

Tafelübung 01

Algorithmen und Datenstrukturen

Lehrstuhl für Informatik 2 (Programmiersysteme)

Friedrich-Alexander-Universität Erlangen-Nürnberg

Wintersemester 2016/2017

Lehrstuhl Informatik 2
Programmiersysteme



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Übersicht

Organisatorisches

- Allgemeines
- Übungsbetrieb
- Übungsblätter und -schein
- Plagiarismus
- CipMap
- Linux-Install-Party

Eine kurze Einführung in Java

- Workflow
- Häufige Fehler

Zahlensysteme

- Motivation
- Zahlen in verschiedenen Basen
- Zwischen Zahlensystemen
umrechnen
- Negative Zahlen
- Entscheidungsgehalt

Organisatorisches

Lehrstuhl Informatik 2
Programmiersysteme



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Wer sind wir?

Wir...

Name AuD-Team, Lehrstuhl für Informatik 2 (Programmiersysteme)

E-Mail aud@i2.cs.fau.de

Ich...

Name TODO

Studium TODO

E-Mail TODO

Folien (und sonstige Materialien)

- zu finden unter <https://www2.cs.fau.de/aud/uebungen>
- Benutzername und Passwort: AUDFAU

Wer seid ihr?

Wer seid ihr?

Studiengang?

Programmier-Erfahrung?

Wer braucht noch Schein/Klausur/beides?

Information und Materialien

- alle **Neuigkeiten** und **Unterlagen** zu AuD:
 - <https://www2.cs.fau.de/aud>
 - Benutzername und Passwort: **AUDFAU**
- Forum der **FSI Informatik**:
 - <https://fsi.informatik.uni-erlangen.de/forum/>
 - viel größerer Leserkreis als diverse Facebook-Gruppen!
 - auch Dozenten und Tutoren lesen mit und beantworten Fragen!
- Abgabe der Übungsaufgaben über das **Exercise Submission Tool (EST)**:
 - <https://est.cs.fau.de/>

Wichtig

Unbedingt **Übungsblatt 0** ("*Wichtige Hinweise*") lesen und beachten!

Ziel des Übungsbetriebs

- der Übungsbetrieb besteht aus mehreren Übungsangeboten:
 - Tafelübungen
 - Rechnerübungen
 - [Intensivübung]
 - [Klausurvorbereitungskurs]
- die Übungen sollen den in der Vorlesung vermittelten **Stoff vertiefen**
 - ~> Konzepte aus der Vorlesung anwenden
 - ~> **Programmiererfahrung** sammeln

Tafelübungen

- Inhalte der **Tafelübungen**:
 - **Wiederholung** des aktuellen Vorlesungsstoffs anhand von **Beispielen**
 - ggf. Besprechung der **Lösungen** des alten Übungsblatts
 - bei besonderen Wünschen \leadsto E-Mail an den Tutor (rechtzeitig!)
 - ggf. **Lösungs-Hinweise** zum aktuellen Übungsblatt

Anmeldung

Jeder von euch muss für eine (*diese!*) Tafelübung im EST angemeldet sein!

Hinweis

Keine Anwesenheitspflicht in den Tafelübungen, aber Teilnahme dringend empfohlen!

Rechnerübungen

- Rechnerübungen bieten individuelle Hilfe bei den Übungsaufgaben
- Rechnerübungen finden in den Rechnerräumen (CIP-Pools) statt
 - die Tutoren helfen bei Problemen gerne weiter 😊
 - **aber:** die Tutoren programmieren keine vollständige Lösung!
 - nicht erst in der Rechnerübung mit den Aufgaben anfangen!
 - erst selbst probieren, bei konkreten Problemen melden

Hinweis

Die Rechnerübungen sind freiwillig und eine Anmeldung ist *nicht* notwendig. Jeder von euch kann beliebig viele Rechnerübungen bei beliebigen Tutoren besuchen.

Intensivübung

- die **Intensivübung** ist ein zusätzliches Angebot für:
 - Programmier-Neulinge, die Probleme bei den Übungsaufgaben haben
 - AuD-Hörer, die Schwierigkeiten mit dem Stoff haben
- Termin wird nach **Bedarf** und **Vereinbarung** zu Semesterbeginn festgelegt
 - für genauere Informationen AuD-Seite regelmäßig besuchen!

Wichtig

Wenn ihr merkt, dass ihr Schwierigkeiten mit dem Stoff habt, besucht bitte **rechtzeitig** Rechner- und Intensivübung und wartet nicht, bis es schon zu spät ist!

