GP1110 – Fundamentals of Programming II

Week 10

- Unit Testing
- Optimization

Software Engineering

- Regression Testing
  - Test any new changes in code.
  - Ensure that changes do not break existing functionality in code.
  - Use existing tests for checking the functionality of new code.

Unit Testing

- Test methods of classes in isolation.
- Elements of incomplete systems can be tested for functionality.
- Ensure compatibility before connecting the elements of a system.

Unit Testing

- Useful for many parts of the software development cycle
  - Debugging
  - Adding new code and functionality
  - Optimization

Automated Unit Testing

- Arrange
  - Set up testing data and code.
- Act
  - Run the unit tests.
- Assert
  - Define expected output.

Automated Unit Testing

- Testing exists as a separate project that can be run for methods.
- Expected results are compared with actual results.

Automated Unit Testing

- Set up code that will test the output of one specific method of a class.
- Prepare all example data that will be required for input into method arguments.
- Write code to compare results.

Automated Unit Testing: Visual Studio C++

- MS Visual Studio features automated unit testing for multiple languages.
  - Includes an interface that runs tests and displays the results.

Automated Unit Testing: Visual Studio C++

- Create a new project within your solution, which will be the unit test.
  - Visual C++ -> Test -> Native Unit Test
  - Name the project so that it can be easily identified as a unit test for a specific class, e.g. UnitTest_MyClass

Automated Unit Testing: Visual Studio C++

- Make sure that the unit testing project depends on the project for testing.
- Right-click on solution “Project Dependencies”.
- Use drop down to select unit test project, check box for main project.

Automated Unit Testing: Visual Studio C++

- Create new filters for both header and source files.
  - Name them in reference to the class, e.g. Header MyClass, Source MyClass
  - Right-click and “Add -> Existing Item” to add the proper source and header files.

Automated Unit Testing: Visual Studio C++

- There will be an auto-generated source file like UnitTest_MyClass.cpp
- Add a line to include the the header file for the class that will be tested.
  - #include "../MyProject/MyClass.h"

Automated Unit Testing: Visual Studio C++

```
namespace UnitTest_MyClass
{
    TEST_CLASS(UnitTest_MyClass)
    {
    public:
        TEST_METHOD(TestMethod_myMethod)
        {
            MyClass myClass;
            Assert::AreEqual(12, myClass.myMethod());
        }
    };
}
```

Automated Unit Testing: Visual Studio C++

- Place all necessary test and example variables in each TEST_METHOD clause
- Use the Assert library to compare the results of calling the method with the expected output for that method.

Automated Unit Testing: Visual Studio C++

- Assert
  - AreEqual(), AreNotEqual(): 2 values
  - AreSame(), AreNotSame(): 2 objects
  - Fail(): Forces a test fail for the block.
  - IsTrue(), isFalse(); boolean test
  - IsNull(), isNotNull(); Null value test

Automated Unit Testing: Visual Studio C++

- Use the Test Explorer to run tests.
  - Test -> Windows -> Test Explorer
  - The unit tests can be run together, or they can be run separately.
  - Latest results will be displayed in the Test Explorer window.

Software Optimization

- Re-evaluate and re-write code for the purpose of improving how well the program works.
- Optimization is towards a specific goal
  - There are many different goals.

Software Optimization

- Common Goals
  - RAM usage
  - CPU and GPU utilization
  - Minimize writing to secondary storage
  - Power usage
  - Stability

Levels of Optimization

- Design
  - Architecture and structure.
  - Choice of technology; including hardware, programming languages, OS, etc.
- Algorithms and Data Structures
  - Efficient and avoids unnecessary work.
  - Use in context of expected data and use.

Levels of Optimization

- Source Code and Assembly Code
  - Most compilers automatically optimize.
- Build and Compile
  - Certain directives and build flags can affect how it runs on hardware.
  - Can optimize build for specific processors and hardware specifications.

Software Optimization

- Utilize built-in tools that monitor and record CPU/GPU and memory usage.
  - Memory Diagnostic Tools
  - Automated Unit Testing run times

Software Optimization

- Consider loops and use of variables on a function stack and pointers.
- Balance optimization with having code that is easy to read.

Homework

- Develop unit tests for 2 classes.
- Each class should have tests for at least 3 methods.