C-Vorkurs, Tag 1, 11. April 2016

Jonas Cleve, Sebastian Faase, Thomas Tegethoff

Organisatorisches

- Kurs ist freiwillig und gibt auch keine LP / ECTS
- Vorlesungen bauen aufeinander auf
- Kontinuierliche Anwesenheit sinnvoll
- Programmiert wird auf 64bit-Linux-Rechnern (z.B. Poolrechner)
- Unterstützung von Mentoring-Programm des Fachbereichs[1]
- Folien werden veröffentlicht auf: `https://page.mi.fu-berlin.de/jonascleve/ss16-c-kurs/`
- Nach der Veranstaltung im KVV
- Tutorien: Ihr arbeitet an Aufgaben, wir helfen euch

[1] `http://www.mi.fu-berlin.de/stud/mentoring/`

Vorkenntnisse

- Wer hat noch keine Programmiererfahrung?

Vorkenntnisse

- Wer hat noch keine Programmiererfahrung?
- Wem sind folgende Konzepte (noch) unklar:

Vorkenntnisse

- Wer hat noch keine Programmiererfahrung?
- Wem sind folgende Konzepte (noch) unklar:
  - Bedingte Code-Ausführung (`if`)?

Vorkenntnisse

- Wer hat noch keine Programmiererfahrung?
- Wem sind folgende Konzepte (noch) unklar:
  - Bedingte Code-Ausführung (`if`)?
  - Schleifen (z.B. `while`, `for`, `...`)?

Vorkenntnisse

- Wer hat noch keine Programmiererfahrung?
- Wem sind folgende Konzepte (noch) unklar:
  - Bedingte Code-Ausführung (`if`)?
  - Schleifen (z.B. `while`, `for`, `...`)?
  - Funktionen?

Ziele des Kurses

- Einführung in die Programmiersprache C
- Vermittlung der wichtigsten Grundkenntnisse für das Programmieren in C
- Weniger Angst vor C
- Nach dem Kurs sollt ihr in der Lage sein:
  - Die Betriebs- und Kommunikationssysteme-Übungen selbstständig zu lösen
  - Euch eventuell fehlende Kenntnisse selbstständig anzueignen

Heutiger Inhalt

- Vorlesung
  - Grundstruktur eines C-Programms
  - Programme kompilieren
  - Text ausgeben
- Tutorium
  - Linux kennenlernen
  - Konsole benutzen
  - Benutzbare Arbeitsumgebung schaffen

Erste Schritte in C

Hallo Welt

Aufgabe: Ein Programm schreiben, welches “Hallo Welt!” ausgibt und sich beendet.
Diese Spezifikation soll eingehalten werden.

Hallo Welt

Aufgabe: Ein Programm schreiben, welches “Hallo Welt!” ausgibt und sich beendet.
Diese Spezifikation soll eingehalten werden.

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    printf("Hallo Welt!\n");
    return EXIT_SUCCESS;
}
```

Hallo Welt

```c
// Standard-Ein-/Ausgabe verwenden
#include <stdio.h>
// Deklaration von EXIT_SUCCESS verwenden
#include <stdlib.h>

// Unser Programm ist in der main-Funktion
int main() {
    // Text "Hallo Welt!" + Zeilenumbruch ausgeben
    printf("Hallo Welt!\n");
    // Mitteilen, dass das Programm erfolgreich war
    return EXIT_SUCCESS;
}
```

Kompilieren

- Quellcode kann so nicht ausgeführt werden
- Kompilieren (Übersetzen in Maschinenbefehle) nötig

Kompilieren

- Quellcode kann so nicht ausgeführt werden
- Kompilieren (Übersetzen in Maschinenbefehle) nötig

- Wir verwenden dafür den `GCC`

```
gcc hello-world.c -o hello-world
#   ^ C-Datei          ^ ausführbare Datei
```

Kompilieren

- Quellcode kann so nicht ausgeführt werden
- Kompilieren (Übersetzen in Maschinenbefehle) nötig

- Wir verwenden dafür den `GCC`

```
gcc hello-world.c -o hello-world
#   ^ C-Datei          ^ ausführbare Datei
```

- Programm testen:

```
$ ./hello-world
Hallo Welt!
$
```

Kompilieren II

- Dem Compiler können zusätzliche Optionen übergeben werden
- Im Kurs kompilieren wir *immer* mit folgenden Optionen:

```
gcc -std=c11 -Wall -Wextra -pedantic -o A A.c
```

  - `-std=c11`: Wir verwenden den C-Standard von 2011
  - `-Wall`: Schalte viele nützliche Warnungen an
  - `-Wextra`: Schalte weitere nützliche Warnungen an
  - `-pedantic`: Erzwinge Befolgung des ISO-C-Standards
  - `-o A`: Die Eingabe `A.c` wird in ausführbare Datei `A` umgewandelt
- Wir wollen *keine* Warnung ignorieren $\rightarrow$ guter Programmierstil
- Man kann bei C viel "Unsinn" machen, ohne das zu merken

Bunter Text

- Jetzt: wollen Text bunt färben
- Gewünschte Ausgabe: Hallo Welt!

Bunter Text

- Jetzt: wollen Text bunt färben
- Gewünschte Ausgabe: Hallo Welt!

- Spezielle Textteile für farbigen Text:
  - `"\033[31m"` für roten Text
  - `"\033[39m"` für "normalen" Text

Bunter Text

- Jetzt: wollen Text bunt färben
- Gewünschte Ausgabe: Hallo Welt!

- Spezielle Textteile für farbigen Text:
  - "\033[31m" für roten Text
  - "\033[39m" für "normalen" Text

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    printf("Hallo ");
    printf("\033[31m");      // ab hier alles ROT
    printf("Welt");
    printf("\033[39m!\n");   // vor "!" wieder normal
    return EXIT_SUCCESS;
}
```

Mehr Farbwechsel

- Gewünschte Ausgabe: Hallo Welt!

