9 Steueranweisungen

if ... then
if ... then ... else
case ... of
goto

Gliederung

Der lineare Programmablauf kann durch Verzweigungen und durch Sprünge nach vorwärts/rückwärts verändert werden.

9.1 Die einseitige Verzweigung if ... then

Zweck: Programmablauf in Abhängigkeit von einer Bedingung einseitig verzweigen.

Format 1:

```
if bedingung
   then a1;
a2;
```

Eine Anweisung *a1*
Nächste Anweisung *a2*

Format 2:

```
if bedingung then
   begin
     a11;
     a12;
     ...;
     a1n;
   end;
a2;
```

Mehr als *eine* Anweisung:
Mit "begin" und "end"
zu einem Anweisungs-
block klammern
(compound statement)

Nächste Anweisung *a2*

bedingung: Boolescher Ausdruck. Wenn die Auswertung "True" ergibt, wird die nach "then" stehende Anweisung *a1* (bzw. der Anweisungsblock) ausgeführt, sonst nicht. Anschließend wird mit der nach "if ... then" stehende Anweisung, die schematisch mit *a2* bezeichnet werden soll, fortgesetzt. Wenn die Bedingung nicht zutrifft, wird gleich mit *a2* fortgesetzt. *a1* kann eine beliebige Anweisung sein und inbesondere auch wieder "if ... then" enthalten. Mehr dazu im Kapitel 9.3.

Der boolesche Ausdruck kann beliebige relationale Operatoren (=, <, >, <=, >=, <>) und logische Operatoren (not, and, or, xor) enthalten. Die Priorität der Operatoren ist in Kapitel 8 behandelt (Stufe 1 = höchste Priorität: **not**, Stufe 2: **and**, Stufe 3: **or**, **xor**). Die relationalen Operatoren haben die Prioritätsstufe 4. Eine andere Abarbeitung muß durch Klammersetzung erzwungen werden.

1. Beispiel:

```
if x = 4711 then WriteLn('Anton Huber');
```

schöner:

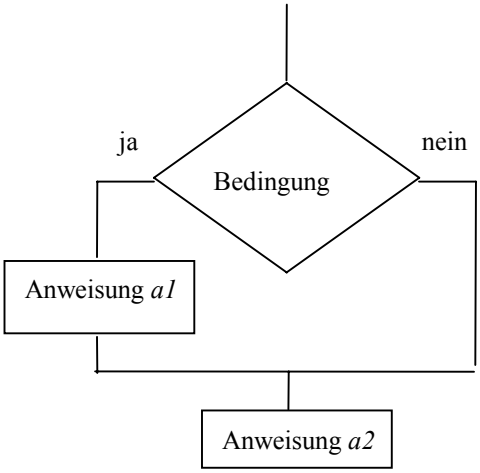
```
if (x = 4711)          { In C Klammerung immer notwendig }
   then WriteLn('Anton Huber');
```

2. Beispiel:

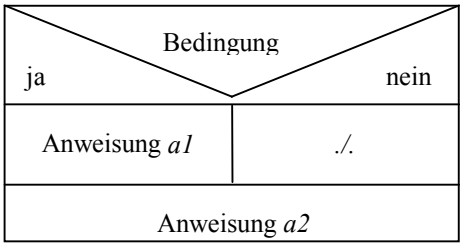
```
if (x = 4711) and (y <> 0) { Klammerung notwendig! }
   then WriteLn('Anton Huber');
```

Graphische Darstellung der einseitigen Verzweigung:

a) im Programmablaufplan PAP (Flußdiagramm):



b) im Struktogramm:




```
program Pas09011; { Demo: Einseitige Programmverzweigung if ..then }

uses
  CRT;
var
  i: Integer;

begin
  ClrScr;
  Write('Geben Sie die Geheimzahl ein: ');
  ReadLn(i);

  if i = 4711
     then WriteLn('Sie haben die Geheimzahl erraten!');

  WriteLn('Drücken Sie die Taste ENTER. Dann erpart man sich in ');
  Write(  'Turbo-Pascal das lästige Fenster-Umschalten.' );
  ReadLn;    { "ReadLn" Warten auf ENTER }
end.
```

9.2 Die zweiseitige Verzweigung if ... then ... else

Zweck: Programmablauf in Abhängigkeit von einer Bedingung zweiseitig verzweigen.

Format:

```
if bedingung
   then a1True   { Kein Semikolon vor diesem "else" }
   else a1False;
a2;              { Nächste Anweisung }
```

Zu *bedingung* siehe 9.1.

