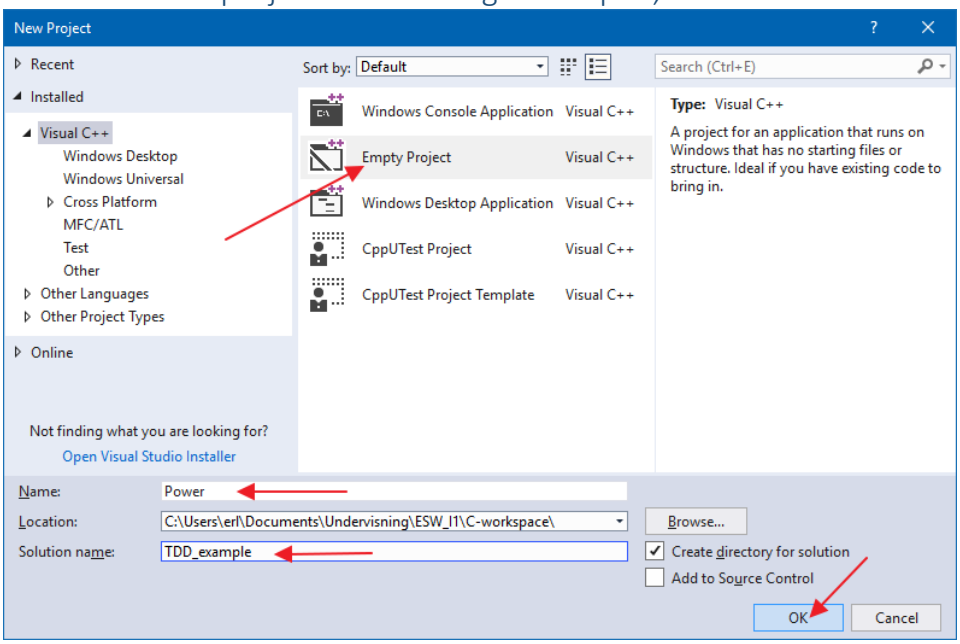
Test Driven Development – a simple example

"**Test-driven development** (**TDD**) is a software development process that relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, then the software is improved to pass the new tests, only. This is opposed to software development that allows software to be added that is not proven to meet requirements." (Wikipedia, 2019)

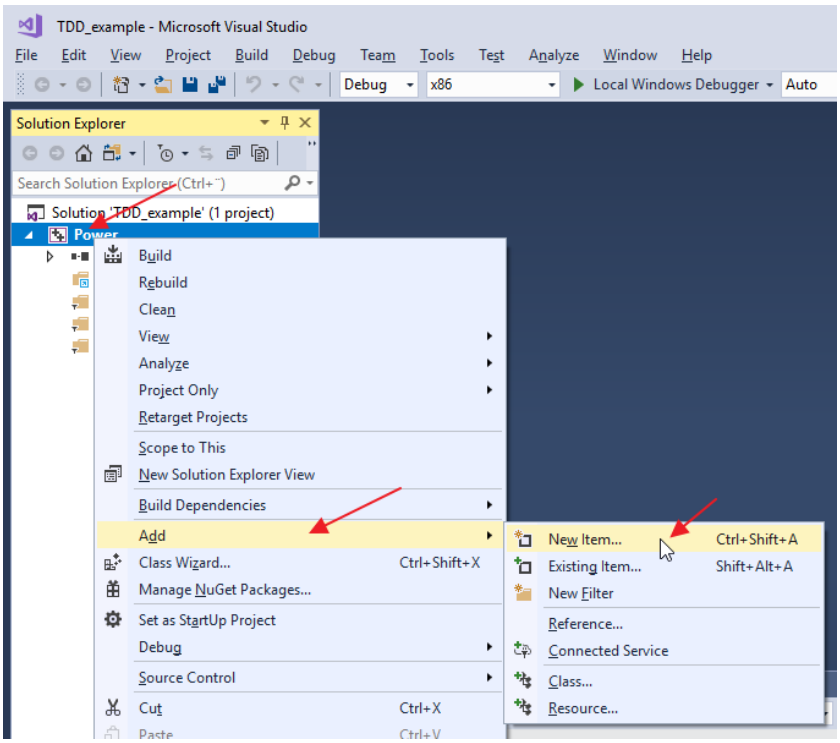
Preparation for this guide:

This example use VisualStudio as IDE and CppUTest as test harness.
A guide on how to install Visual Studio can be found in the document "Visual Studio for C - Installation Guide A2017.pdf", by Ib Havn.
A guide on how to install and setup CppUTest can be found in the document "Simple Use of CppUTest in Visual Studio 2017.pdf" by Ib Havn.

