10 Wiederholungsanweisungen

repeat/until-Schleifen
while-Schleifen
for-Schleifen
Break und Continue

Gliederung

Pascal kennt drei Wiederholungsanweisungen (Schleifen):

- **repeat**/**until**
- **while**
- **for**

Die **repeat**/**until**-Schleife und die **while**-Schleife werden dann eingesetzt, wenn die Anzahl der Wiederholungen *vor* Ausführung der Schleife *nicht* bekannt ist; die **for**-Schleife dagegen dann, wenn die Anzahl der Wiederholungen bekannt ist. Bei den ersten beiden Schleifentypen wird die Wiederholung von einer Bedingung abhängig gemacht.

10.1 Die repeat/until-Schleife

Bei *repeat/until* wird die Schleife solange wiederholt, bis eine am ***Ende*** des Schleifenkörpers stehende ***Bedingung wahr (True) wird***. Diese Schleife wird somit mindestens einmal durchlaufen.

Format:

```
repeat
  Anweisung_1;
  Anweisung_2;
   .....
  Anweisung_n;
until Bedingung;
```

Schleifenkörper

Bedingung Beliebig komplexer boolescher Ausdruck
Anweisung Beliebige Anweisung

Der Schleifenkörper kann beliebig viele Anweisungen enthalten (0, 1, 2, ... n). Im Gegensatz zu **while**- und **for**-Schleifen ist bei **repeat/until** *keine* Blockung mit **begin/end** notwendig, wenn der **repeat/until**-Schleifenkörper mehr als eine Anweisung enthält.

Wenn die *Bedingung* nie wahr werden kann, dann hat man in sehr lobenswerter Weise eine Endlosschleife programmiert; das Programm "hängt sich auf". Solange man sich in der Entwicklungsumgebung von Turbo-Pascal befindet, kann man einen Rettungsversuch mit "Strg+UntBr" unternehmen. Weitere Brutalo-Methoden: Unter DOS mit "Strg+Alt+Entf" Warmstart durchführen oder unter Windows Anwendung beenden. Im Worst-Case Rechner ausschalten und neu starten (Kaltstart). Auch bei der while-Schleife besteht die Gefahr einer Endlosschleife!

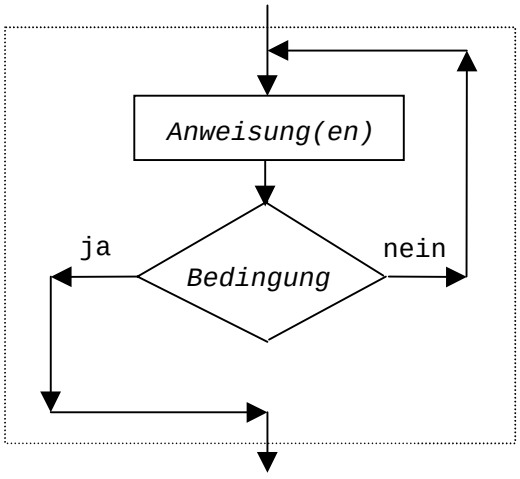
Der Schleifenkörper kann beliebige Anweisungen enthalten; somit z.B. auch weitere *repeat/until*-Schleifen oder andere Schleifen.

Beispiele:

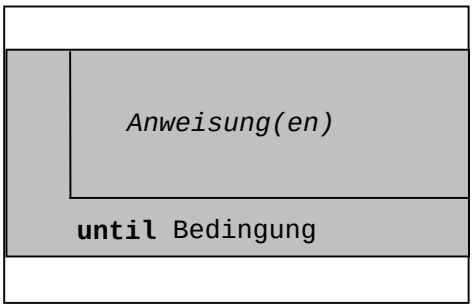
```
.....
x := ....; { Real }
.....
repeat
  x := x + 0.4711;
until x > 47.11;   { Wiederholung bis x > 47.11 }
.....
.....
repeat
until True;        { Leerschleife, 1 "Durchlauf" }
.....
repeat
until False;       { Endlosschleife !!!!! }
.....              { "False" kann nie "True" werden }
.....
```

