Systemnahe Programmierung in C

8 Kontrollstrukturen

J. Kleinöder, D. Lohmann, V. Sieh

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Sommersemester 2025

http://sys.cs.fau.de/lehre/ss25

Goto-Anweisung

```
    ...
    goto Label
    ...
Label:
    ...
```

```
    ...
Label:
    ...
    goto Label
    ...
```

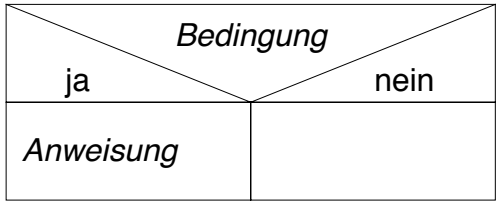
- `goto`-Anweisungen werden selten gebraucht
- `goto` in früheren JAVA-Versionen enthalten
- Edgar Dijkstra: „Go To Statement Considered Harmful”
- sie erzeugen leicht „Spagetti-Code”
- Label darf nicht letzte Anweisung in Funktion sein

`goto`- und `if (...) goto`-Anweisungen sind die einzigen Kontrollstrukturen, die die Hardware wirklich ausführen kann (wichtig für das spätere Verständnis von z.B. Unterbrechungen).

Bedingte Anweisung [=Java]

- `if`-Anweisung (bedingte Anweisung)

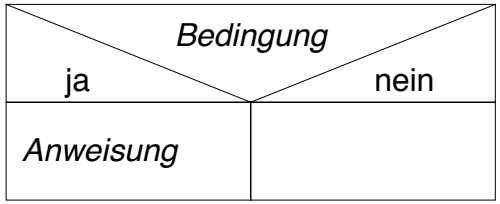
```
if (Bedingung)
    Anweisung;
```



Bedingte Anweisung [= Java]

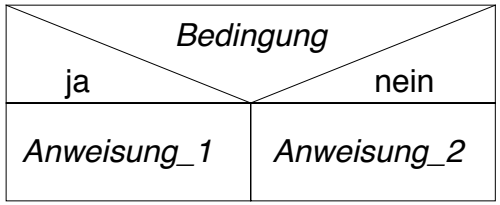
- `if`-Anweisung (bedingte Anweisung)

```
if (Bedingung)
    Anweisung;
```



- `if-else`-Anweisung (einfache Verzweigung)

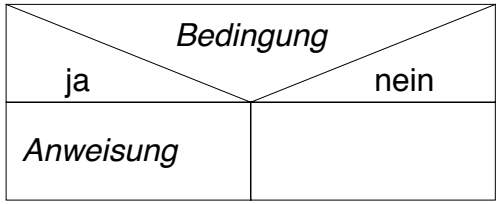
```
if (Bedingung)
    Anweisung1;
else
    Anweisung2;
```



Bedingte Anweisung [= Java]

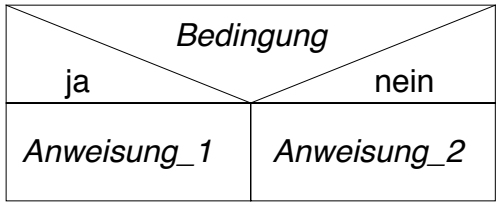
- `if`-Anweisung (bedingte Anweisung)

```
if (Bedingung)
    Anweisung;
```



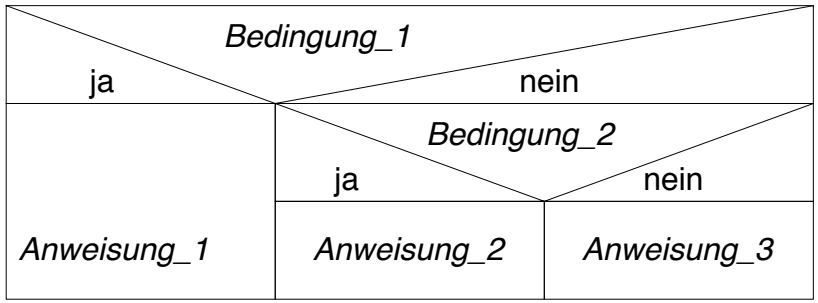
- `if-else`-Anweisung (einfache Verzweigung)

```
if (Bedingung)
    Anweisung_1;
else
    Anweisung_2;
```



