Arduino verstehen und anwenden

Teil 5: Nutzung und Erstellung von Programmbibliotheken

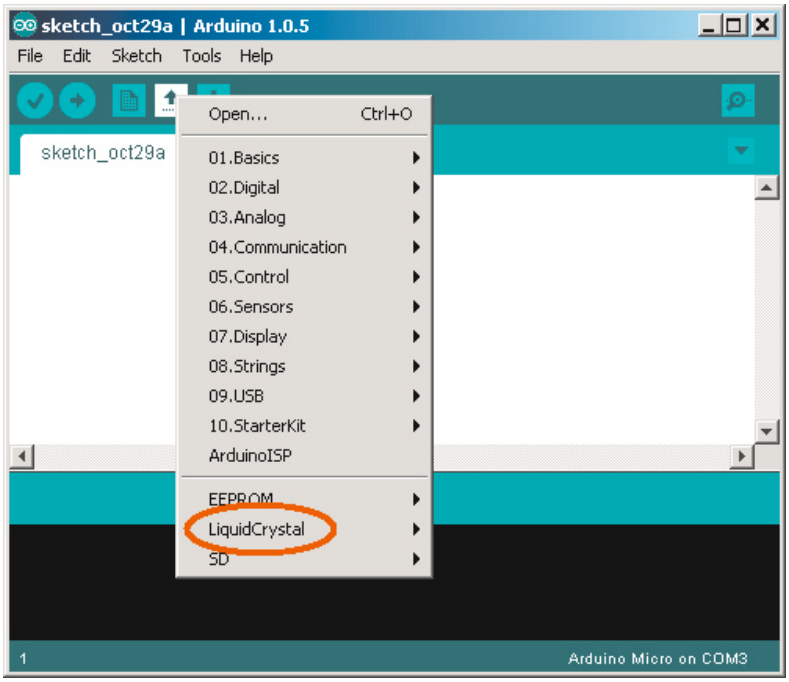


Bild 1: „LiquidCrystal" in der Arduino-IDE

Der fünfte Teil der Artikelserie „Arduino verstehen und anwenden" dreht sich um das Thema Programmbibliotheken.

Sogenannte Bibliotheken oder Libraries gehören zu den besonderen Stärken der Arduino-Idee. Diese Programmierhilfen erleichtern das Erstellen komplexerer Projekte ganz erheblich. So muss man nicht immer das Rad neu erfinden, sondern man kann auf bereits vorhandenen Grundlagen aufbauen.

Zu den sogenannten Standardbibliotheken zählen u. a:

- EEPROM – Schreiben und Lesen des EEPROM-Speichers im Controllerchip
- LiquidCrystal – Ansteuerung eines textbasierten LC-Displays
- SD – Schreiben und Lesen einer SD-Speicherkarte (erfordert ein entsprechendes Shield)
- Servo – Ansteuerung von Modellbauservos
- SoftwareSerial – ermöglicht die Nutzung einer softwarebasierten seriellen Schnittstelle an einem beliebigen Digitalpin des Arduinos
- Stepper – Steuerung von Schrittmotoren (erfordert passende Treiberbausteine)
- Wire – ermöglicht die Kommunikation mit Two-Wire- bzw. I²C-Interface-Bausteinen und -Sensoren wie z. B. DACs oder Echtzeituhren-ICs

Eine der besonders häufig genutzten Bibliotheken ist die LiquidCrystal-Library. Sie ermöglicht das einfache Ansteuern eines LC-Displays. Damit kann der Arduino auch unabhängig vom PC Zahlen und Texte ausgeben. Hochinteressanten praktischen Anwendungen wie etwa einer Digitaluhr oder einem digitalen Innen-und-Außenthermometer mit LCD-Anzeige steht damit nichts mehr im Weg. Diese nützliche Bibliothek soll daher im Folgenden als einführendes Beispiel dienen.

