Christophe PICHAUD
Consultant sur les technologies Microsoft
christophepichaud@hotmail.com | www.windowscpp.net

Les tests en C++

Les outils de tests pour le C++ ne sont pas récents et ils n'ont rien à envier à ceux des langages managés. Bien sûr, la notion de Reflection n'existe pas en C++ mais les fondamentaux existent depuis presque 20 ans ; depuis 2000 exactement. Pour parler testing, on va aussi parler IDE car certains outils sont couplés à Visual Studio, notre IDE préféré.

CPPUnit

Place au pionnier, CPPUnit : c'est le kit outil historique pour C/C++. Il a été forké plusieurs fois et c'est une valeur sûre. C'est le portage de JUnit, l'outil fait en Java.

Il est disponible ici : https://dev-www.libreoffice.org/src/cppunit/

CPPUnit n'est pas intégré à Visual Studio, mais il a l'avantage de marcher sur Linux et sur Windows. Son fonctionnement requiert de le compiler sous forme de lib ou de DLL et ensuite de lancer un Runner : un lanceur de tests de nos tests ! Compiler CPPUnit n'est pas très difficile, il suffit de charger la solution cppunit-1.13.2\src\CppUnitLibraries2010.sln qui est pour Visual Studio 2010 et de faire build sur les projets cppunit et cppunit_dll. Pour voir ce qu'est un programme de test, il suffit d'inclure cppunit-1.13.2\examples\simple\simple.vcxproj dans la solution. La solution requiert cppunitd.lib au link et c'est tout. Voyons à quoi ressemble l'exécution d'une série de tests en C++ : 1

On y trouve des succès et des échecs. On trouve aussi le numéro de la ligne où sont les erreurs. C'est simple et efficace. Maintenant, regardons au niveau du code comment on fait écrit des tests en C++. On commence par écrire une classe de test :

```
#include <cppunit/extensions/HelperMacros.h>

class ExampleTestCase : public CPPUNIT_NS::TestFixture
{
 CPPUNIT_TEST_SUITE( ExampleTestCase );
 CPPUNIT_TEST( example );
 CPPUNIT_TEST( anotherExample );
 CPPUNIT_TEST( testAdd );
 CPPUNIT_TEST( testEquals );
 CPPUNIT_TEST_SUITE_END();

protected:
 double m_value1;
 double m_value2;

public:
 void setUp();

protected:
 void example();
 void anotherExample();
 void testAdd();
 void testEquals();
};
```

La routine setUp() est lancée au démarrage et ici il n'y en a pas, mais la méthode tearDown() est exécutée à la fin. C'est la seule

```
Developer Command Prompt for VS 2017
D:\Dev\cppunit-1.13.2\examples\simple\Debug>simple
ExampleTestCase::example : assertion
ExampleTestCase::anotherExample : assertion
ExampleTestCase::testAdd : assertion
ExampleTestCase::testEquals : assertion
ExampleTestCase.cpp(8) : error : Assertion
Test name: ExampleTestCase::example
double equality assertion failed
- Expected: 1
- Actual  : 1.1
- Delta   : 0.05

ExampleTestCase.cpp(16) : error : Assertion
Test name: ExampleTestCase::anotherExample
assertion failed
- Expression: 1 == 2

ExampleTestCase.cpp(28) : error : Assertion
Test name: ExampleTestCase::testAdd
assertion failed
- Expression: result == 6.0

ExampleTestCase.cpp(45) : error : Assertion
Test name: ExampleTestCase::testEquals
equality assertion failed
- Expected: 12
- Actual  : 13

Failures !!!
Run: 4   Failure total: 4   Failures: 4   Errors: 0

D:\Dev\cppunit-1.13.2\examples\simple\Debug>
```