- `if-else-if`-Kaskade (mehrfache Verzweigung)

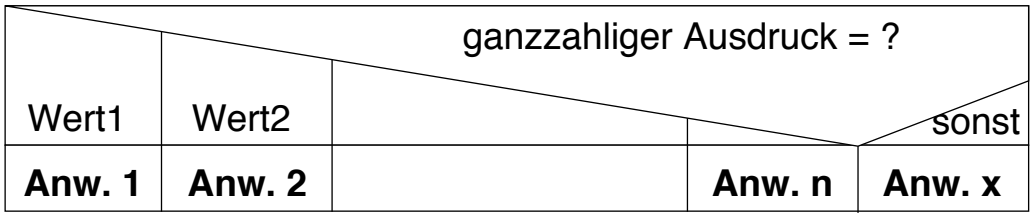
```
if (Bedingung_1)
    Anweisung_1;
else if (Bedingung_2)
    Anweisung_2;
else
    Anweisung_3;
```



Fallunterscheidung [= Java]

`switch`-Anweisung (Fallunterscheidung)

- Alternative zur `if`-Kaskade bei Test auf Ganzzahl-Konstanten

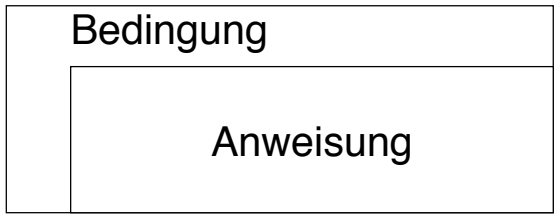



```
switch (Ausdruck) {
case Wert1:
    Anweisung1;
    break;
case Wert2:
    Anweisung2;
    break;
...
case Wertn:
    Anweisungn;
    break;
default:
    Anweisungx;
}
```

Abweisende und nicht-abweisende Schleife [=Java]

Abweisende Schleife

- `while`-Schleife
- Null- oder mehrfach ausgeführt



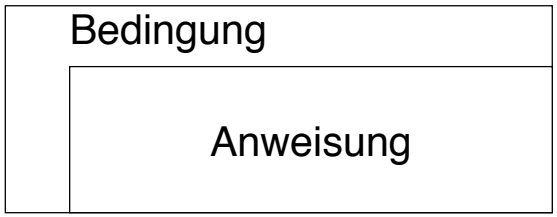
`while(`*Bedingung*`)`
    Anweisung`;`

```
while (
    sb_button_getState(BUTTON0)
        == RELEASED
) {
  ··· // do unless button press.
}
```

Abweisende und nicht-abweisende Schleife [=Java]

Abweisende Schleife

- `while`-Schleife
- Null- oder mehrfach ausgeführt

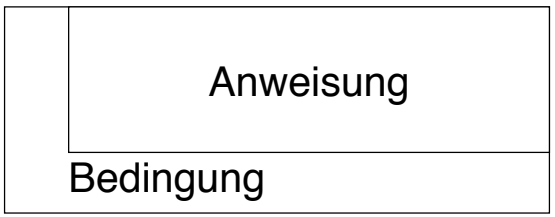



```
while(Bedingung)
    Anweisung;
```

```
while (
    sb_button_getState(BUTTON0)
        == RELEASED
) {
  ··· // do unless button press.
}
```