Übungsblätter

- jeweils **Freitags** gibt es ein neues **Übungsblatt** mit neuen Aufgaben
 - zu finden auf der AuD-Seite und über das EST
 - für die Bearbeitung habt ihr i.d.R. **10 Tage** Zeit (+ evtl. Feiertage)
- die Übungsblätter bestehen aus...
 - Praxis-Aufgaben \Leftrightarrow Theorie-Aufgaben
 - Einzel-Aufgaben \Leftrightarrow Gruppen-Aufgaben

Abgabe der Lösungen

- die Abgabe eurer Lösungen erfolgt **ausschließlich via EST**
 - wir nehmen keine Lösungen per Mail o.ä. an!
- Abgabe jeweils **spätestens (!)** bis **Montag, 10:00 Uhr**
 - Achtung: Server können auch mal **ausfallen...** (siehe letztes jedes Semester)
 - das EST speichert jeweils **letzte Abgabe** (\leadsto „Zwischenstände“ hochladen)

Theorie-Aufgaben

- Abgabe der Theorie-Aufgaben ausschließlich als PDF-Datei in DIN-A4
 - L^AT_EX
 - OpenLibreOffice (Export)
 - PDF-Drucker
 - Scanner im CIP-Pool im zweiten Stock
 - bitte **keine** handgeschriebenen Lösungen **abfotografieren**
 - auf **Dateigröße** achten und große Grafiken verkleinern
 - Abgabe vorher (!) auf **Lesbarkeit** prüfen; unleserlich \leadsto **0 Punkte**
 - auf **Seitenformat** und **Schriftgröße** achten
- die **Korrektur** der Theorie-Aufgaben erfolgt direkt durch den Tutor
 - Rückgabe inkl. Korrekturkommentaren (im PDF) via EST
 - Tutoren schreiben meistens nicht die Lösung ins PDF \leadsto Tafelübung
 - bei Unklarheiten beim Tutor (!) nachfragen

Praxis-Aufgaben

- Bearbeitung der **Praxis-Aufgaben** in der Programmiersprache **Java**
 - Abgabe im Normalfall als **Quelltext-Datei** (*.java)
 - **automatische Korrektur** und Bepunktung durch Tests
 - der **CIP-Pool** (Informatik) ist das **Referenzsystem**
 - alle Abgaben müssen dort **übersetzbar** und **lauffähig** sein!
 - Abgaben müssen mit **javac -encoding ASCII** übersetzbar sein
- ↪ **keine Umlaute** oder **sonstige Sonderzeichen** verwenden!
- auch nicht in Kommentaren!
 - in **Eclipse**: Window >Preferences >General >Workspace >Text file encoding >(other) US-ASCII

Ganz wichtig!

Abgaben, die **nicht übersetzbar** sind, geben generell **0 Punkte**!

Praxis-Aufgaben: Java 7

Wichtig: Java-7-Kompatibilität!

Das EST verlangt **Java-7**-konformen Code!

- entweder JDK 7 installieren...
- ...oder für JDK 8 die Verwendung von Java 7 erzwingen:
 - auf der Konsole: mit Option **javac -source 1.7** übersetzen
 - in Eclipse: Window >Preferences >Java >Compiler >Compiler compliance level >1.7
 - das funktioniert nur in 99,8% der Fälle... mehr dazu, wenn's soweit ist

Praxis-Aufgaben: Spielregeln

- die Schnittstellen der bereitgestellten Klassen dürfen **nicht verändert** werden
 - insbesondere keine Methoden oder Attribute **löschen** oder **umbenennen**!
- eigene Attribute oder Methoden dürfen (falls nicht explizit gefordert) auf **keinen Fall** Teil der **öffentlichen Schnittstelle** (`public`) der Klasse sein
- wird Code aus früheren Abgaben verwendet \leadsto erneut mit abgeben
- Aufgaben immer **komplett bearbeiten**!
 - **Beispiel**: Wenn in d) das Anlegen einer Methode `foo()` gefordert ist, aber die Aufgabe nur bis c) bearbeitet wird, muss trotzdem eine Methode `foo()` angelegt werden (Rumpf evtl. leer lassen, muss aber übersetzbar sein)

Wichtig

Das Einbinden und Verwenden von **fremden Quelltexten oder Bibliotheken** ist (mit Ausnahme der Java-Standardbibliothek) **untersagt**!

Praxis-Aufgaben: JUnit-Tests (I)

JUnit-Tests (*PublicTest.java)

- auf der Konsole:
 - Übersetzen im CIP mit:
`javac -cp ./usr/share/java/junit4.jar *.java`
 - Ausführen im CIP mit:
`java -cp ./usr/share/java/junit4.jar [TestClass]`
- in Eclipse:
 - Project >Properties >Java Build Path >Libraries >Add Library >JUnit 4
 - Übersetzen und Ausführen dann automatisch mit JUnit

Praxis-Aufgaben: JUnit-Tests (II)

JUnit-Testmethoden (@Test)

- werden in zufälliger Reihenfolge ausgeführt
- enthalten Aufrufe in die zu testende Java-Klasse
- Überprüfung der Ergebnisse meist mit `assertEquals(expected, actual)`
- mehr dazu: <http://junit.org/apidocs/org/junit/Assert.html>

Tipp

- die vorgegebenen Testfälle decken meist nicht alle Grenzfälle ab
- ~> schreibt euch eigene zusätzliche Tests
- bestehende Tests nicht verändern
 - sondern: kopieren, die Kopie umbenennen und dann erweitern

Praxis-Aufgaben: Korrektur

- Praxis-Aufgaben werden **automatisiert** bewertet
 - Tests aus vorgegebenen öffentlichen JUnit-Tests + weitere, geheime Tests
- pro Abgabe erscheint im EST eines der folgenden **Symbole**:
 - ✘ Noch keine Abgabe \leadsto **0 Punkte**
 - ⌚ Abgabe wartet auf Testausführung
 - 💀 Übersetzen der Abgabe fehlgeschlagen \leadsto **0 Punkte**
 - ✘ Übersetzen der Abgabe mit Testfall fehlgeschlagen (Signaturen verändert?) \leadsto **0 Punkte**
 - ! JUnit-Tests nicht erfolgreich
 - ✓ JUnit-Tests erfolgreich (garantiert noch **nicht** die volle Punktzahl!)

