CEC Continuing Education Center

DesignNews

Test Automation Design for Embedded Systems

DAY 4 : Unit-Testing Using Test-Driven Development (TDD) Part 2

Sponsored by

DigiKey

BENINGO EMBEDDED GROUP

informa markets

Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Group Chat’ by maximizing the chat widget in your dock.

THE SPEAKER

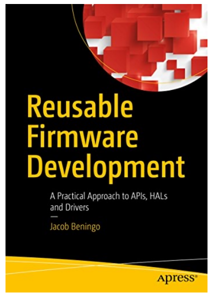
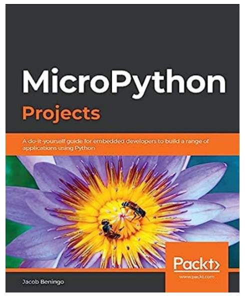
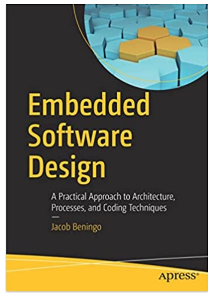
Jacob Beningo

Jacob@beningo.com

Beningo Embedded Group – CEO / Founder

Focus: Embedded Software Consulting and Training

Help teams deliver higher-quality embedded software faster. We specialize in creating and promoting embedded software excellence in businesses around the world.

Blogs for:

- DesignNews.com
- Embedded.com
- EmbeddedRelated.com
- MLRelated.com

Visit www.beningo.com to learn more

01

Do you have enough tests?

Modern Embedded Software Design Principle #4

Principle #4 – Practical, Not Perfect

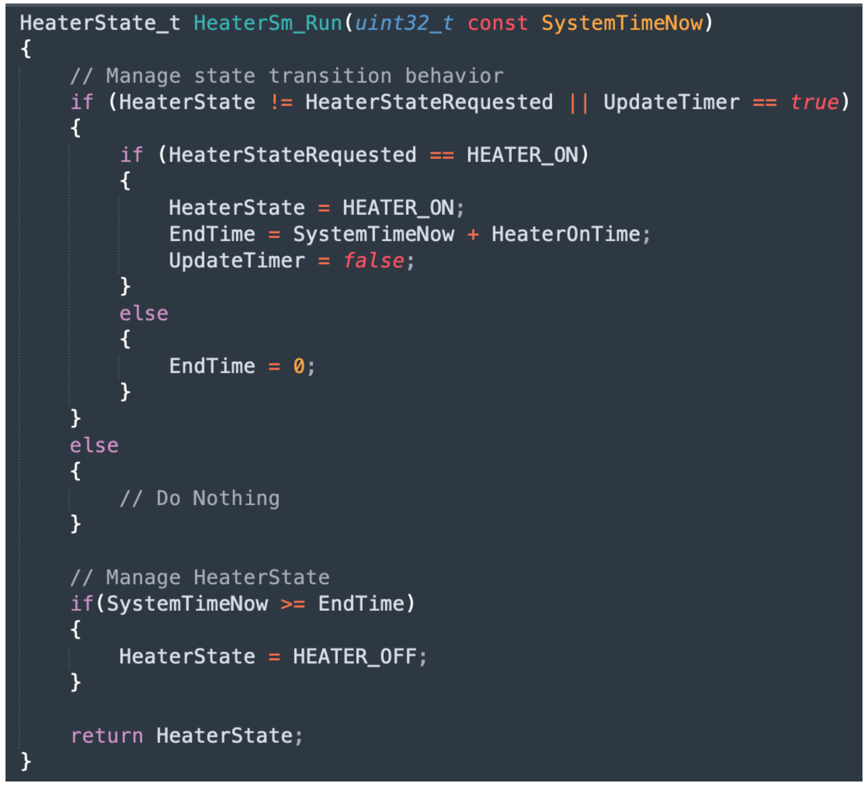
There is no such thing as a perfect system!

