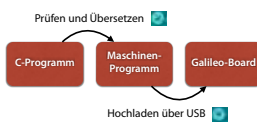


Teile und Herrsche

Programmieren für Ingenieure
Sommer 2015

Andreas Zeller, Universität des Saarlandes

Vom Programm zum Prozessor



Funktionsaufrufe

- Die meisten Funktionen haben *Parameter*, die ihre Funktionsweise bestimmen
- Beim *Aufruf* muss für jeden Parameter ein Wert (*Argument*) angegeben werden

```
digitalWrite(pin_number, value);  
Funktionsname → Wert für pin_number, Wert für value
```

Variablen

- Variablen dienen dazu, Werte zu speichern.
- Mit der Anweisung

```
int led = 13;
```

wird led als eine Variable eingeführt, die mit dem Wert 13 belegt ist.
- Nach der Anweisung steht led stellvertretend für den Variablenwert

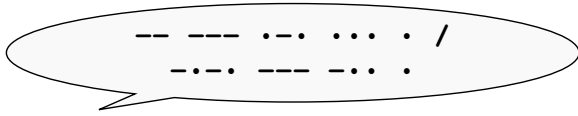
Symbolisches Blinken

```
// Pin 13 has an LED connected on most  
// Arduino boards. Give it a name:  
int led = 13;  
  
void setup() {  
  pinMode(led, OUTPUT);  
}  
  
void loop() {  
  digitalWrite(led, HIGH);  
  delay(1000);  
  digitalWrite(led, LOW);  
  delay(1000);  
}
```

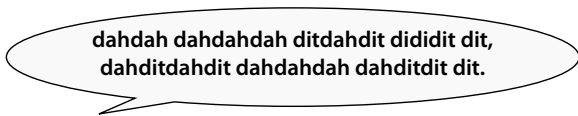
Themen heute

- Eigene Funktionen
- Parameter
- Fallunterscheidungen
- Fehlersuche

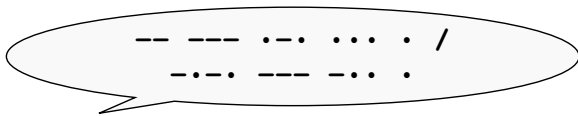
Morse-Code



Morse-Code



Morse-Code



MORSE CODE

Morse-Code

- Ein *Dah* ist dreimal so lang wie ein *Dit*.
- Die Pause zwischen zwei gesendeten Symbolen ist ein *Dit* lang.
- Zwischen Buchstaben in einem Wort wird eine Pause von der Länge eines *Dah* (oder drei *Dits*) eingeschoben.
- Die Länge der Pause zwischen Wörtern entspricht sieben *Dits*.

Wikipedia



```
int dit_delay = 500;           // length of a dit in ms

void loop() {
  // send a dit
  digitalWrite(led, HIGH);
  delay(dit_delay);

  digitalWrite(led, LOW);
  delay(dit_delay);
}
```



```
int dit_delay = 500;           // length of a dit in ms
int dah_delay = dit_delay * 3; // length of a dah in ms

void loop() {
  // send a dit
  digitalWrite(led, HIGH);
  delay(dit_delay);

  digitalWrite(led, LOW);
  delay(dit_delay);

  // send a dah
  digitalWrite(led, HIGH);
  delay(dah_delay);

  digitalWrite(led, LOW);
  delay(dit_delay);
}
```

Berechnung!

Arithmetische Operatoren

In aufsteigender Bindung:

1. Addition (+), Subtraktion (-)
Assoziativität: von links nach rechts
2. Multiplikation (*), Division (/), Modulus (%)
Assoziativität: von links nach rechts
3. Vorzeichen (+, -)
Assoziativität: von rechts nach links

```
int y = -3 + 7 % 3  
int y = (-3) + (7 % 3)
```

Eigene Funktionen

- Wir wollen die Anweisungen für *Dahs* und *Dits* in *eigene Funktionen* zusammenfassen
- Eine eigene Funktion wird wie `setup()` und `loop()` als *Folge von Anweisungen* definiert:

```
void name() {  
    Anweisung 1;  
    Anweisung 2;  
    ""  
}
```



```
int dit_delay = 500; // length of a dit in ms  
int dah_delay = dit_delay * 3; // length of a dah in ms  
  
void loop() {  
    // send a dit  
    digitalWrite(led, HIGH);  
    delay(dit_delay);  
  
    digitalWrite(led, LOW);  
    delay(dit_delay);  
  
    // send a dah  
    digitalWrite(led, HIGH);  
    delay(dah_delay);  
  
    digitalWrite(led, LOW);  
    delay(dit_delay);  
}
```



```
void loop() {  
  dit();  
  dah();  
}  
  
void dit() {  
  // send a dit  
  digitalWrite(led, HIGH);  
  delay(dit_delay);  
  
  digitalWrite(led, LOW);  
  delay(dit_delay);  
}  
  
void dah() {  
  // send a dah  
  digitalWrite(led, HIGH);  
  delay(dah_delay);  
  
  digitalWrite(led, LOW);  
  delay(dit_delay);  
}
```

