Technische Universität Braunschweig

Institute of Operating Systems and Computer Networks
Reliable System Software

Übung Betriebssysteme (BS)

Tafelübung 1: Warmup

Sören Tempel

Wintersemester 2024

Was machen wir in der Tafelübung?

Technische Universität Braunschweig

- Inhalte und Ziele der Tafelübung:
  1. Konzepte aus der Vorlesung praktisch anwenden
  2. Vermittlung von Inhalten zur Bearbeitung der Übungszettel
  3. Besprechung der Aufgabenstellung (alle zwei Wochen)
- Termin: Freitags 9:45 - 11:15 (jede Woche)
- Ort: PK 11.1

Agenda

Technische Universität Braunschweig

1. Vorstellung des Abgabesystems
2. Crashkurs zur C Programmiersprache
3. Vorstellung der ersten Aufgabe
4. Kurzer Crashkurs zu Linux

01t-warmup 2025-10-28

Welche Arten von Aufgaben erwarten mich?

- Fokus: Praktische Anwendung der Vorlesungsinhalte
- Daher: Übungsbetrieb mit Programmieraufgaben
  - Abgabe als Quellcode
  - Interaktion mit den Betriebssystem Schnitttstellen
  - Programmieren in C
  - Programmieren gegen POSIX/Linux

Woher bekomme ich einen Linux Computer?

- Verfügbare Linux Computer:
  1. Dein eigener Laptop
  2. Der IBR-Server: `x1.ibr.cs.tu-bs.de`
  3. Ggf. Windows Subsystem for Linux (WSL)
- Abgabekriterium für die Übung:
  - Es muss auf dem GitLab CI/CD funktionieren

Technische Universität Braunschweig

Allgemeine Anforderungen an Abgaben

Was muss mein Programm können?

Technische Universität Braunschweig

- Welche Funktionen muss ich implementieren?
  - Siehe Aufgabenbeschreibung
- Wie kann ich die Funktionalität prüfen?
- In den Vorgaben sind Tests mit enthalten
  - Werden automatisch vom GitLab CI/CD ausgeführt
  - Aus den Tests ergeben sich „Testcase-Punkte“

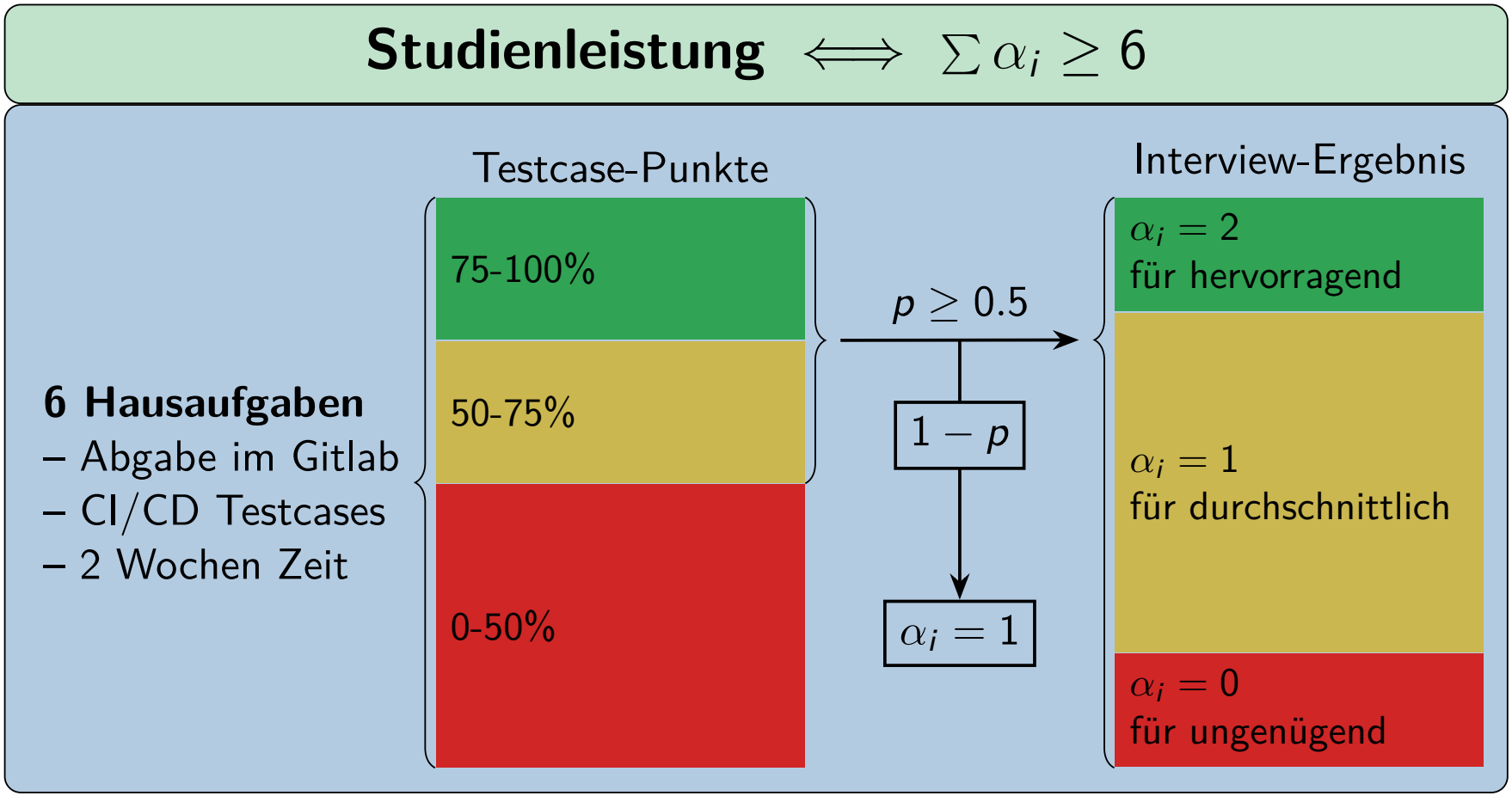
Welche Anforderungen muss der Code erfüllen?

- C-Sprachumfang konform zu C11
- Betriebssystemschnittstelle konform zu SUSv4

Welche Anforderungen muss der Code erfüllen?