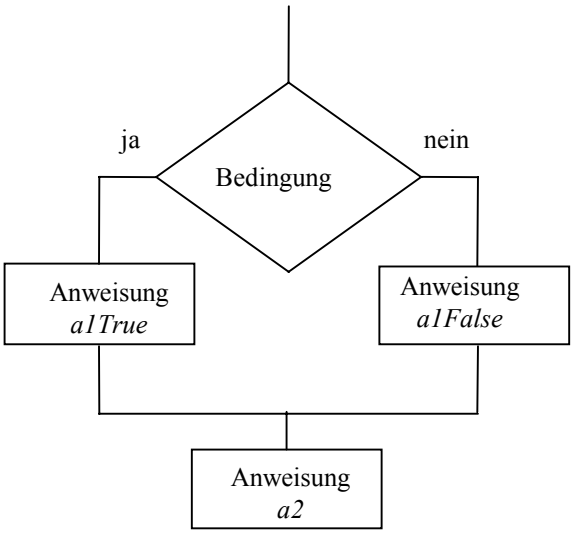
Wenn die Auswertung der Bedingung "True" ergibt, wird *a1True* ausgeführt, anderenfalls *a1False*. Wenn *a1True* oder *a1False* aus mehreren Anweisungen bestehen, dann

wie bei 9.1 mit "begin" und "end" zu einem Anweisungsblock klammern. Die Programmfortsetzung erfolgt nach der Verzweigung mit der folgenden Anweisung *a2*.

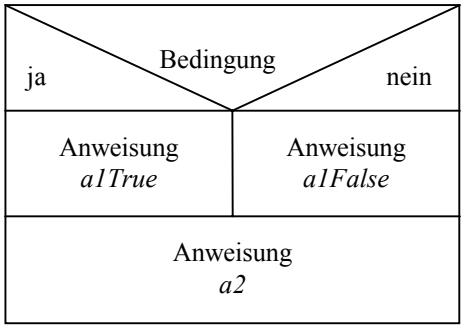
Man beachte, daß vor dem »else«-Zweig einer zweiseitigen Verzweigung kein Semikolon stehen darf!

Graphische Darstellung der zweiseitigen Verzweigung:

a) im Programmablaufplan PAP (Flußdiagramm):



b) im Struktogramm:




```
program Pas09021;  { Demo: Zweiseitige Programmverzweigung
                           "if ... then ... else" }
uses
  CRT;
var
  i: Integer;

begin
  ClrScr;
  Write('Geben Sie die Geheimzahl ein: ');
  ReadLn(i);
  if i = 4711
     then WriteLn('Sie haben die Geheimzahl erraten!')  { kein ";" }
     else begin
            WriteLn(#7);  { Steuerzeichen #7: Bell }
            WriteLn('Sie haben die Geheimzahl vergessen!');
            WriteLn('Das nächste mal zerstöre ich alle Dateien!');
            WriteLn;
          end;
  WriteLn('Drücken Sie die Taste ENTER. Dann erpart man sich in ');
  Write(  'Turbo-Pascal das lästige Fenster-Umschalten.');
  ReadLn;    { "ReadLn" wartet auf ENTER }
```

```
end.
```

9.3 Verschachtelung von Verzweigungen

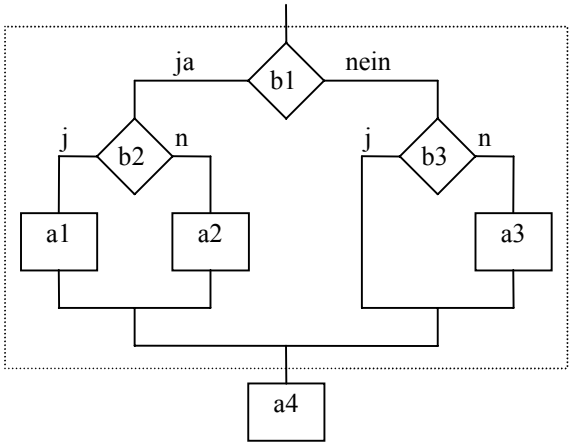
Da die Anweisung nach *then* bzw. nach *else* beliebig sein kann und insbesondere auch wieder eine Bedingung enthalten kann, können ein- und zweiseitige Verzweigungen können beliebig verschachtelt werden.

Dabei gilt: Jedes *else* bezieht sich auf das jeweils letzte *if*, zu dem noch kein else-Zweig angegeben ist.

