Chapter 3

rev. 4

int team = 4; *11/14/2013*

INTRODUCTION TO FRAMEWORK

This testing suite will allow a user to create test cases that will be read and run through our testing harness. Templates are available to create your own test cases following our specified format. All test cases will be run through the component with the specified inputs, created into source code and run. The outputs are collected and inserted into a report with specific information about each test case.

DEPENDENCIES

You must check out the entire repository to run this testing suite. These are the required tools to be able to run this testing suite.

For Test Suite:

- Unix Based System
- BASH or BASH Compatible Shell
- g++
- cstdlib
- Python 2.x
- HTML5/CSS3 Compatible Browser
- Subversion

To Test Snort:

- C or C++
- Source Code

FILE STRUCTURE

This is the structure of the repository.

```
/TEAM4_TESTING (Note: No Spaces in any folder or file names)
/INT_TEAM_EQUALS_4
        /project
        /src
        /bin
        / …
        /scripts
                runAllTests
        /testCases
                testCase100.txt
                testCase101.txt
                ….
                testCase200.txt
                …
        /testCasesExecutables

        /temp #Emptied each time test is run
                testCase100results...
        /oracles
        /docs
                README.txt
        /reports
                testReport.html
```

ACQUIRING THE FRAMEWORK

Use username/password to access team4 subversion. Checkout all files from the SVN repository from the top level (/TEAM4).

Location of repository: https://svn.cs.cofc.edu/repos/CSCI362201302/team4

BUILDING SOURCE CODE

We have made the source code available to you in our repository. Typically one would extract a '.tar.gz' or 'tar.bz2' archive into the project folder. The source code should be contained in a './src/' subfolder inside the project folder.

ARCHITECTURE OF FRAMEWORK

//picture/desc

MAKING TEST CASE TEMPLATE

Reference the test case template located in './resources/testCaseTemplate.txt'.

testnumber
Should be same as file name. Replace template with this number when saving.
requirement
Requirement being tested by the test case.
component
Component being tested. Needs to match that component's file name (case sensitive).
method
Name of method being tested. Only include the name, no parameters (case sensitive).
testinputs
Arguments used to call the function.
oracle
Expected outcome.
linenumber
Line number method is found in the component. These are inclusive.
outputtype
Data type the function is expected to output. The return type should explicitly match the return type of the function being tested, as it will be used to instantiate a variable for that return type.
~DEPENDENCIES~
Required dependencies should be placed here. Anything located at this point will be directly inserted into the resulting C++ file for compilation including:

- Non-Local Variables
- Non-Local Objects
- Inherited Classes
- Instantiated Classes
- Included Interfaces
- Required Libraries
- Required #DEFINE

If they need to use a C library, they must list the equivalent C++ library.To find this, you may go here. Commit this to /testCase to be read by the script.

5 INITIAL TEST CASES

testCase101.txt

```
testNumber|101
requirement|Returns the correct value based on the</br>calculations
component|util.c
method|CalcPct
testInputs|0,0
oracle|0
lineNumber|139-155
outputType|double
~DEPENDENCIES~
#include <cstdint>
```

testCase102.txt

```
testNumber|102
requirement|Returns the correct value based on the</br>calculations
component|util.c
method|CalcPct
testInputs|1,1
oracle|100
lineNumber|139-155
outputType|double
~DEPENDENCIES~
#include <cstdint>
```

testCase103.txt

```
testNumber|103
requirement|Returns the correct value based on the</br>calculations
component|util.c
method|CalcPct
testInputs|0.5,0.5
oracle|0
lineNumber|139-155
outputType|double
~DEPENDENCIES~
#include <cstdint>
```

testCase104.txt

```
testNumber|104
requirement|Returns the correct value based on the</br>calculations
component|util.c
method|CalcPct
testInputs|1,2
oracle|50
lineNumber|139-155
outputType|double
~DEPENDENCIES~
#include <cstdint>
```

testCase105.txt

```
testNumber|105
requirement|Returns the correct value based on the</br>calculations
component|util.c
method|CalcPct
testInputs|2,1
oracle|200
lineNumber|139-155
outputType|double
~DEPENDENCIES~
#include <cstdint>
```

RUNNING TESTING SUITE

Make sure to update from the repository before running the testing suite. The script for the testing suite is located under './scripts/'. From the terminal, you can type './runAllTests'.

The script can also be run from your local version of the repository in your file browser. You can click on the file called 'runAllTests' under the 'scripts' folder. A message will appear, allowing you to either 'run' or 'run in terminal'.

Running the script will create the report document, which will pop up in an internet browser window.

C++ CODE CREATED

Units are tested through direct compilation of the function. The function is extracted directly from the file being tested and is inserted into a C++ template. Any dependencies are inserted before the declaration of main in the template file and if an output is specified, the cout statement is replaced with the user defined code. This file is compiled for testing.

INTERPRETING REPORT

Test Case
Name of the test case being run.
Method
The function or method being tested.
Inputs
The inputs passed to the function or method being tested.
Oracle
The expected output of the tested function or method.
Output
The actual output of the tested function or method.
Results
Pass if actual output matches expected output. Else, fail.
Requirement

//screenshot of an example of the report

EXPERIENCE DURING THE STEP

Analyzing source code to determine which methods to test:
Most files made heavy use of structs which complicated testing. To solve this problem, the testing script was designed to insert a user defined portion of code which could be used to create dummy objects or instantiate structs. *util.c* is the file where we will probably find all tested methods for this project due to it's simplicity in dependencies.

Constructing test cases for methods by creating applicable oracles:
Robert analyzed the code and determined some test partitions and developed test cases that cover partitions. We figured out which test cases we needed to satisfy requirements and what their expected outcomes would be. Once the test cases were determined and we had developed the test case template, Jeremy was able to write the test case text files. It was really only a matter of changing two inputs in each file, as well as fix the requirements for each test case, and these typically were the same for each method. We learned a little more about subversion when building these test cases, as we learned that if you change the file name, you cannot simply 'commit'. You must use the 'move' command.

Build and design automated testing framework:
Initially, during development of the script, we ran into a significant issue. The step

which compiled the tested code was failing with the complaint that g++ was unable to find main(int argc, char **argv). This left Evan confused, as the error did not appear to fit the context of any known problem. After considerable debugging and thinking, the conclusion was reached that the file needed to be closed. This solved the problem and the remainder of the script was smooth sailing.

Documenting results in an HTML report:

We made many changes to our original design throughout the process. Evan and Lynn worked to create the CSS, HTML and Javascript behind the design. With two visions as to where the design should go, we had to make a decision as to how the final report would look. Davida developed a logo to use as our team logo, which we displayed on the report.