- C-Sprachumfang konform zu C11
- Betriebssystemschnittstelle konform zu SUSv4
- **warnungs-** und **fehlerfrei** mit folgenden `gcc`-Optionen übersetzen:
  `-std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror`
  - `-std=c11 -pedantic` erlauben nur C11-konformen Quellcode
  - `-D_XOPEN_SOURCE=700` erlaubt nur SUSv4-konforme Betriebssystemaufrufe

Wdh. Studienleistung: Abgaben und Interviews



Abgabesystem

Stud.IP, GitLab und SSH

Technische Universität Braunschweig

- Stud.IP
  - Forum für inhaltliche und organisatorische Fragen
  - Veröffentlichung der Aufgabenstellung/Zettel als PDF
  - Anmeldung für Code-Interviews
- GitLab
  - Login mit IBR Account
  - Web: `https://gitlab.ibr.cs.tu-bs.de`
  - Koordination mit der PartnerIn
  - Status der Abgaben
- SSH auf Rechner `x1.ibr.cs.tu-bs.de`
  - Shell: `ssh <username>@x1.ibr.cs.tu-bs.de`
  - Voraussetzung: Funktionierende SSH Umgebung
  - Visual Studio Code erlaubt remote Entwicklung

IBR Account erstellen

Falls noch nicht vorhanden:

1. Webseite aufrufen: https://www.ibr.cs.tu-bs.de/passwd/
2. y-Account aktivieren
3. Mit dem GITZ-Account anmelden
4. Erhalt eines gleichnamigen Account im IBR
5. Ermöglicht Anmeldung im GitLab, Nutzung des IBR-Servers, …

Wdh. Zeitlicher Ablauf

Bearbeitungsphase

- ~2 Wochen
- Programmieraufgaben

Anmeldung

1. Aufforderung (per Mail)
2. Termin wählen (StudIP)

Interview-Phase

- ~2 Wochen
- Einzel-Interviews

Bearbeitungsphase (1/2)

Technische Universität Braunschweig

Läuft über das IBR GitLab:

- Pro angemeldetem Studierenden wird von uns ein Git Repository erstellt
- Studierender wird automatisch in dieses Repository eingeladen
- Pfad zum Repository: `vss/teaching/ws25/v_bs/<Team>`

Bearbeitungsphase (1/2)

Läuft über das IBR GitLab:

- Pro angemeldetem Studierenden wird von uns ein Git Repository erstellt
- Studierender wird automatisch in dieses Repository eingeladen
- Pfad zum Repository: `vss/teaching/ws25/v_bs/<Team>`

Zuordnung zu „Team“-Repositories:

- Erfolgt bei der Anmeldung zur Studienleistung via Stud.IP
- Im Stud.IP finden sich auch die Übungszettel

Bearbeitungsphase (2/2)

Technische Universität Braunschweig

Ablauf während des Semesters:

1. Vorgaben für neue Zettel werden uns ins das Repository gepusht

Bearbeitungsphase (2/2)

Ablauf während des Semesters:

1. Vorgaben für neue Zettel werden uns ins das Repository gepusht
2. Bearbeitung der Zettel gemäß der Vorgaben und Aufgabestellung

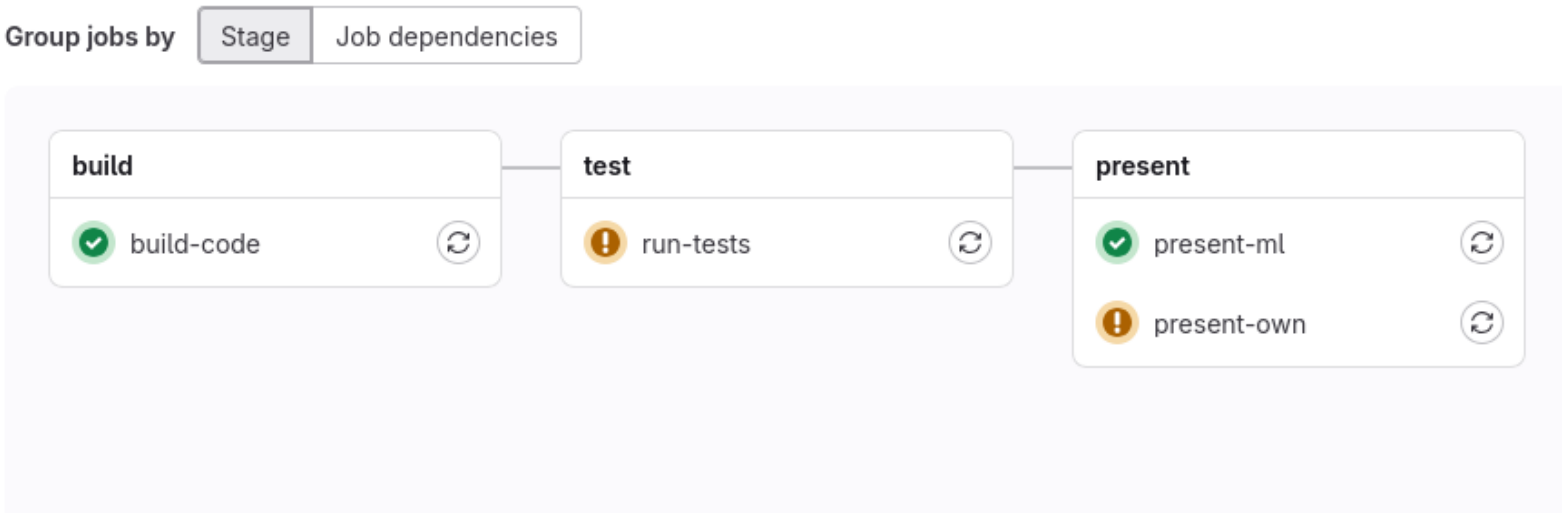
Bearbeitungsphase (2/2)

Ablauf während des Semesters:

1. Vorgaben für neue Zettel werden uns ins das Repository gepusht
2. Bearbeitung der Zettel gemäß der Vorgaben und Aufgabestellung
3. Automatische Überprüfung der Abgabe durch das GitLab CI

