Metalab Kurs
μC-Programmierung in C

Clifford Wolf
Stefan Farthofer

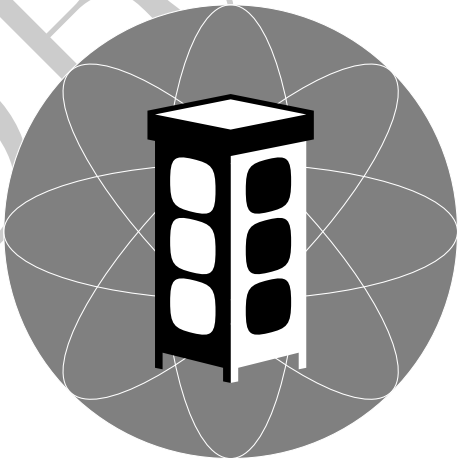
im Oktober 2011

Skriptum zum Metalab Kurs „μC-Programmierung in C“. Diese Unterlagen sind begleitend zum Kurs gedacht. Die Kursinhalte werden in diesem Skriptum kompakt zusammengefasst.

`http://metalab.at/wiki/uCProg_Kurs`

Dieses Skriptum ist vollständig in LaTeX gesetzt. Die Zeichnungen, Schaltpläne und Diagramme wurden mittels PGF und TikZ direkt in TeX erstellt.

DRAFT

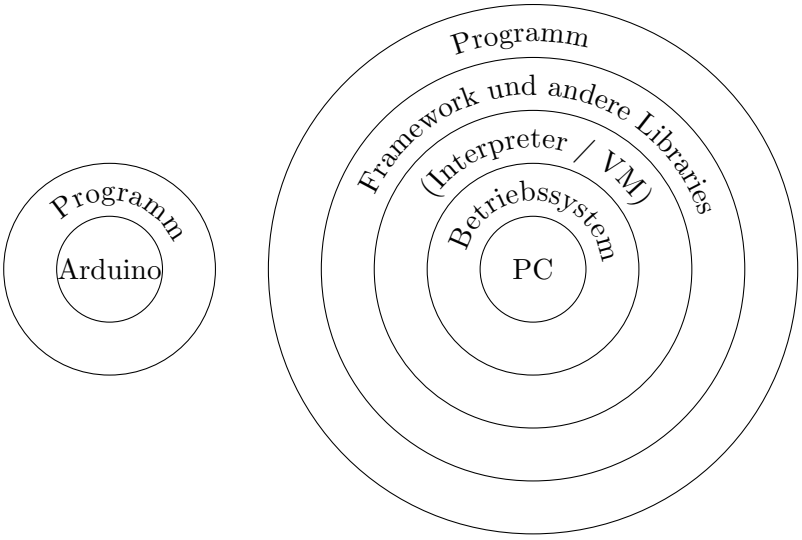


Dieses Werk ist unter der Creative Commons BY-NC-SA-Lizenz lizenziert.
`http://creativecommons.org/licenses/by-nc-sa/3.0/at/`

Block 0: Die Arduino Experimentierumgebung

Der Arduino ist ein kleiner Computer. Mit der Arduino-Software kann der Arduino in C programmiert werden. Diese Programme können über die USB-Schnittstelle auf den Arduino übertragen werden.

Im Gegensatz zum PC hat man beim Arduino direkten Zugriff auf die Hardware.



Das verringert die Komplexität des Gesammtsystems. Somit ist weniger Hintergrundwissen notwendig um Programme zu schreiben.

Genausowenig wie ein PC ohne Maus, Tastatus und Bildschirm sinnvoll ist, so ist ein Arduino ohne die sogenannten *Shields*, die den Arduino um Peripherieelemente erweitert, sinnvoll.

Wir verwenden für unsere Übungen unser eigenes „Ampel-Shield“. Üblicherweise entwickelt man für seine Projekte eigene Shields. Es gibt aber eine grosse Menge verschiedenste fertige Shields in diversen Online Shops zu kaufen.

Übung:

Installieren Sie die Arduino Software und spielen Sie das `Digital/Blink` Example in den Arduino ein.

