

Einführung in C

WS 2014/2015

Literatur:

- B. Kernighan, D. Ritchie: *The C-programming language*, Second Edition, Prentice-Hall, 1988.
- R. Sedgewick: *Algorithms in C*, Third edition, Addison-Wesley, 1998.
- Unix man-Pages
- Internet: www.cppreference.com oder <http://www.cplusplus.com>

Dank an Ralf Wienzek, RWTH Aachen, 1. Version WS 2006/2007

- Historie
- Datentypen
- Operatoren
- Ein-/Ausgabe
- Kontrollstrukturen
- Funktionen
- Ausblick
- Beispiele

- Weiterentwicklung von BCPL und B
 - BCPL: Ende der 60er Jahre von Martin Richards zum Bau von Betriebssystemen und Compilern entwickelt
 - B: Ken Thompson erstellte 1970 mit B das erste UNIX System
- C
 - 1972 von Dennis Ritchie in den Bell Laboratories entwickelt
 - Wurde zur Entwicklung des UNIX-Betriebssystems verwendet
 - Zunächst durch den Klassiker „The C Programming Language“ von Brian Kernighan und Dennis Ritchie beschrieben und 1989 vom amerikanischen ANSI-Institut standardisiert
 - Häufig nicht als Hochsprache angesehen, da maschinennahe Programmierung möglich
 - Hohe Flexibilität, kleiner Sprachumfang (ANSI-C hat nur 32 Schlüsselwörter)
 - keine Schutzmechanismen, kein strenges Typkonzept
 - Programmiersprache für Programmierer

Ein einfaches Programm

```
#include <stdio.h>                                // Funktionsdeklarationen zur Ein-/Ausgabe einbinden

/*****
* Hauptprogramm
*****/

int main() { // Hier beginnt die Programmausführung
    printf( "Hello World.\n" );           // Ausgabe von "Hello World." auf der Standardausgabe
    return 0;                             // Rückgabewert des Programms
}
```

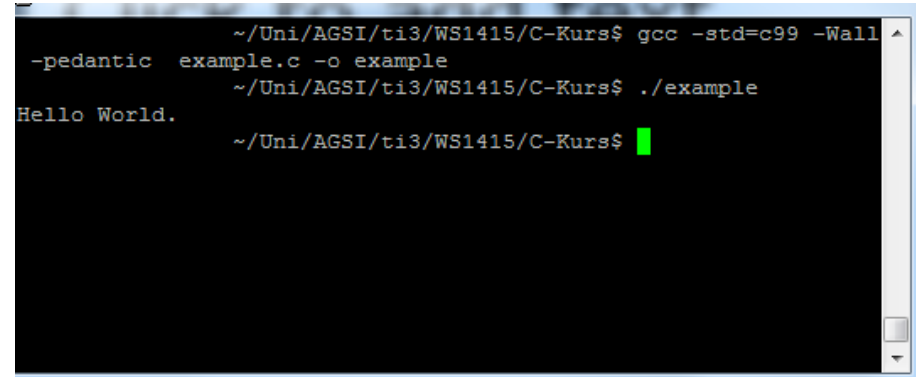
1. Kompilieren mit dem GCC

Mit den Flags:

- `-std=c99` -> Standard festlegen
- `-Wall` -> alle Warnungen anzeigen
- `-pedantic` -> besondere Vorsicht walten lassen
- `-o outputname` -> Kompilatdatei wird angegeben

• 2. Programm ausführen

Mit `./outputname`



```
~/Uni/AGSI/ti3/WS1415/C-Kurs$ gcc -std=c99 -Wall -pedantic example.c -o example
~/Uni/AGSI/ti3/WS1415/C-Kurs$ ./example
Hello World.
~/Uni/AGSI/ti3/WS1415/C-Kurs$
```

- Elementare Typen

Typ	<i>minimaler</i> Wertebereich	Beispiele
char	-127 ... 127	'a', '&', ' ', '\n', '\0'
unsigned char	0 ... 255	
int	-32767 ... 32767	Dezimal: 0, -7, 42 Oktal: -07, 010 Hexadez.: 0x2f, 0xa
unsigned int	0 ... 65535	
(unsigned) short int	wie int	
long int	-2147483647 ... 2147483647	
unsigned long int	0 ... 4294967295	0.0, -1.7, 31415e-4
float	auf 6 Stellen genau	
double	auf 10 Stellen genau	
long double	auf 10 Stellen genau	

- Variablendefinition

```
char c = '7';           // Vorzeichenbehaftete Character-Variable namens c mit Initialisierung
unsigned long int ul;    // Vorzeichenlose 64 Bit Variable namens ul
```

- **Arithmetische Operatoren**

+	Addition
-	Subtraktion, Negation
*	Multiplikation
/	Division
%	Divisionsrest (Modulo)

- **Bit-Operatoren**

&	Bitweises UND
	Bitweises ODER
^	Bitweises XOR
~	Invertieren

- **Shift-Operatoren**

<<	Links-Shift
>>	Rechts-Shift

- **Zeiger-Operatoren**

&	Adressoperator
*	Dereferenzierung

- **Vergleichsoperatoren**

==	Gleich
!=	Ungleich
<	Kleiner
<=	Kleiner oder gleich
>	Größer
>=	Größer oder gleich

- **Logische Verknüpfungen**

&&	Logisches UND
	Logisches ODER
!	Negation

C kennt keinen eigenen Datentyp für Boolesche Variablen. FALSE wird durch 0 und TRUE durch einen Wert ungleich 0 repräsentiert.

