AWK の簡単な使い方

大阪大学 情報処理教育センター　荻原剛志

1　はじめに

本文書では、プログラミング言語（およびシステム）AWK[1] の簡単な使い方と、AWK の入門的な知識について説明をする。ユーザは MS-DOS に関する基礎的な知識を持っているものと仮定している。

本文書で説明している AWK の処理系は jgawk 2.11.1 + 2.0(〜2.9) 版である。基本的な事項のみについて述べているが、日本語の扱いなどは処理系によって異なることがあるので注意してほしい。

以下、ページの右に「☆」を付けた部分は、実際に計算機を使用して試してみる箇所を表している。

2　準備

2.1　プログラムとデータの用意

以下で説明する awk と fsort、および例題用のデータを前もって各自のドライヴ D（ハードディスク）にコピーしておくことにする。

これらは `s:¥new` に格納されている。ファイル名はそれぞれ `awk.exe`, `fsort.exe`, `satell.dat`, `idol.dat` である。これをドライヴ D にコピーするには以下のようにする。 ☆

```
D>cd s:¥new
D>s:scopy awk.exe fsort.exe satell.dat idol.dat
```

`scopy` は、S ドライヴの指定したファイルをドライヴ D 上にコピーするプログラムである。これまで用いていた `sload` と同じことがコマンドラインから行える（詳細は同じディレクトリの `scopy.doc` を参照）。

同じディレクトリにテキストエディタ Ng なども格納されているので、同様な方法でコピーしておいた方がよいだろう。

以上の操作が済んだら、`dir` コマンドで以下のファイルが存在することを確認しよう。 ☆

`awk.exe`	日本語 AWK の実行ファイル
`fsort.exe`	ソーティングプログラム
`satell.dat`	例題データ（衛星）
`idol.dat`	例題データ（アイドル）

2.2　例題データの内容

例題データの内容について簡単に説明する。これら 2 つのデータはテキストファイルであり、`more` コマンドなどで内容を画面で見ることができる。

(1) **衛星データ**

このデータは 43 行からなる、太陽系の惑星の衛星に関するデータである。始めの 3 行を示す。

```
火星 フォボス Phobos 11 14 1877 ホール
火星 デイモス Deimos 12 8 1877 ホール
木星 イオ　 Io　 5 1820 1610 ガリレオ
```

[1] AWK は「オーク」と読む。この言語の設計者 3 人の頭文字に由来する。

1つ、あるいは複数の空白で区切られた文字列を**フィールド**という。このデータのそれぞれのフィールドの意味は、左から、惑星名、衛星名、衛星の英語名、明るさ（等級）、半径 (km)、発見年、発見者、となっている。最近発見されたいくつかの微小衛星についてのデータは含まれていない。また、等級や半径のデータは必ずしも最新のデータではない。等級が確定していない場合には “?” が記入されている。

(2) **アイドルデータ**

このデータは 325 行からなる、いわゆるアイドル（女性）に関するデータである。始めの 3 行を示す。

```
相川恵里 あいかわ えり 720429 162 46 B 長崎 87 第 2 回ロッテ CM アイドル
相田翔子 あいだ しょうこ 700223 159 45 B 東京 Wink, ミス・アップ#9
藍田美豊 あいだ みほ 690831 163 42 A 神奈川 少女隊, 愛田美歩
```

それぞれのフィールドの意味は、左から、氏名（芸名）、姓の読み、名前の読み、生年月日（西暦末尾 2 桁と月日）、身長、体重、血液型、出身地、および備考である[2]。備考の部分のないデータもある。体重、血液型、出身地についてはデータが不明な場合があり、“?” が記入されている。また、「ゆうゆ」のように姓と名を分けられない場合、姓の読みには “-” が記入されている。

2.3 ソーティングプログラム

AWK と組み合わせて利用されることの多いソーティングプログラムについて説明する。

ソーティングとは「並べ換え」のことである。データを小さい順、あるいは大きい順などに並べ換えることを**ソートする**という。fsort はテキストデータを入力としてソーティングを行う。MS-DOS にもソーティングを行う SORT コマンドが用意されているが、fsort の特徴はフィールドを並べ換えのキーとして指定できる点である[3]。漢字を含むデータも扱うことができる。

フィールドの区切りは（連続した）空白かタブ。全角の空白は区切りと見なさず、また、行の先頭の空白は無視する。入力は常にリダイレクションによって行う。オプションには以下のものがある。

