Einführung in die Programmierung

Tutorium 2 ('Kontrollstrukturen-Tutorium')

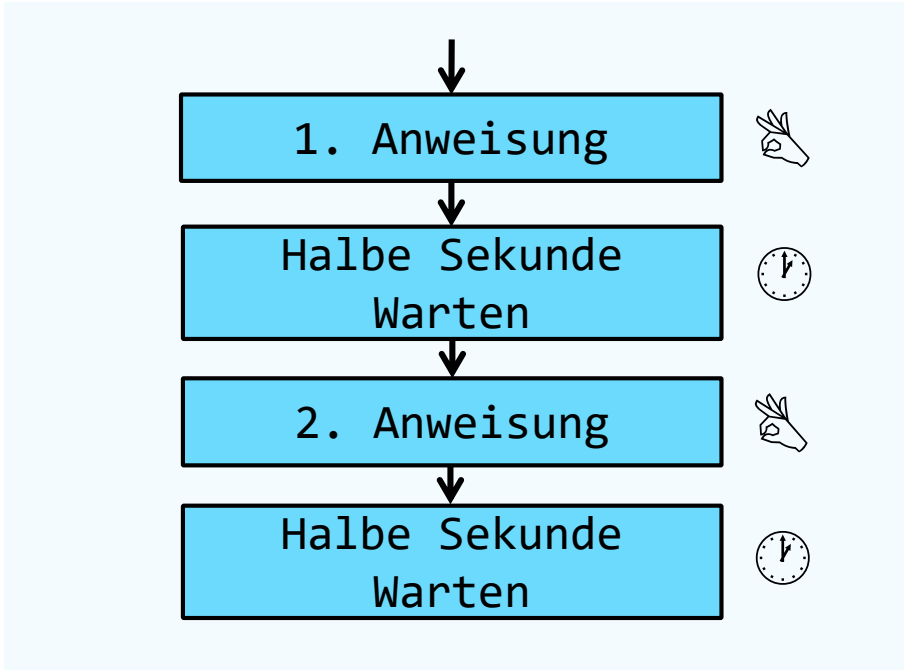
Wintersemester 2025/2026

Arbeitsgruppe Systemsoftware
Angewandte Informatik 12

technische universität
dortmund

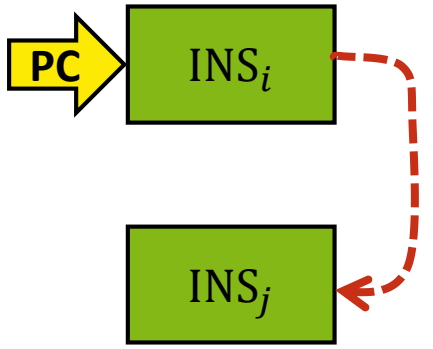
Arbeitsgruppe
Systemsoftware

Beispiel für ein Programm mit linearen Programmfluss

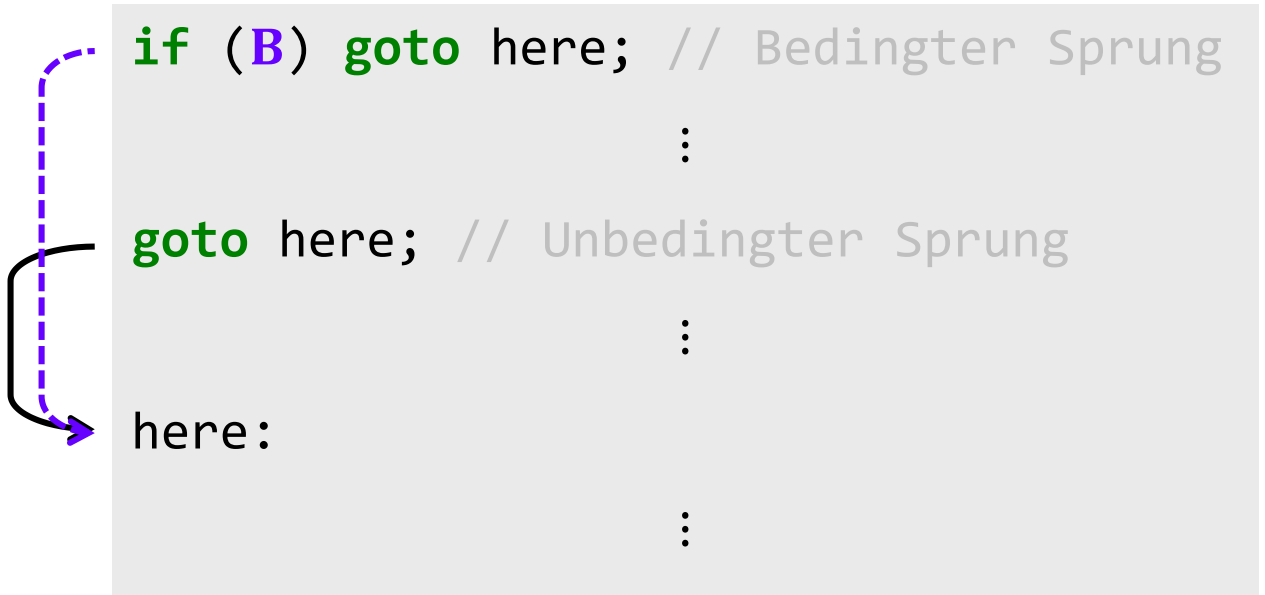


- Die Reihenfolge der Abarbeitung von Anweisungen eines Mikroprogramms wird **Programmfluss** genannt
- **Sprünge** dienen zur **Programmflusskontrolle**
