awk! Hat das vielleicht ´was mit „hässlich“ zu tun?

„Nein!„ kann ich da nur mit voller Überzeugung sagen.

Günter Partosch, HRZ; LOGIN 1/93, Februar 1993

awk ist eine interpretierende Programmiersprache, die für das Manipulieren und Durchsuchen (Suche nach bestimmten Textmustern) von Textdateien konzipiert wurde. Der Name awk steht dabei nicht etwa für eine bestimmte Aufgabe, Funktion oder Eigenschaft (wobei die Ähnlichkeit zu englischen Begriffen wie „awkward“ und „awful“ das nahelegen könnte), sondern gibt einen Hinweis auf die drei Programmautoren Aho, Weinberger und Kernighan (ja, den von Kernighan und Ritchie).

Und damit es da gar kein Missverständnis gibt: Sprechen Sie awk nicht wie das englische „awkward“ oder „awful“ aus. Sagen Sie einfach „a w k“!

Und nun zum eigentlichen Thema:

Inhalt:

Einführung

Im Prinzip sieht jedes awk-Programm wie folgt aus:

```
Bedingung1      {Aktion1}
Bedingung2      {Aktion2}
Bedingung3      {Aktion3}
...
Bedingungn      {Aktionn}
```

In einer Programmzeile kann entweder die Bedingung oder der Aktionsteil weggelassen werden. Dabei darf sich der Aktionsteil auch über mehrere Zeilen erstrecken. Er muss jedoch in der gleichen Zeile wie die zugehörige Bedingung beginnen und die Klammern müssen geschrieben werden. Die Aktion selbst formulieren Sie in einer C-ähnlichen Sprache. Im übrigen dürfen Sie – was das Layout betrifft – ein awk-Programm so schreiben, wie es Ihnen gefällt (natürlich unter Berücksichtigung der obigen Einschränkungen).

Wird ein awk-Programm ausgeführt, so läuft eine – im Programm nicht aufgeführte – äußere Schleife ab, die sequentiell alle Zeilen der zu verarbeitenden Eingabedatei liest und dabei jede Eingabezeile der Reihe nach mit allen aufgeführten Bedingungen vergleicht. Ist eine Bedingung erfüllt, so wird die zugehörige Aktion ausgeführt.

awk zerlegt eine Eingabezeile in Abhängigkeit vom Wert des Feldtrennzeichens `FS` (eine der 14 awk-Standard-Variablen; siehe dazu Tabelle 1) in einzelne Felder, die mit `$1`, `$2`, ...,

`$(NF-1)` und`$(NF)` bezeichnet werden, wobei `NF` eine andere awk-Standard-Variable ist, in der die Anzahl der Felder der Eingabezeile gespeichert wird. Auf die gesamte Eingabezeile kann mit `$0` zugegriffen werden.

Tabelle 1: Standard-Variablen in awk

Variable	Bedeutung	Voreinstellung
`ARGC`	Anzahl der Argumente im awk-Aufruf	
`ARGV`	Argumente des awk-Aufrufs	
`FILENAME`	Name der aktuellen Eingabedatei	
`FNR`	aktuelle Zeilennummer in der aktuellen Eingabedatei	
`FS`	legt die Trennzeichen für die Eingabezeilen fest	**Voreinstellung**: „ „
`NF`	Anzahl der Felder in der aktuellen Eingabezeile	
`NR`	Anzahl der bisher gelesenen Eingabezeilen	
`OFMT`	Standard-Ausgabeformat für Zahlen	**Voreinstellung**: „`%.6g`“
`OFS`	Trennzeichen für die Ausgabezeilen	
`ORS`	Trennzeichen zwischen einzelnen Ausgabezeilen	**Voreinstellung**: „`\n`“
`RLENGTH`	Länge der Zeichenkette, die durch den letzten Aufruf der Funktion `match` gefunden wurde	
`RS`	Trennzeichen zwischen einzelnen Eingabezeilen	**Voreinstellung**: „`\n`“
`RSTART`	Anfangsposition der Zeichenkette, die durch den letzten Aufruf der Funktion `match` gefunden wurde	
`SUBSEP`	Trennzeichen für die Indizes „mehrdimensionaler“ Felder	**Voreinstellung**: „`\034`“

`$1`, `$2` usw. sind Zeichenketten und können dementsprechend im Aktionsteil manipuliert und verwendet werden.

Gegeben sei eine Datei, die Zeilen der Art

```
...
Emil Mayer 23.08.1960 175 cm 76 kg
Erna Müller 29.02.1957 160 cm 60 kg
...
```

mit den Angaben „Vorname“, „Nachname“, „Geburtsdatum“, „Größe“ und „Gewicht“ enthält. Es soll eine Ausgabedatei generiert werden, die nur noch Geburtsdatum und Vorname (in dieser Reihenfolge) enthält. Das folgende awk-Programm leistet das Gewünschte:

Beispiel 1:

```
{print $3, $1}
```

Da ein Bedingungsteil nicht vorhanden ist, wird die aufgeführte Aktion unabhängig von Bedingungen für alle Eingabezeilen ausgeführt: Mit Hilfe der awk-Standard-Funktion `print` wird der Inhalt der Felder `$3` und `$1` in einem Standard-Format ausgegeben.