Demo



Teile und Herrsche

- Idee: Ein Problem in (kleinere) Teilprobleme zerlegen
- Grundprinzip der Informatik
- Grundprinzip politischer Machtausübung

Gaius Julius Cäsar Wikipedia

S senden

```
void morse_S() {  
    dit();  
    dit();  
    dit();  
}
```

oder (kürzer)

```
void morse_S() {  
    dit(); dit(); dit();  
}
```

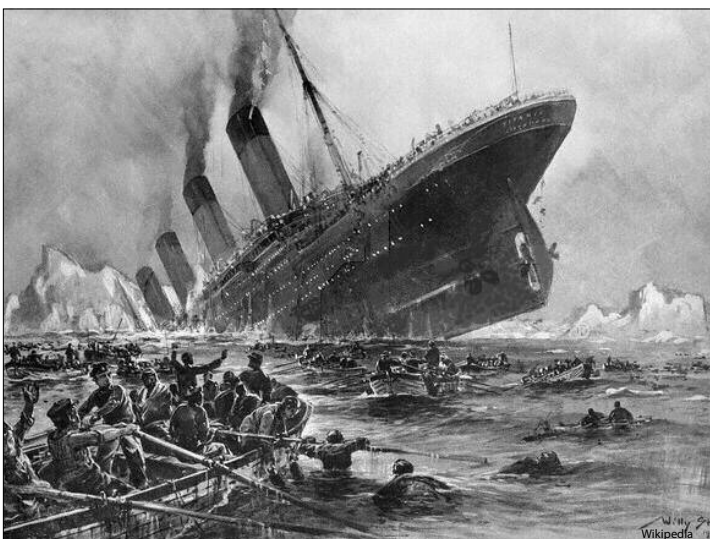
Save Our Souls

```
void morse_S() {  
    dit(); dit(); dit();  
}
```

```
void morse_0() {  
    dah(); dah(); dah();  
}
```

```
void morse_SOS() {  
    morse_S(); morse_0(); morse_S();  
    delay(dit_delay * 6);  
}
```

•••---•••/•••---•••/•••---•••



A
B
C
D
E
F
G
H
I
K
L
M
N
O
P
Q
R
S
T

U
V
W
X
Y
Z

1
2
3
4
5
6
7
8
9
0

... .. - - - SINK

*CQD CQD SOS Titanic
Position 41.44 N 50.24 W.
Require immediate
assistance. Come at
once. We struck an
iceberg. Sinking*

Demo



We are sinking

Eigene Parameter

- Ziel: Eine Funktion `send_number(n)`, die eine gegebene Zahl n per Morse ausgibt
- n wird zum *Parameter* der Funktion

Eigene Parameter

- Parameter werden (mitsamt Typen) bei der Definition in Klammern angegeben

```
void name(int p1, int p2, ...) {  
    Anweisungen...;  
}
```

- Bei uns also:

```
void morse_number(int n) {  
    Anweisungen...;  
}
```

Fallunterscheidung

- Je nach Wert von n müssen unterschiedliche Anweisungen ausgeführt werden:
 - Ist $n = 1$, dann `• - - - -` senden
 - Ist $n = 2$, dann `• • - - -` senden
 - usw.

Fallunterscheidung

- Die *if-Kontrollstruktur* dient zum Programmieren von *Fallunterscheidungen*:

```
if (Bedingung) {  
    Anweisungen...;  
}
```

- Die Anweisungen werden *nur* ausgeführt, wenn die Bedingung erfüllt ist.

Vergleichsoperatoren

In aufsteigender Bindung:

1. Logisches Oder \vee (| |)
2. Logisches Und \wedge (&&)
3. Größenvergleiche (<, >, <=, >=)
4. Gleichheit = (==), Ungleichheit \neq (!=)
5. Logisches Nicht \neg (!) ==, nicht = !

```
if (x >= y && !(x == y))
```

Fallunterscheidung

- Je nach Wert von n müssen unterschiedliche Anweisungen ausgeführt werden:
 - Ist $n = 1$, dann • - - - - senden
 - Ist $n = 2$, dann • • - - - - senden
 - usw.

