Pascal-Kurs für Anfänger:

Teil 2

Jetzt zeigt sich Pascal von der besten Seite: Programme können einfach, klar und problemnah formuliert werden. Pascal hat die Anweisungen dafür.

Dieser zweite Teil unseres Pascal-Kurses stellt die Anweisungen Goto und For, die If-Then-Else-Schleife, die Repeat- und die While-Schleife vor. Danach geht er auf die Verbundanweisung (Begin ... End) ein, mit deren Hilfe man einen Block aus Anweisungen zu einer einzigen Anweisung zusammenschließen kann.

Bevor ein Programm ablaufen kann, muß es in den Computer eingegeben werden. Beim Oxford- und beim Profi-Pascal-Compiler wird ein Editor verwendet, der dem Basic-Editor ziemlich gleicht. Oxford-Pascal verwendet weitgehend die Funktionen des Basic-Editors. Vor jeder Zeile des Pascal-Programms muß eine Zeilennummer stehen, die aber innerhalb von Pascal keinerlei Bedeutung hat. Profi-Pascal verwendet einen eigenen Editor, der ebenfalls zeilenorientiert arbeitet. Er ist in Pascal geschrieben und kann von einem fortgeschrittenen Programmierer geändert werden — der Quellcode wird mitgeliefert.

Nach der Eingabe sollte das Quellprogramms auf der Diskette abgespeichert werden. Bei Profi-Pascal ist dies unbedingt nötig, da der Compiler gesondert in den Arbeitsspeicher geladen wird. Der Compiler übersetzt das Programm und gibt den Objekt-Code auf die Diskette aus. Syntax-Fehler werden dem Benutzer auf dem Bildschirm mitgeteilt. Der Compiler wartet dann darauf, daß der Programmierer die Leertaste drückt, damit die Übersetzung fortgesetzt werden kann. Die Stop-Taste führt dazu, daß in den Editor verzweigt wird. Nach einer fehlerfreien Übersetzung kann das Programm gestartet und ausgeführt werden.

Bei Oxford-Pascal wird zwischen einem Resident-Modus und einem Disk-Modus unterschieden. Der Vorgang beim Disk-Modus gleicht dem Verfahren bei Profi-Pascal. Beim Resident-Modus kann direkt übersetzt und ausgeführt werden, ohne daß auf eine Diskette zugegriffen wird. Ein Abspeichern des Programms ist nicht notwendig, aber empfehlenswert. Oxford-Pascal weist eine größere Ähnlichkeit mit Commodore-Basic auf als Profi-Pascal, weil es das gleiche Betriebssystem benutzt.

Die Anweisungen von Pascal kann man in einfache und in strukturierte Anweisungen einteilen (siehe Bild 1). Von strukturierten Anweisungen spricht man deswegen, weil diese Anweisungen wieder weitere Anweisungen enthalten dürfen.

Einfache Anweisungen

Von den einfachen Anweisungen wurde im ersten Teil bereits die Wertzuweisung besprochen. Sie besitzt kein eigenes Schlüsselwort. Wertzuweisungen besitzen folgende allgemeine Form:

```
Variable := Ausdruck
```

Dabei gilt, daß Variable und Ausdruck in der Regel vom gleichen Datentyp sein müssen. Eine Ausnahme bildet lediglich die Zuweisung eines Integer-Ausdrucks auf eine Real-Variable. Weil hier kein Informationsverlust auftreten kann, ist diese Zuweisung erlaubt. Die Bildung von komplexeren Ausdrücken wird im nächsten Teil behandelt.

Eine Anweisung, die in Basic sehr wichtig ist, sei hier nur am Rande erwähnt: die Goto-Anweisung. Während man in Basic kaum ohne sie auskommen kann, wird sie in Pascal nicht benötigt. Manche Umsteiger haben aber Probleme, ihren Basic-Stil zu vergessen und »Pascal-like« zu programmieren. Sie sollten bei der Verwendung der Goto-Anweisung das folgende beachten:

