これならやれる。
仮想環境を使った
Unityによる単体テストへの挑戦

2022/10/30 組込みCI研究WG　見澤

C言語向けテストハーネスUnityは、
インストールが意外と大変です。
ですが、仮想環境(Docker)を使うと
インストールが楽です。
今回はDockerが使える環境をなんとか用意できる人向けにネイティブCコンパイラとUnityを使った単体テストの導入方法の紹介をします。
次回はCMOCKを活用し、依存関係があるモジュールの単体テストの実施方法を紹介します。

■参考文献

James W. Grenning (著), 蛸島 昭之 (監修), 笹井 崇司 (翻訳).
テスト駆動開発による組み込みプログラミング ーC言語とオブジェクト指向で学ぶアジャイルな設計,
オライリージャパン, 2013, 36p

■参考Webサイト

https://github.com/ThrowTheSwitch/

https://goyoki.hatenablog.com/entry/20120519/1337441410
https://qiita.com/iwatake2222/items/396959d1d7dffee479f7
https://futurismo.biz/archives/1281/

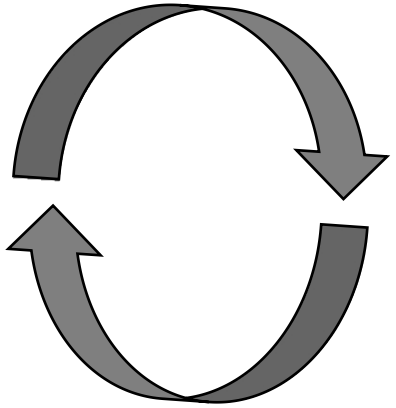
■前提

dockerが使える環境を用意する。

＜私見＞

WindowsにWSL2をインストールして
Linuxを使えるようにするのが良いと思います。
MACは使ったことないので、分かりません。

■構成



単体テスト実行

ローカル

コンテナ
（MadScienceLabDocker）

プロダクトコード

テストコード

マウント
（コンテナ内からローカルのディレクトリを参照する）

