Bedingte Anweisungen (if-else)

Beschreibung

Nicht bei jeder Codedurchführung soll derselbe Programmcode durchgeführt werden. So soll ein Arduino beispielsweise abhängig von der Wahrnehmung eines Sensors eine Lampe leuchten lassen oder nicht. Je nachdem, was der Sensor misst, müssen also verschiedene Codeteile durchgeführt und andere ignoriert werden. Dazu stehen in der Sprache C++ „bedingte Anweisungen“ (engl. conditional statements) zur Verfügung. Das sind Codeabschnitte, deren Durchführung an bestimmte Bedingungen geknüpft sind. Eine bedingte Anweisung hat folgende Form:

```
if (bedingung) {
    anweisung;
}
```

Hierbei ist `bedingung` ein Ausdruck vom Typ `boolean`, also ein Ausdruck, welcher wahr (entspricht `true`) oder falsch (entspricht `false`) ist. Ein Beispiel-Codeausschnitt könnte wie folgt aussehen:

```
if (gemessenerAbstand < 10){
    digitalWrite(LED, HIGH);
}
```

Hier sind einige Beispiele für Ausdrücke, welche man als Bedingung verwenden könnte:

```
true                //true
false               //false
0 == 1              //false
1 < 5               //true
random(0, 2) == 1   //can be true or false
isDigit("a")        //false
```

Für die Schrägstriche `//` sei auf Kommentare verwiesen.

Es ist außerdem möglich, Zahlen, also Ausdrücke vom Typ `int`, als Bedingung zu verwenden. Hiervon ist allerdings abzuraten, da dies das Verständnis des Programmes erschwert.

`anweisung` ist Platzhalter für Code, welcher ausgeführt werden soll, falls `bedingung` wahr ist. Die geschweiften Klammern { } dienen einerseits dazu, mehrere Ausdrücke zusammenzufassen. Außerdem erleichtern sie das Verständnis des Codes, da genau klar ist, welcher Code ausgeführt wird, falls die Bedingung erfüllt ist. Auch wenn diese geschweiften Klammern nicht zwingend nötig sind, sofern man nur eine Anweisung ausführt, ist es eine gute Angewohntheit, die geschweiften Klammern **immer** zu verwenden. Der Code könnte bei mehreren Anweisungen wie folgt aussehen:

```
if (bedingung) {
    anweisung1;
    anweisung2;
}
```

Häufig möchten wir auch in dem Fall, dass `bedingung` falsch ist, etwas anderes ausführen. Dazu dient das Schlüsselwort `else`:

```
// Falls bedingung wahr ist, führe anweisung1 aus.
if (bedingung) {
```

```
    anweisung1;
//  Ansonsten führe anweisung2 aus.
} else {
    anweisung2;
}
```

Die `if-else`-Konstruktion wird häufig fälschlicherweise mit *wenn-dann* übersetzt. *wenn-dann* entspricht aber genau der Konstruktion

```
if (bedingung) {
    anweisung;
}
```

`else` lässt sich besser mit „ansonsten“ übersetzen.

Es ist möglich, nicht nur eine sondern beliebig viele Bedingungen zu prüfen. Hierzu verwenden wir `if-else if`: Die Bedingungen werden von oben nach unten geprüft und die zugehörige Anweisung der ersten wahren Bedingung wird ausgeführt.

```
if (bedingung1) {
    anweisung1;
} else if (bedingung2) {
    anweisung2;
} else {
    anweisung3;
}
```

Hier ist ein Beispiel-Programm, welches herausfindet, ob `"a"` eine Ziffer ist:

```
char myChar = "a";
if (isDigit(myChar)) {
  Serial.println("The character is a number.");
}
else {
  Serial.println("The character is not a number.");
}
```

Für *Serial.println* sei auf den seriellen Monitor verwiesen.

