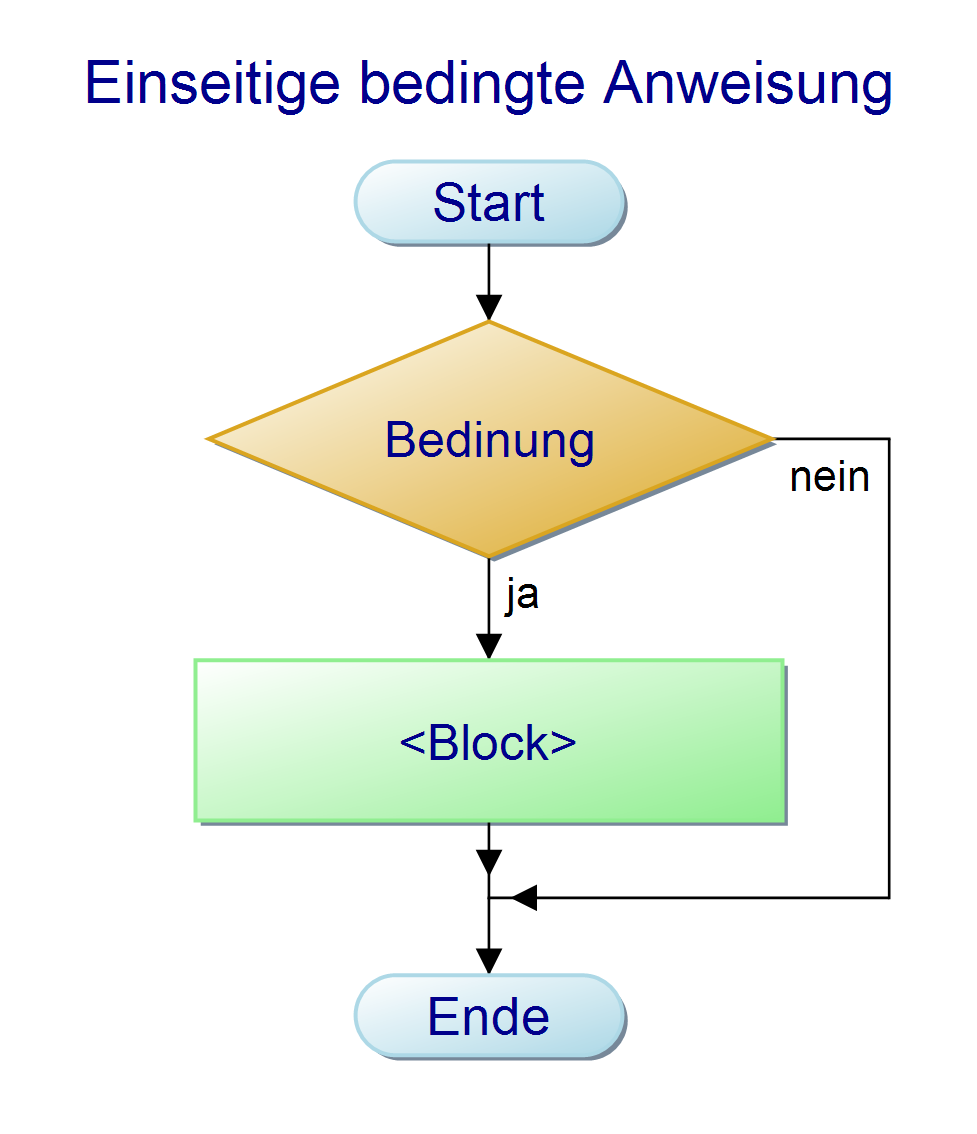
# Bedingte Anweisungen

## Einseitig bedingte Anweisung



Notwendige Infos:

1. eindeutiges Schlüsselwort
2. Bedingung
3. Block mit sauberer Grenze

Herleitung der Syntax aus bisher bekannten Strukturen:

For-Schleife:

for (… <Bed> … ) {

<Block>

}

Syntax der einseitigen bedingten Anweisung:

