Leistungskurs Elektrotechnik Q3 und Q4

Kommentare

```
// einzeiliger Kommentar

/*
   mehrzeiliger
   Kommentar
*/
```

Elementare Datentypen

Typ	Größe in Bit	Wertebereich
`int`	16	Ganzzahl (Integer): -32.768 bis 32.767
`long`	32	Ganzzahl: -2.147.483.648 bis 2.147.483.647
`float`	32	Fließkommazahl: -3.4028235E+38 bis 3.4028235E+38
`double`	64	Fließkommazahl mit doppelter Genauigkeit
`boolean`	8	logischer Wert (Wahrheitswert): `true` oder `false`
`char`	8	Zeichen (ASCII-Code)

Deklaration von Variablen und Konstanten, Wertzuweisung (Beispiele)

```
int a;      // Die Variable a wird deklariert, sie ist vom Typ int.
a = 5;      // Der Variablen a wird der Wert 5 zugewiesen.

//Die Variable b wird deklariert, ihr wird der Wert 0.815 zugewiesen.
double b = 0.815;

//Die Konstante g wird deklariert, ihr wird der Wert 9.81 zugewiesen.
const double g = 9.81;
```

Häufig sind sehr viele Bauelemente (Sensoren, Aktoren) an den Pins eines Mikrocontrollers angeschlossen. Für einen gut lesbaren Programmcode ist es dann wichtig, dass die Nummern der Pins durch aussagekräftige Bezeichner ersetzt werden. Beispiel:

```
//An Pin 9 ist ein Taster angeschlossen.
//Der Zugriff auf diesen Pin ist jetzt über den Bezeichner taster möglich.
const int taster = 9;
```

Eindimensionale Arrays (Beispiel)

```
// Das Array meineLottozahlen wird deklariert.
// Es kann 6 Werte vom Datentyp int speichern (Index 0 bis 5).
int meineLottozahlen[6];

//Im Array meineLottozahlen wird unter dem Index 0 die Zahl 8 gespeichert.
meineLottozahlen[0] = 8;
```

Strings

String-Variablen speichern Zeichenketten und sind keine elementaren Datentypen. Sie können aber ähnlich verwendet werden. Beispiel:

```
//Die Variable str wird deklariert und mit "Guten Tag" gefüllt.
String str = "Guten Tag";
```

Mathematische Operatoren

Operator	Bedeutung	Beispiel
+	Addition	`y = 3 + 4`
-	Subtraktion	`m = 7 - x`
*	Multiplikation	`n = a * b`
/	Division	`y = 10 / 2`
%	Modulo (liefert ganzzahligen Rest einer Division)	`a = 20 % 7`
++	Inkrement	`i++` (i wird um 1 erhöht)
--	Dekrement	`i--` (i wird um 1 verringert)

Vergleichsoperatoren

Operator	Bedeutung	Beispiel
==	gleich	`x == 5`
!=	ungleich	`x != 5`
<	kleiner als	`x < 5`
>	größer als	`x > 5`
<=	kleiner als oder gleich	`x <= 5`
>=	größer als oder gleich	`x >= 5`

Logische Operatoren

Operator	Bedeutung	Beispiel
!	Negation	`!(a > b)`
&&	logisches UND	`(a > b) && (c < 5)`
||	logisches ODER	`(a > b) || (c < 5)`

Kontrollstrukturen

Ja/Nein-Abfrage (Beispiel)

```
if(a < 5)
{
      // Anweisungen im "Ja-Block"
}
      else
      {
            // Anweisungen im "Nein-Block"
      }
```

Falls die Bedingung a < 5 wahr ist, werden die Anweisungen im "Ja-Block" ausgeführt. Falls nicht, werden die Anweisungen im "Nein-Block" ausgeführt.

Kopfgesteuerte Schleife (Beispiel)

```
while(a < 5)
{
      // Anweisungen im Schleifenkörper
}
```

Zuerst wird geprüft, ob die Bedingung a < 5 wahr ist. Falls ja, werden die Anweisungen im Schleifenkörper ausgeführt. Dieser Vorgang wiederholt sich und endet erst dann, wenn die Bedingung a < 5 nicht mehr wahr ist.

Fußgesteuerte Schleife (Beispiel)

```
do
{
      // Anweisungen im Schleifenkörper
}
while(a < 5);
```

Zuerst werden die Anweisungen im Schleifenkörper ausgeführt. Danach wird geprüft, ob die Bedingung a < 5 wahr ist.
Dieser Vorgang wiederholt sich und endet erst dann, wenn die Bedingung a < 5 nicht mehr wahr ist.

