Unit Testing in Cmajor

Seppo Laakko

June 21, 2015

1 Tutorial

Cmajor has a command line tool named **cmunit**, that eases testing Cmajor libraries. Assume that you have a library *Foo* that contains some functions and classes to be tested (Fig. 1).

Figure 1: Foo

```
using System;

namespace Foo
{
    public int Add(int x, int y)
    {
        return x + y;
    }

    public int Sub(int x, int y)
    {
        return x - y;
    }

    public class Cls
    {
        public int Twice(int x)
        {
            return 2 * x;
        }
        public int Square(int x)
        {
            return x * x;
        }

    }
}
```

First we create a project file for the test project:

```
project TestFoo;
target=program;
assembly <testfoo.cma>;
reference <../foo.cml>;
source <testfoo.cm>;
```

It references the library **foo.cml** to be tested and contains one source file **testfoo.cm** that will contain the tests.

Next we create that source file (Fig. 2).

Figure 2: testfoo.cm

```
using System;
using Foo;

namespace Tests
{
    public unit_test void TestAdd()
    {
        #assert(Add(1, 2) == 3);
        #assert(Add(0, 100) == 100);
        // ...
    }

    public unit_test void TestSub()
    {
        #assert(Sub(1, 2) == -1);
        // ...
    }

    public unit_test void TestCls()
    {
        Cls c;
        #assert(c.Twice(3) == 6);
        #assert(c.Square(2) == 4);
        // ...
    }

}
```

The source file contains functions that have a **unit_test** specifier and assert statements. The **unit_test** specifier tells **cmunit** program that the function is a unit test and not just an ordinary function. The asserts contain test expressions that should be true. The **cmunit** executes the asserts and counts how many of them succeeded and how many failed.

Next we run the tests using **cmunit**:

```
C:\Programming\cmajorbin\test\foo\testfoo>cmunit testfoo.cmp
testing project 'TestFoo' using llvm backend and debug configuration...
```

```
C:/Programming/cmajorbin/test/foo/testfoo/testfoo.cm
> Tests.TestAdd
< Tests.TestAdd: 2 assertions, 2 passed, 0 failed
> Tests.TestSub
< Tests.TestSub: 1 assertions, 1 passed, 0 failed
> Tests.TestCls
< Tests.TestCls: 2 assertions, 2 passed, 0 failed
C:/Programming/cmajorbin/test/foo/testfoo/testfoo.cm (ran 3 tests): 3 passed, 0
failed (0 did not compile, 0 exceptions, 0 crashed, 0 unknown results)
test results for project 'TestFoo' (ran 3 tests):
passed             : 3
failed:            : 0
  did not compile  : 0
  exceptions       : 0
  crashed          : 0
  unknown result   : 0

9 seconds
```

cmunit takes each unit test in turn and creates a program whose main() function calls the unit test. It then compiles the program. If the compilation fails, the test fails right away. Otherwise **cmunit** executes the program. Each assert is executed and the results for them are printed. If all asserts succeed the test program returns 0, if some assertion failed the test program returns 1. If the unit test threw an exception, the test program returns 2. If the program receives a segmentation violation signal, 255 is returned. **cmunit** records the return values and prints a report.

2 Reference

```
Cmajor release mode unit test engine version 1.0.0
Usage: cmunit [options] {file.cms | file.cmp}
Compile and run unit tests in solution file.cms or project file.cmp
options:
-config=debug   : use debug configuration (default)
-config=release : use release configuration
-backend=llvm   : use LLVM backend (default)
-backend=c      : use C backend
-file=FILE      : run only unit tests in file FILE
-test=TEST      : run only unit test TEST
-verbose        : print also passed assertions
-D SYMBOL       : define conditional compilation symbol SYMBOL
```

By default **cmunit** compiles and runs all tests in the given project or solution using debug configuration and LLVM backend settings. The configuration used can be given using the -config option and the backend used using the -backend option.

By using the -file=FILE option together with a project or solution compiles and runs only tests in the source file FILE.

By using the -test=TEST option together with a project or solution compiles and runs only given test named TEST.