**if (<Bed>) {**

**<Block>**

**}**

## zweiseitige Bedingte Anweisung:

Wenn wir zwei einseitige bedingte Anweisungen haben,

die sich gegenseitig ausschließen,

wie im Beispiel:

1. divisor == 0
2. divisor != 0

dann ist es unnötig, beide Bedingungen zu prüfen,

weil aus dem Ergebnis der ersten das der zweiten hervorgeht:

1. Ist die erste Bedingung erfüllt,  
   kann die zweite nicht erfüllt sein.
2. Ist die erste Bedingung <nicht> erfüllt,  
   muss die zweite erfüllt sein.

Somit ist das Prüfen der zweiten Bedingung überfüssig.

Wir wollen den überflüssigen Aufwand verhindern

u. suchen eine Struktur,

die - in Abhängikeit von einer Bedingung -

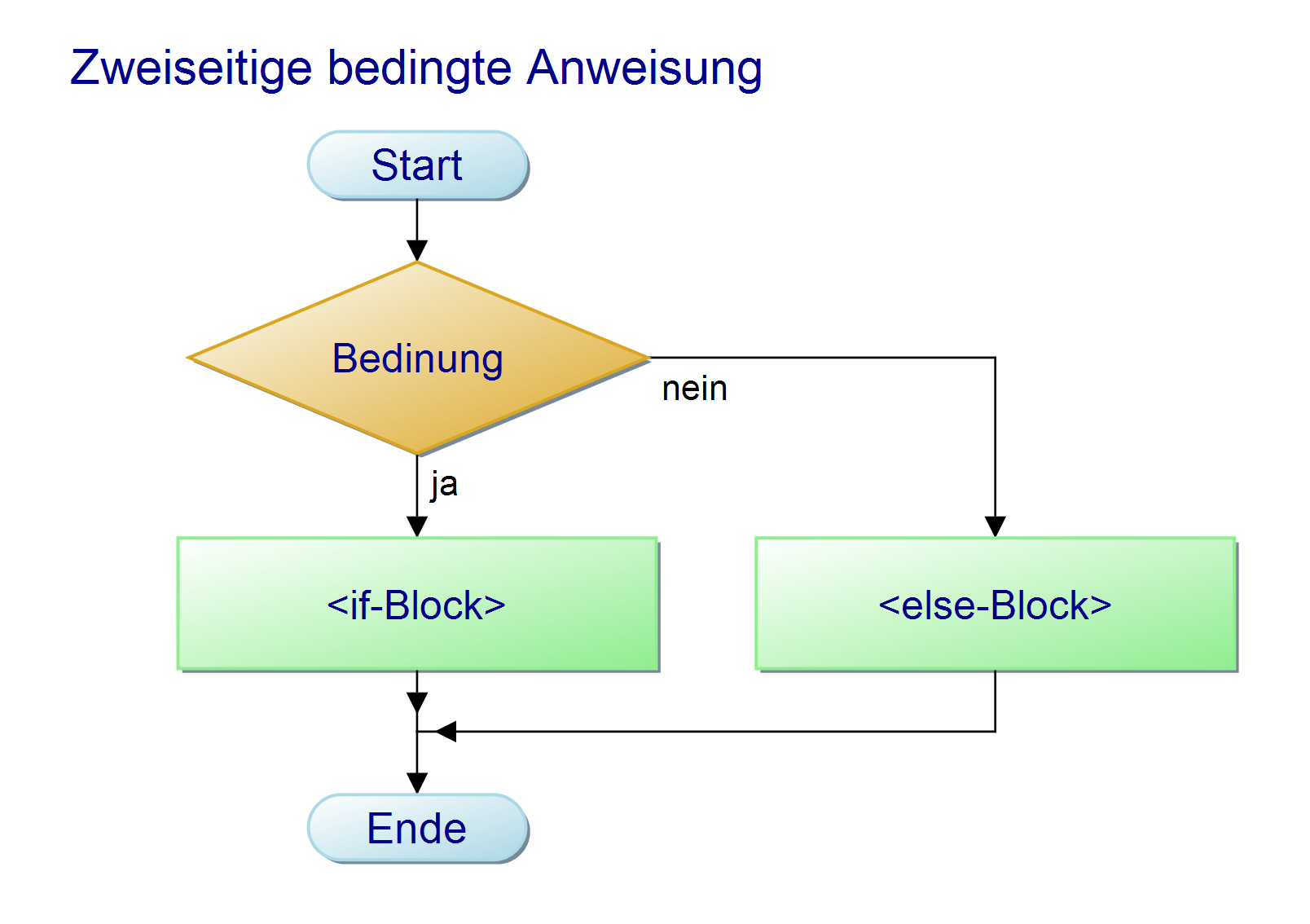
entweder den einen Block ausführt oder den anderen.