Fallunterscheidung

```
// send n in morse code  
void morse_digit(int n) {
```

Fallunterscheidung

```
// send n in morse code  
void morse_digit(int n) {  
    if (n == 0) {  
        dah(); dah(); dah(); dah(); dah();  
    }  
}
```

Fallunterscheidung

```
// send n in morse code  
void morse_digit(int n) {  
    if (n == 0) {  
        dah(); dah(); dah(); dah(); dah();  
    }  
    if (n == 1) {  
        dit(); dah(); dah(); dah(); dah();  
    }  
}
```

```
// send n in morse code
void morse_digit(int n) {
    if (n == 0) {
        dah(); dah(); dah(); dah(); dah();
    }
    if (n == 1) {
        dit(); dah(); dah(); dah(); dah();
    }
    if (n == 2) {
        dit(); dit(); dah(); dah(); dah();
    }
    // usw. für 3-8
    if (n == 9) {
        dah(); dah(); dah(); dah(); dit();
    }
    pause_letter();
}
```

Aufruf

- Einmal definiert, wird morse_digit() wie jede andere Funktion aufgerufen:

```
void morse_digit(int n) {
    // wie oben
}
```

```
void loop() {
    morse_digit(5);
    morse_digit(0);
    morse_digit(2);
    morse_digit(4);
}
```

Position 41.44 N 50.24 W

Demo

Von Ziffern zu Zahlen

- Wie geben wir mehrstellige Zahlen aus?
- Ziel: Funktion `morse_number(n)`, die n durch Aufrufe an `morse_digit()` ausgibt

```
morse_number(5024) →  
  morse_digit(5)  
  morse_digit(0)  
  morse_digit(2)  
  morse_digit(4)
```

Von Ziffern zu Zahlen

- Beobachtung: Will ich 5024 ausgeben, kann ich 502 ausgeben, gefolgt von 4.

```
morse_number(5024) →  
  morse_number(502)  
  morse_digit(4)
```

- Um 502 auszugeben, kann ich 50 ausgeben, gefolgt von 2.

```
morse_number(502) →  
  morse_number(50)  
  morse_digit(2)
```

Von Ziffern zu Zahlen

Allgemeines Prinzip:

1. Hat n mehr als eine Ziffer (d.h. $n \geq 10$), gebe ich zuerst $n / 10$ aus
2. Anschließend gebe ich die letzte Ziffer aus (d.h. $n \bmod 10$)

Demo

Von Ziffern zu Zahlen

So sieht morse_number() aus:

```
void morse_number(int n) {  
    if (n >= 10) {  
        morse_number(n / 10);  
    }  
    morse_digit(n % 10);  
}
```

Von Ziffern zu Zahlen

```
void morse_number(int n) {  
    if (n >= 10) {  
        morse_number(n / 10);  
    }  
    morse_digit(n % 10);  
}
```

```
morse_number(5024)  
→ morse_number(502)  
  → morse_number(50)  
    → morse_number(5)  
      → morse_digit(5)  •••••  
        → morse_digit(0)  - - - - -  
          → morse_digit(2)  •• - - -  
            → morse_digit(4)  •••••
```

Demo

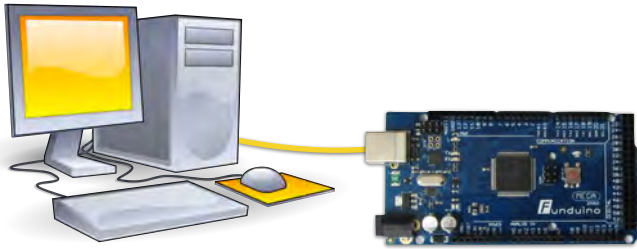
Rekursion

- Ruft eine Funktion sich selbst erneut auf, nennt man dies *Rekursion*
- Jede Berechnung lässt sich durch ausschließlich *Funktionen, Bedingungen* und *Rekursion* ausdrücken
- Alles, was man (irgendwie) berechnen kann, können Sie jetzt programmieren
(im Prinzip jedenfalls)

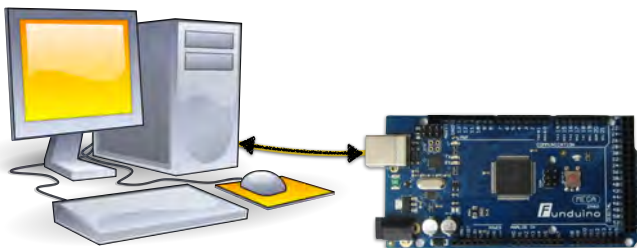
Fehlersuche

- Während einer komplexen Berechnung ist es hilfreich, zu verfolgen, was geschieht
- Hierfür dient die *serielle Ausgabe* der Arduino-Plattform

USB-Anschluss



Datenübertragung



Verfolgen über Werkzeuge → Serieller Monitor

Serial.begin()

- `Serial.begin(baud)` richtet die *serielle Schnittstelle* ein, um mit Geschwindigkeit *baud* (bits/s) Daten zu übertragen
- Beispiel:

```
void setup() {  
  // Transfer at 9600 bits/s  
  Serial.begin(9600);  
}
```