プロダクトコード

テストコード

ネイティブCコンパイラ

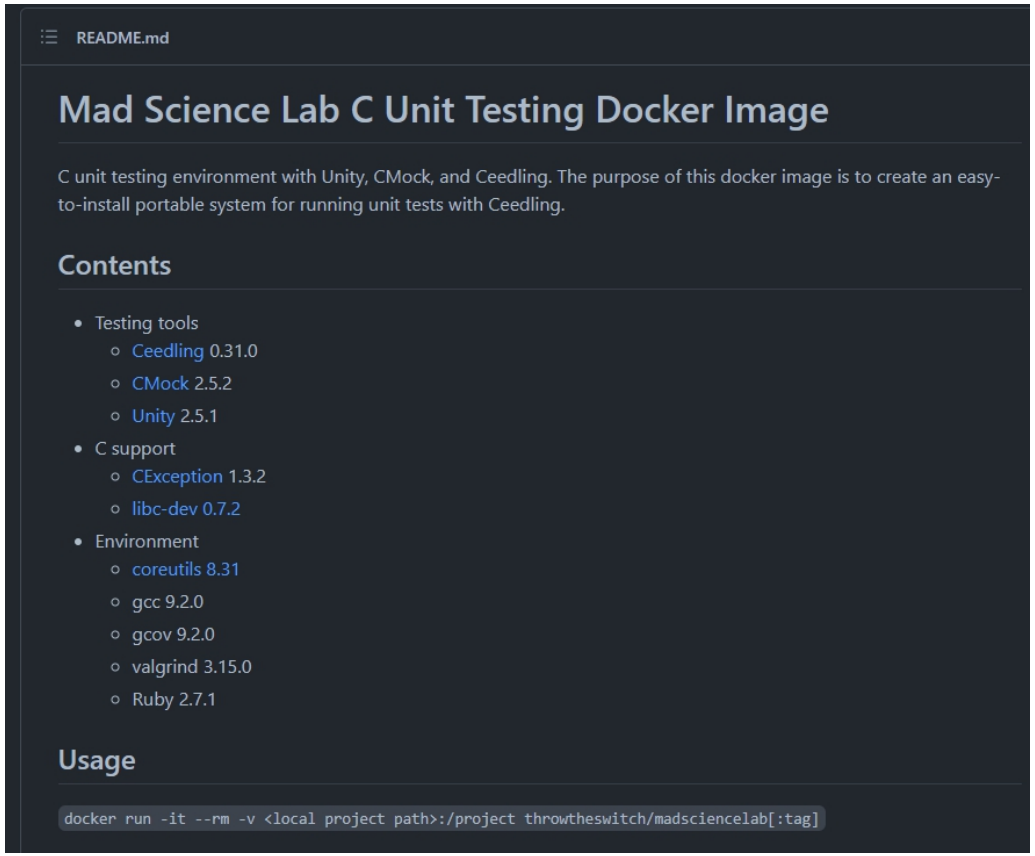
Unity、
CMock、
Ceedling

コーディング、コード修正

Docker（仮想環境）

■MadScienceLabDocker

Unityが使える環境を準備するのに、
本Dockerイメージを使うのがお勧め。（取得方法：次ページ）



README.md

Mad Science Lab C Unit Testing Docker Image

C unit testing environment with Unity, CMock, and Ceedling. The purpose of this docker image is to create an easy-to-install portable system for running unit tests with Ceedling.

Contents

- Testing tools
  - Ceedling 0.31.0
  - CMock 2.5.2
  - Unity 2.5.1
- C support
  - CException 1.3.2
  - libc-dev 0.7.2
- Environment
  - coreutils 8.31
  - gcc 9.2.0
  - gcov 9.2.0
  - valgrind 3.15.0
  - Ruby 2.7.1

Usage

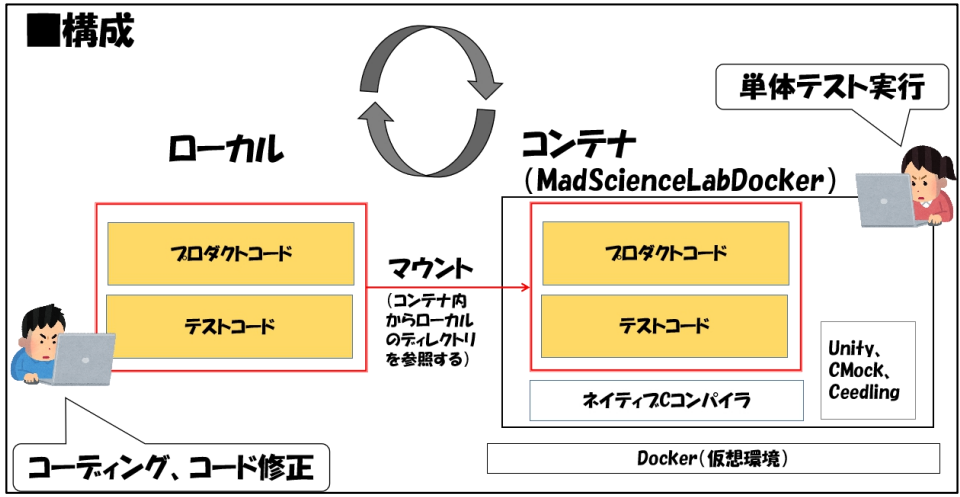
```
docker run -it --rm -v <local project path>:/project throwtheswitch/madsciencelab[:tag]
```

