ANWENDER-DOKUMENTATION	Teil 3	MOS
11/87	Mustererkennungs- und Musterverarbeitungssprache awk	MUTOS 1700

Programmtechnische Beschreibung

Teil 3

Mustererkennungs- und Musterverarbeitungssprache awk

AC A 7100/7150

VEB Robotron-Projekt Dresden

C 8014-0005-1 M 3030

Ausgabe: 11/87

Die Ausarbeitung dieser Dokumentation erfolgte durch ein Kollektiv des
VEB Robotron-Elektronik Dresden Stammbetrieb des VEB Kombinat Robotron.

Im Interesse einer ständigen Weiterentwicklung werden alle Leser gebeten, Hinweise zur Verbesserung der Dokumentation dem Herausgeber mitzuteilen.

Herausgeber:

VEB Robotron-Projekt Dresden
Leningrader Str. 9
Dresden 8010

Kurzreferat

Die vorliegende Schrift ist eine Anleitung zur Anwendung der Mustererkennungs- und Musterverarbeitungssprache awk für das Betriebssystem MUTOS 1700.
Die Grundoperation, die durch awk ausgeführt wird, besteht darin, daß eine Menge von Files nach vorgegebenen Mustern durchsucht wird und auf Zeilen bzw. Felder, die diesen Mustern genügen, spezifizierte Aktionen ausgeführt werden. Dabei können bestimmte Operationen zur Datenauswahl und -transformation einfach ausgedrückt werden. Die awk-Muster können beliebige Bool'sche Kombinationen von regulären Ausdrücken und relationalen Operatoren über Zeichenketten (Strings), Felder, Variable und Array-Elemente enthalten. Die Aktionen können gleichartige Muster-Erkennungs-Konstruktionen wie in den Mustern selbst, arithmetische und Zeichenketten-Ausdrücke und Zuweisungen, sowie if-else-, while-, for-Anweisungen und mehrfache Ausgabe-Ströme einschließen.
Diese Schrift enthält eine Nutzeranleitung sowie eine Diskussion über die Implementierung von awk.

|

Inhaltsverzeichnis Seite

1. Einleitung

Die Programmiersprache awk wurde entworfen, um eine Vielzahl von Problemen bei der Auffindung von Informationen und der Textmanipulation in einfacher Weise zu formulieren und auszuführen.
Die Grundoperation von awk besteht darin, die jeweilige Eingabe nach Zeilen abzusuchen, welche eines der vom Nutzer spezifizierten Muster enthalten bzw. in ihrer Gesamtheit diesem Muster entsprechen. Zu jedem Muster kann eine Aktion spezifiziert werden, welche auf jede Zeile ausgeführt wird, die dem vorgegebenen Muster genügt.
Die Nutzer, welche mit dem MUTOS-Dienstprogramm grep (Kurzbeschreibung aller genannten Dienstprogramme siehe Programmtechnische Beschreibung, Teil 1) vertraut sind, werden bestehende ähnlichkeiten erkennen, obwohl bei awk die Muster in allgemeinerer Form als bei grep angegeben werden können und die erlaubten Aktionen komplizierter sein können als einfach die Ausgabe der betreffenden Zeile. Zum Beispiel gibt das awk-Programm

```
{print $3, $2}
```

die dritte und zweite Spalte (Feld) einer Tabelle in ebendieser Reihenfolge aus. Das Programm

```
$2 ~ /A|B|C/
```

gibt alle Zeilen des Eingabe-Files aus, welche im zweiten Feld ein A, B oder C enthalten. Das Programm

```
$1 != prev { print; prev = $1 }
```

gibt alle Zeilen aus, bei denen sich das erste Feld vom ersten Feld der vorherigen Zeile unterscheidet.

1.1. Benutzung

Das Kommando

```
 awk program [file]
```

führt die awk-Kommandos, die im Programm prog_r_am angegeben sind, auf die Menge der angegebenen Files (file) aus bzw. auf die Standardeingabe, falls keine Files angegeben sind. Die Anweisungen können auch in einem File pfile plaziert und mit dem Kommando

```
awk -f pfile [file]
```

ausgeführt werden.