Auch wenn innerhalb der beiden Editoren eine Zeilennummer einzugeben ist, wird diese Zeilennummer nicht als Sprungmarke verwendet. Der Anwender muß alle Sprungziele, die in Pascal als Label bezeichnet werden, selbst definieren. Als Label sind ganze Zahlen ohne Vorzeichen erlaubt. Bei Profi-Pascal sind dies die Zahlen 0 bis 32767, bei Oxford-Pascal darf die Zahl sogar aus acht Ziffern bestehen. Die Labeldeklaration muß sofort nach dem Programmnamen stehen. Beispiel:

```
program springe;
 label 5,10;
 var i:integer;
begin
 i:=1;
 goto 5;
 10: writeln('Marke 10');
 5: i:=i+1;
 if i<=10 then goto 10
end.
```

Die Verbund-Anweisung

In einem Pascal-Programm stehen alle Anweisungen zwischen (mindestens je) einem Begin und einem End. Zusammen bilden die von Begin und End eingeschlossenen Anweisungen den Anweisungsteil eines Programms, der dem Definitionsteil folgt. Die Block-Klammern Begin und End bezeichnet man auch als Verbundanweisung, weil alle Anweisungen zwischen diesen beiden »Marken« als eine Einheit (ein Block) angesehen werden. Sie werden in Pascal wie ein Befehl behandelt. Anweisungen zwischen Begin und End werden in der Reihenfolge behandelt, in der sie aufgeführt sind. Daraus folgt, daß jedes Pascal-Programm eigentlich nur aus einer Verbund-Anweisung besteht.

Die Verbundanweisung wird benutzt, um Teile des Programms als eine Einheit zu erklären. Wo eine Anweisung stehen kann, darf statt dessen auch eine Verbundanweisung stehen. Dadurch wird erreicht, daß beispielsweise hinter einer If-Anweisung eine ganze Reihe weiterer Anweisungen stehen kann, ohne daß sie mit einem Goto übersprungen werden müssen:

```
 if a=x then
  begin
   a:=10;
   x:=5;
   writeln('irgendetwas')
  end;
a:=a+1; ....
```

In diesem Beispiel werden alle Anweisungen zwischen Begin und End nur dann ausgeführt, wenn die Bedingung »a = x« wahr ist. Sonst wird gleich mit der Anweisung nach End fortgefahren. Die Verbundanweisung ist also ein ganz wesentliches Hilfsmittel, um in Pascal strukturierte Programme (ohne Goto-Anweisung!) zu schreiben.

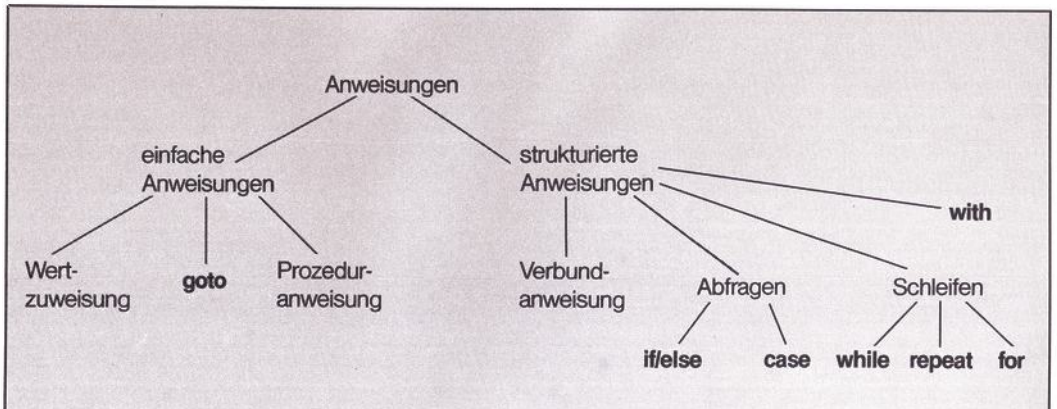

Bild 1. Ein Überblick über die Anweisungen von Pascal

In der Regel wird jede Anweisung mit einem Strichpunkt »;« abgeschlossen. Nur eine Ausnahme gibt es von dieser Regel: endet eine Anweisung direkt vor einem End, so kann der Strichpunkt weggelassen werden.