Ergebnis der Bearbeitungsphase

Für jeden Commit kann man einen Pipeline-Status abfragen:



- Wir werten den letzten fristgerechten Commit und seine zugehörige Pipeline
- Für den `run-test` Job kann man sich auch die Ausgabe angucken
- Die Ausgabe enthält mehr Informationen zu fehlgeschlagenen Tests
- Tests können auch lokal ausgeführt werden mit:

```
python3 tests/unittest.py
```

Anmeldungsphase

Technische Universität Braunschweig

- Anmeldung zum Interview erfolgt über Stud.IP
- Es gibt eine **separate Stud.IP Veranstaltung** für die Interviews
  - Zu dieser werden Sie hinzugefügt **sofern Sie ausgewählt wurden**
  - Sie werden darüber auch per E-Mail informiert
- Wahl eines Interviewtermins erfolgt über diese Veranstaltung:
  - Reiter: Teilnehmende → Gruppen
  - **Anmeldungszeitraum:** Freitag 18:00 - Sonntag 23:59 Uhr
  - Wählen Sie kein Termin zählt der Zettel als **ungenügend**
- Musterlösung wird ebenfalls Freitags veröffentlicht

Interviewphase

- Sie müssen zu dem von ihnen gewählten Termin **pünktlich** erscheinen
- Bringen Sie zum Termin ihren Studierdenausweis mit
- Sie müssen eine Lösung des Zettels vorstellen:
  - $\geq$ 75%: Ihre **eigene Lösung**
  - $\geq$ 50%: **Unsere Musterlösung**
- Darüberhinaus wird es Nachfragen zum Zettel geben
- Ziel: Festellen ob Sie die Lösung selbst geschrieben und verstanden haben

Ergebnis: Hervorragend, durchschnittlich, oder ungenügend

$\Rightarrow$ Tutor:in teilt ihnen das Ergebnis mit und trägt es ein

C Crashkurs

Programmieren in C

Technische Universität Braunschweig

- Grundlegende Konzepte sollten aus Prog. I und Prog. II bekannt sein
- C++ (Prog. II) wurde ursprünglich als Erweiterung für C entwickelt
- Daher werden hier einige Grundlagen vorausgesetzt:
  - Skalare Datentypen
  - Kontrollstrukturen
  - Operatoren
  - …

Programmieren in C

Technische Universität Braunschweig

- Grundlegende Konzepte sollten aus Prog. I und Prog. II bekannt sein
- C++ (Prog. II) wurde ursprünglich als Erweiterung für C entwickelt
- Daher werden hier einige Grundlagen vorausgesetzt:
  - Skalare Datentypen
  - Kontrollstrukturen
  - Operatoren
  - …

Ziel dieser Veranstaltung ist die Auseinandersetzung mit Betriebssystemen

$\Rightarrow$ Die folgende C Einführung ist als kurzer Crashkurs angedacht

Struktur eines C-Programms

```
[<includes>]opt
[<Globale Variablendefinition>]opt
[<Funktionen>]opt
int main([ int argc, char *argv[ ] ]opt)
{
    [<Variablendefinitionen>]opt
    [<Anweisungen>]opt
}
```

- C-Header: `<dateiname>.h`
  - Enthält Funktions- und Variablen**deklarationen**
  - Beschreibt die Funktion und macht sie dem Compiler bekannt
- C-Quellcode: `<dateiname>.c`
  - Enthält Funktions- und Variablen**definitionen**

Vergleich mit Java

hallo.c

```c
#include <stdio.h>

/*Hallo Welt! in C*/
int main()
{
  printf("Hallo Welt!");
}
```

Hallo.java

```java
/*Hallo Welt! in Java*/
public class Hallo
{
  public static void main(String[] args)
  {
    System.out.print("Hallo Welt!");
  }
}
```

Was auffällt:

- Programmcode von Java und C sieht ähnlich aus

→ **Java verwendet C-Syntax**

  - Funktionen, Variablen, Kontrollstrukturen und die meisten Operatoren

Anweisungen: Kontrollstrukturen

Kontrollstrukturen in C

- Bedingte Anweisung (`if, else`)
- Fallunterscheidung (`switch, case`)
- `for`-Schleife
- `while`-Schleife
- `do-while`-Schleife
- Einfacher Sprung (`goto`-Anweisung)

→ **Sehr ähnliche Syntax zu Java**

Funktionen: Definition und Deklaration

Deklaration/Prototyp

- Syntax: `[Typ]opt name( [Typ Param1]opt [, Typ Param2]opt , ...);`
- Beschreibt die Funktion und macht sie dem Compiler bekannt
- Üblich in Header-Datei <dateiname>**.h**

Definition/Implementierung

- Syntax:

```
[Typ]opt name( [Typ Param1]opt [, Typ Param2]opt ...)
{
   [return <Ausdruck>]opt;
}
```

- Auszuführender Code in der Funktion – vgl. Java-Methoden
- In Datei <dateiname>**.c**

Funktionen: Aufruf

- Syntax: `name( [Argument1]opt , [Argument2]opt , ...);`
- Beispiel:

```c
int add(int parA, int parB);  //Deklaration in myAdd.h
```

```c
int add(int parA, int parB)  //Defintion in myAdd.c
{
  int erg = parA + parB;
  return erg;
}
```

```c
#include "myAdd.h"
int main()
{
  int erg = add(1,2); // Aufruf: erg=3
}
```

Strukturen (structs)

```
struct <structureTag>
{
    <Deklaration>;
    ...
};
...
struct <structureTag> <varName_0>;
```

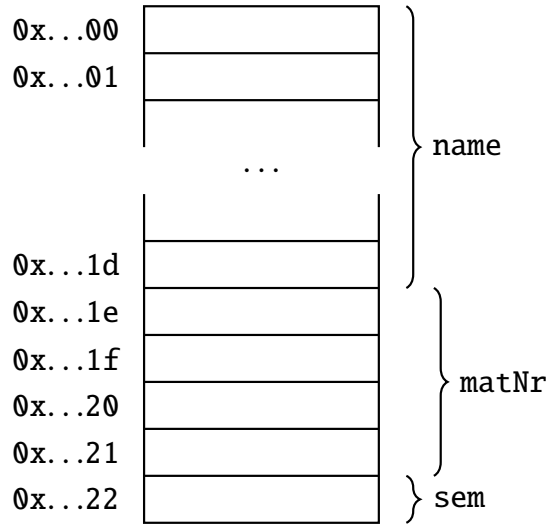
- Zusammenfassung beliebig vieler verschiedener Datentypen zu einem neuen Datentyp
  - Daten in Strukturen: Mitglieder (`Member`) / Komponenten
    - Zugriff auf Member über den Punkt-Operator `.`
    - Member-Namen haben eigenen Scope
  - **Nested Structures**: structs können auch structs enthalten