Nicht-abweisende Schleife

- `do-while`-Schleife
- Ein- oder mehrfach ausgeführt



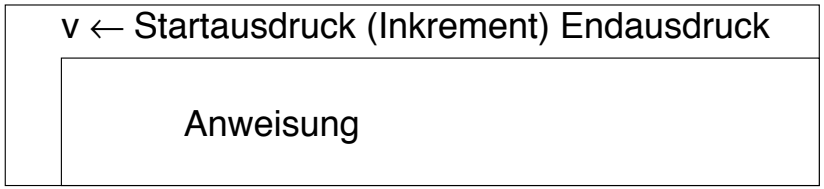

```
do
    Anweisung;
while(Bedingung);
```

```
do {
    ··· // do at least once
} while (
    sb_button_getState(BUTTON0)
        == RELEASED
);
```

Zählende Schleife [=Java]

- `for`-Schleife (Laufanweisung)

```
for (Startausdruck;
     Endausdruck;
     Inkrement–Ausdruck)
  Anweisung;
```



- Beispiel (übliche Verwendung: *n* Ausführungen mit Zählvariable)

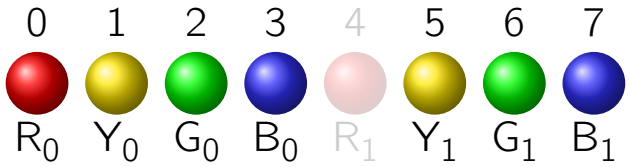
```
uint8_t sum = 0; // calc sum 1+...+10
for (uint8_t n = 1; n < 11; n++) {
    sum += n;
}
sb_7seg_showNumber( sum );
```

- Anmerkungen
  - Die Deklaration von Variablen (`n`) im *Startausdruck* ist erst ab C99 möglich
  - Die Schleife wird wiederholt, solange *Endausdruck* $\neq 0$ (*wahr*)
    $\rightsquigarrow$ die `for`-Schleife ist eine „verkappte“ `while`-Schleife

Schleifensteuerung [=Java]

- Die `continue`-Anweisung beendet den aktuellen Schleifendurchlauf
  ⇝ Schleife wird mit dem nächsten Durchlauf fortgesetzt

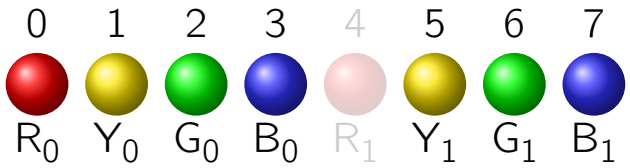
```
for (uint8_t led = 0; led < 8; led++) {
  if (led == RED1) {
    continue;          // skip RED1
  }
  sb_led_on(led);
}
```



Schleifensteuerung [=Java]

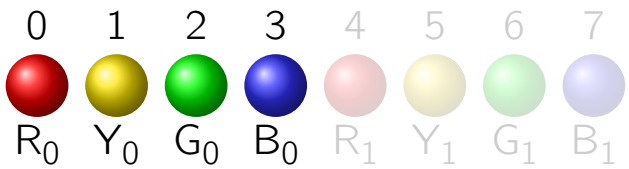
- Die `continue`-Anweisung beendet den aktuellen Schleifendurchlauf
  ⇝ Schleife wird mit dem nächsten Durchlauf fortgesetzt

```
for (uint8_t led = 0; led < 8; led++) {
  if (led == RED1) {
    continue;          // skip RED1
  }
  sb_led_on(led);
}
```



- Die `break`-Anweisung verlässt die (innerste) Schleife
  ⇝ Programm wird *nach* der Schleife fortgesetzt

```
for (uint8_t led = 0; led < 8; led++) {
  if (led == RED1) {
    break;             // break at RED1
  }
  sb_led_on(led);
}
```



Schleifen & `goto`-Anweisungen

Alle Schleifentypen lassen sich **semantisch-äquivalent** in Sequenzen mit `goto`-Anweisungen umbauen. Beispiel:

```
for (uint8_t led = 0; led < 8; led++) {
  if (led == RED1) {
    continue; /* skip RED1 */
  }
  sb_led_on(led);
}
```

```
  uint8_t led = 0;
  goto test;
loop:
  if (led == RED1)
    goto next;
  sb_led_on(led);
next:
  led++;
test:
  if (led < 8)
    goto loop;
end:
```