# C unit testen met Unity framework

Unit testen betekent een stuk code testen die zo klein mogelijk is. Dus een methode of een functie. Je test deze (of eigenlijk wil je eerst de test schrijven zodat je van te voren precies weet wat de code moet doen) zo vroeg mogelijk; hoe eerder je fouten opspoort in het ontwikkelen van je programma hoe makkelijk die eruit te halen zijn.  
Mar bedenk: “*Program testing can be used to show the presence of bugs, but never to show their absence!*”- Edsger Dijkstra  
Hoe goed je ook de fouten eruit haalt die in unittesten tevoorschijn komen, uiteindelijk zal het in gebruik zetten van je code (in productie gaan) laten zien hoe foutvrij alles is. En dat is nooit.

Enfin, we gebruiken het Unity framework en Visual Studio Code om unit testen uit te voeren.  
Haal Unity op (<https://www.throwtheswitch.org/unity>). Visual Studio Code heb je ook al draaien.  
Voor dit eerste voorbeeld ziet de directorystructuur er als volgt uit:  
- UnitTest  
 - Unity  
 - DumbExample.c  
 - DumbExample.h  
 - TestDumbExample.c

Hier zijn UnitTest en Unity folders. Voor dit voorbeeld staat Unity in dezelfde workspace als de code, maar als je eigenlijk maar één keer Unity ergens centraal wilt wegzetten, dan moet je dat in VS Code vermelden onder je compilerpath. Dan is Unity altijd te vinden.

DumpExample.c is de code die we willen gaan testen. Dit zouden we dus zelf geschreven hebben als code die ooit in productie gaat draaien. Code ziet er zo uit:

#include "DumbExample.h"

void setUp (void) {} /\* wordt VOOR elke test gedraaid, hier unit initialisatie dingen zetten. \*/

void tearDown (void) {} /\* wordt NA elke test gedraaid, hier alles netjes opruimen. \*/

int8\_t AverageThreeBytes(int8\_t a, int8\_t b, int8\_t c)

{

return (int8\_t)(((int16\_t)a + (int16\_t)b + (int16\_t)c) / 3);

}

Wat de code doet, is niet zo belangrijk. Hier worden het gemiddelde van 3 bytes berekend.  
  
Om de boel te testen, moeten we een stuk test code schrijven. Die heet hier TestDumExample.c en ziet er zo uit:

#include "unity.h"

#include "DumbExample.h"

void test\_AverageThreeBytes\_should\_AverageMidRangeValues(void)

{

TEST\_ASSERT\_EQUAL\_HEX8(40, AverageThreeBytes(30, 40, 50));

TEST\_ASSERT\_EQUAL\_HEX8(40, AverageThreeBytes(10, 70, 40));

TEST\_ASSERT\_EQUAL\_HEX8(33, AverageThreeBytes(33, 33, 33));

}

void test\_AverageThreeBytes\_should\_AverageHighValues(void)

{

TEST\_ASSERT\_EQUAL\_HEX8(80, AverageThreeBytes(70, 80, 90));

TEST\_ASSERT\_EQUAL\_HEX8(127, AverageThreeBytes(127, 127, 127));

TEST\_ASSERT\_EQUAL\_HEX8(84, AverageThreeBytes(0, 126, 126));

}

int main(void)

{

UNITY\_BEGIN();

RUN\_TEST(test\_AverageThreeBytes\_should\_AverageMidRangeValues);

RUN\_TEST(test\_AverageThreeBytes\_should\_AverageHighValues);

return UNITY\_END();

}

Je ziet hier bovenaan dat unity.h ingevoegd wordt en een headerfile die aangeeft hoe onze methode eruit ziet. Er zijn 3 methoden om te testen of de methode AverageThreeBytes werkt. Je ziet dat het behoorlijk lange namen zijn, maar dan kun je straks meteen zien welke test er wel of niet doorheen is gekomen.  
In de main() worden de testmethoden aangeroepen. Dat kun je doen met RUN\_TEST(<naam methode>). Je start altijd met UNITY\_BEGIN(); (die zet van alles goed om een test te kunnen runnen) en eindigt met return UNITY\_END();.

Pakken we de eerste methode **test\_AverageThreeBytes\_should\_AverageMidRangeValues** dan zien we dat die controleert of wat terugkomt uit de te testen methode een (hex) getal is met het commando TEST\_ASSERT\_EQUAL\_HEX8. De parameters voor dit commando zijn: wat verwacht je terug, de methode die je wilt testen en zijn parameters. De eerste regel test of de methode terugkomt met 40 als je er 30,40 en 50 instopt.  
Uiteraard zijn er nog veel meer manieren om iets te testen, dan behalve een getal. Pointers, arrays, strings etc. Hier is een lijst: <https://embetronicx.com/tutorials/unit_testing/unit-testing-in-c-testing-with-unity/>

Tijd om te kijken of het werkt. Dus gaan we compileren. Dat doen we met het commando (in terminal venster): gcc TestDumbExample.c DumbExample.c ./unity/src/unity.c -o TestDumbExample

Uiteraard is de verwijzing naar Unity hier zo gezet, specifiek voor dit voorbeeld. Als Unity bij jou ergens anders staat, dan uiteraard hier een ander pad opgeven.  
Het resultaat is een .exe met de naam TestDumbExample.exe. Als je die uitvoert zie je (hopelijk!) dit:

TestDumbExample.c:21:test\_AverageThreeBytes\_should\_AverageMidRangeValues:PASS

TestDumbExample.c:22:test\_AverageThreeBytes\_should\_AverageHighValues:PASS

-----------------------

2 Tests 0 Failures 0 Ignored

OK

De twee tests zijn uitgevoerd en kloppen alletwee (of beter gezegd: wat je verwachtte dat jouw code deed, is ook zo).

Dat inkloppen van de gcc-regel wordt al snel saai, dus maak daar een Makerbestand van.  
Of probeer de VS code extentie ‘Test Explorer’ eens uit. Die maakt de wereld ook meteen een stuk mooier.

In de folder ‘examples’ onder Unity staan een groot aantal voorbeelden (die verder gaan dan dit voorbeeld) hoe het testen aan te pakken.

En om het compleet te maken, die DumbExample.h ziet er zo uit:

#include <stdint.h>

int8\_t AverageThreeBytes(int8\_t a, int8\_t b, int8\_t c);