# Grundlagen der Informatik III

Wintersemester 2010/2011

Wolfgang Heenes, Patrik Schmittat

TECHNISCHE UNIVERSITÄT DARMSTADT

# 9. Aufgabenblatt

17.01.2011

**Hinweis**: Der Schnelltest und die Aufgaben sollen in den Übungsgruppen bearbeitet werden. Die Hausaufgaben sind in der Kalenderwoche 4 (24.01. bis 28.01.) bei den Tutoren in **physikalischer Form** (handschriftlich oder gedruckt) abzugeben. Bei allen Abgaben ist der Name des Tutors und die Übungsgruppe deutlich anzugeben. Bei Teamabgaben wird nur eine Lösung eingereicht, die alle Namen der Teammitglieder enthält.

Schicken Sie Ihre Lösungen von Programmieraufgaben zusätzlich zur schriftlichen Abgabe per E-Mail an Ihren Tutor. Kommentieren Sie Ihren Quellcode.

## Aufgabe 1: Schnelltest

| Fragen | Antworten |
|---|---|
| **1.** Wie geht ein Compiler mit einer `#include<.>`-Anweisung um? | ☐ Die entsprechende Library zur `.h`-Datei wird zur Laufzeit vom Lader eingebunden.<br>☐ Der Inhalt der `.h`-Datei wird durch den Preprozessor eingefügt.<br>☐ Die Symbole der `.h`-Datei werden zur Compilezeit in die Symboltabelle eingefügt.<br>☐ Die genutzten Funktionen werden automatisch beim Linken eingefügt.<br>☐ Die `include` Anweisung ist lediglich eine Konvention und besitzt keine Funktionalität. |
| **2.** Warum müssen Assembler zwei Durchgänge durchführen? | ☐ Nicht jeder Opcode kann beim ersten Durchlauf einwandfrei ermittelt werden.<br>☐ Variablen müssen vorher entsprechend ersetzt werden, bevor jeder Befehl assembliert werden kann.<br>☐ Adressen von Sprungmarken sind vorher nicht immer bekannt.<br>☐ Der Assembler muss zuerst die Größe der entstehenden Objektdatei ermitteln.<br>☐ Relative Sprünge können erst nach dem zweiten Durchlauf korrekt gesetzt werden. |
| **3.** Welche Aussagen über absolute und relative Adressen stimmen? | ☐ Das Laden eines Programms mit absoluten Adressen ist am schnellsten durchführbar.<br>☐ Relative Adressen müssen beim Laden nicht verändert werden.<br>Fortsetzung nächste Seite... |

| Fragen | Antworten |
|---|---|
| | ☐ Ein Programm mit relativen Adressen ist nach dem Laden nicht mehr verschiebbar.<br>☐ Absolute Adressen erlauben es flexibel das Programm im Speicher zu verschieben.<br>☐ Dynamisches Laden erlaubt es allgemein schnellere Programmausführung zu ermöglichen. |
| **4.** Welche Aussagen über das Format von ELF-Objektdateien sind korrekt? | ☐ Das ELF-Format ist unabhängig von Little- und Big-Endian.<br>☐ Ausführbarer Code und Datenfragmente werden getrennt voneinander gespeichert.<br>☐ Daten die nur gelesen und nicht geschrieben werden, sollten in `.rodata` abgelegt werden.<br>☐ Im ELF-Header ist die Startadresse von `_start` eingetragen.<br>☐ Ein ELF-Header kann erst generiert werden wenn alle externen Symbole aufgelöst wurden. |
| **5.** Welche Aussagen über das Binden sind korrekt? | ☐ Prinzipiell werden Libraries immer während der Laufzeit gebunden / geladen.<br>☐ Beim Binden mehrerer Objektdateien müssen deren Adressen angepasst werden.<br>☐ Ein C Programm muss immer mit mindestens einer Library gebunden werden.<br>☐ Falls während der Laufzeit externe Referenzen vorhanden sind, muss der Lader die entsprechende Library laden.<br>☐ Die Abschnitte `.rel.text` und `.rel.data` sind nach dem Binden nicht mehr vorhanden. |

## Aufgabe 2: Übersetzungsvorgang

Geben Sie die Reihenfolge der verschiedenen Phasen bei der Übersetzung an. Beschreiben Sie in einem bis zwei Sätze die Aufgaben jeder Phase und geben Sie die genutzten Zwischenprodukte an.