1.2. Programmstruktur

Ein awk-Programm ist eine Folge von Anweisungen der Form:

```
muster {aktion}
muster {aktion}
...
```

Jede Zeile der Eingabe wird der Reihe nach mit jedem der vorgegebenen Muster verglichen. Für jeden Fall, bei dem die Zeile das vorgegebene Muster enthält bzw. in ihrer Gesamtheit dem Muster entspricht, wird die mit dem Muster spezifizierte Aktion ausgeführt. Sind alle Muster getestet, wird die nächste Zeile geholt, und der Vergleich wird erneut ausgeführt.
Bei einer awk-Anweisung kann auch entweder das Muster oder die Aktion weggelassen werden. Wird ein Muster ohne Aktion spezifiziert, werden die Zeilen, die dem Muster genügen, einfach zur Ausgabe (File bzw. Standardausgabe) kopiert. (Demzufolge kann eine Zeile, welche mehreren Mustern genügt, auch mehrmals ausgegeben werden.) Falls eine Aktion ohne Muster spezifiziert wurde, wird diese Aktion auf jede Eingabezeile ausgeführt. Eine Zeile, welche keinem der vorgegebenen Muster genügt, wird ignoriert.
Da sowohl die Muster als auch die Aktionen wahlfrei sind, müssen die Aktionen in geschweifte Klammern eingeschlossen werden, um sie von den Mustern zu unterscheiden.

1.3. Records und Felder

Die Eingabedaten für awk werden in Records unterteilt, welche jeweils durch ein Record-Trennzeichen abgeschlossen werden. Das Standard-Record-Trennzeichen ist das Zeichen <NL> (newline), so daß awk standardmäßig seine Eingabe zeilenweise bearbeitet. Unter awk existiert eine Reihe von vordefinierten Variablen, die im folgenden mit genannt werden. So ist z.B. die Nummer des aktuellen Records über die Variable NNNRRR zu erreichen.
Jeder Eingabe-Record wird in Felder untergliedert. Die Felder sind normalerweise durch Leerzeichen bzw. Tabulatorzeichen <HT> getrennt, jedoch kann das Feldtrennzeichen für die Eingabe geändert werden, wie weiter unten beschrieben wird. Auf die Felder bezieht man sich mit $1, $2 usw., wobei $1 das erste Feld ist und unter $0 der gesamte Eingabe-Record verstanden wird. Felder können auf der rechten Seite von Zuweisungen auftreten. Die Anzahl der Felder im aktuellen Record ist über die Variable NF zugänglich.
Die Variablen FS und RS beziehen sich auf das Feldtrennzeichen bzw. das Record-Trennzeichen für die Eingabe. Sie können jederzeit in ein beliebiges anderes einzelnes Zeichen geändert werden. Um FS das Zeichen c zuzuordnen, kann ebenfalls die wahlfreie Kommando-Option -Fc benutzt werden.
Falls das Record-Trennzeichen nicht gesetzt ist (Zuweisung mit leerem Argumentteil), wird eine leere Eingabezeile als Record-Trennzeichen gedeutet, und Leerzeichen, <HT> und <NL> werden als Feldtrennzeichen behandelt.
Weiterhin enthält die Variable EILENAME den Namen des aktuellen Eingabe-Files.

1.4. Ausgabe

Eine Aktion kann auch ohne zugehöriges Muster angegeben werden, in diesem Fall wird die Aktion auf alle Zeilen ausgeführt. Die einfachste Aktion besteht darin, einen Record vollständig oder teilweise auszugeben. Das wird durch das awk-Kommando print erreicht. Das awk-Programm

```
{ print }
```

gibt jeden Record aus, und kopiert somit die Eingabe unverändert in die Ausgabe. Sinnvoller ist es normalerweise, ein oder mehrere Felder jedes Eingabe-Records auszugeben. Zum Beispiel gibt

```
{ print $2, $1 }
```

die ersten zwei Felder in vertauschter Reihenfolge aus. Einzelne Bestandteile, die in der print-Anweisung durch ein Komma getrennt sind, werden bei der Ausgabe durch das aktuelle Feldtrennzeichen getrennt. Bestandteile, die nicht durch Komma getrennt sind, werden einfach aneinandergehängt, wie das z.B. durch folgendes awk-Programm mit dem ersten und dem zweiten Feld geschieht:

```
{ print $1 $2 }
```