Im obigen Beispiel ist die Aktion von keiner Bedingung abhängig. Als Bedingung können jedoch ohne weiteres Vergleiche und logische Ausdrücke angegeben werden, wie im folgenden Beispiel gezeigt wird:

Beispiel 2:

```
$4>170 && $6<=80  {print $0}
```

Angewendet auf die obige Eingabedatei bedeutet das, dass nur noch diejenigen Eingabezeilen (`$0`) aufgelistet werden, für die Größe>170 (`$4>170`) **und** (`&&`) Gewicht<=80 (`$6<=80`) gilt.

Als Vergleichsoperatoren im Bedingungs- und Aktionsteil sind wie in C `<`, `<=`, `==`, `>=`,`>` und `!=` und als logische Operatoren `!` (not),`&&` (and) und `||` (or) zulässig.

Als häufigste Bedingung werden in awk wohl die von anderen UNIX-Programmen her bekannten Zeichenkettenmuster in Form regulärer Ausdrücke verwendet. Dazu zwei Beispiele:

Beispiel 3:

```
/M[ea][iy]e?r/  {print $2, $3}
```

Als Bedingung ist hier ein regulärer Ausdruck aufgeführt, mit dessen Hilfe alle Eingabezeilen erkannt werden, die den Namen „Meier“ (mit möglichen Varianten wie „Mayer“, „Maier“, „Meyr“, „Meir“, „Mayr“ usw.) enthalten (Das Fragezeichen im regulären Ausdruck deutet an, dass das unmittelbar davor aufgeführte Element auch entfallen kann). Für die betreffenden Zeilen wird dann der Inhalt der Felder `$2` und `$3` in einem Standard-Format ausgegeben.

Beispiel 4:

```
$4 !~ /^[12]?[0-9][0-9]$/  {print $1, $2, $4}
```

Mit Hilfe der hier verwendeten Bedingung kann überprüft werden, ob das Eingabefeld `$4` einer Eingabezeile eine für ein bestimmtes Problem korrekt gebildete Zahl enthält. Genauer gesagt, es wird überprüft, ob ein angegebenes Muster **nicht** (`!~`) auf das Eingabefeld `$4` „passt“. Ist das der Fall, werden die Felder `$1`, `$2` und`$4` der betreffenden Zeile in einem Standard-Format ausgegeben. Das Muster selbst beschreibt eine Zeichenkette, die aus zwei oder drei Ziffern besteht, wobei die erste Stelle nur die Werte`1` und `2` annehmen oder auch entfallen (Spezifikation`?`) kann.

In den bisherigen Beispielen waren die jeweiligen Aktionen immer recht einfach: Es wurden lediglich die Eingabezeilen (bzw. Teile davon) ausgegeben. Es sind natürlich kompliziertere Aktionen denkbar. Beispielsweise ist es kein Problem, zusätzlich zur Ausgabe der Eingabezeilen, Durchschnittsgröße und Durchschnittsgewicht zu berechnen und am Schluss auszugeben:

Beispiel 5:

```
[1] BEGIN   {printf("%-30s %-11s %-6s %-8s\n\n",
            "Vorname Name", "Geburtstag", "Größe", "Gewicht")}
[2]         {printf("%-30s %11s %4d %4d\n", $1 " " $2, $3, $4, $6)
            gewicht += $6; groesse += $4; anzahl ++
            }
[3] END     {printf("\n\n%-30s %-11s %4d %4d\n",
            "Durchschnitt", " ", groesse/anzahl, gewicht/anzahl}
```

Erläuterungen (die Zeilennummern sind natürlich nicht Programmbestandteil):

[1] `BEGIN` ist eine spezielle awk-Bedingung, deren zugeordneter Aktionsteil genau einmal – bevor die Eingabezeilen verarbeitet werden – ausgeführt wird. Hier wird mit Hilfe der awk-Standard-Funktion`printf` eine Kopfzeile ausgegeben. Die Art und Weise, wie die auszugebenden Werte aufgeführt und formatiert werden, geschieht nach den in C üblichen Regeln.

[2] Der hier aufgeführte Aktionsteil ist von keiner Bedingung abhängig, d.h. er wird für alle Eingabezeilen ausgeführt. Im einzelnen besteht der Aktionsteil aus der Auflistung der Eingabezeilen in einem spezifizierten Format, aus der Aufsummierung (`+=`) der Werte für `groesse` und `gewicht` und aus dem Inkrementieren (`++`) der Variablen `anzahl`.