Serial.print()

- Die Funktion Serial.print(x) gibt x auf der *seriellen Schnittstelle* aus
- Serial.println(x): Genauso, aber mit Zeilenende
- Beispiel:

```
void morse_number(int n) {  
    Serial.println(n);  
    Anweisungen...;  
}
```

Text ausgeben

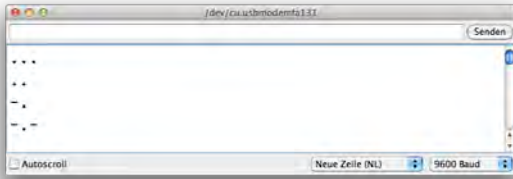
- Mit Serial.print() und Serial.println() kann man auch *Text* ausgeben
- Text wird in "\"" eingeschlossen
- Beispiel:

```
void morse_number(int n) {  
    Serial.print("morse_number");  
    Serial.print(n);  
    Serial.println("");  
    Anweisungen...;  
}
```

Text ausgeben

```
void dit() {  
    Serial.print(".");  
    Anweisungen...;  
}  
  
void dah() {  
    Serial.print("-");  
    Anweisungen...;  
}  
  
void pause_letter() {  
    Serial.println("");  
    delay(letter_delay);  
}
```

Serieller Monitor



Demo

Binärzahlen

- Rechner stellen Daten intern als *Bits* dar – nur 0 und 1
- Zahlen werden im *Binärsystem* gespeichert
- Die Zahl 37 etwa wird gespeichert als

$$\begin{array}{c} 100101 \\ \swarrow \quad | \quad \searrow \\ 32 + 4 + 1 = 37 \end{array}$$

Von Ziffern zu Zahlen

So sieht `morse_number()` aus:

```
void morse_number(int n) {  
    if (n >= 10) {  
        morse_number(n / 10);  
    }  
    morse_digit(n % 10);  
}
```

Kann ich auch eine andere Zahlenbasis als 10 nehmen?

Zahlen binär morsen

Um Zahlen im Binärsystem zu morsen:

```
void morse_binary(int n) {  
    if (n >= 2) {  
        morse_number(n / 2);  
    }  
    morse_digit(n % 2);  
}
```

Demo

Basis 10 und 2

```
void morse_decimal(int n) {
    if (n >= 10) {
        morse_decimal(n / 10);
    }
    morse_digit(n % 10);
}

void morse_binary(int n) {
    if (n >= 2) {
        morse_binary(n / 2);
    }
    morse_digit(n % 2);
}
```

Beliebige Zahlenbasis

Zahlen in Basis *base* ausgeben:

```
void morse_number(int n, int base) {
    if (n >= base) {
        morse_number(n / base, base);
    }
    morse_digit(n % base);
}
```

Demo

Ausblick

- Zuweisungen
- Eigene Schleifen
- Verkehrssteuerung
- Eingabeelemente

Eigene Funktionen

```
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}

void digitalWrite(int pin, int value) {
  pinMode(pin, OUTPUT);
  digitalWrite(pin, value);
  delay(100);
}
```

Eigene Parameter

- Parameter werden (mitsamt Typen) bei der Definition in Klammern angegeben
- Bei uns also:

```
void name(int p1, int p2, ...) {
  Anweisungen;
}

void morse_number(int n) {
  Anweisungen;
}
```

Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

morse_number(5024)
- morse_number(502)
- morse_number(50)
- morse_number(5)
- morse_digit(5)
- morse_digit(0)
- morse_digit(2)
- morse_digit(4)
```

Ablauf verfolgen



Verfolgen über Werkzeuge → Serieller Monitor

Handouts

SINK

```
void morse_S() {
    dit(); dit(); dit();
    pause_letter();
}

void morse_I() {
    dit(); dit();
    pause_letter();
}

void morse_SINK() {
    morse_S(); morse_I(); morse_N(); morse_K();
    pause_word();
}
    ...      ..      -.      --
```

Eigene Parameter

- Parameter werden (mitsamt Typen) bei der Definition in Klammern angegeben

```
void name(int p1, int p2, ...) {
    Anweisungen...;
}
```

- Bei uns also:

```
void morse_number(int n) {
    Anweisungen...;
}
```

```
// send n in morse code
void morse_digit(int n) {
    if (n == 0) {
        dah(); dah(); dah(); dah(); dah();
    }
    if (n == 1) {
        dit(); dah(); dah(); dah(); dah();
    }
    if (n == 2) {
        dit(); dit(); dah(); dah(); dah();
    }
    // usw. für 3-8
    if (n == 9) {
        dah(); dah(); dah(); dah(); dit();
    }
    pause_letter();
}
```

