'avertissement s'affiche comme pour la question précédente.

```
Person.cpp:31:15: error: cannot assign to non-static data member within const member function 'affiche'
    this->age = 0;
    ~~~~~~~~~ ^
Person.cpp:30:14: note: member function 'Person::affiche' is declared const here
void Person::affiche() const {
~~~~~~~~~~~~~^~~~~~~~~~~~~~~
```

Erreur lors de la compilation du code.

Exercice 7.

La première est une fonction #define et est donc accessible partout dans le code. La deuxième est une fonction inline, très utilisée pour rendre le code plus rapide lors de la compilation et de son exécution.
La copie n'est réalisé que dans le cas de la macro (#define) car celle-ci n'alloue pas de mémoire comme vu à la question 4. La copie ne fonctionne pas pour la fonction inline,

étant donné que son but est d'optimiser le temps d'exécution et donc les différentes étapes du code.
Lorsque l'on appelle ces deux fonctions avec des rationnels de type double, pour la première fonction (#define) cela ne fonctionne pas. Cela fonctionne avec la deuxième fonction (inline).

```
Main.cpp:10:5: error: expression is not assignable
    copie1(443.55,8.675);
    ^      ~~~~~~
Main.cpp:3:35: note: expanded from macro 'copie1'
#define copie1(source,dest) source=dest;
                            ~~~~~~^
```

Erreur lors de la compilation du code pour la première fonction.

Nous avons la possibilité de faire des fonctions récursives en inline mais pas en macro (#define).

Exercice 8.

```
3   int somme(int a, int b);
4   float somme(float a, float b);
5   int somme(int *tab1, int *tab2);
```

Prototype des trois fonctions.

Un appel avec deux types short est correct et fonctionne.
Il est possible de créer une fonction avec 3 paramètres du même type (ici int).
Il est possible de définir une fonction qui additionne deux types différentes (ici int et float) ssi le type que retourne la fonction est celui qui a la plage de définition la plus grande (ici float). En effet, sans cela, nous n'aurions pas le résultat exact mais le résultat le plus proche possible de la solution qui respecte la plage de définition la plus petite (ici int).

Exercice 12.

Une fonction "friend" est une fonction qui, même si elle n'est pas membre de la classe, a accès aux données privées de cette classe

Conclusion et perspectives.

Pour conclure cette série de TDs sur la POO avec le langage C++, même si j'ai bien aimé faire les derniers exercices, je trouve que le projet arrive trop tard et qu'un rapport sur cette série de TD ne sera pas efficace pour ceux qui découvrent la POO et surtout le C++.
En effet, il aurait fallu commencer le projet dès le début et laisser ces TD comme support pour comprendre les demandes du projet (même si aujourd'hui on trouve toutes les réponses sur StackOverFlow). De même, il aurait fallu leur faire découvrir Github ou Gitlab qui leur seront très utiles en entreprise (commits, push, pull, branches, request…) pour rendre un travail.