-n	文字列を数として解釈し、その大小によってソートを行う。このとき、数字、“+”、“−”、“.”(小数点) 以外の文字で始まる文字列は 0 と見なす。
-r	通常はデータの小さい順に並べ換えるが、これを大きい順に並べるようにする。
+数	並べ換えるための比較部分を、<数> +1 フィールド目とする。通常は 1 フィールド目（+0 とするのと同じ）だが、これを変更する。
-h	簡単な説明を表示させる。

簡単な使用例を示す。いま、衛星のデータを英語名の順（辞書式順序）で並べ換えることを考えよう。英語名は各行の第 3 フィールドにあるので、これを使って並び換えをするには、+2 というオプションが必要である。ソートした結果をさらに more で確認するには以下のようにする。 ☆

```
D>fsort +2 < satell.dat | more
```

ここに-r オプションを加えてみよう。逆の順序で出力されることがわかる。なお、オプションを並べる順序は特に決っていない。 ☆

```
D>fsort +2 -r < satell.dat | more
```

次に、衛星の半径でソートしてみよう。半径は第 5 フィールドにあるので、+4 とする。 ☆

```
D>fsort +4 < satell.dat | more
```

ところが、これではうまく行かないことがわかる。これは、数字を「数値」としてではなく、文字列として辞書式順序で並べ換えてしまうためである。このような場合は-n オプションを指定すると、指定したフィールドを「数値」として評価し、ソートを行わせることができる。 ☆

[2] データは主に i-dic(3.0.3) による。

[3] この点で UNIX の sort コマンドを模している。

```
D>fsort +4 -n < satell.dat | more
```

練習問題 1

1.1 アイドルデータを、生年月日、身長についてソートしてみよう。

1.2 衛星は明るいものから発見されたと考えてよいだろうか。衛星のデータを、等級、あるいは発見年でソートしてみて、この仮定について考察しよう。

3 AWK の簡単な使用例

この節ではまず、AWK の簡単な使用例をいくつか実際に試してみて、AWK でどのようなことができるのかを見てみることにする。以下はすべて実際に入力して試してみることができる。 ☆

3.1 コマンドラインからの実行

衛星のデータから英語名だけを取り出して表示してみよう。

```
D>awk '{print $3}' < satell.dat
```

リダイレクションの “<” はなくてもよい。
英語名、日本語の読みの順で表示し、さらにそれをソートしてみよう。

```
D>awk '{print $3,$2}' satell.dat | fsort
```

これらの例からわかるように、 “{ }” 内の記述が各データ行に対して実行される。また、“$” とその後に引き続く数で、データのフィールドを指定できる。
どこかに “on” という綴りを含むデータ行を表示しよう。

```
D>awk '/on/' satell.dat
```

“on” を含むデータ行について英語名と、括弧でくくった惑星名を表示する。

```
D>awk '/on/ {print $3, "(" $1 ")" }' satell.dat
```

このように、ある文字列を行のどこかに含むということを、{ } 内の処理を行うための条件とすることができる。条件となる文字列だけ記述するとその文字列を含む行全体が表示される。また、上の例での括弧のように、文字列を表示させるには一組のダブルクォーテーション (") でくくる。
1800 年以前に見つかった衛星について衛星名と惑星名、発見年、発見者を表示する。

```
D>awk "$6 <= 1800 {print $2,$1,$6,$7}" satell.dat
```

{ } 内の処理を行うための条件として、このような式を記述することもできる。“<=” は「より小さいまたは等しい」ことを表す。
ここで注意すべきなのは、“<” という記号（“>” も同様）がリダイレクションを表す特別な記号であるため、AWK に対する指定全体をシングルクォーテーション (') ではなく、ダブルクォーテーション (") で囲まなければならない点である。この場合、文字列を指定の中に含めることはできない。
このような不都合は、AWK への指定をコマンドラインから行おうとするためである。次の節で述べるように、AWK への指定（プログラム）をファイルに記述しておけば、このような問題は生じない。

3.2 プログラムファイルの実行

☆

テキストエディタ（Ng や Stevie など）を用いて、以下のような 1 行からなるファイル sat.awk を作成しよう[4]。

```
$6 <= 1800 {print $2,$1,$6,$7}
```

なお、sat.awk はドライヴ A に作成するものとしよう。ただし Ng などの emacs 系のエディタを用いる場合、最後の行の行末にも改行を入力しておくことが必要である。上の例の場合、“}” の次に改行を入力すること。