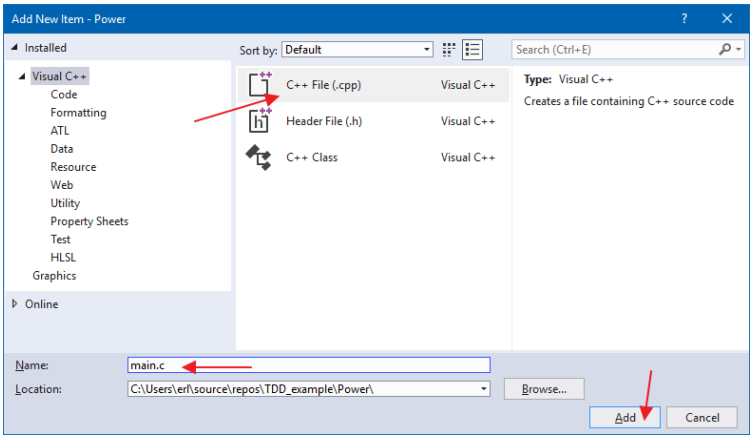
1. Create the C-project that is being developed, in Visual Studio



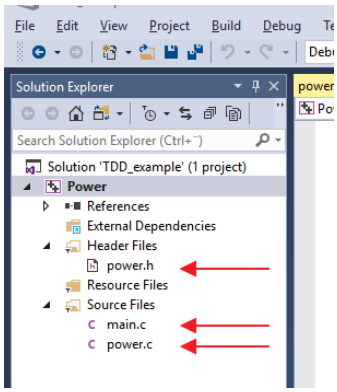
When the new project opens, right-click on the project name and add new item:



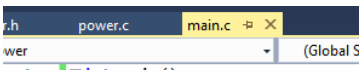
Add a main.c file.

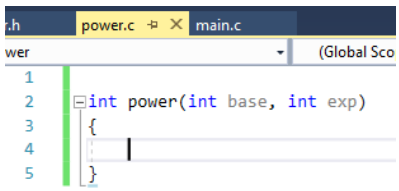


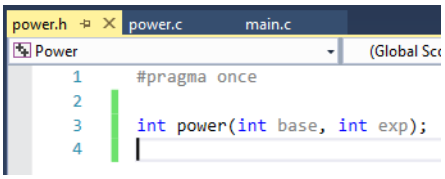
In the same way, add power.c.
Also add power.h by choosing the file type “Header file (.h)”.
After this step, the file tree should look like this:



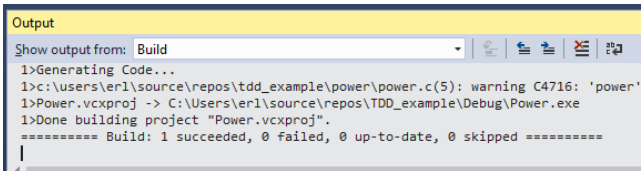
Write the minimal code for main.c, power.c and power.h





