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 y avoir 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'environnement 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

1. Créez un répertoire de travail `POO-MIAGE` où vous stockerez tous vos programmes de POO.Chaque TD devra avoir son propre répertoire comme par exemple `TD1-2`. Les exercices devront être nommés de manière explicite (`exo1` plutôt que `foobar`).
2. Choisissez un éditeur de texte (`emacs`, `vim` ou `geany`) avec lequel vous vous familiariserez peu à peu. Vous pouvez également utilisé un des environement de dévéloppement intégré comme XCode par exemple or Visual Studio.

EXERCICE 1 : Structuration du code

Le code d'un programme du C++ doit contenir

- le programme principal (main) dans un fichier .cpp
- les utilitaires de traitement - dans des fichiers .cpp à part
- les fichiers - entêtes pour les utilitaires de traitement - .hpp

Développer un exemple simple du programme qui permet de lire un nombre entier et d'afficher le nombre lu sur l'écran. La composition du programme :

- la fonction de lecture *int LireNombre()* ;
- le main qui appelle la fonction*LireNombre()*, pour cela utiliser le code suivant :

```
//Prise en main de processus de compilation
#include <iostream>
#include "Nombre.hpp"
using namespace std;

int main(){

  int a= LireNombre();
  cout<<"Nombre lu : " <<a<<endl;
  return 0;

  }
```

- le fichier Nombre.hpp. N'oubliez pas d'inclure les parenthèse de compilation conditionnelle du precompilateur *ifndef, define, endif*
- Pour compile -votre premier porgramme utiliser le compilateur en ligne de commande comme dans l'exemple :
  g++ -c Prog.cpp -o Prog.o - pour compilation uniquement
  g++ MainProg.cpp Prog.o -o TestProg - pou rla création de l'executable

0.1 Arguments par défaut

EXERCICE 2 : Editez le programme ci-dessous dans un fichier exo2.cpp.

Discutez l'ajout d'un appel à la fonction **fct()** sans arguments. Vous expliquerez dans votre compte rendu la solution utilisée.
Quelle modification pouvez-vous faire pour compiler le programme ?

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

0.2 Notion de réference

Passez sur cet exercice si vous n'avez pas encore abordé cette notion en cours
EXERCICE 3 : Discutez la différence entre les trois fonctions d'échange de `echange.cpp`.

```
#include <iostream>
using namespace std;
void echange1(int, int);
void echange2(int *, int *);
void echange3(int &, int &);
int main(int argc, char** argv){
     int n=10, p=20;
     cout<<"avant appel : "<<n<<" "<<p<<"\n";
     echange1(n,p) ;
     cout<<"apres appel : "<<n<<" "<<p<<"\n";
}
void echange1(int a, int b){
    int c;
    c=a;a=b;b=c;
}
```

1. Implémentez les deux autres fonctions.
2. Compilez, exécutez en testant à la suite les trois fonctions.
3. Notez et discutez la différence entre les arguments ainsi que leur influence sur les résultats.

1 Structures et classes

EXERCICE 4 : 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 5 : 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 6 : 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 7 : 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 tracer le nombre d'appels aux constructeurs et destructeurs.
2. Initialiser ces variables et définissez les méthodes dans `Point.cpp`.
3. Modifier les constructeurs et le destructeur de manière 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;
Point r = p;
Point r(p);
```

   Quel constructeur est appelé à chaque fois ? Pour les deux derniers appels, y a-t-il une différence entre les deux syntaxes ?
5. Passez votre constructeur par copie en commentaire et réessayez le test précédent. Que doit-on en conclure ?

EXERCICE 8 : 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 création 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)