- Sprung → 'Programmzähler' (**PC**) wird gezielt 'manipuliert'

Das Programm ist in der Lage auf unterschiedliche Datenlagen auch unterschiedlich zu reagieren!

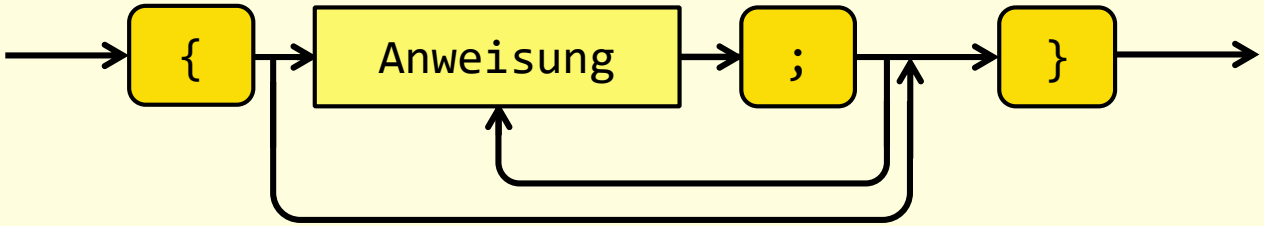


- Die **goto**–**Anweisung** springt zu einer **gesetzten Sprungmarke** im Code:

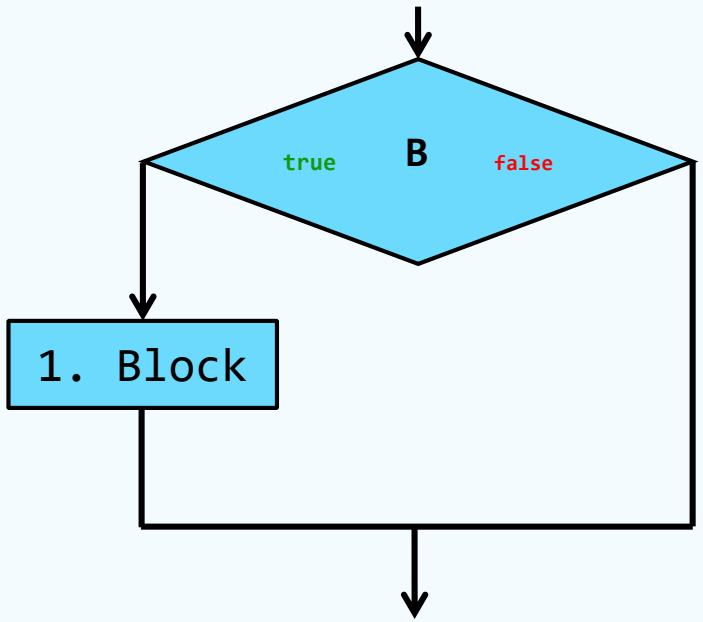


- **goto**-Anweisungen sollten dringend vermieden werden, da durch diese der Programmfluss unnachvollziehbar wird: Höhere Fehleranfälligkeit bei erschwerter Wartbarkeit des Programms!
- Alternativ sollten **Verzweigungen** und **Schleifen** genutzt werden!