Graphische Darstellung der repeat/until-Schleife:

a) im Programmablaufplan (PAP)



b) im Struktogramm



Konkrete Anwendungen der **repeat/until**-Schleife zeigt das folgende Demo-Programm:

```
program Pas10011;  { Demo: repeat/until-Schleife }

uses
  CRT;

var
  i, Summe: Integer;
  Zeichen:  Char;

begin
  ClrScr;
{ ------------------- Beispiel 1: ---------------------------- }
  Write('Beispiel 1: Drücke eine der Tasten "j" oder "n": ');
```

```
{ ------------------- Beispiel 1: ---------------------------- }
  Write('Beispiel 1: Drücke eine der Tasten "j" oder "n": ');
  repeat
    Zeichen := ReadKey;
  until (Zeichen = 'j') or (Zeichen = 'n');
  WriteLn(Zeichen);

  { ------------------- Beispiel 2: ---------------------------- }
  Write(#13#10, 'Beispiel 2: Drücke beliebige Taste: ');
  repeat
  until KeyPressed;
  WriteLn;

  { ------------------- Beispiel 3: ---------------------------- }
  Write(#13#10, 'Beispiel 3: ');
  i     := 1;
  Summe := 0;  { Initialwert 0 für Summation }
  repeat
    Write(i, ' ');  { ¦Beispiel 3: 1 2 3 4 5 }
    Summe := Summe + i;
    Inc(i);
     { Bei "repeat/until-" Schleifen ist keine Blockung mit
       "begin/end" notwendig, wenn der Schleifenkörper mehr
       als eine Anweisung enthält, im Gegensatz zu "while-"
       und "for-" Schleifen. }
  until (i > 10) or (Summe >= 14);
  WriteLn('  Die Summe: ', Summe);
  { ¦Beispiel 3: 1 2 3 4 5  Die Summe: 15    }
  { ------------------------------------------------------------- }

  Write(#13#10, 'Beenden mit Taste "Esc": ');
  repeat
  until ReadKey = #27;
end.
```

10.2 Die while-Schleife

Format 1:

```
while Bedingung do
  Anweisung;
```

Format 2:

```
while Bedingung do
  begin
    Anweisung_1;      ┐
    Anweisung_2;      │
    .....             ├── Schleifenkörper
    Anweisung_n;      ┘
  end;
```

Bedingung Beliebig komplexer boolescher Ausdruck
Anweisung Beliebige Anweisung

Wenn der **while**-Schleifenkörper mehr als *eine* Anweisung enthält, ist unbedingt eine Blockung mit **begin/end** notwendig. Man beachte das resevierte Wort **do** nach der "*Bedingung*".

Wie bei der *repeat/until*-Schleife besteht auch bei der *while*-Schleife die große Gefahr, daß man eine Endlosschleife erhält, nämlich dann, wenn die *Bedingung* immer wahr bleibt. Siehe die entsprechenden Ausführungen bei der *repeat*/*until*-Schleife.

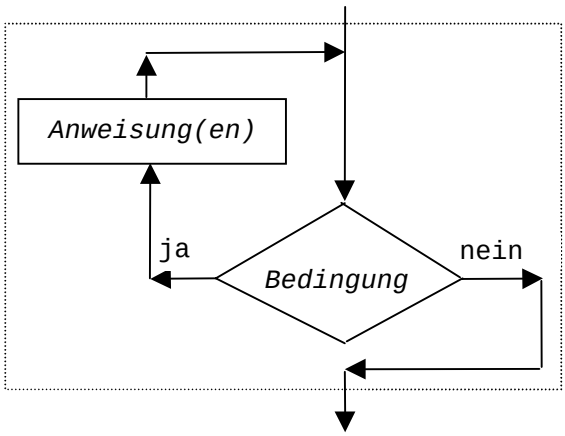
Die *while*-Schleife kann beliebige Anweisungen enthalten, darunter natürlich auch weitere *while*- oder andere Schleifen.

Beispiele:

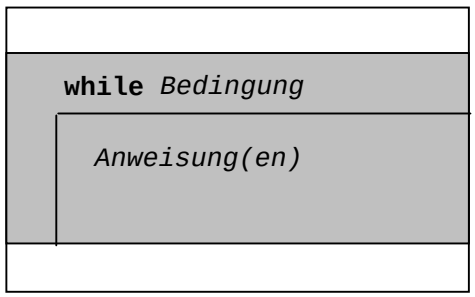
```
.....
x := 47.11;
.....
while x < 4711 do
  x := x + 0.4711;
.....
.....
while False do;   { Leerschleife, kein  "Durchlauf" }
.....
.....
while True do;    { Endlosschleife !!!!! }
.....             { "True" kann nie "False" werden }
.....
```

