# Grundlagen Betriebssysteme und Systemsoftware

## Übungsblatt 01

Simon Ellmann
ellmann@in.tum.de

Wintersemester 2018/19
23. Oktober 2018

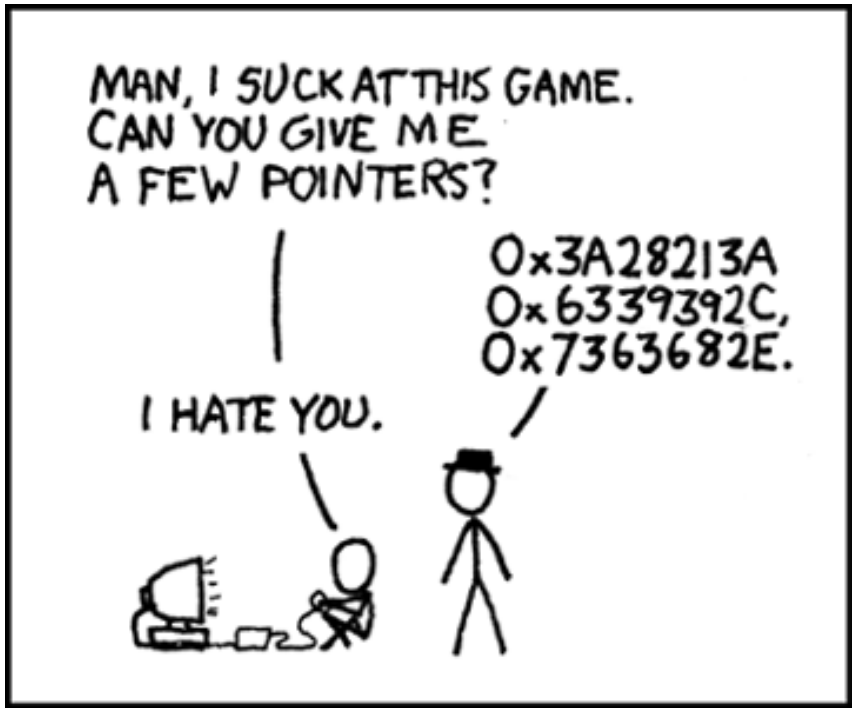
# Organisatorisches



- **Notenbonus** bei 80% der mögl. Hausaufgabenpunkte
- Abgabe in 4er-Gruppen (**in Moodle eintragen**)
- Programme **müssen** kompilieren!

# From Java to C





- Ist C nicht längst obsolet? **Nein**!
- **maschinennahes Programmieren** bei hohem Abstraktionsniveau
  - direkter Speicherzugriff über Pointer
  - einfache Bit-Aritmethik
  - Inline-Assemblercode
  - komplexe Datenstrukturen und Kapselung
- Grundlage für **speicher- und recheneffiziente Programmierung**
  - Betriebssysteme, eingebettete Systeme, Grafikanwendungen, ...
- **Vorbild** für viele moderne Programmiersprachen
  - C++, C#, Java, Perl, PHP, ...

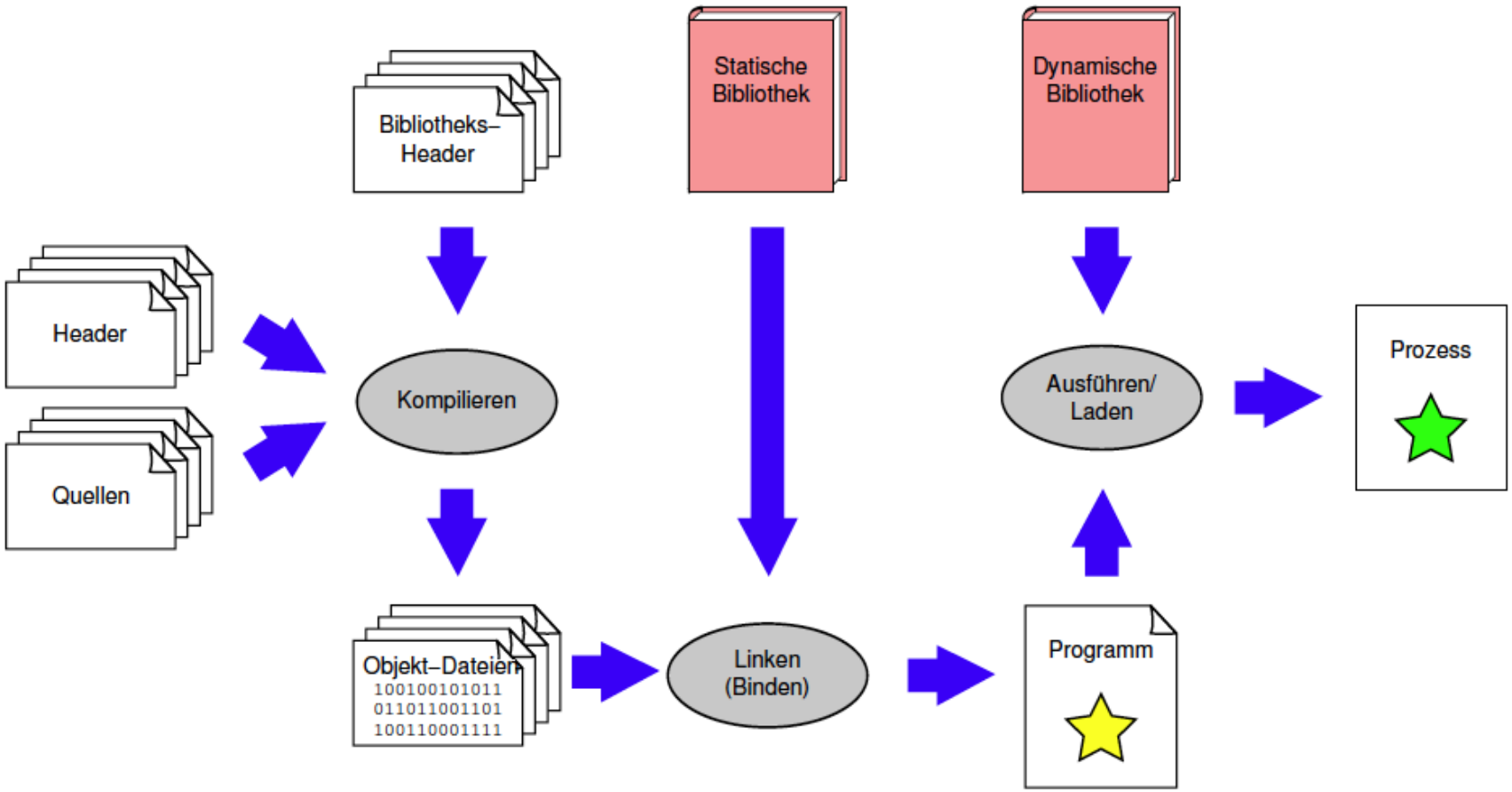
- **Headerdatei**, Endung .h
  - Schnittstelle nach außen, keine Implementierung
  - wird in Quelldatei eingebunden
- **Quelldatei**, Endung .c
  - Implementierung der Funktionalität
  - Nutzung der Schnittstellen anderer Header
- **Objektdatei**, Endung .o
  - kompilierter Maschinencode
  - enthält Referenzen auf Symbole anderer Objekte/Bibliotheken
- **Programmdatei**, Endung .exe bzw. Execute-Bit gesetzt
  - ausführbares Programm
  - enthält evtl. Referenzen zu dynamischen Bibliotheken