Block

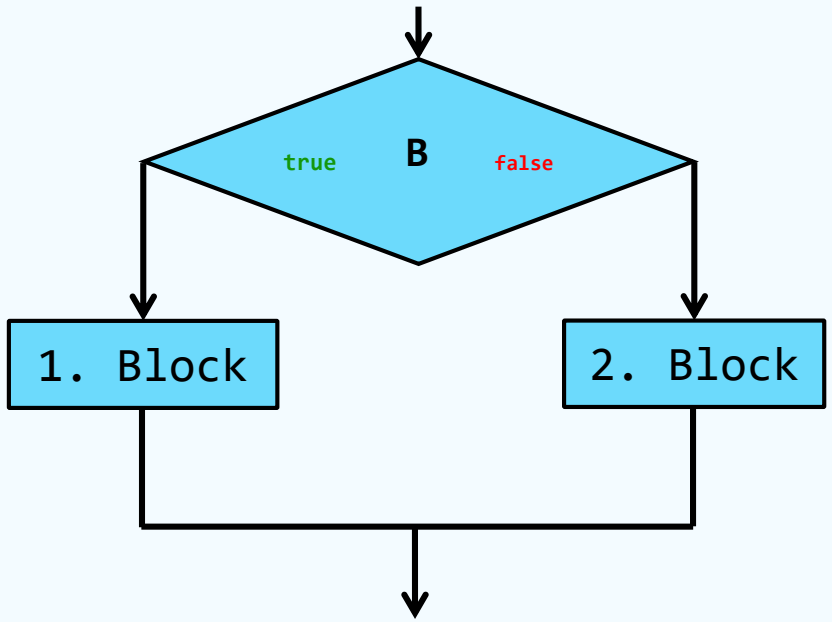


- Die **`if`-Verzweigung** ist ein Werkzeug, Programme zu **verzweigen**
- **`if`-Blöcke** werden nur ausgeführt, wenn die festgelegte Bedingung am Verzweigungspunkt erfüllt ist → Bei $\mathbf{B} = \texttt{true}$ wird der Block ausgeführt:




```
if (B) {
    1. Block
}
```

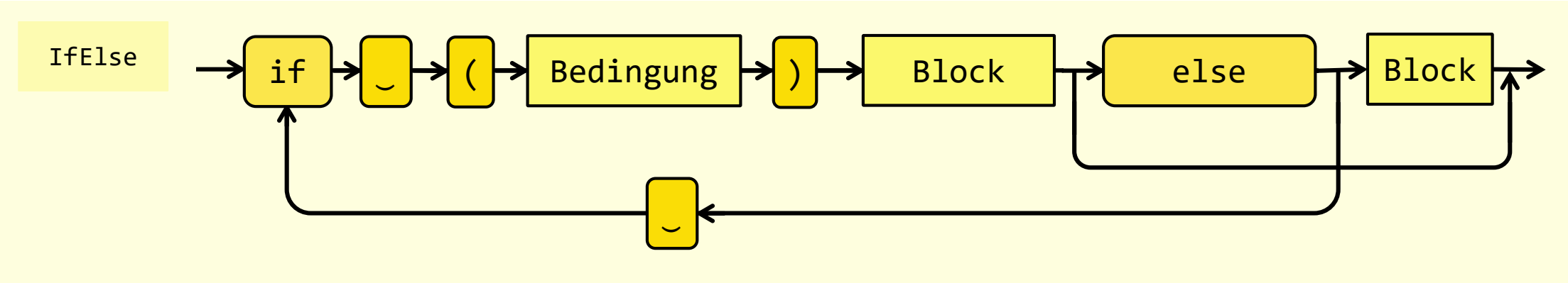
- **Alternativzweige** im Programmfluss durch **else-Blöcke**:
- Bei $\mathbf{B} =$ **false** wird anstelle des **if**-Blockes der **else**-Block ausgeführt



