

Arduino – Programmierpraxis: Messwerterfassung

Eingabe von Werten in Feldvariable (3)

Eine bei der Programmierung des Arduino-Boards recht häufige Variante des Einlesens von Werten in eine Feldvariable ist das **Speichern von Messwerten** eines Sensors. Ein solcher Sensor, ein lichtempfindlicher Widerstand (LDR), wurde bereits bei den Arbeitsanregungen 4) bis 6) der Datei **ST_AA_08Arduino_Verzweigungen** verwendet – sieh' dir bitte diese Aufgabenstellungen und deine Lösungen dazu nochmals an, dann sollte das Verstehen des nebenstehenden Beispielcodes zum Speichern von Messwerten in einer `int`-Feldvariablen namens `intArray` der Größe 5 (vgl. obige Codefragmente) und das darauffolgende Ausgeben der in der Feldvariablen gespeicherten Werte kein Problem sein...

```
int intArray[5];
int index =0;

...

void loop() {
  // Befüllen eines int-Feldes mit Messwerten
  // index wird außerhalb deklariert
  // und mit 0 initialisiert

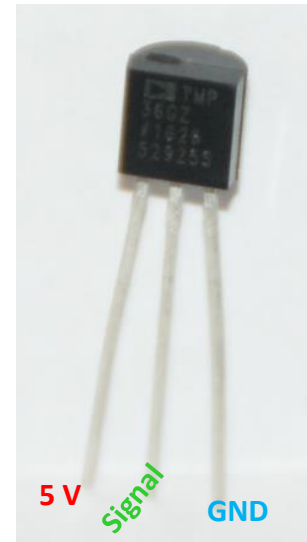
  sensorValue = analogRead(A0);
  if(index < 5){
    intArray[index] = sensorValue;
    index = index + 1;
  }
  else{
    // das Feld ist mit Werten befüllt
    // und kann im else-Zweig über den
    // seriellen Monitor ausgegeben werden
  }
}
```

Charakteristika einiger weiterer Sensoren zur Messwernerfassung

Neben dem bereits bekannten Photowiderstand werden im zweiten Teil der Arbeitsaufträge in der Datei ST_AA_10Arduino_Praxis_Feldvariable die folgenden Sensoren verwendet:

Der **TMP36 Temperatursensor** ist wie der Photowiderstand ein analoger Sensor. Dies bedeutet, dass dieser Sensor einen der aktuell gemessenen Temperatur entsprechenden Spannungswert von 0 V bis 5 V als Zahl von 0 bis 1023 codiert und diese Zahl über einen analogen Steckkontakt dem Arduino-Board als Signal mitteilt.

Der Körper des **TMP36-Temperatursensors** ist auf einer Seite nach außen gewölbt, auf der gegenüberliegenden Seite plan und hat drei Kontakte: Wird der Sensor so gehalten, dass die plane Seite zum Betrachter weist (vgl. nebenstehende Abbildung), so ist der linke Kontakt mit dem Pluspol (**5 V**), der rechte Kontakt mit dem Minuspol (**GND**) und der mittlere Kontakt schließlich mit einem analogen Steckkontakt (z.B. A0) zu verbinden.



Damit können die als Zahlen von 0 bis 1023 codierten Messwerte, d.h. die Signale, im Programm z.B. durch `sensorValue = analogRead(A0);` in einer `int`-Variablen gespeichert werden, und dann durch

$$\text{tempC} = ((\text{sensorValue}/(\text{float})1024) * 5 - 0.5008) * 100;$$

in die entsprechende Temperatur in Grad Celsius umgerechnet werden – beachte dabei, dass die Variable `tempC` vom Datentyp `float` vereinbart werden muss!

Mit dem **HC-SR04-Ultraschallsensor** können Entfernungen von zwei bis 400 Zentimeter nach folgendem Prinzip gemessen werden: Die wie „Augen“ aussehenden Bestandteile des Sensors sind ein Ultraschallsender (in der Abbildung links) und ein Ultraschallempfänger (in der Abbildung rechts). Der Sensor misst, wie lange die vom Sender ausgesandte Ultraschallwelle braucht, um an einem Objekt reflektiert zu werden und zum Empfänger zurückzukommen.