Strukturen: Beispiel

Beispiel

```
struct student
{
  char name[30];        //30x 1Byte
  unsigned int matNr;   //4 Byte
  unsigned char sem;    //1 Byte
};
struct student stud;
```

Anordnung im Speicher



Datenzugriff

```
struct student stud;
stud.matNr  = 2837465;
stud.sem    = 3;
int magicNr = stud.matNr % stud.sem;
```

- Strukturgröße entspricht **mindestens** der addierten Größe aller Member [1]

[1] Der Compiler kann Padding-Bytes hinzufügen, um die Daten an Speicherzugriffsgrenzen auszurichten

malloc

```
void* malloc( size_t size );
```

- Fordert eine zusammenhängende Menge an Speicher von der Größe `size` in Byte vom Betriebssystem an
- Rückgabe: `(void)`-Zeiger auf das erste Speicherelement
  - Kann wie ein Array interpretiert werden
  - Vor Anwendung Zeiger in sinnvollen Datentyp umwandeln (*casten*)
  - Speicher wird **nicht** initialisiert
- **Achtung: Speicher wird nicht automatisch freigegeben!**
- Beispiel:

```c
int* array;

array = (int*) malloc(10 * sizeof(int));
```

free

```
void free( void* p );
```

- Gibt dynamisch angeforderten Speicher wieder frei
- Erwartet einen Zeiger auf den freizugebenden Speicher
- Beispiel:

```
int* array;

array = (int*) calloc(10, sizeof(int));
...
free(array);
```

Kompilieren

- GCC - GNU Compiler Collection
  - `gcc -o <executable> <source1.c source2 ...>`
  - Übersetzt die angegebenen Quelldateien
  - Eingabe: source1.c, source2, ...
  - Ausgabe: Maschinenprogramm <executable>

- 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

gcc 2025-10-28

Gängige Compiler-Warnungen

Technische Universität Braunschweig

- Problem: fehlende Deklaration einer Funktion

```
test.c: In function 'main':
test.c:3:2: warning: implicit declaration of function 'printf'
```

gcc 2025-10-28

Gängige Compiler-Warnungen

Technische Universität Braunschweig

- Problem: fehlende Deklaration einer Funktion

```
test.c: In function 'main':
test.c:3:2: warning: implicit declaration of function 'printf'
```

  - Lösung für Bibliotheksfunktionen: Entsprechendes `#include` ergänzen.

gcc 2025-10-28

Gängige Compiler-Warnungen

Technische Universität Braunschweig

- Problem: fehlende Deklaration einer Funktion

```
test.c: In function 'main':
test.c:3:2: warning: implicit declaration of function 'printf'
```

  - Lösung für Bibliotheksfunktionen: Entsprechendes `#include` ergänzen.

man 3 printf

```
PRINTF(3)         Linux Programmer's Manual         PRINTF(3)

NAME
       printf,  fprintf, dprintf, sprintf, snprintf, vprintf, vfprintf,
       vdprintf, vsprintf,  vsnprintf  -  formatted output conversion

SYNOPSIS
       #include <stdio.h>

       int printf(const char *format, ...);
       int fprintf(FILE *stream, const char *format, ...);
       int dprintf(int fd, const char *format, ...);
       int sprintf(char *str, const char *format, ...);
       int snprintf(char *str, size_t size, const char *format, ...);
```

gcc 2025-10-28

Gängige Compiler-Warnungen

Technische Universität Braunschweig

- Problem: fehlende Deklaration einer Funktion

```
test.c: In function 'main':
test.c:3:2: warning: implicit declaration of function 'printf'
```

  - Lösung für Bibliotheksfunktionen: Entsprechendes `#include` ergänzen.
    - ↪ Manual-Page gibt Auskunft über den Namen der nötigen Headerdateien
  - Lösung für eigene Funktionen: Forward-Deklaration ergänzen.

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

gcc 2025-10-28

Gängige Compiler-Warnungen

Technische Universität Braunschweig

- Problem: inkonsistente Rückgabewerte

```
fun.c: In function 'foo':
fun.c:3:1: warning: 'return' with no value, in function returning non-void
```

gcc 2025-10-28

Gängige Compiler-Warnungen

Technische Universität Braunschweig

- Problem: inkonsistente Rückgabewerte

```
fun.c: In function 'foo':
fun.c:3:1: warning: 'return' with no value, in function returning non-void
```

```
fun.c: In function 'bar':
fun.c:7:1: warning: 'return' with a value, in function returning void
```

gcc 2025-10-28

Gängige Compiler-Warnungen

Technische Universität Braunschweig

- Problem: inkonsistente Rückgabewerte

```
fun.c: In function 'foo':
fun.c:3:1: warning: 'return' with no value, in function returning non-void
```

```
fun.c: In function 'bar':
fun.c:7:1: warning: 'return' with a value, in function returning void
```

  - in der Funktion, die einen Wert zurückliefern soll, fehlt an einem Austrittspfad eine passende `return`-Anweisung
  - in der Funktion, die keinen Wert zurückliefern soll muss der Parameter vom `return` entfernt werden

- Lösung: richtige Verwendung der `return`-Anweisung.

gcc 2025-10-28

Aufgabe 1 - Die Aufgabenstellung

Verkettete Liste

- Wir wollen eine verkette Liste in C implementieren
- Wann setzt man eine verkettete Liste ein?