Focus not on perfection, but on:

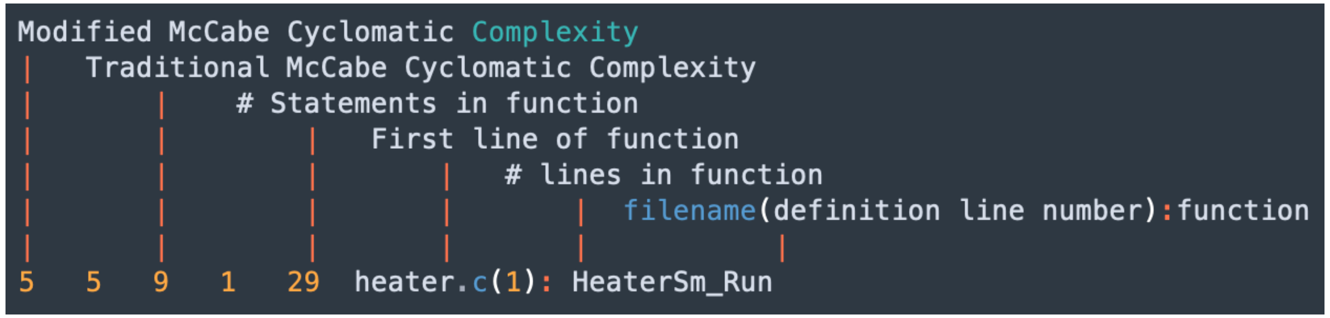
- Functional features
- Error handling that catches unknown bugs
- Logging and error reporting
- OTA updates
- Launching with MVP features
- Identifying high value / high use features
- Fast time to markets

Cyclomatic Complexity

How many tests do I need?


```c
HeaterState_t HeaterSm_Run(uint32_t const SystemTimeNow)
{
    // Manage state transition behavior
    if (HeaterState != HeaterStateRequested || UpdateTimer == true)
    {
        if (HeaterStateRequested == HEATER_ON)
        {
            HeaterState = HEATER_ON;
            EndTime = SystemTimeNow + HeaterOnTime;
            UpdateTimer = false;
        }
        else
        {
            EndTime = 0;
        }
    }
    else
    {
        // Do Nothing
    }

    // Manage HeaterState
    if(SystemTimeNow >= EndTime)
    {
        HeaterState = HEATER_OFF;
    }

    return HeaterState;
}
```

1. Two test cases for the first if/else pair.
2. Two test cases for the if/else pair of the HeaterStateRequested conditional.
3. One test case to decide if the heater should be turned off.