Die If-Anweisung sorgt für eine bedingte Programmausführung

Die If-Anweisung bewirkt, daß eine Anweisung nur dann ausgeführt wird, wenn eine bestimmte logische Bedingung erfüllt ist. Wenn die Bedingung nicht erfüllt ist, dann wird der Else-Teil interessant. Existiert kein Else-Teil (Fall 1), so wird keine weitere Anweisung im If-Then-Else-Block ausgeführt. Sonst ist die Anweisung, die im Else-Teil steht (Fall 2), die nächste auszuführende Anweisung. Es gibt also folgende zwei Formen der If-Anweisung:
Fall 1:

```
IF bed THEN aktion
```

Fall 2:

```
IF bed THEN aktion1 ELSE
aktion2
```

Im ersten Fall wird keine Anweisung ausgeführt, wenn der logische Ausdruck den Wert »false« ergibt. Im zweiten Fall wird dann die dem else folgende »aktion 2« ausgeführt. Als logischer Ausdruck wird ein Ausdruck bezeichnet, der als Ergebnis den Wert »false« oder »true« hat. Dies ist ein Vergleich oder ein Ausdruck mit Variablen vom Typ Boolean. Wichtig ist noch, daß unmittelbar vor einem else auf keinen Fall ein Strichpunkt stehen darf; dies würde zu einer Fehlermeldung führen. Die Syntax betrachtet nämlich die gesamte If/Else-Struktur als eine Anweisung und verbietet daher den Strichpunkt.

Bei der If/Else-Anweisung wird auf jeden Fall eine Anweisung ausgeführt, entweder »aktion 1« oder »aktion 2«.

An dem folgenden Beispielprogramm wird klarer, wann man diese beiden Formen der If-Anweisung braucht:

```
if a>=0 then a:=a+1
  else a:=0;
if (a<0) and (b>c) then
 begin
  a:=a+1;
  c:=b*a;
 end
  else
    begin
     writeln('a = ',a);
     writeln('b = ',b)
    end;
```

Das folgende Beispiel ist falsch!

```
var i:integer;
begin
 i:=10;
 if i then writeln;
end;
```

In diesem Fall ist die Bedingung kein logischer Ausdruck, sondern sie ist vom Typ Integer! Pascal unterscheidet im Gegensatz zu Basic genau zwischen den einzelnen Datentypen und läßt daher die If-Abfrage im letzten Beispiel nicht zu.

In Listing 1 wird nochmals an einem vollständigen Beispiel die Wirkung der If-Anweisung gezeigt. Das Programm Intervall überprüft, ob ein bestimmter Eingabewert sich innerhalb eines Intervalls befindet, oder ob er kleiner oder größer ist. Die Grenzen des Intervalls (a und b) werden als Konstante definiert. Die Konstantendeklarationen werden in Pascal nach der Label-Anweisung, aber vor der Typen- beziehungsweise Variablendeklaration angegeben. Konstanten bekommen wie Variablen einen Namen, ihr Wert läßt sich aber nicht mehr verändern. Das Programm zeigt gleichzeitig, wie in Pascal Eingabewerte überprüft werden.

Das Beispielprogramm enthält einen Kommentar. In Pascal werden Kommentare zwischen die Zeichen »(*« und »*)« eingeschlossen. Beispiel:

```
(* dies ist ein Kommentar *)
(* dies
  ist
  auch einer *)
```

Kommentarzeilen dürfen sich also über mehrere Zeilen erstrecken.

Die Case-Anweisung verarbeitet mehrere Alternativen

In manchen Fällen gibt es nicht nur zwei Alternativen, sondern gleich mehrere. In Basic verwendet man dann entsprechend viele, in sich verschachtelte If-Schleifen. Diese Vorgehensweise ist nicht nur schlecht lesbar, sondern auch fehleranfällig. Ein Pascal-Programm verwendet in einem solchen Fall die Case-Anweisung. In ihrer allgemeinen Form sieht sie folgendermaßen aus:

```
CASE ausdruck OF
 Case-Label1: Anweisung1;
 Case-Label2: Anweisung2;
 ....
 Case-LabelN: AnweisungN
 end;
```

Der Ausdruck nach »case« muß ein skalarer Type sein. Skalare Typen sind Integer, Boolean, Char und Aufzählungstyp. Der Aufzählungstyp wird später besprochen. Real ist nicht erlaubt. Ausdruck (zahl) und Case-Label (1, 2 und 3) müssen vom gleichen Typ sein.
Beispiel:

```
CASE zahl OF
 1: eins;
 2: zwei;
 3: BEGIN eins; zwei; drei
 END;
```

Vor einer Anweisung können mehrere Labels stehen, die durch Komma getrennt werden. Ein Beispiel dazu finden Sie im Programm »Fall« (Listing 2) beispielsweise in der Programmzeile

```
2,3,4: writeln (o'.k'.)
```

Ausgewählt und ausgeführt wird diejenige Anweisung, deren Case-Label mit dem Ergebnis des Ausdruck übereinstimmt. Listing 2 enthält ein Beispiel zur Case-Anweisung. Insgesamt werden drei Fälle unterschieden. Als Ausdruck wird hier ganz einfach eine Integer-Variable verwendet.