Die vordefinierten Variablen NF und NR können ebenfalls benutzt werden. Zum Beispiel bewirkt

```
{ print NR, NF, $0 }
```

die Ausgabe eines jeden Records mit vorangestellter Record- und Feldnummer.
Die Ausgabe kann auf verschiedene Files aufgeteilt werden. Das Programm

```
{ print $1 >"foo1"; print $2 >"foo2" }
```

gibt das erste Feld $1 auf das File foo1 und das zweite Feld $2 auf das File foo2 aus. Auch die Notation ">>" kann benutzt werden. So fügt

```
{ print $1 >>"foo" }
```

seine Ausgabe an das File foo an. (Falls es notwendig ist, werden die Ausgabe-Files in allen Fällen erzeugt.) Der File-Name kann eine Variable, ein Feld oder eine Konstante sein. Zum Beispiel benutzt

```
{ print $1 >$2 }
```

den Inhalt von Feld 2 als File-Namen.
Die Anzahl der möglichen Ausgabe-Files ist begrenzt, gegenwärtig beträgt die maximale Anzahl 10.
Entsprechend der Philosophie des Betriebssystems MUTOS kann die Ausgabe auch über eine Pipe an andere Prozesse weitergegeben werden, z.B. gibt

```
print | "mail bwk"
```

die Ausgabe über das Kommando mail an den Nutzer bwk weiter.
Die Variablen OFS und ORS können benutzt werden, um das augenblickliche Feldtrennzeichen bzw. das augenblickliche Record-Trennzeichen für die Ausgabe zu ändern. Das Record-Trennzeichen für die Ausgabe wird an die Ausgaben der print-Anweisung angefügt.

Die Sprache awk stellt auch die printf-Anweisung zur Formatgestaltung der Ausgabe zur Verfügung. Die Anweisung

```
printf format expr, expr, ...
```

formatiert die Ausdrücke der Liste entsprechend der Spezifikation in format und gibt sie aus. Zum Beispiel gibt

```
printf "%8.2f %10ld\n", $1, $2
```

$1 als achtstellige Gleitkommazahl mit zwei Stellen nach dem Dezimalpunkt und $2 als zehnstellige Dezimalzahl, gefolgt von <NL>, aus. Ausgabetrennzeichen werden bei printf in keinem Fall automatisch erzeugt, sie müssen in jedem Falle selbst hinzugefügt werden, wie in diesem Beispiel demonstriert. Die Verwendung der Funktion printf ist identisch zu der in der Programmiersprache C enthaltenen Funktion printf.

2. Muster

Ein Muster, welches vor einer Aktion steht, wirkt wie ein Selektor, welcher bestimmt, ob eine Aktion auszuführen ist. Eine Vielzahl von Ausdrücken kann als Muster verwendet werden: reguläre Ausdrücke, relationale arithmetische Ausdrücke, Ausdrücke mit Zeichenkettenwert und beliebige Bool'sche Kombinationen der oben genannten.

2.1. BEGIN und END

Das spezielle Muster BEGIN entspricht dem Beginn der Eingabe, bevor der erste Record gelesen wurde. Das Muster END entspricht dem Ende der Eingabe, nachdem der letzte Record verarbeitet wurde. BEGIN und END liefern folglich eine Möglichkeit, vor und nach dem Verarbeiten der Eingabe Aktionen auszuführen, um Initialisierungs- bzw. Abschlußaktionen vorzunehmen.
Zum Beispiel kann das Feldtrennzeichen durch

```
BEGIN  { FS = ":" }
... restlicher Programmtext
```

auf Doppelpunkt ":" gesetzt werden. Oder mit dem Programm

```
END  { print NR }
```

können die Eingabezeilen gezählt werden. Falls BEGIN verwendet wird, muß es das erste Muster sein, welches angegeben ist. Analog muß END das letzte Muster sein, wenn es verwendet werden soll.

