Workshop: ASURO-Programmieren in C

Markus Becker

http://mbecker-tech.de

Bürgernetz Ingolstadt e. V. / ByteWerk

Stand: 25. Dezember 2009

Inhaltsverzeichnis

Weiterführende Informationen

- ASURO-Wiki: `http://www.asurowiki.de`
- Micro-Controller: `http://www.mikrocontroller.net`
- Roboter-Community: `http://www.roboternetz.de`
- Kontakt bei Fragen: `mailto:markus@mbecker-tech.de`

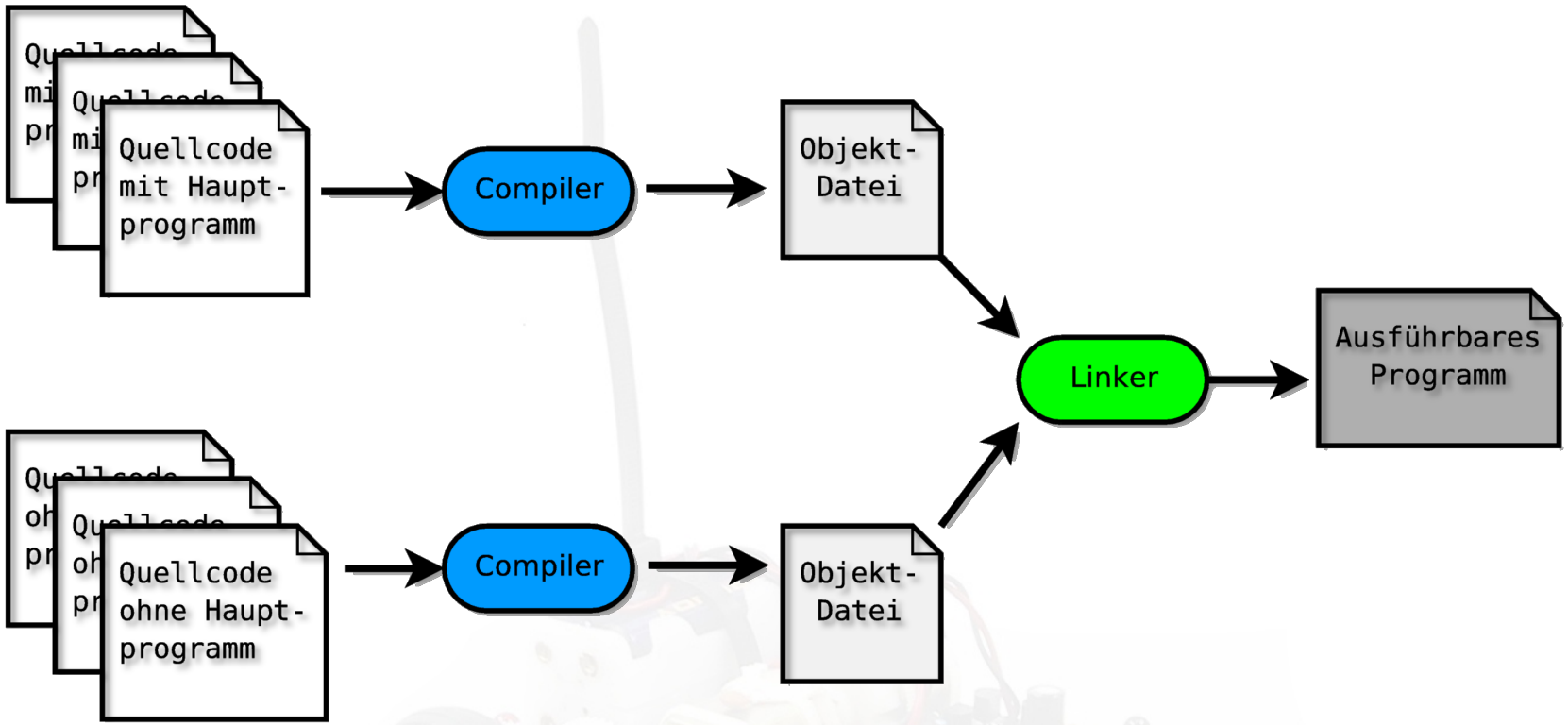
Maschinencode

- Der Micro-Controller versteht nur Maschinencode in Form von 'Nullen und Einsen'
- Jedes Byte steht für eine bestimmte Aufgabe, z. B.
  `1010 0111: Addiere Register x mit Register y`
- Dieser Maschinencode kann auf folgende Arten erzeugt werden:
  - Direkt über Hex-Editor
  - Maschinennahe Programmiersprache: *Assembler*
    ```
    mov r14,r24 # Register 24 --> Register 14
    ```
  - Hochsprache: z. B. *C* (wird in diesem Workshop verwendet)
    ```
    int x = 5; // Variable x mit 5 belegen
    ```