Diskussion:

Welche Eingabe-/Ausgabe-Schnittstellen hat ein Arduino? Recherchieren Sie gegebenenfalls im Internet.

Übung:

Installieren Sie die `Console` Library die wir für diesen Kurs verwenden. Folgen Sie dazu den Anweisungen im `README`-File.

Block 1: Ein Arduino Hello-World Programm

Wir schreiben unser erstes Arduino Programm.

Ein Programm besteht aus beliebig vielen Funktionen die wir selber schreiben müssen. Darüber hinaus existiert eine Sammlung vorgefertigter Funktionen die wie verwenden können (so eine Sammlung heißt *Library*).

```
1  void setup()
2  {
3      consoleInit(9600);
4      consolePrint("Hello World!\n");
5  }
6
7  void loop()
8  {
9      // Hier passiert nichts.
10 }
```

block01_hallo.pde

Jedes Arduino-Programm muss die Funktionen `setup` und `loop` definieren. Die `setup` Funktion wird einmal beim Programmstart ausgeführt. Die `loop` Funktion danach in einer Endlosschleife.

Eine Funktion kann andere Funktionen hintereinander aufrufen.

Funktionen können Werte übergeben bekommen und einen Wert zurückliefern. Das Schlüsselwort `void` im Funktionskopf der Funktionen `setup` und `loop` signalisiert, dass diese Funktionen keine Werte zurückliefern.

Die Funktion `consoleInit` initialisiert die serielle Schnittstelle auf eine Baudrate von 9600 Baud. Die Funktion `consolePrint` gibt einen Text aus. Texte

werden in C in doppelte Anführungszeichen geschrieben, wobei „`\n`“ hier für einen Zeilenumbruch steht.

Der Text nach einem „//“ (bis zum Zeilenende) ist ein Kommentar und wird von der Entwicklungsumgebung ignoriert.

Übung:

Führen Sie das Programm am Arduino aus und sehen Sie sich die Ausgabe auf der seriellen Konsole der Arduino-Software an.

Was passiert wenn Sie den Reset-Knopf am Arduino drücken?

Diskussion:

Wie ist die exakte Syntax für Funktionsaufrufe und Funktionsdefinitionen? Welchen Zweck haben die Strichpunkte im Programmcode?

Diskussion und Experiment:

Was passiert wenn Sie den `consolePrint` Aufruf in die `loop` Funktion verschieben?

Was passiert wenn Sie statt `consolePrint` den Funktionsnamen `consoleprint` verwenden?

Block 2: Rechnen mit dem Arduino

Eine Variable ist ein Name für eine Speicherstelle. Eine Variable wird durch Angabe des Datentypes (hier `int`) gefolgt vom Variablennamen definiert. Der Datentyp `int` kann Ganzzahlen im Wertebereich $-32768 \ldots 32767$ speichern.

Mit dem `=`-Operator (Zuweisungsoperator) kann ein Wert in einer solchen Speicherstelle gespeichert werden.

```
1 int summe;
2 int eingabe;
3
```

```
4  void setup()
5  {
6      consoleInit(9600);
7      summe = 0;
8  }
9
10 void loop()
11 {
12     eingabe = consoleReadDecimal("Geben Sie eine Zahl ein: ");
13     summe = summe + eingabe;
14
15     consolePrintf("Die Summe ist derzeit: %d\n", summe);
16 }
```

block02_summe.pde

Die Funktion `consoleReadDecimal` gibt den als Parameter übergebenen Prompt aus und liest eine Ganzzahl ein, die von der Funktion als Rückgabewert zurückgeliefert wird.

Die Funktion `consolePrintf` gibt wie `consolePrint` ihren ersten Parameter auf der Konsole aus, wobei dieser erste Parameter Platzhalter wie „`%d`“ beinhalten kann, an deren Stelle Textrepresentationen der folgenden Parameter gesetzt werden.

Variablen die ausserhalb von Funktionen definiert werden behalten ihren Wert über die gesamte Lebensdauer des Programms.

Diskussion:

Umgangssprachlich wird der Begriff der Variable und der Begriff der Speicherstelle austauschbar verwendet.