2.2. Reguläre Ausdrücke

Der einfachste reguläre Ausdruck ist eine Literal-Zeichenkette, die in Schrägstriche eingeschlossen ist, wie z.B.

```
/Schmidt/
```

Das ist bereits ein vollständiges awk-Programm, welches alle Zeilen ausgibt, in denen der Name "Schmidt" erscheint. Falls diese Zeile "Schmidt" als Teil eines längeren Wortes enthält, wird sie ebenfalls ausgegeben, wie z.B. bei

```
Meier-Schmidtmann
```

Reguläre awk-Ausdrücke umfassen die Formen regulärer Ausdrücke, die im MUTOS-Text-Editor ed und im Dienstprogramm grep verwendet werden. Zusätzlich erlaubt awk wie im Programm lex die Verwendung von runden Klammern "(" und ")" zum Gruppieren, von "|" für Alternativen, von "+" für "ein oder mehr" und von "?" für "kein oder ein". Klassen von Zeichen können abgekürzt werden. So bedeutet "[a-zA-Z0-9]" die Menge aller Buchstaben und Ziffern. Als Beispiel sei das awk-Programm

```
    /[Ss]chmidt|[Mm]eier|[Mm]ueller/
```

angeführt, welches alle Zeilen ausgibt, die irgendeinen der Namen "Schmidt", "Meier" oder "Mueller" enthalten, unabhängig davon, ob sie mit Großbuchstaben beginnen oder nicht.
Reguläre Ausdrücke (mit den oben angeführten Erweiterungen) müssen ebenso wie bei ed und sed in Schrägstriche eingeschlossen werden. Innerhalb eines regulären Ausdrucks sind Leerzeichen und die für reguläre Ausdrücke geltenden Meta-Zeichen signifikant. Um die Meta-Zeichen-Bedeutung eines dieser Zeichen aufzuheben, ist ihm ein inverser Schrägstrich "\" voranzustellen. Ein Beispiel dazu ist das Muster

```
    /\/.*\//
```

welches beliebigen Zeichenketten entspricht, die in Schrägstriche eingeschlossen sind.
Man kann auch spezifizieren, daß irgendein Feld oder eine Variable einem regulären Ausdruck entspricht (bzw. nicht entspricht), indem die Operatoren "~" und "!~" benutzt werden. Das Programm

```
    $1 ~ /[jJ]ohann/
```

gibt alle Zeilen aus, deren erste Felder "johann" oder "Johann" entsprechen. Man sollte jedoch beachten, daß z.B. auch "Johannes", "Johann-Georgenstadt" usw. erfaßt werden. Um tatsächlich nur [jJ]ohann zu erfassen, ist

```
    $1 ~ /^[jJ]ohann$/
```

zu benutzen. Das Zeichen "^" bezieht sich auf den Anfang einer Zeile oder eines Feldes; das Zeichen "$" jeweils auf das Ende.

2.3. Relationale Ausdrücke

Ein awk-Muster kann ein relationaler Ausdruck sein, wobei die üblichen relationalen Operatoren "<", "<=", "==", "!=", ">=" und ">" eingeschlossen sind. Ein Beispiel ist das Muster

```
    $2 > $1 + 100
```

durch das Zeilen ausgewählt werden, bei denen das zweite Feld um wenigstens 100 größer als das erste Feld ist. In entsprechender Weise gibt

```
    NF % 2 == 0
```

Zeilen mit einer geraden Anzahl von Feldern aus.
Wenn bei relationalen Tests keiner der Operanden numerisch ist, wird ein Zeichenkettenvergleich durchgeführt, in allen anderen Fällen findet ein numerischer Vergleich statt. Folglich sucht

```
    $1 >= "s"
```

alle Zeilen, die mit einem "s", "t", "ü, usw. beginnen. Falls keine weiteren Informationen spezifiziert sind, werden alle Felder als Zeichenketten behandelt, so führt das Programm

```
    $1 > $2
```

einen Zeichenkettenvergleich aus.

2.4. Kombination von Mustern

Ein Muster kann eine beliebige Bool'sche Kombination von Mustern sein, wobei die Operatoren "||" ("oder"), "&&" ("und") und "!" ("nicht") zur Anwendung kommen können. Zum Beispiel sucht

```
    $1 >= "s" && $1 < "t" && $1 != "schmidt"
```

Zeilen, bei denen das erste Feld mit "s" beginnt, wobei es aber nicht den Inhalt schmidt haben darf. Die Operatoren "&&" und "||" garantieren, daß ihre Operanden stets von links nach rechts berechnet werden. Die Berechnung wird abgebrochen, sobald der Wahrheitswert (true oder false) feststeht.