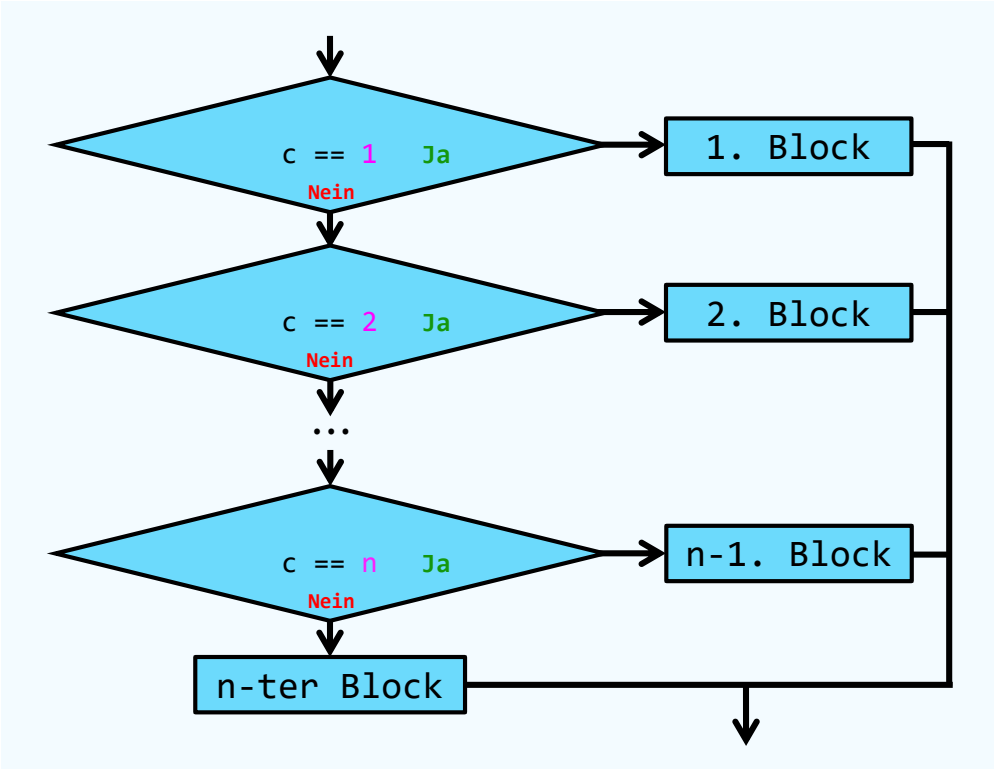

```
if (B) {
    1. Block
} else {
    2. Block
}
```

```
if (B) {
    1. Block
} else if (C) {
    2. Block
} else {
    3. Block
}
```