Die Spannungsversorgung des Sensors erfolgt über die beiden äußeren Steckkontakte: Der mit `vcc` beschriftete Kontakt ist mit dem Pluspol (**5 V**), der mit `Gnd` beschriftete Kontakt mit dem Minuspol (**GND**) zu verbinden. Die beiden mittleren, mit `Trig` bzw. `Echo` beschrifteten Kontakte sind jeweils mit einem der digitalen Kontakte am Arduino-Board zu verbinden – die Nummern dieser beiden Kontakte

werden nachfolgend als in den Variablen `triggerPin` und `echoPin` gespeichert angesehen. Der Modus des `triggerPin`-Kontaktes ist im `setup`-Programmblock auf `OUTPUT`, jener des `echoPin`-Kontaktes auf `INPUT` zu setzen.

Der Code zur Messung der Laufzeit des Ultraschalls bzw. zur Berechnung der Entfernung (in cm) ist nebenstehendem Codefragment zu entnehmen.

Zuerst wird über die gelb unterlegten Programmbe-
fehle das Aussenden einer Ultraschallwelle bewirkt („getriggert“).

```
//collecting data
digitalWrite(triggerPin, LOW);
delay(5);
digitalWrite(triggerPin, HIGH);
delay(10);
digitalWrite(triggerPin, LOW);
timePassed = pulseIn(echoPin, HIGH);
distance = timePassed*0.03432/2;
```

Mit dem Befehl `timePassed = pulseIn(echoPin, HIGH)`; wird dann die Laufzeit des Ultraschalls (in Mikrosekunden) in der `long`-Variablen `timePassed` gespeichert. Die Umrechnung der Laufzeit in die Entfernung erfolgt durch Multiplikation mit der Schallgeschwindigkeit in Luft. Diese beträgt

(etwa) 343,2 Meter pro Sekunde,
also 34320 Zentimeter pro Sekunde
oder 0,03432 Zentimeter pro Mikrosekunde.

Da die Laufzeit aber der Zeit entspricht, die die Ultraschallwelle zum Messobjekt und wieder zurück benötigt hat, wird das Ergebnis dieser Multiplikation noch durch Zwei dividiert.

Wiederverwendung von Befehlen und Fragen – Programmbibliotheken (engl.: Libraries):

Bei den Arbeitsanregungen zum Erfassen und Auswerten von Sensor-Messwerten (Aufgaben 1) und 2) der Datei [ST_AA_11Arduino_Praxis_Meswerterfassung](#) werden immer wieder die selbst programmierten Fragen zur Berechnung statistischer Maßzahlen verwendet. Bislang mussten dazu diese selbst programmierten Fragen in jedem neuen Programmierprojekt codiert werden – der Code wurde dadurch lang und unübersichtlich.