Maschinencode aus C-Code generieren

Die Erstellung eines ausführbaren Programms mit C erfordert verschiedene Teilschritte:

1. Mit einem geeigneten Editor wird der C-Quelltext geschrieben
2. Der Compiler (*avr-gcc*) erzeugt aus dem Quellcode Objektdateien
3. Der Linker fügt verschiedene Objektdateien zu ausführbarem Maschinencode zusammen



Visualisierter Ablauf bei der Generierung von Maschinencode

Grundgerüst eines C-Programms

- Jedes Programm hat eine `main`-Funktion, die als erstes ausgeführt wird
- Einrücken erleichtert Lesbarkeit

```
1 #include "bib.h" // Einbinden von Bibliotheken
2
3 void main (void) {
4   // hier steht nachher die Befehlsfolge,
5   // die der ASURO ausfuehren soll
6 }
```

Listing 1: Grundgerüst des C-Programms

Bezeichner in C

- Bezeichner (Variablennamen, Funktionsnamen) müssen mit einem Buchstaben beginnen
- danach können sie auch Ziffern und Unterstriche enthalten
- maximal 32 Zeichen
- zur besseren Lesbarkeit 'sprechende' Namen verwenden
- Gültig sind z. B. `var`, `var_1`, `var1`
- Ungültig sind z. B. `1var`, `_var`, `123`, `_123`

C-Syntax

- 'Regeln' analog Grammatik und Interpunktion
- Blöcke werden in geschweifte Klammern zusammengefasst
- Anweisungen werden mit einem Semikolon abgeschlossen
- *C* unterscheidet Groß- und Kleinschreibung (*'case-sensitive'*!)

```
1 void main (void) {
2   int i = 0; // Anweisung
3   ....
4   if (i == 0) {
5     // Block
6     ...
7   }
8 }
```

Listing 2: Beispiel zur C-Syntax

Kommentare

- 'Notizen' zur Dokumentation des Codes
- Erleichterung der Wartbarkeit des Codes über größeren Zeitraum

```
1 void main (void) {
2   // Einzeiliger Kommentar
3   ....
4   /*
5   hier steht ein
6   mehrzeiliger Kommentar
7   */
8 }
```

Listing 3: Kommentare

Standardbibliothek

- Es existiert eine Sammlung von Grundfunktionen, z. B. für:
  - Auslesen von Sensordaten
  - Ansteuerung der Aktoren (Motoren, LEDs)
  - Kommunikation über *RS232viaIR*
  - Zeitsteuerung
  - ...
- Abstraktion von der tieferen Funktion des Micro-Controllers
- ASURO beherrscht quasi *out-of-the-box* komplexe Vorgänge, wie z. B. Kurven fahren
- Einbinden in Quellcodedatei mit `#include ¨../asuro.h¨;`

Variablen

- Speicherplatz für z. B. Zwischenergebnisse
- haben einen nicht veränderlichen *Datentyp*
- es gelten die Regeln für Bezeichner in *C*
- Konvention: keine Großbuchstaben
- müssen vor Verwendung *definiert* werden:
  Syntax: `<Datentyp> [Variablenname];`
- sind nur in dem Block gültig, in dem sie definiert wurden

```
1 void main (void) {
2   float spannung;
3   int anzahl, index; // Zwei auf einmal
4   int i = 0; // mit Initialisierung
5   ....
```

Listing 4: Variablendefinition

Datentypen

- Ganzzahlige Datentypen (Integer):

Name	**Größe**	**min.**	**max.**
`char`	1 Byte	−128	+127
`unsigned char`	1 Byte	0	255
`int`	2 Byte	−32768	+32767
`unsigned int`	2 Byte	0	65535

- Gleitkommazahlen (Floating Point):

Name	**Größe**	**min.**	**max.**
`float`	4 Byte	ca. $-3.4 \cdot 10^{38}$	ca. $+3.4 \cdot 10^{38}$

Felder (Arrays)