2.5. Durch Muster spezifizierte Bereiche

Anstelle eines einzelnen Musters zur Auswahl einer Aktion können auch zwei durch Komma getrennte Muster angegeben werden, wie zum Beispiel in

```
   muster1 , muster2 { ... }
```

In diesem Fall wird die spezifizierte Aktion für alle Zeilen zwischen einem Auftreten von muster1 und dem nächstfolgenden Auftreten von muster2 (einschließlich) ausgeführt. Zum Beispiel gibt

```
    /start/, /stop/
```

alle Zeilen zwischen "start" und "stop" aus, während

```
    NR == 100, NR == 200 { ... }
```

die spezifizierte Aktion für alle Eingabezeilen von 100 bis 200 ausführt.

3. Aktionen

Eine awk-Aktion ist eine Folge von Aktionsanweisungen, welche durch Zeilenwechselzeichen oder Semikolon abgeschlossen sind. Diese Aktionsanweisungen können benutzt werden, um eine Vielzahl von Buchführungsaufgaben und Zeichenkettenmanipulationen zu realisieren.

3.1. Installierte Funktionen

Eine der in awk realisierten Funktionen ist eine Längenfunktion, welche die Länge einer Zeichenkette berechnet. Das Programm

```
{print length, $0}
```

gibt jeden Record mit vorangestellter Längenangabe aus. Die Bezeichnung length selbst stellt eine "Pseudo-Variable" dar, die die Länge des aktuellen Records liefert. Der Ausdruck length(argument) stellt eine Funktion dar, die die Länge ihres Arguments liefert, wie zum Beispiel im Programm

```
{print length($0), $0}
```

welches zum obigen äquivalent ist. Das Argument argument kann ein beliebiger Ausdruck sein.
Die Sprache awk stellt auch die arithmetischen Funktionen sqrt, log, exp und int zur Berechnung von Quadratwurzel, natürlichem Logarithmus, Exponentialfunktion und ganzem Teil der jeweiligen Argumente zur Verfügung.
Der Name einer dieser fest installierten Funktionen ohne Argument bzw. ohne Klammern liefert den Wert dieser Funktion für den gesamten Record. Das Programm

```
length < 10 || length > 20
```

gibt alle Zeilen aus, deren Länge kleiner als 10 bzw. größer als 20 ist.
Die Funktion substr(s, m, n) erzeugt die Teilzeichenkette von s, welche auf Position m beginnt (Anfangswert ist 1) und maximal n Zeichen lang ist. Falls n weggelassen wird, reicht die Teilzeichenkette bis zum Ende von s. Die Funktion index(s1, s2) liefert die Position, an der die Zeichenkette s2 in s1 auftritt bzw. Null, wenn sie nicht auftritt.
Die Funktion sprintf(f, e1, e2. ...) liefert den Wert der Ausdrücke e1, e2, usw. im printf-Format, welches durch f spezifiziert ist. Folglich setzt z.B.

```
x = sprintf("%8.2f %10ld", $1, $2)
```

x auf die Zeichenkette, die durch das Formatieren der Werte von $1 und $2 erzeugt wird.

3.2. Variable, Ausdrücke und Zuweisungen

In awk nehmen die Variablen entsprechend dem Kontext numerische (Gleitkomma-) bzw. Zeichenkettenwerte an. Zum Beispiel ist in

```
x = 1
```

x offensichtlich eine Zahl, während in

```
x = "Schmidt"
```

x offensichtlich eine Zeichenkette ist. Zeichenketten werden in Zahlen umgewandelt und umgekehrt, wann immer der Kontext das verlangt. Zum Beispiel ordnet

```
x = "3" + "4"
```

x den Wert 7 zu. Solchen Zeichenketten, die in einem numerischen Kontext nicht als Zahlen interpretiert werden können, wird generell der numerische Wert Null zugeordnet, aber man sollte auf diese Eigenschaft nicht bauen.
Standardmäßig werden alle nicht vordefinierten Variablen auf den Null-String (leere Zeichenkette) initialisiert, welcher den numerischen Wert Null hat. Dadurch werden die meisten der BEGIN-Sektionen nicht benötigt. Zum Beispiel können die Summen der ersten zwei Felder durch

```
        { s1 += $1; s2 += $2 }
END     { print s1, s2 }
```

berechnet werden.
Arithmetische Berechnungen werden intern in Gleitkommadarstellung ausgeführt. Die arithmetischen Operatoren sind "+", "-", "*", "/" und "%" (mod). Die Inkrement- bzw. Dekrement-Operatoren "++" bzw. "--", wie sie in der Programmiersprache C verwendet werden, sind ebenso verfügbar wie die Zuweisungsoperatoren "+=", "-=", "*=", "/=" und "%=". Alle diese Operatoren können in Ausdrücken verwendet werden.