Graphische Darstellung der while-Schleife:

a) im Programmablaufplan



b) im Struktogramm



Das folgende Demo-Programm zeigt verschiedene Anwendungen der **while**-Schleife:

```
program Pas10021;    { Demo: while-Schleifen }

uses
  CRT;   { Für: ClrScr, Delay, KeyPressed und ReadKey }
var
  x, Delta_x : Real;  { Man ändere später auf "Double", siehe unten }
  Zeichen:     Char;
  Zahl:        Integer;
begin
  ClrScr;
  { ------------------- Beispiel 1: ------------------------------- }
  WriteLn(#13#10, 'Beispiel 1');
  x       := 0.6;
  Delta_x := 0.1;     { Wenn "while-" und "for-" Schleifen mehr     }
  while x <= 1.0 do   { als eine Anweisung im Schleifenkörper, dann }
    begin             { ist unbedingt mit "begin/end" zu blocken.   }
      Write(x:6:2);       { ¦  0.60  0.70  0.80  0.90  1.00  }
      x := x + Delta_x;   { Läuft bei "Real" nur bis 0.90,   }
    end;                  { erst mit "Double" bis 1.00 !!!   }
  WriteLn;
  { ------------------- Beispiel 2: ------------------------------- }
  WriteLn(#13#10, 'Beispiel 2');
  Zeichen := 'a';
  while Zeichen <= 'e' do
    begin
      Write(Zeichen);         { ¦abcde        }
      Inc(Zeichen);
    end;
  WriteLn(#13#10, Zeichen);   { ¦f            }
  { ------------------- Beispiel 3: ------------------------------- }
  WriteLn(#13#10, 'Beispiel 3');
  Zahl := 4711;
  while  Zahl > 4711  do      { Diese Schleife wird mit den vor- }
    Inc(Zahl);                { liegenden Daten übersprungen     }
  WriteLn(Zahl);              { ¦4711        }
  { ------------------- Beispiel 4: Tastaturpuffer leeren ---------- }
  WriteLn(#13#10, 'Beispiel 4');
  Delay(1000);   { Warteschleife 1000 ms, nur für Demo }
  while KeyPressed do    { Wenn kein Leeren des Tastaturpuffers, }
    Zeichen := ReadKey;  { dann können Tastaturspielereien bei   }
                         { längeren Programmlaufzeiten zu uner-  }
                         { wünschten Programmabläufen führen     }
  Write('Eingabe "0" oder "1": ');
  repeat
    Zeichen := ReadKey;
  until (Zeichen = '0') or (Zeichen = '1');
  if Zeichen = '0'
     then WriteLn(#13#10, '0: Sie haben 1 Mio DM Bankschulden')
     else WriteLn(#13#10, '1: Sie haben 1 Mio DM im Lotto gewonnen');

  { ------------------- Beispiel 5: Tastaturpuffer nicht leeren ---- }
```

```
  WriteLn(#13#10, 'Beispiel 5');
  Delay(1000);   { Warteschleife 1000 ms, nur für Demo }
  Write('Eingabe "0" oder "1": ');
  repeat
    Zeichen := ReadKey;           { So nicht !!!!!!!!!!!!!!! }
  until (Zeichen = '0') or (Zeichen = '1');
  if Zeichen = '0'
     then WriteLn(#13#10, '0: Sie haben 1 Mio DM Bankschulden')
     else WriteLn(#13#10, '1: Sie haben 1 Mio DM im Lotto gewonnen');
  { ------------------------------------------------------------- }
  Write(#13#10#7, 'Beenden mit Taste "Esc": ');
  while KeyPressed do
    Zeichen := ReadKey;
  repeat
  until ReadKey = #27;
end.
```

10.3 Die for-Schleife

Die *for*-Schleife (auch Zählschleife genannt) wird dann eingesetzt, wenn die Anzahl der Wiederholungen bekannt ist.

Die *for*-Schleife benutzt eine *Laufvariable*, die von einem *Anfangswert* schrittweise bis zu einem *Endwert* erhöht (inkrementiert) wird oder von einem *Endwert* schrittweise bis zu einem *Anfangswert* erniedrigt (dekrementiert) wird.