理由

1. Linuxを使うスキルに自信が無い場合、
   一からUnityが使える環境を
   自分で準備できない。または時間がかかる。

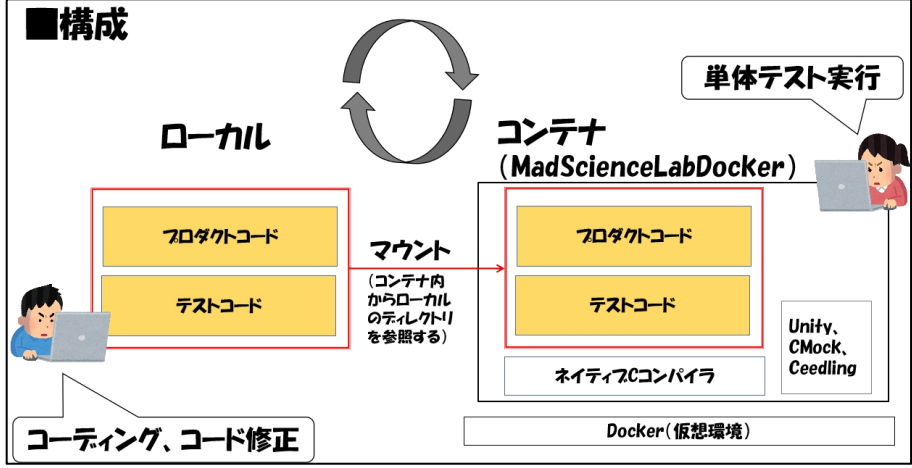
2. MadScienceLabDockerを使うと、
   Unityを含めた便利ツールが
   すぐ使えるようになる。

■MadScienceLabDockerの取得方法




```
$ git clone https://github.com/ThrowTheSwitch/MadScienceLabDocker.git
```

■コンテナの実行方法



docker run -it --rm -v <local project path>:/project throwtheswitch/madsciencelab[:tag]

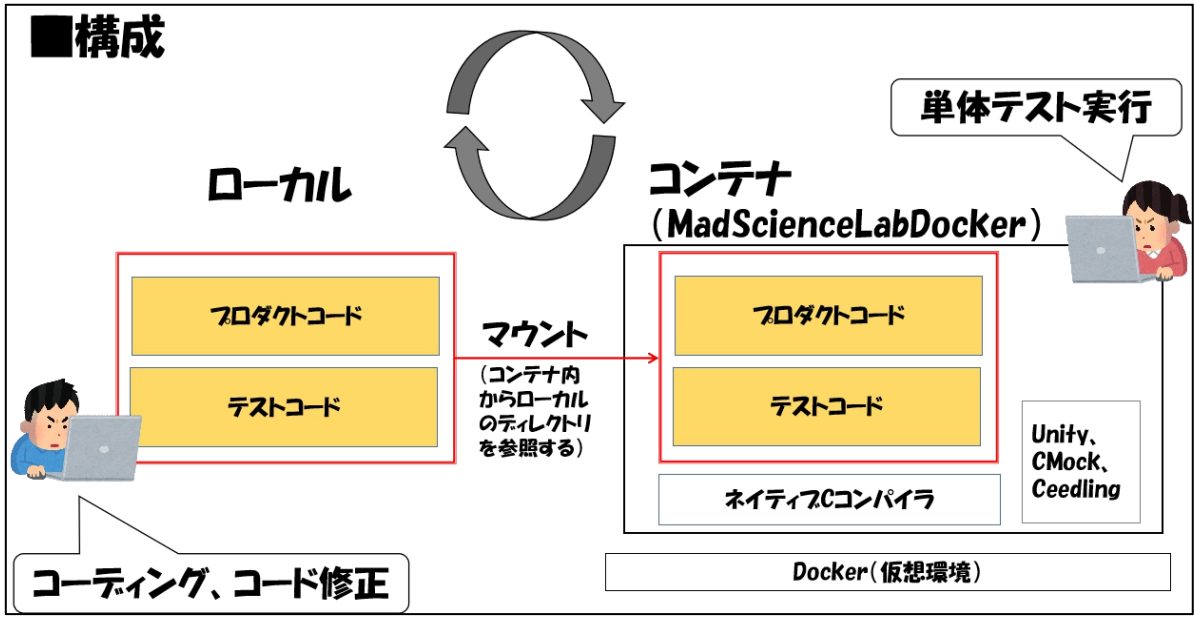
(例)

1. ローカルのMadScienceLabDockerディレクトリに移動

2. *$docker run -it --rm -v ~/environment/dev/unittest2/hoge:/project/hoge throwtheswitch/madsciencelab*

■新規プロジェクト作成

ceedling new YourNewProjectName



(例)