Eine elegantere Möglichkeit der Wiederverwendung bereits codierter Programmteile bieten sogenannte **Programmbibliotheken**: In einer Programmbibliothek wird der **Code von** (selbst programmierten) **Befehlen oder Fragen gesammelt** – diese Befehle oder Fragen stehen dann in jedem neuen Programmierprojekt zur Verfügung, in das die betreffende Programmbibliothek eingebunden (wir sagen auch: importiert) wird. Was beim Erstellen einer einfachen Programmbibliothek an Grundlegendem zu beachten ist, wird nachfolgend am Beispiel der Lösung zu Aufgabe 6) der Datei [ST_AA_06Arduino_Modularisierung](#) gezeigt. Dort hast du Befehle zum Morsen der Buchstaben 'D', 'A', 'S', 'P' und 'T' codiert, damit der Satz „Das passt“ komfortabel gemorst werden kann. Der Lösungscode dafür kann zum Beispiel folgendermaßen aussehen:

```
int ledPin = 10;
int charPin = 5;
int wordPin = 3;
```

```
void shortSignal(int pin){
  digitalWrite(pin,HIGH);
  delay(200);
  digitalWrite(pin,LOW);
  delay(200);
}

void longSignal(int pin){
  digitalWrite(pin,HIGH);
  delay(600);
  digitalWrite(pin,LOW);
  delay(200);
}

void letterD(int pin){
  longSignal(pin);
  shortSignal(pin);
  shortSignal(pin);
}

void letterA(int pin){
  shortSignal(pin);
  longSignal(pin);
}

void letterS(int pin){
  shortSignal(pin);
  shortSignal(pin);
  shortSignal(pin);
}

void letterP(int pin){
  shortSignal(pin);
  longSignal(pin);
  longSignal(pin);
  shortSignal(pin);
}


void letterT(int pin){
  longSignal(pin);
}

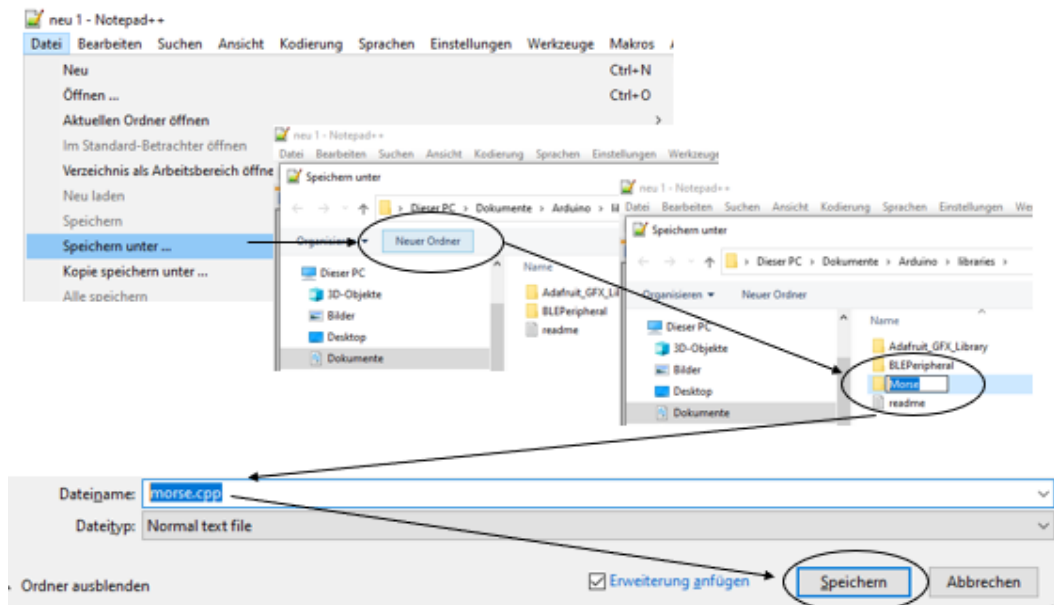
void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(charPin, OUTPUT);
  pinMode(wordPin, OUTPUT);
  digitalWrite(ledPin,LOW);
  digitalWrite(charPin,LOW);
  digitalWrite(wordPin,LOW);
}

void loop() {
  letterD(ledPin);
  digitalWrite(charPin,HIGH);
  delay(600);
  digitalWrite(charPin,LOW);
  delay(200);
  // a
  letterA(ledPin);
  digitalWrite(charPin,HIGH);
  delay(600);
  digitalWrite(charPin,LOW);
  delay(200);
  // s
  letterS(ledPin);
  digitalWrite(wordPin,HIGH);
  delay(600);
  digitalWrite(wordPin,LOW);
  delay(5000);
  // p
  letterP(ledPin);
  digitalWrite(charPin,HIGH);
  delay(600);
  digitalWrite(charPin,LOW);
  delay(200);
  // a
  // etc.
}
```

Um die mit dem Rechteck gekennzeichneten Befehle in einer **Programmbibliothek namens morse.cpp** zusammenzufassen, kopieren wir deren Code und fügen ihn in eine neue Datei namens morse.cpp ein. Da wir diese Datei nicht in der gewohnten Arduino-Programmierungsumgebung erzeugen und bearbeiten können, verwenden wir stattdessen das Editor-Programm Notepad++, das auf deinem Computer installiert ist.

Wie dabei vorzugehen ist, wird dir anhand einer Folienpräsentation gezeigt. – zum Nachlesen sind die wichtigsten Schritte nachfolgend zusammengefasst:

- Schritt:** Kopieren des benötigten Codes (siehe vorhergehende Seite) aus der Arduino-Projektdatei.
- Schritt:** Starten des Editorprogramms Notepad++ durch Doppelklick mit der linken Maustaste auf das Desktop-Symbol:
 
- Schritt:** Speichern der neuen Datei unter dem Namen **morse.cpp** in einem Unterordner des Dateionders für Arduino-Programmibliotheken:



- Schritt:** Einfügen des zuvor codierten Codes und Hinzufügen der grau unterlegten Codezeilen:

```

9
10 #include "Arduino.h"
11 #include "morse.h"
12
13 void shortSignal(int pin) {
14     digitalWrite(pin, HIGH);
15     delay(200);
16     digitalWrite(pin, LOW);
17     delay(200);
18 }
19
20 void longSignal(int pin) {
21     digitalWrite(pin, HIGH);

```

...hernach: Erneutes Speichern der Datei!