Die Marken in einer Case-Anweisung, die »Case-Labels«, dürfen auf keinen Fall mit den normalen Labels im Programm verwechselt werden. Sie brauchen nicht gesondert in einer Label-Deklaration definiert werden. Und ein Versuch, mit dem »goto« ein solches Case-Label anzuspringen, ist natürlich erfolglos!

Eine Frage ist noch offen: was passiert, wenn der Ausdruck einen Wert ergibt, für den kein Case-Label vorhanden ist ? Dieser Fall ist in Standard-Pascal nicht definiert und wird daher von jedem Compiler anders behandelt. Bei Oxford-Pascal erhält man den Laufzeitfehler »CASE ERROR«. Bei Profi-Pascal wird das Programm mit der nächsten Anweisung ohne Fehlermeldung fortgesetzt.

Profi-Pascal besitzt eine weitere Variante der Case-Anweisung, mit der dieses Problem behandelt werden kann. Alle Fälle, die nicht in der Liste vorkommen, werden mit einem Else-Label abgefangen. Das sieht dann beispielsweise so aus:

```
case zahl of
 1: eins;
 2: zwei;
 3: begin eins; zwei; drei
 end
 else vier
 end;
```

In den Fällen, in denen der skalare Typ »zahl« ungleich 1,2 oder 3 ist, wird der Else-Zweig aufgerufen. Die Anweisungen hinter den Case-Labels in unserem Beispiel (nämlich »eins«, »zwei« und »drei«) sind Aufrufe von Unterprogrammen. In Pascal werden Prozeduren mit ihrem Namen aufgerufen.

Wiederholung mit der For-Anweisung

Die Wiederholung eines Programmausschnitts wird durch die For-Anweisung erreicht. Vor Betreten der Schleife muß bereits bekannt sein, wie oft die Schleife durchlaufen werden soll. Der Wert des Schleifenindexes darf also nicht in der Schleife selbst berechnet werden! In Listing 3 wird innerhalb einer Schleife die Fakultät einer positiven ganzen Zahl berech-

```
program intervall;
(* es wird ueberprueft, ob ein
Eingabewert innerhalb eines
Intervalls liegt *)
(* a <= x <= b *)
const a= -10.0;
      b=  20.0;
var x:real;
begin
writeln('Bitte Eingabe');
read(x);
if x>b then writeln('x: ',x:5:2,' groesser als obere Schranke')
else
begin
  if x>=a then writeln('x: ',x:5:2,' innerhalb der Schranke')
  else
    writeln('x: ',x:5:2,' kleiner als untere Schranke')
end
end.
```

Listing 1. Beispiel zur If-Then-Else-Klausel

```
program fall;
var note:integer;
    buchstabe:char;
begin
  read(note);
  if (note>=1) and (note<=6) then
    case note of
    1: writeln('sehr gut');
    2,3,4: writeln('o.k.');
   5,6: writeln('schwach')
   end
 else writeln('liegt ausserhalb des Bereichs')
end.
```

Listing 2. Die Case-Anweisung wählt unter mehreren Möglichkeiten

net. Die Laufvariable (im Beispiel i) muß vom Typ Integer, Boolean, Char oder Aufzählungstyp sein. Anfangs- und Endwert der Schleifenvariablen können Konstanten, Variablen oder Ausdrücke desselben Typs sein. Die Laufvariable zählt die Anzahl der Wiederholungen. Vor jedem Durchlauf wird sie mit dem Endwert verglichen. Es wird überprüft, ob bereits genug Wiederholungen ausgeführt worden sind.

In der folgenden allgemeinen Form der For-Anweisung nennen wir die Laufvariable »vor«. Der Anfangswert heißt »a-Wert«. »e-Wert bezeichnet den Endwert.

```
FOR var := a-wert TO e-wert DO
aktion
```

oder

```
FOR var := a-wert DOWNTO
e-wert DO aktion
```

Nach jeder Wiederholung wird der Anfangswert entweder um eins erhöht (to) oder vermindert (downto). Eine Step-Anweisung wie in Basic gibt es in Pascal nicht.

