Arduino verstehen und anwenden

Teil 4: Programmierpraxis: Befehle, Variablen und Funktionen

Im vierten Teil der Serie „Arduino verstehen und anwenden" geht es um den Einstieg in die Programmierpraxis. Hier werden die Strukturen vorgestellt, ohne die keine höhere Programmiersprache auskommt:

- Syntaxelemente
- Konstanten und Variablen
- Operatoren
- Kontrollstrukturen

Wie sonst auch soll in diesem Beitrag die Praxis nicht zu kurz kommen. So werden die einzelnen Abschnitte wieder durch Anwendungsbeispiele für den Arduino ergänzt und vertieft.

Die Sprachelemente von „Processing" bzw. „C"

Nach dem Durcharbeiten der ersten drei Artikel dieser Serie ist man nun bereits in der Lage, einfachere Programme zu verstehen, zu modifizieren und in den Controllerchip des Arduinos zu laden. Es ist nun also an der Zeit, sich etwas näher mit den Details der Programmierung zu beschäftigen.

Nachfolgend werden daher die wesentlichen allgemeinen Sprachelemente von Processing erläutert. Weitere Erklärungen zu speziellen Befehlen und Anweisungen folgen in den jeweiligen Anwendungen.

Programm-Struktur

Wie wir in vorhergehenden Artikeln am Beispiel „Blink" schon gesehen haben, besteht ein Sketch aus zwei Hauptteilen:

1) void setup()
2) void loop()

Die Setup-Funktion wird beim Programmstart genau einmal aufgerufen. Hier werden Schnittstellen und Pin-Funktionen etc. initialisiert.

Void setup() muss immer im Programm stehen, auch wenn nichts explizit definiert werden muss. Die Routine hat nämlich auch die Aufgabe, Standardfunktionen zur Verfügung zu stellen.

In der Endlosschleife loop() wird der eigentliche Programmcode festgelegt. Alle in der „loop" enthaltenen Anweisungen und Befehle werden nacheinander immer wieder ausgeführt, bis der Controller von der Versorgungsspannung getrennt wird.

Einfache Syntaxelemente

Ähnlich wie eine natürliche Sprache ihre Satzzeichen hat, so benötigt auch Processing einen gewissen Grundvorrat an Syntaxelementen:

- Semicolon ;
  Das Semicolon schließt jeden Befehl ab.
- Geschweifte Klammern {}
  Geschweifte Klammern fassen einen Befehlsblock zusammen. Beispiele hierzu finden sich weiter unten bei den Kontrollstrukturen.
- Einzeiliger Kommentar //
  Mit dem Doppel-Slash wird ein einzeiliger Kommentar gekennzeichnet.
- Mehrzeiliger Kommentar /* */
  Erstreckt sich ein Kommentar über mehrere Zeilen muss dieser mit /* begonnen und mit */ beendet werden.

Konstanten und Variablen

Zu den wichtigsten Elementen einer Programmiersprache gehören die Konstanten und Variablen. Konstanten dienen zur Kennzeichnung unveränderlicher Werte. In Processing wurden einige vordefinierte Konstanten festgelegt. Diese machen den Programmcode einfacher lesbar und erleichtern zudem den Einstieg in die Arduino-Programmierung. Die wichtigsten vordefinierten Konstanten sind:

- **TRUE/FALSE**
  FALSE ist als 0 (Null) definiert,
  TRUE ist ungleich 0.
- **HIGH/LOW**
  HIGH bedeutet Logiclevel 1, d. h. ON oder 5 V,
  LOW bedeutet Logiclevel 0, d. h. OFF oder 0 V.
- **INPUT/OUPUT**
  Sind für die pinMode()-Funktion vorgesehen und legen fest, ob ein PIN als Eingang (INPUT) oder Ausgang (OUTPUT) genutzt werden soll.

Variablendefinition

Die Festlegung von Variablen ist bereits aus früheren Artikeln bekannt, z. B. wird mit der Programmzeile:

```
int ledPin = 10;
```

eine Variable „ledPin" festgelegt und auch gleich auf den Wert 10 gesetzt.

Variablen zeichnen sich dadurch aus, dass sie während des Programmablaufes verändert werden.

Da Variablen in der Programmierung von höchster Wichtigkeit sind, wurden verschiedene Variablen-Typen festgelegt. Diese sind in Tabelle 1 zusammengefasst.

Die verschiedenen Typen dienen dazu, die begrenzten Speicherplatzressourcen eines Mikrocontrollers optimal zu nutzen. Man sollte also immer den Typen auswählen, der nicht mehr Speicher belegt als notwendig.

