Der Zuweisungsoperator
=

Der Zuweisungsoperator wird verwendet, um Variablen ihren Inhalt zu geben. Dabei gilt, dass der Wert von rechts neben dem Operator nach links kopiert wird.

int x = 8;

Die Anweisung bewirkt, dass der Inhalt der Variable x nach Abarbeitung der Anweisung 8 beträgt. Die Zahl 8 tritt hier als Zahlenkonstante auf.

int y = x;

Die Anweisung bewirkt, dass der Inhalt der Variable y nach Abarbeitung der Anweisung den gleichen Wert wie die Variable x besitzt.

Die Vergleichsoperatoren

Mit Vergleichsoperatoren ist es möglich, den Inhalt von Variablen mit dem Inhalt von anderen Variablen oder Konstanten zu vergleichen.

== Test auf Gleichheit
!= Test auf Unleichheit
< kleiner als
> größer als
<= kleiner oder gleich
>= größer oder gleich

Das Ergebnis eines Vergleichs ist immer ein Wert vom Datentyp bool: true oder false.

keyword if

Die Kontrollstruktur dient dazu, abhängig von Bedingungen Quellcode auszuführen oder wegzulassen.

temp_warning | Arduino 1.8.19

Datei Bearbeiten Sketch Werkzeuge Hilfe

temp_warning §

```
#include <SoftwareSerial.h>

const int tempsensorIN = A0;
const int warnLED = 2;

void setup()
{
    Serial.begin(9600);

    pinMode(warnLED, OUTPUT); // Daten an LED schicken
    digitalWrite(warnLED, LOW); // Startwert: LED aus
}

void loop()
{
    int tempsensorval = analogRead(tempsensorIN);
    float volt = (tempsensorval / 1024.0) * 5;
    float temp = (volt -0.5) * 100;
    Serial.print("Tempratur in Grad Celsius: ");
    Serial.println(temp);

    if(temp>=23)
    {
        digitalWrite(warnLED, HIGH); // WarnLED an!
    }
    else
    {
        digitalWrite(warnLED, LOW); // WarnLED aus
    }

    delay(1000); // 1 Sekunden warten
}
```

Die Variable

Eine Variable dient der Speicherung von Daten und besteht immer aus Datentyp und Variablenname.

Der Variablenname soll möglichst genau den Inhalt der Variable beschreiben.

Der Datentyp legt fest, wie der Inhalt der Variable interpretiert wird.

elementare Datentypen

int ... ganze Zahlen
float ... Fließkommazahlen
bool ... Wahrheitswert
char ... ASCII-Zeichen

Funktionen

Eine Funktion ist eine Zusammenfassung von Anweisungen. Typischerweise wird ein in sich geschlossener Algorithmus als Funktion implementiert.

Durch die Verwendung von Funktionen wird Quellcode strukturiert und reduziert, da eine einmal geschriebene Funktion beliebig oft aufrufbar ist. Außerdem müssen Änderungen am Algorithmus nur an einer einzigen Stelle gemacht werden und nicht überall dort, wo er eingesetzt wird.

Das Erstellen einer Funktion folgt immer dem gleichen Schema: Sie besteht aus einem Funktionskopf und einem Funktionskörper.

```
Datentyp des Rückgabewertes   Funktionsname   (Parameterliste)
{
    Algorithmus / Anweisung(en);
}
```

Der Funktionskopf ist eine Art Bedienungsanleitung, er beschreibt also, wie wir eine bestimmte Funktion verwenden.

Der Funktionsname dient dabei wie der Variablenname als kurze Beschreibung, welche Aufgabe erledigt oder was von Algorithmus berechnet wird. Der Datentyp des Rückgabewertes gibt an, welchen Datentyp das Ergebnis besitzt. In der Parameterliste befinden sich Variablen, deren Werte zur Berechnung des Ergebnis notwendig sind.

Wait what?
Um dieses Detail zu verstehen, muss man wissen, wie Daten gespeichert werden: Im Kleinsten kennt ein PC nur die beiden Zustände "Strom an" und "Strom aus", also 1 und 0.

Der Speicher eines PC besteht nun aus sehr vielen Zellen (Bits), die den 1 oder 0 annehmen können. Die kleinste Einheit, die mit den Zellen gebildet werden kann, ist ein Byte und besteht aus 8 einzelnen Bit.

Legen wir nun eine Variable an, wird dafür Speicherplatz reserviert. Da im Speicher nun immer nur 1en oder 0en stehen, bestimmt der Datentyp, was der Inhalt der Variable bedeuten soll.

