

Hello C

C-Kurs 2014, 1. Vorlesung

Thilo Michael
Folien: Theresa Enghardt

<http://wiki.freitagrunde.org>

15. September 2014



This work is licensed under the *Creative Commons Attribution-ShareAlike 3.0 License*.

- ▶ ISIS-Kurs (Infos, Links):
<https://www.isis.tu-berlin.de/course/view.php?id=8675>
- ▶ Vortragsfolien:
https://docs.freitagrunde.org/Veranstaltungen/ckurs_2014/unterlagen/ (bald)
- ▶ Übungsaufgaben:
<http://wiki.freitagrunde.org/Ckurs/Übungsaufgaben>
- ▶ Buch „C von A bis Z“ (online frei verfügbar):
http://openbook.galileocomputing.de/c_von_a_bis_z/

- 1 Hello, C-Kurs
- 2 Funktionen
- 3 Datentypen und Operatoren
- 4 Programmfluss
- 5 Hilfestellungen
- 6 Häufige Fehler
- 7 Zusammenfassung



4!

Schreiben: Eure Arbeitsumgebung

- ▶ Eigener Rechner (Linux, Mac OS, Windows...)
- ▶ oder eine Workstation im TEL 106/206 (vorzugsweise mit Linux)
- ▶ C-Compiler ist normalerweise schon installiert,
wenn nicht: `sudo apt-get install build-essential`
- ▶ einfacher Texteditor

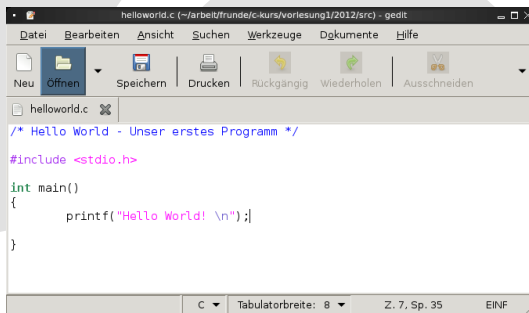


4!

Schreiben: Eure Arbeitsumgebung

- ▶ Eigener Rechner (Linux, Mac OS, Windows...)
- ▶ oder eine Workstation im TEL 106/206 (vorzugsweise mit Linux)
- ▶ C-Compiler ist normalerweise schon installiert,
wenn nicht: `sudo apt-get install build-essential`
- ▶ einfacher Texteditor

- ▶ **gedit** (Linux)
- ▶ **nano** (Linux)
- ▶ **notepad++** (Windows)



The screenshot shows the gedit text editor window titled "helloworld.c (~/.arbeit/frunde/c-kurs/vorlesung1/2012/src) - gedit". The menu bar includes Datei, Bearbeiten, Ansicht, Suchen, Werkzeuge, Dokumente, and Hilfe. The toolbar contains icons for Neu, Öffnen, Speichern, Drucken, Rückgängig, Wiederholen, and Ausschneiden. The file name "helloworld.c" is shown in the tab. The code content is as follows:

```
/* Hello World - Unser erstes Programm */  
  
#include <stdio.h>  
  
int main()  
{  
    printf("Hello World! \n");  
}
```

The status bar at the bottom indicates the encoding is "C", the tab width is "Tabulatorbreite: 8", the current position is "Z. 7, Sp. 35", and the document is "EINF" (unmodified).

Schreiben: Hello World

src/helloworld.c

```
1 /* Hello World –  
2 Unser erstes Programm */  
3 #include <stdio.h>  
4  
5 int main()  
6 {  
7     printf("Hello World! \n");  
8     return 0;  
9 }
```

Schreiben: Hello World

src/helloworld.c

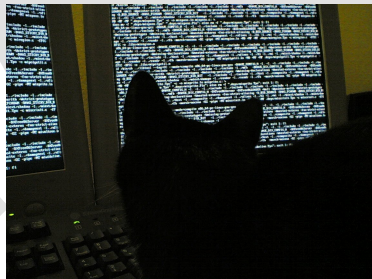
```
1  /* Hello World –  
2  Unser erstes Programm */  
3  #include <stdio.h>  
4  
5  int main()  
6  {  
7      printf("Hello World! \n");  
8      return 0;  
9  }
```