- Mehrere Variablen gleichen Typs lassen sich zu Feldern (Arrays) gruppieren
- Alle Elemente eines Feldes liegen hintereinander im Speicher
- Der Name des Feldes steht für die Position im Speicher und nicht für einen Wert innerhalb des Feldes
- Erstes Element an der Stelle 0!

```
1 void main (void) {
2   int linedata[2]; // Integer-Array der Laenge
        2
3   int data = {1, 2, 3}; // Mit Initialisierung
4   linedata[0] = 0; // erstes Element
5   linedata[1] = 0; // zweites Element
6 }
```

Listing 5: Arrays

Konstanten

- Zur Vermeidung von Zahlen im Code (*'magic numbers'*)
- Möglichkeit der Wiederverwendung
- Konvention: nur Großbuchstaben
- Definition über *Präprozessoranweisung*:
  Syntax: `#define <Bezeichner> <Zahl>`

```
1 #include "../lib/asuro.h"
2 #define MAXLENGTH 5
3
4 void main (void) {
5   int feld[MAXLENGTH];
6   ....
7 }
```

Listing 6: Variablendefinition

Wichtige Operatoren

- Arithmetische Operatoren
  - + Addition, - Subtraktion; Bsp: `a = x + i;`
  - * Multiplikation, / Division; Bsp: `a = x * i;`
  - % Modulu (Divisionsrest); Bsp: `a = 10 % 3; // 1`
- Relationale Operatoren
  - == gleich, != ungleich; Bsp: `a = (4 == 3); // False`
  - < kleiner als, > größer als; Bsp: `a = (4 > 3); // True`
  - <= kleiner gleich, >= größer gleich
  - Ergebnis: `True` (nicht Null) bzw. `False` (Null)

Wichtige Operatoren

- Logische Operatoren
  - `!` Negation; Bsp: `a = !(4 > 3); // False`
  - `&&` logisches UND, `||` logisches ODER
  - Beispiel: `a = ((3 > 2) || (3 < 2)); // True`
- Operatoren zur Bitmanipulation
  - `~` NOT; Beispiel:
    `a = ~ 9; // -10` (~0000 1001 → 1111 0110)
  - `&` bitweises UND, `|` bitweises ODER; Beispiel:
    `a = 3 || 9; // 11` (0011 | 1001 → 1011)
  - `<<` Shift left, `>>` Shift right; Beispiel:
    `a = 1 << 3; // 8` (0001 drei mal nach rechts → 1000)

Übung 1

```
1 void main (void) {
2   int i = 15, j = 30, c;
3
4   c = j * 2 + i;
5   c = (i + j >= 5) && (i % 5 == 0);
6   c = i & (1 << 6);
7 }
```

Listing 7: Übung 1

- Welchen Wert hat die Variable c in den Zeilen 4, 5 und 6?

Lösung zu Übung 1

```
1  int i = 15, j = 30, c;
2
3  c = j * 2 + i;
4  // = 30 * 2 + 15 = 75
5
6  c = (i + j >= 5) && (i % 5 == 0);
7  // = (30 + 15 >= 5) && (15 % 5 == 0)
8  // = True && True = True
9
10 c = i & (1 << 6);
11 // = 15 & (1 << 6)
12 // = 15 & 64
13 // = 0000 1111 & 0100 0000 = 0
```

Listing 8: Lösung zu Übung 1

Übung 2

```
1 #define LINKS 0
2 #define RECHTS 1
3 void main (void) {
4   int data[2] = {12, 13};
5   int diff;
6
7   diff = data[RECHTS] - data[LINKS];
8
9   }
```

Listing 9: Übung 2

- Welchen Wert hat die Variable `diff` in Zeile 7?

Lösung zu Übung 2

```
1  #define LINKS 0
2  #define RECHTS 1
3  void main (void) {
4    int data[2] = {12, 13};
5    int diff;
6
7    diff = data[RECHTS] - data[LINKS];
8    // = data[1] - data[0]
9    // = 13 - 12 = - 1
10
11   }
```

Listing 10: Lösung zu Übung 2

if-Bedingung

- Bedingte Ausführung von Anweisungsblöcken
- Parameter: Logischer Ausdruck (z. B. Vergleich)
- Mehrere Ausdrücke über logische Operatoren verknüpfbar

```
1  void main (void) {
2    ...
3    if (diff > 0) {
4      drive_left (); // JA, diff > 0
5    }
6    else {
7      drive_right (); // NEIN, diff <= 0
8    }
9    if ((diff > 0) && (xy != 1)) {
10     ...
11 }
```