__ Assembler:

__ Compiler:

__ Linker:

__ Preprozessor:

## Aufgabe 3: Übersetzung verschiedener Sprachen

Geben Sie zu den folgenden Übersetzungsrichtungen an ob diese problemlos durchführbar sind. Falls dies nicht der Fall sein sollte, geben Sie die auftretenden Probleme an.

a) IA32 Assembler → Maschinencode

b) Maschinencode → IA32 Assembler

c) C → IA32 Assembler

d) IA32 Assembler → C

## Aufgabe 4: Kontextfreie Grammatiken I

Welche der gegebenen Folgen von Terminale können mit der entsprechenden Grammatik hergeleitet werden?

a)

| | |
|---|---|
| Startsymbol: X | 423.55.324 |
| X → YXY | 82299228 |
| X → . | 4521.254 |
| Y → 1\|2\|3\|4\|5\|6\|7\|8\|9\|0 | 5833.1294 |

b)

| | |
|---|---|
| Startsymbol: S | acbbba |
| S → aXY \| Z | caabcba |
| X → aX \| c | acbcba |
| Y → bYb \| X | aaacaaaaac |
| Z → Za \| XY | aaabcb |

## Aufgabe 5: Kontextfreie Grammatiken II

Für die folgenden Aufgaben sei $\Sigma$ die Menge der verfügbaren Terminale, geben Sie für jede Aufgabenstellung eine entsprechende kontextfreie Grammatik an.

a) Sei $\Sigma := \{+, -, /, *, (, ), 0, ..., 9\}$. Geben Sie eine Grammatik für die Menge aller arithmetischen Terme mit Ganzahlen an.

b) Sei $\Sigma := \{+, -, /, *, (, ), 0, ..., 9, \neg\}$. Geben Sie eine Grammatik für die Menge aller arithmetischen Terme in Postfixnotation mit Ganzahlen an.

Hinweis: $\neg$ soll hierbei als Trennzeichen zwischen den Zahlen dienen und wäre mit der Return-Taste auf einem Taschenrechner gleichbedeutend. $1\neg43\neg23+*$

c) Sei $\Sigma := \{a, b, c, \$\}$. Geben Sie eine Grammatik für die Menge aller möglichen Palindrome an, die in der Mitte ein $ enthalten.

d) Sei $\Sigma := \{I, V, X, L, C\}$. Geben Sie eine Grammatik für die Menge aller römischen Zahlen bis 100 'C' an.

Hinweis: Sie können dabei zwischen folgenden Darstellungen frei wählen. 99 → IC oder XCIX

e) Sei $\Sigma := \{0, ..., 9, .\}$. Geben Sie eine Grammatik für die Menge aller gültigen IP4 Adressen an.

## Hausaufgabe 1: Compilerausfall (5 Punkte)

Gegeben ist folgendes ANSI-C Codefragment. Leider ist ihr Compiler nicht in der Lage diese Zeilen zu übersetzen. Bilden Sie die folgenden Schritte eines Compilers nach und ermitteln Sie die Zwischenergebnisse des Übersetzungsvorgangs sowie die Symboltabelle selbstständig.

- Lexikalische Analyse
- Syntaktische Analyse
- Semantische Analyse
- Zwischencodeerzeugung
- Codeoptimierer
- IA32 Codegenerator

Informationen über die Variablen werden in die Symboltabelle übernommen:

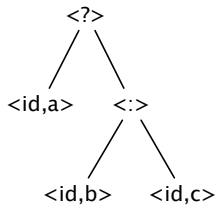
```
char m,n,b;
int i;
float f;
```

Codefragment:

```
f = 2 * f + i / b ? m : n;
```

Hinweis: Nutzen Sie für die Anweisung `a ? b : c` folgenden verschiedenen Darstellungsformen.

