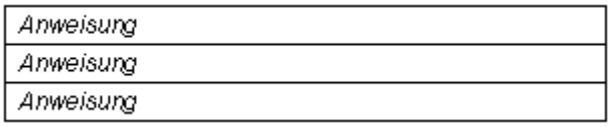
Kontrollstrukturen in Pascal und Struktogramme

In *Pascal* werden Anweisungen (i.d.R. Nachrichten an Objekte) nicht nur sequentiell aufgelistet. Zur Steuerung komplexer Abläufe verwendet man *Kontrollstrukturen* (Wiederholungen, Fallunterscheidungen o.ä). Die Realisierung solcher Programmstrukturen in *Pascal* wird in diesem Dokument erläutert.
Daneben verwenden wir eine heute gebräuchliche Form der grafischen Darstellung von Programmstrukturen, die *Nassi-Shneiderman-Diagramme* oder auch kurz *Struktogramme*. Mit ihnen kann man unabhängig von einer konkreten Programmiersprache die Struktur eines Programms deutlich machen.

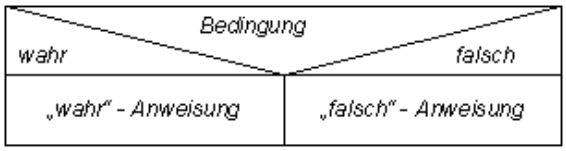
1. Sequenz
Die einfachste Form der Programmstruktur ist die *Sequenz*. Mehrere Anweisungen werden nacheinander ausgeführt. In *Pascal* werden sie jeweils durch ein Semikolon voneinander getrennt. Durch Klammerung mit den Schlüsselwörtern **`begin`** und **`end`** werden sie zu *einer* Anweisung zusammengefasst.



Syntax (Schreibweise) :

```
begin
   <Anweisung>;
   <Anweisung>;
   <Anweisung>;
end
```

2. Ein- und zweiseitige Auswahl
Oft müssen Anweisungen nur unter ganz bestimmten Bedingungen ausgeführt werden. Dazu benötigt man eine *bedingte Anweisung* oder auch *Auswahl (Selektion)* .



Die Bedingung wird zunächst ausgewertet. Ist sie wahr, so wird die "wahr" - Anweisung (oder die "wahr" Anweisungen) ausgeführt. Ist die Bedingung falsch, so wird die "falsch" - Anweisung (oder die "falsch" Anweisungen") ausgeführt. Falls man den "falsch" - Teil nicht benötigt, kann dieser auch weggelassen werden. In diesem Fall spricht man statt von einer *zweiseitigen* von einer *einseitigen Auswahl*.

Syntax

```
if <Bedingung> then
   <Anweisung>
else
   <Anweisung>;
```

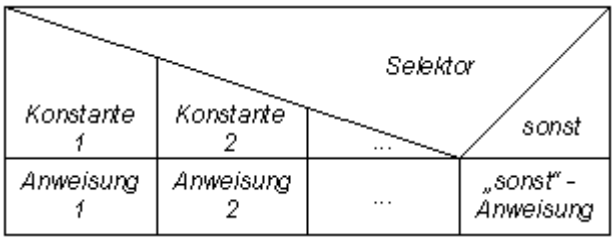
Sollen mehrere Anweisungen - also eine Sequenz - ausgeführt werden, ergibt sich folgende Schreibweise (Syntax):

```
if <Bedingung> then
   begin
      <Anweisung>;
      <Anweisung>;
      <Anweisung>;
   end
else
   begin
      <Anweisung>;
      <Anweisung>;
      <Anweisung>;
   end;
```

Da "**if then else**" eine Anweisung darstellt, darf in beiden Fällen vor dem **else** kein Semikolon stehen.

3. Mehrfache Auswahl

In der bedingten Anweisung können nur maximal zwei Fälle unterschieden werden. Häufig reicht das allerdings nicht aus. Mit Hilfe der *Mehrfach-Auswahl* können mehr Fälle behandelt werden.



Je nachdem mit welcher Konstanten der Selektor übereinstimmt, wird die entsprechende Anweisung ausgeführt. Tritt keiner dieser Fälle ein, wird die "sonst"- Anweisung ausgeführt.

