Linux, Knoppix, Mac OS X, Open Source: Vorteile von Unix et al. in Chemie & Biologie

Teil 14: Awk – Kontrollstrukturen

Röbbe Wünschiers

Computer sollen uns stupide Arbeit abnehmen. Dazu gehört z.B. die wiederholte Ausführung einer Berechnung oder die Steuerung einer Maschine mit vorgegebenen Parametern. Für die Programmierung derartiger Aufgaben stehen jeder ordentlichen Programmiersprache, wie Awk, Kontrollstrukturen zur Verfügung. Mit ihrer Hilfe lassen sich bestimmte Aufgaben wiederholt ausführen und der Programmablauf kann beeinflusst werden. Dies macht Programme erst agil und flexibel.

Wenn die Temperatur von 27°C überschritten ist, schalte die Kühlung ein. Solange aus der Datei *xyz.txt* eine Zeile gelesen werden kann, speichere sie in einen Array. Diese Aufgaben erfordern Kontrollstrukturen. Im ersten Fall muss z.B. der Wert einer Variablen, welche die Temperatur enthält, ständig überprüft werden. In Abhängigkeit von dem Variablenwert wird die eine oder andere Funktion ausgeführt. Im zweiten Fall wird eine Schleife sooft durchlaufen, bis eine bestimmte Bedingung zutrifft. Üblicherweise überprüfen Kontrollstrukturen Bedingungen auf ihren Wahrheitsgehalt. Ist der Wert der Variablen *temp* größer als 27? Ja oder nein? Kann aus der Datei *xyz.txt* noch eine Zeile gelesen werden? Ja oder nein? Diese binären Entscheidungen stehen im engen Zusammenhang mit der binären Logik der Computer allgemein – Strom oder kein Strom, das ist meist die Frage. Im folgenden werde ich Ihnen die wichtigsten Kontrollstrukturen mit je einem Beispiel vorstellen.

Bedingte Verzweigungen

Die grundlegende Kontrollstruktur auf der alle anderen Kontrollstrukturen aufbauen ist die ja/nein Abfrage (Abbildung 1). Bei der Programmierung spricht man allerdings selten von ja oder nein sondern von wahr (engl. *true*) oder falsch (engl. *false*). Der zu überprüfende Ausdruck ist in den meisten Fällen ein relationaler Zahlen- oder Zeichenausdruck, wie in den Tabellen 1 und 2 dargestellt. Betrachten wir ein einfaches Beispiel auf Basis der Textdatei *enzym.txt*. Wenn der Km-Wert eines Enzyms größer als 2 ist, dann wollen wir den betreffenden Enzymnamen ausdrucken, sonst nicht. Dazu verwenden in Terminal 1 das Skript *if.awk* (natürlich gibt es eine elegantere Lösung; siehe relationale Zahlenmuster in CLB 09/04).

Terminal 1

```
01    $ awk -f if.awk enzym.txt
02    Protease
03    $
```

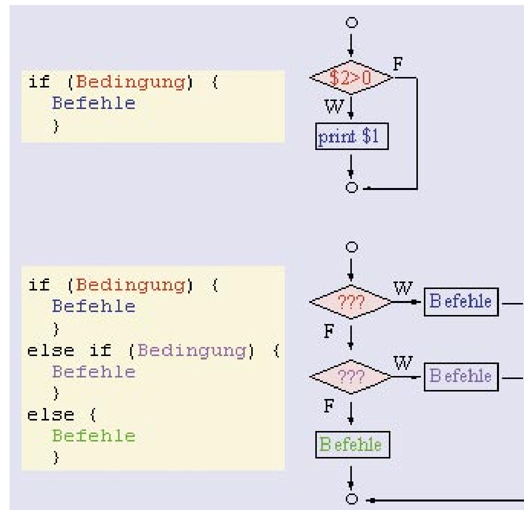

Abbildung 1: `if`-Konstrukt. Mit Hilfe der `if`-Verzweigung kann der Programmablauf in Abhängigkeit von verschiedenen Bedingungen gelenkt werden.