Gruppen-Abgabe

- Gruppen-Aufgaben werden in Gruppen mit 2 Studenten bearbeitet
 - Gruppen sind nicht fest und können bei jeder Abgabe neu gebildet werden
- jeder Student erhält pro Übungsblatt einen neuen Gruppenabgabecode
 - zu finden im EST
- derjenige, der zuerst eine Version über das EST abgibt, muss den entsprechenden Code des Partners eingeben
- jeder muss überprüfen, ob die Gruppen-Abgabe funktioniert hat!

Hinweis

Zu Beginn wird es etwas mehr Gruppen-Aufgaben geben, später im Semester dafür mehr Einzelaufgaben. Insgesamt sind Einzel- und Gruppen-Aufgaben in etwa ausgeglichen.

Zusatz- und Bonus-Aufgaben

- auf manchen Übungsblättern befinden sich Zusatz- oder Bonus-Aufgaben:
 - **Zusatz-Aufgaben**
 - werden zwar korrigiert, aber es gibt **keine Punkte**
 - im Normalfall etwas schwerer als die “normalen” Aufgaben
 - **Bonus-Aufgaben**
 - bei diesen Aufgaben können **zusätzliche Punkte** erworben werden
 - die Punkte zählen nicht zu der maximal erreichbaren Punktzahl
- vermutlich über Weihnachten: **Bonusblatt** (nur Bonusaufgaben)

Scheinkriterium

Scheinkriterium

Um den Schein zu bestehen, müssen sowohl in den Einzel- als auch in den Gruppen-Aufgaben **jeweils mindestens 60% der Punkte** erreicht werden. In fast allen Prüfungsordnungen ist der Schein **Voraussetzung zum Bestehen des Moduls!**

Tipp 1

Nicht den Anschluss verlieren! Wenn Probleme auftreten, dann **rechtzeitig** Rechner- und Intensivübung besuchen.

Tipp 2

Das Bonusblatt ist eine gute Gelegenheit, um wieder Punkte aufzuholen 😊

Plagiarismus

- **Plagiiere strikt verboten!**
 - ~ Einzel-Aufgaben **alleine** bearbeiten
 - ~ Gruppen-Aufgaben in den **2er-Gruppen** bearbeiten
- wir führen eine manuelle Überprüfung durch und setzen eine Software ein, um Ähnlichkeiten zwischen Abgaben zu entdecken!

Konsequenzen des Plagiiere

0 Punkte für “Spender” und “Empfänger”.

Prävention

„Sperrung“ des Home-Verzeichnisses mittels `chmod 700 ~`.

Weitere Informationen zum Thema

<https://www2.cs.fau.de/teaching/plagiarismus.html>

CipMap


- in den Rechnerübungen wird die CipMap benutzt
 - erreichbar unter <http://www.cipmap.de>
- bei Fragen nicht melden, sondern in die CipMap eintragen
 - Reihenfolge der Betreuung nach FCFS (*first come, first served*)
 - somit gerechtere Aufteilung des Tutors unter den Studenten

CipMap: Erklärung (1)

- CipMap im normalen Modus (zeigt Rechnerbelegung)

