

Programmieren in C++

SS 2019

Vorlesung 1, Dienstag 23. April 2019
(kickoff, admin, ein erstes Programm)

Prof. Dr. Peter Thiemann
Programmiersprachen
Institut für Informatik
Universität Freiburg

Die heutige Vorlesung

■ Organisatorisches

- Ablauf Vorlesungen, Übungen, Projekt
- Prüfungstechnisches Punkte, Note, ECTS & Aufwand
- Art der Vorlesung Voraussetzungen, Lernziel, Stil

■ Inhalt

- Erstes Programm Approximation von Pi
mit allem Drumherum Kompilieren, Unit Test, Checkstyle
Makefile, SVN, Jenkins

Übungsblatt 1: Approximation des Goldenen Schnitts

■ Vorlesungen

- Jeden Dienstag von 10.15 - 11.45 Uhr im HS 026
 - Nicht am 11.6. (Pfingstpause)
 - Insgesamt 13 Termine
 - Livestream und Aufzeichnung
 - Folien + Audio
 - Technik & Schnitt: Maximilian Nazarati
- Assistent der Vorlesung: Philipp Krause
- Alle Kursmaterialien auf der Webseite:
 - Aufzeichnungen (Links), Folien, Übungsblätter, Code aus der Vorlesung + evtl. zusätzliche Hinweise, Musterlösungen

■ Übungsblätter

- Die Übungen sind der wichtigste Teil der Veranstaltung
- Jede Woche ein Übungsblatt, insgesamt **11**
- Abgabe bis zur nächsten Vorlesung
- **Selber** machen!

Sie können gerne zusammen über die Übungsblätter nachdenken, diskutieren, etc. ... aber die Programme müssen Sie zu **100%** selber schreiben

Auch das teilweise Übernehmen gilt als Täuschungsversuch

■ Das Projekt

- Am Ende der Veranstaltung gibt es eine etwas größere Programmieraufgabe
- Umfang ca. 4 Übungsblätter
- Weniger Vorgaben als bei den Übungsblättern
- Fängt schon zwei Wochen vor Vorlesungsende an

■ "Übungsgruppen"

- Sie bekommen jede Woche Feedback zu Ihren Abgaben

Von Ihrer*m Tutor*in über SVN, siehe Folie 25

- Für Fragen aller Art gibt es ein **Forum**

Siehe Link auf dem Wiki + mehr dazu auf Folie 27

- Fragestunden oder nach der Vorlesung ansprechen
- Bei Problemen, die sich nicht über das Forum lösen lassen, können Sie Ihren Tutor um ein persönliches Treffen bitten

■ Punkte

- Sie bekommen wunderschöne Punkte, maximal 16 pro Übungsblatt, das sind maximal 176 Punkte für Ü0 – Ü10
- Für das Projekt gibt es maximal 80 Punkte
- Macht insgesamt 256 Punkte
- Für das Ausfüllen des Evaluationsbogens am Ende gibt es +16 Punkte

■ Gesamtnote

- Die ergibt sich linear aus der Gesamtpunktzahl am Ende

128 – 139: 4.0; 140 – 151: 3.7; 152 – 163: 3.3

164 – 175: 3.0; 176 – 187: 2.7; 188 – 199: 2.3

200 – 211: 2.0; 212 – 223: 1.7; 224 – 235: 1.3

336 –: 1.0


- **Außerdem:** Zum Bestehen müssen mindestens 88 Punkte in den Übungen (Ü0 – Ü10) und mindestens 40 Punkte im Projekt erreicht werden
- **Außerdem 2:** Sie müssen sich mindestens einmal mit Ihrem Tutor / Ihrer Tutorin treffen, dazu mehr in einer der späteren Vorlesungen

■ ECTS Punkte und Aufwand

- Informatik / ESE / MST / BOK: 6 ECTS Punkte
- Das sind $6 \times 30 = 180$ Arbeitsstunden insgesamt
- Davon 120 Stunden für die ersten 11 Vorlesungen + Übungen
 - also etwa 10 Stunden Arbeit / Woche
 - also etwa 8 Stunden pro Übungsblatt
- Bleiben 60 Stunden für das Projekt und die dazugehörigen (letzten beiden) Vorlesungen + Übungen

Wer schon Vorkenntnisse hat, wird weniger Arbeit haben

■ Voraussetzungen

- Sie wissen schon, wie Programmieren "im Prinzip" abläuft, z.B.:
 - die Zahlen von 1 bis 10 ausgeben
 - berechnen, ob eine gegebene Zahl  ist
 - Space shuttle flight system...
- Verständnis einiger Grundkonzepte
 - Variablen, Konditionale, Schleifen, Funktionen

■ Wenn Ihnen das alles gar nichts sagt

- ... können Sie trotzdem mitmachen, es wird aber dann mehr Arbeit für Sie, als es den ECTS Punkten entspricht

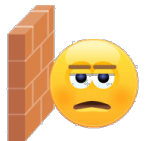
■ Lernziel

- Programmieren in C/C++ nach den Regeln der Kunst
... im Umfang von 500 – 1000 Zeilen
- Gute **Struktur**, gute **Namen**, gute **Dokumentation**
- Verwendung von Standardtools und –techniken:
 - Unit Tests `assert, catch 2`
 - Styler `cpplint.py`
 - Build System `make und Jenkins`
 - Versionskontrolle `SVN`

