Yrkes Akademin
Vi hjälper dig att lyckas!

C Programming

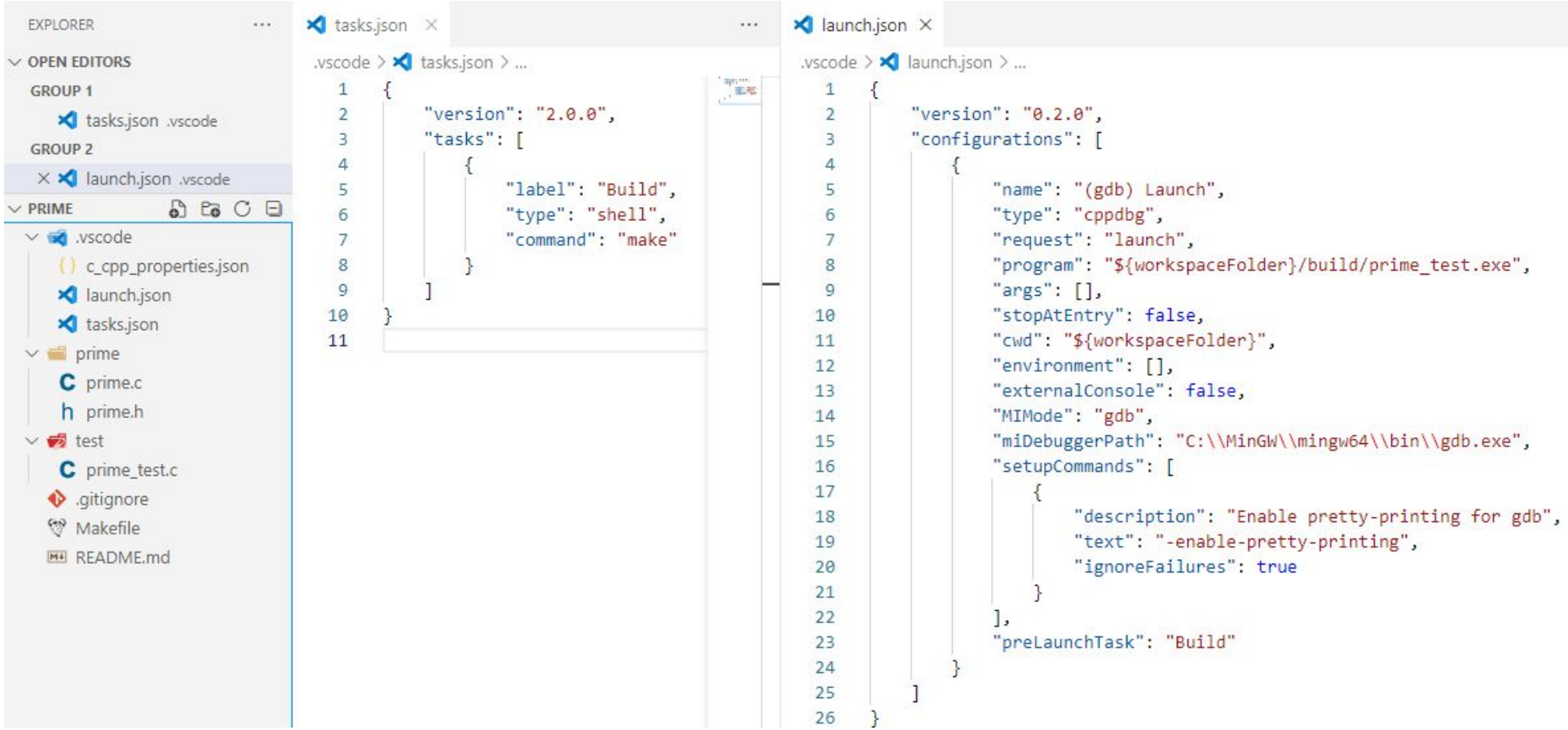
GDB, Unity & TDD

GDB

- Debugging is the process of finding and resolving problems within a program using a debugger.
- **GDB** is the GNU Project debugger which can be used to debug programs written in C language.
- To compile your program for debugging you need to use **-g**. E.g. ***gcc -g main.c -o main***
- You can use **gdb** to debug your program in command line or in Visual studio code.
- To use gdb & gcc to build, debug and run in visual studio code
  - Click on the run icon and then click on the **Run and Debug** button
  - Select C++(GDB/LLDB) in the opened list. And then Select **Default Configuration**
  - A json file (**launch.json**) is opened. Then set the path to your program: "program": "path/to/your/program"
  - Then set the path of the debugger(gdb): "miDebuggerPath": "C:\\MinGW\\mingw64\\bin\\gdb.exe"
  - To make a task to compile & run your program click on **Terminal > Configure Tasks..** in the main menu.
  - Then click on **Create tasks.json file from template** and then **Others Examples to run an arbitrary external command**.
  - The **tasks.json** file is created. Change the label from **echo** to **Build**
  - In the **command** write the shell commands to compile and run your program.
  - Then open **launch.json** and add "preLaunchTask": "Build" to the **configurations**.

