CLASSES ET OBJETS

1 Objectifs du TD

1) Installation de l'environnement de travail de POO.
2) Compréhension du processus de compilation en C++.
3) Passage de paramètre par référence ou par copie
4) Ecriture de classes simples en C++.

2 Environnement de travail

Créez un répertoire nommé `C++` et déplacez-vous y. A partir de maintenant, tous vos programmes seront mis dans ce répertoire. Dans cette première partie, nous allons présenter les outils que nous utiliserons pour éditer, compiler et enfin exécuter nos programmes `C++`.

EXERCICE 1 – Sous `emacs`, éditez (puis sauvegardez) le programme `monprog.cpp` suivant :

```
#include <iostream>
int main(){
    cout << ''Hello world !'' <<endl;
    return 1;
}
```

Remarque – *L'outil pour compiler notre programme est le compilateur* `g++`.

1) *Compilez votre programme en exécutant dans un terminal la commande : g++ -c monprog.cpp -o monprog.o*
2) *Effectuez ensuite l'édition de liens en exécutant dans un terminal la commande : g++ obj1.o obj2.o ... monprog.o -o lancez_moi*
3) *Pour exécuter votre programme il suffit maintenant d'exécuter dans le terminal la commande : ./lancez_moi*

3 Éléments de syntaxe

EXERCICE 2 – Soit le programme suivant :

```
1  #include <iostream>
2
3   int main(){
4       int n = 10, p = 20;
5       void fct(int,int=12);
6       fct(n,p);
7       fct(n);
8       fct();
9       return 0;
10    }
11  void fct(int a,int b){
12      cout << ''premier argument : '' << a << endl;
13      cout << ''second argument  : '' << a << endl;
14  }
```

1) Quelle(s) ligne(s) risque(nt) de poser problème à la compilation ?
2) Que faut-il rajouter au programme pour que cette erreur ne se produise plus ?

3) Corrigez, compilez et testez votre programme.

EXERCICE 3 – Discutez la différence entre les fonctions suivantes :

```
void fct1(int x){
   x++;
}
void fct2(int &x){
   x++;
}
```

Lequel de ces deux appels sera refusé, `fct1(10)` ou bien `fct2(10)` ?

4 Structures et classes

EXERCICE 4 – Ecrire une structure `Point` formée de deux `double` (abscisse et ordonnée) et d'un nom, qui est un caractère.

Ecrire les fonctions :
- `init` qui permet d'initialiser un point avec une abscisse, une ordonnée et un nom ;
- `print` qui affiche les caractéristiques d'un point ;
- `translate` qui pratique la translation d'un point selon deux valeurs de type `double`.

Reprendre la définition du type `point` en transformant la structure en une classe.

EXERCICE 5 – Le but de cet exercice est d'écrire une classe `Compte` pour gérer les comptes bancaires d'un certain nombre de clients. Les informations suivantes sont disponibles sur ces comptes :
- un compte est défini par trois informations : le numéro du compte, le nom du client et le solde ;
- un numéro identifie de manière unique un compte. Il est attribué de manière automatique à la création d'un nouveau compte ;
- le nom du client ne change pas. Un même client peut avoir plusieurs comptes.

1) Écrire la classe `Compte` en donnant des méthodes de consultation du compte, de retrait et de dépôt.

EXERCICE 6 – On veut à présent écrire une classe `Banque` pour gérer les comptes des différents clients. Cette nouvelle classe définira par conséquent un tableau `comptes` contenant tous les comptes que l'on désire gérer. Cette classe devra définir les méthodes suivantes :

```
void add(Compte c);
// ajoute un compte à la liste des comptes à gérer
void delete(int numero);
// supprime le compte dont le numéro est donné par numero
Compte getCompte(int numero);
// retourne le compte dont le numéro est donné par numero
```

1) Écrire cette classe.

Remarque : en cas de besoin, vous pouvez rajouter d'autres méthodes dans la classe `Banque` et/ou dans la classe `Compte`.