Week 8 – Session 1: Unit Testing

In this laboratory session you are going to learn how to write Unit Tests to test your own code. This is an invaluable and essential technical skill for any programmer to learn.

Challenge 1: Preparation

Before you can roll your sleeves up and start writing some unit tests you need to do some preparation.

Task 1 Get the source code - duration 5 minutes

Take yourself over to the lab solutions in GitHub and fork the solution for StudentModule (in my github) into your own GitHub account.

Task 2 Clone the repo - duration 10 minutes

You can launch GitKraken and create a local clone (on the PC HD) of the repo from your GitHub account. Remember where on the local PC you clone the repo to.

You can also use the built-in Github clone/check-out function available in your Visual Studio. If all else failed, then you can copy the zip folder, extract and open the solution in your Visual Studio (manual method)

Task 3 Review the code - duration 10 minutes

In week four we focused on adding a vector of modules to our student class. This represents a sufficiently complex code base for us to explore unit testing. Take ten minutes to explore the source code in the `Classes Challenges` project.

Challenge 2: Creating and configuring the testing project – duration 15 minutes

You now need to add a testing project (call it `UnitTestModule`) to the Visual Studio solution and configure it for testing. Basic steps include:

- Change the default processor architecture to 64-bit if required.
- Add a reference in the test project to the project you want to test.
- Configure the output of the project being tested to be a static library instead of an executable.
- Add the header files of the classes you are testing to the pch.h header in the test project.

Challenge 3: Testing the module class (`TestingModule`)

Task 1 Make a test plan - duration 10 minutes

Create a "`TestingModule`" namespace. Review the module class and for the following methods consider what data is required to test the methods and how you would test them:

- Module custom constructor
- Get and Set module Code
- Get and set module credit points

Task 2 Test the module class - duration 20 minutes

Write an individual Unit test for the five methods listed above. Remember to lay them out in the format:

```
//Arrange
//Act
//Assert
```

Challenge 4: Test the Student class (`TestingStudent`)

Task 1 Make a test plan - duration 10 minutes

Create a `TestingStudent` namespace. Review the student class and for the following methods consider what data is required to test the methods and how you would test them:

- `Student(std::string name, std::string registration, std::string course, int yearofStudy, std::vector<Module> m)`
- `AddModule(Module m)`
- `AddModule(std::string moduleTitle, std::string moduleCode, int moduleCreditPoints, int moduleMark)`
- `CalculateClassification()`
- `DeleteModule(std::string moduleCode)`

Task 2 Test the module class - duration 30 minutes

Write an individual Unit test for the five methods listed above. Remember to lay them out in the format:

```
//Arrange
//Act
//Assert
```

You may need to write support functions in the Student class to enable testing. E.g., returning the number of modules, a student currently has in their list of modules.