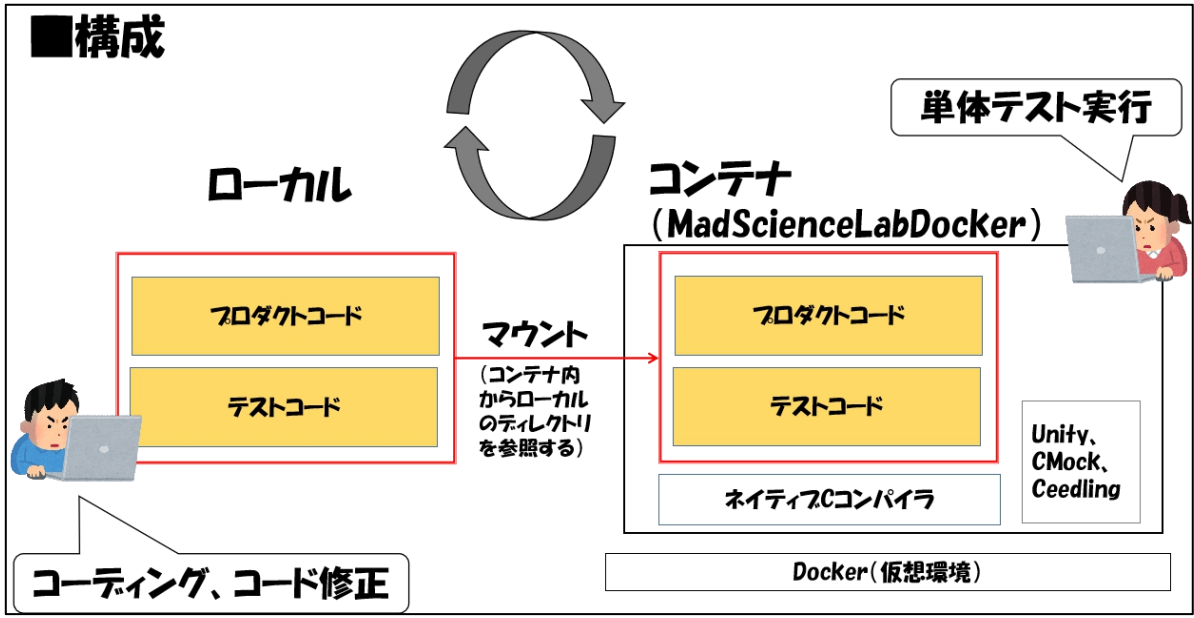
1. コンテナでマウントしたhogeディレクトリの中にあるhoge3ディレクトリに移動