Art der Vorlesung 3/6

- Die Vorlesung ist eher **breit** als **tief**
 - Die Aufgaben sind eher einfach
 - Die Konstrukte / Konzepte die wir verwenden auch
 - Sie sollen vor allem lernen **guten Code** zu schreiben
 - Mit allem Drum und Dran
 - Das ist nämlich genau das, was viele Leute, die es sich selber beibringen oder beigebracht haben, nicht können
 - Mit dem richtigen Drumherum macht Programmieren viel mehr Spaß

Sonst wird viel Zeit mit Debuggen verschwendet, die besser ins Programmieren investiert wäre



■ Art der Vermittlung

- Live Programmierung; no magic
- Details:
 - Siehe Dokumentation jeweils letzte Folie jeder Vorlesung
 - Google und stackoverflow sind deine Freunde
- Ich werde vor allem immer das erklären, was Sie auch gerade brauchen (für das jeweilige Übungsblatt)

■ Fragen, Fragen, Fragen

- Selber ausprobieren, aber nicht zu lange, dann fragen!
Hier in der Vorlesung oder über das Forum → Folie 27

■ Ansatz "low-level"

- Linux, Kommandozeile, Texteditor, Makefile
- Aufwändige Entwicklungsumgebungen / IDEs (z.B. Eclipse oder NetBeans) sind was für Fortgeschrittene, die den "low level" beherrschen

Wer sich auskennt, kann eine IDE benutzen, die Sachen müssen aber trotzdem "low-level" in unser SVN hochgeladen werden (insbesondere mit Makefile)

Das ist ggf. mehr Arbeit

■ Linux

- Sie können Ihren eigenen Linux-Rechner verwenden
- Oder auf einem der Pool-Rechner arbeiten
- Für Teilnehmer aus anderen Universen steht auf der Webseite ein **Linux-Image** zur Verfügung

Da sind g++, svn, gtest und alles, was für diese Vorlesung notwendig ist, vorinstalliert

- Wer unbedingt Windows verwenden will:

Windows 10: hat eine integrierte Linux-Shell (Ubuntu)

Windows 8.0: Cygwin (auf eigene Gefahr)

Windows 3.0: **sie haben da irgendwas verpasst**

Ein erstes Programm 1/12

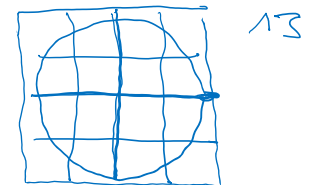
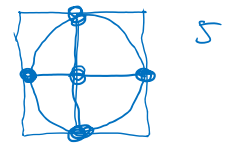
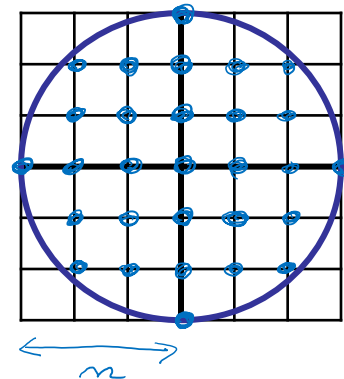
- Wir schauen uns ein einfaches Problem an
- Wir werden es **live** in ein Programm umsetzen
- Es kommt weniger auf das Problem an, als auf das ganze **Drumherum**
- Genau wie bei der Bearbeitung des Übungsblatts

■ Approximieren der Zahl **π** über die Kreismethode

Berechne die Anzahl k aller ganzzahligen Gitterpunkte (x, y) mit $|x|, |y| \leq n$, die innerhalb des Kreises mit Radius n um den Ursprung liegen (also $x^2 + y^2 \leq n^2$)

Für $n \rightarrow \infty$ entspricht der Anteil dieser Gitterpunkte k/n^2 dem Verhältnis der Fläche des Kreises ($\pi \cdot n^2$) zur Fläche des Gitters ($4n^2$) ... also $\pi / 4$

$n = 3$
im Kreis : 29



- Version 1

- Eine Textdatei für den gesamten Programmcode

`ApproximationOfPi.cpp`

- Sie können dafür einen beliebigen Texteditor verwenden

■ Kompilieren

- Wir benutzen den Gnu C++ Compiler, kurz **g++**

```
g++ -o ApproximationOfPi ApproximationOfPi.cpp
```

Ich benutze in der Vorlesung die g++ **Version 5.4.0** ...
das Linux Image (auf dem Wiki) ebenso

Ältere oder experimentelle Versionen auf eigene Gefahr!