Einbinden einer Bibliothek in einen Arduino-Sketch

Die LCD-Bibliothek wird bereits mit der Standard-Arduino-IDE mitgeliefert. Sie arbeitet mit allen LC-Displays, die mit dem bekannten HD44780-Treiber von Hitachi kompatibel sind. Die Anschlusskonfiguration dieser Module hat sich als Quasi-Standard durchgesetzt. Praktisch alle gängigen Anzeigeeinheiten mit 14 oder 16 Pins und 1 oder 2 Zeilen à 16 Zeichen sind somit mit der hier vorgestellten Bibliothek verwendbar.

Bild 1 zeigt, wie die Beispielprogramme der Bibliothek „LiquidCrystal" im Hauptmenü der IDE ausgewählt werden können.

Im zugehörigen Untermenü finden sich dann verschiedene Beispielprogramme dieser Library.

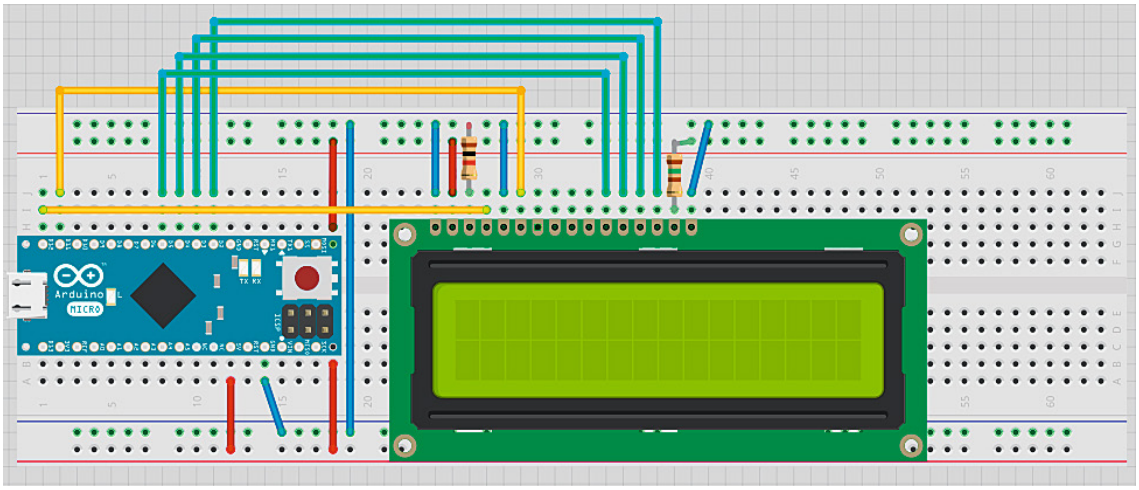


Bild 2: Anschluss eines LC-Displays an den Arduino Micro

Bild 3: Die LCD-Bibliothek in Aktion

Zunächst muss aber natürlich das LC-Display mit dem Arduino verbunden werden. Bild 2 zeigt das zugehörige Aufbaubild. Eine praktische Ausführung des Aufbaus ist in Bild 3 zu sehen.

Nachfolgend sind die notwendigen Verbindungen der Deutlichkeit halber nochmals in Tabellenform angegeben:

LCD RS	-	Digitalpin 12
LCD Enable	-	Digitalpin 11
LCD D4	-	Digitalpin 5
LCD D5	-	Digitalpin 4
LCD D6	-	Digitalpin 3
LCD D7	-	Digitalpin 2
LCD R/W	-	Ground

Anwendung der Bibliothek LiquidCrystal.h

Unter Verwendung der Bibliothek LiquidCrystal.h kann das zugehörige Hauptprogramm sehr einfach ausfallen:

```
// Hallo Arduino MICRO

#include <LiquidCrystal.h>


LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{ lcd.begin(16, 2);
  lcd.setCursor(5, 0);
  lcd.print("Hallo, ");
  lcd.setCursor(0, 1);
  lcd.print(" Arduino MICRO!  ");
}

void loop() {}
```

Nach dem Einbinden der Bibliothek mit

```
#include <LiquidCrystal.h>
```

müssen also nur noch die Pin-Verbindungen zum LCD angegeben werden:

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

Nach der Initialisierung mit

```
lcd.begin(16, 2);
```

entsprechend 16 Zeichen in 2 Zeilen, kann im Setup bereits direkt auf das LC-Display geschrieben werden.