Wie verwaltet der Computer (Arduino) intern die Speicherstellen? Muss man als Programmierer auch auf dieser hardwarenäheren Ebene mit dem Speicher arbeiten?

Diskussion:

Wie werden im Computer (Arduino) Zahlenwerte intern gespeichert?

Übung:

Was passiert wenn man dem Programm 4× mal die Eingabe „10000“ übergibt? Erklären sie Ihre Beobachtungen.

Block 3: Arduino IO-Pins

Die meisten Pins des Arduino (Atmega μC) sind frei als Eingabe- oder Ausgabepins verwendbar. Wenn der Pin als Eingabe verwendet wird kann optional ein interner Pullup-Widerstand zugeschaltet werden.

```
1  int pinValue;
2
3  void setup()
4  {
5      pinMode(2, INPUT);
6      pinMode(13, OUTPUT);
7      digitalWrite(2, 1); // enable pullup resistor
8  }
9
10 void loop()
11 {
12     pinValue = digitalRead(2);
13     digitalWrite(13, pinValue);
14 }
```

block03_pinio.pde

Die `pinMode` Funktion konfiguriert einen Pin als Ein- oder Ausgabepin. In diesem Beispiel wird Pin Nummer 2 als Eingabepin und der Pin Nummer 13 als Ausgabepin verwendet. Der Pin Nummer 13 ist am Arduino auch direkt mit der LED „`L`“ verbunden.

Für Eingabepins aktiviert/deaktiviert `digitalWrite` den Pullup-Widerstand und für Ausgabepins setzt `digitalWrite` den Wert.

Bei einem Eingabepin kann mit der Funktion `digitalRead` der aktuelle Wert ausgelesen werden.

Die Wörter `INPUT` und `OUTPUT` werden in der Arduino-Library als Synonyme für die Zahlenwerte 0 und 1 definiert.

Diskussion und Übung:

Was macht das Beispielprogramm? Welche externe Beschaltung ist notwendig?

Probieren Sie das Programm aus.

Diskussion:

Wesshalb definiert man Synonyme (wie `INPUT` und `OUTPUT`) für Zahlenwerte? Was sind die Vor- und Nachteile?

Diskussion:

In welchen Fällen braucht man einen Pullup-Widerstand? Wann kann ein Pullup-Widerstand stören?

Block 4: Definieren weiterer Funktionen

Wir schreiben eine eigene Funktion mit Parameter:

```
1  void blinkPin(int pin)
2  {
3      digitalWrite(pin, 1);
4      delay(1000);
5      digitalWrite(pin, 0);
6      delay(100);
7  }
8
9  void setup()
10 {
11     pinMode(13, OUTPUT);
12     pinMode(12, OUTPUT);
13 }
14
15 void loop()
16 {
17     blinkPin(12);
18     blinkPin(12);
19     blinkPin(13);
20 }
```

block04_blink.pde

Wie die Funktionen `setup` und `loop` liefert unsere `blinkPin`-Funktion keinen Rückgabewert, was durch das Schlüsselwort `void` signalisiert wird. Im Gegensatz zu `setup` und `loop` akzeptiert `blinkPin` einen Parameter `pin` vom Datentyp `int`. Dazu wurde der Parameter in die runden Klammern gesetzt, die bei parameterlosen Funktionen wie `setup` und `loop` leer bleiben. Bei Funktionen mit mehreren Parametern werden diese durch Kommata getrennt:

```
1  void blinkPin(int pin, int lengthMs)
2  {
3      digitalWrite(pin, 1);
4      delay(lengthMs);
5      digitalWrite(pin, 0);
6      delay(100);
7  }
8
9  void setup()
10 {
11     pinMode(13, OUTPUT);
12     pinMode(12, OUTPUT);
13 }
14
15 void loop()
16 {
17     blinkPin(12, 1000);
18     blinkPin(12, 2000);
19     blinkPin(13, 3000);
20 }
```

block04_blink2.pde

Die Funktion `delay` wartet übrigens die angegebene Anzahl von Millisekunden.

Diskussion:

Wie ist die vollständige Syntax für eine Funktionsdefinition mit Parametern?