- ▶ Zeilen 1 und 2: Kommentar (kann mehrzeilig sein)
- ▶ Zeile 3: Standardfunktionen für Ein-/Ausgabe werden eingebunden
- ▶ Zeile 5: Funktion *main*: Keine Argumente, Rückgabetyt Integer (int)
- ▶ Zeile 7: Befehl, der Text ausgibt
- ▶ Zeile 8: Rückgabewert der Funktion
- ▶ Jeder Befehl endet mit einem Semikolon

Kompilieren

Der Compiler übersetzt den Quellcode in ein ausführbares Programm.

Wir benutzen die **GNU Compiler Collection (gcc)**



- ▶ Quellcode speichern (Dateiname beliebig, Endung .c)
- ▶ Shell öffnen (Kommandozeile)
 - ▷ z.B. gnome-terminal (Linux), Terminal (Mac OS), cmd (Windows)

```
$ gcc helloworld.c -o HelloWorld
$
```

Kompilieren eines C-Programms

Kompilieren: Optionen

```
$ ls -l
-rwxr-xr-x 1 theresa theresa 4874 Aug 30 19:08 HelloWorld
-rw-r--r-- 1 theresa theresa  131 Aug 30 19:08 helloworld.c
```

Ergebnis des Kompilierens

Wichtige Optionen für gcc:

Option	Beschreibung
-o <i>filename</i>	Ausgabedatei bekommt den Namen <i>filename</i>
-Wall	Warnungen werden angezeigt
-Wextra	Noch mehr Warnungen werden angezeigt
-std=c99	Benutze C99-Standard (Version der Sprache C von 1999: ggf andere Schlüsselwörter und sonstige Feinheiten)

```
$ gcc --help
Usage: gcc [options] file...
Options: [...]
```

Optionen auflisten

```
$ ./HelloWorld  
Hello World!  
$
```

Ausgabe unseres Programms

- ▶ Compiler erzeugt ausführbaren Binärcode, d.h. Anweisungen, die die CPU versteht und direkt ausführen kann
- ▶ Daher kein Zusatzprogramm, keine Laufzeitumgebung wie bei Java notwendig!
- ▶ Aber: Ihr müsst für jede Plattform (Typ der CPU) neu kompilieren.
- ▶ Unter Linux kompiliertes Programm läuft nicht unter Windows

1 Hello, C-Kurs

2 Funktionen

3 Datentypen und Operatoren

4 Programmfluss

5 Hilfestellungen

6 Häufige Fehler

7 Zusammenfassung



4!

Funktionen: Deklaration und Definition

Deklarieren: Bekanntgeben, dass diese Funktion existiert

```
Rückgabetyt Funktionsname(Parametertyp1, ...);
```

Definieren: Festlegen, was diese Funktion tut (ggf. gleichzeitig deklarieren)

```
Rückgabetyt Funktionsname(Parametertyp1 Parametername1, ...)
{
    Anweisung1;
    ...
    return Wert;
}
```

Achtung:

Funktionen müssen immer erst deklariert werden, bevor sie benutzt werden!

Funktionen benutzen

src/funktionen.c

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int zahl = 5;
6     zahl = pluszwei(zahl);
7     return 0;
8 }
9
10 int pluszwei(int input)
11 {
12     return input + 2;
13 }
```

Führt ggf. zu einer Warnung des Compilers:

```
$ gcc funktionen.c -o fun -Wall
funktionen.c: In function 'main':
funktionen.c:6:2: warning: implicit declaration of function
      'pluszwei' [-Wimplicit-function-declaration]
```

Warnung: Implizite Deklaration

Funktionen benutzen

Mögliche Lösungen:

- ▶ Reihenfolge der Funktionen tauschen
oder
- ▶ Funktionsdeklaration, bevor sie benutzt wird

src/funktionen2.c