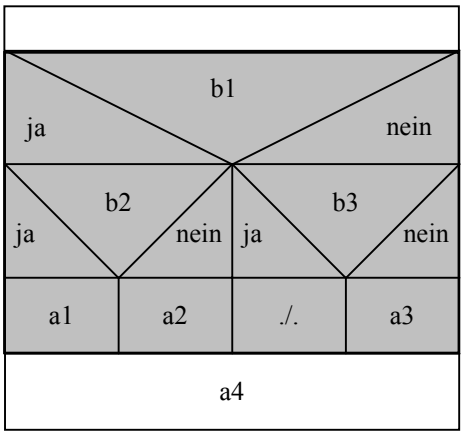
Die folgenden Graphiken zeigen beispielhaft eine derartige Situation als Programmablaufplan und als Struktogramm und die programmtechnische Lösung in schematischer Pascal-Schreibweise.

Dabei steht "*b*" allgemein für eine beliebige Bedingung, "*a*" allgemein für eine beliebige Anweisung, "j" bzw. "ja" für "True" und "n" bzw. "nein" für "False"

a) im Programmablaufplan PAP (Flußdiagramm):



b) im Struktogramm:



c) in schematischer Pascal-Programmierung:

```
if b1
   then if b2
           then a1
           else a2
   else if b3
           then
           else a3;
a4;
```

Wie das schematische Beispiel mit der Bedingung *b3* zeigt, kann auch der then-Zweig entfallen. Klarer wäre aber die Programmierung mit einer Negierung der Bedingung *b3* wie folgt:

```
if b1
   then if b2
           then a1
           else a2
   else if not b3
           then a3;
a4;
```

```
program Pas09031;  { Demo: Verschachtelung von Verzweigungen }
uses
  CRT;

const
  a1 = 'a1';
  a2 = 'a2';
  a3 = 'a3';
  a4 = 'a4';

var
  B1, B2, B3: Boolean;

begin
  ClrScr;
  B1 := False;  { nur für Demo }
  B2 := True;
  B3 := False;

  { ---- Version 1 ------- }
  if B1
     then if B2
             then WriteLn(a1)
             else WriteLn(a2)
     else if B3
             then              { then-Zweig leer, unschön! }
             else WriteLn(a3);

  { ---- Version 2 (Bedingung B3 negiert, ---> klarer ------}
  if B1
     then if B2
             then WriteLn(a1)
             else WriteLn(a2)
     else if not B3
             then WriteLn(a3);
  WriteLn(a4);
  ReadLn;
end.
```

9.4 Selektion mit "case ... of"

Anweisungsfolgen wie z.B.

```
if x = 1 then anweisung1;
if x = 2 then anweisung2;
if x = 3 then anweisung3;
...
if x = N then anweisungN;
```

sind aufwendig in der Schreibweise und ineffizient, da jeder Fall abgeprüft werden muß. Die Umstellung auf zweiseitige Verzweigung führt leicht zu unübersichtlichen Verschachtelungen.

Eine effiziente und klare Programmierung gestattet in diesen Fällen die Selektionsanweisung "case ... of".

Zweck: Auswahl eines Falles aus mehreren Möglichkeiten

Format:
```
case ordinalausdruck of
  konstante1: anweisung1;
  konstante2: anweisung2;
  ....;
  konstanteN: AnweisungN;
[ else anweisung_fuer_restliche_Faelle ];  { optional }
end;
```

Das eingangs geschilderte Problem kann dann wie folgt effizient gelöst werden:

```
case x of
  1: anweisung1;
  2: anweisung2;
  3: anweisung3;
  ...
  N: anweisungN;
end;
```

Wenn ein Fall erkannt und abgearbeitet ist, werden die restlichen Fälle nicht mehr geprüft; die case-Anweisung wird verlassen.

Als Selektor dient ein Ordinal-Ausdruck, dessen Ordinalwert zwischen -32768 und +32767 liegen muß.

Der Datentyp des Selektors kann somit sein: Integer, ShortInt, Byte, Char und Boolean. Teilbereichstypen und Aufzählungstypen sind zulässig, soweit sie der genannten Ordinalwert-Forderung genügen. Ausdrücke mit den Datentypen Word und LongInt sind somit nicht als Selektor zulässig. Die Typen String, Real usw. sind nicht ordinal und somit auch nicht zulässig.