Syntaxbaum




Zwischencodeerzeugung

```
  if a goto L1
  t1 = c
  goto L2
L1:
  t1 = b
L2:
```

## Hausaufgabe 2: Analyse von Sprungtabellen (5 Punkte)

Die Implementierung von Sprungtabellen ist auf unterschiedliche Weise möglich. Eine Implementierung ist im Folgenden dargestellt.

```
.data
intout:     .string "Ergebnis␣%d␣\n"
 i:         .long 3 # Auswahl
stab:       .long .L1,.L2,.L3 # Sprungmarken
 c:         .long 0
.text
.globl main
main:
     movl i,%ebx # %ebx mit i laden
     subl $1,%ebx # Anfang ohne Offset
     jmp *stab(,%ebx,4)

.L1: movl $7,c
     jmp .ende
.L2: movl $3,c
     jmp .ende
.L3: movl $8,c
     jmp .ende

.ende:   movl c,%eax
         pushl %eax # Wert im %eax ausgeben
         pushl $intout
         call printf

# Exit
movl $1, %eax
int $0x80
```

a) Welchen Nachteil hat diese Implementierung bzgl. einer korrekten und sicheren Ausführung? Geben Sie an, welche Ausgaben bei welchen Eingaben zu erwarten sind.

b) Die Variable *c* für das Ergebnis ist in dieser Implementierung mit 0 initialisiert. Dies ist allerdings nicht notwendig, da die Variable ja geschrieben wird, bevor sie gelesen wird. Welche Möglichkeiten gibt es uninitialisierte Variablen in Assembler anzugeben? Wird in dem verschiebbaren Objekt-File für eine uninitialisierte Variable Speicher reserviert?

Die folgende Implementierung setzt auch eine Sprungtabelle um.

```
.data
intout:     .string "Ergebnis␣%d␣\n"
 i:         .long 3 # Auswahl
stab:       .long .L1,.L2,.L3 # Sprungmarken
 c:         .long 0
.text
.globl main
main:
     leal i,%edi
     movl (%edi),%ebx
     subl $1,%ebx
     movl %ebx, 8(%edi)
     jmp *stab(,%ebx,4)

.L1: movl $7,c
     jmp .ende
.L2: movl $3,c
     jmp .ende
.L3: movl $8,c
     jmp .ende
```

```
.ende:   movl c,%eax
         pushl %eax # Wert im %eax ausgeben
         pushl $intout
         call printf

# Exit
movl $1, %eax
int $0x80
```

c) Geben Sie an, welche Ausgaben bei welchen Eingaben zu erwarten sind. Woher können ggf. auftretende Probleme kommen?

Betrachtet wird nochmal das erste Programm.

```
.data
intout:     .string "Ergebnis␣%d␣\n"
 i:         .long 3 # Auswahl
stab:       .long .L1,.L2,.L3 # Sprungmarken
 c:         .long 0
.text
.globl main
main:
     movl i,%ebx # %ebx mit i laden
     subl $1,%ebx # Anfang ohne Offset
     jmp *stab(,%ebx,4)

.L1: movl $7,c
     jmp .ende
.L2: movl $3,c
     jmp .ende
.L3: movl $8,c
     jmp .ende

.ende:   movl c,%eax
         pushl %eax # Wert im %eax ausgeben
         pushl $intout
         call printf

# Exit
movl $1, %eax
int $0x80
```

d) Ändern Sie dieses Programm so, dass die Sprungtabelle nicht mehr im **.data** Segement vereinbart ist. Variablen deren Wert zu Übersetzungszeit 0 sind, sollen als uninitialisierte Variablen vereinbart werden.

e) Übersetzen Sie das Programm und analysieren Sie das verschiebbare Objekt-File. Wie sehen die Segmente **.txt**, **.data** und **.rodata** aus. Erläutern Sie die Einträge des **.rodata** Segments mit den Bezügen zum **.txt** Segment.