Wozu sind die Parameter überhaupt gut? Welche andere Möglichkeit gibt es Daten zwischen aufrufender und aufgerufener Funktion auszutauschen?

Übung:

Schreiben Sie eine funktion `sumDiff` mit zwei `int`-Parametern die die Summer und die Differenz der beiden Parameter auf der seriellen Konsole aus-

gibt.

Diskussion:

Für die beiden `blink`-Programme werden zwei LEDs benötigt. Wie muss man diese mit dem Arduino verschalten damit das Programm funktioniert?

Block 5: Rückgabewerte

Wir schreiben eine Funktion die einen Wert zurückliefert:

```
1  int getFlashDurationMs()
2  {
3      return consoleReadDecimal("Wieviele Sekunden? ") * 1000;
4  }
5
6  void blinkPin(int pin, int lengthMs)
7  {
8      digitalWrite(pin, 1);
9      delay(lengthMs);
10     digitalWrite(pin, 0);
11 }
12
13 void setup()
14 {
15     consoleInit(9600);
16     pinMode(13, OUTPUT);
17 }
18
19 void loop()
20 {
21     blinkPin(13, getFlashDurationMs());
22 }
```

block05_flash.pde

Wenn eine Funktion einen Rückgabewert hat (kein `void`), dann muss das letzte Statement der Funktion ein `return`-Statement sein, dass den Wert zurückliefert.

Im Übrigen: der Stern ist der Operator zum Multiplizieren und wie man in Zeile 21 sehen kann ist es auch möglich Funktionsaufrufe ineinander zu verschachteln.

Experiment:

Was passiert wenn man nach dem `return`-Statement weiteren Code (zum Beispiel `consoleWrite("Hallo Welt!\n");` einfügt?

Diskussion:

Wie ist die Syntax des `return`-Statements?

Wozu sind Rückgabewerte überhaupt gut? Welche andere Möglichkeit gibt es Daten zwischen aufgerufener und aufrufender Funktion auszutauschen?

Block 6: Globale und lokale Variablen

Variablen die ausserhalb von Funktionen definiert werden heissen „globale Variablen“. Sie behalten ihren Wert über die gesammte Lebensdauer des Programmes.

Variablen die innerhalb von Funktionen definiert werden heissen „lokale Variablen“. Sie behalten ihren Wert lediglich über die Laufzeit der Funktion. Mit Hilfe von lokalen Variablen kann die Funktion aus Block 5 wie folgt umgeschrieben werden:

```
1 int getFlashDurationMs()
2 {
3     int durationSeconds;
4     durationSeconds = consoleReadDecimal("Wieviele Sekunden? ");
5     return durationSeconds * 1000;
6 }
```

block06_durationms.pde

Da hier der Zwischenwert einen Namen hat ist diese Version der Funktion lesbarer.

Da lokale Variablen ihren Wert nur über die Laufzeit der Funktion behalten braucht die lokale Variable (im Gegensatz zur globalen Variable) nicht permanent Speicher. Diese Variante der Funktion ist sogar genauso effizient wie die vorhergehende, weil der Compiler in der vorhergehenden Version der Funktion automatisch eine lokale Variable ohne Namen für den Zwischenwert anlegen musste.

Diskussion:

Was sind die Parallelen und Unterschiede zwischen lokalen Variablen und Parametern?

Was passiert wenn eine lokale Variable und eine globale Variable den gleichen Namen haben? Was passiert wenn lokale Variablen verschiedener Funktionen den gleichen Namen haben?

Diskussion:

Kann man eine oder beide der globalen Variablen im Beispiel zu Block 2 auch als lokale Variablen definieren?