Mit

```
lcd.setCursor(5, 0);
```

wird die aktuelle Schreibposition auf das 5. Zeichen in der 1. Zeile gesetzt (die Zeilenzählung beginnt wie in der IT üblich mit 0).

Dann erfolgt die Ausgabe des Textes einfach mit

```
lcd.print("Hallo, ");
```

und

```
lcd.print(" Arduino MICRO!  ");
```

Hier zeigt sich deutlich, wie die Verwendung einer Bibliothek die Programmierung ganz wesentlich vereinfacht. Nur wenige Programmzeilen genügen, um ein alphanumerisches Display anzusteuern. Wollte man diese Aufgabe ohne die Hilfe der Bibliotheksfunktionen lösen, so würde daraus ein Projekt, das auch einen erfahrenen Programmierer viele Stunden lang beschäftigen würde.

Wenn man sich einen ersten Eindruck verschaffen möchte, wie die Ansteuerung eines LC-Displays im Detail aussieht, dann kann man die Datei „LiquidCrystal.cpp" im Verzeichnis …Arduino\arduino-1.0.5\libraries\LiquidCrystal mit einem Editor, wie etwa dem Windows-WordPad, öffnen. Hier zeigt sich, welcher Aufwand durch den Einsatz der fertigen Bibliothek eingespart werden konnte.

Erstellen eigener Bibliotheken

Fertige Bibliotheken sind aber natürlich nur die eine Seite der Medaille. Während der Einsteiger sicherlich überwiegend bereits vorgefertigte Libraries nutzt, wird der fortgeschrittene Anwender irgendwann den Punkt erreichen, an welchem eine eigene Bibliothek erstellt werden soll. Nachfolgend werden daher die Schritte zur Erstellung eigener Bibliotheken vorgestellt.

Bereits im 2. Teil zu dieser Beitragsreihe wurde ein einfaches Programm zum Senden des SOS-Notfallsignals im Morsecode entwickelt. Hier soll dieses Beispiel noch einmal aufgegriffen werden, um die Erstellung einer Programmbibliothek zu erläutern.

Da alle erforderlichen Programmstrukturen und Befehle aus dem 3. und 4. Teil dieser Artikelserie bekannt sind, kann das SOS-Programm nun schon deutlich kompakter und eleganter aufgebaut werden als noch im früheren Beitrag:

```
// SOS

int pin = 13;

void setup()
{ pinMode(pin, OUTPUT);
}

void loop()
{ S();O();S(); delay(1000);}

void S()
{ for(int i = 1; i <= 3; i++)
  { digitalWrite(pin, HIGH); delay(200);
    digitalWrite(pin, LOW);  delay(200);
  }
  delay(500);
}

void O()
{ for(int i = 1; i <= 3; i++)
  {  digitalWrite(pin, HIGH); delay(1000);
     digitalWrite(pin, LOW);  delay(200);
  }
  delay(500);
}
```

SOS_short.ino

Nun sollen verschiedene Teile des Sketches in eine Library ausgegliedert werden. Hierfür bieten sich natürlich die beiden Routinen S() und O() an. Diese beiden Programmteile übernehmen die Aufgabe, die Morsesequenzen für die Buchstaben S und O zu erzeugen. Nachdem die Prozeduren über

void S() und void O()

definiert wurden, können sie über die Anweisungen S() und O() aufgerufen werden.