Der *else*-Zweig ist optional. Die eckigen Klammern in der Formatbeschreibung dienen nur als Hinweis auf die Option; sie dürfen im gegebenen Fall nicht eingegeben werden. Vor diesem "case/else" darf ein Semikolon stehen, bekanntlich aber nicht vor dem "else" einer "if/then/else-Konstruktion.

Jede Anweisung kann eine einfache Anweisung (nur *eine* Anweisung) sein oder ein mit *begin* und *end* geklammerter Anweisungsblock mit zwei oder beliebig vielen Anwei-

sungen. Beim optionalen *else*-Zweig kann die Klammerung entfallen; man sollte sie der Einheitlichkeit wegen dennoch vornehmen.

Statt einer Konstanten ist auch eine Konstanten-Liste oder ein Konstanten-Bereich und auch eine Kombination zulässig. Die Listenelemente sind mit einem Komma voneinander zu trennen. Ein Bereich wird durch den Kleinstwert und dem Größtwert dargestellt, dazwischen sind zwei Punkte zu setzen.

```
program Pas09041;  { Demo:  Selektion mit "case ... of"
uses
  CRT;
const
  Escape = #27;    { #27: Ordnungsnummer für ESC nach ASCII }
var
  Zeichen: Char;

begin
  ClrScr;
  repeat
    Write('Drücke eine Taste (Abbruch mit Taste "Esc": ');
    Zeichen := ReadKey;
    if Zeichen = #13
       then Write(' ')      { sonst unschöner Zeilenvorschub }
       else Write(Zeichen);
    case Zeichen of
         '+':       WriteLn('   Plus-Zeichen');
         '-':       WriteLn('   Minus-Zeichen');
         '!', '?':  WriteLn('   Ausrufe- oder Fragezeichen');
         'a'..'z':  WriteLn('   Kleinbuchstabe');
         'A'..'Z':  WriteLn('   Großbuchstabe');
         '0'..'9':  WriteLn('   Ziffernzeichen');
        #13:        WriteLn('   Taste RETURN');
        Escape:     WriteLn('   Taste ESC'); { <--- Semikolon hier
                                               vor diesem "else" zulässig,
                                               aber nicht notwendig.
                                               Bei "if then/else" dagegen
                                               kein Semikolon vor "else" }
      else          WriteLn('   Ein sonstiges Zeichen ');
    end; { von "case ... of" }
  until Zeichen = Escape;
end.
```

9.5 Die Sprunganweisung "goto"

Zweck: Unbedingter oder bedingter Sprung zu einer mit einem Label (Marke) gekennzeichneten Programmstelle.

Format 1: `goto` *`marke`*`;` unbedingter Sprung
Format 2: `if` *`bedingung`* `then goto` *`marke`*`;` bedingter Sprung

Programmsprünge sind in Turbo-Pascal nur innerhalb des gleichen Programmblocks möglich. Man kann also nicht vom Hauptprogramm in eine Routine springen oder umgekehrt und das ist gut so! Mit den Strukturierungsmöglichkeiten von Pascal sollten Programmsprünge nur für Testzwecke oder zur Behandlung von Fehlersituationen

benutzt werden. Die Sprünge können sowohl nach vorwärts, als auch rückwärts gerichtet sein.

Die Sprungmarken müssen in der Label-Deklaration aufgeführt sein. Die Labelbezeichner können beliebige Bezeichner, aber auch Ganzzahlen aus dem Bereich 0 bis 9999 sein. An der Sprungstelle muß dem Labelbezeichner unbedingt ein Doppelpunkt nachgesetzt werden. Nichtbenutzte Labels werden vom Compiler nicht beanstandet.

```
program Pas09051;  { Demo "goto-Anweisung" }
                   { GOTO-Programme sind verdammenswert! }
uses
  CRT;

label
  Fall1, Fall2, Ende;

var
  x: Integer;

begin
  ClrScr;
  Write('Eingabe Integer-Zahl: ');
  ReadLn(x);

  if (x = 4711)
     then goto Fall1                { Bedingter Sprung   }
     else goto Fall2;

  Fall1:                            { Doppelpunkt nach einem Label }
  WriteLn('x hat den Wert 4711');
  goto Ende;                        { Unbedingter Sprung }

  Fall2:                            { Das 2. Label }
  WriteLn('x hat nicht den Wert 4711');
  goto Ende;                        { Überflüssige "goto"-Anweisung }

  Ende:                             { Das 3. Label }

  ReadLn;                           { Wartet auf Tastendruck ENTER }
end.
```