**Thema, Ziele:** Testen, Unit-Testing mit CppUnit

**Tipp:** Um CpppUnit auf Ihrem eigenen Rechner zu verwenden, müssen Sie das entsprechende Package "CppUnit" (Cygwin: Bereich "Devel") installieren.

## Aufgabe 1: Das Unit-Testing-Framework "CppUnit" in Betrieb nehmen

### 1. Das CppUnit Demo-Programm ausführen

Auf dem Skript-Server finden Sie im Verzeichnis dieser Übung den Ordner "Vorlage_ueb05_cppunit" mit einem Programm ähnlich dem in der Vorlesung vorgestellten Programm zum Thema CppUnit.

Kopieren Sie den Ordner "Vorlage_ueb05_cppunit" in Ihr Arbeitsverzeichnis (U-Drive).

Wechseln Sie nun in der Cygwin-Shell in Ihr Verzeichnis "test" von "Vorlage_ueb05_cppunit". (Dazu können Sie den Befehl "cd …" verwenden. Der Befehl "pwd" zeigt Ihnen, in welchem Verzeichnis Sie sich aktuell befinden).

Geben Sie nun den Befehl "make" ein, um das im Unterverzeichnis "test" enthaltene Make-File auszuführen. Als Folge werden die Source-Files kompiliert und gelinkt. Dabei entstehen im Unterverzeichnis "bin" die zwei ausführbaren Dateien "testmain.exe" und "testmain2.exe".

Führen Sie diese mit Hilfe des Befehls "`../bin/testmain.exe`" bzw. "`../bin/testmain2.exe`" aus.

### 2. (Freiwillig) UnitTest in Eclipse

Sie können Ihr Test-Projekt auch in Eclipse ausführen. Vorgehen:

- Erstellen Sie ein neues leeres C++-Projekt.
- Importieren Sie die Quelldateien (File → Import → General → File-System, navigieren Sie zum Projekt → OK; selektieren Sie src und test-Ordner).
- Eclipse kann in einem Projekt nur ein Hauptprogramm erzeugen. Entscheiden Sie sich für eines der beiden testmain-Dateien. Schliessen Sie die andere sowie "appmain.cpp“ (im src-Ordner) vom Build aus: Kontextmenü der Datei → Resource Configurations → Exclude from Build…, Select All → OK.
- Sorgen Sie dafür, dass der Link-Vorgang die CppUnit-Bibliothek findet: Kontext-Menü des Projekts → Properties; C/C++-Build, Settings, Tool Settings, Cygwin C++ Linker, Libraries; Libraries: Add: `cppunit`

### 3. Das CppUnit-Demo-Programm modifizieren

a) In der Klasse "Author" ist der Vergleichsoperator == bereits vollständig implementiert. Die Implementation dieses Operators ist aber fehlerhaft.
   - Ergänzen Sie nun die Klasse "AuthorTest" mit einem entsprechenden Test für diesen Operator, bei dem auch der Fehler aufgedeckt wird. Korrigieren Sie den Fehler noch nicht!
   - Korrigieren Sie erst danach den Fehler in der Klasse `Author`.

b) Erweitern Sie die Klasse "Book" um den *Stub* für den Vergleichsoperator == (noch nicht implementieren!). Ergänzen Sie danach die Klasse "BookTest" um einen möglichst vollständigen Unit-Test für diesen Operator (stören Sie sich vorerst nicht daran, dass diese Tests fehlschlagen.)

   Implementieren Sie nun zum Schluss den Vergleichsoperator für "Book" und führen Sie die Unit-Tests aus. Die Tests sollten nun nicht mehr fehlschlagen.

## Aufgabe 2: Repetitions- und Verständnisfragen zum Testen allgemein

a) Erklären Sie, den Unterschied zwischen Verifikation und Validierung.

b) Erklären Sie, den Unterschied zwischen Black-Box-Test und White-Box-Tests.

c) Erklären Sie, was man unter dem "Test-First" Prinzip versteht.

d) Erklären Sie, anhand eines *eigenen* Beispiels, was man unter Äquivalenzklassen versteht.

## Aufgabe 3: Repetitions- und Verständnisfragen zu Unit-Testing

a) Beim Programm von Aufgabe 1 haben Sie zum Testen Methoden geschrieben, die Sie nicht selber aufrufen. Fragen:
   1. Wer ruft diese Funktionen auf?
   2. Woher weiss dieser Programmteil überhaupt, welche Methoden er aufrufen soll?
   3. Wie nennt man dieses Prinzip?
   4. Erklären Sie, warum man dieses Prinzip wohl so nennt.

b) Zwei Programmierer arbeiten am gleichen Projekt. Programmierer A benutzt konsequent Unit-Testing, Programmierer B findet das überflüssig und meint es genüge, sein Programm zwischendurch auszuprobieren ohne zusätzlichen Testcode zu schreiben. Das Ende des Projektes naht.

   Frage: Was sind wohl die Erfahrungen der beiden Programmierer. Welcher hat wohl mehr Stress?

## Aufgabe 4: "MyDate" testen mit "CppUnit"

Verwenden Sie CppUnit um Ihre Klasse "MyDate" zu testen.

**Vorgehen:**

1. Kopieren Sie dazu Ihre `MyDate`-Dateien in eines neues Verzeichnis, welches die gleiche Dateistruktur wie "Vorlage_ueb05_cppunit" aufweist (inkl. Unterverzeichnisse: `src`, `test`).
2. Übernehmen Sie die Dateien des test-Verzeichnis‘ von "Vorlage_ueb05_cppunit" und passen Sie das Makefile ("test/GNUmakefile") an.
3. Erstellen Sie eine Klasse "MyDateTest" welche CppUnit verwendet (z.B analog "BookTest.cpp") und implementieren Sie die Testklasse mit sinnvollen Testmethoden. Hierzu können sie evt. Ihr früheres Testprogramm modifizieren oder zumindest Teile davon verwenden.

## Aufgabe 5: Spezielle Tests (Verständnisfrage)

Moderne Rechnersysteme, insbesondere solche aus dem Embedded-Bereich müssen vielfältige Aufgaben erfüllen und müssen demzufolge entsprechende Tests durchlaufen.

Siehe http://www.youtube.com/watch?v=KvLoUNlwNBM .

Frage: Um was für eine Art von Test handelt es sich hier?

## Some Kind of Unit-Testing