Mehr Farbwechsel

- Gewünschte Ausgabe: Hallo Welt!

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    printf("\033[31mH\033[39ma\033[31ml\033[39ml");
    printf("\033[31mo \033[39mW\033[31me\033[39ml");
    printf("\033[31mt\033[39m!\n");

    return EXIT_SUCCESS;
}
```

Mehr Farbwechsel

- Gewünschte Ausgabe: Hallo Welt!

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    printf("\033[31mH\033[39ma\033[31ml\033[39ml");
    printf("\033[31mo \033[39mW\033[31me\033[39ml");
    printf("\033[31mt\033[39m!\n");

    return EXIT_SUCCESS;
}
```

- Versteht noch jemand was?

Alternativlösung

```c
#include <stdio.h>
#include <stdlib.h>

#define RED     "\033[31m"
#define NORMAL  "\033[39m"

int main() {
    printf(RED "H" NORMAL "a" RED "l" NORMAL "l");
    printf(RED "o " NORMAL "W" RED "e" NORMAL "l");
    printf(RED "t" NORMAL "!\n");

    return EXIT_SUCCESS;
}
```

#define

- Mittels `#define` kann *Text ersetzt* werden, *bevor* das Programm wirklich kompiliert wird

#define

- Mittels `#define` kann *Text ersetzt* werden, *bevor* das Programm wirklich kompiliert wird

- ```
  #define RED       "\033[31m"
  #define NORMAL    "\033[39m"
  printf(RED "H" NORMAL "a" RED "l");
  ```
  ```

#define

- Mittels `#define` kann *Text ersetzt* werden, *bevor* das Programm wirklich kompiliert wird
- ```
  #define RED       "\033[31m"
  #define NORMAL    "\033[39m"
  printf(RED "H" NORMAL "a" RED "l");
  ```
- wird zu

  ```
  printf("\033[31m" "H" "\033[39m" "a" "\033[31m" "l");
  ```
  ```

#define

- Mittels `#define` kann *Text ersetzt* werden, *bevor* das Programm wirklich kompiliert wird
- ```
  #define RED       "\033[31m"
  #define NORMAL    "\033[39m"
  printf(RED "H" NORMAL "a" RED "l");
  ```
- wird zu

  ```
  printf("\033[31m" "H" "\033[39m" "a" "\033[31m" "l");
  ```

- Die so definierten *Konstanten* heißen *Makros* und werden zur Abgrenzung vollständig groß geschrieben
  ```

Hallo “Name”

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    printf("Hallo \"");
    printf(argv[1]);
    printf("\"\n");
    return EXIT_SUCCESS;
}
```

Hallo “Name” – sichere Variante

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    printf("Hallo \"");
    printf("%s", argv[1]);
    printf("\"\n");
    return EXIT_SUCCESS;
}
```

`printf`

- Ist eine Funktion mit *variabler* Parameteranzahl
  - `int printf(const char *format, ...)`
  - *mindestens* ein Parameter
  - *Platzhalter* im ersten Parameter setzen nachfolgende Werte ein

printf

- Ist eine Funktion mit *variabler* Parameteranzahl
  - `int printf(const char *format, ...)`
  - *mindestens* ein Parameter
  - *Platzhalter* im ersten Parameter setzen nachfolgende Werte ein
- Beispiele:
  - `printf("A", "B") => "A"`
  - `printf("A%s", "B") => "AB"`
  - `printf("%sA", "B") => "BA"`
  - `printf("1: %s, 2: %d, 3: %f", "Hi", 423, 354.2) =>`
    `"1: Hi, 2: 423, 3: 354.200000"`

printf

- Ist eine Funktion mit *variabler* Parameteranzahl
  - `int printf(const char *format, ...)`
  - *mindestens* ein Parameter
  - *Platzhalter* im ersten Parameter setzen nachfolgende Werte ein
- Beispiele:
  - `printf("A", "B") => "A"`
  - `printf("A%s", "B") => "AB"`
  - `printf("%sA", "B") => "BA"`
  - `printf("1: %s, 2: %d, 3: %f", "Hi", 423, 354.2) =>`
    `"1: Hi, 2: 423, 3: 354.200000"`
- Auswahl möglicher Platzhalter:
  - `%s` – Text
  - `%d` – Zahlen (auch negativ), `%u` – Zahlen (nur positiv)
  - `%f` – Kommazahlen

Was haben wir (bisher) heute gelernt?

Was haben wir (bisher) heute gelernt?

- Allgemein
  - Grundstruktur eines C-Programms
  - Aufgabenstellungen sollten exakt bearbeitet werden
  - Quellcode sinnvoll formatieren
- C-spezifisch
  - Programme kompilieren
  - Text ausgeben
  - Bunten Text ausgeben
  - Makros (`#define`) für oft wiederholten / unübersichtlichen Code definieren
  - `printf`