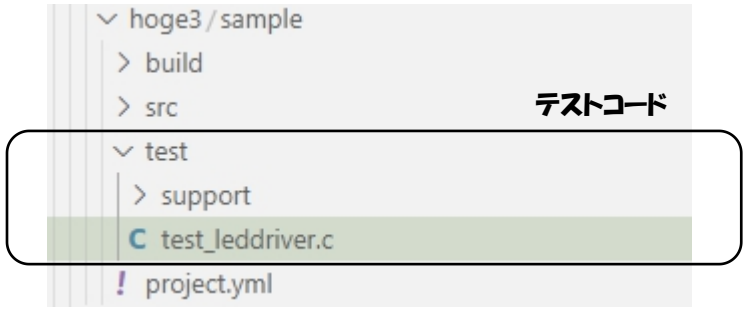
2. 新規プロジェクト作成
   $ceedling new sample

テンプレート(src、test、etc)が入った、
指定したsampleという名前のディレクトリが作られる。

■テストコードの作成

ローカルでテストコードを作成する。





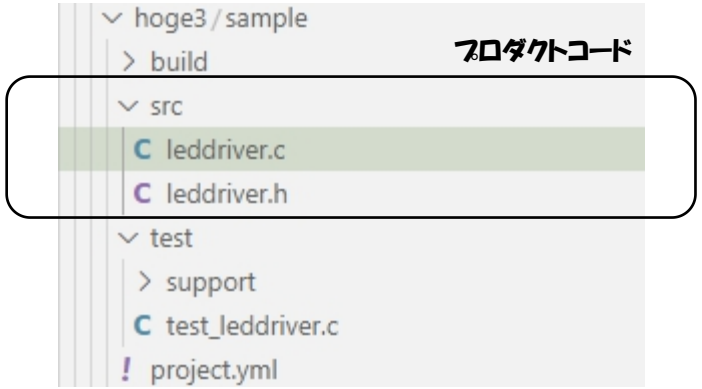
テストコード

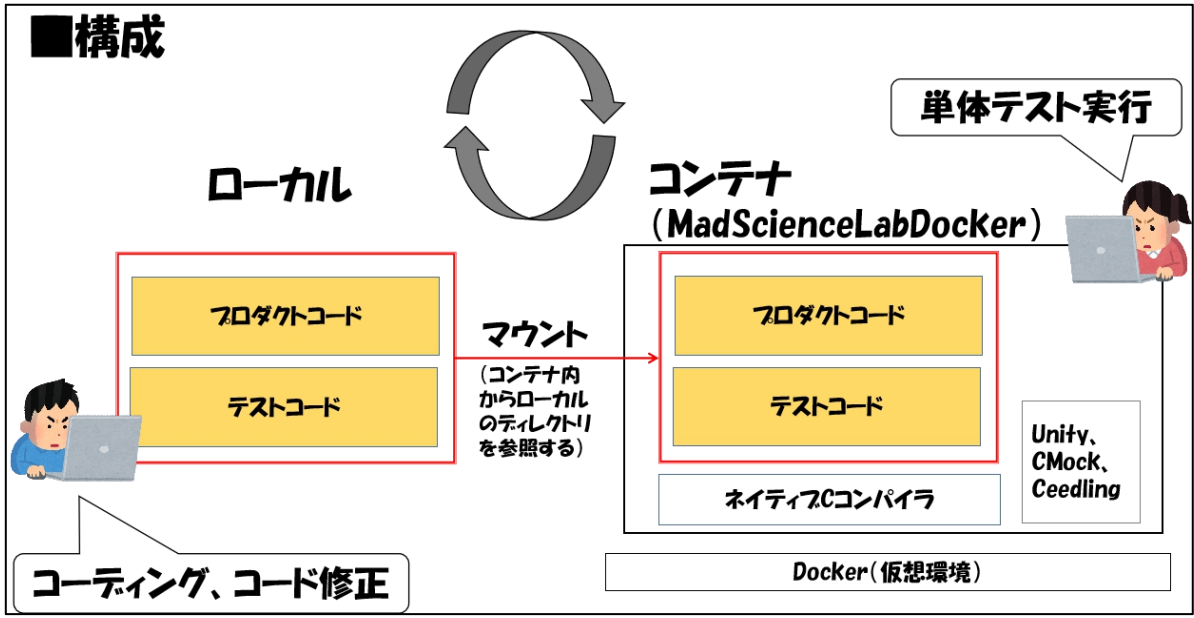
environment > dev > unittest2 > hoge > hoge3 > sample > test > C test_leddriver.c > ...

```c
#include "unity.h"
#include "leddriver.h"

void setUp(void)
{
}

void tearDown(void)
{
}

void test_init_led_address(void)
{
    uint16_t virtualleds = 0xffff;
    LedDriveer_Create(&virtualleds);
    TEST_ASSERT_EQUAL_HEX16(0, virtualleds);
}

```

■プロダクトコードの作成

ローカルでプロダクトコードを作成する。





プロダクトコード

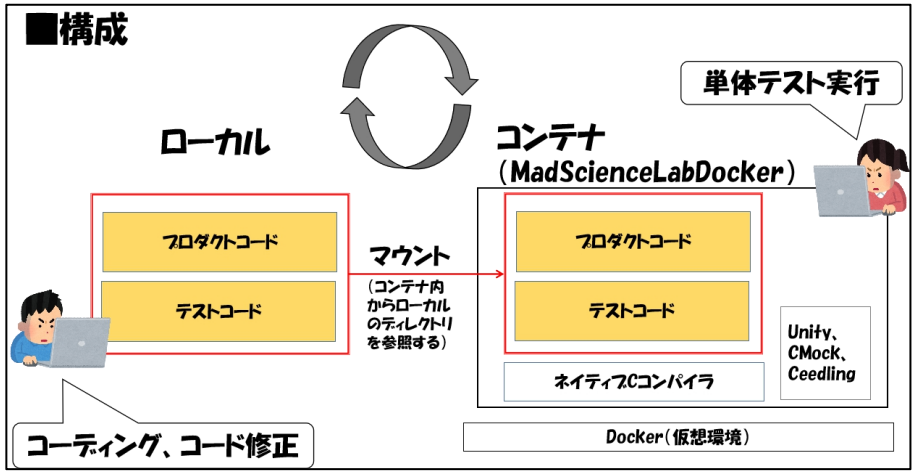
environment > dev > unittest2 > hoge > hoge3 > sample > src > C leddriver.c > ...

```c
#include "leddriver.h"

void LedDriveer_Create(uint16_t *address)
{
}

