# Tutorium

C Crashkurs & CoFee

Max Schrötter
schroetter@cs.uni-potsdam.de

Institute for Computational Science
University of Potsdam

17. Oktober 2025

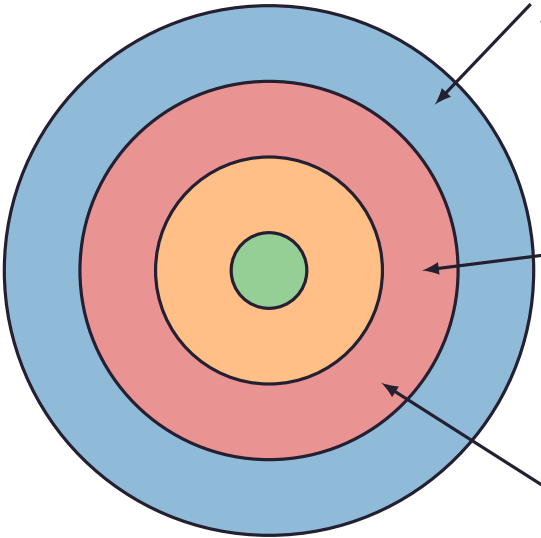
# Agenda

1. C in 30 Minuten
2. CoFee
3. Makefile

# Schlüsselwörter

| Datentypen | Anweisungen | | Modifiers | Typdeklarationen |
|---|---|---|---|---|
| char | if | for | const | struct |
| short | else | goto | static | union |
| int | while | return | volatile | enum |
| long | do | sizeof | extern | typedef |
| float | case | switch | register | |
| double | default | continue | inline | |
| void | break | | signed | |
| | | | unsigned | |
| | | | **restrict** | |

? Was passiert wenn ich Keywords ignoriere? →**undefiniertes Verhalten**!



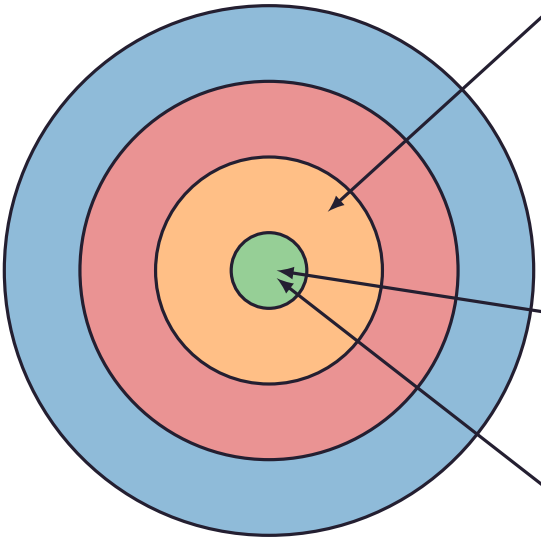
## Expressable in Assembly

```
add_i64_overflow:
    mov     qword ptr [rdx], rdi
    add     qword ptr [rdx], rsi
    seto    al
    ret
```

## Expressable in C

```
int int_overflow(int a) {
  assert(a + 100 > a);
  printf("%d %d\n",a + 100, a);
  return a; }
$ gcc test-ov.c
$ ./a.out
200 100
-2147483549 2147483647
```

⚠ undefined behavior



## C Standard Compliant

```c
int chkadd(int a, int b) {
  if(b < 0)
    return chksub(a,-b);
  else if(INT_MAX - b < a) {
    fputs("int overflow\n",
    ↪  stderr);
    return INT_MAX; }
  return a + b; }
```

## Secure C[1]

- Behandlung aller Fehlerfälle
- keine unsicheren Funktionen
- keine unsicheren Konvertierungen
- ...

✅ eure Abgaben

[1] According to SEI CERT C Coding Standard

# Non-C-Standard Error Handling

```c
void contains_null_check(int *p)
{
    int dead = *p;
    if(p==0)
        return;
    *p = 4;
}
```

```
0000000000001180 <collection_null_check>:
    1180:       c7 07 04 00 00 00       movl   $0x4,(%rdi)
    1186:       c3                      ret
```

**⚠ First check for errors then use the variable!**

The compiler otherwise assumes that the variable does not contain errors!

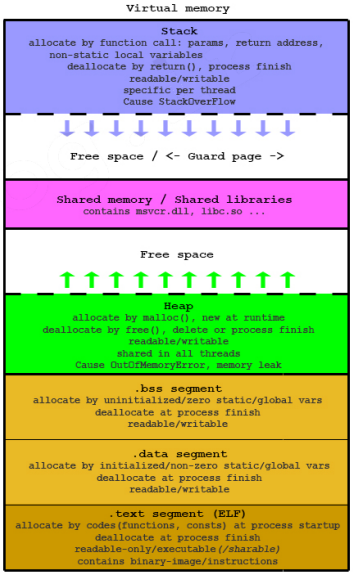
# Speicher

- Beim Definieren von lokalen Variablen wird Speicher auf dem Stack reserviert
- Variablen die auf dem Stack allokiert wurden werden am Ende des Blocks freigegeben
- Speicher aufm Stack ist limitiert (80KB alpine linux)
- **`calloc`**, `malloc` um persistenten und kontinuierlichen Speicher auf dem Heap zu reservieren

```
int* arr = malloc(sizeof(int)*5)
```

**C initialisiert Speicher nicht mit 0**

[1] http://yousha.blog.ir/post/45



# Pointer

Pointer sind Verweise(Zeiger) auf den Speicher.

- In einer Pointervariable wird die Adresse gespeichert auf den der Pointer verweist.
- Pointer haben einen Typ genau wie Variablen.

Deklaration:

- int *p;
- void *v;

Operationen

- * dereference: liefert den Wert auf den der Pointer zeigt
- & reference: liefert die Adresse der Variable

# Pointer-Arithmetik

```c
#include <stdio.h>

int main(int argc,char** argv) {
  double f=1.2;
  double *pf=&f;
  printf("%f \n",*pf);

  int arr[5] = {1,2,3,4,5};
  int* p=arr;
  printf("%d ",*p++);
  printf("%d ",p[1]);
}
```

**Arrays $\neq$ Pointer!**

# Declaration/Definition/Initialisierung

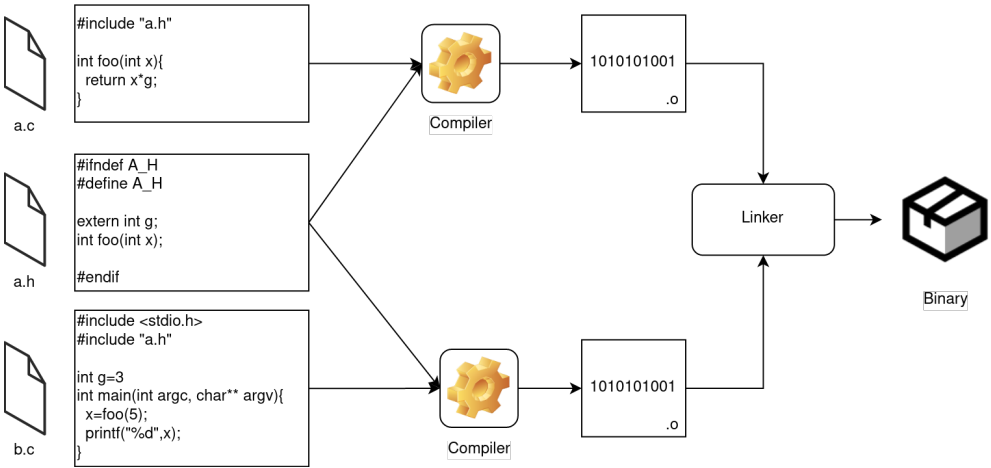
| | Variable | Funktion |
|---|---|---|
| Declaration | Bekanntmachen des Namens in der Translation Unit<br>`extern int a;`<br>`int a;` | Bekanntmachen des Funktionsprototyps<br>`int foo(void);` |
| Definition | Reservieren des Speichers<br>`int a;` | Funktionsprototyp + Implementation<br>`int foo(void) {return 1;}` |
| Initialisierung | Wertzuweisung<br>a=2; | - |

# Declaration/Definition/Initialisierung

| | Variable | Funktion |
|---|---|---|
| Declaration | Bekanntmachen des Namens in der Translation Unit<br>`extern int a;`<br>`int a;` | Bekanntmachen des Funktionsprototyps<br>`int foo(void);` |
| Definition | Reservieren des Speichers<br>`int a;` | Funktionsprototyp + Implementation<br>`int foo(void) {return 1;}` |
| Initialisierung | Wertzuweisung<br>a=2; | - |

Was ist eine Translation-Unit?

# C Compiler


a.c
#include "a.h"
int foo(int x){
return x*g;
}
a.h
#ifndef A_H
#define A_H
extern int g;
int foo(int x);
#endif
b.c
#include <stdio.h>
#include "a.h"
int g=3
int main(int argc, char** argv){
x=foo(5);
printf("%d",x);
}
Compiler
Compiler
1010101001
.o
1010101001
.o
Linker
Binary

# Scope

```c
#include <stdio.h>

int main(int argc, char ** argv){

  int x=0;
  for(int i=0;i<2;i++){
    printf("x:%d\n",x);
    printf("i:%d\n",i);
    int x=3;
    printf("x:%d\n",x);
  }
  printf("x:%d",x);
  printf("i:%d",i);
}
```

Was ist die Ausgabe?

| a) | b) | c) | d) |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 3 | 3 | 3 | 3 |
| 3 | 3 | 3 | 0 |
| 1 | 1 | 1 | 1 |
| 3 | 3 | 3 | 3 |
| 3 | 0 | 3 | 0 |
| 1 | 1 | e | e |

# Error Handling der C Standard Library

- C hat keine Exceptions →Fehlerbehandlung via Rückgabewert **!**
- viele lib-C Funktionen geben 0 bei Erfolg und -1 im Fehlerfall zurück
  →Ergebnisse werden via Parameter zurückgegeben →Pointer
- Wenn Fehlerindikator und Rückgabewert in einer Variable abbilden lassen wird diese zurückgegeben
  Beispiel: `malloc` oder `open`
- Grund für die Fehler wird häufig in der globalen Vairiable `errno` gespeichert
  →`perror` benutzen!

**Lest die Manpages !**

# Error Handling Konvention

- Funktionen geben im Fehlerfall negative Werte (Fehlercode) zurück, diser gibt auch an welchen Fehler aufgetreten ist!
  Beispiel: -1 kein Speicher, -2 Datei nicht gefunden ...
- Funktionen geben im Erfolgsfall 0 zurück
- beachtet die Aufgabenstellung

# Error Handling Pattern

## Early Return

```
int foo(int** mem, size_t size, int
↪  max) {
  if (size < 1) { return -1; }
  if (size < max) { return -2; }
  *mem = calloc(size, sizeof(int));
  if (*mem == NULL) { return -3; }
  for (int i = 0; i < max; i++) {
    (*mem)[i] = i;
  }
  return 0;
```

## Nested If

```
int foo(int** mem, size_t size, int
↪  max) {
  if (size >= 1) {
    if (size >= max) {
      *mem = calloc(size,
      ↪  sizeof(int));
      if (*mem != NULL) {
        for (int i = 0; i < max;
        ↪  i++)
        { (*mem)[i] = i; }
      } else { return -3; }
    } else { return -2; }
  } else { return -1; }
  return 0;
```

**❗ Benutzt das “Early Return” Pattern ❗**

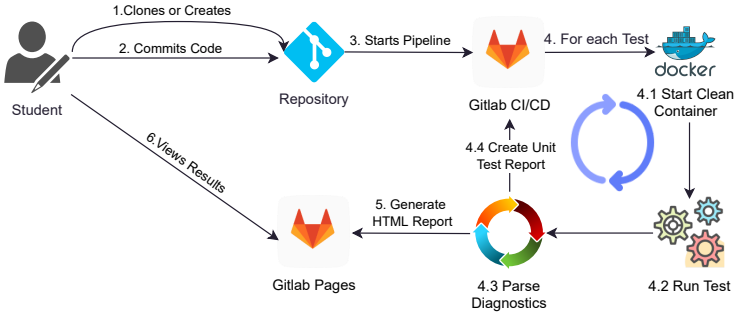
# Agenda

1. C in 30 Minuten
2. CoFee
3. Makefile

# CoFee - Continuous Feedback

- Framework zur Integration von Code Analyse für C-Hausaufgaben

CoFee
Continuous Feedback



# CoFee: Wie funktioniert es?

CoFee benutzt open Source Tools zur Code Analyse:

- **CSA** - Clang Static Analyzer
- **Clang-Tidy**
- **Sanitizer**
  - AddressSanitizer
  - MemorySanitizer
  - UndefinedBehaviorSanitizer
  - ThreadSanitizer
- **Valgrind**
- **CPPCheck**
- **GCC/LD**

# CoFee: Wie funktioniert es?

CoFee benutzt open Source Tools zur Code Analyse:

- **CSA** - Clang Static Analyzer
- **Clang-Tidy**
- **Sanitizer**
  - AddressSanitizer
  - MemorySanitizer
  - UndefinedBehaviorSanitizer
  - ThreadSanitizer
- **Valgrind**
- **CPPCheck**
- **GCC/LD**

Eigene Tools zum Finden von:

- fehlender Fehlerbehandlung
- veralteten Funktionen
- Unittests

Vielen Dank an:

- Matthias Habich
- Maximilian Falk
- Kai Schlabitz

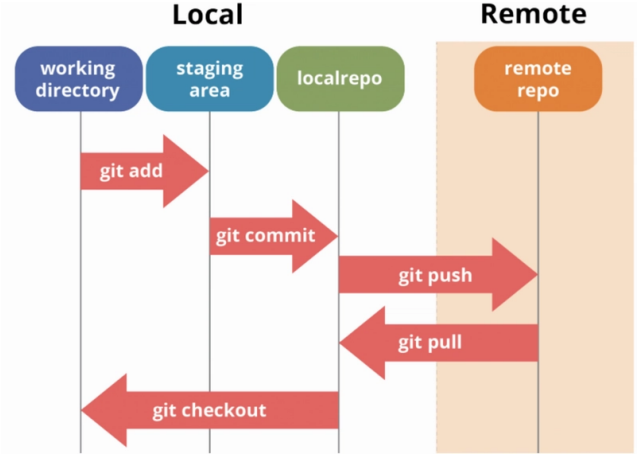
Bei Interesse Projekt und Bachelorarbeit möglich!

# CoFee: How to check your Code?

1. Login to `gitup.uni-potsdam.de`
2. Öffnen sie die Gruppe “GBR Vorlangen”
3. Wähle die Aufgabe für die Sie Code testen lass möchten
4. Wählen Sie Fork aus.
5. Wähle einen Projekt Namen und stelle sicher, dass in der **Projekt URL deine Gruppe ausgewählt ist**. →Create project.
6. Lade dein Quellcode hoch via git oder das + Symbol.

Anleitung: `https://www.cs.uni-potsdam.de/bs/teaching/docs/courses/ws2025/gbr/material/cofee_manual.pdf`

# Git Workflow



# Live Coding

# Agenda

1. C in 30 Minuten
2. CoFee
3. Makefile

# Makefile

- Makefiles enthalten die Anweisungen zum kompilieren eines Projektes
- Makefiles arbeiten dabei regelbasiert

```
target: prerequisites ...
        command
```

- Befehle sind durch einen **Tab** eingerückt!
- Bsp:

```
all: test.c
        gcc -Wall -Wextra --pedantic -o test
```

- Makefiles unterstützen Variablen und Patterns
- komplexeres Beispiel ist unter Material auf der Webseite zu finden: Makefile

# Hilfe

- C ist nicht einfach!
- Du kommst nicht weiter?
- Du weißt nicht was die Fehlermeldung bedeutet?
- Erstelle ein Ticket in deinem Projekt und beschreibe dein Problem!
- **Wichtig:** In der Beschreibung @Tutoren inkludiren!
  Autovervollständigung kann manchmal fehlschlagen!