Diskussion:

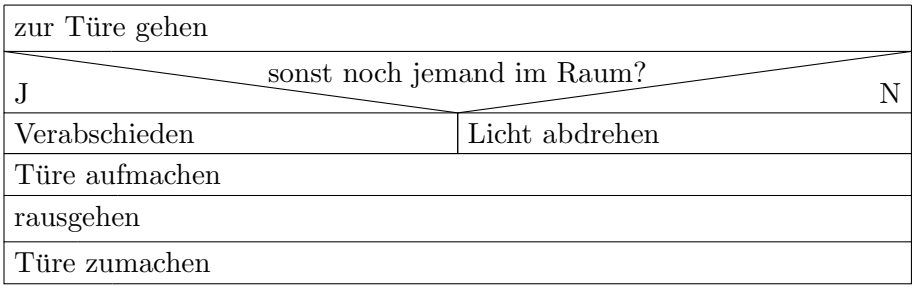
Man kann mit Kommata getrennt auch mehrere Variablen auf ein mal definieren und man kann der Variable bereits direkt in der Definition einen initialen Wwert zuweisen. Zum Beispiel:

```
int a = 1, b = 2, c = a + b;
```

Was sind die Vor- und Nachteile?

Block 7: Struktogramme

Ein Struktogramm ist eine graphische Darstellungsform einer Funktion die besonders leicht zu lesen ist. Dabei wird die Funktion als Rechteck dargestellt, wobei zur Visualisierung der einzelnen Verarbeitungsschritte das Rechteck in kleinere Rechtecke unterteilt wird:



Im Struktogramm stellen Blöcke untereindander eine Sequenz von hintereindander auszuführenden Aktionen dar. Blöcke die nebeneinander stehen stellen alternative Sequenzen dar.

Diskussion:

Wir haben noch keine C Sprachkonstrukte für Fallunterscheidungen kennengelernt. Wie sehen daher die Struktogramme für alle C-Funktionen aus die wir bis jetzt geschrieben haben?

Diskussion:

Was ist der Vorteil von einem Struktogramm? Wozu braucht man Struktogramme?

Übung:

Verarbeiten Sie folgende Abfragen und Aktionen in ein sinnvolles Struktogramm:

- Ist Milch da?
- Ist die Milchpackung offen?
- Ist die Milchpackung leer?
- Kühlschrank aufmachen.
- Kühlschrank zumachen.
- Milchpackung aufmachen.
- Milchpackung wegwerfen.
- Milch in Heissgetränk leeren.
- Milch kaufen.

(Mehrfachverwendungen sind möglich.)

Block 8: Fallunterscheidungen

In C werden Fallunterscheidungen mit dem `if`-Statement umgesetzt:

```
1  void setup()
2  {
3      pinMode(2, INPUT);
4      pinMode(3, INPUT);
5      pinMode(4, INPUT);
6
7      pinMode(12, OUTPUT);
8      pinMode(13, OUTPUT);
9
10     // enable pullup resistors
11     digitalWrite(2, 1);
12     digitalWrite(3, 1);
13     digitalWrite(4, 1);
14 }
15
16 void loop()
17 {
18     int valueA, valueB;
19     int doSwitch = digitalRead(2);
20
21     if (doSwitch)
22     {
23         valueA = digitalRead(3);
24         valueB = digitalRead(4);
25     }
26     else
27     {
28         valueA = digitalRead(4);
29         valueB = digitalRead(3);
30     }
31
32     digitalWrite(12, valueA);
33     digitalWrite(13, valueB);
34 }
```

block08_if.pde

Hierbei werden mehrere Statements zu Blöcken zusammengefasst, die von geschwungenen Klammern eingeschlossen werden. Die Einrückungen dienen der besseren Lesbarkeit.

Die Bedingung, die dem `if`-Statement in runden Klammern mitgegeben wird, ist vom Datentyp `int`. Ein Zahlenwert ungleich 0 führt zur Ausführung des Blocks oder Statements unmittelbar nach dem `if`-Statement. Ein Zahlenwert von 0 führt zur Ausführung des `else`-Zweiges, wenn vorhanden.

Wenn ein Zahlenwert auf diese Weise interpretiert wird, nennt man Ihn Wahrheitswert. Ein Zahlenwert von ungleich 0 wird *Wahr* und ein Zahlenwert von 0 wird *Falsch* genannt. Ausdrücke die auf Wahrheitswerte führen werden *boolsche Ausdrücke* genannt.

Diskussion:

Wie ist die Syntax des `if`-Statements?

Wann kann man die geschwungenen Klammern weglassen?