```
1 #include <stdio.h>
2 int pluszwei(int);
3
4 int main()
5 {
6     int zahl = 5;
7     zahl = pluszwei(zahl);
8     return 0;
9 }
10
11 int pluszwei(int input)
12 {
13     return input + 2;
14 }
```

- 1 Hello, C-Kurs
- 2 Funktionen
- 3 Datentypen und Operatoren**
- 4 Programmfluss
- 5 Hilfestellungen
- 6 Häufige Fehler
- 7 Zusammenfassung



4!

Ganze Zahlen mit Wertebereich¹

	Anzahl Bits	signed	unsigned
char	8	-128 .. 127	0 .. 255
short	16	-32768 .. 32767	0 .. 65535
int	32	-2147483648 .. 2147483647	0 .. 4294967295
long long int	64	-9.22E+18 .. 9.22E+18	0 .. 1.84E+19

Fließkommazahlen mit Wertebereich

float	1.40E-38 .. 3.40E+38
double	4.94E-308 .. 1.80E+308

Kein Boolean: Zahlen sind Wahrheitswerte, 0 ist False, alles andere True

Kein String: Array von Buchstaben (char), ähnlich einzeilige Tabelle

¹Gilt für 32-Bit-Architekturen

src/array.c

```
int i = 1;  
int zahlenfolge[32]; /* Eine Folge von 32 Zahlen vom  
    Typ int */
```