Bei dem Beispiel in Listing 3 wird zunächst überprüft, ob der Eingabewert gleich Null ist. In diesem Fall steht das Ergebnis bereits fest. Bei einem Eingabewert größer Null wird die Fakultät berechnet und mit allen Zwischenergebnissen ausgegeben. N-Fakultät bedeutet:
1 * 2 * .... * N-1 * N .
So ist beispielsweise die Fakultät für
N=4 : 1 * 2 * 3 * 4 = 24 .

Schleifen mit »while« oder mit »repeat«?

Wenn vor Beginn der Schleife noch nicht feststeht, wie oft die Anweisungen wiederholt werden sollen, dann hilft die While- oder die Repeat-Schleife dem Programmierer weiter.

Solange eine bestimmte Bedingung erfüllt ist, wird bei der While-Schleife ein bestimmter Programmausschnitt ausgeführt. Eine While-Schleife ist folgendermaßen aufgebaut:

```
WHILE bed DO aktion
```

Der logische Ausdruck entscheidet, ob die Anweisung hinter dem »do« ausgeführt wird oder nicht. Ist das Ergebnis »true«, wird die Anweisung ausgeführt; bei dem Ergebnis »false« wird sie nicht ausgeführt. Die Anzahl der Wiederholungen hängt also von der Bedingung ab. Es ist möglich, daß der Ausdruck sofort den Wert »false« ergibt und die Anweisung damit überhaupt nicht ausgeführt wird. Es gibt auch Fälle, in denen die Schleife nicht beendet wird, weil der Ausdruck niemals den Wert »true« erhält. Einen solchen Fall nennt man eine unendliche Schleife. Diesen Fall sollte

```
program fakultaet;
var i,f,n:integer;
begin
 f:=1;
 writeln('bitte n eingeben');
 readln(n);
 if n=0 then writeln('Fakultaet = 1')
 else begin
  if n>0 then begin
   for i:=1 to n do begin
    f:=f*i;
    writeln(i:4,' Fakultaet: ',f:6);
   end
 end
 else writeln('negativ')
end;
end.
```

Listing 3. Die Fakultät-Funktion — mit der For-Schleife berechnet

```
program sum;
var i,k,n,summe:integer;
begin
 n:=1000;
 summe:=0;
 k:=0;
 while summe <= n do
 begin
  writeln('bitte Wert eingeben');
  read(i);
 if i>0 then summe:=summe+i
end;
writeln(summe)
end.
```

Listing 4. Beispielprogramm mit einer While-Schleife

```
program schleife;
var summe,k:integer;
begin
 summe:=0;
 i:=0;
 repeat
 i:=k+1;
 summe:=summe+k
until k>=10;
writeln('Summe: ',summe)
end.
```

Lisitng 5. Aufsummieren in einer Repeat-Anweisung

ein Programmierer auf jeden Fall vermeiden! Deshalb muß man auf diesen Fall achten, wenn das Ergebnis des logischen Ausdrucks auch innerhalb der Schleife geändert wird.

Das Beispiel in Listing 4 zeigt, wie innerhalb einer While-Schleife so lange Zahlen eingelesen und aufsummiert werden, bis die Zahl 1000 überschritten wird. Anhand dieses Beispiels nochmals die einzelnen Schritte bei While:

1. Der logische Ausdruck wird berechnet.
2. Falls der logische Ausdruck »true« ergibt, wird die Anweisung hinter Do ausgeführt — in diesem Fall eine Verbundanweisung. Eine Verbundanweisung wird immer wie eine einzelne Anweisung behandelt. Ist der logische Ausdruck »false«, so wird nicht diese Anweisung ausgeführt, sondern die darauf folgende. In diesem Fall heißt der Aufruf also:

```
writeln(summe);
```

3. Die Schritte 1 und 2 werden so lange ausgeführt, bis der logische Ausdruck »false« ist.

In Bild 2 werden die Teilschritte der While-Anweisung grafisch dargestellt.