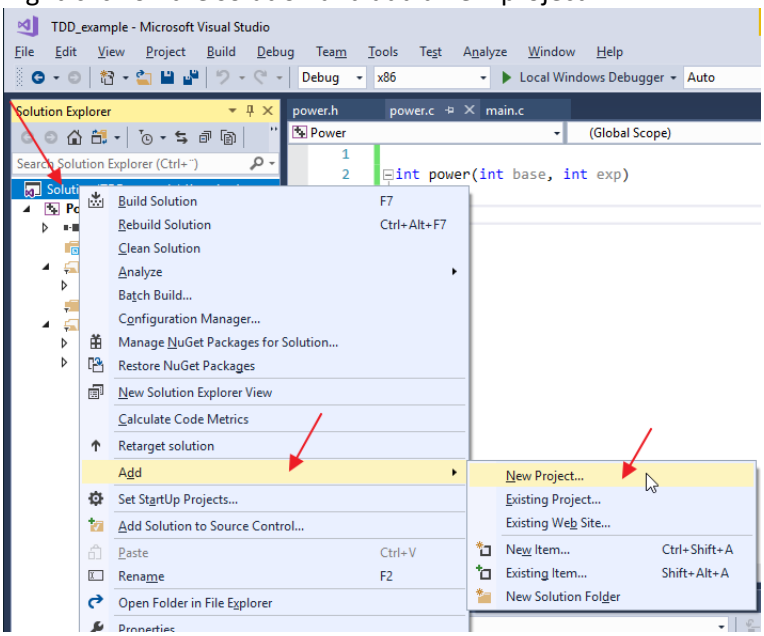


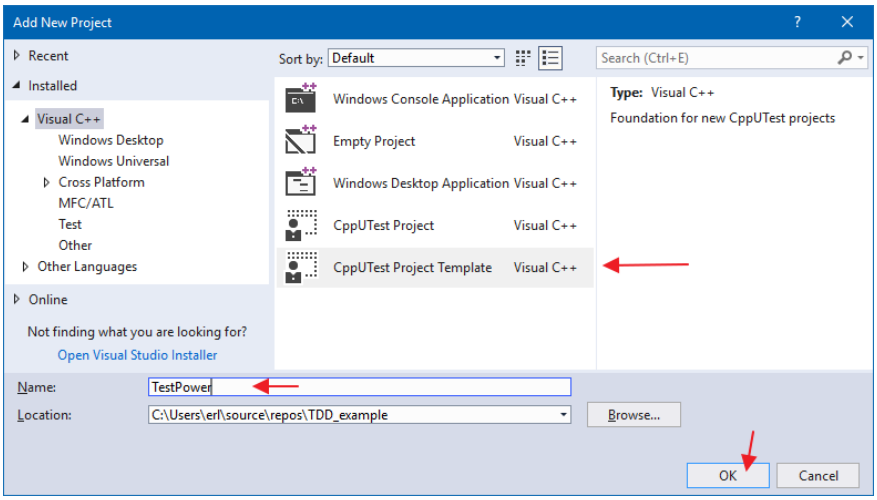
Check that the project can compile by pressing F7



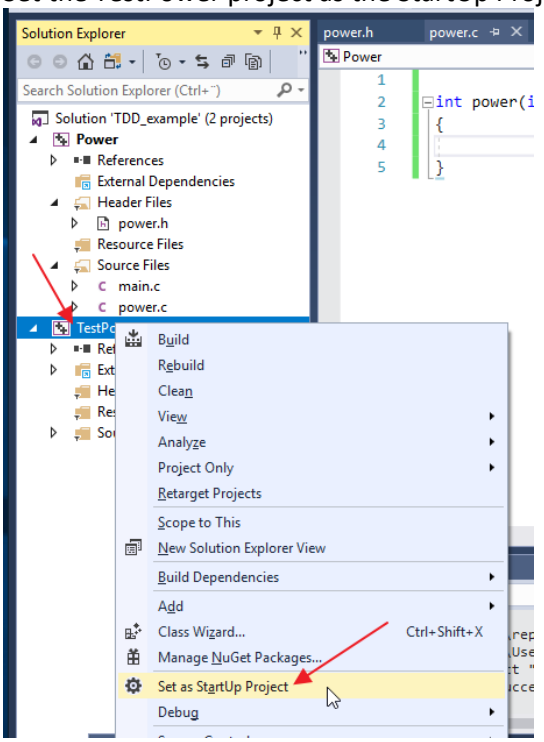
2. Set up a test project with CppUTest, to drive the development process

Right-click on the solution and add a new project.