```

environment > dev > unittest2 > hoge > hoge3 > sample > src > C leddriver.h > ...

```c
#ifndef _H_LEDDRIVER_
#define _H_LEDDRIVER_

#include <stdint.h>

extern void LedDriveer_Create(uint16_t *address);

#endif /* _H_LEDDRIVER_ */

```

■単体テスト実行
コンテナでceedlingコマンドを実行。




```
/project/hoge/hoge3/sample # ceedling


Test 'test_leddriver.c'
-----------------------
Compiling leddriver.c...
Linking test_leddriver.out...
Running test_leddriver.out...

-------------------
FAILED TEST SUMMARY
-------------------
[test_leddriver.c]
  Test: test_init_led_address
  At line (16): "Expected 0x0000 Was 0xFFFF"

--------------------
OVERALL TEST SUMMARY
--------------------
TESTED:  1
PASSED:  0
FAILED:  1
IGNORED: 0

---------------------
BUILD FAILURE SUMMARY
---------------------
Unit test failures.
```

プロダクトコードの初期化処理が未実装のため、テストに失敗する。

テストコード

environment > dev > unittest2 > hoge > hoge3 > sample > test > C test_leddriver.c > ...

```c
#include "unity.h"
#include "leddriver.h"

void setUp(void)
{
}

void tearDown(void)
{
}

void test_init_led_address(void)
{
    uint16_t virtualleds = 0xffff;
    LedDriveer_Create(&virtualleds);
    TEST_ASSERT_EQUAL_HEX16(0, virtualleds);
}

```

プロダクトコード

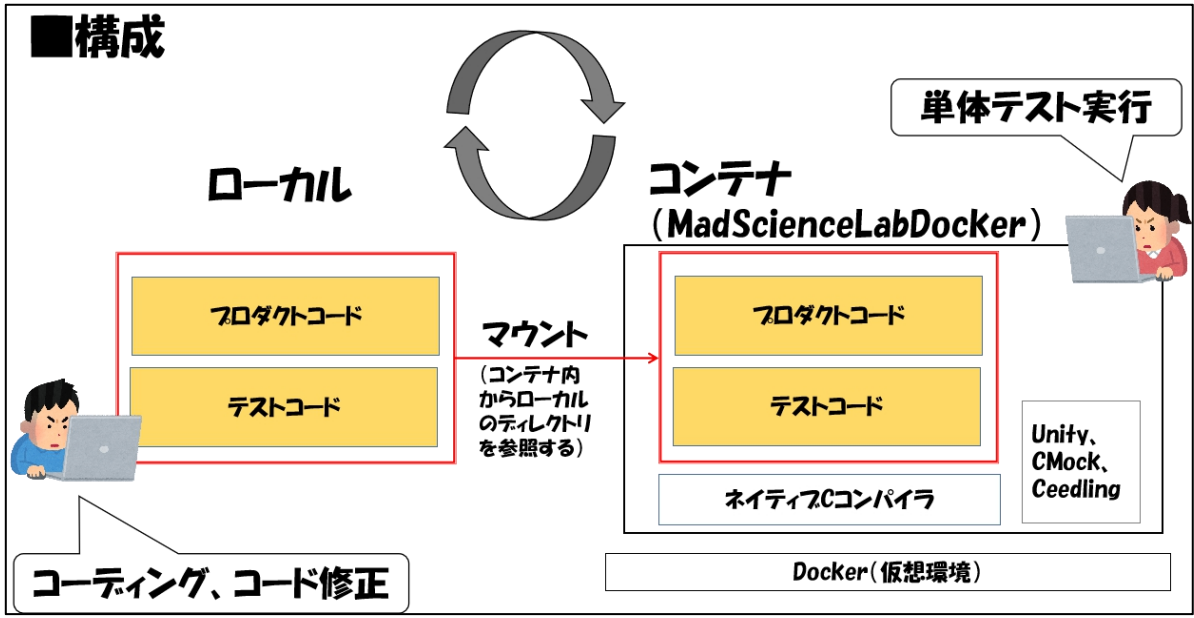
environment > dev > unittest2 > hoge > hoge3 > sample > src > C leddriver.c > ...

```c
#include "leddriver.h"

void LedDriveer_Create(uint16_t *address)
{
}

