

# TP n°1

## Introduction à C++

### 1 Prise en main

#### Exercice 1 (Un programme Simple)

Créez un repertoire `ObjetsAvances/TP1/Calcullette`, puis écrivez (dans un éditeur de votre choix) un programme `calcullette.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`

**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 série 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 repertoire qui contient le fichier **Makefile**, la commande **make cible** se chargera d'exécuter les commandes de votre cible, après avoir rafraîchi les dépendances si c'est nécessaire.

1. Dans le repertoire de votre calcullette 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 exécuter **./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 tous les fichiers d'extensions **.o** utilisés en les séparant par un espace.
- et lors de la compilation d'une classe, les dépendances 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 un point du plan par ses coordonnées. Écrivez 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 modificateurs,
  - une méthode d'affichage.

Vous utiliserez un fichier **Makefile** pour effectuer la compilation.

Pensez au couple **#include <iostream>**, **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.

## 2 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 `codebloccs calculette.cpp` & puis appuyez sur l'icone de compilation/exécution (flèche verte avec un engrenage). Vous pouvez aussi créer un nouveau projet, puis y créer de nouvelles classes. Reprenez votre second exercice comme si vous l'aviez développé avec Codebloccs. (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.

## 3 Exercices supplémentaires

**Exercice 4** On rappelle que le sinus de la librairie `cmath` est défini pour un angle exprimé en radians et que les valeurs des angles se calculent proportionnellement en fonction de l'unité :  $[0; 2\pi]$  en radians,  $[0; 360[$  en degrés,  $[0; 400[$  en grades.

Vous pouvez utiliser la constante `M_PI` de la librairie `cmath`.

1. Écrivez une fonction `sinus(double, char)` à deux arguments permettant de calculer le sinus d'un angle donné dans une unité parmi {degrés, radians, grades}.
2. Vous pouvez paramétrer votre fonction pour que la valeur par défaut soit le radian. Vérifiez en appelant `sin(M_PI/2)`.

**Exercice 5** (compatibilités de types)

Pour cet exercice, **n'utilisez pas** `using namespace std;`, qui peut induire un comportement inattendu.

1. Écrivez deux fonctions appelées `plus` permettant de calculer :
  - (a) la somme de deux `int` et renvoyant un `int`,
  - (b) la somme de deux `double` et renvoyant un `double`.Appelez la en passant : deux `int`, un `int` et un `short`, deux `float`, deux `double`, un `int` et un `double`. Que dit le compilateur ? Pourquoi ?
2. Écrivez deux fonctions appelées `somme` qui permettent de calculer la somme des éléments d'un tableau d'`int` pour l'une et de `double` pour l'autre. Appelez la fonction `somme` en passant : un tableau d'`int`, un tableau de `short`, un tableau de `double`. Que dit le compilateur ? pourquoi ?

## **A faire chez vous**

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