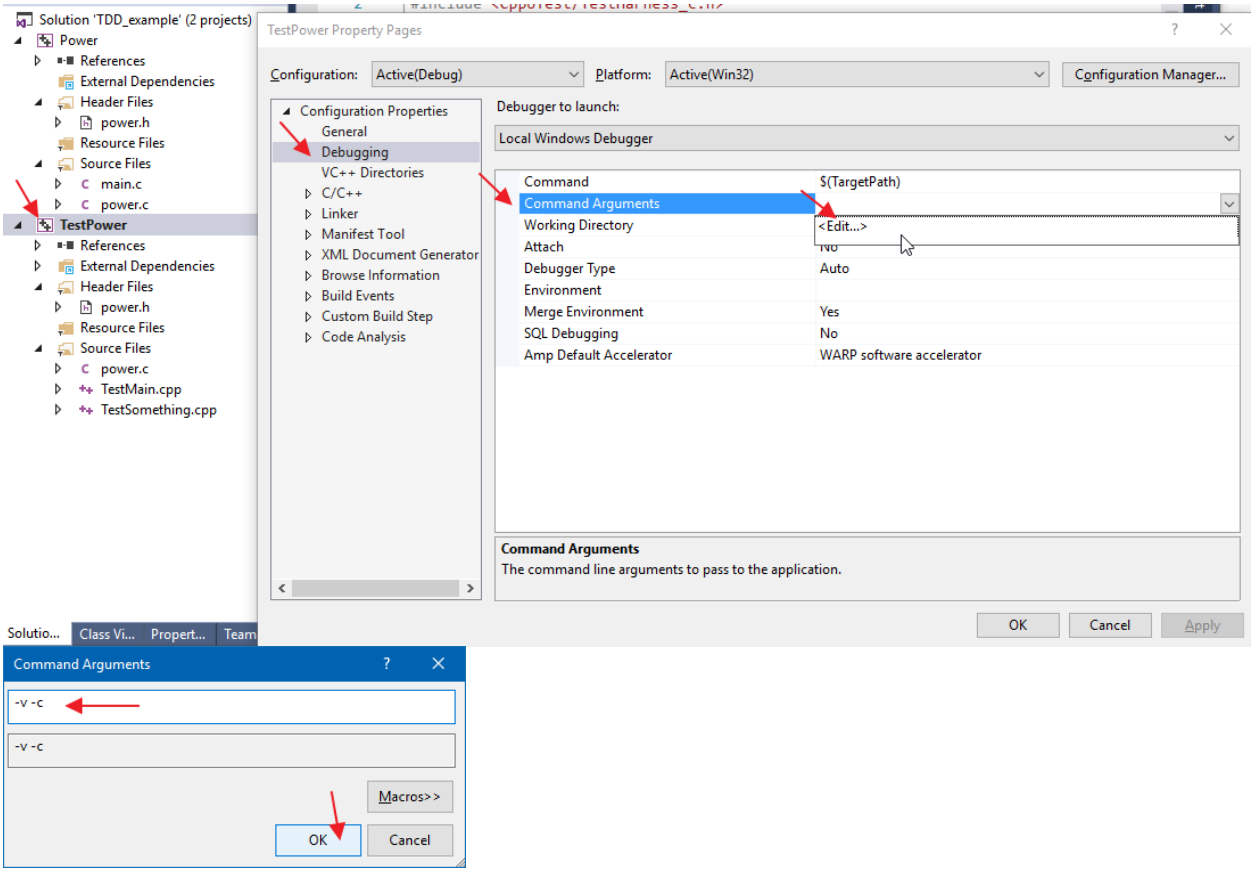




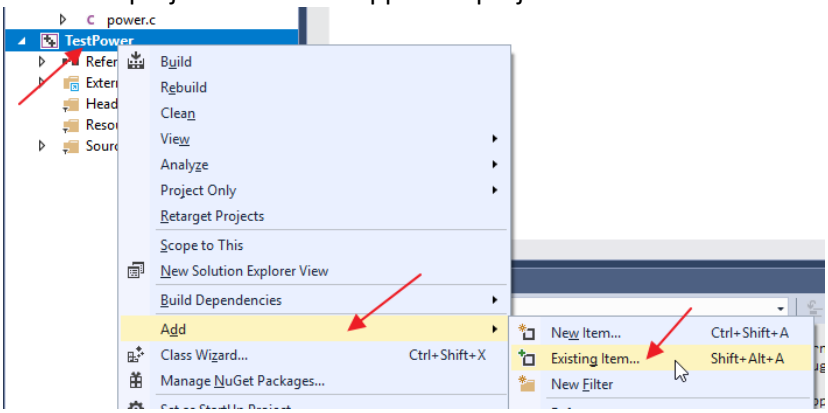
Set the TestPower project as the StartUp Project



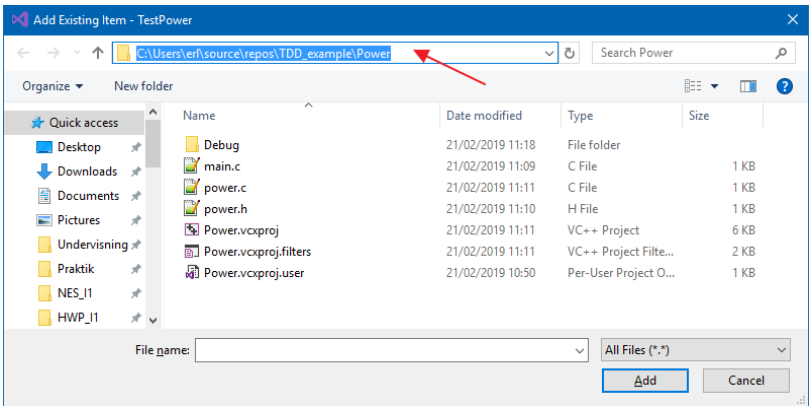
Setup command arguments for the debugger so passed tests are shown in green and failed tests are shown in red.
Right-click the CppUProject and then Properties. Add the two switches “-v –c” to Command Arguments.