Listing 11: Beispiel zu if

else-if-Bedingung

- Verketten von `if`-Anweisungen

```
1  void main (void) {
2    ...
3    if (diff > 0) {
4      drive_left ();
5    }
6    else if (diff <= 0){
7      drive_right ();
8    }
9    else {
10     drive_straight ();
11   }
12 }
```

Listing 12: Beispiel zu else-if

switch-Bedingung

- Ketten von `else if`-Anweisungen vereinfachen
- Mit `break` wird wieder aus dem `switch`-Block gesprungen
- `default`-Zweig für Fehlerbehandlung nützlich

```
1  switch (diff) {
2    case 1:
3      drive_left ();
4      break;
5    case -1:
6      drive_right ();
7      break;
8    default:
9      error ();
10 }
```

Listing 13: Beispiel zu switch

while-Schleife

- Bedingte Wiederholung von Anweisungsblöcken
- Parameter: wie `if`
- Achtung! Gefahr von *Endlosschleifen*

```
1 void main (void) {
2   ...
3   while (diff > 0) {
4     drive_left ();
5     diff = diff - 1;
6   }
7 }
```

Listing 14: Beispiel zu while

Warum Funktionen

- Anweisungsblöcke lassen sich zu Funktionen zusammenfassen
- Möglichkeit zur Wiederverwendung
- Bessere Lesbarkeit
- Viele fertige Funktionen bereits in der Bibliothek vorhanden
- Funktionen können Parameter übergeben werden
- Funktionen können selbst einen (Ergebnis-)Wert zurückgeben
- Jedes Programm besitzt eine `main`-Funktion

Syntax bei Funktionen

- Aufruf: `[Rückgabewert = ]<Funktionsname> ([Parameter]);`
  Beispiel: `MotorSpeed (255, 255);`
- Definition: es gelten die Regeln für Bezeichner in *C*
  `<Rückgabewerttyp> <Funktionsname> ([Typ Parameter]){`
  `// hier folgt der Funktionscode}`
  Beispiel: `int maximum (int par1, int par2){...}`
- Definition von Funktionen ohne Rückgabewert / Parameter:
  `void <Funktionsname> (void){....}`
  Beispiel: `void beep (void){...}`
- Rückgabe eines Wertes am Ende mit `return <variable>`
  Beispiel: `return x;`
- Funktionen müssen VOR ihrem ersten Aufruf definiert werden, also am besten oben in der Datei zwischen `#define`'s und `main ()`

Beispiel zu Funktionen

```
1  int max (int a, int b) {
2    if (a > b)
3      return a;
4    else
5      return b;
6  }
7  void main (void) {
8    int i = 15, j = 30, m = 0;
9    m = max (i, j); // m = 30
10   max (i, j); // Ergebnis verfaellt
11 }
```

Listing 15: Beispiel zu Funktionen

void Init (void)

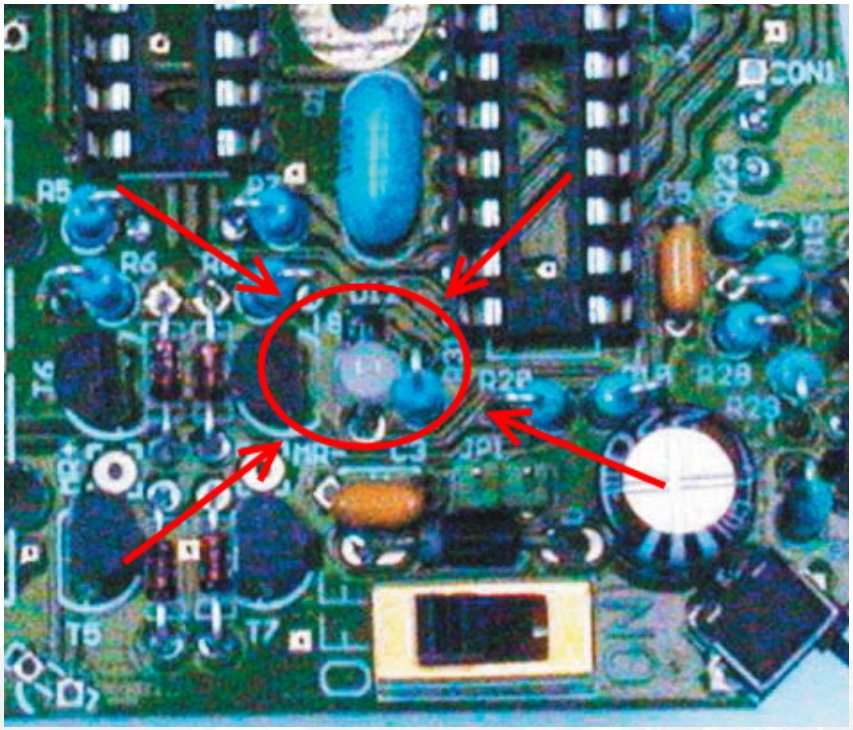
- Initialisierung des Micro-Controllers (Frequenz, Ein-/Ausgangspins, Timer, ...)
- Deklaration von Konstanten fertiger Funktionen
- muss am Anfang der `main`-Funktion stehen

```
1 void main (void) {
2   int i = 0;
3   ....
4   Init ();
5   ....
6 }
```

