

# Übungen zu Systemprogrammierung 1

## Ü1 – Speicherverwaltung

---

Wintersemester 2023/24

Luis Gerhorst, Jonas Rabenstein, Eva Dengler, Dustin Nguyen, Christian Eichler,  
Jürgen Kleinöder

Lehrstuhl für Informatik 4  
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Verteilte Systeme  
und Betriebssysteme



FRIEDRICH-ALEXANDER  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT



2.1 Portable Programme

2.2 Anforderungen an Abgaben

2.3 Übersetzen von Programmen

3.1 Verkettete Listen

4.1 Aufgabe 1: lilo

5.1 Gelerntes anwenden



2.1 Portable Programme

2.2 Anforderungen an Abgaben

2.3 Übersetzen von Programmen

3.1 Verkettete Listen

4.1 Aufgabe 1: lilo

5.1 Gelerntes anwenden



- Entwicklung portabler Programme durch Verwendung definierter Schnittstellen

## ANSI C11

- Normierung des Sprachumfangs der Programmiersprache C
- Standard-Bibliotheksfunktionen, z. B. `printf`, `malloc`

## Single UNIX Specification, Version 4 (SUSv4)

- Standardisierung der Betriebssystemschnittstelle
- Wird von verschiedenen Betriebssystemen implementiert:
  - Solaris, HP/UX, AIX (*SUSv3*)
  - Mac OS X (*SUSv3*)
  - ... und natürlich Linux (*SUSv4, aber nicht offiziell zertifiziert*)



## ANSI C11

- Von Microsoft Visual C/C++ nicht unterstützt :-)
- GCC läuft auch unter Windows :-)

## Single UNIX Specification, Version 4 (SUSv4)

- Von Microsoft Windows nicht unterstützt :-)
- UNIX-Kompatibilitätsschicht für Windows: Cygwin (<http://cygwin.com/>)
  - ... ist aber eher frickelig :-|

- Neu in Windows 10 (seit 1607): Windows Subsystem for Linux (WSL)
  - Damit vollwertiges Linux unter Windows verfügbar
  - Aber nicht von uns unterstützt
  - CIP ist weiterhin Referenzsystem!



2.1 Portable Programme

**2.2 Anforderungen an Abgaben**

2.3 Übersetzen von Programmen

3.1 Verkettete Listen

4.1 Aufgabe 1: lilo

5.1 Gelerntes anwenden



- C-Sprachumfang konform zu ANSI C11
- Betriebssystemschnittstelle konform zu SUSv4
- gcc übersetzt Abgabe **warnungs-** und **fehlerfrei** im CIP-Pool mit folgenden Optionen:

```
-std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror
```

- `-std=c11 -pedantic`: nur ANSI-C11-konformer C-Quellcode
  - `-D_XOPEN_SOURCE=700`: nur SUSv4-konforme Betriebssystemaufrufe
- **Korrekte** Fehlerbehandlung (dazu später mehr)
- Keine globalen Variablen, `static` Variablen nur falls nötig
- Korrekturrichtlinien beachten!  
<https://sys.cs.fau.de/lehre/current/sp1/uebung#korrektur>



2.1 Portable Programme

2.2 Anforderungen an Abgaben

**2.3 Übersetzen von Programmen**

3.1 Verkettete Listen

4.1 Aufgabe 1: lilo

5.1 Gelerntes anwenden





- Übersetzen einer Quelldatei mit gcc:

```
gcc -o test test.c
```

- Zur Erinnerung: Starten der ausführbaren Datei test mit ./test
- Verhalten des gcc kann durch Optionen beeinflusst werden
  - g Erzeugt Debug-Symbole in der ausführbaren Datei
  - c Übersetzt Quellcode in Maschinencode, erzeugt aber kein ausführbares Programm
  - Wall aktiviert weitere Warnungen, die auf mögliche Programmierfehler hinweisen
  - Werror gcc behandelt Warnungen wie Fehler



```
student@cip:~ $ gcc -o test test.c
test.c: In function 'main':
test.c:3:2: warning: implicit declaration of function 'printf'
```

- Bibliotheksfunktion: Entsprechendes `#include` fehlt.  
Manual-Page gibt Auskunft über Namen der nötigen Headerdateien:

```
$ man 3 printf
printf(3)                                Library Functions Manual
NAME
    printf, (...)
SYNOPSIS
    #include <stdio.h>
    int printf(const char *restrict format, ...);
```

- eigene Funktion: Forward-Deklaration fehlt

```
test.c: In function 'foo':
test.c:3:1: warning: control reaches end of non-void function
```