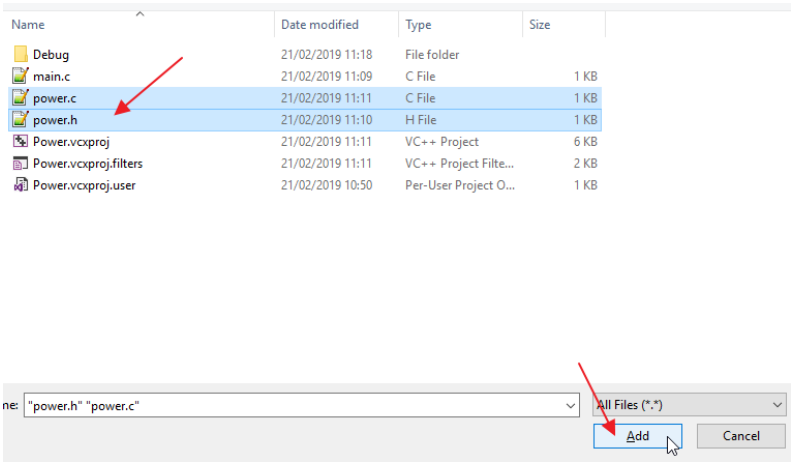
Add the C-project files to the CppUTest project.

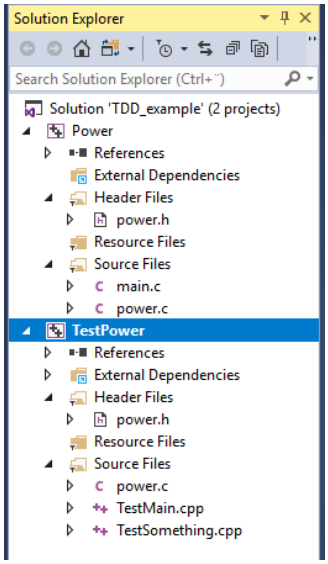


Navigate to the source folder of the files.
Click the path-line and copy the path to the C-project

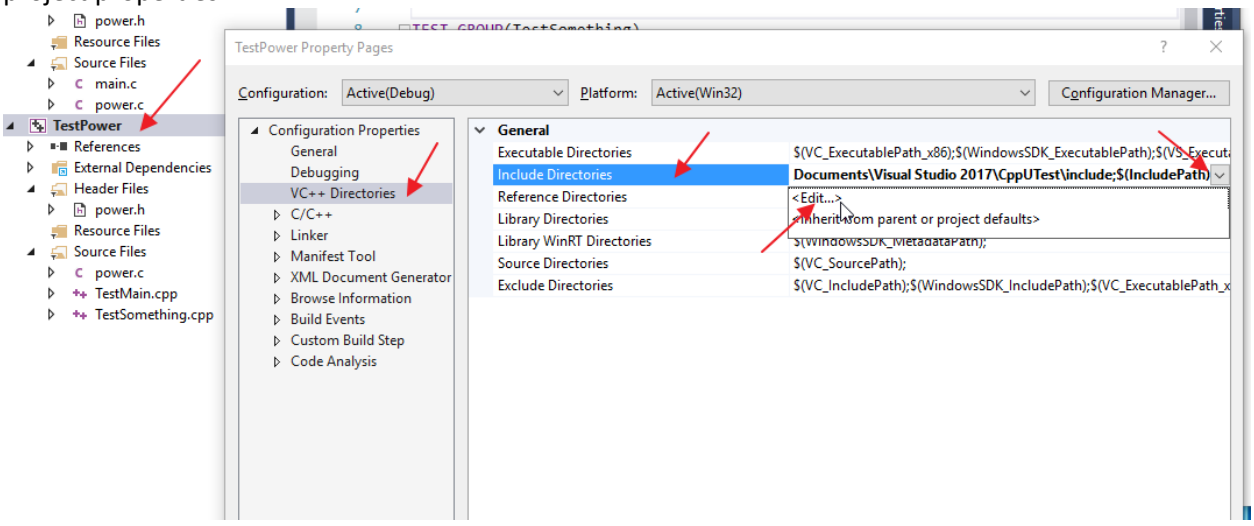


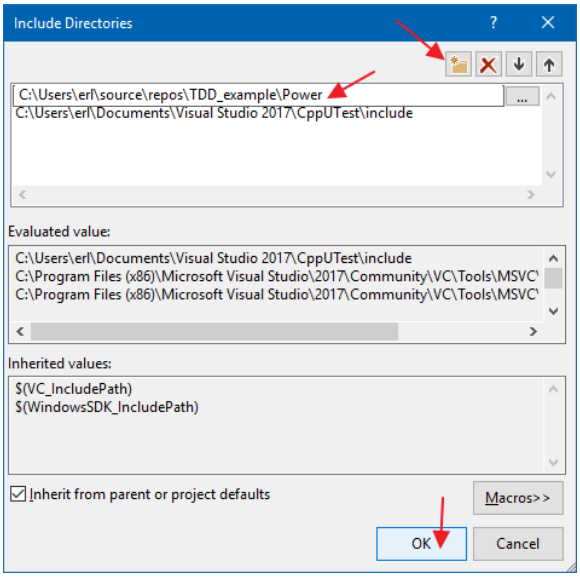
Mark both power.c and power.h and click Add



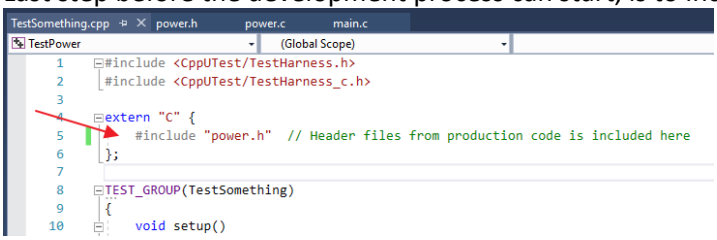


Paste the C-project path into the include directories of the CppUTest project properties.