Listing 16: Beispiel zur Init ()

void StatusLED (unsigned char color)

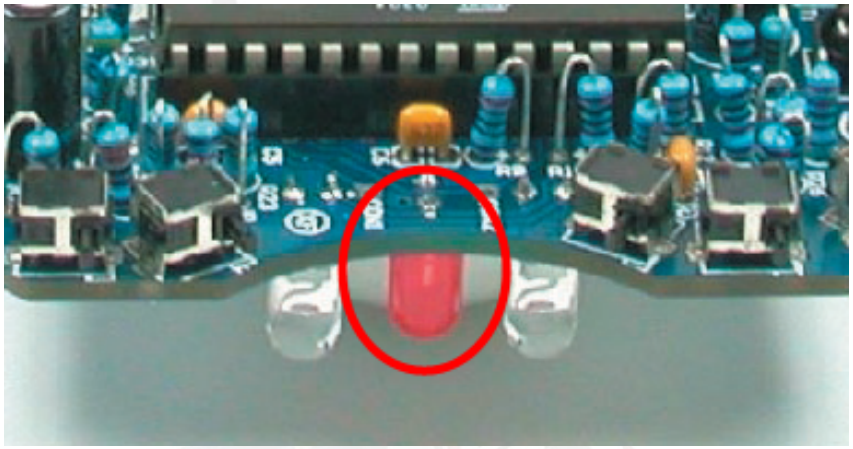
- Beeinflussung der Farbe der Status-LED



- Parameter: Vordefinierte Konstanten:
  `OFF`, `GREEN`, `RED` oder `YELLOW`

void FrontLED (unsigned char status)

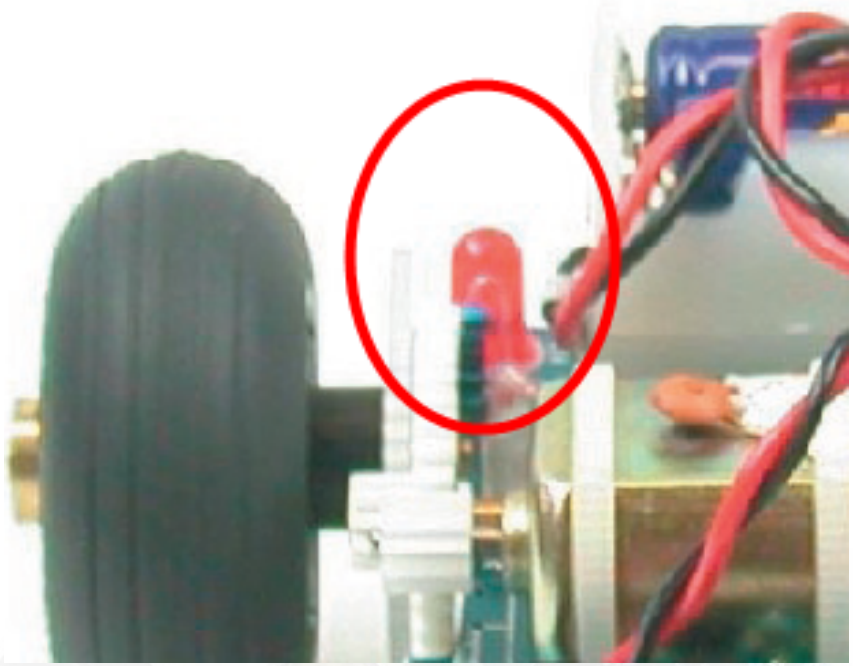
- Ein-/Ausschalten der Front-LED



- Parameter: Vordefinierte Konstanten `ON` oder `OFF`
  Beispiel: `FrontLED (ON); // einschalten`
- Verwendung: 'Straßenbeleuchtung' bei Linienerkennung

void BackLED (unsigned char left, unsigned char right)

