




Softwareentwicklung
Praktikum

[Tutorium]

vision



& [Gruppe 16, 17]



Themen fuer jetzt


- Themen
- Ex1
- Testen
- Valgrind
- pipes
- Fragen

- Ex 1
- Testen, Testen, Testen
- Makefiles



**Naechstes Tutorium:
31.03.2009
reine Fragestunde!**

vision



Ex 1 - Tipps


- Themen
- Ex1
- Testen
- Valgrind
- pipes
- Fragen

- Hinweise:
 - <http://tuogl.tugraz.at/52225/files/116/804/Kommentar+DDD+Phase+1.pdf>
- DDD muss verwendet werden
 - Loesungsvorschlag umsetzen
 - so programmiert wird wie im DDD
 - Aktuelle Version verwenden!

Abgabe (02.04.2009)

- Code, Makefile -> example1.zip/tar.gz
- Workload.txt, example1.pdf

vision




Optimiert testen

- Themen
- Ex1
- Testen**
- Valgrind
- pipes
- Fragen

- Step – by Step
 - z.B.: mit gdb & Code::blocks
 - --> visueller Programmablauf
 - --> kleine Ueberpruefungen
 - --> Einzelfaelle
- Testcases
 - selber schreiben; Vergleiche
 - --> ein kompletter Programmdurchlauf
 - --> wie wir testen
 - --> Newsgroup esp!

vision




Optimiert testen

- Themen
- Ex1
- Testen**
- Valgrind
- pipes
- Fragen

- Cpp Unit Test
 - Testen einzelne Funktionalitaet
 - Testen automatisiert
 - Testen mit Golden Device, vorberechneten Werten, ...
 - <http://cppunit.sourceforge.net/cppunit-wiki>
 - <http://beans.seartipy.com/2007/11/26/unit-testing-c-programs-using-cppunit-in-eclipse-ide-on-windows/>

vision



Optimiert Testen

- Themen
- Ex1
- Testen**
- Valgrind
- pipes
- Fragen

- Programm durchdenken, an einem Beispiel: tut es auch wirklich das, was es tun soll
- Debug-Ausgaben
 - Welche Werte haben die Variablen zu bestimmten Zeitpunkten im Programm
- Valgrind
 - Von großem Vorteil bei zufälligem Verhalten
 - Wichtig: Valgrind-Fehler sind immer(!) Fehler im Programm (auch wenn alles funktioniert) – also ernstnehmen!

vision

Optimiert Testen

- Programm in sinnvolle Methoden / Klassen aufteilen
- Keine Variablen oder Parameter recyceln
- Sinnvolle Kommentare schreiben
- Variablen sprechende Namen geben
- Pointer nach Freigabe auf NULL setzen
- Parameterliste im Konstruktor verwenden
- Destruktor für Freigabe von Membern nützen
- (alle) Variablen initialisieren
- Kein Goto verwenden
- Immer { ... } machen, außer man weiß was man tut
- Condingstandard beachten

Optimiert Testen



Current release: [valgrind-3.3.0](http://valgrind.org/)

Valgrind

<http://www.cprogramming.com/debugging/valgrind.html>

- Compilerflags fuer gcc
 - -Wall alle Fehler
 - -o name des binary
 - -g debug informationen
 - -fstack-protector-all prueft statische Arraygrenzen
- Optionen fuer Valgrind
 - --tool=memcheck aktiviert die Speicherueberwachung
 - --leak-check=yes prueft auf speicherloecher

Linux Pipes

- Themen
- Ex1
- Testen
- Valgrind
- pipes**
- Fragen

- `./program > log.txt`
 - Schreibt den Output des Programms in eine Datei
- `./program 1>log.txt 2>error.log`
 - Schreibt den stdout des Programms nach log.txt und den stderr nach error.log
- `valgrind --tool=memcheck --leak-check=yes ./troubles 1>log.txt 2>error.log`

vision

Linux Pipes

- Themen
- Ex1
- Testen
- Valgrind
- pipes**
- Fragen

- `./program < input.txt`
 - Uebergibt den Inhalt von input.txt an das Programm
- `./own_solution < sampleTxt >log.txt`
- `./rsf_solution < sampleTxt >log2.txt`
- Differ
 - Vergleicht 2 Dateien ob sie sich unterscheiden
 - `Diff [datei1] [datei2]`
 - `Diff log.txt log2.txt`

vision

Fragen

- Themen
- Ex1
- Testen
- Valgrind
- pipes**
- Fragen**

? Alle Klarheiten beseitigt ?



vision