Werfen wir einen Blick auf das Skript *if.awk*. Die Entscheidung wird in Zeile 4 getroffen, das Kommando lautet also `if` (Bedingung ist wahr) `{Befehle}`. Wir könnten die Zeilen 4-6 auch in einer Zeile schreiben: `if ($2+0>2){print $1}`. Es ist aber sinnvoll sich von Anfang an die in längeren Programmen übersichtlichere Form anzugewöhnen. Die Bedingung die wir testen lautet `$2+0>2`. Die Variable *$2* enthält jeweils den Wert der rechten Spalte der Datei *enzym.txt*. Aber warum addieren wir Null? Der Grund ist, dass der erste Wert, den *$2* annimmt, "Km" ist – und Awk interpretiert "Km" als größer 2. Wir vergleichen hier Äpfel mit Birnen, bzw. Text mit Zahlen. Damit Awk den Wert der Variablen *$2* auf alle Fälle als Zahl interpretiert, führen wir einfach eine mathematische Operation aus.

Das vorhergehende Beispiel war die einfachste Form der `if`-Abfrage. Mit Hilfe des `if-else`-Konstrukts lassen sich mehrere Fälle voneinander unterscheiden. Wenn der Km-Wert größer als 2 ist, dann gebe die ganze Zeile aus, wenn er kleiner als 1 ist, dann setzte den Km-Wert gleich Null und gebe die Zeile aus, in allen anderen Fälle gebe den Text „deleted" aus. Diese Fallunterscheidung erreichen wir in Skript *if-else*.

Tabelle 1: Relationale Zahlenausdrücke

Ausdruck	Bedeutung
`n == v`	Wahr, wenn Variable *n* gleich Zahl *v*.
`n != v`	Wahr, wenn Variable *n* ungleich Zahl *v*.
`n < v`	Wahr, wenn Variable *n* kleiner als Zahl *v*.
`n <= v`	Wahr, wenn Variable *n* kleiner oder gleich Zahl *v*.
`n > v`	Wahr, wenn Variable *n* größer als Zahl *v*.
`n >= v`	Wahr, wenn Variable *n* größer oder gleich Zahl *v*.

Ausdruck	Bedeutung
n == "s"	Wahr, wenn Variable *n* gleich Textstring *s*.
n != "s"	Wahr, wenn Variable *n* ungleich Textstring *s*.
n < "s"	Zeichen-für-Zeichen Vergleich zwischen Variable *n* und Textstring *s*. Wahr, wenn *n* lexikographisch kleiner als *s* ist. "Haus" ist größer als "Antenne" und "abcd" ist größer als "abc".
n <= "s"	Wie oben, aber wahr, wenn Variable *n* lexikographisch kleiner oder gleich Textstring *s* ist.
n > "s"	Wie oben, aber wahr, wenn Variable *n* lexikographisch größer als Textstring *s* ist.
n >= "s"	Wie oben, aber wahr, wenn Variable *n* lexikographisch größer oder gleich Textstring *s* ist.

Tabelle 2: Relationale Zeichenausdrücke

awk. Darüber hinaus überprüfen wir in der Zeile 4, ob der Wert von *$2* dem Text "Km" entspricht. Auf diese Weise brauchen wir nicht, wie in dem Skript *if.awk*, Null zu der Variable *$2* addieren. Die Ausführung des Skriptes ist in Terminal 2 gezeigt.

Terminal 2

```
01    $ awk -f if-else.awk enzym.txt
02    Enzyme     Km
03    Protease   2.5
04    Hydrolase  0
05    deleted
06    $
```

Sie können eine beliebige Anzahl von `else-if`-Blöcken für Fallunterscheidungen einfügen. Es ist aber auch möglich ein einfaches `if-else`-Konstrukt ohne `else if`-Blöcke zu verwenden. Auf diese Art und Weise können Sie sehr flexibel in den Programmablauf eingreifen.

Schleifen

Schleifen (engl. *loops*) bauen auf den zuvor besprochenen bedingten Verzweigungen auf. Wir haben in der Vergangenheit bereits Schleifen zum Auslesen aller Elemente eines Arrays oder Hashes angewendet (siehe CLB 10/04). In diesem Fall wird mit der Schleife `for (index in array) {print array[index]}` nacheinander jeder Index des Arrays *array* in der Variablen *index* gespeichert und die Anweisungen in den geschwungen Klammern ausgeführt. Dies ist in Terminal 1 unter Anwendung des Skriptes *for-array.awk* gezeigt.