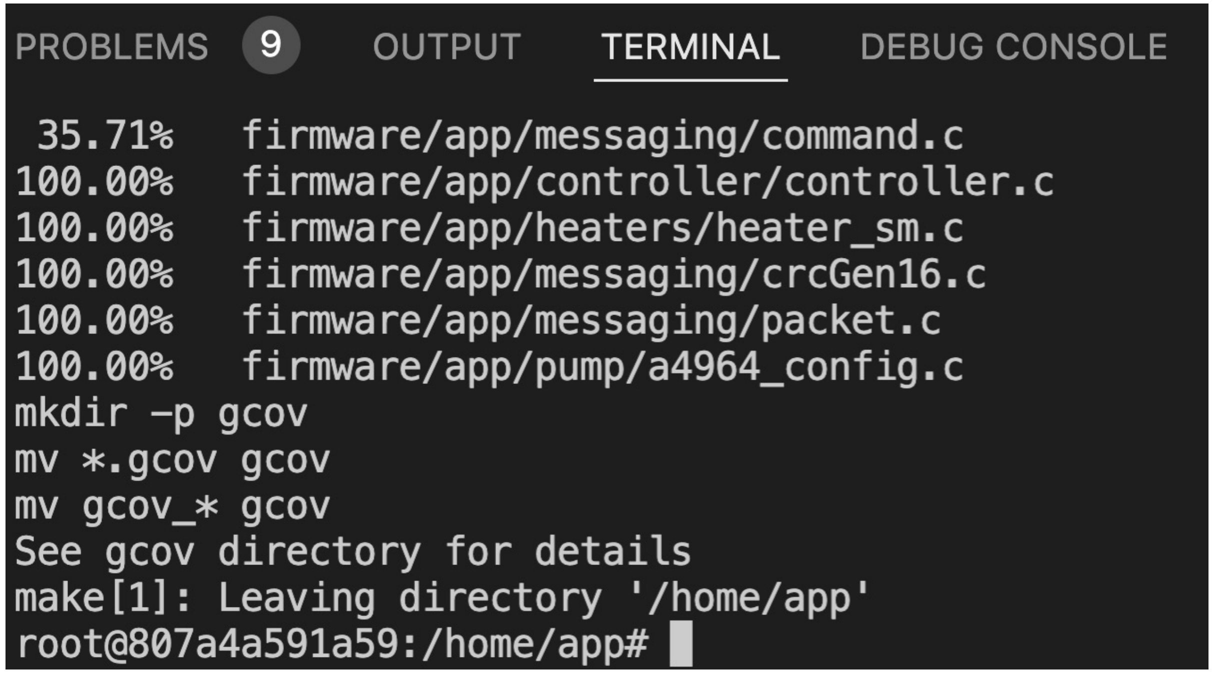

```
Modified McCabe Cyclomatic Complexity
|   Traditional McCabe Cyclomatic Complexity
|       |    # Statements in function
|       |        |   First line of function
|       |        |       |   # lines in function
|       |        |       |       |  filename(definition line number):function
|       |        |       |       |           |
5   5   9   1   29  heater.c(1): HeaterSm_Run
```

Cyclomatic Complexity

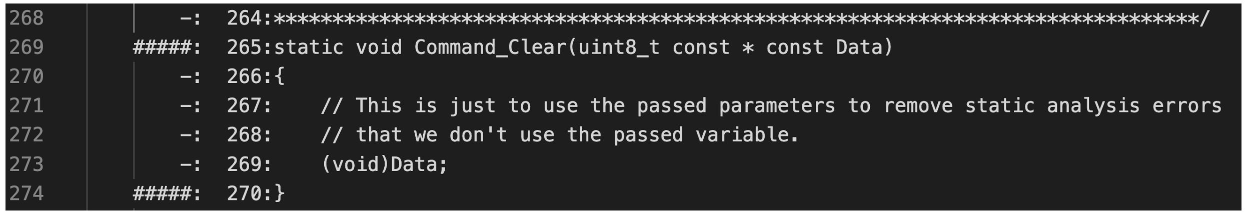
McCabe Cyclomatic Complexity (Cyclomatic Complexity) is a measurement that can be performed on software that measures the number of linearly independent paths within a function.

- The minimum number of test cases required to test the function.
- The risk associated with modifying the function.
- The likelihood that the function contains undiscovered bugs.

Code Coverage



```
 35.71%   firmware/app/messaging/command.c
100.00%   firmware/app/controller/controller.c
100.00%   firmware/app/heaters/heater_sm.c
100.00%   firmware/app/messaging/crcGen16.c
100.00%   firmware/app/messaging/packet.c
100.00%   firmware/app/pump/a4964_config.c
mkdir -p gcov
mv *.gcov gcov
mv gcov_* gcov
See gcov directory for details
make[1]: Leaving directory '/home/app'
root@807a4a591a59:/home/app# 
```



```
        -:  264:******************************************************************************/
    #####:  265:static void Command_Clear(uint8_t const * const Data)
        -:  266:{
        -:  267:    // This is just to use the passed parameters to remove static analysis errors
        -:  268:    // that we don't use the passed variable.
        -:  269:    (void)Data;
    #####:  270:}
```