Syntax

```
case  of
   <Konstante> : <Anweisung>;
   <Konstante> : <Anweisung>;
   ...
else
   <Anweisung>;
end;
```

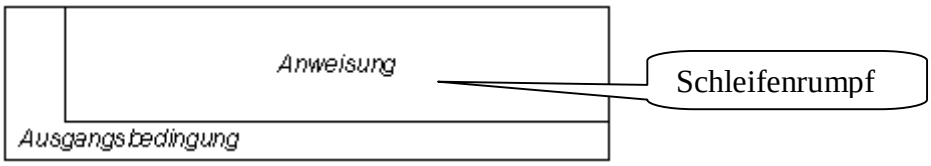
Soll im "sonst" - Fall nichts passieren, kann man diesen auch weglassen.

Syntax

```
case <Selektor> of
  <Konstante> : <Anweisung>;
  <Konstante> : <Anweisung>;
  ...
end;
```

4. Schleife mit Austrittsbedingung

In vielen Problemstellungen ist es notwendig, bestimmte Teile eines Programms wiederholt auszuführen. Eine Form ist die *Schleife mit Austrittsbedingung*.



Die Anweisung wird **solange** wiederholt, **bis** die Ausgangsbedingung wahr wird. Die Anweisung wird also mindestens einmal ausgeführt.

Syntax

```
repeat
  <Anweisung>;
until <Bedingung>;
```

Da durch die Konstruktion in *Pascal* mit den Schlüsslwörtern "**`repeat until`**" die Anweisung bereits geklammert ist, kann man auch im Fall einer Sequenz von Anweisungen (anders als für eine Sequenz vorgesehen) auf die Klammerung mit **`begin`** und **`end`** verzichten.

```
repeat
  <Anweisung>;
  <Anweisung>;
  <Anweisung>;
until <Bedingung>;
```

5. Schleife mit Eintrittsbedingung

Analog zur Schleife mit Ausgangsbedingung gibt es auch eine Schleife mit Eintrittsbedingung. Hier wird vor Ausführung der Anweisung eine Bedingung geprüft.

Solange die Eingangsbedingung wahr ist, wird die Anweisung wiederholt. Falls die Bedingung schon am Anfang falsch ist, wird die Anweisung überhaupt nicht ausgeführt.

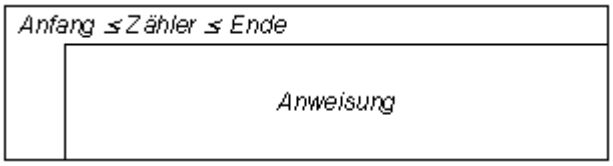
Syntax

```
while <Bedingung> do
   <Anweisung>;
```

`bzw. :`

```
while <Bedingung> do
   begin
      <Anweisung>;
      <Anweisung>;
      <Anweisung>;
   end;
```

6. Zählschleife (behandeln wir etwas später)
Häufig wird die Anzahl der Durchläufe einer Schleife dadurch kontrolliert, dass man sie mitzählt. Diese spezielle Form einer Schleife nennt man Zählschleife.



Dabei verwendet man einen aufwärts oder abwärts laufenden Zähler. Zunächst wird der Zähler auf seinen Anfangswert gesetzt. Nach der Ausführung der Anweisung wird der Zähler dann automatisch um eins erhöht. Dies geschieht solange, bis der Zähler den angegebenen Endwert überschreitet. Da sowohl Anfangs- als auch Endwert (i. A.) vor Ausführung der Wiederholung bekannt sind, steht auch die Anzahl der Wiederholungen von vornherein fest.

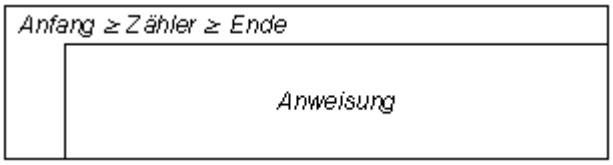
Syntax

```
for <Zaehler> := <Anfangswert> to <Endwert> do
   <Anweisung>;
```

bzw. :

```
for <Zaehler> := <Anfangswert> to <Endwert> do
   begin
      <Anweisung>;
      <Anweisung>;
      <Anweisung>;
   end;
```

Bei der anderen Variante der Zählschleife wird nicht bei jedem Durchlauf um eins hoch-, sondern um eins heruntergezählt:



Syntax

```
for <Zaehler> := <Anfangswert> downto <Endwert> do
  <Anweisung>;
```

bzw. :

```
for <Zaehler> := <Anfangswert> downto <Endwert> do
   begin
      <Anweisung>;
      <Anweisung>;
      <Anweisung>;
   end;
```