このようなファイルを作成しておいて、次のコマンドを実行してみよう。

```
D>awk -f a:sat.awk satell.dat
```

前の節で実行したように、1800 年以前に発見された衛星に関する情報が表示される。-f はファイルから AWK のプログラムを読み込むという指定である。

次に sat.awk を次のように変更し、同様にして実行してみよう。

```
$7 == "ガリレオ" {print $2,$1,$6,$7}
```

これは発見者がガリレオである衛星のみを表示する。“==” は「等しい」ことを表す。「ガリレオ」を囲むダブルクォーテーションは半角であることに注意。

アイドルデータに対するプログラムとして、以下のようなファイル id.awk をドライヴ A に作成しよう（行末の改行を忘れないように）。

```
($5 >= 160) && ($6 != "?") && ($6 < 45)
```

ここで、“&&” は「～かつ～」（論理積）を表す。また、“!=” は「等しくない」ことを、“<” は「本当に小さい」ことを表す。従って、このプログラムの意味は「身長が 160cm 以上で、かつ、体重のデータが記入されていて、かつ、体重が 45Kg 未満であるデータ行を表示する」である。これは上と同様に、以下のようにして実行できる。

```
D>awk -f a:id.awk idol.dat
```

「身長−100 に 0.9 を掛けたものが体重の目安」と言われる。そこで、アイドルデータについて

体重 / (身長 − 100)

を計算したらどうなるだろうか。次のようなファイルを作って実行してみよう。

```
$6 != "?" {print $1, $6/($5-100)}
```

さらに、結果を第 2 フィールドについてソートしてみよう[5]。

このように、フィールドのデータを用いた数式を記述することもできる。AWK 言語のさまざまな機能については次節で整理して述べる。

練習問題 2

2.1 月の半径は約 1738Km である。これより大きな半径を持つ衛星のデータ行を表示しなさい。

2.2 月の半径を 1 とした時の各衛星の半径の割合を計算して、名前と共に表示しなさい。

2.3 アイドルデータでは、グループのメンバーの場合は備考の部分にグループ名が記入されている。「CoCo」のメンバーの名前だけを書き出しなさい。

[4] 拡張子は awk でなくてもよいが、わかりやすさの点からこのように習慣づけるのがよい。
[5] このデータが作りものらしいことがわかるだろう。

4　言語の構成と機能

この節では、AWK の言語としての概要を示す。AWK は構文や演算子など、多くの要素を C 言語から取り入れている。

4.1　パターン-アクション文

4.1.1　プログラムの動き

最も簡単な AWK のプログラムは、以下のような**パターン-アクション文**の集まりである。

パターン { **アクション** }
パターン { **アクション** }
...

AWK は、入力した各行に対してプログラムのパターンを順番に評価し、パターンがマッチした場合に対応するアクションを起動する。以下の例は、アイドルデータに対するプログラムで、身長または体重の条件にあてはまった場合に { } 内が実行される。両方の条件に当てはまれば両方とも実行される。

```
$5 >= 165 {print $1, "身長:" $5 "cm"}
($6 != "?") && ($6 < 45) {print $1, "体重:" $6 "Kg"}
```

パターンは省略できるが、その場合、すべての入力行についてアクションが実行される。また、アクションも省略できるが、その場合はパターンに適合した入力行全体が表示される。

これまでの例で、パターンあるいはアクションが省略されているものはどれだろうか。

4.1.2　パターンの種類

パターンとして、以下のものを使うことができる[6]。

(1) **式** { **アクション** }

式の値を評価して、真（あるいは 0 でない数値）であればアクションを実行する。式とその値については **4.2**で述べる。

(2) **/パターン/** { **アクション** }

パターンが含まれている入力行に対してアクションを実行する。パターンには文字列、正規表現が使用できる。正規表現については **4.5**で述べる。

(3) `BEGIN` { **アクション** }

入力が読み始められる前に一度だけ実行される特別なパターンである。

(4) `END` { **アクション** }

入力をすべて読み終わってから実行される特別なパターンである。

4.2　式

4.2.1　定数

(1) **数定数**

数値を直接表現するもので、`100` のような整数表現、`3.1416` のような小数表現が可能である。

[6] これ以外にも存在するが、やや複雑なため省略する。

(2) **文字列定数**

"Symphony" とか "交響曲" といった、半角のダブルクォーテーションで囲まれたもの。英字（半角）と漢字（全角）を混在させてもよい。**改行**（行の終わりを表す）を表すために、特別に ¥n という 2 文字組を使う。「¥」自体を表すには ¥¥とする必要がある。同様に、「"」を表すには ¥" とする。