CipMap

02.151

02.135 

01.155

01.153

00.153

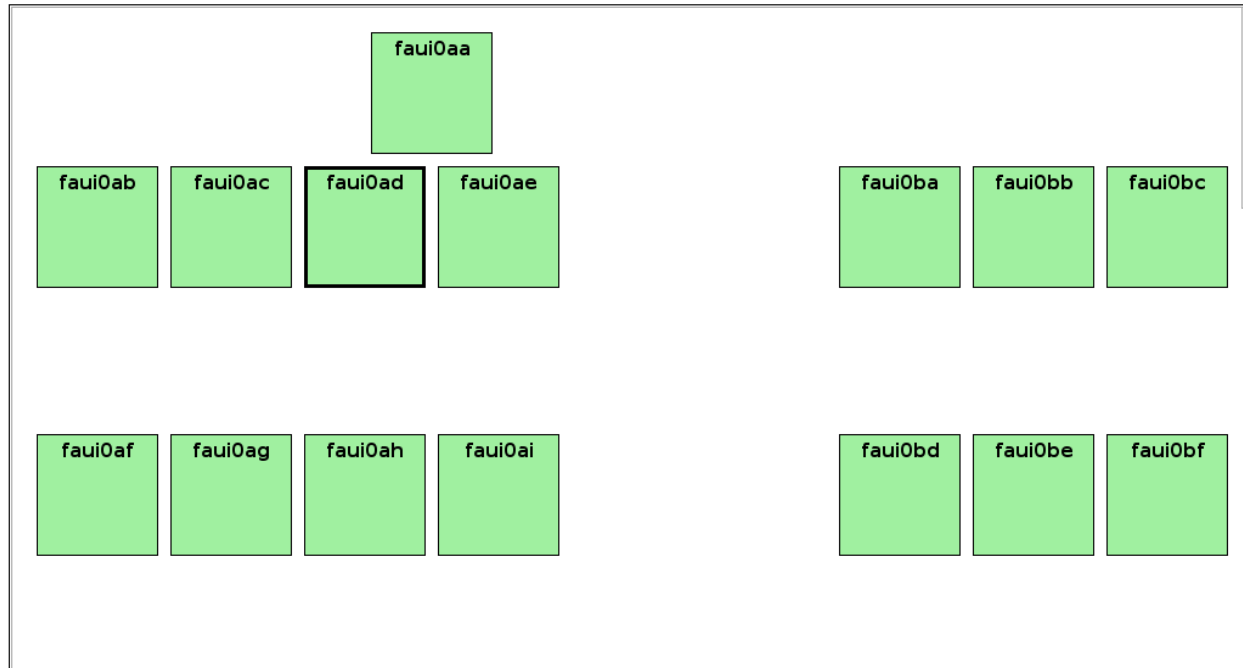
00.156

0.01-142

Faq

Opt-In

Lecture Mode
(BETA)




CipMap: Erklärung (2)

- Raum auswählen (Bibcip \leadsto 02.135) und auf „Lecture Mode“ klicken

CipMap

02.151

02.135 

01.155

01.153

00.153

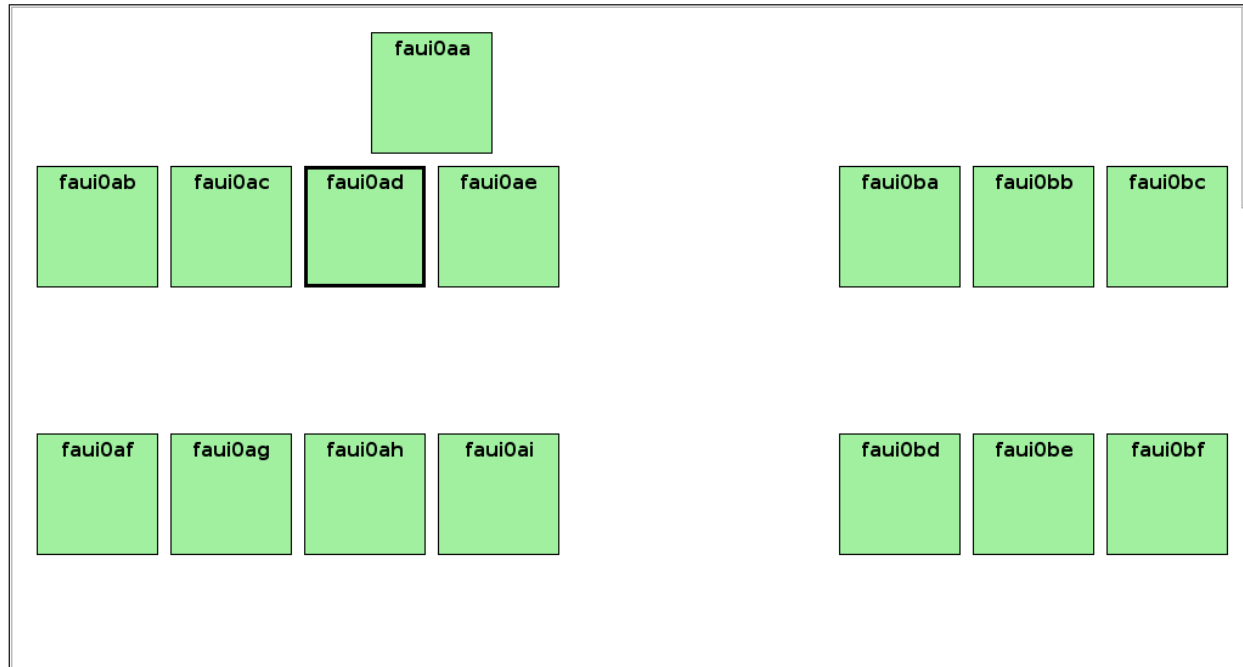
00.156

0.01-142

Faq

Opt-In

Lecture Mode
(BETA)



CipMap: Erklärung (3)

- Ansicht ändert sich!
 - weiß: kein *Request*
 - hellrot: *Request* gestellt
 - dunkelrot: der nächste in der Reihe
- mit „Request tutor“ wird man eingereiht
- bei erneutem Klick wird man wieder ausgereiht

CipMap: Erklärung (4)