```

初期化処理が未実装

■プロダクトコードの修正

ローカルでプロダクトコードを修正する。



修正前

environment > dev > unittest2 > hoge > hoge3 > sample > src > C leddriver.c > ...

```c
#include "leddriver.h"

void LedDriveer_Create(uint16_t *address)
{
}

```

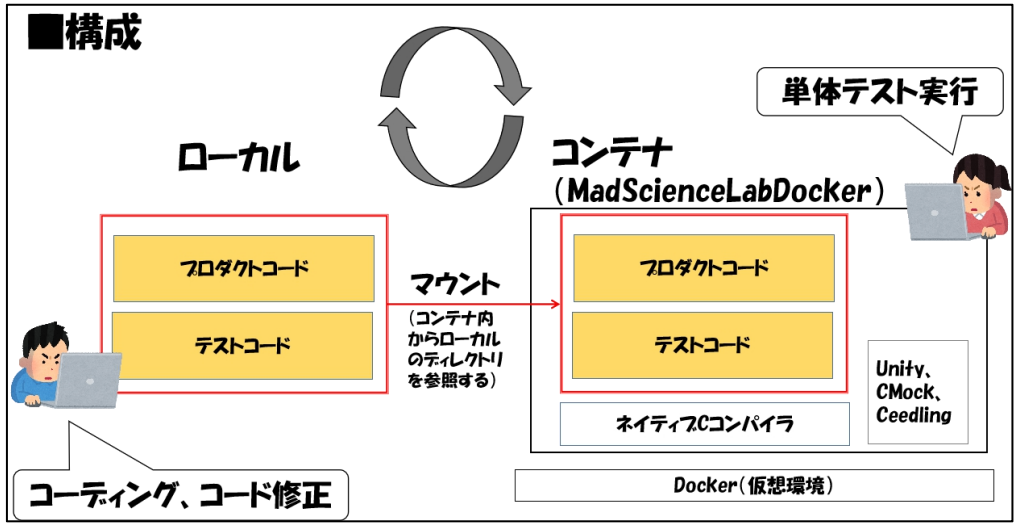
修正後

environment > dev > unittest2 > hoge > hoge3 > sample > src > C leddriver.c > ...

```c
#include "leddriver.h"

void LedDriveer_Create(uint16_t *address)
{
    *address = 0;
}

```

■単体テストの再実行

コンテナでceedlingコマンドを実行。



修正前

```
/project/hoge/hoge3/sample # ceedling


Test 'test_leddriver.c'
-----------------------
Compiling leddriver.c...
Linking test_leddriver.out...
Running test_leddriver.out...

-------------------
FAILED TEST SUMMARY
-------------------
[test_leddriver.c]
  Test: test_init_led_address
  At line (16): "Expected 0x0000 Was 0xFFFF"

--------------------
OVERALL TEST SUMMARY
--------------------
TESTED:  1
PASSED:  0
FAILED:  1
IGNORED: 0

---------------------
BUILD FAILURE SUMMARY
---------------------
Unit test failures.
```

テスト失敗

修正後

```
/project/hoge/hoge3/sample # ceedling


Test 'test_leddriver.c'
-----------------------
Running test_leddriver.out...

--------------------
OVERALL TEST SUMMARY
--------------------
TESTED:  1
PASSED:  1
FAILED:  0
IGNORED: 0
```

テスト成功

以上