Es soll immer genau einer der beiden ausgeführt werden:

Niemals beide, u. auch niemals weder noch.

Dazu erweitern wir den Ablaufplan der einseitig bed. Anw.,

um auf den der zweiseitigen bedingten Anweisung zu kommen:



Was kommt dazu: der else-Block

Weil wir nun zwei Blöcke haben,

vergeben wir unterschiedliche Namen.

Wir erweitern die Syntax wie folgt:

Syntax der zweiseitigen bedingten Anweisung:

**if (<Bed>) {**

**<if-Block>**

**} else {**

**<else-Block>**

**}**

Analyse des Benefits beim Übergang von zwei unabhängigen einseitigen-

in eine zweiseitige Bedingte Anweisung:

1. Reduktion des Quelltextes:  
   if (<Bed>) wird ersetzt durch else:  
   Die Bedingung ist mindestens eine Ziffer lang.  
   Von daher Reduktion des Quelltextes
2. Reduktion der Laufzeit:  
   Es wird nur eine anstatt unnötiger Weise zwei Bedingungen geprüft
3. Gewinn an Übersichtlichkeit:  
   Sind zwei unabhängige komplexe Bedingungen vorhanden,  
   sieht man nicht auf den ersten Blick,  
   ob sie sich gegenseitig ausschließen,  
   oder ob es Fälle gibt,   
   für die beide oder keine der Bedingungen erfüllt ist.  
   Bei der zweiseitigen bed. Anw. ist nur eine Bedingung vorhanden.  
   Aufgrund der Struktur der zweiseitigen bed. Anw. ist klar,

dass nur genau einer der Blöcke ausgeführt wird.

## Kaskade:

Sollen mehr als zwei Blöcke alternativ ausgeführt werden,

kann eine Struktur hergeleitet werden durch "Einsetzen in sich selbst":

In den else-Block einer zweiseitigen bed. Anw.  
wird eine weitere z.b.A. eingesetzt.

So entstehen drei Blöcke u. zwei Bedingungen.

Da wir nun mehrere Bedingungen u. Blöcke haben,  
nummerieren wir sie durch.

Dabei lassen wir der Übesichtlichkeit halber die geschweiften Klammern weg,  
da "Block" schon assoziiert,  
dass geschweifte Klammern gesetzt werden müssen,  
sofern ein Block mehr als eine Anweisung beinhaltet:

if (<Bed\_1>)

<Block\_1>

else

<else-Block> if (<Bed\_2>)

<Block\_2>

else

<else-Block>

Kaskade mit 3 Blöcken u. 2 Bedingungen:

if (<Bed\_1>)