Außerdem ergeben sich bei der korrekten Auswahl von Variablen-Typen auch Geschwindigkeitsvorteile. So kann der Arduino Integer-Berechnungen deutlich schneller ausführen als Float-Operationen.

Bei einfachen Programmen, die nur wenige Berechnungen enthalten, fällt das natürlich kaum ins Gewicht. Wenn allerdings komplexere Rechenvorgänge mit umfangreichen Daten erforderlich sind, so kann die Verwendung einer reinen Integer-Arithmetik über den Erfolg oder Misserfolg eines Projektes entscheiden.

Klassische Beispiele, die eine hohe Rechengeschwindigkeit erfordern, sind schnelle digitale Regelungen. Wird hier mit optimierten Integer-Werten gearbeitet, kann der Arduino bereits recht komplexe Aufgaben wie etwa die Lageregelung eines Quadrokopters übernehmen.

Variablenfelder

Variablenfelder, auch als „arrays" bezeichnet, stellen eine Zusammenfassung von Variablen dar, auf die mit Hilfe einer Index-Zahl zugegriffen werden kann.

Alle nachfolgenden Anweisungen legen jeweils ein Array fest:

```
int myIntegers[4];
int OutPins[] = {1, 3, 5, 7, 9};
int mySpecialValues[6]
= {1, 3 ,3, -4, -1, 5};
char infotext[6] = „hallo";
```

Tabelle 1

Die wichtigsten Variablen-Typen

Typ	Bytes	Bereich		
Byte	1	0	bis	255
Char	1	-127	bis	127
Int	2	-32.768	bis	32.767
Long	4	-2.147.483.648	bis	2.147.483.647
Float	4	-3,4028235E+38	bis	3,4028235E+38

Mit diesen Definitionen werden jeweils Felder mit 4, 5, 6 und wieder 6 Variablen erzeugt.

Der Zugriff auf ein Feldelement erfolgt durch Zuweisungen unter Benutzung des jeweiligen Index, z. B:

```
OutPins[0] = 1;
x = mySpecialValues[4];
```

Wichtig:
Die Indizes eines Array beginnen mit 0, das heißt, das erste Feldelement hat immer den Index „0".

Daraus ergibt sich, dass für ein Feld mit beispielsweise 4 Variablen der letzte Index 3 lautet. Wird eine Feldvariable mit dem Index 4 abgerufen, ergibt sich ein zufälliger Wert!

Beispiel:

```
int myNumbers[4]={1,4,5,2};
// myArray[3]   enthält den Wert 2,
// myArray[4]   ist ungültig und liefert einen
zufälligen Wert!
```

Mathematische Operatoren

Um mit Variablen rechnen zu können, werden Operatoren benötigt. Die wichtigsten Operatoren sind die Rechenzeichen. Sie haben in Processing die üblichen mathematischen Bedeutungen:

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo, d. h. Rest der Division

Auch für die Vergleichsoperatoren gelten die normalen mathematischen Bedeutungen:

==	ist gleich
!=	ungleich
<	kleiner
>	größer
<=	kleiner gleich
>=	größer gleich
=	Zuordnung

Lediglich die Zeichen wie etwa „≤" für „kleiner gleich" wurden durch ASCII-kompatible Doppelzeichen wie „<=" ersetzt.

Hinweis:
Die Zuordnung „=" darf nicht mit dem Gleichheitsoperator „==" verwechselt werden!
Im ersten Fall wird einer Variablen ein Wert zugewiesen, z. B.:

```
ledPin = 10;
```

Im zweiten Fall wird auf Gleichheit überprüft, z. B.:

```
if (a == 1)
```

Die Boole'schen Logikoperatoren werden durch die folgenden Zeichen realisiert:

&&	(and)
||	(or)
!	(not)

Auch sie besitzen die aus der Mathematik bekannte Bedeutung.

Die in C üblichen Abkürzungen, auch als Compound-Operatoren bekannt, sind in Processing ebenfalls verfügbar:

++	Increment:	x++	steht für x = x+1
--	Decrement:	x--	steht für x = x-1
+=	Compound Addition:	x += y	steht für x = x+y
-=	Compound Subtraktion:	x -= y	steht für x = x-y
*=	Compound Multiplikation:	x*= y	steht für x = x*y
/=	Compound Division:	x/= y	steht für x = x/y

Kontrollstrukturen

Zur Steuerung von Programmabläufen sind in Processing bzw. C verschiedene Strukturen vorgesehen. IF/ELSE und FOR sind die beiden wichtigsten.