- bei Fragen auf „Request tutor“ klicken

CipMap

02.135

RUEB-AuD

Request tutor

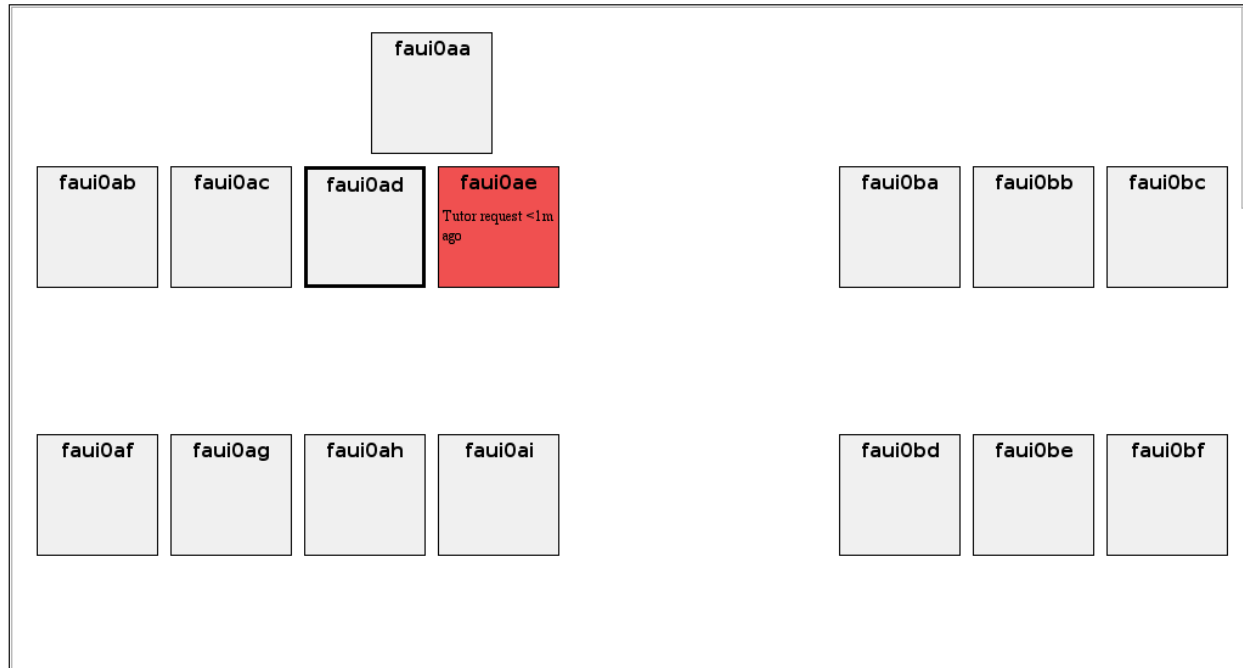
- faui0ae <1m ago

Faq

Opt-In

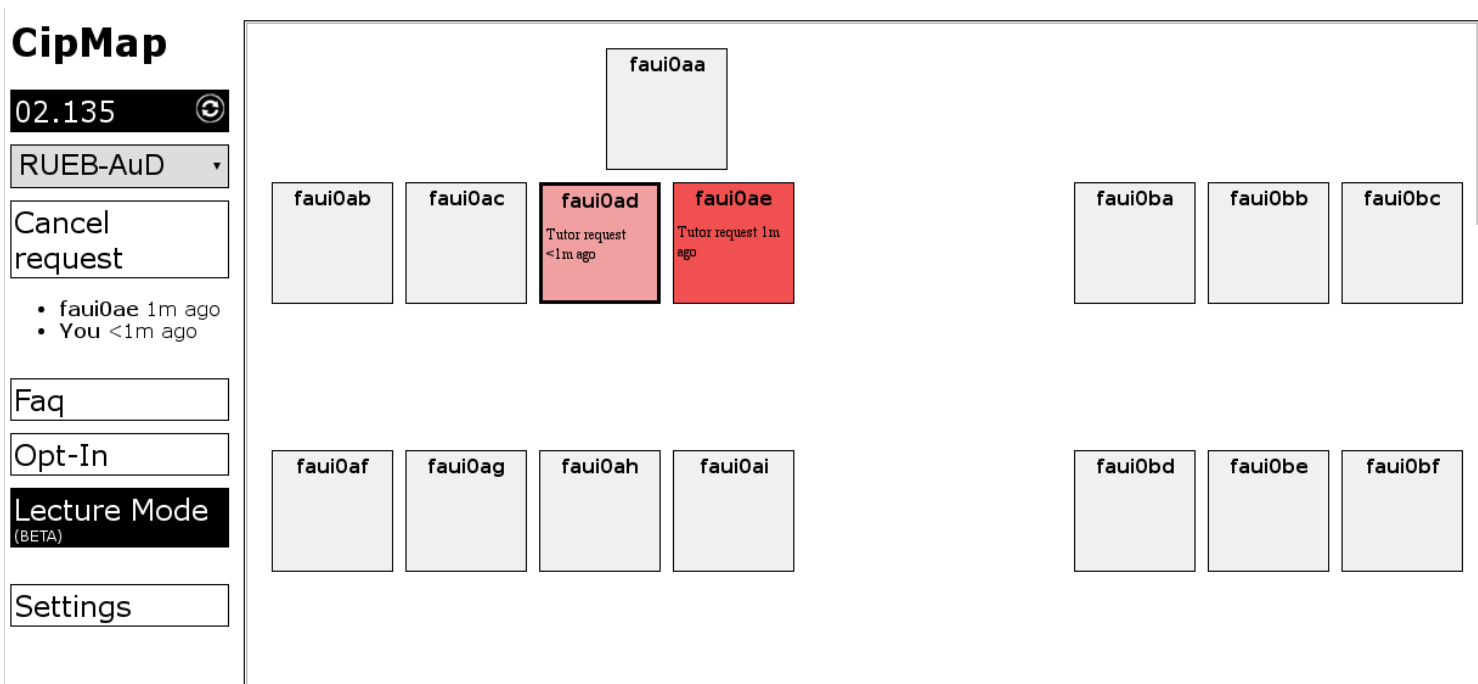
Lecture Mode
(BETA)