```
zahlenfolge[0] = 3;  
zahlenfolge[i] = 9876;  
zahlenfolge[2] = zahlenfolge[1] + 10;  
zahlenfolge[31] = -23;
```

- ▶ Länge des Arrays muss bei der Deklaration statisch festgelegt werden
- ▶ Man muss sie sich (z.B. für Schleifen) selbst merken, am besten in einer Variable speichern!
- ▶ Elegantere Lösungen lernt ihr später kennen.

Operatoren: Überblick

Operator	Kategorie
<, >, <=, >=, ==, !=	Vergleichsoperatoren
+, -, *, /, %	Rechenoperationen
!, &&,	Logische Operatoren
++, --	Prefix-/Postfix- Inkrement/Dekrement
+=, -=, *=, /=, %=	Rechenoperation mit Zuweisung z.B. <code>a += 3</code> entspricht <code>a = a + 3</code>

- ▶ Es gilt Punkt- vor Strichrechnung
- ▶ Im Zweifelsfall: Klammern setzen
- ▶ Zusätzlich: Bitweise Operatoren, siehe nächste Folie...

Bitweise Operatoren: Überblick

Bitweise Operatoren

Zahlen als Binärzahlen vorstellen, dann die beiden ersten Bits miteinander verknüpfen, dann die zweiten Bits, dann die dritten...

Operator	Bezeichnung
&	Bitweise Und-Verknüpfung $10 \& 12 = \quad 1010_2 \& 1100_2 =$

4!

Bitweise Operatoren: Überblick

Bitweise Operatoren

Zahlen als Binärzahlen vorstellen, dann die beiden ersten Bits miteinander verknüpfen, dann die zweiten Bits, dann die dritten...

Operator	Bezeichnung
&	Bitweise Und-Verknüpfung $10 \& 12 = \quad 1010_2 \& 1100_2 = \quad 1000_2 = \quad 8$
	Bitweise Oder-Verknüpfung $10 12 = \quad 1010_2 1100_2 =$

4!

Bitweise Operatoren: Überblick

Bitweise Operatoren

Zahlen als Binärzahlen vorstellen, dann die beiden ersten Bits miteinander verknüpfen, dann die zweiten Bits, dann die dritten...

Operator	Bezeichnung
&	Bitweise Und-Verknüpfung $10 \& 12 = 1010_2 \& 1100_2 = 1000_2 = 8$
	Bitweise Oder-Verknüpfung $10 12 = 1010_2 1100_2 = 1110_2 = 14$
^	Bitweise XOR-Verknüpfung $10 \wedge 12 = 1010_2 \wedge 1100_2 =$

Bitweise Operatoren: Überblick

Bitweise Operatoren

Zahlen als Binärzahlen vorstellen, dann die beiden ersten Bits miteinander verknüpfen, dann die zweiten Bits, dann die dritten...

Operator	Bezeichnung
&	Bitweise Und-Verknüpfung $10 \& 12 = 1010_2 \& 1100_2 = 1000_2 = 8$
	Bitweise Oder-Verknüpfung $10 12 = 1010_2 1100_2 = 1110_2 = 14$
^	Bitweise XOR-Verknüpfung $10 \wedge 12 = 1010_2 \wedge 1100_2 = 0110_2 = 6$
<<	Bitweise Shift (Verschiebung) nach links $11 << 2 = 1011_2 << 2 =$

Bitweise Operatoren: Überblick

Bitweise Operatoren

Zahlen als Binärzahlen vorstellen, dann die beiden ersten Bits miteinander verknüpfen, dann die zweiten Bits, dann die dritten...

Operator	Bezeichnung
&	Bitweise Und-Verknüpfung $10 \& 12 = 1010_2 \& 1100_2 = 1000_2 = 8$
	Bitweise Oder-Verknüpfung $10 12 = 1010_2 1100_2 = 1110_2 = 14$
^	Bitweise XOR-Verknüpfung $10 \wedge 12 = 1010_2 \wedge 1100_2 = 0110_2 = 6$
<<	Bitweise Shift (Verschiebung) nach links $11 << 2 = 1011_2 << 2 = 101100_2 = 44$
>>	Bitweise Shift (Verschiebung) nach rechts $11 >> 2 = 1011_2 >> 2 =$

Bitweise Operatoren: Überblick

Bitweise Operatoren

Zahlen als Binärzahlen vorstellen, dann die beiden ersten Bits miteinander verknüpfen, dann die zweiten Bits, dann die dritten...

Operator	Bezeichnung
&	Bitweise Und-Verknüpfung $10 \& 12 = 1010_2 \& 1100_2 = 1000_2 = 8$
	Bitweise Oder-Verknüpfung $10 12 = 1010_2 1100_2 = 1110_2 = 14$
^	Bitweise XOR-Verknüpfung $10 \wedge 12 = 1010_2 \wedge 1100_2 = 0110_2 = 6$
<<	Bitweise Shift (Verschiebung) nach links $11 << 2 = 1011_2 << 2 = 101100_2 = 44$
>>	Bitweise Shift (Verschiebung) nach rechts $11 >> 2 = 1011_2 >> 2 = 10_2 = 2$
~	Bitweises Komplement (Umkehrung) $\sim 10 = \sim 1010_2 =$

Bitweise Operatoren: Überblick

Bitweise Operatoren

Zahlen als Binärzahlen vorstellen, dann die beiden ersten Bits miteinander verknüpfen, dann die zweiten Bits, dann die dritten...

Operator	Bezeichnung
&	Bitweise Und-Verknüpfung $10 \& 12 = 1010_2 \& 1100_2 = 1000_2 = 8$
	Bitweise Oder-Verknüpfung $10 12 = 1010_2 1100_2 = 1110_2 = 14$
^	Bitweise XOR-Verknüpfung $10 \wedge 12 = 1010_2 \wedge 1100_2 = 0110_2 = 6$
<<	Bitweise Shift (Verschiebung) nach links $11 << 2 = 1011_2 << 2 = 101100_2 = 44$
>>	Bitweise Shift (Verschiebung) nach rechts $11 >> 2 = 1011_2 >> 2 = 10_2 = 2$
~	Bitweises Komplement (Umkehrung) $\sim 10 = \sim 1010_2 = 0101_2 = 5$

- 1 Hello, C-Kurs
- 2 Funktionen
- 3 Datentypen und Operatoren
- 4 Programmfluss**
- 5 Hilfestellungen
- 6 Häufige Fehler
- 7 Zusammenfassung



4!

Fallunterscheidung mit if

src/entscheidung.c