Übung:

Schreiben sie ein Programm mit dem Sie feststellen können ob die Bindung des `else`-Statements so wie im linken oder so wie im rechten Listing angedeutet ist funktioniert:

```
if (foo)
    if (bar)
        a = 1;
    else
        a = 2;
```

```
if (foo)
    if (bar)
        a = 1;
else
    a = 2;
```

Diskussion:

Warum ist es sinvoll und wichtig richtig einzurücken?

Block 9: Boolsche Ausdrücke

Wahrheitswerte können durch den Vergleich von Zahlenwerten gebildet werden:

```
1  int getFlashDurationMs()
2  {
3      int durationSeconds =
4              consoleReadDecimal("Wieviele Sekunden? ");
5
6      if (durationSeconds < 0)
7          durationSeconds = 0;
8      else if (durationSeconds > 10)
```

```
9           durationSeconds = 10;
10
11      return durationSeconds * 1000;
12  }
```

block09_rel.pde

Dabei können folgende Operatoren verwendet werden:

a `<` b	a kleiner b
a `<=` b	a kleiner oder gleich b
a `==` b	a gleich b
a `!=` b	a ungleich b
a `>=` b	a grösser oder gleich b
a `>` b	a grösser b

Wahrheitswerte können auch durch Verknüpfungen von Wahrheitswerten gebildet werden:

```
1   void setup()
2   {
3       pinMode(2, INPUT);
4       pinMode(3, INPUT);
5       pinMode(13, OUTPUT);
6       digitalWrite(2, 1); // enable pullup resistor
7       digitalWrite(3, 1); // enable pullup resistor
8   }
9
10  void loop()
11  {
12      if (!digitalRead(2) && !digitalRead(3))
13          digitalWrite(13, 1);
14      else
15          digitalWrite(13, 0);
16  }
```

block09_logic.pde

Dabei können folgende Operatoren verwendet werden:

a `&&` b	a und b
a `||` b	a oder b
`!` a	nicht a

Diskussion:

Wie interpretiert der C-Compiler folgenden Ausdruck?

```
a < b < c
```

Wie kann man diesen Ausdruck so umschreiben, dass er die in der Mathematik übliche Bedeutung hat?

Übung:

Schreiben Sie das Programm `block09_logic.pde` so um, dass die LED leuchtet, wenn mindestens eine der Tasten gedrückt ist.

Diskussion:

Wie müsste das Programm verändert werden, sodass die LED leuchtet, wenn genau eine der Tasten gedrückt ist?

Diskussion:

Erklären Sie Zweck und Funktionsprinzip des folgenden Programms:

```
1  void setup()
2  {
3      pinMode(2, INPUT);
4      pinMode(3, INPUT);
5      pinMode(12, OUTPUT); // red LED
6      pinMode(13, OUTPUT); // green LED
7      digitalWrite(2, 1); // enable pullup resistor
8      digitalWrite(3, 1); // enable pullup resistor
9  }
10
11 void loop()
12 {
13     if (!digitalRead(2) || !digitalRead(3))
14     {
15         delay(100);
16         if (!digitalRead(2) && !digitalRead(3))
17             digitalWrite(13, 1);
18         else
19             digitalWrite(12, 1);
20         delay(1000);
21     }
22     digitalWrite(12, 0);
23     digitalWrite(13, 0);
24 }
```

block09_sync.pde

Anhang I: Angehängte Dateien

Dieses PDF-Dokument enthält angehängte Dateien. Die meisten PDF-Reader erlauben es, diese Dateien zu extrahieren. Um das zu tun, muss man mit der rechten (zweiten) Maustaste auf den rot gesetzten Dateinamen klicken und den entsprechenden Eintrag im Kontextmenu wählen.

Datei 1: `texsourcen.zipx`
Die vollständigen LaTeX-Sourcen zu diesem Skriptum als ZIP Datei.
(Nach dem Abspeichern aus dem PDF nach `*.zip` umbenennen! Acrobat Reader erlaubt leider keine `*.zip` Dateien in PDFs. Daher dieser Hack..)

Datei 2: `examples.zipx`
Der Sourcecode zu allen Beispielprogrammen.

Datei 3: `Console.zipx`
Die Arduino `Console` Library die wir in diesem Kurs verwenden.