- Implizite Konvertierung
 - Enthält ein Ausdruck Variablen oder Konstanten verschiedener Datentypen, wird der Typ einzelner Operanden automatisch umgewandelt. Dabei wird der Operand mit kleinerem Wertebereich an den größeren Datentyp angepasst.
- Explizite Konvertierung
 - Der Programmierer gibt durch Voranstellen des Datentyps an, in welchen Typ die nachfolgende Variable oder Konstante umgewandelt werden soll.
- Beispiele
 - `float f = 1.0/3`
ergibt 0.333333: erster Operand ist Gleitkommazahl, d.h. implizite Konvertierung von 3 zu 3.0 und Durchführung einer Fließkommadivision
 - `float f = 1/3`
ergibt 0.0: ganzzahlige Division von 1 durch 3 ergibt 0, implizite Konvertierung von 0 zu 0.0
 - `float f = (int)0.333333 * 3`
ergibt 0.0: explizite Konvertierung von 0.333333 zu 0, anschließende Multiplikation mit 3 und impliziter Konvertierung zu 0.0
 - `float f = (float)1/3`
ergibt 0.333333: explizite Konvertierung von 1 zu 1.0, implizite Konvertierung von 3 zu 3.0 und Durchführung einer Fließkommadivision
 - `float f = (float)(1/3)`
ergibt 0.0: ganzzahlige Division von 1 durch 3 ergibt 0, explizite Konvertierung von 0 zu 0.0

printf(Kontrollstring, arg1, arg2, ...)

- Flexible Ausgabefunktion
- Die Funktion wertet den Kontrollstring aus und formatiert die Argumente entsprechend
- Einfacher Text im Kontrollstring wird unverändert ausgegeben
- Platzhalter im Kontrollstring werden durch die Werte der weiteren Argumente ersetzt:

%d Dezimalzahl

%x Hexadezimalzahl

%u Vorzeichenlose Zahl

%c ein Zeichen

%s Zeichenkette

%f Gleitkommazahl

- **Beispiel**

```
int i=42;
```

```
printf("%d als Hex.-Zahl: %x\n", i, i);
```

scanf(Kontrollstring, arg1, arg2, ...)

- Gegenstück zu printf
- Liest Zeichen von der Standard-Eingabe, interpretiert sie gemäß des Kontrollstrings und speichert die Wert in den Argumenten
- **Achtung:** Argumente müssen Zeiger auf die Speicherbereiche sein, in denen die Werte abgelegt werden sollen.
- Kontrollstring enthält:
 - Whitespaces, die als Trennzeichen dienen und ignoriert werden
 - Gewöhnliche Zeichen, die als nächstes Zeichen in der Eingabe vorkommen müssen
 - Formatelemente, die mit "%" beginnen

- **Beispiel**

```
int i;
```

```
float x;
```

```
char str[80];
```

```
scanf("%d %f %s", &i, &x, str);
```

- if - Anweisungen

```
if (Bedingung)
    Anweisung;
[else
    Anweisung;]
```

- Beispiele

```
if (a > b)
    max = a;
else
    max = b;
```

```
if (arg[0]=='-')
    if (arg[1]=='v') {
        version=1;
        printf("Version 1\n");
    } else
        printf("Unknown option.\n");
```

- Verschachtelte if-Anweisungen

```
if (Bedingung1)
    Anweisung1;
else if (Bedingung2)
    Anweisung2;
else if (Bedingung3)
    Anweisung3;
else
    Anweisung4;
```

Bedingungen 1-3 werden in dieser Reihenfolge geprüft. Sobald eine Bedingung erfüllt ist, wird die entsprechende Anweisung ausgeführt. Ist keine Bedingung erfüllt, wird Anweisung4 ausgeführt.

- while-Schleife

```
while (Bedingung)
    Anweisung;
```

Solange die Auswertung der Bedingung einen Wert ungleich 0 liefert, wird der Anweisungsblock durchlaufen.

- do-while-Schleife

```
do
    Anweisung;
while (Bedingung);
```

Die Anweisung wird immer mindestens ein mal ausgeführt. Liefert beim Erreichen des while-Statements die Auswertung der Bedingung einen Wert ungleich 0, wird die Anweisung erneut ausgeführt.

- for-Schleife

```
for (AnwInit; Bedingung; AnwSchleife)
    Anweisung;
```

Beim Erreichen des for-Statements wird zunächst AnwInit ausgeführt. Ist danach die Schleifenbedingung erfüllt, wird der Schleifenkörper durchlaufen. Nach jedem Schleifendurchlauf wird AnwSchleife ausgeführt und falls die Schleifenbedingung noch erfüllt ist, ein weiterer Durchlauf gestartet.