Format 1:
```
for Laufvariable := Anfangswert to Endwert do
  Anweisung;
```

Format 2:
```
for Laufvariable := Anfangswert to Endwert do
  begin
    Anweisung_1;
    Anweisung_2;
    .....
    Anweisung_n;
  end;
```
Schleifenkörper

Format 3:
```
for Laufvariable := Endwert downto Anfangswert do
  Anweisung; { Wenn mehrere Anweisungen: "begin/end" }
```

Laufvariable	Variable mit ordinalem Datentyp
Anfangswert, *Endwert*	Beliebiger Ausdruck (Konstante, Variable, Term) mit gleichem ordinalen Datentyp wie *Laufvariable*.
Anweisung	Beliebige Anweisung

Wenn der **for**-Schleifenkörper mehr als eine Anweisung enthält, ist unbedingt eine Blockung mit **begin**/**end** notwendig.

Man beachte die resevierten Wört **do** bzw. **downto** nach "*Endwert*" bzw. "*Anfangswert*".

- Beim Format 1 und 2 der *for*-Schleife wird die *Laufvariable* nach jedem Durchlauf um eine ordinale Einheit erhöht (positive Schrittweite, aufsteigende Schleife). Wenn der *Endwert* erreicht ist, wird die Schleife ein letztes mal durchlaufen.
- Beim Format 3 ist der Vorgang ähnlich; der Unterschied besteht lediglich darin, daß die Schleife vom *Endwert* bis zum *Anfangswert* abgearbeitet wird (negative Schrittweite, absteigende Schleife).

Wenn der *Anfangswert* gleich oder größer ist als der *Endwert*, dann werden die Anweisungen in der Schleife nicht ausgeführt. Die *for*-Schleife verhält sich in dieser Hinsicht ähnlich wie die *while*-Schleife.

Die *for*-Schleifen sind ein Spezialfall der *while*-Schleifen und können immer durch diese ersetzt werden; dennoch sollte man es nicht tun, da *for*-Schleifen wesentlich schneller abgearbeitet werden.

Die *for*-Schleife ist nicht nur für Integer-Typen, sondern für alle Ordinaltypen definiert (*Integer*, *Byte*, *Word*, *ShortInt*, *LongInt*, *Char*, *Boolean*, Aufzählungstypen und Teilbereichstypen).

Der Datentyp *Real* ist bekanntlich nicht ordinal und kann somit nicht für *Laufvariable*, *Anfangswert* und *Endwert* in Pascal-**for**-Schleifen verwendet werden, im Gegensatz zu BASIC und PostScript.

Die *Laufvariable* kann im Schleifenkörper verändert werden, z.B. mit:

```
"Laufvariable := Laufvariable + 3;"
```

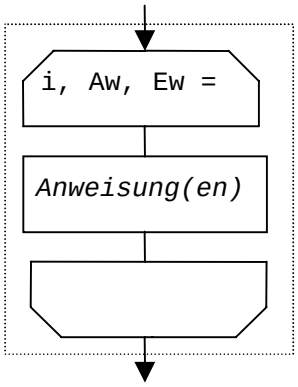
Anfangswert und *Endwert* können dagegen im Schleifenkörper nicht verändert werden, auch wenn sie als Variablen in der *for*-Schleife aufgeführt sind; der Compiler bringt aber keine Fehlermeldung!

Da der Schleifenkörper wie bei allen Schleifen beliebige Anweisungen enthalten kann, können *for*-Schleifen auch verschachtelt werden (*for*-Schleife in einer *for*-Schleife). Beim Bearbeiten von Matrizen sind Doppel-*for*-Schleifen sehr praktisch.

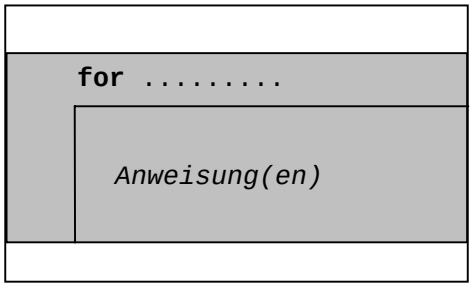
Die Gefahr einer Endlosschleife besteht bei *for*-Schleifen praktisch nicht, es sei denn, daß man bei einer aufsteigenden *for*-Schleife die *Laufvariable* im Schleifenkörper dekrementiert oder bei einer absteigenden inkrementiert.

