C++: CONTINUOS INTEGRATION

JUDITH ESQUIVEL VÁZQUEZ

CODING STANDAR

http://www.sourceformat.com/coding-standard.htm

C/C++ Coding Style Standards

→ Buy Now

→ Download Trial

You can download the coding standards, coding style guides, code conventions, code guidelines, manuals and references for several general programming languages from here for free at your own risk. All trademarks, registered trademarks, product names and company names or logos mentioned herein are the property of their respective owners.

C/C++	Java	C#	Delphi/Pascal
PHP	ASP	Visual Basic/VBS	Python
Perl	JavaScript	Assembly	SQL

C/C++ Coding Style Standards

Adobe PDF Format

Possibility C++ Coding Standard	Todd Hoff	192 KB
Ellemtel C++ Coding Standard and Rules	Henricson & Nyquist	200 KB

CHECK CODING STANDAR AND PRETTY PRINT CODE

Name	Latest version	Programmed in	License	Comments
Clang-format	3.3	C++, Python	Open source	Promising tool under development that usesclang's abstract syntax tree. Presently formatting is fixed to 3 different standard formats. It does not provide options to customize format. See slides and video fromLLVM meeting april 2013.
Astyle (artistic style)	2.03 (April 2013)	C++	Open source	Supports C/C++, C# and Java. Good support for different options, but do not offer full freedom for customized format. Is maintained by only a few developers and have few maintenance releases (approx. one release every year).
Jindent		(most likely Java)	Commercial	Supports C/C++ and Java/SQLJ. Seems to support very customizeable formatting, see e.g. documentation.
Uncrustify			Open source	
PolyStyle			Commercial	
SQCBW				
GC!GreatCode				
Pork				

CHECK CODING STANDAR AND PRETTY PRINT CODE

Vera++	1.1.1	C++ with Boost library. Rules are written in Tcl	Open source (Boost Software License)	
Bcpp (C++ Beautifier)		C++		
Make pretty		Perl and Bash and uses Bcpp	Open Source (GPL)	
KWStyle			Commercial	
Universal Indent GUI	1.2.0 (January 2012)			Not itself a code layout tool, but a utility that presents a GUI for activating a code layout (beautifier/indenter/tidy) tools for different programming languages. Some of these listed in of this table.
cppcheck	1.59 (March 2013)	C++ with Qt	Open source	Is not really a code layout tool, but a static analysis tool.

VERA++

- Ensure that the source code complies with the given *coding standards and conventions*.
- Provide source code *metrics and statistics*.
- Perform automated *transformations* of the source code, which can range from *pretty-printing* to *diagnostics* to *fault injection* and advanced testing.
- Vera++ is a text only tool with no GUI, but it works smoothly with many graphical tools.: qtcreator, Microsoft Visual C++ 2010, Xcode, Cdash
- Rules are written in Tool Command Language

VERA++

boost_main.cpp – vera++ – Qt Creator

Projets

Interpreter.cpp
Interpreter.h
Parameters.cpp
Parameters.h
Profiles.cpp
Profiles.h
Reports.cpp
Reports.h
RootDirectory.cp
RootDirectory.h
Rules.cpp
Rules.h
Transformations
Transformations
structures
SourceFiles.cpp
SourceFiles.h
SourceLines.cpp
SourceLines.h
Tokens.cpp
Tokens.h
boost_main.cpp

boost_main.cpp — boost_main(int, char *[]): int

```
298                 return EXIT_FAILURE;
299             }
300             foreach (const std::string & r, rules)
301             {
302                 Vera::Plugins::Rules::executeRule(r);
303             }
304         }
305         else if (vm.count("transform"))
306         {
307             if(vm.count("profile"))
308             {
309                 std::cerr << "ERROR: --profile and --transform can't be used
at the same time." << std::endl;
310                 std::cerr << visibleOptions << std::endl;
311                 return EXIT_FAILURE;
312             }
313             Vera::Plugins::Transformations::executeTransformation(transform);
314         }
315         else
316         {
317             Vera::Plugins::Profiles::executeProfile(profile);
318         }
319
```

Documents ouverts
Interpreter.cpp
boost_main.cpp

Problèmes

T008: keyword 'if' not followed by a single space	boost_main.cpp	307
L004: line is longer than 100 characters /Users/glehmann/src/vera/src/boost_main.cpp	boost_main.cpp	309
L005: too many consecutive empty lines	boost_main.cpp	326
L005: too many consecutive empty lines	boost_main.cpp	348
L003: trailing empty line(s)	boost_main.cpp	350

```
vincent@hal:/tmp/vera++-1.2.1$ "/tmp/vera++-1.2.1/build/src/vera++" "--root" "/tmp/vera++
-1.2.1" "--rule" "L005" "/tmp/vera++-1.2.1/tests/L005.cpp"
/tmp/vera++-1.2.1/tests/L005.cpp:6: too many consecutive empty lines
```

CPPCHECK

Unlike C/C++ compilers and many other analysis tools it does not detect syntax errors in the code. Cppcheck primarily **detects the types of bugs that the compilers normally do not detect.** The goal is to detect only real errors in the code (i.e. have zero false positives).

Detect various kinds of bugs in your code.