Settings



CipMap: Erklärung (5)

- *Request* ist nun gestellt \rightsquigarrow auf Tutor **warten** und **weiterarbeiten** 😊



Linux-Install-Party

Zweck

- Installation einer Linux-Distribution auf **eurem** Rechner
 - auch im **Dual Boot** mit einem existierenden Betriebssystem
- Unterstützung durch **FSI-Mitglieder** und **CIP-Admins**

Zeit & Ort

Mittwoch, 9. November 2016 ab 13:00 Uhr in 02.152-113

Mehr Infos

<http://fsv.tf/lip16>

Wichtig!

Bitte vorher ein **Backup** der Daten auf eurem Rechner machen!

Zahlensysteme

Lehrstuhl Informatik 2
Programmiersysteme



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Motivation

- für einen Menschen ist (meist) klar:
 - 100 € meint einen Betrag von einhundert Euro
 - 42 ist die Dezimalzahl zweiundvierzig
- ein Computer „rechnet mit Nullen und Einsen“
 - 00110001 00110000 00110000 00100000 11100010 10000010 10101100
kann einen Betrag von einhundert Euro meinen
 - 0010 1010 *kann* die Dezimalzahl zweiundvierzig sein

Daten im Computer

- in einem Computer werden sämtliche Daten **binär** gespeichert
- je nach „Kontext“ werden die Daten verschieden **interpretiert**

Zahlen in verschiedenen Basen

Darstellung von Zahlen zu beliebigen Basen

Eine Sequenz

$$(r_{l-1}, r_{l-2}, \dots, r_1, r_0)_{(b)}$$

von Ziffern r_i mit der Länge l und der Basis b , $r_i \in \{0, 1, \dots, b-1\}$, repräsentiert die vorzeichenlose, ganze Zahl

$$s = \sum_{i=0}^{l-1} (r_i \cdot b^i).$$

Dabei ist b^i die Wertigkeit der Ziffer r_i . Weiter vorne stehende Ziffern haben eine höhere Wertigkeit als weiter hinten stehende (*Big Endian*).

Angabe der Basis

Eine Zahlendarstellung muss eindeutig sein!

Wenn aus dem Kontext nicht **eindeutig** ersichtlich ist, um welche Basis es sich handelt, muss diese **explizit angegeben** werden!

Beispiele

- $1303_{(10)}$, $1303_{(8)}$
- später lernen wir noch andere Möglichkeiten für die Notation kennen

Dezimalsystem

- Dezimalsystem \equiv Zahlensystem zur Basis 10
 - $r_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- unser „alltägliches“ Zahlensystem
 - \leadsto häufig Umrechnung in dieses Zahlensystem notwendig

Beispiel \leadsto Formel scheint zu stimmen 😊

$$1303_{(10)} \stackrel{(10)}{=} (1 \cdot 10^3) + (3 \cdot 10^2) + (0 \cdot 10^1) + (3 \cdot 10^0) = 1303$$

Binärsystem

- Binärsystem \equiv Zahlensystem zur Basis 2
 - $r_i \in \{0, 1\}$
 - eine Ziffer nennen wir auch Bit (*binary digit*)
- in der Digitaltechnik verwendetes Zahlensystem
 - Form, in der Daten in einem Rechner gespeichert werden
- Möglichkeiten der Kennzeichnung:
 - $1011_{(2)}$
 - 0b1011

Beispiel

$$1011_{(2)} \stackrel{(10)}{=} (1 \cdot 2^3) + (0 \cdot 2^2) + (1 \cdot 2^1) + (1 \cdot 2^0) = 11$$

Sedezimalsystem („Hexadezimalsystem“)

- Sedezimalsystem \equiv Zahlensystem zur Basis 16
 - $r_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$
- wir werden später sehen:
 - erlaubt kompakte Darstellung von Binärzahlen
- Möglichkeiten der Kennzeichnung:
 - $BABE_{(16)}$
 - $0xBABE$

Beispiel

$$BABE_{(16)} \stackrel{(10)}{=} (11 \cdot 16^3) + (10 \cdot 16^2) + (11 \cdot 16^1) + (14 \cdot 16^0) = 47806$$

Oktalsystem

- Oktalsystem \equiv Zahlensystem zur Basis 8
 - $r_i \in \{0, 1, 2, 3, 4, 5, 6, 7\}$
- wie bei Zahlen im Dezimalsystem:
 - erlaubt kompakte Darstellung von Binärzahlen
- Möglichkeiten der Kennzeichnung:
 - $1303_{(8)}$
 - 01303

Beispiel

$$1303_{(8)} \stackrel{(10)}{=} (1 \cdot 8^3) + (3 \cdot 8^2) + (0 \cdot 8^1) + (3 \cdot 8^0) = 707$$