Audience POLL Question

What's the best way to determine if you have enough tests?
- Code Coverage Report
- Cyclomatic Complexity
- Design Principles / Philosophy
- Use Test-Driven Development
- Other

02

gcov and JUnit

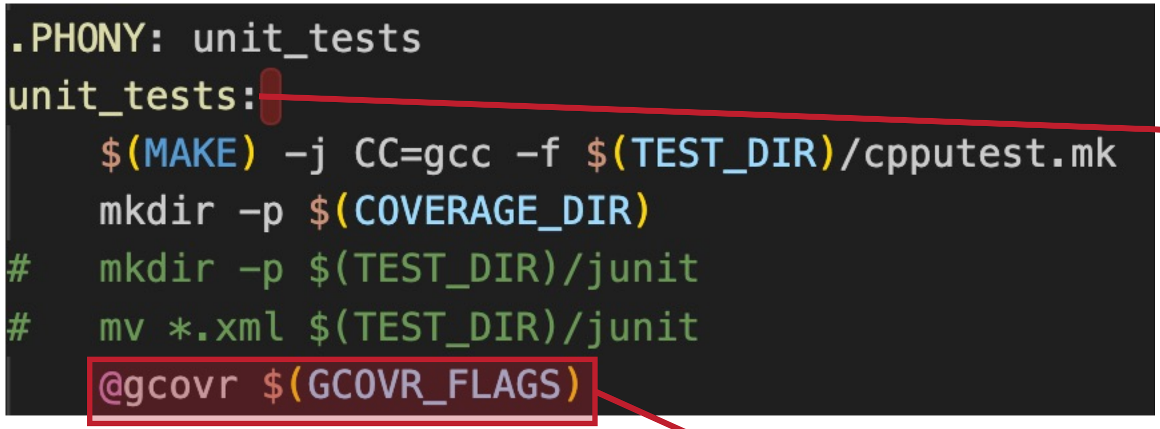
gcov

gcov is a test coverage program used in software testing that helps you analyze how effectively your codebase is tested by your test suite.

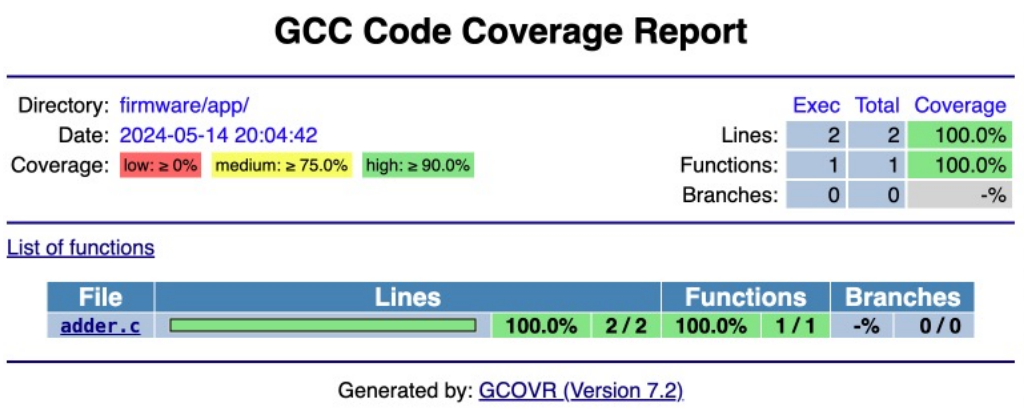
It is part of the GNU Compiler Collection (GCC) and is used in conjunction with GCC to test C, C++, and other supported programming languages.

