Fiche méthodologique de projet

T.4 Mettre en oeuvre des tests unitaires

Objectif

Tester unitairement des méthodes d'une classe.

Préambule

La meilleure façon d'exécuter des tests unitaires est d'utiliser une procédure automatique. En effet, les tests doivent pouvoir être exécutés sans intervention humaine et donner un résultat chiffré (par exemple 80 tests réussis sur 100).

Pour cela on va mettre en oeuvre le framework **CppUnit** (pour C++).

Un framework est une infrastructure logicielle qui facilite la conception des applications par l'utilisation de bibliothèques de classes ou de générateurs de programmes, soit dit en quelques mots : un cadre de développement.

Mise en oeuvre de cppunit

Récupérer l'archive cppunit-x.x.x.tar.gz sur le serveur ou sur le site du projet (http://sourceforge.net/projects/cppunit) et effectuer la procédure suivante :

```
$ su root
# cd /usr/local
# tar zxvf cppunit-1.12.0.tar.gz
# cd /usr/local/cppunit-1.12.0/
# ./configure
# make
# make check
# make install
```

Les répertoires d'installation importants sont :

```
# ll /usr/local/lib/libcppunit*
# ll /usr/local/include/cppunit/
```

Vérification post-installation :

```
# cat /etc/ld.so.conf | grep "/usr/local/lib"
/usr/local/lib
```

Présence du répertoire /usr/local/lib dans /etc/ld.so.conf sinon l'ajouter et faire un ldconfig :

```
# echo "/usr/local/lib" >> /etc/ld.so.conf
# ldconfig
```

Maintenant, vous pouvez quitter la session root :

```
# exit
$
```

Exemple

On va écrire une classe de test nommée **TestTriangle**.

Rappel : il est très important de strictement séparer le code du test du code à tester.

Fichier: TestTriangle.h

```
#ifndef CPP_UNIT_TESTTRIANGLE_H
#define CPP_UNIT_TESTTRIANGLE_H
#include <cppunit/extensions/HelperMacros.h>

class Triangle ; //la classe à tester

class TestTriangle : public CPPUNIT_NS::TestFixture
{ CPPUNIT_TEST_SUITE( TestTriangle );
  CPPUNIT_TEST( testTriangleEquilateral );
  CPPUNIT_TEST_SUITE_END();
public:
   TestTriangle();
   virtual ~TestTriangle();
protected:
  void testTriangleEquilateral();
};
#endif
```

Fichier: TestTriangle.cpp

```
#include <cppunit/config/SourcePrefix.h>
#include "TestTriangle.h"
#include "Triangle.h" // Classe à tester
CPPUNIT_TEST_SUITE_REGISTRATION( TestTriangle );
TestTriangle::TestTriangle() {}
TestTriangle::~TestTriangle() {}
//Classes valide n°1 : 3 valeurs positives égales
void TestTriangle::testTriangleEquilateral()
{
   Triangle triangle(2, 2, 2);
   CPPUNIT_ASSERT_EQUAL( Triangle::EQUILATERAL, triangle.isTriangle() );
}
```

Il ne reste plus qu'à écrire le programme de test.

Fichier : Main.cpp

```
#include <cppunit/BriefTestProgressListener.h>
#include <cppunit/CompilerOutputter.h>
#include <cppunit/extensions/TestFactoryRegistry.h>
#include <cppunit/TestResult.h>
#include <cppunit/TestResultCollector.h>
#include <cppunit/TestRunner.h>

int main( int argc, char* argv[] )
{
  // Create the event manager and test controller
  CPPUNIT_NS::TestResult controller;
  // Add a listener that colllects test result
  CPPUNIT_NS::TestResultCollector result;
  controller.addListener( &result );
  // Add a listener that print dots as test run.
  CPPUNIT_NS::BriefTestProgressListener progress;
  controller.addListener( &progress );
  // Add the top suite to the test runner
  CPPUNIT_NS::TestRunner runner;
  runner.addTest( CPPUNIT_NS::TestFactoryRegistry::getRegistry().makeTest() );
  runner.run( controller );
  // Print test in a compiler compatible format.
  CPPUNIT_NS::CompilerOutputter outputter( &result, CPPUNIT_NS::stdCOut() );
  outputter.write();
  return result.wasSuccessful() ? 0 : 1;
}
```

Résultats

On compile les différents fichiers sources et on exécute le programme de test :

```
$ g++ -g -O2 -c -I/usr/local/include/cppunit/ Triangle.cpp
$ g++ -g -O2 -c -I/usr/local/include/cppunit/ TestTriangle.cpp
$ g++ -g -O2 -c -I/usr/local/include/cppunit/ Main.cpp
$ g++ -g -O2 -o testTriangle -I/usr/local/include/cppunit/ -ldl -lcppunit
Triangle.o TestTriangle.o Main.o

$ ./testTriangle
TestTriangle::testTriangleEquilateral : OK
TestTriangle::testTriangleIsocele : OK
TestTriangle::testTriangleScalene : OK
TestTriangle::testTriangleInvalide1 : OK
TestTriangle::testTriangleInvalide2 : OK
TestTriangle::testTriangleInvalide3 : assertion
TestTriangle.cpp:65:Assertion
Test name: TestTriangle::testTriangleInvalide3
equality assertion failed
- Expected: 0
- Actual  : 1

Failures !!!
Run: 6   Failure total: 1   Failures: 1   Errors: 0
```

Remarque : le choix des classes de test est très important car on s'aperçoit ici que c'est le deuxième jeu de valeurs qui permet de détecter l'erreur.