Die Repeat-Anweisung ist in einem gewissen Sinne die Umkehrung der While-Anweisung. Bei ihr werden die Wiederholungen so lange ausgeführt, bis eine Bedingung erfüllt ist. Die allgemeine Form der Repeat-Anweisung lautet:

```
REPEAT Anweisung1; ... ;
       AnweisungN
UNTIL logischer Ausdruck;
```

Wenn der logische Ausdruck den Wert »true« hat, dann ist das Schleifenende erreicht. Solange er »false« ist, wird die Schleife wiederholt. Bei der Repeat-Anweisung wird (im Gegensatz zu While!) die Anweisung mindestens einmal durchlaufen. Ein Unterschied besteht auch darin,

Fortsetzung auf Seite 141

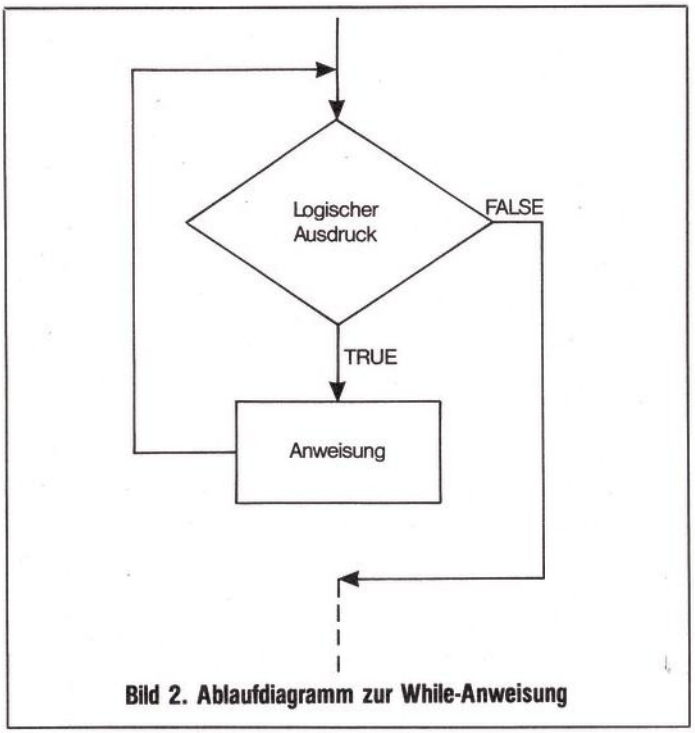

Bild 2. Ablaufdiagramm zur While-Anweisung

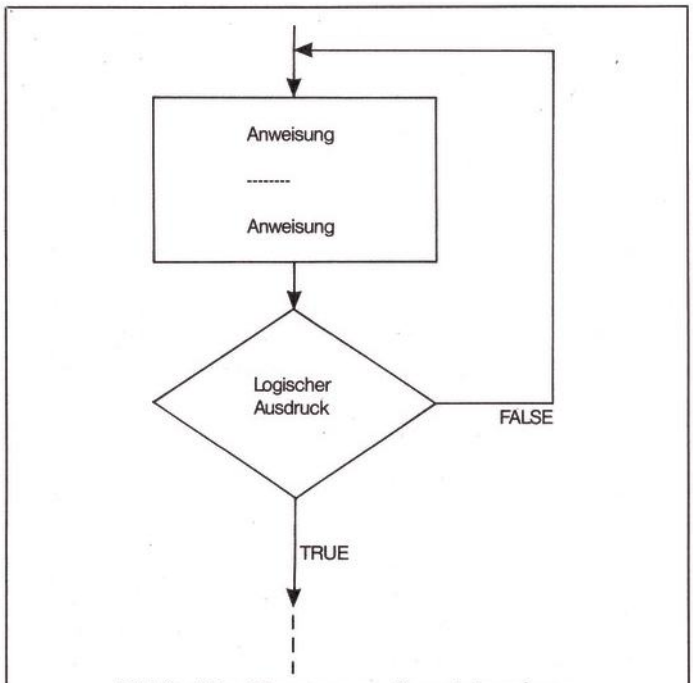

Bild 3. Ablaufdiagramm zur Repeat-Anweisung

WHILE	REPEAT
Test vor Schleifendurchlauf	Test nach Schleifendurchlauf
Test auf Fortsetzung der Schleife	Test auf Abbruch der Schleife
Wiederholungen, solange die Bedingung erfüllt ist	Wiederholungen, bis eine Bedingung erfüllt ist

Tabelle 1. Die Unterschiede zwischen While und Repeat