Université Paris Diderot
Master 1

Langages à Objets Avancés
Année 2016-2017

TP n°1

Introduction à C++

Exercice 1 (Un programme Simple)

Créez un repertoire `ObjetsAvances/TP1/Calculette`, puis écrivez (dans un éditeur de votre choix) un programme `calculMental.cpp`. Le programme affichera deux entiers tirés au hasard et demandera à l'utilisateur d'en donner la somme tant qu'il n'a pas la réponse exacte. Il comptera le nombre d'échecs et l'indiquera à la sortie.
Indications :

- pour tirer un nombre au hasard :
```
#include <cstdlib>
#include <ctime>
...
srand(time(nullptr));
int x=rand();
```
- vous compilerez avec `g++ --std=c++11 -Wall` (en cas de Pb assurez vous de lancer celui qui est dans `/opt/csw/bin`)

Exercice 2 En c++ la compilation est séparée, de sorte qu'il est inutile de recompiler des codes qui n'ont pas été modifiés. Nous utiliserons l'outil "make" pour nous y retrouver. En annexe à ce TP vous trouverez une description d'autres environnements de travail que vous pourrez préférer plus tard, mais il vous faut de toutes façons savoir configurer un Makefile.

La paire (commande make / fichier Makefile) permet de faire une séries de compilations à l'aide d'une seule commande. Makefile est un fichier texte qui contient des règles de la forme :

```
cible : dependances
        commandes
```

les dépendances sont d'autres cibles, séparées par des espaces, ou bien un nom de fichier pour indiquer le cas où celui ci aurait été modifié. (Attention à la syntaxe : avant "commandes" il doit y avoir une tabulation).

Dans le répertoire qui contient le fichier Makefile, la commande `make cible` se chargera d'exécuter les commandes de votre cible, après avoir rafraîchit les dépendances si c'est nécessaire.

1. Dans le répertoire de votre calculette créez un fichier `Makefile` contenant :

```
CPP=g++ --std=c++11 -Wall

all :  calculette

calculette : calculette.o
     $(CPP) -o calculette calculette.o
calculette.o : calculette.cpp
     $(CPP) -c calculette.cpp
clean :
     rm *.o
```

puis à la console faites `make all` (ou simplement `make`).
Vous pourrez alors executer `./calculette`.

2. Faites ensuite `make clean` pour appliquer la règle qui efface les fichiers intermédiaires produits.

Remarques :
— en général lors de l'édition des liens il vous faut lister tout les fichiers d'extentions `.o` utilisés en les séparant par un espace.
— et lors de la compilation d'une classe, les dependances portent à la fois sur le fichier de description `.hpp` et sur le fichier de code `.cpp`

Exercice 3 (Classes)

Rappels sur les bonnes pratiques
— Un fichier de `déclarations` d'une classe est un fichier `MaClasse.hpp`, qui ne contient que les déclarations des attributs et les prototypes des méthodes ; la `définition`, c'est à dire le code des méthodes, sera mise dans un fichier `MaClasse.cpp`
— Pour éviter des problèmes de compilation dus à des inclusions multiples, commencez le fichier `MaClasse.hpp` par la directive de compilation suivante :

```
#ifndef MACLASSE
#define MACLASSE
```

Et finissez par

```
#endif
```

Mettez le nom après le `#define` en majuscule pour qu'il n'y ait pas d'ambiguïté avec le nom de la classe.

1. On cherche à écrire une classe qui modélise les points du plan par ses coordonnées. Ecrivez trois fichiers : `Point.hpp` de description, `Point.cpp` de réalisation, et `Test.cpp` où vous aurez un `main` pour tester la création d'un objet et son affichage. Pour cette question votre classe se réduira à :
— un constructeur à deux arguments,
— des accesseurs,
— des modifieurs,
— une méthode d'affichage,
Vous utiliserez un fichier Makefile pour effectuer la compilation.
Pensez au couple `#include <cstdlib>`, `using namespace std;`
Faites la distinction entre la syntaxe `#include <...>` et `#include "..."`.

2. Ajoutez une méthode `distance(Point)` qui retourne la distance entre deux points (indication : vous utiliserez `#include <cmath>`). Vérifiez en calculant la distance du point de coordonnées 1,1 à l'origine.
3. Ecrivez une classe triangleOrienté dont les 3 sommets s'appelleront respectivement a,b,c selon l'ordre dans lequel ils apparaissent au moment de la construction.
4. Testez son affichage, et écrivez une méthode qui calcule son périmètre.
5. On considère une variable p de type Point de coordonnées (1,0), et un triangle construit entre autres avec le point p passé en argument de son constructeur. Qu'arrive t'il au triangle après l'exécution de `p.setX(2);` ? Que pouvez vous dire de la nature du passage des paramètres aux méthodes ?
6. Lorsqu'un triangle a un périmètre trop grand, on souhaite le rapetisser de la facon suivante : soit XY le coté plus long d'un triangle de sommets X, Y, Z. On remplace X par le milieu M de XY, de sorte que le triangle soit à présent constitué des points M,Y et Z. Ecrivez une méthode `void rapetisse()` qui effectue cette manipulation, ainsi que la méthode intermédiaire `milieu`.

Environnements de travail

1. (avec emacs) Sous `emacs` en appuyant simultanément sur Alt et x, puis en complétant le mini-menu avec `compile`, vous pourrez ensuite lancer la commande `make` sans avoir à aller dans la console.
2. (avec codeblocks) Pour une compilation simple, en reprenant le premier exercice, lancez `codeblocs calculMental.cpp &` puis appuyez sur l'icone de compilation/exécution (flèche verte avec un engrenage). Vous pouvez aussi créer un nouveau projet, puis y creer de nouvelles classes. Reprenez votre second exercice comme si vous l'aviez développé avec Codeblocs. (Il est possible qu'il vous faille paramétrer le compilateur pour utiliser la version c++11)
3. (avec netbeans) Netbeans est également installé sur nos machines. Reprenez également le second exercice pour vous faire la main avec netbeans en créant un nouveau projet reprenant l'exercice 2.

A faire chez vous

Installez l'un des environnements C++ sur votre ordinateur personnel, assurez vous d'avoir le standard c++11.