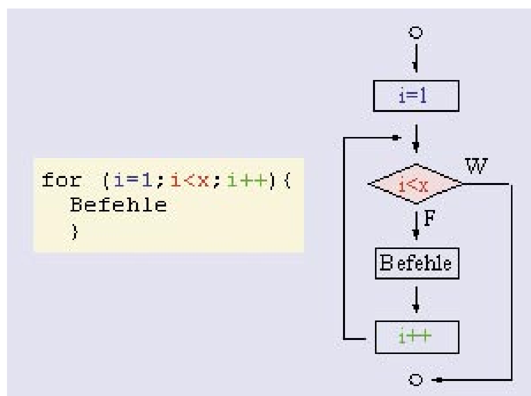

Abbildung 2: `for`-Konstrukt. Die `for`-Schleife besteht aus einer Initialisierung, der Bedingung, die es zu prüfen gilt und einem Zähler. Die Schleife ist abgeschlossen, wenn die Bedingung nicht mehr erfüllt ist. F=falsch, W=wahr.

Terminal 3

```
01    $ awk -f for-array.awk
02    a[x]: zwei
03    a[0]: 1
04    a[2]: 3
05    $
```

Das Skript *for-array.awk* benötigt keine Eingabe, daher ist der Code in einem `BEGIN`-Block eingeschlossen. In Zeile 4 des Skriptes erstellen wir 3 Arrayelemente (es handelt sich sogar um einen Hash, da ein Index ein Buchstabe ist), die wir anschließend in Zeile 7, gewürzt mit etwas Text, ausgeben. Beachten Sie, dass wir die Elemente des Arrays *a* nicht einfach mit dem Kommando `print a` ausgeben können. Probieren Sie es aus, indem Sie das Kommentarzeichen (#) in Zeile 5 entfernen und das veränderte Skript ausführen. Sie werden eine Fehlermeldung der Art "*awk: for-array.awk:5: fatal: attempt to use array `a' in a scalar context*" erhalten.

Die Ausgabe der Elemente eines Arrays oder Hashes mittels einer `for`-Schleife ist eine spezielle Anwendung. `for`-Schleifen lassen sich viel allgemeiner einsetzen. Dies wird in dem Skript *for.awk* verdeutlicht, das die Quadrate der Zahlen 1-5 berechnet.

Terminal 4

```
01    $ awk -f for.awk
02    1 4 9 16 25
03    $
```

Das `for`-Konstrukt in Zeile 4 des Skriptes *for.awk* mag auf den ersten Blick etwas verwirrend aussehen, ist aber ganz klar strukturiert. Die Anzahl der zu durchlaufenden Schleifen wird durch 3 funktionelle Einheiten bestimmt, den Initiatiator, die Bedingung und den Zähler (Abbildung 2). Während der Initiation (`i=1`) wird der Wert einer Startvariablen, hier die Variable *i*, gesetzt. Der Zähler (`i++`) erhöht den Wert der Variabeln *i* bei jedem Durchlauf um 1. Statt der Kurzform `i++` kann auch `i=i+1` geschrieben werden. Die Anzahl der Durchläufe wird durch die Bedingung `i<=5` überprüft. Die Zählervariable *i* steht bei jedem Durchlauf aktualisiert zur Verfügung. Wir quadrieren sie in Zeile 5 mit dem Kommando `i**2` und drucken das Ergebnis aus. Um nicht jede berechnet Zahl in einer neuen Zeile zu haben, setzen wir den Wert des Ausgabe-Zeilentrennzeichens (Variable *ORS*) in Zeile 3 als Leerzeichen (siehe Teil 11 in CLB 09/04). Daher müssen wir nach erfolgter Ausgabe in Zeile 6 einen abschließenden Zeilenumbruch ausgeben.

Wie die Abbildung 2 zeigt, ist die `for`-Schleife ein zusammengesetztes Konstrukt, dass unter anderem eine bedingte Verzweigung verwendet. Es gibt zudem noch zwei Derivate des `for`-Konstrukts: `while` und

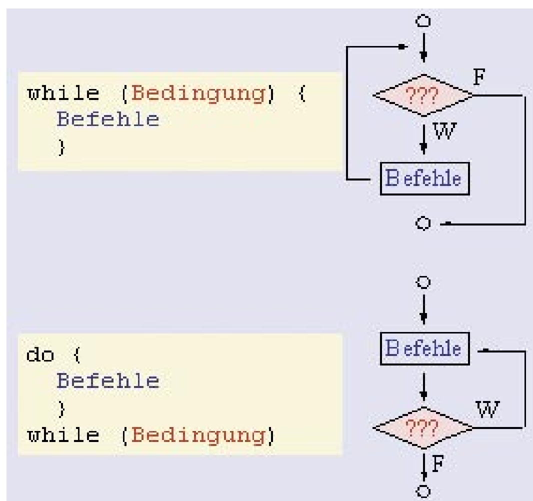

Abbildung 3: `while`-Konstrukt. Die `while`-Schleife und die verwandte `do-while` Schleife wird solange durchlaufen, solange die Bedingung erfüllt ist.