3.3. Feld-Variable

In awk haben Felder alle wesentlichen Eigenschaften einfacher Variablen, sie können in arithmetischen oder Zeichenkettenoperationen angewendet und zugewiesen werden. Folglich kann man das erste Feld durch eine Folgenummer ersetzen, wie in:

```
{ $1 = NR; print $1,$2,$3 }
```

oder man kann zwei Felder in ein drittes akkumulieren, wie in:

```
{ $1 = $2 + $3; print $1,$2,$3 }
```

oder man kann einem Feld eine Zeichenkette zuweisen:

```
{ if ($3 > 1000)
     $3 = "zu gross"
  print $1,$2,$3
}
```

wobei hier das dritte Feld durch "zu gross" ersetzt wird, wenn dies zutrifft.
Bezugnahmen auf Felder können numerische Ausdrücke sein, wie in

```
{ print $i, $(i+1), $(i+n) }
```

Ob ein Feld numerisch oder als Zeichenkette behandelt wird, ist vom Kontext abhängig. In zweideutigen Fällen, wie in

```
if ($1 == $2) ...
```

werden die Felder als Zeichenketten behandelt.
Jede Eingabezeile wird automatisch so in Felder aufgeteilt, wie dies erforderlich ist. Es ist auch möglich, jede beliebige Variable bzw. Zeichenkette in Felder aufzuteilen, so teilt:

```
n = split(s, array, sep)
```

die Zeichenkette s in array[1], ... ,array[_n]. Die Anzahl der gefundenen Elemente wird zurückgegeben. Falls das sep-Argument angegeben ist, wird es als Feldtrennzeichen verwendet, anderenfalls wird FS als Trennzeichen verwendet.

3.4. Verkettung von Zeichenketten

Zeichenketten können verkettet werden. Zum Beispiel liefert

```
length($1 $2 $3)
```

die Länge der ersten drei Felder. Oder zum Beispiel in der print-Anweisung

```
print $1 " ist " $2
```

werden beide Felder getrennt durch "ist" ausgegeben. Auch Variable und numerische Ausdrücke können in Verkettungen erscheinen.

3.5. Arrays

Array-Elemente werden nicht deklariert. Sie werden in dem Moment erzeugt, wo sie im Programm erscheinen. Indizes können einen beliebigen, von Null verschiedenen Wert besitzen, wobei nichtnumerische Zeichenketten eingeschlossen sind.
Als Beispiel eines konventionellen numerischen Indexwertes weist die Anweisung

```
x[NR] = $0
```

den aktuellen Eingabe-Record dem NR-ten Element des Arrays x zu. In der Tat ist es vom Prinzip her möglich (obwohl vermutlich langsam), die gesamte Eingabe in beliebiger Reihenfolge mit dem awk-Programm

```
     { x[NR] = $0 }
END  { ... programm ... }
```

zu verarbeiten. Die erste Aktion überträgt lediglich jede Eingabezeile in das Array x.
Array-Elemente können durch nichtnumerische Werte benannt werden, wodurch awk Fähigkeiten erhält, die in etwa den Assoziativ-Speicher-Eigenschaften von

Snobol-Tabellen entsprechen. Angenommen, die Eingabe enthält Felder mit Werten wie Apfel, Orange usw. Dann werden durch das Programm

```
/Apfel/    { x["Apfel"]++ }
/Orange/   { x["Orange"]++ }
END        { print x["Apfel"], x["Orange"] }
```

Zähler für die genannten Array-Elemente inkrementiert, und nach der vollständigen Verarbeitung der Eingabe werden diese ausgegeben.