Vergleichsoperatoren

Die Programmiersprache C++ bietet die folgenden Operatoren, um Zahlen miteinander zu vergleichen.

```
a == b  //Sind a und b gleich?
a != b  //Sind a und b ungleich?
a < b   //Ist a kleiner als b?
a > b   //Ist a größer als b?
a <= b  //Ist a kleiner gleich b?
a >= b  //Ist a größer gleich b?
```

Verknüpfung von Bedingungen

In C++ gibt es logische Operatoren für „nicht“, „oder“ und „und“.

```
!       //nicht
&&      //und
||      //oder
```

Diese werden wie folgt verwendet:

```
!bedingung                  // Gilt bedingung nicht?
bedingung1 && bedingung2    // Gelten beide Bedingungen?
bedingung1 || bedingung2    // Gilt mindestens eine der beiden Bedingungen?
```

Code

```
int led = 12;

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

void loop() {
  // Lege den Abstand als Gleitkommazahl (Datentyp "float") 20 fest.
  // An diesem Punkt könnte man den Abstand auch messen, siehe
  // [Ultraschallsensor].
  float abstand = 20.0;

  // Falls der Abstand kleiner als 10 ist, schalte die LED ein.
  if (abstand < 10) {
    digitalWrite(led, HIGH);

  // Ansonsten schalte die LED aus.
  } else {
    digitalWrite(led, LOW);
  }
}
```

Aufgaben

(1) **Wenn die Ampel grün ist, fährt das Auto, ansonsten bleibt es stehen. Schreibe dies als bedingte Anweisung.**

```
int LED_GRUEN = 9;
int LED_GELB = 10;
int LED_ROT = 11;
int MOTOR_LINKS  = 12;
int MOTOR_RECHTS = 13;
if(digitalRead(LED_ROT) == HIGH){
    digitalWrite(MOTOR_LINKS, LOW);
    digitalWrite(MOTOR_RECHTS, LOW);
} else if(digitalRead(LED_GELB) == HIGH){
    digitalWrite(MOTOR_LINKS, LOW);
    digitalWrite(MOTOR_RECHTS, LOW);
} else if(digitalRead(LED_ROT) == HIGH && digitalRead(LED_GELB) == HIGH){
    digitalWrite(MOTOR_LINKS, LOW);
    digitalWrite(MOTOR_RECHTS, LOW);
}

// oder besser so:
if(digitalRead(LED_ROT) == HIGH || digitalRead(LED_GELB) == HIGH){
    digitalWrite(MOTOR_LINKS, LOW);
    digitalWrite(MOTOR_RECHTS, LOW);
}
```

(2) Wenn die Ampel rot leuchtet, oder wenn wenn die Ampel gelb leuchtet, oder beides, steht das Auto.

```
int LED_GRUEN = 9;
int LED_GELB = 10;
int LED_ROT = 11;
int MOTOR_LINKS  = 12;
int MOTOR_RECHTS = 13;
if(digitalRead(LED_ROT) == HIGH){
    digitalWrite(MOTOR_LINKS, LOW);
    digitalWrite(MOTOR_RECHTS, LOW);
} else if(digitalRead(LED_GELB) == HIGH){
    digitalWrite(MOTOR_LINKS, LOW);
    digitalWrite(MOTOR_RECHTS, LOW);
} else if(digitalRead(LED_ROT) == HIGH && digitalRead(LED_GELB) == HIGH){
    digitalWrite(MOTOR_LINKS, LOW);
    digitalWrite(MOTOR_RECHTS, LOW);
}

// oder besser so:
if(digitalRead(LED_ROT) == HIGH || digitalRead(LED_GELB) == HIGH){
    digitalWrite(MOTOR_LINKS, LOW);
    digitalWrite(MOTOR_RECHTS, LOW);
}
```

Häufige Fragen und Probleme

Warum schreibt man das logische Und als `&&` und nicht als `&`?

Das Zeichen `&` ist als Bit-Operation reserviert. Eine Bit-Operation verknüpft zwei Zahlen miteinander abhängig von deren Schreibweise als Binärzahl. Dies ist in C++ eine wichtige Operation, da C++ aus der Programmiersprache C entstanden ist, welche sehr maschinennah arbeitet, bei welcher man also fast direkt am Speicher des PCs programmiert. Hierbei sind Bit-Operationen sehr schnell.

Warum schreibt man den Vergleich zweier Werte als `==` und nicht als `=`?

Das einfache Gleich `=` ist reserviert für eine Zuweisung.