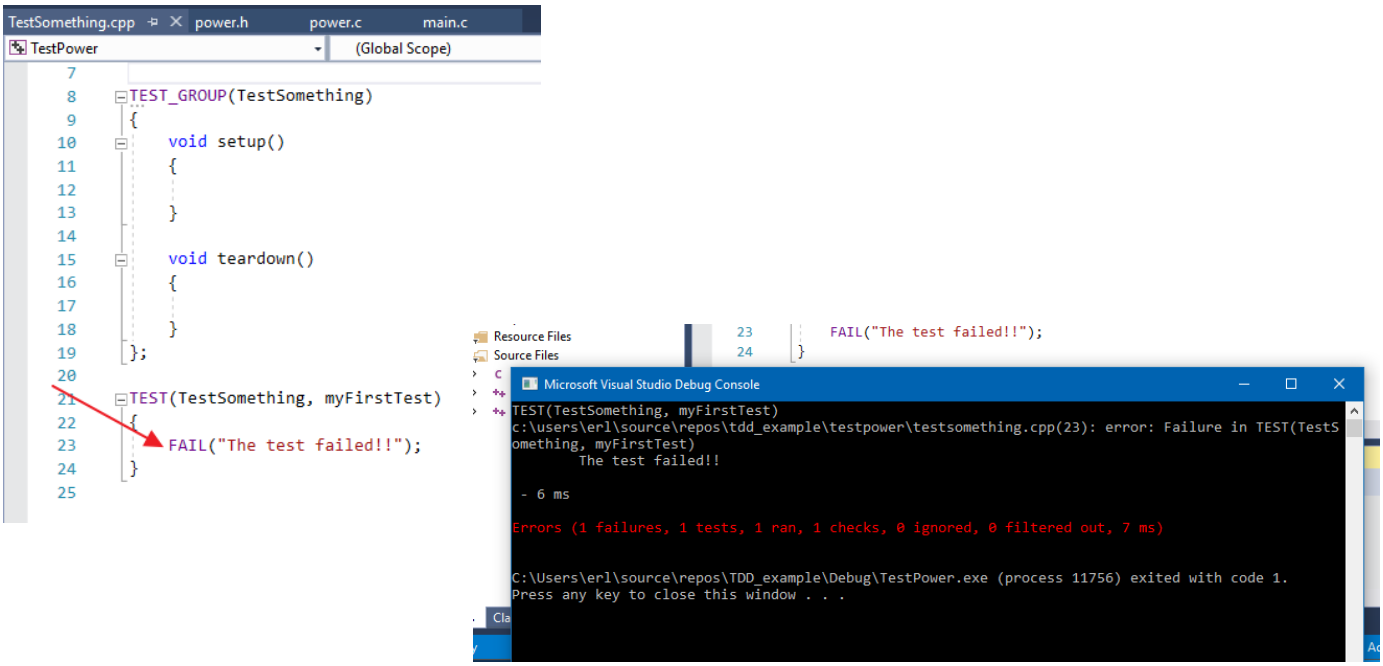


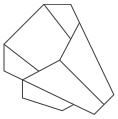


Last step before the development process can start, is to include power.h in the CppUTest project.



There is already one test in the CppUTest project. This test will always fail. Run it with Ctrl-F5 to see the result.





3. Test Driven Development

This example will use TDD to show the process of developing a program module to calculate mathematical powers, e.g. $2^3 = 8$.
The interface for this function is given by the power.h file, that was created above.

```
Int power(int base, int exp);
```

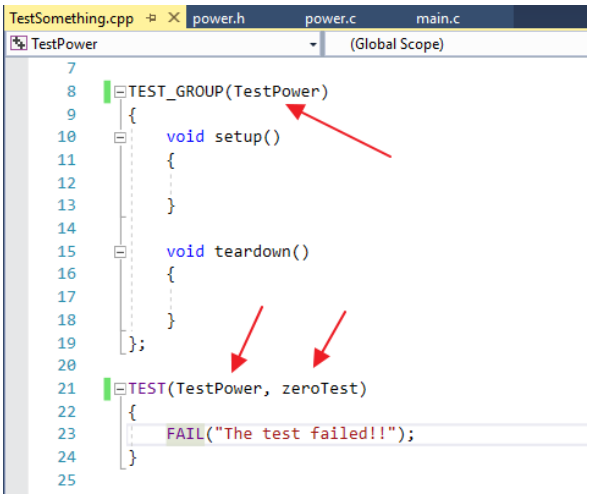
The goal is to add only the necessary code, in small incremental steps and ensure that every feature is fully tested, and at the same time guarantee that refactoring doesn't break the code.