- Beispiel

```
int i, sum=0;
```

```
for (i=0; i<=10; i=i+1)
    sum = sum+i;
```

```
for (i=10; i; i=i-1)
    sum = sum+i;
```

```
for (;;)
    puts(„Endlos-Schleife“);
```

- **break**

- Anwendbar bei **switch**, **for**-, **while**- und **do-while**-Schleifen
- Verhindert die Ausführung weiterer Anweisungen innerhalb der Schleife
- Führt zum sofortigen Verlassen von for-, while und while-do-Schleifen
- Kann die Lesbarkeit von Programmen erhöhen, da nicht sämtliche Bedingungen in den Schleifenkopf untergebracht werden müssen

- **Beispiel**

```
while (1) {                // Endlosschleife
    ...
    if ( input==0 )
        break;            // Beenden der Schleife
    ...
}
```

- **continue**

- Anwendbar bei **for**-, **while**-, **do-while**-Schleifen
- Bewirkt eine sofortige Rückkehr zur Schleifenanweisung

- **Beispiel**

```
for (i=0; i<7; i++) {
    if ((i==2) || (i==4))
        continue;
    printf( "%i " );
}
```

Ergibt:

0 1 3 5 6

Funktionen – Definition

- Definition

```
Rückgabetyt Funktionsname( Parameterliste ) {
    Anweisungen
```

```
    return (Rückgabewert)    // führt zum sofortigen Beenden der Funktion und der
    Rückgabewert wird als Ergebnis an die aufrufende Funktion zurückgegeben.
}
```

- Beispiel

```
int max;
int maxi( int a, int b ) {
    int max=a;           // lokale Variable, die nur innerhalb dieser Funktion gültig ist
    if (a<b)
        max=b;
    return max;    // Verlassen der Funktion und Rückgabe von max
}
```

```
void printHallo( void ) {           // Funktion ohne Parameter und Rückgabewert
    printf( "Hallo.\n" );
}
```

```
max = maxi( 1,2 );    // Aufruf von maxi mit Parametern 1 und 2
printHallo();         // Aufruf von printHallo
```

Um in C Funktionen anderer Module verwenden zu können, müssen diese Funktionen zunächst *deklariert* werden. Hierdurch wird dem Compiler mitgeteilt, welche Parameter eine Funktion erwartet und welchen Rückgabewert sie besitzt.

Beispiel:

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

Der dazugehörige Programmcode ist in einer Objektdatenbank abgelegt, die zum Schluss zum Programm "hinzugelinkt" wird (siehe Compiler-Handbuch).

Header Dateien

Für ein Modul werden die Funktionsprototypen in der Regel in Header-Dateien (.h) zusammengefasst, die wie folgt eingebunden werden können.

```
#include "dateiname.h" bzw. #include <dateiname.h>
```

Beispiele häufig verwendeter Header Dateien

stdio.h:	Funktionen zur Ein-/Ausgabe (printf, scanf, fopen, fclose, ...)
string.h:	Funktionen zur Manipulation von Zeichenketten (strlen, strcpy, ...)
math.h:	Mathematische Funktionen (sin, cos, sqrt, ...)

- Die Einführung ist unvollständig!
- Weitere Themen werden in den Tutorien stetig besprochen.
- Weitere Themen:
 - Structs
 - Beispiel

```
struct punkt {  
    int x;  
    int y;  
};
```
 - Arrays
 - Zeiger
 - Weitere Datentypen
 - Dynamische Speicherverwaltung
 - Und vieles mehr....

```
#include <stdio.h>

typedef char* STRING;

void func1( int var1, STRING var2 );

int main (char argc, char *argv[] )
{
    int i;
    printf( "Dateiname: %s\n", argv[0] );
    for (i=1; i<argc; i++)
        func1( i, argv[i] );
    return 0;
}

void func1( int var1, STRING var2 )
{
    printf( "%i. Parameter: %s\n", var1, var2 );
}
```

```
$ gcc -o bsp bsp.c
$ bsp
Dateiname: bsp
$ bsp -23 zwei
Dateiname: bsp
1. Parameter: -23
2. Parameter: zwei
$ _
```


Ein einfaches Programm 2

```
#include <stdio.h>                                // Funktionsdeklarationen zur Ein-/Ausgabe einbinden

int maxi( int a, int b );                          // Funktionsdeklaration

int global_max = 0;                                // Globale Variable global_max (mit 0 initialisiert)

/*****
* Hauptprogramm
*****/

int main( int argc, char *argv[] ) {               // Hier beginnt die Programmausführung
    printf( "Hello World.\n" );                    // Ausgabe von "Hello World." auf der Standardausgabe
    global_max = maxi( 1,2 );                      // Aufruf der Funktion maxi
    return 0;                                       // Rückgabewert des Programms
}

/*****
* Funktion zur Maximumberechnung
*****/

int maxi( int a, int b ) {                          // Funktionsdefinition
    if (a>b)
        return a;
    else
        return b;
}
```