Notions de base: Classes et Objets

Les exercices sont présentés grossièrement dans un ordre croissant de difficulté. Comme vous le constaterez très souvent en Programmation, un problème peut être résolu de différentes manières. Il peut donc plusieurs solutions à un exercice. Cependant, les meilleures solutions sont souvent les plus simples, et définitivement celles qui suivent la logique du cours et des travaux dirigés.

Les exercices marqués d'une étoile (*) sont réservés aux plus avancés sur les notions de base.

Objectifs

— Prise en main de l'environement de travail (Editeur + extensions de fichiers)
— Compréhension du processus de compilation en C++
— Types de Constructeurs, passage de paramètres
— Premiers pas en C++

Environnement de travail

- Créez un répertoire de travail `POO-MIAGE` où vous stockerez tous vos programmes de POO.
- Choisissez un éditeur de texte (`emacs` ou `vim`) avec lequel vous vous familiariserez peu à peu.

1 Hello World C++ et Entrée/Sortie

EXERCICE 1 : Soit le programme `hello.cpp` suivant :

```
#include <iostream>
using namespace std;

int main(){
    cout << "Hello world!" << endl;
    return 1;
}
```

Utilisation du compilateur *g++* pour compiler nos programmes :

1. Dans un terminal, compiler en executant la commande *g++ -c hello.cpp -o hello.o*
2. L'edition de liens se fera en executant la commande *g++ hello.o -o prog*
3. Executer votre premier programme : *./prog*

EXERCICE 2 : Modifiez le programme précédent pour qu'il demande à l'utilisateur son nom pour lui afficher une salutation personnaliséé (i.e, "Hello *name* !").

2 Quelques spécificités

2.1 Arguments par défault

EXERCICE 3 : Editez le programme ci-dessous dans un fichier exo3.cpp.
Discutez l'ajout d'un appel à la fonction ***fct()*** *sans arguments.*
Quelle modification pouvez-vous faire pour compiler le programme ?

```
#include <iostream>                                                    1
using namespace std;                                                   2
main()                                                                 3
{                                                                      4
    int n=10, p=20;                                                    5
    void fct(int, int=12); // proto avec une valeur par défaut         6
    fct(n,p);              // appel 'normal'                           7
    fct(n);                // appel avec un seul argument              8
}                                                                      9
                                                                       10
void fct (int a, int b)   // en-tête habituelle                       11
{                                                                      12
    cout << "premier argument : " << a << "\n";                        13
    cout << "second argument : " << b << "\n";                         14
}                                                                      15
                                                                       16
```

2.2 Notion de réference

Passez sur cet exercice si vous n'avez pas encore abordé cette notion en cours
EXERCICE 4 : Discutez la différence entre les trois fonctions d'échange de `echange.cpp`.

```
#include <iostream>                                                    1
using namespace std;                                                   2
main(){                                                                3
    void echange1(int, int);                                           4
    void echange2(int *, int *);                                       5
    void echange3(int &, int &);                                       6
    int n=10, p=20;                                                    7
    cout << "avant appel : " << n << " " << p << "\n";                 8
    echange1(n,p);                                                     9
    cout << "apres appel : " << n << " " << p << "\n";                 10
}                                                                      11
void echange1(int a, int b){                                           12
    int c;                                                             13
    c = a; a = b; b = c;                                               14
}                                                                      15
```

1. Donnez des implémentations pour les deux autres fonctions.
2. Compilez, exécutez et discutez les résultats.

3 Structures et classes

EXERCICE 5 : On se propose dans cet exercice d'écrire un programme permettant de créer un point p dont on affichera et modifiera les coordonnées.

1. Ecrire une structure `Point` qui contiendra les réels correspondant à l'abscisse et à l'ordonnée et d'un nom qui est un caractère de l'alphabet.
2. Ajouter des fonctions :
   — *initialise* pour attribuer des valeurs aux "coordonnées" d'un point.
   — *deplace* pour modifier les coordonnées d'un point.
   — *affiche* pour afficher les coordonnées d'un point.
3. Transformer la structure en classe, en tenant compte du masquage d'informations.
4. Effectuer une tentative d'utilisation directe des variables d'abscisse et d'ordonnée. (Changer les types de protection et re-compiler)

*EXERCICE 6 : Discuter la notion d'encapsulation au niveau de la classe.
Utiliser cette notion pour écrire un fonction *distant* calculant la distance entre deux points.

EXERCICE 7 : Répartissez le contenu du programme de l'exercice précédent entre les fichiers `Point.h` (la classe) et `Point.cpp` (les méthodes).

1. Désormais le champ `nom` de votre classe point est une chaine de caractères. Adaptez le programme.
2. Remplacer *initialise* par un constructeur initialisant par défaut l'ordonnée à 0.
3. Tester la déclaration '*Point p ;*' puis adapter au besoin la classe pour la compilation.
4. Ajouter un constructeur de copie et un destructeur à la classe `Point`

*EXERCICE 8 : On se propose d'ajouter des compteurs pour étudier le nombre d'appels aux différents constructeurs et au destructeur.

1. Ajouter à la classe `Point` des variables statiques privées *nb_ default nb_ copie nb_ param nd_ destruc* de type `int`, et des méthodes publiques permettant d'afficher ces variables pour des besoins de trace.
2. Initialiser ces variables et définissez les méthodes dans `Point.cpp`.
3. Modifier les constructeurs et le destructeur de maniere à ce que les variables définies précédemment contiennent respectivement le nombre d'appels effectués aux constructeurs par défaut, avec paramètre, de copie, et au destructeur
4. Vérifier le fonctionnement du programme en testant les appels suivants :

```
Point p(1.2,3);
Point p(2);
Point p;
```

5. Tester votre constructeur par défaut en exécutant :

```
Point p(1.2,3);
Point p(2);
```

6. Passez votre constructeur par copie en commentaire et réessayez le test précedent. Que doit-on en conclure ?

EXERCICE 9 : Compte en Banque !

Dans cet exercice vous créerez une classe `Compte` pour gérer le compte bancaire d'un client. On devra retrouver un certain nombre d'informations sur le compte, à savoir :

— le numero du client qui est unique et attribué automatiquement à la créeation du compte.
— le nom du client qui ne peut changer durant toute l'existence du compte.
— le solde !

1. Ecrire la classe `Compte` en donnant les méthodes usuelles d'utilisateur pour la consultation, le retrait et le depôt.
2. Créer une autre classe `Banque` pour gérer les comptes de différents clients :
   (a) Une Banque gère plusieurs comptes (tableau de comptes)
   (b) Une banque peut croître (ajout d'un compte à la liste de compte à gérer)
   (c) Un client peut quitter la banque (suppression d'un compte à partir de son numèro)
   (d) Un agent de banque vérifie des informations des comptes (Obtenir un compte à partir de son numèro)

*EXERCICE 10 : Donnez trois constructeurs pour la classe `Compte` avec 3 arguments, 2 arguments et 1 argument.

Grouper ces constructeurs en un seul.