5. Schritt: Erstellen und Codieren der Header-Datei **morse.h** im Editor Notepad++:

```
morse.h x
1
2 #ifndef morse_h
3 #define morse_h
4
5 #include "Arduino.h"
6
7 // function prototypes
8
9 void letterA(int pin);
10 void letterD(int pin);
11 void letterP(int pin);
12 void letterS(int pin);
13 void letterT(int pin);
14
15 #endif
```

...hernach: Speichern der Datei im selben Ordner wie die Programmbibliothek **morse.cpp**!

6. Schritt: Einbinden der Programmbibliothek **morse** in ein neues Arduino-Programmierprojekt:

```
#include <morse.h>
//using a single LED to morse Das passt

int ledPin = 10;
int charPin = 5;
int wordPin = 3;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(charPin, OUTPUT);
  pinMode(wordPin, OUTPUT);
  digitalWrite(ledPin, LOW);
  digitalWrite(charPin, LOW);
  digitalWrite(wordPin, LOW);
}

void loop() {
  // D
  letterD(ledPin);
  digitalWrite(charPin, HIGH);
  delay(600);
  digitalWrite(charPin, LOW);
  delay(200);
  // a
  letterA(ledPin);
  digitalWrite(charPin, HIGH);
```

...hernach: Übersetzen/Kompilieren des Programms => die Programmbibliothek wird mitübersetzt...

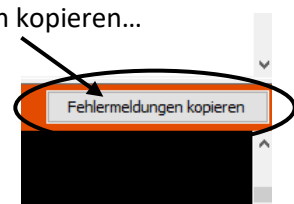
7. Schritt, im Fall von Fehlern in der Programmbibliothek:

```
letterA(ledPin);
digitalWrite(charPin, HIGH);
delay(600);

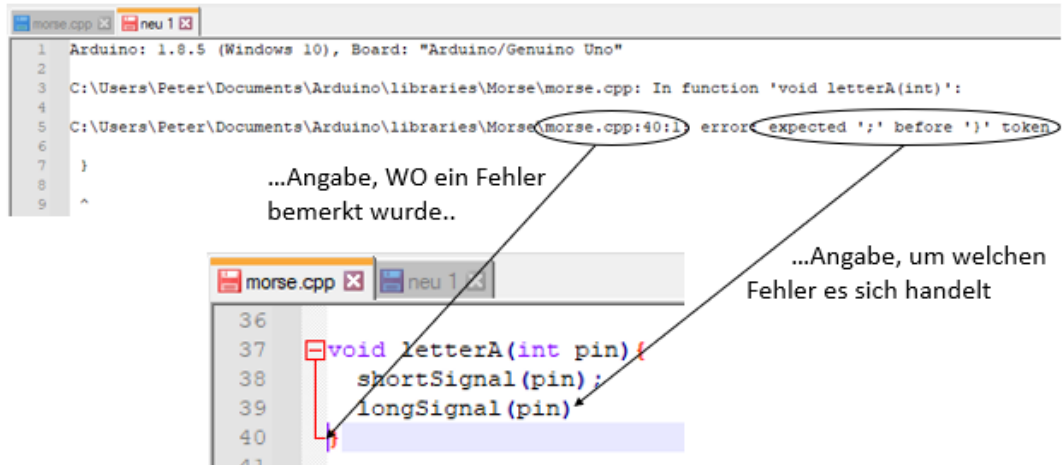
Fehler beim Kompilieren für das Board Arduino/Genuino Uno.

exit status 1
Fehler beim Kompilieren für das Board Arduino/Genuino Uno.
```

...detaillierte Fehlermeldungen kopieren...



...und in eine neue Datei im Notepad++ Editor einfügen:



The screenshot shows two windows of the Notepad++ editor. The top window displays the beginning of a C++ file named 'morse.cpp'. The code includes an Arduino IDE header and a function definition for 'letterA'. An error message is shown: 'C:\Users\Peter\Documents\Arduino\libraries\Morse\morse.cpp:40:1 error: expected ';' before ')' token'. Two annotations with arrows point to the error message: one points to the location 'morse.cpp:40:1' with the text '...Angabe, WO ein Fehler bemerkt wurde..' and the other points to the error text 'error: expected ';' before ')' token' with the text '...Angabe, um welchen Fehler es sich handelt'. The bottom window shows a zoomed-in view of the function definition: 'void letterA(int pin) { shortSignal (pin); longSignal (pin)'. A red cursor is positioned at the end of the function body, and a red arrow points from the error message in the top window to this location.

...so sollten auch in Programmibibliotheken die meisten Codierungsfehler rasch behoben werden können!