4.2.2 欄変数

例題プログラムでの $1, $2 といったもの。「$」の後にフィールド（欄）の番号 n を指定することで、その時読んだデータ行の n 番目のフィールドを表すことができる。

$0 という欄変数は特別に、その時読んだデータ行全体を表す。

4.2.3 組込変数

特別な意味を持つ変数がいくつか定められている（ここに挙げたものがすべてではない）。

NF	現在のデータ行にあるフィールドの数 (Number of Fields)
NR	いままでに読んだデータの行数 (Number of Records)
RSTART	組込関数 match で用いる
RLENGTH	組込関数 match で用いる（→ **4.2.6, 4.5**）

次のプログラムは、各行の先頭に行番号を付けて出力を行う。 ☆

```
{ print NR ": " $0 }
```

4.2.4 ユーザ定義の変数

変数とは、計算の結果である値や文字列を一時的に格納できる場所で、それぞれに名前が付けられる。変数の名前は基本的には（半角）英文字の連続としてユーザが自由に決めることができる。先頭に現れなければ数字を含んでいてもよい。例えば、a, type, kanjiname, alien2 のような名前を変数として使うことができる。

変数に値を割り当てることを、値を**代入する**という。代入は一般に

変数 = 式

という形式で記述できる（詳細は **4.2.5**で示す）。

ユーザ定義の変数は、使用された時に存在するようになり、始めの値としては 0、または空の文字列を持っている。

4.2.5 演算子

すでに例で用いたような +,− などを**演算子**と呼ぶ。以下ではこれらの意味と使い方を述べる[7]。

演算子と数や変数の間は、紛らわしくなければ続けて書くこともできる。適当に空白をあけて記述してもよい。例えば、 a=$1+$2 としても a = $1 + $2 としてもよい。しかし、ひとまとまりの記号、例えば++を + + というように記述してはいけない。

(1) **算術演算子**

+	加算	/	除算。 x/y は x と y の商を表す。
−	減算	%	剰余算。 x%y は x を y で割った余りを表す。
*	乗算	^	指数演算。 x^y は x^y を表す。

[7]条件演算子 "?:" は省略した。

数学の記法と異なり、x と y の乗算は x y とは書けない。必ず*を使って記述することに注意。

− は $-x$ のようにしても用いられ、 $-1 * x$ と同じ意味を表す。

*, /, % は +, − よりも**優先順位**が高い。つまり、数学での計算と同様に、乗除算と加減算が同時に現れる式では乗除算が優先される。例えば $x + y * z$ という式は $x + (y * z)$ のことである。加減算を先に行うには ( ) でくくればよい。例えば $(x + y) * z$ のようにする。

優先順位についてはこの節の最後で述べる。

(2) **関係演算子**

==	等しい	!=	等しくない
<	(左が) 小さい	>	(左が) 大きい
<=	(左が) 小さいか等しい	>=	(左が) 大きいか等しい

x==y は x の値が y の値と等しいかどうか調べ、等しい場合に「真」として 1、そうでない場合「偽」として 0 を関係演算の値とする。x!=y は逆に x と y が等しくない場合に 1、等しい場合に 0 を値とする。

<, >, <=, >= についても同様に評価する。ただし、文字列は辞書式順序に従って大小関係が評価される。例えば "ISIS" > "ISHTAR" は真である。

関係演算子の優先順位は算術演算子より低い。

(3) **論理演算子**

&&	論理積。 x && y は「x かつ y」を表す。
||	論理和。 x || y は「x または y」を表す。
!	論理否定。 !x は「x ではない」を表す。

通常は比較演算の結果をいくつか組み合わせて、より複雑な条件を構成するために用いられる。

&&は両側の 2 つの値が両方とも真の時に真となる。||は両側の値のどちらか一方、または両方が真の時に真となる。!は右側の値が真の時に偽、偽の時に真となる。

(4) **欄**

$を式の左に付けることで、その数の順番のフィールドを表すことができる。例えば、NF は組込変数 (→ **4.2.3**) で、その時読んだデータ行が持つフィールドの数を表している。従って、$NF は一番最後 (一番右) のフィールドを表し、$(NF−1) は最後から 2 番目のフィールドを表す。

欄を表す$は演算子としては一番優先順位が高い。従って、$(NF−1) において ( ) は必須である。$NF−1 は「最後のフィールドのデータから 1 引いた数」を表してしまうからである。