`do-while` (Abbildung 3). In beiden Fällen wird ein Befehlsblock solange ausgeführt, bis eine Bedingung nicht mehr erfüllt ist. Im Gegensatz zu dem `do-while`-Konstrukt wird in dem `while`-Konstrukt erst eine Bedingung überprüft und dann die Befehle ausgeführt (Abbildung 3). Die Befehle haben immer Einfluss auf die Bedingung, da es sich sonst um eine Endlosschleife handeln würde. Das Skript *while.awk* gibt ein Beispiel für die Anwendung des `while`-Konstrukts. Es kehrt die Felder jeder Zeile einer Eingabedatei um, wie es in Terminal 5 für die Datei *enzym.txt* gezeigt ist.

Terminal 5

```
01      $ awk -f while.awk enzym.txt
02      Km         Enzyme
03      2.5        Protease
04      0.4        Hydrolase
05      1.2        ATPase
06      $
```

In dem `BEGIN`-Block in Zeile 3 schalten wir die Ausgabe-Zeilentrennung aus. Dann speichern wir die Anzahl der Felder der aktuellen Zeile (Variable *NF*, *number of fields*) in der Variablen *i*. Solange der Wert der Variablen *i* größer als Null ist wird ein Feld gefolgt von einem Tabulator ausgegeben und der Wert von *i* um 1 erniedrigt (`i--`; entspricht `i=i-1`). Auf diese Weise werden alle Felder der aktuellen Zeile umgekehrt.

Schleifen unterbrechen

Manchmal ist es notwendig das Durchlaufen einer Schleife vorzeitig zu beenden. Beispielsweise weil eine bestimmte Rechenzeit überschritten wurde oder weil die Erfüllung einer anderen Bedingung die Ausführung der aktuellen Schleife hinfällig macht. Um dies zu bewerkstelligen gibt es zwei Kommandos, `break` und `continue` (Abbildung 4). Während `break` die aktuelle Schleife komplett verlässt, wird mit `continue` der nächste Schleifendurchgang gestartet. Beide Befehle können in `for`-, `while`- und `do-while`-Konstrukten verwendet werden.

Dezemberskript

Lassen Sie uns abschließend, der Dezember-Ausgabe der CLB gebührend und hoffend, dass sie rechtzeitig ausgeliefert wird und Sie rechtzeitig üben, das "Dezemberskript" *dez.awk* anschauen. Netter Weise hat es genau 31 Zeilen und gibt die verbleibenden Tage bis Weihnachten bzw. Silvester in Form eines Fortschrittsbalken an – allerdings nur im Dezember. Dies ist in Terminal 6 gezeigt.

Terminal 6

```
01      $ awk -f dez.awk
02      |----------------------#------#|
03       +++++++
04      $
```

Bis auf 2 Zeilen sollte Ihnen das Skript keine Probleme bereiten. In Zeile 5 verwenden wir den Befehl `"date +%m%e"|getline date` um das Systemdatum, genauer den Monat (`%m`) und den Tag (`%e`), in die Variable *date* zu lesen. Der 14. Dezember würde also als "12 14" gespeichert werden. In Zeile 6 trennen wir diese Zeile in den Array *d* auf. Dazu