Verkettete Liste

Technische Universität Braunschweig

- Wir wollen eine verkette Liste in C implementieren
- Wann setzt man eine verkettete Liste ein?
- Wir brauchen einen Mechanismus, mit dem Listenelemente
  - in a-priori nicht bekannter Anzahl
  - zur Laufzeit des Programmes erzeugt und zerstört werden können

Aufgabenstellung: lilo — Einfach verkettete FIFO-Liste

- Zielsetzung
  - Programmiere eine einfach verkettete Liste
  - Dynamische Speicherverwaltung und Umgang mit Zeigern

Aufgabenstellung: lilo — Einfach verkettete FIFO-Liste

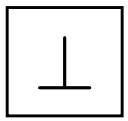
- Zielsetzung
  - Programmiere eine einfach verkettete Liste
  - Dynamische Speicherverwaltung und Umgang mit Zeigern
- Zu implementierende Schnittstelle:
  - `int list_append(list_t *list, int value)`:
    Fügt einen neuen, nicht-negativen Wert in die Liste ein.
    Erfolg: Rückgabe des eingefügten Werts
    Fehler (duplikat, negativ): Rückgabe -1
  - `int list_pop(list_t * list)`:
    Entfernt den ältesten Wert in der Liste und gibt diesen zurück.
    Fehlerfall (leer): -1 wird zurückgeliefert

Aufgabenstellung: lilo — Einfach verkettete FIFO-Liste

- Zielsetzung
  - Programmiere eine einfach verkettete Liste
  - Dynamische Speicherverwaltung und Umgang mit Zeigern
- Zu implementierende Schnittstelle:
  - `int list_append(list_t *list, int value)`:
    Fügt einen neuen, nicht-negativen Wert in die Liste ein.
    Erfolg: Rückgabe des eingefügten Werts
    Fehler (duplikat, negativ): Rückgabe -1
  - `int list_pop(list_t * list)`:
    Entfernt den ältesten Wert in der Liste und gibt diesen zurück.
    Fehlerfall (leer): -1 wird 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.

`lilo` - Funktionsweise

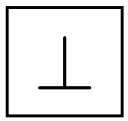
`head`



- Zustand: leer
- Operationen:

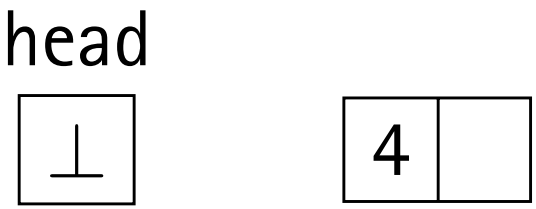
`lilo` - Funktionsweise

`head`



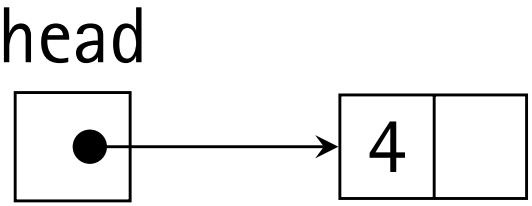
- Zustand: leer
- Operationen:
  1. `list_append(head, 4)`

`lilo` - Funktionsweise



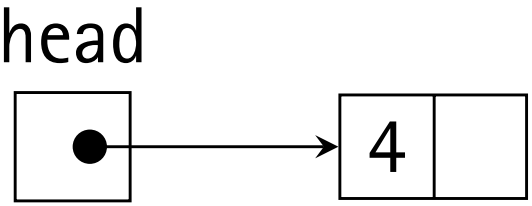
- Zustand: leer
- Operationen:
  1. `list_append(head, 4)`

`lilo` - Funktionsweise



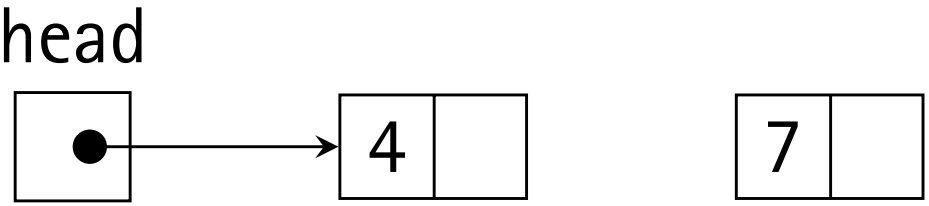
- Zustand: 1 Element
- Operationen:
  1. `list_append(head, 4)`

`lilo` - Funktionsweise



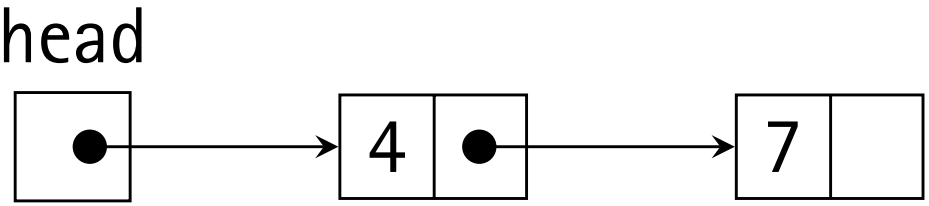
- Zustand: 1 Element
- Operationen:
  1. `list_append(head, 4)` =4

`lilo` - Funktionsweise



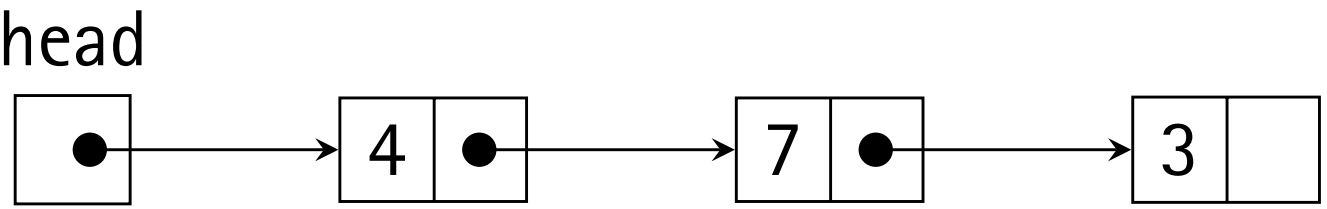
- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)`