Allgemeiner Aufbau einer typischen **if-else**-Verzweigung



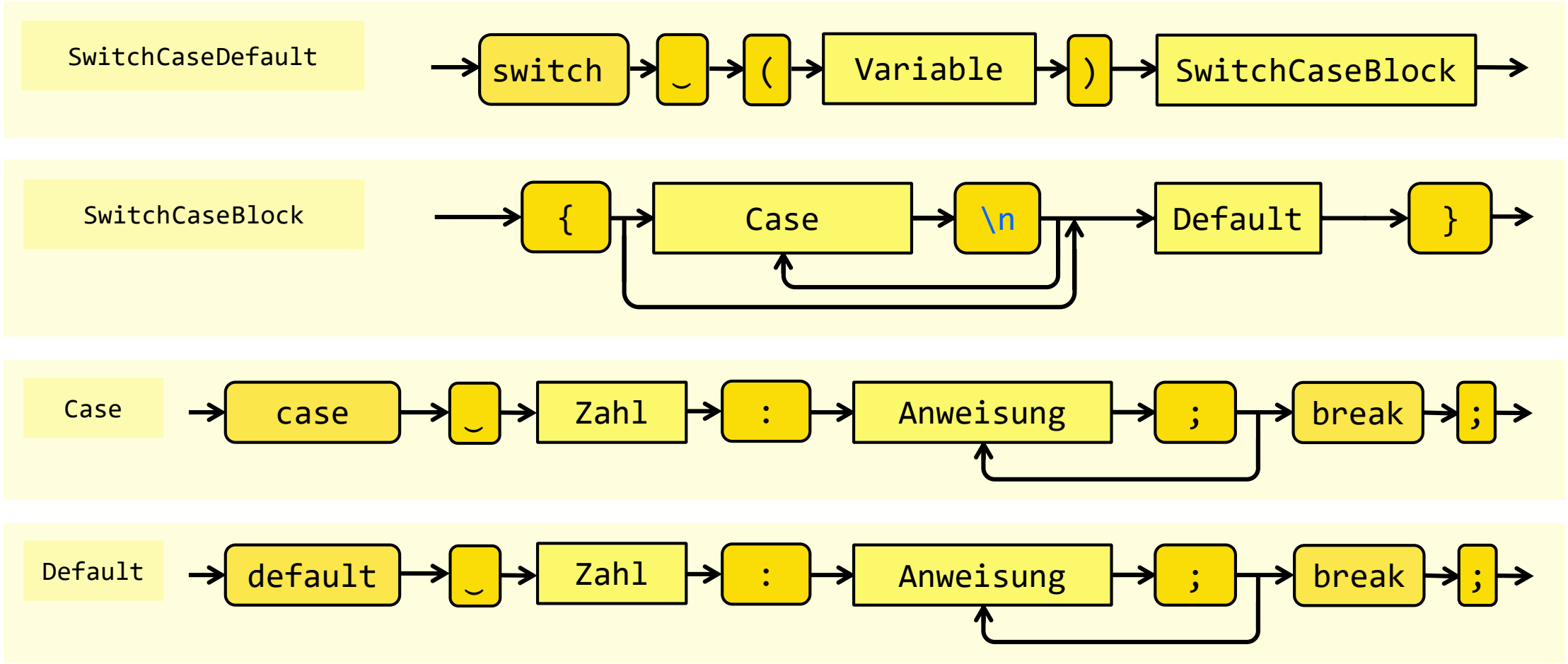
`switch-case-default`-Verzweigung




```
switch (c) {
    case 1: 1. Block ; break;
    case 2: 2. Block ; break;
    case 3: 3. Block ; break;
                ⋮
    case n: n-1. Block ; break;
    default: n-ter Block ; break;
}
```

- **`break`** veranlasst einen Sprung an das Ende des **`switch`**-Blocks
- Der **`default`**-Fall wird ausgeführt, wenn **`c`** keinem der **`case`**s entspricht

Allgemeiner Aufbau einer typischen **switch-case-default**-Verzweigung

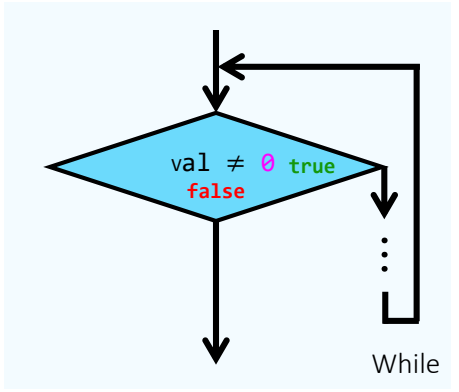


Mit der **While-Schleife** können Programmteile mehrfach ausgeführt werden, so lange eine bestimmte Bedingung erfüllt ist. Man spricht:

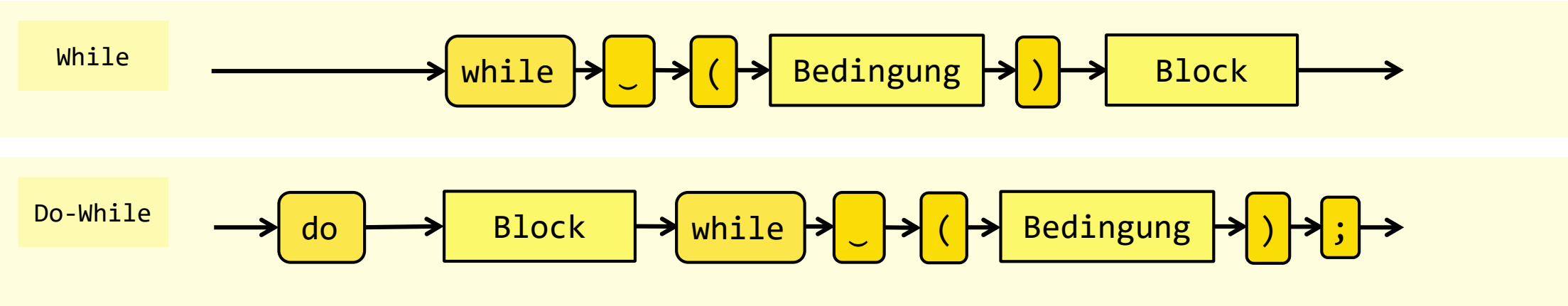
While ..., do ... ≙ Während ..., tue ... bzw. Do ... while ... ≙ Tue ... und wiederhole so lange wie ....

```
while (val != 0) { ··· }
```

```
⋮
Anfang:
⋮
LDA     (val)
BEQZ    (Ende)
JMP     (Anfang)
Ende:
⋮
```



Allgemeiner Aufbau einer typischen **`while`**-Schleife



Das Folgende Programm soll einen Countdown anzeigen, der sekündlich heruntergezählt wird. Die Variable **n** enthält den aktuellen Zählerstand.

→ Die Bedingung **n** ≠ **0** wird vor jedem Schleifendurchlauf geprüft

→ Gilt zu beginn **n** ≔ **0**, so wird die Schleife kein einziges mal ausgeführt

```
void countdown(unsigned int n) {
    while (n != 0) {
        delay(1000);
        printf("%03u\r", n);
        n--;
    }
}
```

Das Folgende Programm soll einen Countdown anzeigen, der sekündlich heruntergezählt wird. Die Variable **n** enthält den aktuellen Zählerstand.

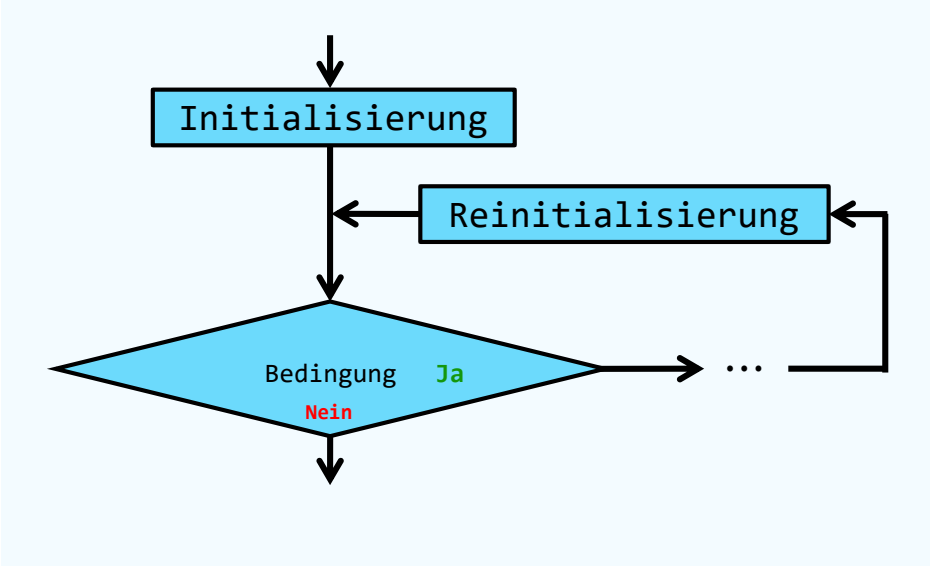
→ Die Bedingung **n** ≠ 0 wird nach jedem Schleifendurchlauf geprüft

→ Gilt zu beginn **n** ≔ 0, so wird die Schleife einmal ausgeführt

```
void countdown(unsigned int n) {
    do {
        delay(1000);
        printf("%03u\r", n);

    } while (n-- != 0);
}
```

Anders als bei der While-Schleife kann zusätzlich bei einer sogenannten **For-Schleife** im **Schleifenkopf** eine Variable (**Laufvariable**) initialisiert und vor jedem Durchlauf nach dem Prüfen der Bedingung **reinitialisiert**




```
for (Initialisierung; Bedingung; Reinitialisierung) { … }
```