- Der Befehl oben erzeugt Maschinencode, der wie folgt ausgeführt werden kann

```
./ApproximationOfPi
```

Ohne die `-o` Option würde das ausführbare Programm einfach "a.out" heißen, was jetzt nicht so eingängig ist

■ Unser Programm, Version 2

- Wir schreiben jetzt zwei Dateien (warum sehen wir gleich)

ApproximationOfPi.cpp

ApproximationOfPiMain.cpp

- Die erste Datei enthält nur die Funktion
- Die zweite Datei enthält das restliche Programm und liest zu Beginn die erste Datei ein

```
#include "./ApproximationOfPi.cpp"
```

In Vorlesung 2 werden wir sehen, dass das nicht optimal ist, aber für heute (und das Ü0) ist das völlig in Ordnung

■ Unit Tests

- Wir schreiben jetzt noch eine dritte Datei

`ApproximationOfPiTest.cpp`

- Diese enthält eine Funktion, die unsere Funktion testet und ein generisches main, das einfach alle Tests ausführt

```
#include <gtest/gtest.h>
```

```
#include "./ApproximationOfPi.cpp"
```

```
TEST(ApproximationOfPi, approximatePi) { ... }
```

```
int main(int argc, char** argv) {  
    ::testing::InitGoogleTest(&argc, argv);  
    return RUN_ALL_TESTS();  
}
```

■ Überprüfung des Stils

- Bisher haben wir "nur" versucht, unser Programm zum Laufen zu bringen
- Selbstverständlich sollte unser Code auch schön und für anderen Menschen gut lesbar sein
- Es gibt extra Programm dafür, wir benutzen **cpplint.py**
- Um damit den Stil aller .cpp Dateien zu überprüfen, können wir einfach schreiben

```
python cpplint.py *.cpp
```

- Liefert sehr detaillierte und in aller Regel selbsterklärende Fehlermeldungen zum Stil des Codes

■ Makefile und make

- Woher wissen andere, wie sie unseren Code kompilieren, testen, oder seinen Stil überprüfen können?
- Wir schreiben eine Datei **Makefile** mit folgender Syntax

```
<target>:  
    <Befehl 1>  
    <Befehl 2>  
    ...
```

Achtung: jede der Befehlszeilen muss mit einem TAB anfangen!

- **make <target>** in dem Verzeichnis, in dem diese Datei steht, bewirkt die Ausführung der entsprechenden Befehle

make kann noch viel mehr... mehr dazu in den nächsten Vorlesungen

■ Daphne

- Daphne ist unser Kursverwaltungssystem, das es mir, den Tutoren, und hoffentlich auch Ihnen das Leben leichter macht
 - **Registrieren Sie sich bitte** nach der Vorlesung dort
 - Sie kriegen dann automatisch Zugang zu
SVN, Jenkins, Forum ... siehe nächste drei Folien
- Diese Subsysteme sind per se unabhängig von Daphne und in Daphne "nur" übersichtlich zusammengefasst
- Es läuft alles über einen Username + Passwort, nämlich das vom RZ, das Sie auch für Anmeldungen etc. nutzen

■ SVN (Subversion)

- Ein SVN Repository ist einfach ein Verzeichnisbaum mit Dateien, die bei uns zentral auf einem Rechner liegen
- Jeder, der sich (via Daphne) bei uns registriert, hat ein Unterverzeichnis dort → [URL](#) siehe Ihre Daphne-Seite
- Sie bekommen eine Kopie dieses Verzeichnisses mit
`svn checkout <URL> --username=<Ihr RZ Username>`
- In Ihrer Arbeitskopie können Sie dann Sachen ändern, Unterordner und Dateien hinzufügen, etc.
 - `svn add <file name>` fügt eine Datei erstmals hinzu
 - `svn commit <file name>` lädt die Änderungen zu uns hoch

■ Jenkins

- Mit Jenkins können Sie überprüfen, ob die Version Ihres Codes, die Sie zu uns hochgeladen haben, auch funktioniert
- Jenkins schaut dazu einfach nach dem Makefile und führt dann die folgenden Befehle aus

`make clean` Löscht Nebenprodukte

`make compile` Kompiliert ihren Code

`make test` Führt die Unit Tests aus

`make checkstyle` Prüft den Stil Ihres Codes

- Sie bekommen dann angezeigt, ob es funktioniert hat, und auf Wunsch auch die komplette Ausgabe

■ Forum

- Machen Sie bitte regen Gebrauch davon und haben Sie **keine Hemmungen**, Fragen zu stellen

Insbesondere wenn Sie bei einem Fehler mit eigenem Nachdenken und Google und Co nicht weiterkommen

Gerade in C++ können Kleinigkeiten zu stundenlangem Suchen führen, was dann sehr frustrierend ist

- Geben Sie sich aber gleichzeitig Mühe, Ihre Fragen möglichst konkret und genau zu stellen

Eine kurze Anleitung dazu findet sich auf Seite 3 des Ü1, eine etwas längere Anleitung auf dem Wiki

■ Arbeitsaufwand

- Sie haben **eine Woche** Zeit
- Die meiste Arbeit wird das Drumherum sein
- Fangen Sie deswegen bitte **rechtzeitig** an und fragen Sie auf dem Forum, wenn Sie nicht weiterkommen
- Fragestunden:

■ SVN

- <http://subversion.apache.org/>
- Kurze Einführung dazu auf dem Wiki

■ C++

- <http://www.cplusplus.com/doc/tutorial/>
- Am Anfang nur elementare Sachen