`lilo` - Funktionsweise



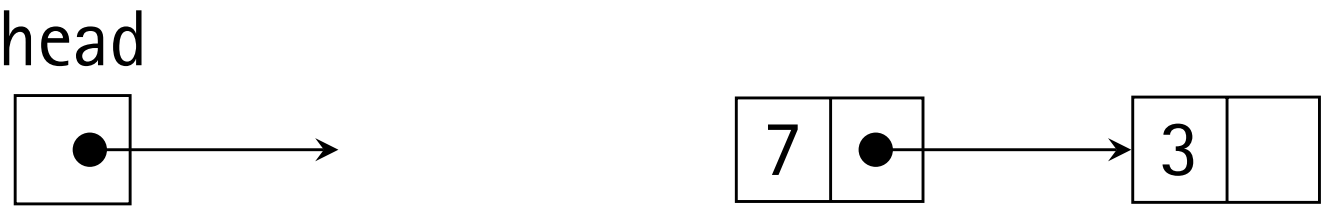
- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7

`lilo` - Funktionsweise



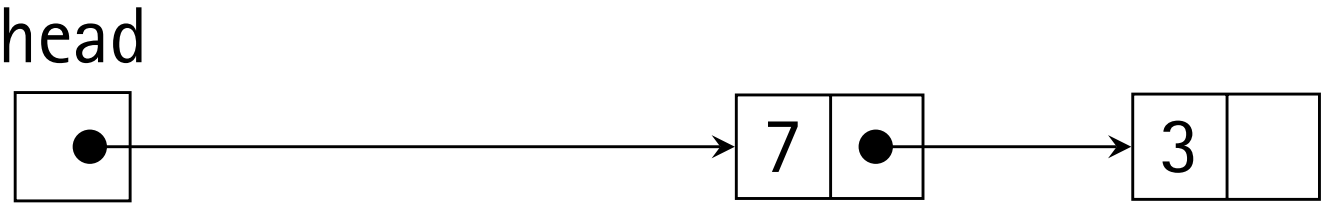
- Zustand: 3 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3

`lilo` - Funktionsweise



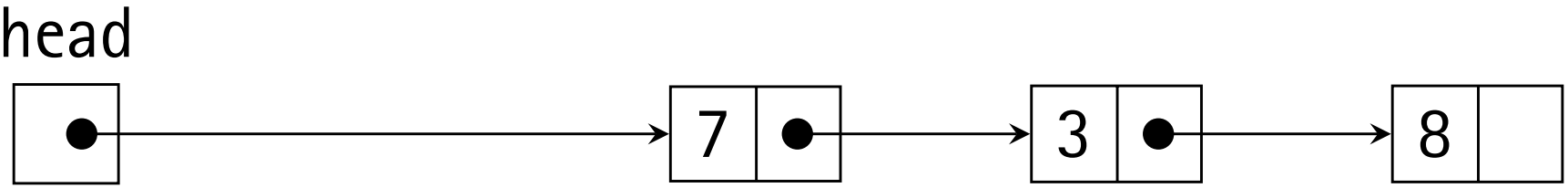
- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4

`lilo` - Funktionsweise



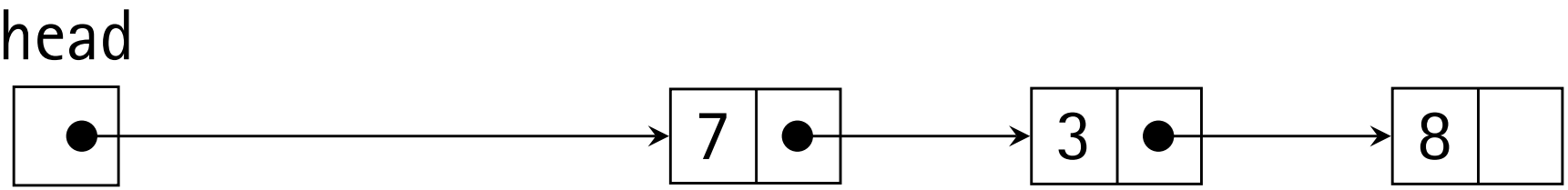
- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4

`lilo` - Funktionsweise



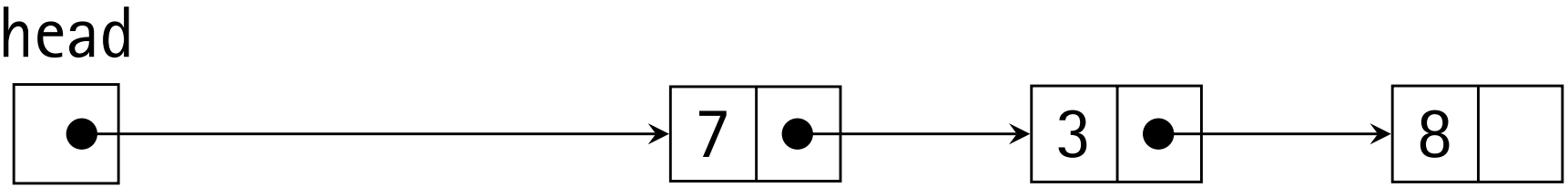
- Zustand: 3 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8

`lilo` - Funktionsweise



- Zustand: 3 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8
  6. `list_append(head, 8)`

`lilo` - Funktionsweise



- Zustand: 3 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8
  6. `list_append(head, 8)` =-1

`lilo` - Funktionsweise

- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8
  6. `list_append(head, 8)` =-1
  7. `list_pop(head)` =7

`lilo` - Funktionsweise

- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8
  6. `list_append(head, 8)` =-1
  7. `list_pop(head)` =7
  8. `list_append(head, -5)`

`lilo` - Funktionsweise

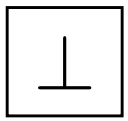
- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8
  6. `list_append(head, 8)` =-1
  7. `list_pop(head)` =7
  8. `list_append(head, -5)` =-1

`lilo` - Funktionsweise

- Zustand: 1 Element
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8
  6. `list_append(head, 8)` =-1
  7. `list_pop(head)` =7
  8. `list_append(head, -5)` =-1
  9. `list_pop(head)` =3

`lilo` - Funktionsweise

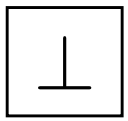
head



- Zustand: leer
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8
  6. `list_append(head, 8)` =-1
  7. `list_pop(head)` =7
  8. `list_append(head, -5)` =-1
  9. `list_pop(head)` =3
  10. `list_pop(head)` =8