3.6. Steuerflußanweisungen

Die Sprache awk besitzt die elementaren Steuerflußanweisungen if-else, while, for und es können wie bei der Sprache C Verbundanweisungen durch geschweifte Klammern gebildet werden. Im Abschnitt 3.3 wurde die if-Anweisung gezeigt, ohne sie zu beschreiben. Die in Klammern angegebene Bedingung wird berechnet, und falls sich der Wert true ergibt, wird die dem if folgende Anweisung ausgeführt. Der else-Teil ist wahlfrei.
Die while-Anweisung entspricht vollständig der von C. Um z.B. alle Eingabefelder in der Form "jeweils eins pro Zeile" auszugeben, ist zu programmieren:

```
i = 1
while (i <= NF) {
     print $i
     ++i
}
```

Die fffooorrr-Anweisung entpricht ebenfalls genau der von C. Die Anweisung

```
for (i = 1; i <= NF; i++)
     print $i
```

leistet das gleiche wie die oben angegebene while-Anweisung.
Es gibt eine weitere Form der for-Anweisung, die geeignet ist, auf die Elemente eines assoziativen Arrays zuzugreifen. Die Anweisung

```
for ( i in array )
    anweisung
```

führt anweisung aus, wobei i der Reihe nach auf alle Elemente von array gesetzt wird. Auf die Elemente wird in einer sich zufällig ergebenden Reihenfolge zugegriffen. Dadurch kann Chaos entstehen, wenn i geändert wird, oder wenn während der Ausführung der Schleife irgendwelche neuen Elemente erzeugt werden.
Der Ausdruck im Bedingungsteil einer if-, while- oder for-Anweisung kann relationale Operatoren wie "<", "<=", ">", ">=", "==" ("gleich") und "!=" ("ungleich") enthalten, fest vorgegebene reguläre Ausdrücke mit den Entsprechungsoperatoren "~" und "!~", die logischen Operatoren "||", "&&" und "!" und natürlich die runden Klammern zum Gruppieren.
Die break-Anweisung veranlaßt ein sofortiges Verlassen eines klammernden while oder for. Die continue-Anweisung veranlaßt, daß die nächste Iteration beginnt.
Die Anweisung next veranlaßt awk, sofort zum nächsten Record zu springen und damit zu beginnen, die Muster wieder vom Anfang her durchzugehen. Die Anweisung exit bewirkt, daß sich das Programm so verhält, als ob das Ende der Eingabe erreicht wäre.
In awk-Programmen können Kommentare auf folgende Weise untergebracht werden: sie beginnen mit dem Zeichen "#" und enden mit dem Ende der entsprechenden Zeile,

wie z.B. in:

```
    print x, y      # das ist ein Kommentar
```

4. Zum Entwurf

Das System MUTOS enthält bereits verschiedene Programme, die so funktionieren, daß sie die Eingabe durch einen Selektionsmechanismus laufen lassen. Das erste und einfachste, grep, gibt einfach alle Zeilen aus, die einem einzelnen spezifizierten Muster entsprechen. Das Programm egrep verarbeitet allgemeinere Muster, d.h. reguläre Ausdrücke in voller Allgemeinheit. Das Programm fffgrep sucht nach einer Menge von Schlüsselworten mittels eines besonders schnellen Algorithmus. Der Strom-Editor sed besitzt die meisten Eigenschaften des Editors ed, angewendet auf einen Eingabe-Strom. Keines dieser Programme besitzt numerische Berechnungsmöglichkeiten, logische Relationen oder Variable.

Das Programm lex enthält allgemeine Erkennungsmechanismen für reguläre Ausdrücke, und besitzt, indem es als Generator für C-Programme dient, ein im wesentlichen offenes Ende bezüglich seiner Fähigkeiten. Die Benutzung von lex setzt jedoch die Kenntnis des Programmierens in C voraus, und ein lex-Programm muß vor seiner Benutzung compiliert und geladen werden, wodurch die Benutzung für einmalige Anwendungen erschwert wird.

Mit der Sprache awk wurde der Versuch unternommen, einen weiteren Teil des Spektrums von Möglichkeiten auszufüllen. Sie besitzt allgemeine Fähigkeiten bezüglich regulärer Ausdrücke und arbeitet das Eingabe-File zeilenweise über eine Eingabe-/Ausgabe-Schleife ab. Die Sprache awk besitzt auch bequeme numerische Verarbeitungsmethoden, Variable, allgemeinere Auswahlmöglichkeiten und die Möglichkeit der Beeinflussung des Steuerflusses in den Aktionen. Verlangt wird weder Compilierung noch Kenntnis der Programmiersprache C. Letztendlich besitzt awk eine bequeme Möglichkeit, um auf Felder innerhalb von Zeilen zuzugreifen. In dieser Hinsicht besitzt es gegenüber den anderen genannten Programmen einmalige Fähigkeiten.