Zählschleife (Beispiel)

```
for(int i = 0; i <= 2; i++)
{
      // Anweisungen im Schleifenkörper
}
```

Die Anweisungen im Schleifenkörper werden dreimal (für i = 0, i = 1 und i = 2) ausgeführt.

Schleifen vorzeitig beenden

Alle Schleifen können durch den Befehl `break` vorzeitig beendet werden.

Switch-Case-Anweisung

```
switch (var) {
      case label1:
        // Anweisung 1
        break;
      case label2:
        // Anweisung 2
        break;
      default:
        // Anweisung 3
        break;
    }
```

Switch vergleicht den Inhalt der Variablen `var` mit den Werten label1, label2, label3 und führt dann den dazugehörigen Anweisungsblock 1, 2 oder 3 aus. Der Befehl `break` beendet die Switch-Case-Anweisung.

Wichtige Funktionen

pinMode()

Konfiguriert den Pin als Ein- oder Ausgang. Beispiele:

```
const int roteLED = 5;      //an Pin 5 ist eine rote LED angeschlossen
pinMode(roteLED, OUTPUT);   //Pin 5 als Ausgang setzen
const int taster = 9;       //an Pin 9 ist ein Taster angeschlossen
pinMode(taster, INPUT);     //Pin 9 als Eingang setzen
```

digitalRead()

Liest den Zustand des Pins entweder HIGH (1, true) oder LOW (0, false) ein und gibt diesen als Wert 0 oder 1 zurück. Beispiel:

```
const int taster = 7;
int wert = digitalRead(taster); // liest von Pin 7 digital ein
```

digitalWrite()

Den Pin auf logisch HIGH oder LOW setzen.

Beispiel: `digitalWrite(4, HIGH); // setzt den digitalen Pin 4 auf HIGH`

analogRead()

Liest vom angegebenen Analogeingang ein.

Beispiel: `int wert = analogRead(A0);  // liest Pin A0 analog ein`

analogWrite()

Gibt ein PWM-Signal am definierten Pin aus. Beispiel:

```
analogWrite(3, 120); //PWM-Signal am Pin 3 mit Pulsweite 120
```

delay()

Stoppt die Ausführung des Programms für eine bestimmte Zeit.

Beispiel: `delay(500); // pausiert für 500 Millisekunden`

delayMicroseconds()

Stoppt die Ausführung des Programms für eine bestimmte Zeit.

Beispiel: `delayMicroseconds(100); // pausiert für 100 Mikrosekunden`

millis()

Gibt die Zeitdauer in Millisekunden zurück, welche seit dem Start des Mikrocontrollers vergangen ist. Beispiel:

```
long t1 = millis();
  // beliebiger Code
long t2 = millis();
```

Die Differenz `t2-t1` entspricht der Zeit in Millisekunden, welche zwischen den beiden Aufrufen der Funktion `millis()` vergangen ist.

Befehle zum Verwenden eines Servos

#include <Servo.h>

Bindet die Servo-Bibliothek in den Quellcode ein.

Servo einServo;

Erstellt ein Servo-Objekt mit dem Namen einServo.

einServo.attach()

Konfiguration eines Pins zum Anschluss eines Servos.

Beispiel: `einServo.attach(9); // Servo einServo ist an Pin 9 angeschlossen`

einServo.write()

Fährt den Servo auf einen bestimmten Winkel.

Beispiel: `einServo.write(45); // Servo einServo fährt auf die 45°-Position`

Befehle zum Verwenden des seriellen Monitors

`Serial.begin()`
Der serielle Monitor wird geöffnet und die Geschwindigkeit der Datenübertragung in bit pro Sekunde wird festgelegt. Beispiel: `Serial.begin(9600);`

`Serial.print()`

Text sowie Inhalte von Variablen werden ohne Zeilenumbruch auf dem seriellen Monitor ausgegeben.
 	
Beispiel: Ausgabe von einem Text und dem Inhalt der variablen Zahl.
`Serial.print("Text");`
`Serial.print(zahl);`

`Serial.println()`
Identisch mit dem Befehl `Serial.print()`, jedoch folgt nach der Ausgabe ein Zeilenumbruch.

Befehle für das I2C-LCD-Display

`#include <LiquidCrystal_I2C.h>` **`#include <Wire.h>`**
Bindet die nötigen Bibliotheken in den Quellcode ein.