```
3 int main()  
4 {  
5     if (1 == 0)  
6     {  
7         printf("Die Welt geht unter!\n");  
8     }  
9     else  
10    {  
11        printf("Alles in Ordnung...\n");  
12    }  
13 }
```

- ▶ Ausdruck in der Klammer wahr: Erster Abschnitt wird ausgeführt
- ▶ Hinter dem else kann auch noch ein if (Bedingung) stehen
- ▶ Trifft keine der vorigen Bedingungen zu, wird der letzte Abschnitt (Else-Zweig) ausgeführt
- ▶ Achtung! == ist Vergleich, = wäre Zuweisung

Fallunterscheidung mit if

src/entscheidung2.c

```
3 int main()  
4 {  
5     int i=1;  
6     if (i)  
7     {  
8         printf("Diese Bedingung ist wahr.\n");  
9     }  
10 }
```

- ▶ Es gibt keinen Boolean-Datentyp und auch keine Wahrheitswerte *true* oder *false*
- ▶ int wird als Ersatz benutzt: 0 ist *false*, alles andere ist *true*

Fallunterscheidung mit ? :

Eine andere Möglichkeit der Fallunterscheidung: Der tertiäre ?:-Operator
condition ? value if true : value if false

src/entscheidung3.c

```
int alter = 25;  
//Wie viele Mietzekatzen??  
int katzen = ( alter > 50 ) ? 5 : 2;
```

4!

Fallunterscheidung mit ? :

Eine andere Möglichkeit der Fallunterscheidung: Der tertiäre ?:-Operator
condition ? value if true : value if false

src/entscheidung3.c

```
int alter = 25;  
//Wie viele Mietzekatzen??  
int katzen = ( alter > 50 ) ? 5 : 2;
```



Switch Case

src/entscheidung4.c

```
int jahr = 2014;
switch(jahr)
{
    case 0: start_counting();
        break;
    case 1969: printf("Erster Mensch auf dem Mond\n");
        break;
    default: printf("Keine Ahnung, was los ist\n");
}
```

Achtung!

Das *break* nicht vergessen, sonst werden die folgenden Fälle auch durchlaufen!

Schleifen

- ▶ Wie in Java gibt es zwei Arten von Schleifen:
- ▶ **for**-Schleife für das Durchlaufen eines bekannten Intervalls
- ▶ **while**-Schleife für eine definierte Abbruchbedingung



Schleifen: for

src/schleife1.c

```
int berechne_summe(int obergrenze) {  
    int i;  
    int summe;  
    for (i = 0 ; i < obergrenze; i++)  
    {  
        summe += i;  
    }  
    return summe;  
}
```

Achtung!

Zählvariable im Schleifenkopf definieren ist nur im C99-Standard erlaubt. Summenvariable muss außerhalb der Schleife deklariert werden, damit sie nach der Berechnung noch verfügbar ist

Schleifen: while

src/schleife2.c

```
int test = 0;
while (test) {
    printf("Es geht! \n");
}
```

src/schleife3.c

```
int test = 234205;
while (test) {
    printf("Es geht! \n");
}
```

Eine dieser Schleifen ist eine Endlosschleife... Welche?

Schleifen: while

src/schleife2.c

```
int test = 0;
while (test) {
    printf("Es geht! \n");
}
```

src/schleife3.c

```
int test = 234205;
while (test) {
    printf("Es geht! \n");
}
```

Eine dieser Schleifen ist eine Endlosschleife... Welche?

Die untere, da alles, was nicht 0 ist, true ist

- 1 Hello, C-Kurs
- 2 Funktionen
- 3 Datentypen und Operatoren
- 4 Programmfluss
- 5 Hilfestellungen**
- 6 Häufige Fehler
- 7 Zusammenfassung



4!

Kommandozeile effizienter nutzen

- Kompilieren und ausführen in einem Schritt:

```
$ gcc helloworld.c -o HelloWorld && ./HelloWorld
Hello World!
$
```

Verknüpfen zweier Kommandozeilenbefehle

- Befehl noch einmal ausführen: Pfeiltaste nach oben (ggf mehrfach, oder Strg+r und Suchwort eingeben)
- Dateinamen vervollständigen: Die ersten paar Buchstaben eingeben, dann Tab drücken
- wenn nichts passiert, nochmal Tab drücken: Dateinamen, die passen, werden angezeigt