Running gcov

Makefile



```
.PHONY: unit_tests
unit_tests:
	$(MAKE) -j CC=gcc -f $(TEST_DIR)/cpputest.mk
	mkdir -p $(COVERAGE_DIR)
#	mkdir -p $(TEST_DIR)/junit
#	mv *.xml $(TEST_DIR)/junit
	@gcovr $(GCOVR_FLAGS)
```



GCC Code Coverage Report

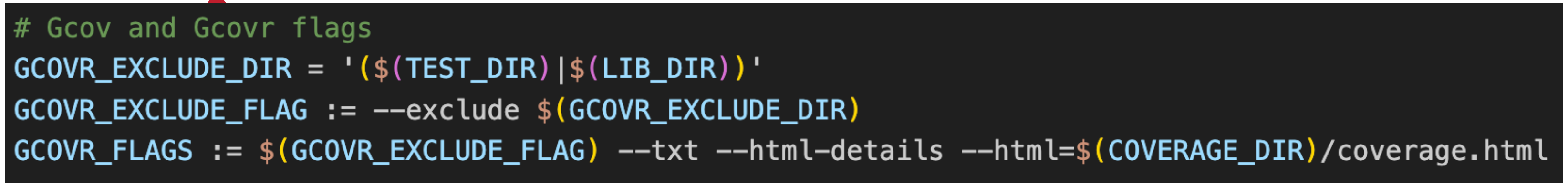
Directory: firmware/app/
Date: 2024-05-14 20:04:42
Coverage: low: ≥ 0% medium: ≥ 75.0% high: ≥ 90.0%

	Exec	Total	Coverage
Lines:	2	2	100.0%
Functions:	1	1	100.0%
Branches:	0	0	-%

List of functions

File	Lines			Functions		Branches	
adder.c	🟩	100.0%	2 / 2	100.0%	1 / 1	-%	0 / 0

Generated by: GCOVR (Version 7.2)



```
# Gcov and Gcovr flags
GCOVR_EXCLUDE_DIR = '($(TEST_DIR)|$(LIB_DIR))'
GCOVR_EXCLUDE_FLAG := --exclude $(GCOVR_EXCLUDE_DIR)
GCOVR_FLAGS := $(GCOVR_EXCLUDE_FLAG) --txt --html-details --html=$(COVERAGE_DIR)/coverage.html
```

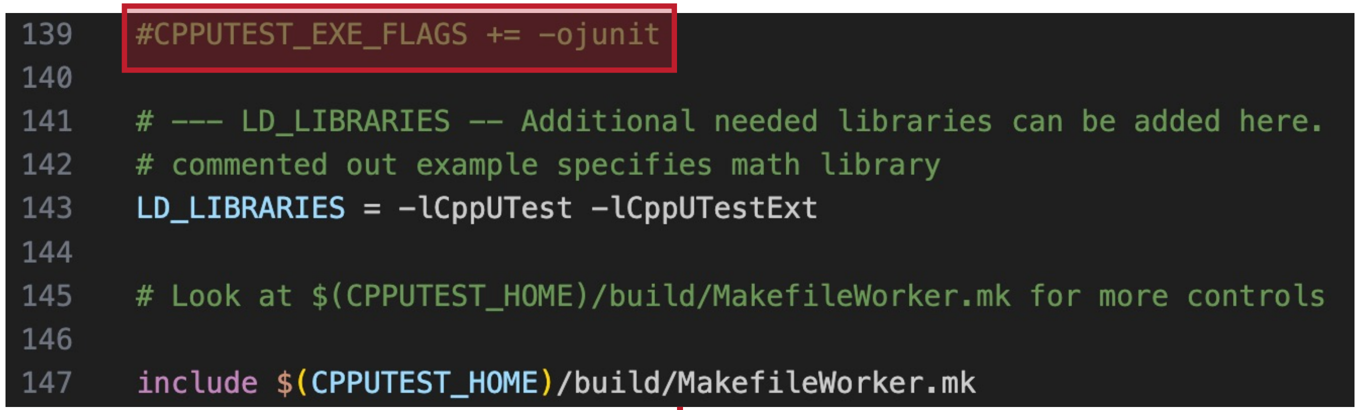
JUnit

JUnit is a popular unit testing framework used primarily for testing Java applications. It is an open-source framework that provides a simple and accessible method for writing and running automated tests to ensure your code behaves as expected.