Use ZOMBIES (Grenning, 2019) to make sure that relevant tests are written.

TDD Guided by ZOMBIES

Z: Zero
O: one
M: Many
B: Boundary behaviours
I: Interface definition
E: Exercise Exceptional behaviour
S: Simple Scenarios, Simple Solutions

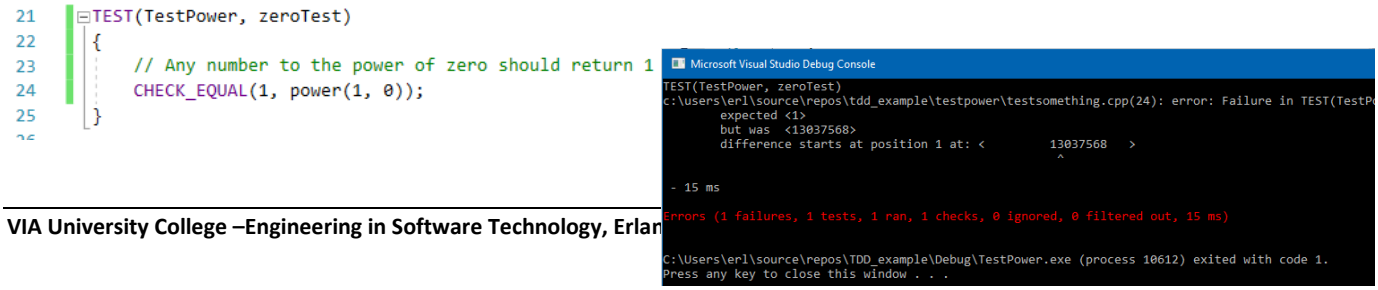
In the "TestSomething.cpp" file, change the TEST_GROUP name to "TestPower", and the TEST to "zeroTest"

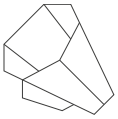


The syntax for writing the unit tests is found in the CppUTest manual.

Zero tests:

The first test should test the Zero condition, so a couple of tests are written to check if the power-function behaves correct when zero is given as exponent.





When run (Ctrl+F5), this test will fail, of course, because no code has been written yet. This is, though, an important step to verify that the test will fail, if the module under test behaves incorrect.
Add sufficient code to pass the test:

```
2   int power(int base, int exp)
3   {
4       return 1;
5   }
```

```
Microsoft Visual Studio Debug Console
TEST(TestPower, zeroTest) - 0 ms

OK (1 tests, 1 ran, 1 checks, 0 ignored, 0 filtered out, 0 ms)


C:\Users\erl\source\repos\TDD_example\Debug\TestPower.exe (proces
Press any key to close this window . . .
```

Green! Success! Sufficient code has been written to implement this feature.

Write another test to check 0^0. This is per definition 1.

```
21  TEST(TestPower, zeroTest)
22  {
23      // Any number to the power of zero should return 1
24      CHECK_EQUAL(1, power(1, 0));
25      CHECK_EQUAL(1, power(0, 0));
26  }
```

Run the test (Ctrl+F5).
Green! Another success!

Next feature. The power-function should return 0 when base is zero, and exponent is > zero (notice that the power-function is not specified to work with negative exponents as it can only return integers, so no need for tests of that)
Write a test.

```
21  TEST(TestPower, zeroTest)
22  {
23      // Any number to the power of zero should return 1
24      CHECK_EQUAL(1, power(1, 0));
25      CHECK_EQUAL(1, power(0, 0));
26
27      // Zero to the power of any positive integer > 0 = 0
28      CHECK_EQUAL(0, power(0, 1));
29  }
```

```
Microsoft Visual Studio Debug Console
TEST(TestPower, zeroTest)
c:\users\erl\source\repos\tdd_example\testpower\testsomething.cpp(28): error: Failure in TEST
        expected <0>
        but was  <1>
        difference starts at position 0 at: <          1          >
                                             ^

 - 15 ms

Errors (1 failures, 1 tests, 1 ran, 3 checks, 0 ignored, 0 filtered out, 15 ms)


C:\Users\erl\source\repos\TDD_example\Debug\TestPower.exe (process 11152) exited with code 1.
Press any key to close this window . . .
```

Red!
It's time for refactoring.