- if
  überprüft ob eine bestimme Bedingung erfüllt ist:

```
if (Variable > 99)
{  // Anweisungen...
}
```

- if...else
  gestattet eine weitergehende Kontrolle des Programmflusses. Es können mehrer Bedingungen unterschieden werden:

```
if (Input < 500)
{  // Aktion A
}
else
{  // Aktion B
}
```

- for
  erlaubt es, eine Anweisung zu wiederholen. Die Anzahl der Wiederholungen ist dabei nicht von irgendwelchen Bedingungen abhängig:

```
for (init; cond; inc)
  {  //Anweisungen...
  }
```

Zunächst wird eine Startbedingung „init" gesetzt. Jedesmal wenn die Schleife durchlaufen wurde, wird die Bedingung „cond" geprüft und die Aktion „inc" ausgeführt. Wenn die Bedingung nicht mehr erfüllt ist, wird die Schleife beendet.

- while()
  while-Schleifen werden so lange durchlaufen, bis die Abbruchbedingung erfüllt ist:

```
while(logischer Ausdruck)
{  // Anweisungen...
}
```

- do... while
  Die do...while-Schleife funktioniert ähnlich wie die while-Schleife mit dem Unterschied, dass die Bedingung erst am Ende der Schleife geprüft wird. Die do...while-Schleife wird also immer mindestens einmal durchlaufen.
- do

```
{// Anweisungen...
} while (logischer Ausdruck);
```

- break
  Mit break kann jede do-, for-, oder while-Schleife verlassen werden.

Programmierpraxis

Nachdem nun verschiedene Programmelemente theoretisch beleuchtet wurden, sollen diese in einem Praxisbeispiel umgesetzt werden.

Das folgende Programm „squares" soll eine Tabelle der Quadratzahlen von 1 bis 9 in gut lesbarer Form ausgeben.

```
// sqares

int x;
int y[10];

void setup()
{  Serial.begin(9600);
   Serial.println("Squares table");
   Serial.println();
}

void loop()
{ for (int x = 1; x<=9; x++)
  {  y[x] = x*x;
     Serial.print(x, DEC);
     Serial.print("     ");
     Serial.println(y[x], DEC);
  }
  while(1);
}
```

Zunächst wird eine Variable vom Typ Integer definiert. Dann folgt die Deklaration eines Arrays y, für das 10 Elemente reserviert werden.

Die Anweisungen im Setup-Teil sorgen für die Kommunikation zum PC. Sie werden etwas weiter unten erläutert.

Die Zeile

```
for (int x = 1; x<=9; x++)
```

definiert eine Schleife, welche die Zahlen 1 bis 9 durchläuft.

Mit

```
y[x] = x*x;
```

werden den Elementen des Arrays y die Quadrate des Laufindex x zugewiesen. Der Laufindex der Schleife wird zudem in der nächsten Programmzeile,

```
Serial.print(x, DEC);
```

an den PC ausgegeben. Die Zeile

```
Serial.print("     ");
```

sorgt durch Einfügen von Leerzeichen für eine lesbare Formatierung der ausgegebenen Werte.

Schließlich werden die im Array gespeicherten Quadratzahlen ausgegeben:

```
Serial.println(y[x], DEC);
```

Die Anweisung while(1) sorgt dafür, dass das Programm hier stehenbleibt. Bild 1 zeigt den Code nach erfolgreicher Compilierung in der IDE.

Nach dem Laden des Programms in den Arduino muss noch der sogenannte „Serielle Monitor" gestartet werden. Dazu muss auf das Lupensymbol rechts oben in der IDE geklickt werden (siehe Bild 2). Hier erfolgt dann die Ausgabe der gewünschten Quadratzahlen-Tabelle, so wie in Bild 3 dargestellt.

Um den Ausgabekanal zu aktivieren, wurde im setup die Zeile

```
Serial.begin(9600);
```

eingefügt. Danach können dann über die Serial.print-Anweisung Werte und Texte wie beispielsweise

```
Serial.println("Squares table");
```

am PC-Bildschirm ausgegeben werden.

Über

```
Serial.print(x, DEC);
```

wird der Wert der Variablen x zum seriellen Monitor gesendet. Der Zusatz „DEC" sorgt dabei dafür, dass die Variable im Dezimalsystem ausgegeben wird. Wird dem Print-Befehl die Endung „ln" (für engl. „line") angefügt, so wird eine neue Ausgabezeile begonnen.

Weitere Details zu Nutzung dieser seriellen Schnittstelle folgen in einem späteren Artikel.