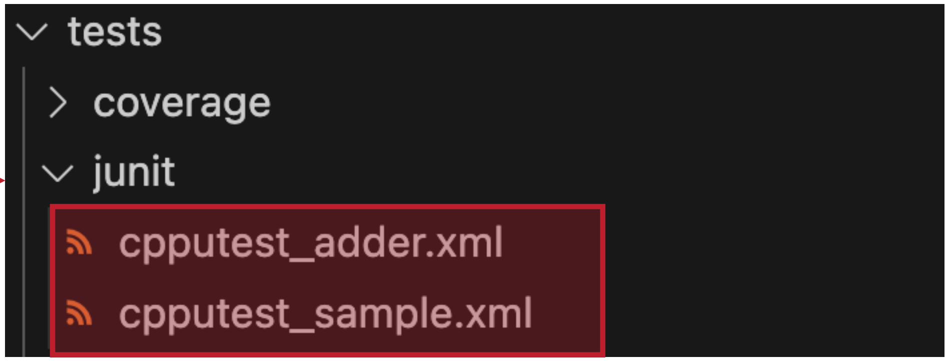
Why would we want to use junit in test automation with embedded systems?

JUnit

cpputest.mk



```
#CPPUTEST_EXE_FLAGS += -ojunit

# --- LD_LIBRARIES -- Additional needed libraries can be added here.
# commented out example specifies math library
LD_LIBRARIES = -lCppUTest -lCppUTestExt

# Look at $(CPPUTEST_HOME)/build/MakefileWorker.mk for more controls

include $(CPPUTEST_HOME)/build/MakefileWorker.mk
```



```
tests
  coverage
  junit
    cpputest_adder.xml
    cpputest_sample.xml
```

Audience POLL Question

What do gcov and junit do for you?

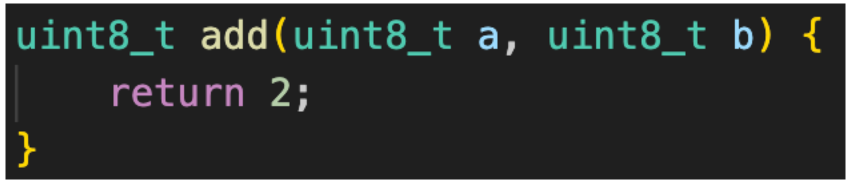
- Complexify the build system during testing
- Provide code coverage reports and analysis
- Provide test case results
- None of the above

03

Back to TDD

Our Test List

We left adder in an undesirable state:



```c
uint8_t add(uint8_t a, uint8_t b) {
    return 2;
}
```

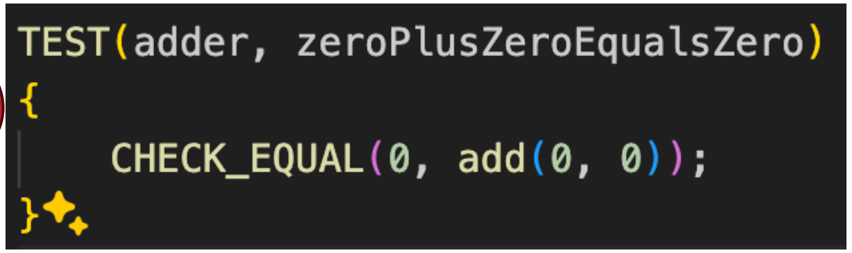
Continue writing our tests . . .