`LiquidCrystal_I2C lcd(0x27, 16, 2);`
Erstellt ein LCD Objekt mit dem Namen `lcd` für ein Display mit der Adresse 0x27, welches 2 Zeilen mit jeweils 16 Zeichen darstellen kann.

`lcd.init()`
Das Display wird initialisiert.

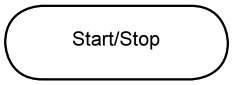
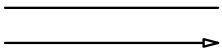
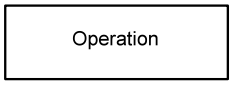
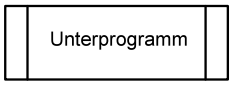
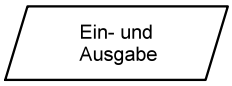
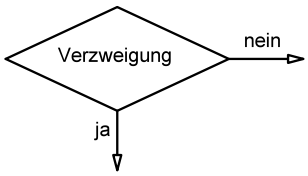
`lcd.clear()`
Der auf dem Display dargestellte Inhalt wird gelöscht.

`lcd.backlight()`
Die Hintergrundbeleuchtung des Displays wird eingeschaltet.

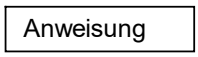
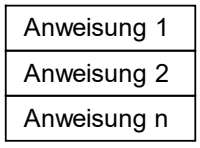
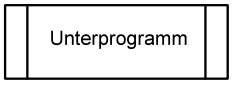
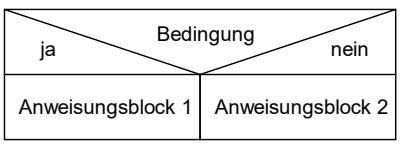
`lcd.setCursor(x,y)`
Die Stelle, an der die nächsten Symbole geschrieben werden, wird auf Zeichen x und Zeile y festgelegt. Beispiel: Setzen des Cursors auf das erste Zeichen der ersten Zeile. `lcd.setCursor(0,0);`

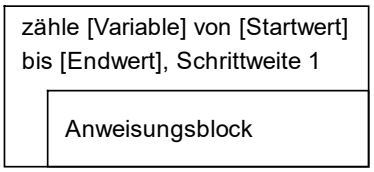
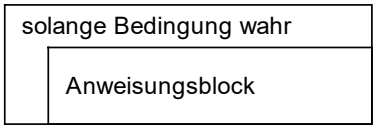
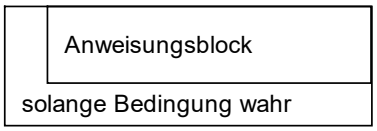
`lcd.print()`
Auf dem Display können Werte und Zeichenketten ausgegeben werden. Beispiel: Ausgabe der Zeichenkette Text. `lcd.print(„Text“);`

Symbole für einen Programmablaufplan (DIN 66001)

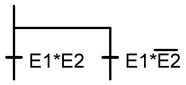
Symbol	Bezeichnung	Bemerkung
	Start/Stop	Markiert Start und Ende des Programmablaufplans.
	Verbindung	Markiert Verbindungen zwischen einzelnen Elementen.
	Operation	Befehle, wie z.B. Rechenoperationen und Zuweisungen.
	Unterprogramm	Aufruf eines Unterprogramms bzw. einer Funktion oder einer Prozedur.
	Ein- und Ausgabe	Befehle, die Ein- oder Ausgaben erzeugen.
	Verzweigung	Prüfen einer (Schleifen-) Bedingung.

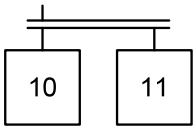
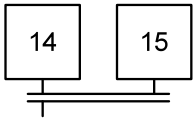
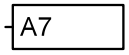
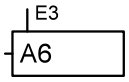
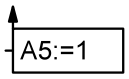
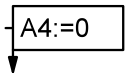
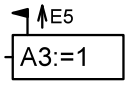
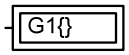
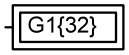
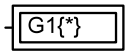
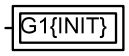
Symbole für ein Struktogramm (DIN 66261)

Symbol	Bezeichnung	Bemerkung
	Anweisung	
	Anweisungsblock	Die Anweisungen werden von oben nach unten durchlaufen.
	Unterprogramm	Aufruf eines Unterprogramms bzw. einer Funktion oder einer Prozedur.
	alternative Verzweigung	Wenn die Bedingung erfüllt ist, wird der Anweisungsblock 1 und andernfalls der Anweisungsblock 2 ausgeführt.