```
$ gcc hello<Tab><Tab>
hellomars.c    helloworld.c
$ gcc hellowo<Tab>
```

Namen vervollständigen

- Abbruch: Strg+c

- ▶ Code kommentieren, damit er auch von Menschen gelesen werden kann!
- ▶ Mehrzeilig: `/* Beliebig langer Textabschnitt */`
 - ▷ Von jedem C-Compiler akzeptiert
 - ▷ Kopf der Datei mit AutorIn, Beschreibung des Programms, Lizenz
 - ▷ Kopf einer Funktion mit Erklärung der Parameter u.ä.
 - ▷ Auskommentieren mehrerer Zeilen, um Fehler Stück für Stück auszumerzen
- ▶ Kommentare bis Zeilenende: `// Bemerkung zur aktuellen Zeile`
 - ▷ „Strenge“ Compiler könnten den Kommentar nicht akzeptieren
 - ▷ verwendet zum Erklären einer Zeile, z.B. wenn sie sonst nach einiger Zeit nicht mehr auf Anhieb verständlich wäre

- ▶ Der Kommandozeilenbefehl *man* kann nicht nur Hilfe zu Befehlen anzeigen, sondern auch zu C-Funktionen und Systemrufen (lernt ihr in TechGl3)
 - ▷ Sektion 2: System Calls
 - ▷ Sektion 3: C Library Functions
- ▶ Mit *-s* könnt ihr der *man*-Funktion die gewünschte Sektion übergeben
- ▶ besonders nützlich: EXAMPLES

```
$ man -s3 printf
```

Aufruf des *man*-Befehls

PRINTF(3)

Linux Programmer's Manual

PRINTF(3)

NAME

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

SYNOPSIS

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

```
int printf(const char *format, ...);  
[...]
```

DESCRIPTION

The functions in the printf() family produce output according to a format as described below. [...]

Eingebaute Hilfe: Manuals

- Es gibt sogar Manualseiten ganzer C-Headerdateien

```
$ man stdio.h
```

- Die müsst ihr allerdings erst mal installieren
- Unter Debian/Ubuntu beispielsweise:

```
$ sudo apt-get install manpages-posix manpages-posix-dev
```

Manualseiten nachinstallieren

Versionen des C-Standards

- ▶ Einige Dinge, die in älteren Standards nicht gehen, sind in neueren erlaubt
- ▶ Der benutzte Standard kann mit `gcc -std=version` (klein geschrieben) beim Kompilieren eingestellt werden
- ▶ Nur eine Übersicht, ihr müsst noch nicht alles darin kennen:

Version	Änderungen gegenüber vorigen Versionen
K&R-C	Ursprungsversion, weniger Standardbibliotheken, heutzutage kaum noch verwendet
C90	Funktionsprototypen, Präprozessor, Bibliotheken normiert
C95	Standardmakros, insbesondere ' <code>__STDC_VERSION__</code> ' zum Auslesen der C-Version
C99	Komplexe Zahlen, Variablendeklaration im Schleifenkopf <code>_Bool</code> , <i>restricted</i>

- 1 Hello, C-Kurs
- 2 Funktionen
- 3 Datentypen und Operatoren
- 4 Programmfluss
- 5 Hilfestellungen
- 6 Häufige Fehler**
- 7 Zusammenfassung



4!

Häufige Fehler

src/fehler1.c

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Hello World")
6     return 0;
7 }
```

```
$ gcc fehler1.c -o fehler1
fehler.c: In function 'main':
fehler1.c:6:2: error: expected ';' before 'return'
```

Häufige Fehler

src/fehler1.c

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf(" Hello World")
6     return 0;
7 }
```