Graphische Darstellung der for-Schleife:

a) im Programmablaufplan



b) im Struktogramm
(ähnlich wie while-Schleife)



Das folgende Demo-Programm zeigt verschiedene Anwendungen der *for*-Schleife:

```
program Pas10031;   { Demo: for-Schleifen  }
uses
  CRT;
const
  n    =  4;
  iMin =  1;
  iMax = 10;
var
  i, j, Summe: Integer;
  Buchstabe:   Char;
  Hund:        Boolean;
  Farbe:       (Blau, Gruen, Gelb, Rot);    { Aufzählungstyp }
begin
  ClrScr;
{ --------------- Beispiel 1: for-Schleife mit Integer ----------- }
  Write(#13#10, 'Beispiel 1: ');
  for i := 5 to 10  do
    Write(i, ' ');  { ¦Beispiel 1: 5 6 7 8 9 10   }
  WriteLn;
  { --------------- Beispiel 2: for-Schleife mit Integer ----------- }
  Write(#13#10, 'Beispiel 2: ');
  for i := 10 downto 5  do
    Write(i, ' ');   { ¦Beispiel 2: 10 9 8 7 6 5   }
  WriteLn;
  { --------------- Beispiel 3: for-Schleife mit Integer ----------- }
  Summe := 0;  { Initialwert 0 für Summation, für Produktbildung: .  }
  for i := 1 to n do
    Summe := Summe + i; { Typischer Fall von Computermißbrauch }
  WriteLn(#13#10, 'Beispiel 3: Summe 1 bis ', n, ': ', Summe,
                  '. Oder nach "n*(n + 1)/2": ', n*(n + 1) div 2);
  { ¦Beispiel 3: Summe 1 bis 4: 10. Oder nach "n*(n + 1)/2": 10  }
{ --------------- Beispiel 4: for-Schleife mit Integer ----------- }
  Write(#13#10, 'Beispiel 4: ');
```

```
  for i := (iMin + iMax) div 2  to  iMax - 1 do
    Write(i, ' ');      { ¦Beispiel 4: 5 6 7 8 9   }
  WriteLn;
  { --------------- Beispiel 5: for-Schleife mit Integer ----------- }
  { Die Laufvariable darf in der Schleife verändert werden,
    nicht aber Anfangswert und Endwert  }
  Write(#13#10, 'Beispiel 5: ');
  for i := 1 to 10 do
    begin
      Write(i, ' ');    { ¦Beispiel 5: 1 3 5 7 9   }
      Inc(i);           { Bei "Dec(i)" ergäbe sich }
    end;                { eine Endlosschleife!     }
  WriteLn;
  { --------------- Beispiel 6: for-Schleife mit Char -------------- }
  Write(#13#10, 'Beispiel 6: ');
  for Buchstabe := 'a' to  'k' do
    Write(Buchstabe);   { ¦Beispiel 6: abcdefghijk    }
  WriteLn;
{ --------------- Beispiel 7: for-Schleife mit Boolean ----------- }
  Write(#13#10, 'Beispiel 7: ');
  for Hund := True downto False do
    Write(Hund, ' ');   { ¦Beispiel 7: TRUE FALSE }
  { Das ist der Beweis: Es gibt echte und falsche Hunde! }
  WriteLn;
  { --------------- Beispiel 8: for-Schleife mit Aufzählungstyp ---- }
  Write(#13#10, 'Beispiel 8: ');  { Hinweis: Aufzählungstypen können }
                                  { nicht eingegeben oder ausgegeben werden }
  for Farbe := Blau to Rot do
    case Farbe  of
      Blau:  Write('blau '); { ¦Beispiel 8: blau              }
      Gruen: Write('grün '); { ¦Beispiel 8: blau grün         }
      Gelb:  Write('gelb '); { ¦Beispiel 8: blau grün gelb    }
      Rot:   Write('rot ');  { ¦Beispiel 8: blau grün gelb rot }
    end;
  WriteLn;
  { --------------- Beispiel 9: Doppel-for-Schleife mit Integer ---- }
  WriteLn(#13#10, 'Beispiel 9: Multiplikationstafel ',
                  'mit Doppelschleife');
  Write(' ':6);
  for j := 1 to 5 do       { einfache Schleife }
    Write(j:2, ': ');
  WriteLn;
  for i := 1 to 3  do     { Zeile i.  Äußere Schleife    }
    begin                 {           der Doppelschleife }
      Write(i:2, ': ');
      for j := 1 to 5 do  { Spalte j. Innere Schleife    }
        Write(i*j:4);     {           der Doppelschleife }
      WriteLn;
    end;
  WriteLn;
  { ¦Beispiel 9: Multiplikationstafel mit Doppelschleife }
  { ¦       1:  2:  3:  4:  5:                           }
  { ¦ 1:    1   2   3   4   5                            }
  { ¦ 2:    2   4   6   8  10                            }
  { ¦ 3:    3   6   9  12  15                            }
```

```
{ ------------------------------------------------------------ }
  Write(#13#10, 'Beenden mit beliebigem Tastendruck ... ');
  repeat                { Besser: Vorher noch   }
  until ReadKey <> '';  { Tastaturpuffer leeren }
end.
```