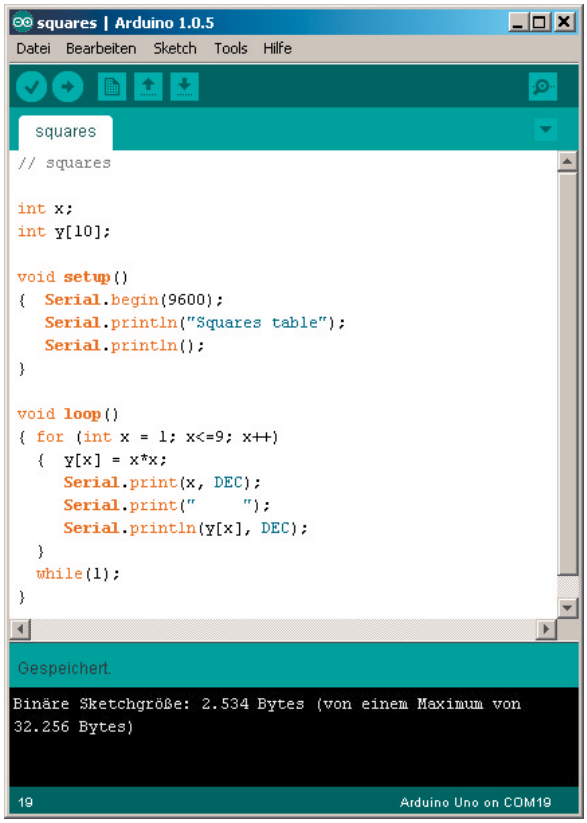

Bild 1: Das Programm „squares" in der IDE

Bild 2: Start des seriellen Monitors

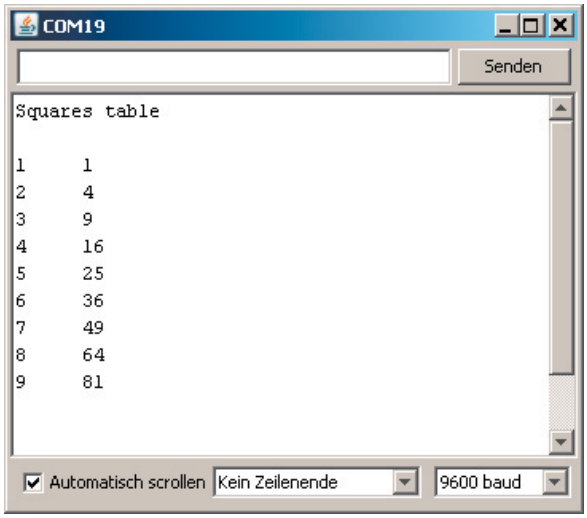

Bild 3: Ausgabe von Quadratzahlen

Übungen und Variationen

Falls Sie mit den Programmelementen von C noch nicht besonders gut vertraut sind, sollten Sie die folgenden Übungen durchführen:

- Lassen Sie sich die Quadratzahlen von 10 bis 100 berechnen und ausgeben.
- Ersetzen Sie die for-Schleife durch eine while-Schleife.
- Ersetzen Sie den Compound-Operator x++ durch die entsprechende Langschreibweise.
- Berechnen Sie nur die Quadrate der ungeraden Zahlen zwischen 11 und 25!
- Für Zahlen größer 4 soll nicht das Quadrat, sondern die 3. Potenz ausgegeben werden (Hinweis: if-Verzweigung!).

Spezielle Funktionen

Processing verfügt über spezielle Funktionen, die vor allem den Einstieg in die Programmiersprache C sehr erleichtern. Diese Funktionen werden der Übersichtlichkeit halber im Folgenden zusammengefasst:

- pinMode(pin, mode)
  konfiguriert den „pin" entweder als Eingang oder als Ausgang (mode = INPUT oder mode = OUTPUT)
- digitalWrite(pin, value)
  setzt den „pin" auf high oder low (value = HIGH oder value = LOW)
- int digitalRead(pin)
  liest den Pegel von „pin" und gibt entsprechend HIGH oder LOW zurück

Zeitsteuerung

Bereits im Programm „Blink" wurde von Zeitverzögerungen Gebrauch gemacht. Die folgenden Befehle stehen dafür zu Verfügung.

- unsigned long millis()
  gibt die Anzahl der Millisekunden an, die vergangen sind, seit das aktuelle Programm gestartet wurde

Hinweis
Die Variable millis springt nach ca. 50 Tagen auf null zurück. Bei Langzeitanwendungen ist dies unbedingt zu beachten, da es sonst zu unerwarteten Effekten kommen kann!