## Headerdatei: foo.h

```
1 void foobar(int x);
```

## Quelldatei: foo.c

```
1 #include "foo.h"
2
3 void foobar(int x) {
4   return x*2;
5 }
```


Bibliotheks–
Header
Statische
Bibliothek
Dynamische
Bibliothek
Header
Quellen
Kompilieren
Ausführen/
Laden
Prozess
Objekt–Dateien
100100101011
011011001101
100110001111
Linken
(Binden)
Programm

Syntax der **Kontrollstrukturen** von Java sehr ähnlich zu C:

```
for (;;) { ... }
while () { ... }
do { ... } while ();
if () { ... } else { ... };
switch () { case (): ...; break; default: ...; }
```

- Zuweisung: `a = b;`
- Arithmetik: `+ - * /` und `++ -- += -= *= /=`
- Vergleiche: `&& || !`
- Bitarithmethik: `<< >> ~ & | ^`
- Variablen müssen explizit initialisiert werden und sollten am Anfang von Funktionen deklariert werden (seit C99 nicht mehr zwingend)!

- Definition und Verwendung von Funktionen analog zu Java
- bei Verwendung von Funktionen vor ihrer Definition **müssen** diese zuerst **deklariert** werden

- `main`-Funktion bildet **Einsprungspunkt** beim Start des Programms
- `argc` liefert die Anzahl der Kommandozeilen-Parameter
- `**argv` ist ein Array von Zeigern auf diese Parameter
- **Rückgabewert** von `main` ungleich 0: Fehlercode

```
1 #include <stdio.h>
2
3 int main(int argc, char **argv){
4     printf("Hello, world!\n");
5     return 0;
6 }
```

- Größe und Wertebereiche der Datentypen **variieren** auf unterschiedlichen Architekturen
- C-Standard garantiert nur **Mindestgrößen**
- **Ausnahme**: `char` ist immer genau 1 Byte groß

Typische Werte für x86:

| Typ | Bytes | Wertebereich | | |
|---|---|---|---|---|
| char | 1 | −128 | ... | +127 |
| unsigned char | 1 | 0 | ... | +255 |
| short | 2 | −32768 | ... | +32767 |
| unsigned short | 2 | 0 | ... | +65535 |
| int | 4 | −2.147.483.648 | ... | +2.147.483.647 |
| unsigned int | 4 | 0 | ... | +4.294.967.295 |
| long | 4 | −2.147.483.648 | ... | +2.147.483.647 |
| unsigned long | 4 | 0 | ... | +4.294.967.295 |
| long long | 8 | $-2^{63}$ | ... | $+2^{63}-1$ |
| unsigned long long | 8 | 0 | ... | $+2^{64}-1$ |

- repräsentieren Adressen im Speicher
- & Referenzierung
- * Dereferenzierung
- Nullpointer: zeigt auf kein gültiges Speicherobjekt

```
1  int a = 1;
2
3  int  *b = &a;     // Pointer auf a
4  int   s = *b;     // Wert von b
5  int  *t =  b;     // Adresse von b
6  int **u = &b;     // Pointer auf b
7
8  char text[10] = "test";
9      *(text+1) = 'u';
```

# From Java to C



# POSIX-Funktionen



```c
int printf(const char *format, ...);

ssize_t read(int fildes, void *buf, size_t nbyte);

ssize_t write(int fildes, const void *buf, size_t nbyte);

void *malloc(size_t size);

void free(void *ptr);
```