[3] `END` ist eine spezielle awk-Bedingung, deren zugeordneter Aktionsteil genau einmal – nachdem alle Eingabezeilen verarbeitet wurden – ausgeführt wird. Hier besteht die Aktion darin, dass mit Hilfe der awk-Standard-Funktion `printf` die Durchschnittswerte für Größe und Gewicht in einem spezifizierten Format ausgegeben werden.

Im Beispiel werden die in C üblichen Operatoren `+=` und`++` verwendet. Ebenso sind in einem awk-Programm die Zuweisungsoperatoren `-=`, `/=`, `*=`, `%=` (Divisionsrest) und `^=` (Exponentiation) sowie der Operator – (Dekrementieren) zulässig.

In den bisherigen Programmbeispielen wurden die awk-Standard-Funktionen`print` und `printf` verwendet. Eine fast ebenso häufig eingesetzte Funktion ist `gsub`, mit deren Hilfe Zeichenketten in einer Zeile gegen andere ausgetauscht werden können. Dazu ein Beispiel:

Beispiel 6:

```
/ä/     {gsub(/ä/, "ae")}
/Ä/     {gsub(/Ä/, "Ae")}
/ö/     {gsub(/ö/, "oe")}
/Ö/     {gsub(/Ö/, "Oe")}
/ü/     {gsub(/ü/, "ue")}
/Ü/     {gsub(/Ü/, "Ue")}
/ß/     {gsub(/ß/, "ss")}
        {print}
```

Mit Hilfe der in den Bedingungen angegebenen Muster für die deutschen Umlaute und das Eszet und der awk-Funktion `gsub` werden in den betreffenden Zeilen diese Zeichen gegen die entsprechenden Umschreibungen ausgetauscht. Die letzte Programmzeile ist notwendig, damit das Ergebnis der vorgenommenen Umsetzungen auch die Standardausgabe ausgegeben wird.

Ein abschließendes Beispiel soll zeigen, dass Sie im Aktionsteil assoziative Felder (Felder also, bei denen auf ein Feldelement nicht durch einen einfachen Index Bezug genommen wird, sondern über eine Zeichenkette) und die in C üblichen Kontrollstrukturen verwenden können.

Beispiel 7:

```
[1] BEGIN   {FS="[^a-zA-Z0-9äÄöÖüÜáàâéèêíìîóòôúùû]"}
[2]         {for (j=1; j<=NF; j++)
                 {anzahl [$j] ++}
            }
[3] END     {for (j in anzahl)
                 {printf ("%-40s %4d\n", j, anzahl [j] ) }
            }
```

Erläuterungen (die Zeilennummern sind nicht Bestandteil des Programms):

[1] `BEGIN` ist eine spezielle awk-Bedingung, die genau einmal am Programmanfang ausgeführt wird. Hier wird mit Hilfe eines regulären Ausdrucks festgelegt, dass die awk-Variable `FS` (in der die Feldtrennzeichen für die Eingabezeilen festgelegt werden) bestimmte Zeichen nicht annimmt. Alle anderen nicht aufgeführten Zeichen sind dann Trennzeichen.

[2] Diese Aktion ist von keiner Bedingung abhängig, d.h. sie wird für alle Eingabezeilen ausgeführt. Hier wird die Eingabezeile in einzelne Zeichenketten zerlegt und in einer Schleife das jeweils entsprechende Feldelement des assoziativen Felds `anzahl` inkrementiert.

[3] `END` ist eine spezielle awk-Bedingung, die genau einmal am Programmanfang ausgeführt wird. Hier wird mit Hilfe einer Schleife der Inhalt des Felds `anzahl` in einem spezifizierten Format ausgegeben. Dabei wird die Schleife so lange durchlaufen, wie es noch Feldelemente gibt. **Übrigens**: Eigentlich wurde im Beispiel eine programmtechnische Sünde begangen (aber hier durchaus zulässig), da die Variable `j` in der Anweisung [2] als Integer-Variable und in Anweisung [3] als Zeichenketten-Variable verwendet wird.

Aufruf

Jetzt muss eigentlich nur noch gesagt werden, wie awk aufzurufen ist. Da gibt es im wesentlichen zwei Möglichkeiten:

a) awk-Programm in einer Datei:

```
awk -f awk-programmdatei {eingabedatei}
```

b) awk-Programm in den Aufruf eingebettet

```
awk 'awk-programm' {eingabedatei}
```

Bei beiden Möglichkeiten kann der Teil in den geschweiften Klammern beliebig oft – auch 0mal – wiederholt werden. Wird keine Eingabedatei angegeben, so wird von der Standardeingabe gelesen, was meistens die Tastatur ist.