10.4 Vorzeitiger Schleifenabbruch mit Break Wiederholung von Schleifen mit Continue

Zu Break:

Gelegentlich besteht das Problem, daß eine Schleife (*repeat/until*, *while* oder *for*) in Abhängigkeit von einer Bedingung vorzeitig beendet werden soll. In den neueren Versionen von Turbo-Pascal steht für dieses Problem mit der Standardprozedur Break eine elegante Lösung zur Verfügung. Grundsätzlich kann dieses Problem bei allen Schleifentypen auch mit anderen Mitteln und auch ohne Verwendung des (verpönten) *goto*-Sprungs gelöst werden. Die folgende Sequenz soll nur eine Vorstellung geben, wie die Lösung ohne Break beispielsweise in einer for-Schleife aussehen könnte:

```
    .....
    for Laufvariable := Anfangswert to Endwert do
      begin
        { Teil 1 des Schleifenkörpers }
        .....
        if Abbruchbedingung
          then Laufvariable := Endwert
          else begin
                 { restlicher Schleifenkörper }
                 .....
               end;
      end;
    { Anweisung nach der Schleife }
```

Mit Break-Lösung sieht wesentlich eleganter aus:.

```
    ....
    for Laufvariable := Anfangswert to Endwert do
      begin
        { Teil 1 des Schleifenkörpers }
        .....
        if Abbruchbedingung
           then Break;
        { restlicher Schleifenkörper }
        .....
      end;
    { Anweisungung nach der Schleife }
```

Die Verwendung von Break ist nicht an eine Bedingung gebunden; ein "unbedingtes" Break ergibt aber keinen praktischen Sinn. Das folgende Demo-Programm zeigt eine konkrete Anwendung der Break-Prozedur in einer repeat/until-Schleife:

```
program Pas10041;  { Demo: Vorzeitiger Schleifenabbruch mit "Break" }
uses               { 27050493 }
  CRT;
const
  Esc = #27;       { Zeichen #27: Escape }
var
  Ch: Char;
begin
  ClrScr;
  repeat
    Write('Eingabe Zeichen, Ende mit Taste "Esc": ');
    Ch := ReadKey;
    if Ch = Esc     { Mit "Break" können alle Schleifen }
       then Break;  { vorzeitig abgebrochen werden.     }
    GotoXY(WhereX, WhereY);
    WriteLn('Eingabezeichen: ', Ch);
  until False;  { Wäre ohne "Break" eine Endlosschleife }
  Write(#7);
end.
```

Zu Continue:

Gelegentlich besteht die Aufgabe, daß eine Schleife in Abhängigkeit einer Bedingung nicht weiter abgearbeitet werden soll, sondern vom Anfang an wiederholt werden soll. Auch dieses Problem kann mit normalen Pascal-Mitteln gelöst werden; mit der Standardprozedur "Continue" aber wesentlich eleganter. Wie *Break* kann auch *Continue* ohne Bedingung verwendet werden, was aber keinen praktischen Sinn ergibt. Das folgende Demo-Programm zeigt die Anwendung von Continue in einer repeat/until-Schleife:

```
program Pas10042;  { Demo: Schleifenwiederholung mit "Continue" }
uses               { 37030492 }
  CRT;
var
  Ch: Char;
begin
  ClrScr;
  WriteLn('Demo "Continue": Das Programm nimmt alle Zeichen an,');
  WriteLn('                 zeigt aber nur Großbuchstaben an');
  repeat
    Write(#13#10, 'Eingabe Zeichen, Ende mit Taste "Z": ');
    Ch := ReadKey;
    if not (Ch in ['A'..'Z']) { Mit "Continue" können alle Schleifen }
       then Continue;         { von Anfang an wiederholt werden.     }
    GotoXY(38, WhereY);
    WriteLn('Großbuchstabe: ', Ch);
    { Hier könnten       }
    { noch viele         }
    { Anweisungen stehen }
  until Ch = 'Z';
end.
```