1

subtilité de cette classe. Pour le reste, on utilise les macros pour définir des tests.

```
#include <cppunit/config/SourcePrefix.h>
#include "ExampleTestCase.h"

CPPUNIT_TEST_SUITE_REGISTRATION( ExampleTestCase );

void ExampleTestCase::example()
{
 CPPUNIT_ASSERT_DOUBLES_EQUAL( 1.0, 1.1, 0.05 );
 CPPUNIT_ASSERT( 1 == 0 );
 CPPUNIT_ASSERT( 1 == 1 );
}


void ExampleTestCase::anotherExample()
{
 CPPUNIT_ASSERT (1 == 2);
}

void ExampleTestCase::setUp()
{
 m_value1 = 2.0;
 m_value2 = 3.0;
}

void ExampleTestCase::testAdd()
```

```
{
 double result = m_value1 + m_value2;
 CPPUNIT_ASSERT( result == 6.0 );
}


void ExampleTestCase::testEquals()
{
 long* l1 = new long(12);
 long* l2 = new long(12);

 CPPUNIT_ASSERT_EQUAL( 12, 12 );
 CPPUNIT_ASSERT_EQUAL( 12L, 12L );
 CPPUNIT_ASSERT_EQUAL( *l1, *l2 );

 delete l1;
 delete l2;

 CPPUNIT_ASSERT( 12L == 12L );
 CPPUNIT_ASSERT_EQUAL( 12, 13 );
 CPPUNIT_ASSERT_DOUBLES_EQUAL( 12.0, 11.99, 0.5 );
}
```

Comme vous pouvez le distinguer sur le code source, on utilise des macros pour déclarer ce qui nous intéresse. La mécanique est de faire appel à des routines (macros) de type ASSERT avec des conditions à remplir sinon le test est en erreur. Pour exécuter le test, il faut un host CPPUnit, dans le même module que vos tests :

```
#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()
{
 // 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;
}
```

Le code est toujours le même donc on en fait un code partagé et le tour est joué… Vous allez me dire que le résultat est un peu spartiate. En effet, on utilise Visual Studio et cela sert, bien que les tests soient visualisables dans la fenêtre de Tests du Visual… Examinons une autre suite de tests.

Google Test Adapter

Vous allez tiquer… Google fait un add-in pour VS ? Eh oui ! Et même qu'il est open-source sous GitHub. Ici : https://github.com/csolten-born/GoogleTestAdapter . Il est présent aussi sur le marketplace Visual Studio. Cet add-in tire parti de la fenêtre « Test Explorer » dans Visual Studio. Par contre, il faut compiler une solution comme suit. Ouvrez la solution googletest-master\googletest\msvc\2010\gtest.sln. Il y a plusieurs projets mais le plus simple pour comprendre la philosophie Google Tests c'est gtest_unittest-md. Il contient un fichier énorme avec plein de cas de tests. Pour faire la liaison avec VS, il faut compiler le projet et afficher la fenêtre Test Explorer. Voici à quoi cela ressemble : 2

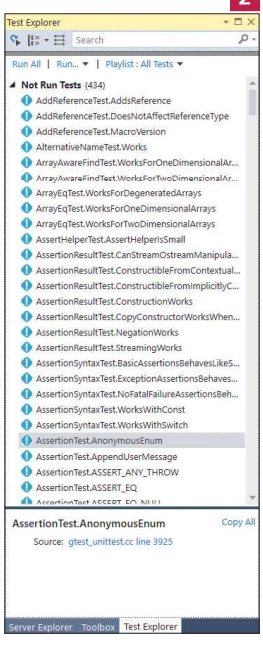
Au niveau du code, on va construire un ou plusieurs jeux de tests comme suit. Le fichier main ressemble à cela :

```
#include "gtest/gtest.h"

int main(int argc, char **argv) {
  ::testing::InitGoogleTest(&argc, argv);
  return RUN_ALL_TESTS();
}
```

On ajoute le fichier gtest-all.cc dans notre projet comme ça, le projet contient le hosting de Google Test. Créons une classe de tests dans un fichier MyTests.h :

```
#pragma once

#include "gtest/gtest.h"

class MyTests : public testing::Test
{
public:
  MyTests() {}
  ~MyTests() {}

public:
  int Add(int i, int j) { return i + j; }

  virtual void SetUp()
  {
     printf("SetUp()...\n");
  }
```