```
$ gcc fehler1.c -o fehler1
fehler.c: In function 'main':
fehler1.c:6:2: error: expected ';' before 'return'
```

Problem:

In Zeile 5 fehlt das Semikolon am Ende der Zeile.

Häufige Fehler

src/fehler2.c

```
1 int main() /* A hello world program */  
2 {  
3     printf("Hello World");  
4     return 0;  
5 }
```

```
$ gcc fehler2.c -o fehler2  
src/fehler2.c: In function 'main':  
src/fehler2.c:3:2: warning: incompatible implicit  
      declaration of built-in function 'printf' [enabled by  
      default]
```

Häufige Fehler

src/fehler2.c

```
1 int main() /* A hello world program */  
2 {  
3     printf("Hello World");  
4     return 0;  
5 }
```

```
$ gcc fehler2.c -o fehler2  
src/fehler2.c: In function 'main':  
src/fehler2.c:3:2: warning: incompatible implicit  
      declaration of built-in function 'printf' [enabled by  
      default]
```

Problem:

Das `#include <stdio.h>` wurde vergessen.

Häufige Fehler

src/fehler3.c

```
1 #include <stdio.h>
2
3 void anzahl(unsigned int personenzahl) {
4     if ( personenzahl = 0 ) {
5         printf("Niemand zu Hause!\n");
6     }
7     else {
8         printf("Komm vorbei!\n");
9     }
10 }
```

```
$ gcc fehler3.c -o fehler3
```


Häufige Fehler

src/fehler3.c

```
1 #include <stdio.h>
2
3 void anzahl(unsigned int personenzahl) {
4     if ( personenzahl = 0 ) {
5         printf("Niemand zu Hause!\n");
6     }
7     else {
8         printf("Komm vorbei!\n");
9     }
10 }
```

```
$ gcc fehler3.c -o fehler3
```

Problem:

In der Bedingung hinter if wird nicht verglichen, sondern zugewiesen!
Somit kommt immer das selbe heraus.

- 1 Hello, C-Kurs
- 2 Funktionen
- 3 Datentypen und Operatoren
- 4 Programmfluss
- 5 Hilfestellungen
- 6 Häufige Fehler
- 7 Zusammenfassung**



4!

Was haben wir gelernt?

- ▶ Quellcode schreiben, mit `gcc quellcode.c -o programmname` kompilieren und mit `./programmname` ausführen
- ▶ Funktionen und Variablen erst deklarieren, dann definieren/benutzen
- ▶ In C gibt es viele Operatoren, um mit einzelnen Bits zu hantieren
- ▶ Programmfluss steuern mit *if*, *for*, *while* und *switch()* *case*
- ▶ Der Compiler gibt Warnungen und Fehlermeldungen aus, die helfen können, Fehler zu finden - aber nicht immer.

Ausblick

Im Tutorium heute nachmittag wird Ein- und Ausgabe (z.B. von Variablen) behandelt. Es wird vier Tutorien bei vier verschiedenen Leuten geben, auf jeder Etage zwei.

Ein kleiner Vorgeschmack:

src/io-beispiel.c

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int antwort = 42;
6     printf("Die Antwort ist %d!\n", antwort);
7     return 0;
8 }
```

```
$ gcc src/io-beispiel.c -o io-beispiel && ./io-beispiel
Die Antwort ist 42!
```

Und jetzt?

- ▶ Feedbackzettel ausfüllen und mitnehmen
(Braucht ihr nachher noch fürs Tutorium)

Üben!

- ▶ Übungsaufgaben:
<http://wiki.freitagrunde.org/Ckurs/Übungsaufgaben>
- ▶ TEL 106 / 206 (Hochhaus am Ernst-Reuter-Platz)
- ▶ Dort laufen TutorInnen rum, die euch helfen.
- ▶ Mittagspause von 13:00 Uhr bis 14:15 Uhr
- ▶ Nicht vergessen: 14:15 Uhr Tutorium (im TEL!) zum Thema *Ein- und Ausgabe (printf, scanf, fopen)*
- ▶ Morgen früh um 10:15 Uhr: Tutorium (im TEL!) zum Thema *Datenstrukturen*