Resources: Understanding the output of Unit Tests

I often have questions regarding interpretation of unit test output.

Here is the complete output from a run:

```
$ make test
./unittest.out
Running main() from src/gtest_main.cc
[==========] Running 22 tests from 4 test suites.
[----------] Global test environment set-up.
[----------] 5 tests from getStringTest
[ RUN ] getStringTest.displaysPrompt
[ OK ] getStringTest.displaysPrompt (0 ms)
[ RUN ] getStringTest.displaysDifferentPrompt
[ OK ] getStringTest.displaysDifferentPrompt (0 ms)
[ RUN ] getStringTest.readsWord
[ OK ] getStringTest.readsWord (0 ms)
[ RUN ] getStringTest.readsOneWord
[ OK ] getStringTest.readsOneWord (0 ms)
[ RUN ] getStringTest.readsAnotherSingleWord
[ OK ] getStringTest.readsAnotherSingleWord (0 ms)
[----------] 5 tests from getStringTest (0 ms total)

[----------] 5 tests from getIntegerTest
[ RUN ] getIntegerTest.displaysPrompt
[ OK ] getIntegerTest.displaysPrompt (0 ms)
[ RUN ] getIntegerTest.displaysDifferentPrompt
[ OK ] getIntegerTest.displaysDifferentPrompt (1 ms)
[ RUN ] getIntegerTest.readsInteger
[ OK ] getIntegerTest.readsInteger (0 ms)
[ RUN ] getIntegerTest.readsOneInteger
[ OK ] getIntegerTest.readsOneInteger (0 ms)
[ RUN ] getIntegerTest.readsAnotherSingleInteger
[ OK ] getIntegerTest.readsAnotherSingleInteger (0 ms)
[----------] 5 tests from getIntegerTest (1 ms total)

[----------] 5 tests from getDoubleTest
[ RUN ] getDoubleTest.displaysPrompt
[ OK ] getDoubleTest.displaysPrompt (0 ms)
[ RUN ] getDoubleTest.displaysDifferentPrompt
[ OK ] getDoubleTest.displaysDifferentPrompt (0 ms)
[ RUN ] getDoubleTest.readsDouble
[ OK ] getDoubleTest.readsDouble (0 ms)
[ RUN ] getDoubleTest.readsOneDouble
[ OK ] getDoubleTest.readsOneDouble (0 ms)
[ RUN ] getDoubleTest.readsAnotherSingleDouble
[ OK ] getDoubleTest.readsAnotherSingleDouble (0 ms)
[----------] 5 tests from getDoubleTest (0 ms total)

[----------] 7 tests from assignment1Test
[ RUN ] assignment1Test.displaysPrompts
[ OK ] assignment1Test.displaysPrompts (0 ms)
[ RUN ] assignment1Test.givesReturnValue
[ OK ] assignment1Test.givesReturnValue (0 ms)
[ RUN ] assignment1Test.displaysOneLoop
tests/test_01_04_assignment1.cpp:73: Failure
Expected equality of these values:
1
mResponse
Which is: 3
[ FAILED ] assignment1Test.displaysOneLoop (0 ms)
[ RUN ] assignment1Test.displaysTwoLoops
[ OK ] assignment1Test.displaysTwoLoops (0 ms)
[ RUN ] assignment1Test.displaysManyLoops
tests/test_01_04_assignment1.cpp:113: Failure
Expected equality of these values:
100
mResponse
Which is: -3
[ FAILED ] assignment1Test.displaysManyLoops (0 ms)
[ RUN ] assignment1Test.displaysZeroLoops
```

```
tests/test_01_04_assignment1.cpp:130: Failure
Expected equality of these values:
0
mResponse
Which is: 3
[ FAILED ] assignment1Test.displaysZeroLoops (0 ms)
[ RUN ] assignment1Test.displaysNoNegativeLoops
tests/test_01_04_assignment1.cpp:147: Failure
Expected equality of these values:
-1
mResponse
Which is: -7
[ FAILED ] assignment1Test.displaysNoNegativeLoops (0 ms)
[----------] 7 tests from assignment1Test (0 ms total)

[----------] Global test environment tear-down
[==========] 22 tests from 4 test suites ran. (1 ms total)
[ PASSED ] 18 tests.
[ FAILED ] 4 tests, listed below:
[ FAILED ] assignment1Test.displaysOneLoop
[ FAILED ] assignment1Test.displaysManyLoops
[ FAILED ] assignment1Test.displaysZeroLoops
[ FAILED ] assignment1Test.displaysNoNegativeLoops

4 FAILED TESTS
Makefile:18: recipe for target 'test' failed
make: *** [test] Error 1
```

Now, let's talk about how to digest it. First, the end of the report is a summary:

```
[==========] 22 tests from 4 test suites ran. (1 ms total)
[ PASSED ] 18 tests.
[ FAILED ] 4 tests, listed below:
[ FAILED ] assignment1Test.displaysOneLoop
[ FAILED ] assignment1Test.displaysManyLoops
[ FAILED ] assignment1Test.displaysZeroLoops
[ FAILED ] assignment1Test.displaysNoNegativeLoops

4 FAILED TESTS
```

So, there were 4 kinds of tests that failed: displaysOneLoop, displaysManyLoops, etc. When the summary is listed, you should note the first failed test and find details about that test.

In this case displaysOneLoop was the first to fail. To get details, scroll back up to find the details on this test. Here's the detailed information for this test:

```
[ RUN ] assignment1Test.displaysOneLoop
tests/test_01_04_assignment1.cpp:73: Failure
Expected equality of these values:
1
mResponse
Which is: 3
[ FAILED ] assignment1Test.displaysOneLoop (0 ms)
```

Note that you can look at the actual unit test code.
It's located in tests/test_01_04_assignment1.cpp, on line 73. Here's the code for that test:

```
62 TEST_F( assignment1Test, displaysOneLoop ) {
63
64 /* Setup */
65 mInputStream.str( "Blue 1 3.14" );
66 mOutput = "1 Blue 3.14\n";
67
68 /* Stimulus */
69 mResponse = assignment1( mInputStream, mOutputStream );
70
71 /* Comparison */
72 EXPECT_EQ( mPrompts + mOutput, mOutputStream.str( ) );
73 EXPECT_EQ( 1, mResponse );
74
75 /* Tear-down */
76 // Done automatically
```

```
77 }
```

Also note that the failed test message tells us the test expected a 1, but it saw a 3. If you look at line 73 of the code, you can see the 1 was hard coded into the test, and the variable mResponse is set to the return value of assignment1() (on line 69).

I think we can read from this that assignment1() isn't returning the expected value.

That's how I would interpret the failed test.

From here, it remains to figure out why assignment1() should be returning a 1 in this case, especially since there was an earlier test that wanted it to return 3. I suggest re-reading the assignment description for the assignment1() function.