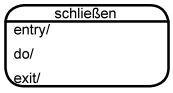
	zählergesteuerte Schleife (for-Schleife)	Nach jedem Durchlauf des Schleifenkörpers (Anweisungsblock) wird die Zählvariable um die Schrittweite inkrementiert. Die Schleife wird verlassen, wenn der Endwert überschritten wurde.
	kopfgesteuerte Schleife (while-Schleife)	Zuerst wird die Bedingung geprüft. Wenn die Bedingung erfüllt ist, wird der Schleifenkörper (Anweisungsblock) durchlaufen. Anschließend wird die Bedingung erneut geprüft.
	fußgesteuerte Schleife (do-while-Schleife)	Zuerst wird der Schleifenkörper (Anweisungsblock) durchlaufen und danach wird die Bedingung geprüft. Wenn sie erfüllt ist, wird der Schleifenkörper erneut durchlaufen.

Symbole Grafcet (DIN EN 60848)

Symbol	**Bezeichnung**	**Bemerkung**
	Initialschritt, Anfangsschritt	Wird beim Einschalten des Systems aktiviert.
	Schritt	Normaler Bearbeitungsschritt
	Wirkungslinie	Verbindet Schritte untereinander.
	Transition	Definiert die logische Bedingung (wahr oder falsch), so dass der nächste Schritt aktiv geschaltet wird.
	Zielverweis, Sprung	Aktiviert den angegebenen Schritt.
	Rückführung, Wirkungslinie nach oben gerichtet	Verbindet Grafcet-Elemente miteinander, die Wirkungsrichtung ist nach oben.
	Alternativ-Verzweigung	Die einzelnen Transitionen müssen sich gegeneinander ausschließen.

	Parallel-Verzweigung	Die angeschlossenen Schritte werden gleichzeitig aktiviert.
	Synchronisierung	Die nachfolgende Transition ist erst wirksam, wenn die vorherigen Schritte aktiv sind.
	Kontinuierliche Aktion	Schreib den Schrittzustand (Wahr) kontinuierlich in den Operanden.
	Kontinuierliche Aktion mit Bedingung	Schreibt in den Operanden Wahr, wenn der Schritt und die Bedingung Wahr sind. Ansonsten wird in den Operanden Falsch geschrieben.
	Speichernd wirkende Aktion bei Aktivierung	Der Operand (Beispiel Wahr) wird mit Beginn des Schrittes einmalig beschrieben.
	Speichernd wirkende Aktion bei Deaktivierung	Der Operand (Beispiel Falsch) wird mit Ende des Schrittes einmalig beschrieben.
	Speichernd wirkende Aktion bei einem Event	Der Operand (Beispiel Wahr) wird einmalig beschrieben, wenn der Schritt aktiv und anschließend das Event (z.B. positive Flanke E5) erfüllt ist.
	Zwangssteuerung leere Situation	Alle Schritte eines Teilgrafcet deaktivieren.
	Zwangssteuerung Situation	Bestimmte Schritte eines Teilgrafcet aktivieren und halten.
	Zwangssteuerung aktuelle Situation	Aktuellen Zustand der Schritte einfrieren bzw. halten.
	Zwangssteuerung Anfangssituation	Alle Schritte im Teilgrafcet deaktivieren. Alle Initialschritte im Teilgrafcet aktivieren.

Symbole UML Zustandsdiagramm

Symbol	Bezeichnung	Bemerkung
	Startzustand	Pseudo-Zustand. Besitzt keine eingehenden Transitionen, sondern nur eine Ausgangstransition, die in den Startzustand führt.
	Endzustand	Pseudo-Zustand. Besitzt nur eine eingehende Transition und gibt an, dass ein Bereich beendet wurde.
	Transition	Verbindet die einzelnen Zustände miteinander. Transitionen enthalten Bedingungen, die für das Weiterschalten in einen anderen Zustand verantwortlich sind.
	Vereinigung / Gabelung	Aufspaltung oder Vereinigung von mehreren parallelen Zuständen.
	Kreuzung / Entscheidung	Punkte (Knoten), von denen mehrere Transitionen ausgehen oder zufließen können.
	Zustand allgemein	Allgemeiner Zustand. Die diesem Zustand zugeordneten Aktionen werden ausgeführt.
	Zustand mit zugeordnetem Verhalten	Zustand mit zugeordneten Verhaltensspezifikationen. Einmalige Aktionen die beim Betreten des Zustands ausgeführt werden (entry). Permanente Aktionen, die während des Zustands ausgeführt werden (do). Einmalige Aktionen, die beim Verlassen des Zustands ausgeführt werden (exit).