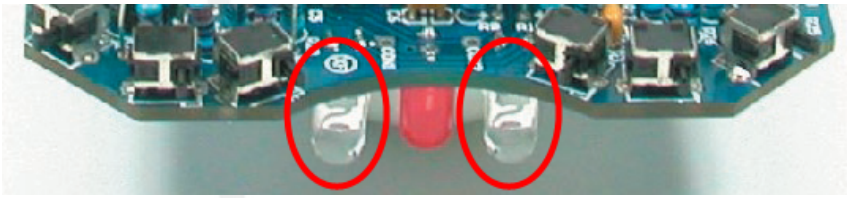
- Ein-/Ausschalten der hinteren LEDs



- Parameter: Vordefinierte Konstanten `ON` oder `OFF`
  Beispiel: `BackLED (ON, OFF); // links an, rechts aus`
- Verwendung: 'Blinker' oder einfach nur zu Testzwecken

void LineData (unsigned int *data)

- Auslesen der beiden Photodioden am Boden



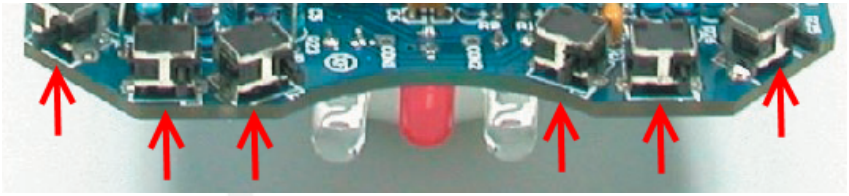
- Parameter: Adresse eines `unsigned int`-Arrays, die Funktion schreibt die Daten in dieses Array
- Wertebereich: maximal hell: 1023, maximal dunkel: 0

```
1 void main (void) {
2   unsigned int data[2];
3   Init ();
4   LineData (data);
5   if (data[LEFT] < data[RIGHT])
6     // Rechts neben Linie
7 }
```

Listing 17: Beispiel zu LineData

void LineData (unsigned char PollSwitch(void)

- Auslesen der vorderen Kollisionstaster



- Rückgabewert: `unsigend char`, in dem jedes Bit einem Schalter entspricht

Bit	Wert	Taster
0	1	K6
1	2	K5
2	4	K4
3	8	K3
4	16	K2
5	32	K1

```
1  void main (void) {
2    unsigned int switch;
3    Init ();
4    MotorDir (FWD, FWD);
5    MotorSpeed (125, 125);
6    while (1) {
7      switch = PollSwitch ();
8      if (switch != 0) { // Kollision
9        if(switch <= 7){ // Tastsignal RECHTS
10               BackLED(OFF, ON);
11               hindernis_rechts();
12       }
13       else { // Tastsignal LINKS
14               BackLED(ON, OFF);
15               hindernis_links();
16       }
17     }
18   }
19 }
```

Listing 18: Beispiel zu PollSwitch

void MotorDir (unsigned char left, unsigned char right)

- Beeinflussung der Drehrichtung der beiden Motoren
- Parameter: Vordefinierte Konstanten:

Parameter	**Wirkung**
FWD	vorwärts
RWD	rückwärts
BREAK	Bremsen
FREE	Freilauf

- Beispiel: Links vorwärts, rechts Stillstand:

```
MotorDir (FWD, BREAK);
```

void MotorSpeed (unsigned char left, unsigned char right)

- Beeinflussung der Drehzahl der beiden Motoren
- Parameter: 2x `unsigned char`, Wertebereich: 0 . . . 255
- Beispiel: Links maximale Drehzahl, rechts Stillstand: `MotorSpeed (255, 0);`

```
1 void main (void) {
2   Init ();
3   while (1) {
4     MotorDir (FWD, FWD); // vorwaerts
5     MotorSpeed (255, 255); // vollgas
6   }
7 }
```

Listing 19: Beispiel zur Motoransteuerung (Geradeausfahrt)

Vielen Dank für die Aufmerksamkeit!