C++ 2A
TD 1

Ce TD introduit les bases de C++. Le site `http://en.cppreference.com/w/` contient une documentation du langage C++ et de sa bibliothèque.

1 Variables

1.1 Types

1.1.1 Entiers

À la différence de Java, les types entiers peuvent être signés ou non signés. Par exemple une variable de type **`char`** contient une valeur entière codée sur 8 bits comprise entre -128 et +127, alors qu'une variable de type **`unsigned char`** contient une valeur comprise entre 0 et 255. Le type **`bool`** est un type entier qui est en identique au type **`int`**.
Questions :

1. char c = 127; ++c;. Quelle valeur prend la variable c ?
2. int a = 15; int b = 4; int c = a / b; Quelle est la valeur de c ?
3. int a; Quelle est la valeur de a ?
4. int a = 5; int b = ++a;
5. int a = 5; int b = a++;
6. auto x = 3; Quel est le type de x ?
7. auto x = 3u; Quel est le type de x ?
8. auto x = 3ull ; Quel est le type de x ?

1.1.2 Flottants

Il existe 2 types flottants en C++ : **`float`** et **`double`**. **Attention, les nombres flottants ne sont pas les nombres réels !**. Les valeurs flottantes sont suffixées par la lettre `f`, par exemple `3.5f`, alors que les valeurs doubles ne sont pas suffixées, par exemple `3.5`.
Questions :

1. float f = 1.0f/3.0f ; Quelle est la valeur de f ?
2. Que change le fait de changer le type de `float` à `double` ?
3. int x = 1.5; Quelle valeur contient x ?
4. int x = 1.2; Quelle valeur contient x ?
5. int x = 1.7; Quelle valeur contient x ?
6. auto x = 1.5; Quel est le type de x ?
7. auto x = 1.5f; Quel est le type de x ?

1.1.3 Tableaux statiques

Les tableaux Java sont des tableaux dynamiques stockés dans le tas, il faut utiliser **`new`** pour les allouer. En C++, les tableaux peuvent être dynamiques (voir prochain TD) mais aussi statiques c'est à dire de taille fixe et stockés dans la pile. Leur taille n'est donc plus modifiable une fois les tableaux alloués :

```
int t0[5];  // tableau de 5 entiers
int t1[] = {1, 2, 3, 4, 5}; // tableau de 5 entiers
int t2[3] = { 2, 5 }; // tableau de 3 entiers dont les 2 premiers sont 2 et 5
std::array< float, 5 > t3 = { 1.0f, 2.0f, 3.0f, 4.0f, 5.0f }; // interface STL
std::array< float, 5 > t3{ { 1.0f, 2.0f, 3.0f, 4.0f, 5.0f } };
```

Questions :

1. int t0[5]; int x = t0[0]; x = ?

2. int t0[5]; int x = t0[10]; x = ?
3. int t0[5] = {}; int x = t0[0]; x = ?
4. std :: array< int, 3 > t0 { { 1, 2, 3 } }; auto x = t0.at(2); x = ?
5. std :: array< int, 3 > t0 { { 1, 2, 3 } }; auto x = t0.at(10); x = ?

1.1.4 Chaînes de caractères

Les chaînes de caractères sont manipulées à l'aide du type `string` qui se manipule de manière similaire au type `String` en Java. Voir `http://en.cppreference.com/w/cpp/string/basic_string` pour la liste des méthodes disponibles sur ce type.

```
string s0 = "Bonjour";
std::cout << s0 << std::endl;
std::cout << s0[ 0 ] << std::endl;  // On peut accéder à un caractère de la chaîne comme à un tableau.
```

1.2 Constantes

En Java, les constantes sont précédées du modificateur `final`, C++ utilise le modificateur **`const`** qui peut être placé à gauche ou à droite du type suivant le cas.

```
const int a = 3; // Lecture de droite à gauche : a est un int constant
int const a = 3; // idem mais 'à l'anglaise' : a is a const int
```

Questions :

1. int const a = 3; ++a; ?
2. int a = 5; int const b = a; ?