<Block\_1>

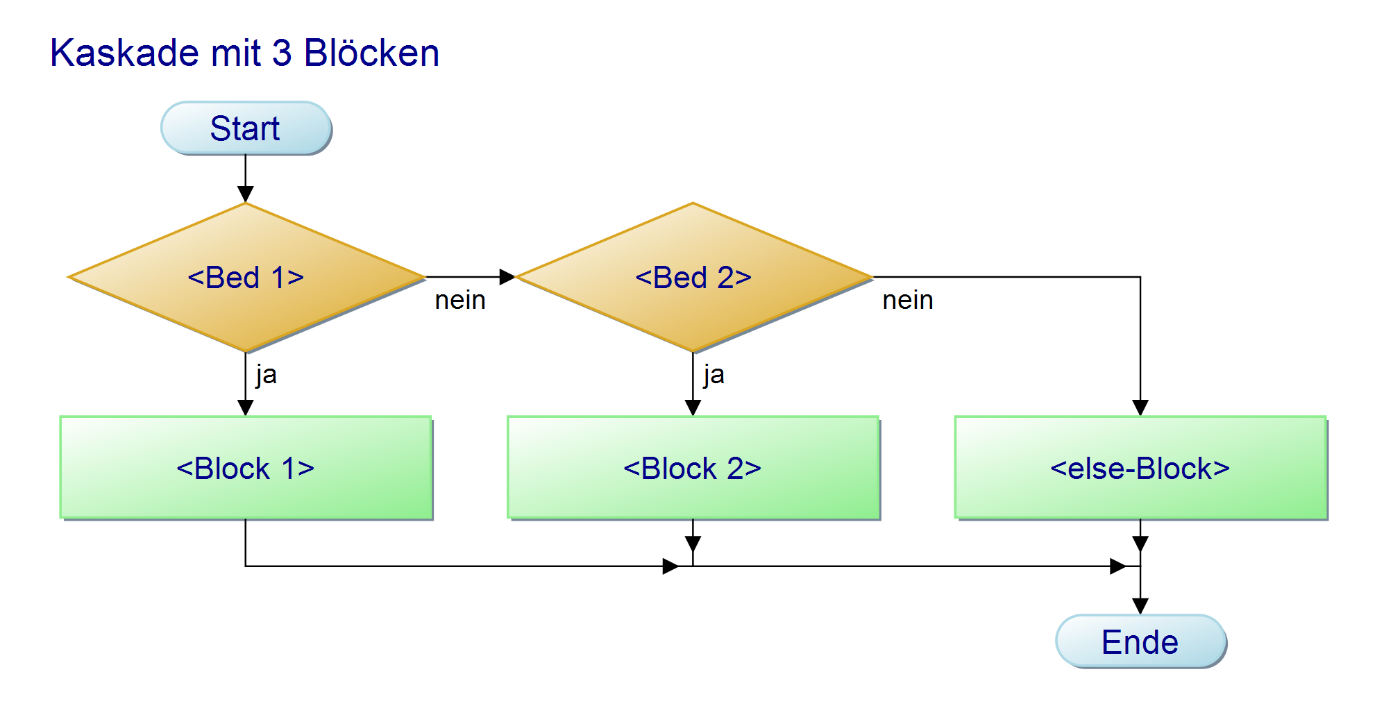
else if (<Bed\_2>)

<Block\_2>

else

<else-Block>

PAP für die Kaskade mit 3 Blöcken:



Zuerst wird Bedingung 1 geprüft.

Ist sie erfüllt, wird Block 1 ausgeführt u. die Struktur ist beendet.

Ist sie nicht erfüllt, wird Bedingung 2 geprüft.

Ist Bed. 2 erfüllt, wird Block 2 ausgeführt u. die Struktur ist beendet.

Ist Bed. 2 nicht erfüllt, wird der else-Block ausgeführt.

Dieser ist optional.

Wäre er nicht da, würde keine Anweisung ausgeführt werden.

Der else-Block hat -sofern vorhanden- eine Sonderrolle als "Sammelbecken"

für alle nicht abgefangenen Fälle.

Dies lässt sich beliebig oft wiederholen,

so dass wir n Bedingungen u. dazu gehörig n Blöck bekommen.

Kaskade für eine beliebige Anzahl von Bedingungen u. Blöcken:

if (<Bed\_1>)

<Block\_1>

else if (<Bed\_2>)

<Block\_2>

…

else if (<Bed\_n>)

<Block\_n>

else

<else-Block>