Die bereits in mehreren Beispielen verwendete Funktion

- delay(ms)
  bewirkt eine Verzögerung von ms, also Millisekunden, im Programmablauf
- delayMicroseconds(us)
  sorgt für eine Verzögerung von us, also Mikrosekunden, im Programmablauf

Mathematische und trigonometrische Funktionen

Abschließend sollen hier noch die wichtigsten mathematischen Funktionen angesprochen werden. Diese haben auch in Processing ihre übliche Bedeutung:

- min(x, y); max(x, y); abs(x)
  liefern das Minimum, das Maximum von x und y bzw. den Absolutwert von x
- pow(base, exponent); sqrt(x)
  berechnen die Potenzfunktion und die Quadratwurzel
- sin(rad); cos(rad); tan(rad)
  liefern entsprechend die Sinus-, Kosinus- und Tangenswerte im Bogenmaß

Zufallszahlen

Die Funktion

- randomSeed(seed)
  startet den Pseudo-Zufallsgenerator; mit seed wird dabei der Startpunkt in einer pseudo-zufälligen Zahlenreihe festgelegt
- long random(min, max)
  erzeugt eine Pseudo-Zufallszahl zwischen min und max

Praxisübungen

- Ändern Sie das Programm „squares" so ab, dass es die Quadratwurzeln der ganzen Zahlen zwischen 0 und 9 berechnet!
  Welcher Variablentyp ist dabei für y sinnvoll?
- Erstellen Sie eine Tabelle der Sinuswerte für die Winkel zwischen 0° und 360° in 10°-Schritten!
- Dass der Arduino auch einfache Grafiken ausgeben kann, zeigt Bild 4. Schaffen Sie es, das zugehörige Programm zu erstellen?

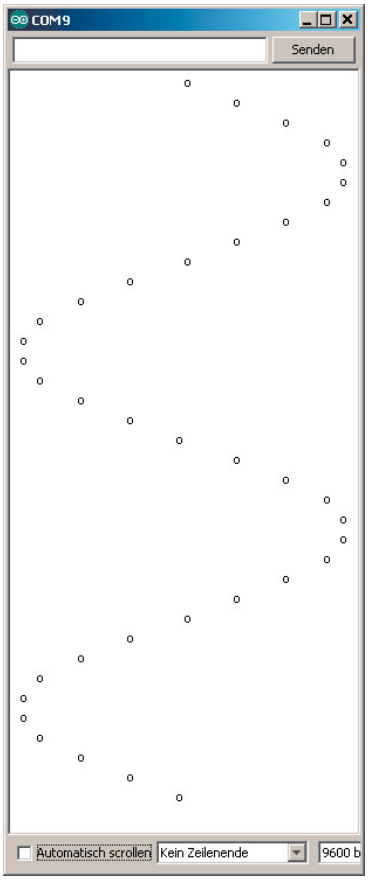

Bild 4: Arduino zeichnet eine Sinus-Kurve!

Ausblick

Im nächsten Artikel wird erklärt, wie Programm-Bibliotheken für den Arduino nutzbringend eingesetzt werden können. Auch die Erstellung eigener Bibliotheken wird erläutert werden.

Empfohlenes Material

Natürlich können die Programmelemente einer so umfangreichen Sprache wie C bzw. Processing in Rahmen eines einzelnen Beitrags nicht in allen Details dargelegt werden. Weitergehende Informationen zum Thema finden sich jedoch im Mikrocontroller-Onlinekurs [1]. Weiterführende praktische Beispielprogramme und interessante Anwendungen sind in [2] zu finden. ELV

Weitere Infos:

[1] Mikrocontroller-Onlinekurs,
Franzis-Verlag, exklusiv für ELV, 2011, Best.-Nr. J3-10 20 44
[2] G. Spanner: Arduino – Schaltungsprojekte für Profis,
Elektor-Verlag, 2012, Best.-Nr. J3-10 94 45

Preisstellung April 2014 – aktuelle Preise im Web-Shop

Empfohlene Produkte/Bauteile:	Best.-Nr.	Preis
Arduino-Uno-Platine R3	J3-10 29 70	€ 27,95
USB-2.0-Verbindungskabel A auf B	J3-09 55 56	€ 0,95
Mikrocontroller-Onlinekurs	J3-10 20 44	€ 99,–

Alle ARDUINO-Produkte wie Mikrocontroller-Platinen, Shields, Fachbücher und Zubehör finden Sie unter: www.arduino.elv.de