Umrechnung: beliebiges System \mapsto Dezimalsystem

Die Umrechnung von einer Zahl in einem beliebigen System in das Dezimalsystem können wir bereits 😊

Umrechnung: Dezimalsystem \mapsto beliebiges System

Divisionsmethode/Modulo-Methode

Zahl N aus dem Dezimalsystem in System zur Basis b umwandeln:

- N ganzzahlig durch b teilen \leadsto ganzzahliger Quotient N' , Rest R
- R als Ziffer „vorne an das Ergebnis anhängen“
- falls $N' \neq 0$: mit N' nach gleichem Schema verfahren

Beispiel

N	b	Quotient N'	Rest R
4867	/ 16	304	3
304	/ 16	19	0
19	/ 16	1	3
1	/ 16	0	1

$$\leadsto 4867_{(10)} = 1303_{(16)}$$

Umrechnung: beliebiges System \mapsto beliebiges System

- Umrechnung zwischen **beliebigen Zahlensystemen**:
 - häufig ist der „Umweg“ über Dezimalsystem am einfachsten
 - System zur Basis $a \mapsto$ Dezimalsystem \mapsto System zur Basis b
- **aber**: es gibt einen **Trick** bei „besonderen“ Zahlensystemen

Umrechnung „ohne Umwege“

Wenn zwei Zahlensysteme mit den Basen a und b gegeben sind, und es gilt $a^e = b$ für ein $e \in \mathbb{N}$, dann kann eine Ziffer aus dem Zahlensystem zur Basis b in e Ziffern des Zahlensystems zur Basis a umgewandelt werden.

Beispiel

- Binärsystem: Basis 2, Sedezimalsystem: Basis 16
- $2^4 = 16$

\leadsto vier Binärziffern \leftrightarrow eine Sedezimalziffer

Trick bei besonderen Zahlensystemen

Zur Erinnerung

vier Binärziffern \leftrightarrow eine Sedezimalziffer

Beispiel

Umrechnung von $101010_{(2)}$ in das Sedezimalsystem:

$$\begin{array}{cc|cc} 0010 & & 1010 & \\ \updownarrow & & \updownarrow & \\ 2 & & A & \end{array}$$

$\leadsto 101010_{(2)} = 2A_{(16)}$

Negative Zahlen

Hinweis

Wir gehen im Folgenden von Darstellungen im **Binärsystem** aus. Wir können diese Darstellungen aber auch in das Sedezimalsystem oder andere Zahlensysteme umwandeln.

Wichtig

Spätestens, wenn wir auch negative Zahlen darstellen wollen, müssen wir mit Zahlen **fester Länge** arbeiten, d.h. die Anzahl an Bits vorgeben – insbesondere auch, weil das vorderste Bit das **Vorzeichen** angibt.

Hinweis

Im Folgenden arbeiten wir immer mit **8 Bit** langen Zahlen.

Möglichkeiten zur Darstellung negativer Zahlen

- es gibt im Wesentlichen drei **verschiedene Darstellungen** negativer Zahlen:
 - **Vorzeichen-Betrags-Darstellung**
 - vorderstes Bit gibt Vorzeichen an, restliche Bits den Betrag
 - **Einerkomplement-Darstellung**
 - für negative Zahlen alle Bits des Betrags invertieren
 - **Zweierkomplement-Darstellung**
 - für negative Zahlen alle Bits des Betrags invertieren und 1 addieren
 - diese Darstellung wird tatsächlich verwendet

Vorzeichen-Betrags-Darstellung

Vorzeichen-Betrags-Darstellung

Das vorderste Bit gibt das Vorzeichen an, wobei 0 positive Zahlen und 1 negative Zahlen kennzeichnet. Die restlichen Bits stellen den Betrag der Zahl dar.

Beispiel: ± 13

$+13_{(10)} \mapsto \underline{0}0001101_{(2)} \mapsto 0x0D$
 $-13_{(10)} \mapsto \underline{1}0001101_{(2)} \mapsto 0x8D$

Probleme

Zwei Darstellungen der Null (± 0), positive und negative Zahlen müssen bei Rechnungen getrennt behandelt werden.

Einerkomplement-Darstellung

Einerkomplement-Darstellung

Positive Zahlen werden wie vorhin gezeigt dargestellt (auf führende Null achten!). Für negative Zahlen werden alle Bits der dazugehörigen positiven Zahl invertiert.

Beispiel: ± 13

$+13_{(10)} \mapsto 00001101_{(2)} \mapsto 0x0D$
 $-13_{(10)} \mapsto 11110010_{(2)} \mapsto 0xF2$

Probleme

Zwei Darstellungen der Null (± 0).

Zweierkomplement-Darstellung (I)

Zweierkomplement-Darstellung

Positive Zahlen werden wie vorhin gezeigt dargestellt (auf führende Null achten!). Für negative Zahlen werden alle Bits der dazugehörigen positiven Zahl invertiert, anschließend wird 1 addiert.

Beispiel: ± 13

$$+13_{(10)} \mapsto 00001101_{(2)} \mapsto 0x0D$$

$$-13_{(10)} \mapsto 11110010_{(2)} + 1 = 11110011_{(2)} \mapsto 0xF3$$

Vorteile

Nur noch eine Darstellung der Null, positive und negative Zahlen können bei Rechnungen gleich behandelt werden.