Die Bibliothek soll aber auch noch eine einfache Möglichkeit bieten, den Port, an welchen das Signal ausgegeben werden soll, zu wählen.

Aufbau einer Bibliothek

Eine Library muss immer mindestens zwei Dateien enthalten:

1. das Header-File mit der Extension .h
2. das Source-File mit der Extension .cpp

Im Header-File werden alle für die Library erforderlichen Definitionen festgelegt. Hierfür wird zunächst eine sogenannte Class (engl. für „Klasse") benötigt. In einer Klasse können mehrere Funktionen und Variablen zusammengefasst werden. Hier hat man zwei Möglichkeiten zu unterscheiden:

Public Class („öffentliche Klasse") und

Private Class („private Klasse")

Die als öffentlich definierten Teile können von allen Nutzern der Bibliothek verwendet werden. Auf private Teile kann nur innerhalb der Klasse zugegriffen werden. Jede Klasse verfügt über eine spezielle Funktion, den sogenannten Constructor. Dieser wird benutzt, um jeweils eine Instanz einer Klasse zu erzeugen. Der Constructor hat den gleichen Namen wie die zugehörige Klasse und gibt kein Argument zurück.

In einen klassischen Sketch wird das Standard-File Arduino.h immer automatisch mit eingebunden. Bei einer Bibliotheksdatei dagegen muss man dies manuell erledigen, um sicherzustellen, dass alle erforderlichen arduinospezifischen Definitionen zur Verfügung stehen.

Um Probleme mit der Mehrfacheinbindung einer Bibliothek zu vermeiden, wird der gesamte Code in eine entsprechende if-Abfrage eingebettet. Damit muss die Datei MorseCode.h wie folgt aussehen:

```
// morse code generation

#ifndef MorseCode_h
#define MorseCode_h

#include "Arduino.h"

class Morse
{ public:
    Morse(int pin);
    void S();
    void O();
  private:
    int _pin;
};

#endif
```

MorseCode.h

Nun muss noch die zugehörige .ccp-Datei erstellt werden. Hier muss natürlich zunächst die zugehörige .h-Datei eingebunden werden. Auch die Arduino.h ist nochmals manuell zu aktivieren. Dann folgt die Definition der einzelnen Funktionen. Um festzulegen, dass diese Teil einer Klasse sind, muss man den Namen der Klasse gefolgt von einem doppelten Doppelpunkt voranstellen (::). Damit sieht die ccp.-Datei so aus:

```
// morse code generation

#include "Arduino.h"
#include "MorseCode.h"

Morse::Morse(int pin)
{ pinMode(pin, OUTPUT);
  _pin = pin;
}

void Morse::S()
{ for(int i = 1; i <= 3; i++)
  { digitalWrite(_pin, HIGH); delay(200);
    digitalWrite(_pin, LOW);  delay(200);
  }
```

```
  delay(500);
}

void Morse::O()
{ for(int i = 1; i <= 3; i++)
  {  digitalWrite(_pin, HIGH); delay(1000);
     digitalWrite(_pin, LOW);  delay(200);
  }
  delay(500);
}
```

MorseCode.cpp

Mit diesen beiden Dateien ist die Bibliothek bereits fertiggestellt.

Das SOS-Programm wird damit unschlagbar einfach:

```
// SOS

#include <MorseCode.h>
Morse morse(13);
void setup(){}
void loop()
{ morse.S(); morse.O(); morse.S(); delay(1000);
}
```

SOS_w_lib.ino

Man muss also nur noch die Bibliothek einbinden, und die darin enthaltenen Funktionen stehen für die freie Verwendung zur Verfügung.

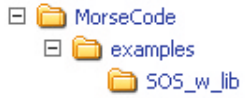
Der Arduino hat nun sozusagen das Morsealphabet gelernt und sendet beispielsweise auf den Befehl morse.S(); automatisch drei kurze Lichtimpulse, also das optische Morsezeichen für „S" aus. Natürlich kann man auch den Rest des Alphabets in die Bibliothek integrieren und ist dann in der Lage, sehr komfortabel ganze Texte in Morsecode mit dem Arduino zu versenden.