2



```
  virtual void TearDown()
  {
      printf("TearDown()...\n");
  }
};
```

Dans le fichier MyTests.cpp, il y a cela :

```
#include "MyTests.h"

TEST_F(MyTests, fn1) { printf("fn1\n"); }
TEST_F(MyTests, fn2) { printf("fn2\n"); }

TEST_F(MyTests, AddOK)
{
  EXPECT_EQ(10, Add(5, 5));
}

TEST_F(MyTests, AddFails)
{
  EXPECT_EQ(10, Add(15, 5));
}
```

Vous pouvez constater que le code est très simple. On teste deux fonctions et deux méthodes. A chaque fois, la fonction SetUp() est appelée et à la fin la fonction TearDown() aussi.
Si je compile ce projet et que je le Run, la fenêtre Test Explorer de Visual Studio affiche cela : 3
Si vous n'avez pas Visual Studio, Google Test peut se lancer en ligne de commandes : 4
Le fichier gtest.cc contient de nombreux tests et de nombreuses macros. Il est possible de jouer avec les exceptions, les paramètres d'entrée et de nombreuses autres choses. C'est immense comme suite de test.

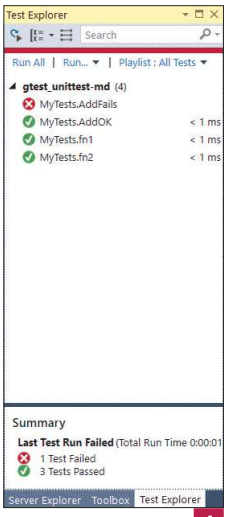
Boost.Test

La librairie Boost est particulièrement bien connue dans la communauté C++. De plus, elle possède une lib qui se nomme Test. La première chose à faire de builder Boost et ses librairies. Commençons par le début, il faut downloader Boost. Allez sur boost.org et téléchargez l'archive 7z pour Windows. Extrayez-la en local et ouvrez un VS command prompt. Tapez bootstrap.bat. Cela va compiler le tool de build. Ensuite tapez cela :

```
bjam toolset=msvc-14.1 variant=debug,release threading=multi
link=shared address-model=64
```

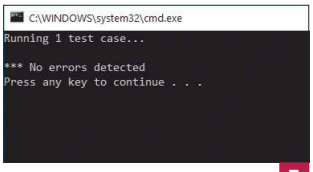
Cela prend un peu de temps et vous allez avoir dans le répertoire stage\lib toutes les libs… L'avantage de Boost, c'est que c'est bien fait et bien documenté. Le répertoire libs\test\example contient des jeux de tests avec différents types de ASSERT. Le principe est toujours le même, on fait des tests ou des suites de tests. Il est possible de faire des tests cases avec ou sans paramètres et sur des templates.
Voici comment faire un test minimal avec Boost.Test :

```
// ConsoleApplication1.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
#include <boost/test/included/unit_test.hpp>
using namespace boost::unit_test;

void free_test_function()
{
  BOOST_TEST(true /* test assertion */);
}

test_suite* init_unit_test_suite(int /*argc*/, char* /*argv*/[])
{
  framework::master_test_suite().
      add(BOOST_TEST_CASE(&free_test_function));

  return 0;
}
```



4

La fonction init_unit_test_suite est requise. Chaque Framework possède son main, nous l'avons vu.
Si on lance ce programme console, on obtient ça : 5



5

Intégration continue

Que ce soit CPPUnit ou Google Tests, le mode console retourne un code qui permet de savoir si c'est PASSED ou FAILED. Il est ainsi possible de faire tourner les TU dans une chaîne d'intégration continue.

Conclusion

Ecrire des tests unitaires en C/C++ est très simple. On utilise les macros ASSERT de son framework de test préféré et le tour est joué. Quand on fait des TU, l'intelligence est dans les TU et non dans le framework de tests. Il existe d'autres framework comme catch, etc… Mais le principe est toujours le même.
A vous de jouer ! •

3