Visual Studio Code - GDB

- An example of launch.json and tasks.json



Unity

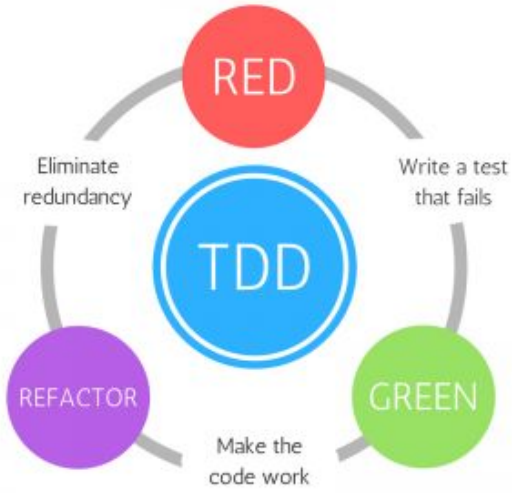
- Unity is a portable unit testing framework written in standard C and supports testing of embedded systems. Unity Test
- Unity uses assertions to test the actual values and the expectations
- Assertions are statements of what we expect to be true about the code under test.
  - E.g. TEST_ASSERT_EQUAL_UINT8(expected, actual);
    - Checks if the expected value and the actual value are equal or not
    - If they are not equal, it means the test is failed
- Unity is used to test modules. Means that the code under test shall be module(s).
  - A module is a source(.c) and a header(.h) file.
  - To test a module you also need to create a test file (.c) and include unity.h
    - The test cases are implemented in this file.

Test-Driven Development (TDD)

- Test-Driven Development is a technique for building software incrementally
  - Small steps help form better components
- A failing test is followed immediately by code satisfying the test
  - Healthy code growth, components are less prone of bugs.
- The focus is on the requirements instead of coding
  - Tests shall be based on the requirements
  - Tests shall enforce a specific behaviour
  - Tests shall encourage refactoring and make components easier to understand
    - Refactoring is the process of restructuring code without changing its external behavior.
- Test automation is key to TDD
  - Testing is automatically performed by machine over and over. No need for manual testing.

Test-Driven Development (TDD)

- Three rules of TDD
  - Write production code only to pass a failing unit test.
  - Write no more of a unit test than sufficient to fail.
    - Compilation failures are failures
  - Write no more production code than necessary to pass the one failing unit test.
- The steps of the TDD cycle
  - Add a small test
  - Run all the tests and see the new one fails
  - Make the small changes needed to pass the failing test
  - Run all the tests and see the new one is passed
  - Refactor to remove duplication and improve expressiveness



Unity - Example

OPEN EDITORS
- prime_test.c test

PRIME
- .vscode
  - c_cpp_properties.json
  - launch.json
  - tasks.json
- prime
  - prime.c
  - prime.h
- test
  - prime_test.c
- .gitignore
- Makefile
- README.md

test > prime_test.c > ...

```c
#include <unity.h>
#include "prime.h"

/**
 * @brief This function is called before running every test function to initialize the environment for the test.
 *        For example if the module needs initialization or to start, we can do it in this function,
 *        or reseting the required variables for the test.
 */
void setUp(void) {}

/**
 * @brief This function is called after running every test function to cleanup the environment.
 *        For example if the module needs to free memory, to be stoped, or etc. we can do it in this function
 */
void tearDown(void) {}

// This is a test function. In these test cases we are going to check for prime numbers
void test_prime_numbers(void)
{
    TEST_ASSERT_TRUE(prime_check(2));
    TEST_ASSERT_TRUE(prime_check(3));
    TEST_ASSERT_TRUE(prime_check(23));
}

// This is another test case. In these test cases we are going to check for non-prime numbers
void test_non_prime_numbers(void)
{
    TEST_ASSERT_FALSE(prime_check(0));
    TEST_ASSERT_FALSE(prime_check(1));
    TEST_ASSERT_FALSE(prime_check(20));
    TEST_ASSERT_FALSE(prime_check(1000));
}

int main(void)
{
    UNITY_BEGIN(); // Start Unity

    // Run test functions
    RUN_TEST(test_prime_numbers);
    RUN_TEST(test_non_prime_numbers);

    return UNITY_END(); // End Unity
}
```