Make a list of tests:

- Zero values: a = 0, b = 0
- Zero and non-zero: a = 0, b = 10
- Zero and non-zero: a = 10, b = 0
- Positive Numbers: a = 50, a = 70
- Max without overflow: a = 125, b = 130
- Overflow: a = 200, b = 100
- “Typical” Values: a = 23, b = 77
- Symmetrical values: a = 123, b = 123
- Boundary values: a = 254, b = 1
- Overflow by 1: a = 255 + 1

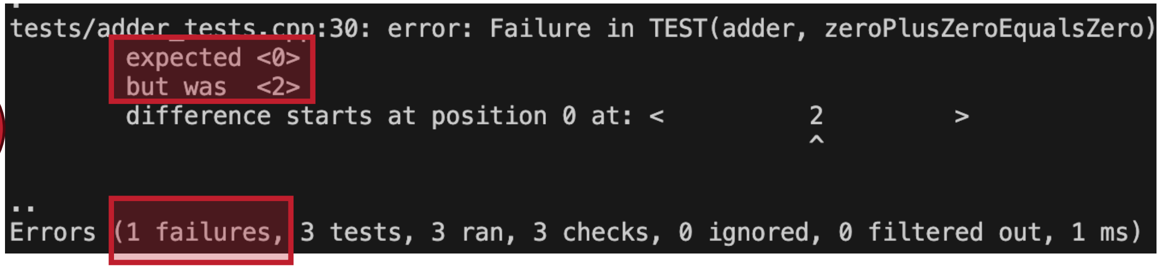
Write the 0 + 0 Test

1



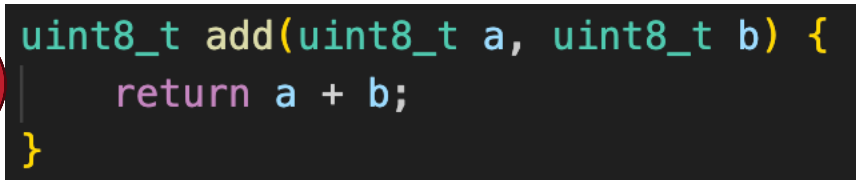
```
TEST(adder, zeroPlusZeroEqualsZero)
{
    CHECK_EQUAL(0, add(0, 0));
}
```

2



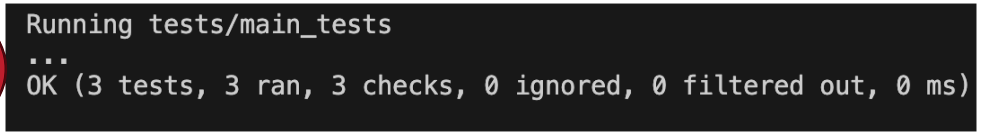
```
tests/adder_tests.cpp:30: error: Failure in TEST(adder, zeroPlusZeroEqualsZero)
        expected <0>
        but was  <2>
        difference starts at position 0 at: <          2         >
                                                       ^

..
Errors (1 failures, 3 tests, 3 ran, 3 checks, 0 ignored, 0 filtered out, 1 ms)
```

3



```
uint8_t add(uint8_t a, uint8_t b) {
    return a + b;
}
```

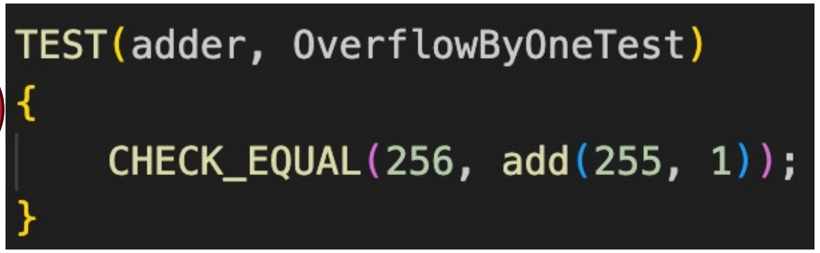
4



```
Running tests/main_tests
...
OK (3 tests, 3 ran, 3 checks, 0 ignored, 0 filtered out, 0 ms)
```