Write code that will satisfy the new requirement.

```
2   int power(int base, int exp)
3   {
4       if (exp == 0)
5       {
6           return 1;
7       }
8       return 0;
9   }
```

```
Microsoft Visual Studio Debug Console
TEST(TestPower, zeroTest) - 0 ms

OK (1 tests, 1 ran, 3 checks, 0 ignored, 0 filtered out, 15 ms)


C:\Users\erl\source\repos\TDD_example\Debug\TestPower.exe (process
Press any key to close this window . . .
```

Green! Yet another success!

One tests

Create a new test, belonging to the same Test Group for testing the behaviour with one-arguments.
Write the test to check 1^1.

```
31  TEST(TestPower, oneTests)
32  {
33      // Any number to the power of 1 is the number itself
34      CHECK_EQUAL(1, power(1, 1));
35  }
```

```
Microsoft Visual Studio Debug Console
TEST(TestPower, oneTests)
c:\users\erl\source\repos\tdd_example\testpower\testsomething.cpp(34): error: Failure in TEST(TestPower, oneTests)
        expected <1>
        but was  <0>
        difference starts at position 0 at: <          0         >
                                            ^


 - 15 ms
TEST(TestPower, zeroTest) - 0 ms

Errors (1 failures, 2 tests, 2 ran, 4 checks, 0 ignored, 0 filtered out, 15 ms)
```

Red! Need to refactor.

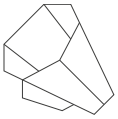
```
2   int power(int base, int exp)
3   {
4       if (exp == 0)
5       {
6           return 1;
7       }
8       return base;
9   }
```

```
Microsoft Visual Studio Debug Console
TEST(TestPower, oneTests) - 0 ms
TEST(TestPower, zeroTest) - 0 ms

OK (2 tests, 2 ran, 4 checks, 0 ignored, 0 filtered out, 17 ms)


C:\Users\erl\source\repos\TDD_example\Debug\TestPower.exe (proce
Press any key to close this window . . .
```

Notice the simplicity of the code – and it already fulfils many of the requirements.

Many tests

Time for testing values greater than zero and one.
Create a new Test Group and write a test to check 2^1.



```
37  TEST(TestPower, manyTests)
38  {
39      // Check the power when base > 1
40      CHECK_EQUAL(2, power(2, 1));
41  }
42
```

```
Microsoft Visual Studio Debug Console
TEST(TestPower, manyTests) - 0 ms
TEST(TestPower, oneTests) - 0 ms
TEST(TestPower, zeroTest) - 0 ms

OK (3 tests, 3 ran, 5 checks, 0 ignored, 0 filtered out, 0 ms)


C:\Users\erl\source\repos\TDD_example\Debug\TestPower.exe (proces
Press any key to close this window . . .
```

Green!

Write a test to check 2^2.

```
37  TEST(TestPower, manyTests)
38  {
39      // Check the power when base > 1
40      CHECK_EQUAL(2, power(2, 1));
41
42      // Check the power when exponent > 1
43      CHECK_EQUAL(4, power(2, 2));
44  }
```

```
Microsoft Visual Studio Debug Console
TEST(TestPower, manyTests)
c:\users\erl\source\repos\tdd_example\testpower\testsomething.cpp(43): error: Failure in TEST(TestPower, manyTests)
        expected <4>
        but was  <2>
        difference starts at position 0 at: <          2         >
                                             ^


 - 15 ms
TEST(TestPower, oneTests) - 0 ms
TEST(TestPower, zeroTest) - 0 ms

Errors (1 failures, 3 tests, 3 ran, 6 checks, 0 ignored, 0 filtered out, 15 ms)
```

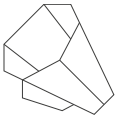
Red!

Need for refactoring.

```
2   int power(int base, int exp)
3   {
4   int result = 1;
5
6       if (exp == 0)
7       {
8           return 1;
9       }
10      while (exp > 0)
11      {
12          result *= base;
13          exp--;
14      }
15      return result;
16  }
```

```
Microsoft Visual Studio Debug Console
TEST(TestPower, manyTests) - 0 ms
TEST(TestPower, oneTests) - 0 ms
TEST(TestPower, zeroTest) - 0 ms

OK (3 tests, 3 ran, 6 checks, 0 ignored, 0 filtered out, 15 ms)


C:\Users\erl\source\repos\TDD_example\Debug\TestPower.exe (process
Press any key to close this window
```

Green! Success.



References

Grenning, J., 2019. *James Grenning's Blog.* [Online]
Available at: http://blog.wingman-sw.com/
[Accessed 21 February 2019].
Wikipedia, 2019. *Test-driven_development.* [Online]
Available at: https://en.wikipedia.org