Zweierkomplement-Darstellung (II)

In Java

```
byte foo = (byte)(96 + 82); // 8 Bit, vorzeichenbehaftet
System.out.format("foo = %d\n", foo);
```

Ausgabe

```
foo = -78
```

Grund

$$\begin{array}{rcl}
 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & = & +96_{(10)} \\
 + & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & = & +82_{(10)} \\
 \hline
 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & = & -78_{(10)}
 \end{array}$$

Entscheidungsgehalt

Entscheidungsgehalt

Der Entscheidungsgehalt H einer Menge entspricht der Anzahl an Bit, die man benötigen würde, um jedes Element dieser Menge eindeutig mit einem Bitmuster zu identifizieren.

Beispiel

Vier Bitmuster nötig, um die vier Himmelsrichtungen *Nord*, *Ost*, *Süd*, *West* zu kodieren:

$Nord \mapsto 00$ $Ost \mapsto 01$ $Süd \mapsto 10$ $West \mapsto 11$

Diese vier Bitmuster können mit 2 Bit dargestellt werden, d.h. der Entscheidungsgehalt der Menge der Himmelsrichtungen ist 2 bit.

Berechnung

$$H(M) = \lceil \log_2(|M|) \rceil$$

Eine kurze Einführung in Java

Lehrstuhl Informatik 2
Programmiersysteme



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Entwicklung in der Programmiersprache Java

- in AuD verwendete Programmiersprache: **Java**
 - benötigt wird das **Java Development Kit** (JDK)
 - im CIP vorhanden, aber auch zu Hause kostenlos nutzbar¹
- Quelltext muss vor der Ausführung zunächst **übersetzt** werden
 - das Programm **javac** übersetzt den Quelltext in **Java-Bytecode**
- Java-Bytecode wird **nicht nativ** auf der Hardware ausgeführt
 - sondern: von der **Java Virtual Machine** (JVM) **interpretiert**
 - Start der Laufzeitumgebung zur Ausführung eines Programms mit **java**
 - Vorteil: **Plattformunabhängigkeit**

¹siehe <https://www2.cs.fau.de/aud/organisation/index.html#infrastructure>

Klassischer Workflow

Workflow

```
$> gedit MeinProgramm.java  
  
... Datei bearbeiten ...  
  
$> ls  
MeinProgramm.java  
  
$> javac MeinProgramm.java  
  
$> ls  
MeinProgramm.java  MeinProgramm.class  
  
$> java MeinProgramm
```

Nicht vergessen...

Nach jeder **Quelltext-Änderung** muss das Programm **neu übersetzt** werden.

Einschub: Eclipse

- **Eclipse**: professionelle **Entwicklungsumgebung**
 - sog. IDE: *Integrated Development Environment*
- kann häufige Arbeitsschritte **verkürzen** oder **automatisieren**
 - Klassengerüste erstellen
 - Methodenstümpfe erstellen
 - Syntaxfehler anzeigen und verbessern
 - Methoden- und Attributnamen vorschlagen
 - ...
- integrierter **Debugger** zur Fehlersuche

Anfängertipp

- zunächst ohne IDE programmieren und Fehler selbst suchen
- für spätere, umfangreiche Aufgaben auf Eclipse umsteigen

Aufbau einer Java-Datei

Beispiel für den Aufbau einer Java-Datei

```
public class MeinProgramm {  
  
    public static void main(String[] args) {  
        // hier Code  
    }  
  
}
```

main-Methode

Die Ausführung des Programms beginnt immer in der main-Methode.

Wichtig

Die Datei muss genauso heißen wie die Klasse!

Fehler beim Übersetzen

- ein Programm kann nur dann übersetzt werden, wenn es den **syntaktischen und semantischen Regeln** von Java genügt
- andernfalls meldet der Übersetzer einen **Fehler** und **bricht ab**
 - **Fehlermeldung** gibt Hinweis darauf, was kaputt ist (s.u.)
- solche Fehler gehören dazu und passieren recht leicht \leadsto **keine Sorge** 😊

Was tun bei einem Fehler?

Fehlermeldung lesen und versuchen, den Fehler zu beheben! 😊

Beispiele für Fehlermeldungen (I)

Strichpunkt vergessen

```
Volumen.java:4: ';' expected
      int breite = 7
                  ^
```

Was will uns der Übersetzer sagen?

- Fehler in der Datei Volumen.java ...
- ... in Zeile 4 ...
- ... an der gekennzeichneten Stelle (*vermutlich...*)

Unerwartetes Dateiende (geschweifte Klammer vergessen?)

```
Volumen.java:9: reached end of file while parsing
}
^
```

Beispiele für Fehlermeldungen (II)

Falscher Dateiname

```
Test.java:1: class Volumen is public, should be declared in a file
named Volumen.java
public class Volumen {
      ^
```

Verwendung einer nicht deklarierten Variable

```
Volumen.java:4: cannot find symbol
symbol   : variable laenge
location: class Volumen
    laenge = 5;
      ^
```

... und viele mehr ... ☺

Fragen? Fragen!

(hilft auch den anderen)

Lehrstuhl Informatik 2
Programmiersysteme



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT