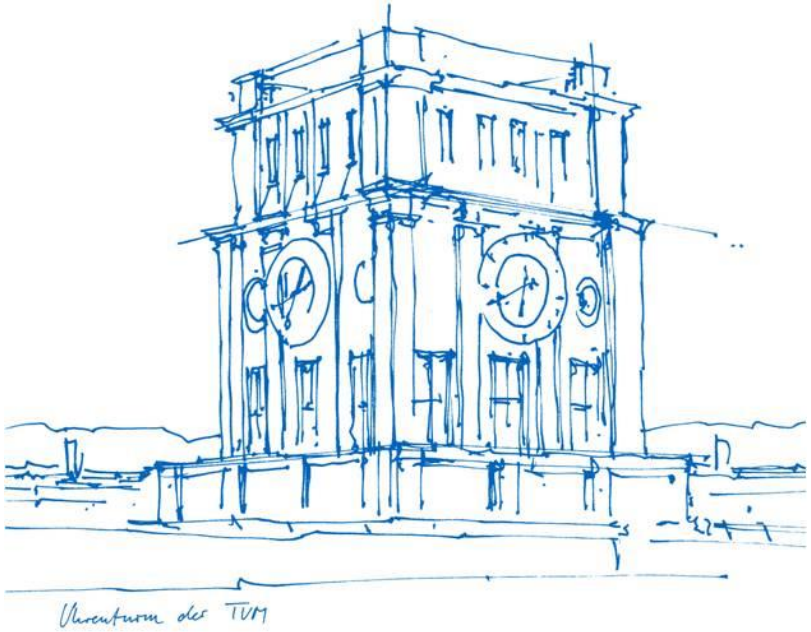
Grundlagenpraktikum: Rechnerarchitektur

SoSe 2025

~ *Danial Arbabi*
danial.arbabi@tum.de



Zulip-Gruppen

GRP 01: Montag 10:00

MI 03.13.10

#GRA/S - Tutorial-GRP-01

GRP 03: Montag 14:00

MI 01.06.20

#GRA/S - Tutorial-GRP-03

Tutoriums-Website

- https://home.cit.tum.de/~arb

oder

- https://arb.tum.sexy

Disclaimer:
Dies sind keine offiziellen Materialien, somit besteht keine Garantie auf Korrektheit und Vollständigkeit.
Falls euch Fehler auffallen, bitte gerne melden.

Programmierbeispiel: typedef

Wiederholung

File-I/O

fopen

- Öffnen von Files:

```
FILE *fopen(const char *pathname, const char *mode);
```

- `const char *pathname:` Pfad der Datei
- `const char *mode: Berechtigungen, mit denen die Datei geöffnet wird`
  - `O_RDONLY → « r »`
  - `O_WRONLY → « w »`
  - `O_RDWR → « r+ / w+ »`
  - `… « a / a+ »`
- `fopen` *(man 3 fopen)* verwendet im Hintergrund Systemcall `open` *(man 2 open)*

`Rückgabe: Pointer auf FILE-Struktur`

File-I/O

fclose

- Schließen von Files:

```
int fclose(FILE *stream);
```

- `FILE *stream`: Pointer auf Filestruktur

`Rückgabe: 0 wenn erfolgreich, sonst EOF und errno gesetzt`

File-I/O

fread

- Lesen von Files:

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

- `void *ptr:` Speicherbereich, in dem Fileinhalt gespeichert wird
- `size_t size:` Größe der zu lesenden Items in Bytes
- `size_t nmemb:` Anzahl der zu lesenden Items
- `FILE *stream:` File, aus der gelesen werden soll

`Rückgabe:` Anzahl der gelesenen Items

File-I/O

fwrite

- Schreiben von Files:

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

- `const void *ptr`: Pointer auf einen konstanten Speicherbereich, aus dem gelesen wird
- `size_t size`: Größe der zu schreibenden Items in Bytes
- `size_t nmemb`: Anzahl der zu lesenden Items
- `FILE *stream`: File, die beschrieben wird

`Rückgabe:` Anzahl gelesener Items

File-I/O

fstat

- `Erhalten von File Informationen:`

```
int fstat(int fd, struct stat *statbuf);
```

- `int fd:` Filedeskriptor (bekommt man mit fileno(FILE*))
- `struct stat *statbuf:` Pointer auf stat Struktur, worin die Information geschrieben wird
- z.B. `statbuf->st_size` für die Dateigröße

`Rückgabe:` 0 wenn erfolgreich, ansonsten -1 (errno wird gesetzt)

Aufgabe:

T4-1 File IO

Makefile

Was passiert in den beiden Zeilen?

```
main: main.c xor_cipher.c
	$(CC) $(CFLAGS) -o $@ $^
```

Makefile

Was passiert in den beiden Zeilen?

```
main: main.c xor_cipher.c
    $(CC) $(CFLAGS) -o $@ $^
```

- Ein solches Konstrukt heißt „`Rule`“

`target: prerequisite1 prerequisite2 ...` (prerequisite kann auch ein weiteres Target sein)

```
        recipe1
        recipe2
```

- Quelldateien: `main.c` und `xor_cipher.c`
- Shell-Befehle: `$(CC) $(CFLAGS) -o $@ $^` (kann z.B. auch “`echo Hallo`” sein)
- `Output-Name: main`

Makefile

Woher kommt $(CC) und $(CFLAGS)

```
main: main.c xor_cipher.c
    $(CC) $(CFLAGS) -o $@ $^
```

Makefile

Woher kommt $(CC) und $(CFLAGS)

```
main: main.c xor_cipher.c
    $(CC) $(CFLAGS) -o $@ $^
```

- CFLAGS wird ganz oben definiert:

```
# Add additional compiler flags here
CFLAGS=-O0
```

Makefile

Woher kommt $(CC) und $(CFLAGS)

```
main: main.c xor_cipher.c
    $(CC) $(CFLAGS) -o $@ $^
```

- CFLAGS wird ganz oben definiert:

```
# Add additional compiler flags here
CFLAGS=-O0
```

- Woher kommt aber CC?

→ https://www.gnu.org/software/make/manual/html_node/Implicit-Variables.html

Makefile

Womit wird nun $@ und $^ ersetzt?

```
main: main.c xor_cipher.c
	$(CC) $(CFLAGS) -o $@ $^
```

https://www.gnu.org/software/make/manual/html_node/Automatic-Variables.html

Makefile

Womit wird nun $@ und $^ ersetzt?

```
main: main.c xor_cipher.c
    $(CC) $(CFLAGS) -o $@ $^
```

- $@ = The file name of the target of the rule
- $^ = The names of all the prerequisites, with spaces between them

Makefile

Ausführen

- Beim Ausführen von make werden zubauende Targets angegeben
- Wenn **keins** spezifiziert, wird das **erste** Target genommen

Makefile

Was passiert nach diesen Befehlen?

- make
- make main
- make clean all
- make CFLAGS=-O3 -Wall -Wextra
- make -j2

Makefile

Was passiert nach diesen Befehlen?

- make
  - Hier wird das erste Target gebaut, in diesem Fall `all`.
- make main
- make clean all
- make CFLAGS=-O3 -Wall -Wextra
- make -j2

Makefile

Was passiert nach diesen Befehlen?

- make
  - Hier wird das erste Target gebaut, in diesem Fall `all`.
- make main
  - Hier wird das Target `main` gebaut.
- make clean all
- make CFLAGS=-O3 -Wall -Wextra
- make -j2

Makefile

Was passiert nach diesen Befehlen?

- make
  - Hier wird das erste Target gebaut, in diesem Fall `all`.
- make main
  - Hier wird das Target `main` gebaut.
- make clean all
  - Hier wird zuerst das Target `clean` ``gebaut'', dann `all`.
- make CFLAGS=-O3 -Wall -Wextra

- make -j2

Makefile

Was passiert nach diesen Befehlen?

- make
  - Hier wird das erste Target gebaut, in diesem Fall `all`.
- make main
  - Hier wird das Target `main` gebaut.
- make clean all
  - Hier wird zuerst das Target `clean` ``gebaut'', dann `all`.
- make CFLAGS=-O3 -Wall -Wextra
  - Hier wird das erste Target gebaut, zudem werden die `CFLAGS` angepasst.
- make -j2

Makefile

Was passiert nach diesen Befehlen?

- make
  - Hier wird das erste Target gebaut, in diesem Fall `all`.
- make main
  - Hier wird das Target `main` gebaut.
- make clean all
  - Hier wird zuerst das Target `clean` ``gebaut'', dann `all`.
- make CFLAGS=-O3 -Wall -Wextra
  - Hier wird das erste Target gebaut, zudem werden die `CFLAGS` angepasst.
- make -j2
  - Spezifiziert parallel laufenden Jobs (Kommands)

Makefile

Mehr Mals make? + PHONY

`Mehrmals make:`

- `make` verfolgt Dependencies auf Basis der Modifikationszeit
- Ist ein `target` neuer als alle `prerequisites`, muss es nicht erneut gebaut werden

PHONY:

- `.PHONY: <target>` (z.B. .PHONY: clean)
- Konflikt mit Datei mit selbem Namen wie `<target>` vermeiden, die nichts mit dem Target zu tun hat

Aufgabe:

T4-3 Makefile

	Befehl	Beschreibung
	`run`	Führt Programm innerhalb von gdb aus
`b`	`break <ZeilenNr/Funktion>`	Setzt Breakpoint an angegebener ZeilenNr/Funktion
`p`	`print <Variable>`	Zeigt Wert von Variable an
	`x <Addr>`	Inspektion von Speicherinhalten an angegebener Adresse
	`x/[Format] <Addr>`	Identisch zu `x` mit Spezifikation von Ausgabeformat
	`x/[Länge][Format] <Addr>`	Identisch zu `x/[Format]` mit Spezifikation von Anzahl angezeigter Elemente
`s`	`step`	Führt Programm schrittweise aus, tritt in Funktionen ein
`n`	`next`	Führt Programm schrittweise aus, Funktionsaufruf ein Schritt
`c`	`continue`	Setzt Ausführung bis nächsten Breakpoint fort
	`finish`	Verlässt aktuelle Funktion und fährt mit Ausführung fort
	`info break`	Zeigt Informationen über Breakpoints an
	`delete <Breakpoint-Nr>`	Löscht angegebenen Breakpoint
	`disable <Breakpoint-Nr>`	Deaktiviert angegeben Breakpoint
	`enable <Breakpoint-Nr>`	Aktiviert angegeben Breakpoint

Nützliche gdb Befehle

Aufgabe:

T4-2 Valgrind