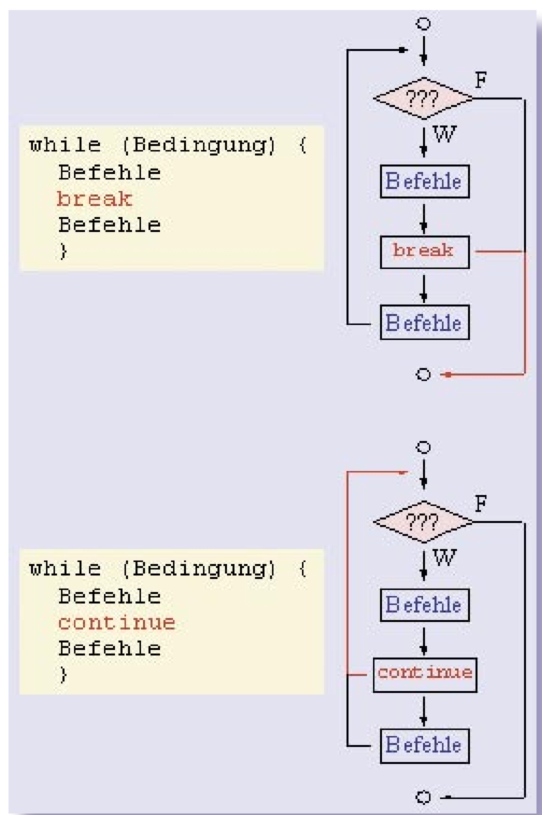

Abbildung 4: `break`/`continue`-Kommandos. Um Schleifen vorzeitig zu unterbrechen werden die Befehle `break` und `continue` verwendet. Während `break` die Schleife verlässt, bewirkt `continue` den nächsten Schleifendurchgang.

verwenden wir den `split` Befehl, der schon in Teil 12 (CLB 10/04) Anwendung fand. *d[1]* enthält nun den aktuellen Monat und *d[2]* den aktuellen Tag. In Zeile 9 wird das Skript nach einer Textausgabe sofort unterbrochen, wenn der aktuelle Monat nicht der Dezember ist. Es ist eben ein Dezemberskript, mit dem ich Ihnen erholsame Festtage wünsche.

Neue Befehle in dieser Ausgabe

`if` bedingte Verzweigung
`if-else` bedingte Verzweigung
`if-else if-else` bedingte Verzweigung
`for` flexibles Schleifenkonstrukt
`while` einfache Schleife
`while-do` einfache Schleife
`“xxx“getline var` führe Systemkommando `“xxx“` aus und speichere Ergebnis in Variable *var*

Skripte und Dateien

Textdatei *enzym.txt*

```
Enzyme     Km
Protease   2.5
Hydrolase  0.4
ATPase     1.2
```

Skript *if.awk*

```
01      # if.awk
02      # Verzweigung
03      {
04      if ($2+0>2){
05        print $1
06        }
07      }
```

Skript *if-else.awk*

```
01      # if-else.awk
02      # Verzweigungen
03      {
04      if ($2==“Km“){
05        print $0
06        }
07      else if($2+0>2){
08        print $0
09        }
10      else if ($2+0<1){
11        print $1“\t0“
12        }
13      else {
14        print “deleted“
15        }
16      }
```

Script *for-array.awk*

```
01      # for-array.awk
02      # Beispiel für eine Schleife
03      BEGIN{
04       a[0]=1; a[“x“]=“zwei“; a[2]=3
05       # print a
06       for (i in a){
07        print “a[“i“]: “a[i]
08         }
09      }
```

Script *for.awk*

```
01      # for.awk
02      # Einfache Schleife
03      BEGIN{ORS=“ “
04       for (i=1; i=5; i++){
05         print i**2}
06         print “\n“
07      }
```

Skript *while.awk*

```
01      # while.awk
02      # Felder umkehren
03      BEGIN{ORS=““}
04       {
05       i=NF
06       while (i>0) {
07         print $i“\t“; i--
08          }
09       print “\n“
10      }
```

Skript *dez.awk*

```
01      # dez.awk
02      # wie lange noch
03      BEGIN{
04       ORS=““
05       “date +%m%e“|getline date
06       split(date,d,“ “)
07       if (d[1]!=12) {
08         print “Es dauert noch ...\n“
09         exit
10          }
11       print “|“
12       for (i=1;i<=31;i++){
13         if (i==24 || i==31){
14           print “#“
15           }
16         else {
17           print “-“
18           }
19         }
20       print “|\n “
21       for (i=1;i<=d[2];i++){
22         print “+“
23         }
24       print “\n“
25       if (d[1]==12 && d[2]==24){
26         print “Frohes Fest“
27         }
28       else if (d[1]==12 && d[2]==31){
29         print “Frohes Neues Jahr“
30         }
31      }
```