Die Größe des Speicherplatzes (Anzahl Bits) richtet sich dabei ausschließlich nach dem Datentyp. Für den Datentyp int (ganze Zahl) werden gewöhnlich mindestens 4 Byte (32 Bit reserviert). Da hinein passen dann also 2^{32} Zahlen: die Werte von -2.147.483.648 bis 2.147.483.647.

Zur Verdeutlichung betrachten wir den Datentyp int: Er bestimmt, dass der Inhalt des Speichers eine ganze Zahl darstellt.

Würde nur 1 Bit Speicher reserviert, könnten wir nur die Zahl 1 und 0 speichern. Haben wir 2 Bit zur Verfügung, ergibt das die Zahlen 0 bis 3:

0	0	← 0
0	1	← 1
1	0	← 2
1	1	← 3

Uuuuund 3 Bit:

0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Uuuuund 4 Bit:

0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Bibliothek

Bei vielen Themen muss das Rad nicht neu erfunden werden! Algorithmen beispielsweise in der Mathematik oder physikalische Gesetze sind allgemeingültig. Sie wurden bereits in Funktionen implementiert und zu Bibliotheken zusammengefasst. Diese Bibliotheken können wir und in unser Programm einbinden und die Funktionen daraus verwenden.

Der Befehl zum Einbinden einer Bibliothek lautet
#include <Name_der_Bib>
und steht ganz oben im Programm.

Um Bibliotheken bzw. deren Funktionen benutzen zu können, muss man nun natürlich noch wissen, welche es gibt und wie sie definiert sind. All diese Infos stehen in der Dokumentation einer Bibliothek.

Hier einige Beispiele:
* * *
int quersumme (int zahl)
Die Funktion berechnet die Quersumme einer ganzen Zahl. Das Ergebnis ist ebenfalls eine ganze Zahl.
* * *
float preis_durchschnitt (float *preise, int anz)
Die Funktion bekommt ein Feld mit Preisen und deren Anzahl übergeben und berechnet deren Durchschnitt.
* * *
bool is_prim (int zahl)
Die Funktion gibt zurück, ob eine übergebene Zahl eine Primzahl ist (true) oder nicht (false).

Objektorientierte Programmierung
aka
abstrakte Datentypen

Die wirkliche Welt ist leider komplex. Denken wir an Mobilität, fallen uns die verschiedensten Möglichkeiten ein, von A nach B zu kommen: Mit dem Fahrrad, dem Auto dem Zug, E-Bike, Bus, Straßenbahn, Traktor, Segelboot, Fähre, Flugzeug, Helikopter, Hovercraft, ...

Alle sind unterschiedlich und haben dennoch Gemeinsamkeiten: eine Art Antrieb, Geschwindigkeiten, Strecken, eine Anzahl beförderter Personen, ... Wollen wir ein Fahrzeug allerdings verkaufen statt benutzen, benötigen wir nun ganz andere Eigenschaften. Mit unseren einzelnen Variablen und den elementaren Datentypen stoßen wir nun schnell an Grenzen.

Hier hilft uns die Objektorientierte Programmierung. Für den Zweck des Programms werden die benötigten Eigenschaften eines Objektes in einer Klasse gesammelt. Der Name der Klasse dient dann als neuer Datentyp.

Außerdem stehen Methoden bereit, um typische Operationen mit unserem Objekt auszuführen. (Methoden sind Funktionen, die zu einer Klasse gehören.)

Wie sieht eine Klasse aus und
wie wird sie benutzt?
Die Definition einer Klasse beginnt mit dem Schlüsselwort class und besitzt einen privaten Bereich für alle Eigenschaften und einen öffentlichen Bereich für die Methoden. Im Beispiel geht es um Studierende:

```
class student
{
    private:
        string vorname;
        string nachname;
        int matrikel_nr;
        float gesamt_note;
    public:
        student();
        student(string vn, string nn, int mnr);
        float berechnung_gesamtnote();
        void print_data();
};
```

Wofür brauchen wir das?
Kommunikation!

Es steht eine serielle Schnittstelle für die Kommunikation zwischen PC und Arduino zur Verfügung, die als Klasse umgesetzt in einer Bibliothek zur Verfügung steht.
Die Dokumentation findet sich online:
https://docs.arduino.cc/learn/built-in-libraries/software-serial

```
int main()
{
    student A("Ariane", "Jacobs", 12345);
    float note = A.berechnung_gesamtnote();
    A. print_data();
};
```

32 Bit (4 Byte)

64 Bit (8 Byte)