- in der Funktion, die einen Wert zurückliefern soll, fehlt an einem Austrittspfad eine passende `return`-Anweisung



2.1 Portable Programme

2.2 Anforderungen an Abgaben

2.3 Übersetzen von Programmen

**3.1 Verkettete Listen**

4.1 Aufgabe 1: lilo

5.1 Gelerntes anwenden



- Wann setzt man eine verkettete Liste ein?
- Anforderungsanalyse für verkettete Liste
  - Wieviele Listenelemente gibt es maximal?
  - Welche Lebensdauer muss ein Listenelement besitzen?
  - In welchem Kontext muss ein Listenelement sichtbar sein?
- Wir brauchen einen Mechanismus, mit dem Listenelemente...
  - ... in a-priori nicht bekannter Anzahl
  - ... zur Laufzeit des Programmes erzeugt und zerstört werden können

## Warum geht das nicht auf dem Stack?

→ vgl. Vorlesung A-II-50, A-II-85



## ■ Beispiel in Java: Aufgabe 10.1 (Menschenkette) in AuD SS 2011

```
WaitingHuman somebody = new WaitingHuman("Mrs. Somebody");
WaitingHuman nobody = new WaitingHuman("Mr. Nobody");
somebody.add(nobody);
```

- In Java: Neues Listenelement wird mit Hilfe von `new` instanziiert
  - ⇒ Reservieren eines Speicherbereiches für das Objekt
  - ⇒ Initialisieren des Objektes durch Ausführen des Konstruktors

## ■ In C: Anlegen eines Listenelementes mittels `malloc(3)`

```
struct listelement *newElement;
newElement = malloc(sizeof(*newElement));
if (newElement == NULL) {
    // Fehlerbehandlung
}
```

- Zurückgegebener Speicher hat undefinierten/zufälligen Wert
  - ⇒ Initialisierung muss per Hand erfolgen



- Explizite Initialisierung mit definiertem Wert: `memset(3)`

```
memset(newElement, 0, sizeof(struct listelement));
```

- Mit `0` vorinitialisierter Speicher: `calloc(3)`

```
struct listelement *newElement;  
newElement = calloc(1, sizeof(*newElement));  
if (newElement == NULL) { /* Fehler */ }
```

- Im Gegensatz zu Java kein Garbage-Collection-Mechanismus in C
  - Speicherbereich muss von Hand mittels `free(3)` freigegeben werden
  - Nur Speicher, der mit `malloc(3)`, `calloc(3)` oder `realloc(3)` angefordert wurde, darf mit `free(3)` freigegeben werden!
  - Zugriff auf freigegebenen Speicherbereich ist undefiniert



2.1 Portable Programme

2.2 Anforderungen an Abgaben

2.3 Übersetzen von Programmen

3.1 Verkettete Listen

**4.1 Aufgabe 1: lilo**

5.1 Gelerntes anwenden



## ■ Zielsetzungen

- Kennenlernen der Umgebung und Entwicklungswerkzeuge
- Dynamische Speicherverwaltung und Umgang mit Zeigern
- Verwendung des Abgabesystems

## ■ Strukturdefinition

```
struct listelement {  
    int value;  
    struct listelement *next;  
};  
typedef struct listelement listelement; // optional
```





- Nur folgende Funktionen zu implementieren
  - `insertElement()`: Fügt einen neuen, nicht-negativen Wert in die Liste ein, wenn dieser noch nicht vorhanden ist. Tritt ein Fehler auf, wird -1 zurückgegeben. Ansonsten wird der eingefügte Wert zurückgegeben.
  - `removeElement()`: Entfernt den ältesten Wert in der Liste und gibt diesen zurück. Ist die Liste leer, wird -1 zurückgeliefert.
- Keine Listen-Funktionalität in der `main()`-Funktion
  - Allerdings: Erweitern der `main()` zum Testen erlaubt und **erwünscht**
- Sollte bei der Ausführung einer verwendeten Funktion (z. B. `malloc(3)`) ein Fehler auftreten, sind keine Fehlermeldungen auszugeben. Im Fehlerfall wird -1 zurück gegeben.



2.1 Portable Programme

2.2 Anforderungen an Abgaben

2.3 Übersetzen von Programmen

3.1 Verkettete Listen

4.1 Aufgabe 1: lilo

5.1 Gelerntes anwenden



## „Aufgabenstellung“

- Programm schreiben, welches ähnlich wie echo funktioniert
  - Alle Argumente werden ausgegeben
  - Jedes Zeichen in einem Argument wird auf eigener Zeile ausgegeben
- Programme übersetzen