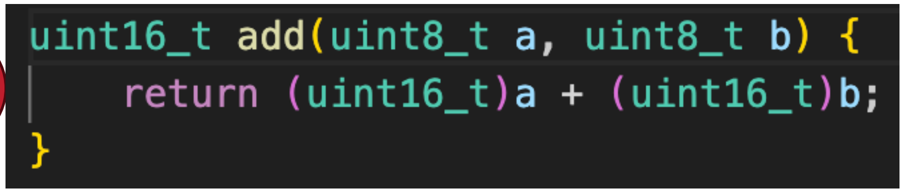
Write the 255+1 Test

1



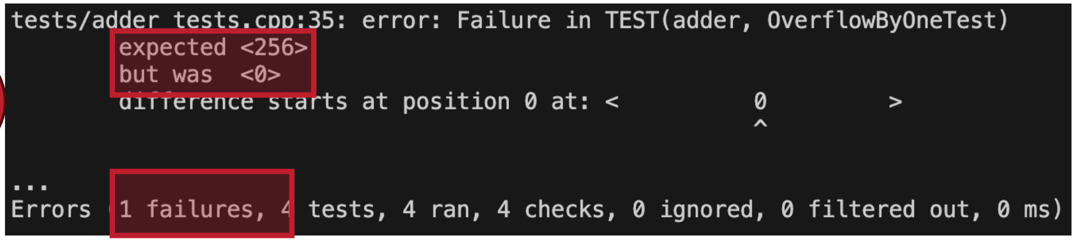
```
TEST(adder, OverflowByOneTest)
{
    CHECK_EQUAL(256, add(255, 1));
}
```

3



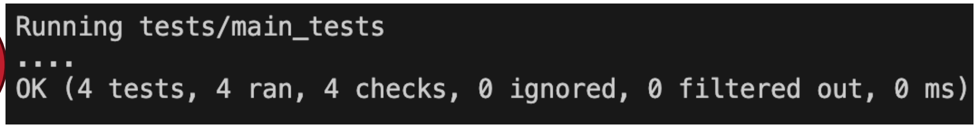
```
uint16_t add(uint8_t a, uint8_t b) {
    return (uint16_t)a + (uint16_t)b;
}
```

2



```
tests/adder_tests.cpp:35: error: Failure in TEST(adder, OverflowByOneTest)
        expected <256>
        but was  <0>
        difference starts at position 0 at: <          0         >
                                                       ^

...
Errors (1 failures, 4 tests, 4 ran, 4 checks, 0 ignored, 0 filtered out, 0 ms)
```

4



```
Running tests/main_tests
....
OK (4 tests, 4 ran, 4 checks, 0 ignored, 0 filtered out, 0 ms)
```

Audience POLL Question

How valuable do you see TDD being?

a) Highly Valuable
b) Providing some value
c) No value
d) Negative value, causing more problems than solving

Next Steps

04

CEC Continuing Education Center

BENINGO EMBEDDED GROUP

DigiKey

Test Automation Build System

Build System Example

- Docker container build system
- Makefile-based
- Cmake with Ninja Example
- Compilation scripts
- Integrated tools like cpputest

https://mailchi.mp/beningo/beningo-devops

Additional Resources

Please consider the resources below:

- Jacob's Blogs
- Jacob's CEC courses
- Embedded Software Academy

- Embedded Bytes Newsletter
  - http://bit.ly/1BAHYXm

www.beningo.com

Consulting

Coaching

Training

Next Steps

- ✓ Introduction to Test Automation Design
- ✓ Using Docker for a Test Automation Environment
- ✓ Unit-Testing Using Test-Driven Development Part 1
- ✓ Unit-Testing Using Test-Driven Development Part 2
- Automating System-Level Testing

CEC Continuing Education Center

DesignNews

Thank You

Sponsored by

DigiKey

BENINGO EMBEDDED GROUP

informa markets