- Out of bounds checking
- Memory leaks checking
- Detect possible null pointer dereferences
- Check for uninitialized variables
- Check for invalid usage of STL
- Checking exception safety
- Warn if obsolete or unsafe functions are used
- Warn about unused or redundant code
- Detect various suspicious code indicating bugs
- …

UNIT TEST C++

- CppUnit
- Boost.Test
- CppUnitLite
- NanoCppUnit
- Unit++
- CxxTest
- Unit Test ++

PruebaEstudiante.h

```
#ifndef PRUEBAESTUDIANTE_H_
#define PRUEBAESTUDIANTE_H_

#include <cppunit/extensions/HelperMacros.h>

#include "Estudiante.h"

class PruebaEstudiante : public CPPUNIT_NS::TestFixture
{
    CPPUNIT_TEST_SUITE( PruebaEstudiante );
    CPPUNIT_TEST( pruebaConstructor );
    CPPUNIT_TEST( pruebaAsignarCalificacion );
    CPPUNIT_TEST_SUITE_END();

    public:
        void setUp();
        void tearDown();

        void pruebaConstructor();
        void pruebaAsignarCalificacion();
};

#endif // PRUEBAESTUDIANTE_H_
```

PruebaEstudiante.cpp

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

#include "PruebaEstudiante.h"

#include <iostream>
#include <string>

void PruebaEstudiante::setUp(){}
void PruebaEstudiante::tearDown(){}

CPPUNIT_TEST_SUITE_REGISTRATION( PruebaEstudiante );

void PruebaEstudiante::pruebaConstructor() {
    Estudiante e(std::string("DNB"), std::string("13"))
    CPPUNIT_ASSERT("13" == e.getNumeroEstudiante());
    CPPUNIT_ASSERT("DNB" == e.getNombreEstudiante());
}

void PruebaEstudiante::pruebaAsignarCalificacion() {
    Estudiante e(std::string("DNB"), std::string("13"));
    e.asignarCalificacion(std::string("Matemáticas"), 7);
    e.asignarCalificacion(std::string("Historia"), 8);
    CPPUNIT_ASSERT(e.getCalificacion(std::string("Matemáticas")) ==
    CPPUNIT_ASSERT(e.getCalificacion(std::string("Historia")) == 8);
}
```

PruebaEstudianteMain.cpp

```
#include <cppunit/CompilerOutputter.h>
#include <cppunit/extensions/TestFactoryRegistry.h>
#include <cppunit/ui/text/TestRunner.h>

int main(int argc, char* argv[])
{
    CPPUNIT_NS::Test *suite = CPPUNIT_NS::TestFactoryRegistry::getRe

    CppUnit::TextUi::TestRunner runner;
    runner.addTest(suite);
    bool exito = runner.run();
    return exito ? 0 : 1;
}
```

Una vez implementadas nuestras pruebas, sólo nos falta pasárselas a nuestro programa. Para ello podemos usar un sencillo Makefile con el código de más abajo. En la primera línea se requiere la ruta de instalación de cppunit que puede variar en según que casos.

Makefile

```
CPPUNIT_PATH=/usr/include/cppunit/

PruebaEstudianteMain: PruebaEstudianteMain.o PruebaEstudiante.o Estu
    g++ -o PruebaEstudianteMain PruebaEstudianteMain.o PruebaEstudia

Curso.o: Curso.cpp Curso.h
    g++ -c Curso.cpp

Estudiante.o : Estudiante.cpp Estudiante.h
    g++ -c Estudiante.cpp

PruebaEstudiante.o: PruebaEstudiante.cpp PruebaEstudiante.h
    g++ -c PruebaEstudiante.cpp -I ${CPPUNIT_PATH}

PruebaEstudianteMain.o: PruebaEstudianteMain.cpp
    g++ -c PruebaEstudianteMain.cpp -I ${CPPUNIT_PATH}

clean:
    rm -f *.o PruebaEstudianteMain
```

DOCTEST

DocTest++

- Python allows doctests - tests that can be embedded within the documentation of a function. This has the advantage of seeing all the test case associated with a bit of code right then and there.
- DocTest++ bring this functionality to C/C++ code. Working in conjunction with UnitTest++, DocTest++ extracts fragments from the documentation in a header and source files and generates UnitTest++ specific test cases for them.

```
/**
* Adds two numbers.
* @test(TestAddition)
*     CHECK(3, add2(1, 2));
* @endtest
*/
int add2(int a, int b) { return a + b; }

UnitTest++.
TEST(TestAddition) { CHECK(3, add2(1, 2)); }
```

CODE COVERAGE

- 1. Testwell CTC++
- 2. CoverageMeter
- 2. CoverageMeter
- 4. GCT
- 5. CppUnit
- 6. Dynamic Code Coverage
- 7. TCAT C/C++
- 8. COVTOOL
- 9. gcov
- 10. xCover

JENKINS FOR C++: INSTALATION AND CONFIGURATION

- **DOxygen** :document generation
- **CppCheck** :source code inspection
- **CppUnit or Google Test** : unit test
- **gcov and gcovr** and the Cobertura plugin :code coverage.
- **gprof**: profiling report
- http://www.yolinux.com/TUTORIALS/Jenkins-Cpp-builds.html