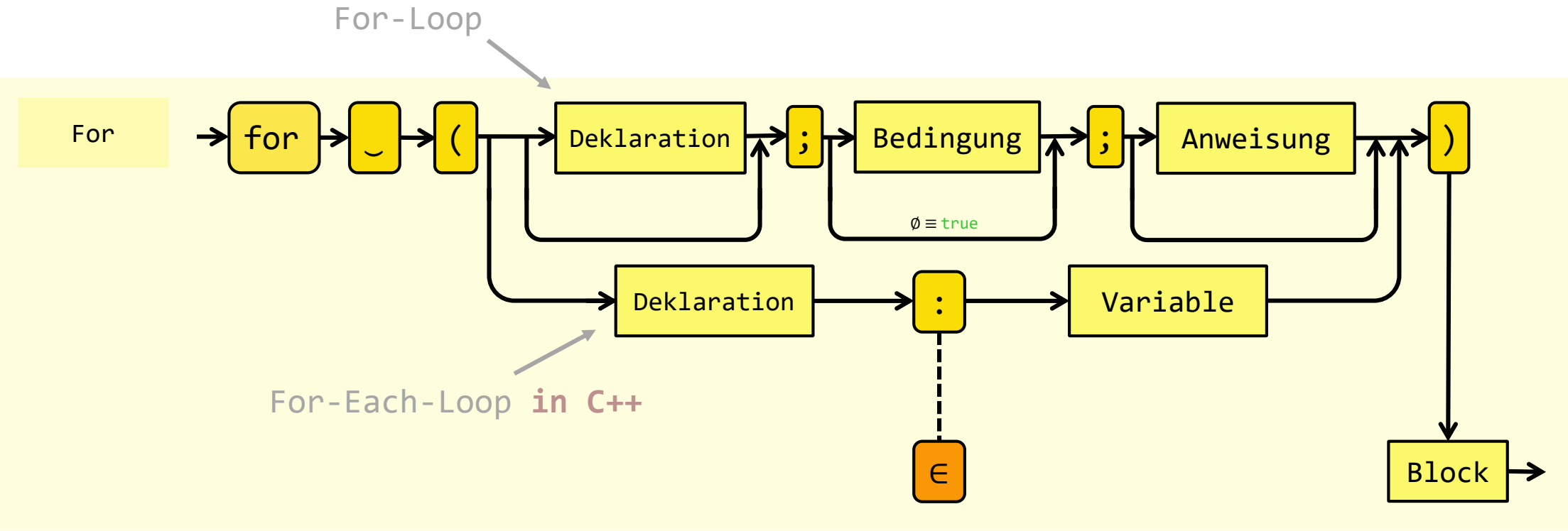
```
void countdown(unsigned int n) {
    for (int k = n; k != 0 ; k--) {
        delay(1000);
        printf("%03u\r", k);
    }
}
```

Das folgende Programm inkrementiert Werte der Elemente eines Arrays:

```
// For-Loop
for (int k = 0; k < n; k++) { arr[k]++; }
```

```
// For-Each-Loop in C++
for (auto &b : array) { b++; }
```

Allgemeiner Aufbau einer typischen **for**-Schleife



- Das Schlüsselwort **`break`** erlaubt eine Schleife abzubrechen
  - Der Programmzähler springt 'hinter' das **Schleifenende**
  - Sollte nur dann eingesetzt werden, wenn es nötig ist
- Mit dem Schlüsselwort **`continue`** wird das Programm am **Schleifenbeginn** fortgesetzt

```
mul(a, b)                                   p := 0
1  for (i := 0; i < 8; i := i + 1) :
2     if (b_0 = 1) : p := p + a
3        b := b >> 1
4        a := a << 1
5  return p
```

Code

```
20 for (int i = 0; i < 8; i = i + 1) {
21   if (b & 1){ result += a;}
22      b >> = 1;
23      a << = 1;
24 }
```

$$\ln(x+1) \approx \sum_{i=1}^{5} (-1)^{i+1} \frac{x^i}{i} \qquad \forall x \in \,]-1,1[$$

Code

```
20 float result = 0.0; term = x;
21 for (int i = 1; i <= 5; ++i) {
22   result += term;
23   term *= -x * i / (i + 1)
24 }
```