Die Sprache awk unternimmt auch den Versuch, Zeichenketten und Zahlen vollständig zu integrieren, indem alle Größen sowohl als Zeichenketten als auch als Zahlen behandelt werden, wobei zum spätest möglichen Zeitpunkt entschieden wird, welche Darstellung die jeweils geeignete ist. In den meisten Fällen kann der Nutzer jedoch einfach die Unterschiede ignorieren.

Die Sprache awk besitzt eine leistungsfähige und trotzdem leicht verständliche Syntax, die der Bearbeitung von Files gut angepaßt ist. Zum Beispiel wäre das Nichtvorhandensein von Deklarationen und impliziten Initialisierungen sicherlich keine gute Idee für eine allgemeine Programmiersprache, aber für eine Sprache, die dazu gedacht ist, winzige Programme zu schreiben, die sogar nur aus einer einzigen Kommandozeile bestehen können, ist es durchaus angenehm.

In der Praxis bietet sich die Benutzung von awk für zwei große Anwendungsgebiete an. Das eine könnte man "Berichtsgenerierung" nennen (Verarbeitung einer Eingabe, um bestimmte Zählergrößen, Summen, Sub-Totale usw. zu extrahieren). Das schließt auch das Schreiben trivialer Daten-Prüf-Programme ein, wie z.B. die überprüfung, ob ein Feld tatsächlich nur numerische Informationen enthält, oder daß bestimmte Begrenzungszeichen in geeigneter Weise ausgerichtet sind. An dieser Stelle ist die gleichzeitige Verarbeitung von Text und numerischen Daten sehr wertvoll.

Ein zweites Anwendungsgebiet ist die Datentransformation, d.h. daß Daten von der Form, wie sie ein Programm erzeugt, in die Form umgesetzt werden, wie sie ein anderes Programm erwartet. Im einfachsten Fall werden lediglich bestimmte Felder ausgewählt, unter Umständen mit bestimmten Vertauschungen.

5. Implementierung

Zur Implementierung von awk sind die Sprachentwicklungswerkzeuge verwendet worden, die im Betriebssystem MUTOS verfügbar sind. Zur Formulierung der Grammatik sind die Möglichkeiten des yacc genutzt worden, die lexikale Analyse ist mittels lex ausgeführt worden. Die Erkennungsmechanismen für reguläre Ausdrücke sind deterministische endliche Automaten, die direkt aus den Ausdrücken konstruiert werden. Ein awk-Programm wird in einen Parser-Baum übersetzt, welcher dann durch einen einfachen Interpreter direkt ausgeführt wird.

Die Sprache awk ist nach dem Gesichtspunkt leichter Benutzbarkeit entworfen worden, wobei die Verarbeitungsgeschwindigkeit nicht so sehr im Vordergrund stand. Ohnehin bewirken die verzögerte Auswertung des Variablentyps und die Notwendigkeit, die Eingabe in Felder aufzuteilen, daß nur schwer hohe Verarbeitungsgeschwindigkeiten erreichbar sind. Trotzdem hat sich die Verarbeitungsgeschwindigkeit nicht als unvertretbar langsam erwiesen.

Bei der Lösung jeweils gleicher Aufgaben mit den Programmen wc, grep, egrep, fgrep, sed, lex und awk (im Rahmen der jeweiligen Möglichkeiten) hat sich gezeigt, daß awk erwartungsgemäß nicht so schnell ist wie die spezialisierten Werkzeuge wc, sed oder die Programme der grep-Familie, jedoch ist es schneller als das allgemeinere Werkzeug lex. (Hierbei wurden bei lex die Zeiten für das Compilieren und Laden nicht mitgerechnet.) In den meisten Fällen sind die Aufgaben als Programme in den genannten anderen Sprachen vergleichbar einfach wie die entsprechenden awk-Programme zu formulieren. Aufgaben, in denen auf Felder bezug genommen wird, sind als awk-Programme wesentlich einfacher darstellbar.