(5) **文字列の連接**

複数の文字列を結合して新しい文字列を作る。明示的な演算子はなく、文字列や数を順番に並べることで表現される。数は自動的に文字列に変換される。例えば、$1 と$2 の値がそれぞれ 7 と 10 ならば、$1 "月" $2 "日" は "7 月 10 日" という文字列を作る。

(6) **代入演算子**

AWK では代入も演算子のひとつとして考える。 $1 の値が 15 の時、 x = $1 + 10 とすると、変数 x は値として 25 を持つようになる。このとき、 x = $1 + 10 自体は式と見なすことができ、その値は x の新しい値 25 である。

代入演算子 "=" は、関係演算子 "==" と混同しやすいので注意が必要である。

x = x + 10 という式は、変数 x の値を 10 増やすことを意味している。AKW ではこれを簡単のために x += 10 と記述することができる。このような代入演算子には

```
+=    -=    *=    /=    %=    ^=
```

が用意されている。例えば x *= 2 は x = x * 2 のことを表している。

(7) **インクリメント・デクリメント演算子**

x = x + 1 あるいは x += 1 という演算はプログラムを記述する上でかなり頻繁に用いられる。そこで、AWK ではこの式を ++x と記述できる。同様に x = x - 1 という式は --x と記述できる。

この “++” をインクリメント演算子、“--” をデクリメント演算子という。

x++ と記述しても x の値を 1 だけ増やすことができるが、式として用いた場合には ++x と値が異なる。例えば x の値が 0 だったとすると、 y = ++x の場合 y の値は 1 になるが、 y = x++ とすると y の値は 0 である。つまり、 ++x は x の値を 1 つ増やしてからそれを式の値とするが、 x++ は x の値を式の値としてから x を 1 つ増やす。このような評価方法は --x、x-- についても同様である。

(8) **照合演算子**

~	パターンを含む	!~	パターンを含まない

演算子の左側の式が右側のパターンを含むかどうか調べる。例えば、x ~ /iz/ という式は、変数 x に iz というパターンが含まれている場合に真になる。変数 x の値が "ionization" などであれば真である。

パターンには正規表現（→ **4.5**）が使用できる。また、端末によっては “~” という文字が入力できないものがあるが、この時には代わりに “-” という記号を用いる。

(9) **配列要素**

配列 a に a[x] という要素が存在する場合に x in a は真になり、存在しなければ偽となる。配列については **4.4**で詳しく述べる。

(10) **優先順位について**

AWK には多数の演算子があるため、それらの間の優先順位に気をつかう必要がある。例えば、

```
a == b c + 2 || ! d && e != 1
```

という式はどういう順序で評価されるのだろう? 紛らわしい場合には積極的に ( ) でくくるべきである。

優先順位の高い順に左から並べる。

欄　++と--　^　!　負符号　乗除　加減　連接　関係　照合　in　&&　||　代入

4.2.6　組込関数

ある数や文字列から何らかの数、文字列を計算して値として返すものを関数と呼ぶが、**組込関数**とは AWK に既に備えられた関数のことを言う。例えば数学では sin を正弦関数として用いるが、AWK においても sin という組込関数があり、sin(x) とすることで x の正弦を計算することができる。

以下に数値を計算するための算術関数、文字列を操作するための文字列関数を示す[8]。

(1) **算術関数**

atan2(y, x)	y/x の逆正接	sqrt(x)	x の平方根 $\sqrt{x}$
sin(x)	x の正弦	exp(x)	x の指数関数 e^x
cos(x)	x の余弦	log(x)	x の自然対数 $log_e x$
int(x)	x の整数部		

[8] これら以外にもいくつか関数が用意されている。

三角関数はラジアンを用いる。int は小数点以下を切り捨てる。

いくつかの有用な定数がこれらを用いて計算できる。例えば atan2(0,-1) でπが、exp(1) で自然対数の底 e が計算できる。

(2) **文字列関数**

jindex(s,t)	s の中で最初の t の位置（なければ 0）を返す。
jlength(s)	s に含まれる文字数を返す。
jsubstr(s,p)	s の p 番目から始まる文字列を返す。
jsubstr(s,p,n)	s の p 番目から始まる長さ n の文字列を返す。
match(s,r)	rにマッチする部分文字列を s が含んでいるか調べる。 含んでいなければ 0 を、含んでいればその位置を返す（→ **4.5**）。
sprintf(...)	書式に従った文字列を作る（→ **4.3.8**）。