`lilo` - Funktionsweise

`head`

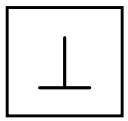


- Zustand: leer
- Operationen:

1. `list_append(head, 4)` =4
2. `list_append(head, 7)` =7
3. `list_append(head, 3)` =3
4. `list_pop(head)` =4
5. `list_append(head, 8)` =8
6. `list_append(head, 8)` =-1
7. `list_pop(head)` =7
8. `list_append(head, -5)` =-1
9. `list_pop(head)` =3
10. `list_pop(head)` =8
11. `list_pop(head)`

`lilo` - Funktionsweise

head



- Zustand: leer
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8
  6. `list_append(head, 8)` =-1
  7. `list_pop(head)` =7
  8. `list_append(head, -5)` =-1
  9. `list_pop(head)` =3
  10. `list_pop(head)` =8
  11. `list_pop(head)` =-1

`lilo` - Datenstruktur

Technische Universität Braunschweig

```
typedef struct list_element {

  ...

} list_element_t;

typedef struct list {

  ...

} list_t;
```

`lilo` - Datenstruktur

Technische Universität Braunschweig

```c
typedef struct list_element {
    Zeiger auf das nächste Listenelement
  ...
    Wert des Elements
} list_element_t;


typedef struct list {
    Zeiger auf das erste Listenelement
  ...
} list_t;
```

Funktionen für die Hausaufgabe

Funktionen für die Hausaufgabe

`int printf(const char *format, ...);`	Formatierte Ausgabe von Zeichenketten Kann auch in Dateien und `char[]` schreiben. printf(3)
`void *malloc(size_t size);`	Allokiert `size` Bytes Speicher auf dem Heap. malloc(3)
`void free(void *ptr)`	Gibt den zuvor allokierten Speicher, auf den `ptr` zeigt, wieder frei. free(3)

- Fehler können wie in den Man-Pages beschrieben, auftreten und sind gemäß der Aufgabenstellung abzufangen
- Es soll keine Ausgabe von Fehlernachrichten geschehen.

Linux Crashkurs

Zugriff per SSH

- SSH ermöglicht das Fernsteuern eines anderen Rechners über ein virtuelles Terminal.
- Alle Befehle, die man in dieses Terminal eintippt, werden auf dem entfernten Rechner ausgeführt.

Zugriff per SSH

- SSH ermöglicht das Fernsteuern eines anderen Rechners über ein virtuelles Terminal.
- Alle Befehle, die man in dieses Terminal eintippt, werden auf dem entfernten Rechner ausgeführt.

Verbinden mit der Referenzplattform

```
ssh <user>@x1.ibr.cs.tu-bs.de
```

- `<user>` ist dein Benutzername.
- `x1.ibr.cs.tu-bs.de` ist der entfernte Rechner.
- Zugangsdaten (Nutzername, Passwort) entsprechen dem IBR Account.

Erster Login

Technische Universität Braunschweig

```
$ ssh <user>@x1.ibr.cs.tu-bs.de
The authenticity of host 'x1.ibr.cs.tu-bs.de' can't be established.
ED25519 key fingerprint is SHA256:M1qnkzcGJmJIV2wAlrosTuPjK0aNfHfvG/tj
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

SSH-Fingerprint

- eindeutiger Fingerabdruck für jeden Rechner
- garantiert, dass man mit dem richtigen Rechner redet
- wird in `~/.ssh/known_hosts` gespeichert
- alle bekannten Fingerprints stehen in `/etc/ssh/ssh_known_hosts`

Die Shell

Technische Universität Braunschweig

- Was ist die Shell?
  - Mediator zwischen Anwender und Computer
  - Interpreter für Befehle
  - Arbeitsumgebung
  - erlaubt Interaktion mit dem Betriebssystem

```
$
```

bash 2025-01-08

Die Shell

Technische Universität Braunschweig

- Was ist die Shell?
  - Mediator zwischen Anwender und Computer
  - Interpreter für Befehle
  - Arbeitsumgebung
  - erlaubt Interaktion mit dem Betriebssystem

```
$ echo foobar
```

bash 2025-01-08

Die Shell

Technische Universität Braunschweig

- Was ist die Shell?
  - Mediator zwischen Anwender und Computer
  - Interpreter für Befehle
  - Arbeitsumgebung
  - erlaubt Interaktion mit dem Betriebssystem

```
$ echo foobar
foobar
$
```

bash 2025-01-08

Die Shell

Technische Universität Braunschweig

- Was ist die Shell?
  - Mediator zwischen Anwender und Computer
  - Interpreter für Befehle
  - Arbeitsumgebung
  - erlaubt Interaktion mit dem Betriebssystem

```
$ echo foobar
foobar
$
```

- Steuerbefehle

CTRL-C	beendet Prozess
CTRL-Z	pausiert Prozess
bg	Prozess im Hintergrund fortsetzen
fg	Prozess im Vordergrund fortsetzen
CTRL-R	In Befehlshistorie suchen
↑↓	Befehlshistorie durchblättern

bash 2025-01-08

bash verwenden

awk	Programmiersprache zum Bearbeiten von Textdaten
cat	Dateien lesen
cd	Verzeichnis wechseln
cp	Datei kopieren
cut	Ausschneiden von Spalten
date	Datum und Zeit anzeigen
echo	Ausgabe erzeugen
find	Dateien auflisten
grep	Dateien durchsuchen
head	Ausgabe der ersten Zeilen
kill	Prozesse töten
less	Seitenweise Ausgabe
ls	Ordnerinhalt auflisten
man	Hilfeseiten anzeigen
mv	Datei verschieben
nano	Einfacher Texteditor
ps	Prozesse anzeigen
pwd	Aktuelles Verzeichnis anzeigen
rm	Datei löschen
rmdir	Ordner löschen
sed	Manipulation von Textdaten
ssh	Shell auf entferntem Rechner
tail	Ausgabe der letzten Zeilen
tar	Archivierungswerkzeug
uniq	Gleiche Zeilen löschen
wc	Zeichen und Zeilen zählen

manpages – das Hilfesystem unter Unix

Typische Verwendung

`man <Befehl>`

man echo

```
ECHO(1)                    User Commands                    ECHO(1)

NAME
       echo - display a line of text

SYNOPSIS
       echo [OPTION]... [STRING]...