Bei der zweiten Möglichkeit kann sich der Aufruf über mehrere Zeilen erstrecken und wird daher am besten in eine Shell-Prozedur eingebettet. Korrekte Aufrufe sind dann:

Beispiel 8:

```
awk -f beispiel.awk eingabe*
```

Das awk-Programm, das sich auf der Datei `beispiel.awk` befindet, wird auf alle Dateien angewandt, deren Name mit `eingabe` beginnt.

Beispiel 9:

```
awk ' {printf("%s-%4d %s\n",FILENAME, FNR, $0) }' eingabe*
```

Das auszuführende awk-Programm befindet sich nicht in einer Datei, sondern ist zwischen zwei Apostrophe in den Aufruf eingebettet. Durch Programm selbst werden die Zeilen aller Dateien, deren Name mit `eingabe` beginnt, nummeriert und mit dem Dateinamen auf der Standardausgabe aufgelistet.

Üblicherweise wird das Ergebnis eines awk-Aufrufs auf die Standardausgabe (was meistens der Bildschirm ist) geschrieben. Ist das nicht gewünscht, so muss die Ausgabe durch> auf eine Ergebnisdatei umgeleitet oder durch `|` zu einem weiteren Programm weitergeleitet werden:

Beispiel 10:

```
awk -f beispiel.awk eingabe* > ergebnis
```

oder

Beispiel 11:

```
awk ' {printf("%s-%4d %s\n", FILENAME, FNR, $0) }' eingabe* | more
```

Im Beispiel 10 werden nacheinander alle Dateien im aktuellen Verzeichnis, deren Name mit `eingabe` beginnt, mit Hilfe des awk-Programms, das sich in der Datei `beispiel.awk` befindet, verarbeitet. Die Ausgabe wird auf die Datei `ergebnis` umgelenkt.

Im Beispiel 11 wird das auszuführende awk-Programm zwischen zwei Apostrophen im Aufruf aufgeführt. Das Programm selbst ist von keiner Bedingung abhängig, d.h. es werden für alle Eingabezeilen Dateiname (`FILENAME`), Zeilennummer (`FNR`) und Inhalt (`$0`) ausgegeben. Als Eingabedateien dienen alle Dateien im aktuellen Verzeichnis, deren Name mit`eingabe` beginnt. Die Programmausgabe wird zum UNIX-Kommando`more` weitergeleitet.

Verfügbarkeit

awk ist Bestandteil aller UNIX-Systeme. Probleme könnte es gegebenenfalls mit älteren Versionen geben. Sollte das Programm wider Erwarten doch nicht vorhanden sein, so gibt es eine Public-Domain-Version von GNU, bei deren Beschaffung Ihnen das HRZ behilflich sein kann. Leider sind auf den verschiedenen UNIX-Systemen auch verschiedene Zeichensätze installiert, so dass Sie u.a. bei den deutschen Umlauten „vorsichtig“ sein sollten. Im übrigen gibt es auch eine Version für PCs mit der Bezeichnung **gawk** (auch von GNU), die den üblichen PC-Zeichensatz beherrscht.

Literatur

Literatur zu awk gibt es selbstverständlich auch:

Staubach, Gottfried: *UNIX-Werkzeuge zur Textmusterverarbeitung. awk lex yacc*; Springer; Berlin, Heidelberg, New York, London, Paris, Tokyo, Hong Kong; 1989; 157 Seiten; ISBN 3-540-51232-2 und ISBN 0-387-51232-2 (Reihe „*Informationstechnik und Datenverarbeitung*“)

Herold, Helmut: *AWK und SED*; Addison-Wesley; Bonn, München, Paris, Reading (Massachusetts), Menlo Park (California), New York, Don Mills (Ontario), Wokingham (England), Amsterdam, Milan, Sydney, Tokyo, Singapore, Madrid, San Juan, Seoul,

Mexico City, Tapei (Taiwan); 1991; 327 Seiten; ISBN 3-89319-344-8 (Reihe „*UNIX und seine Werkzeuge*“)

und natürlich das Originalhandbuch

Aho, Alfred V.; Kernighan, Brian W.; Weinberger, Peter J.: *The Awk Programming Language*; Addison-Wesley; Reading (Massachusetts), Menlo Park (California), New York, Don Mills (Ontario), Wokingham (England), Amsterdam, Bonn, Sydney, Singapore, Tokyo, Madrid, Bogotá, Santiago, San Juan; 1988; 210 Seiten; ISBN 020107981-X

Abschließende Bemerkung

In einem solchen relativ kurzen Artikel können natürlich nur die Anfangsgründe der awk-Programmierung behandelt und verschiedene Einzelheiten nur kurz angedeutet oder gar nicht behandelt werden. Ebenso kann nicht auf die Schwierigkeiten, Unzulänglichkeiten und Probleme (die gibt es tatsächlich) im Umgang mit awk eingegangen werden.