Abschließend sei noch auf die Datei keywords.txt hingewiesen. Diese ist für die Funktion der Bibliothek nicht unbedingt erforderlich, bietet aber den Komfort, dass die in der Bibliothek verwendeten Schlüsselwörter in der gewohnten Farbcodierung erscheinen. Hierzu muss man die verwendeten Funktionsnamen in eine keywords.txt-Datei eintragen und mit „KEYWORD1" für die Klasse bzw. „KEYWORD2" für die Funktionen kennzeichnen:

```
Morse  KEYWORD1
S      KEYWORD2
O      KEYWORD2
```

keywords.txt

Will man nun die erzeugte Bibliothek der Allgemeinheit zur Verfügung stellen, so muss man nur die drei Dateien (xxx.h, xxx.ccp und xxx.txt) weitergeben. Zusätzlich wird meist noch eine Beispieldatei mitgeliefert, die die Anwendung der Bibliotheksfunktionen demonstriert. Die Bibliothek besteht also aus einem Verzeichnis „Morsecode".



In diesem Verzeichnis müssen sich die Dateien

MorseCode.h
MorseCode.cpp
keywords.txt

befinden.

Zusätzlich sollte noch ein Unterverzeichnis „examples" angelegt werden, in welches die Beispieldatei

SOS_w_lib.ino

abgelegt wird. Damit ist die Bibliothek prinzipiell vollständig.

Ideal ist es natürlich, wenn dazu noch weitere Informationen geliefert werden, in welchen alle Funktionen genau dokumentiert sind. Die Erfahrung zeigt allerdings, dass diese Dokumentation bei den meisten im Internet veröffentlichten Libraries fehlt. Trotzdem können viele dieser kleinen Helfer nach kurzer Einarbeitung und dem Studium der einzelnen Dateien, insbesondere der Beispieldatei, sehr nutzbringend angewendet werden.

Ausblick

Die Arbeit mit Programmbibliotheken sollte nun keine Probleme mehr bereiten. Der Einsatz dieser nützlichen Hilfsmittel ist immer dann zu empfehlen, wenn es darum geht, Standardprobleme zu lösen.

Es ist in jedem Fall lohnend, vor der Implementierung eines neuen Projekts zu prüfen, ob entweder für das gesamte Vorhaben oder aber auch nur für wichtige Teile geeignete Bibliotheken existieren.

Der nächste Artikel wird sich um das Thema „Sensortechnik und Messwerterfassung" drehen. Auch dazu sind umfangreiche Bibliotheken vorhanden. Allerdings erfordert der Einsatz spezieller thermischer oder optoelektronischer Sensoren, etwa im Bereich der Hausautomatisierung oder der Robotersteuerung, doch auch entsprechende Kenntnisse im Bereich der Erfassung und Wandlung analoger Daten.

Empfohlenes Material

- Arduino Micro, Best.-Nr. J4-10 97 74, € 24,95
- Mikrocontroller-Onlinekurs, Best.-Nr. J4-10 20 44, € 99,–
- Auch viele Lernpakete von Franzis wie etwa „Elektronik mit ICs" enthalten Materialien wie ein lötfreies Steckbrett, Widerstände und LEDs etc., die für den Aufbau von Schaltungen mit dem Arduino Micro gut geeignet sind.

Preisstellung Juni 2014 – aktuelle Preise im Web-Shop

Weitere Infos:

- G. Spanner: Arduino – Schaltungsprojekte für Profis, Elektor-Verlag, 2012, Best.-Nr. J4-10 94 45, € 39,80
- AVR-Mikrocontroller in C programmieren, Franzis-Verlag, 2012, Best.-Nr. J4-09 73 52, € 39,95

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