1.3 Portée

Les règles de portée des variables en C++ sont similaires à celles de Java.
En C++, une variable peut être également déclarée en dehors de toute fonction ou classe, elle est alors déclarée globale à tout le programme :

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

int gi = 42;

int main()
{
  cout << gi + 2 << endl;
}
```

On évite cependant d'utiliser les variables globales car cela rend le code difficile à débugger et à maintenir.

2 Structures conditionnelles et boucles

Les structures conditionnelles et les boucles s'écrivent de la même manière qu'en Java.

```
if(cond) {
  // code...
}
else {
  // autre code...
}
// ...
for(int i = 0 ; i < 10 ; ++i) {
  // ...
}
while(cond) {
  // ...
}
```

D'autres formes sont disponibles et seront vues dans un prochain TD.

1. Utiliser une boucle **for** pour afficher les valeurs entières de 0 à 10.
2. Créer un tableau de 10 entiers et afficher seulement les valeurs paires.

3 Procédures et fonctions

Contrairement à Java, il est possible en C++ de déclarer des procédures et des fonctions en dehors d'une classe. C'est le cas notamment pour la fonction principale `main`. En Java, tous les arguments sont passés par copie. **Attention, en Java une instance de type tableau ou de type non primitif est une adresse, c'est donc l'adresse qui est copiée, par le tableau ou l'objet pointé (il faut utiliser la méthode clone pour copier un objet).** En C++ il est possible de choisir si un argument est passé par copie ou par référence. Par exemple :

```
void fct( int a ); // passage par copie
int fct( int a ); // passage par copie, retour d'une valeur
void fct( int & a ); // passage par référence
```

1. Créer une procédure de prototype **void** `addOne( int i )` ajoutant 1 à la valeur passée en paramètre.
2. Que produit le code suivant :

   ```
   int i = 5;
   addOne( i );
   cout << i << endl;
   ```

3. Passer la variable `i` par référence et retester le code.
4. Reprendre les exercices de la section précédente sous forme de fonctions.
5. Écrire une fonction de tri d'un tableau.

4 Types complexes

La définition d'une classe en C++ est presque identique à celle du Java :

```
class A
{
private:
  int a;
  int b;
public:
  A() // constructeur
  {
    a = b = 0; // syntaxe non disponible en Java !
  }
  // on peut aussi écrire A() : a(0), b(0)

  void addOne() // méthode
  {
    a += 1;
    b += 1;
  }
};
```

Attention à ne pas oublier le ; après la définition de la classe

1. Ajouter un constructeur à 2 arguments afin d'initialiser `a` et `b`.
2. Ajouter des accesseurs pour modifier les champs `a` et `b`.
3. Ajouter une méthode `display()` affichant les valeurs contenues dans une instance de la classe.

4.1 Mots clés

En C++, les méthodes ne modifiant pas les valeurs contenues dans une instance peuvent (doivent !) être déclarées constantes en ajoutant le mot clé **`const`** après leur prototype.

1. Modifier le code précédent pour les méthodes concernées.
2. Essayer d'ajouter le mot clé **`const`** sur une méthode modifiant une des variables et recompiler. Quelle erreur est levée ?

De la même manière qu'en Java les variables et méthodes peuvent être précédées du mot clé **`static`** pour les attacher à la classe et non à l'instance.

4.2 Surcharge d'opérateurs

En C++, il est possible de surcharger les opérateurs de base, comme par exemple le +. Pour redéfinir le comportement de l'opérateur + pour la classe A il faut ajouter la définition suivante en dehors de la classe :

```
A operator+( A const & a0, A const & a1 )
{
  // ...
}
```

Les arguments sont passés par référence et l'ajout du mot clé **`const`** indique que les arguments ne peuvent être modifiés.
La liste des opérateurs disponibles en C++ est présentée à l'adresse `http://en.wikipedia.org/wiki/Operators_in_C_and_C++`.

1. Définir l'opérateur + entre 2 instances `a0` et `a1` de la classe `A`. L'instance retournée contiendra la somme des variables (`a = a0.a + a1.a` et `b = a0.b + a1.b`) des 2 instances.
2. Définir l'opérateur << pour pouvoir afficher le détail d'une instance de la classe `A` de la manière suivante :

   ```
   A a0(3, 54);
   cout << a0 << endl;
   ```