DESCRIPTION
       Echo the STRING(s) to standard output.

       -n     do not output the trailing newline
```

man 2025-10-27

Bedienung von `man`

Technische Universität Braunschweig

Die wichtigsten Tasten

- **Scrollen (zeilenweise)**: Pfeiltaste hoch/runter
- **Scrollen (seitenweise)**: Bild auf/ab
- **Suchen**: `/suchbegriff<ENTER>`
- **Nächster Treffer**: `n`
- **Vorheriger Treffer**: `N`
- **Beenden**: `q`

Tipp: Auch andere Befehle wie `less` lassen sich so bedienen!

Man Sections

Technische Universität Braunschweig

- Aufteilung in Kategorien, unter anderem:
  1. Ausführbare Programme oder Shell-Befehle
  2. Systemaufrufe (Kernel-Funktionen)
  3. Bibliotheksaufrufe (Funktionen in Programmbibliotheken)
  7. Verschiedenes (inkl. Makropaketen und Konventionen)

```
$ man -f write
write (1) - send a message to another user
write (2) - write to a file descriptor
write (3posix) - write on a file
write (1posix) - write to another user
```

Notation auf Übungszetteln: `manpage(section)`
Abruf der Manual-Page mit dem Befehl: `man -s <section> <manpage>`

Bonus: C - Antipattern

C - So geht's **nicht**

Technische Universität Braunschweig

- Problem:

```c
int stack_pop(stack_t *stack) {
  stack_elem elem* = stack->first;
  stack->first = stack->first->next;
  free(elem);
  return elem->value;
}
```

C - So geht's **nicht**

Technische Universität Braunschweig

- Problem: use after free

```
int stack_pop(stack_t *stack) {
  stack_elem elem* = stack->first;
  stack->first = stack->first->next;
  free(elem);
  return elem->value;
}
```

Der Speicher, auf den stack_elem zeigt, ist nach dem free() nicht mehr gültig!

C - So geht's **nicht**

Technische Universität Braunschweig

- Problem: use after free

```
int stack_pop(stack_t *stack) {
  stack_elem elem* = stack->first;
  stack->first = stack->first->next;
  free(elem);
  return elem->value;
}
```

Der Speicher, auf den stack_elem zeigt, ist nach dem free() nicht mehr gültig!

- Lösung: Zwischenspeichern

```
int stack_pop(stack_t *stack) {
  stack_elem *top_of_stack = stack->first;
  int tos_value = top_of_stack->value;
  stack->first = stack->first->next;
  free(top_of_stack);
  return tos_value;
  }
```

C - So geht's **nicht**

Technische Universität Braunschweig

```c
struct family {
  int num_adults;
  int num_children;
  int num_pets;
}
```

- Problem:

```c
struct family *create_family(int adults, int children, int pets) {
  struct family f;
  f.num_adults = adults;
  f.num_children = children;
  f.num_pets = pets;
  return &f;
}
```

C - So geht's **nicht**

Technische Universität Braunschweig

```c
struct family {
  int num_adults;
  int num_children;
  int num_pets;
}
```

- Problem: return local address

```c
struct family *create_family(int adults, int children, int pets) {
  struct family f;
  f.num_adults = adults;
  f.num_children = children;
  f.num_pets = pets;
  return &f;
}
```

f liegt auf dem Stack im Aufrufrahmen der create_family Funktion.
Beim Verlassen der Funktion verliert f die Gültigkeit.
Der zurückgegebene Zeiger zeigt damit auf ungültigen Speicher!

C - So geht's **nicht**

```
struct family {
  int num_adults;
  int num_children;
  int num_pets;
}
```

- Problem: return local address

```
struct family *create_family(int adults, int children, int pets) {
  struct family f;
  f.num_adults = adults;
  f.num_children = children;
  f.num_pets = pets;
  return &f;
}
```

f liegt auf dem Stack im Aufrufrahmen der create_family Funktion.
Beim Verlassen der Funktion verliert f die Gültigkeit.
Der zurückgegebene Zeiger zeigt damit auf ungültigen Speicher!

- Lösung: Dynamische Speicheranforderung

```
struct family *create_family(int parents, int children, int pets) {
  struct family *f = malloc(sizeof(struct family));
  f->num_adults = adults;
  f->num_children = children;
  f->num_pets = pets;
  return f;
}
```

C - So geht's **nicht**

Technische Universität Braunschweig

```c
struct family {
  int num_adults;
  int num_children;
  int num_pets;
}
```

- Problem:

```c
int calculate_theme_park_costs(int parents, int children, int pets) {
  struct family f = {parents,children,pets};
  int overall_costs = 0;
  int bus_costs = compute_bus_tickets(&f);
  int ticket_costs = compute_ticket_costs(&f);
  overall_costs = bus_costs + ticket_costs;
  return overall_costs;
}
```

C - So geht's **nicht**

Technische Universität Braunschweig

```c
struct family {
  int num_adults;
  int num_children;
  int num_pets;
}
```

- Problem:

```c
int calculate_theme_park_costs(int parents, int children, int pets) {
  struct family f = {parents,children,pets};
  int overall_costs = 0;
  int bus_costs = compute_bus_tickets(&f);
  int ticket_costs = compute_ticket_costs(&f);
  overall_costs = bus_costs + ticket_costs;
  return overall_costs;
}
```

C - So geht's ~~nicht~~ doch


```c
struct family {
  int num_adults;
  int num_children;
  int num_pets;
}
```

- Problem:

```c
int calculate_theme_park_costs(int parents, int children, int pets) {
  struct family f = {parents,children,pets};
  int overall_costs = 0;
  int bus_costs = compute_bus_tickets(&f);
  int ticket_costs = compute_ticket_costs(&f);
  overall_costs = bus_costs + ticket_costs;
  return overall_costs;
}
```

Kein Problem!
f „lebt“ länger, als der Funktionsaufruf dauert.

Gute Lösung für zeitlich begrenzten „dynamischen“ Speicher