“j” で始まる名前の文字関数[9]は日本語に対応しており、半角文字も全角文字も 1 文字と考えて処理を行う。例えば、jlength("Bell の定理") の値は 7 であり、jsubstr("言語 APL",3) の値は"APL"となる。一方、関数 match は日本語に対応していないので注意が必要である。

4.2.7 データの型

AWK には数と文字列という２つのデータ型がある。通常のプログラムでは型は自動的に適切なものに変換されるため、あまり意識しなくてもよい。例えば 10 + "015" という式では "015" は自動的に数に変換され、計算結果は 25 となる。また、 10 "015" では 10 は文字列に変換され、結果は"10015" となる。

しかし、明示的にどちらかの型に変換させたいという場合もある。その場合には上で示した方法を利用して、例えば、数として比較したい時と文字列として比較したい時は次のように表現できる。x=10, y="015" と考えると比較の結果が異なることがわかるだろう。

x "" > y ""	文字列として比較
x + 0 > y + 0	数として比較

4.3 文

ここではプログラムの書き方について触れ、次いで、AWK が備えている文について説明する[10]。

4.3.1 プログラムの書き方

アクションの中には複数の文が記述できる。文は通常 1 行に 1 つずつ書く。あるいはセミコロン (“;”) で区切って 1 行に複数の文を記述することもできる。

左大括弧 (“{”) はパターンと同じ行に書かなくてはならないが、アクションや右大括弧は次の行以降に書いてもよい。

変数、定数や演算子、括弧などの間は、紛らわしくなければ続けて書くことができる。また、適当に空白（あるいはタブ）を置いたり、改行することもできる。しかし、ひとまとまりの記号やシンボル、例えば for を、 f or のように記述してはならない。

“#” は注釈の開始を示し、ここから行末までに注釈として任意の文字を記入することができる。先頭の文字が “#” である行、および空白行（何も書かれていない行）は無視される。

例えば、

[9] これらはこの処理系独自の日本語に対応した関数である。それぞれ “j” の付かない日本語非対応の関数が存在する。

[10] AWK には C 言語と同様に do-while, continue などの文も備えられているが、ここでは省略する。また、ユーザ定義関数についても本文書では述べない。

```
$6!="?" { x=$6/($5-100); print $1,x }
```

というプログラムと

```
$6 != "?" {
              x = $6 / ($5 - 100) # 体重/(身長 - 100)
              print $1, x
          }
```

というプログラムは等価である。

また、これはAWKの言語上の要請ではないが、構文が複雑になってくるとプログラムが読みにくくなりやすい。このため、行の先頭に空白を置いて、文の構造を把握しやすくすることが習慣として広く行われている。これを**段づけ**（あるいは字下げ: indentation）という。

練習問題 3

3.1 右のプログラムはアイドルデータに対して実行できる。このプログラムは何をするか考えてみなさい。

```
$5 > 165 { ++n }
END { print n "人" }
```

3.2 アイドルの身長の平均を計算するプログラムを作成しなさい。

4.3.2 if 文

if文は、ある条件が真の時と偽の時で実行する文を変えるために使われる。

```
if（式）文
```

この場合、式が真の場合に文が実行され、偽の場合は実行されない。

```
if（式）文1 else 文2
```

この場合、式が真の場合に$文_1$が実行され、偽の場合は$文_2$が実行される。

条件によって複数の文を実行させたい場合、それらの文を { } の対でくくる。次の例では、条件が真になった場合、2つの代入が実行される。

右のプログラムは、衛星データで最大の半径を持つものを探してその名前と半径を出力する。変数 max の値は最初 0 である。新しく読んだ半径がその時の max より大きければそれを max の新たな値とする。これを繰り返せば、最後には最大の半径が max の値になっているはずである。

```
{
    if ( $5 > max ) {
        max = $5
        name = $2
    }
}
END { print name,max }
```

4.3.3 while 文

ある条件が成り立つ間、繰り返しある文を実行したい場合に用いる。

```
while（式）文
```

まず式が評価され、真ならば文が実行され、偽ならば実行しない。文が実行された場合、文の実行後再び式が評価され、式が真である限り、文が繰り返し実行される。繰り返したい文が複数ならば { } でくくればよい。

右のプログラムは、入力されたデータのフィールドを1行にひとつずつ出力する。

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

4.3.4 for 文

この文も条件が成立する間、繰り返しを行う。

`for(`式$_1$`;` 式$_2$`;` 式$_3$`)` 文

これは以下の while 文と等価である。従って、while 文のプログラム例は右のようにも記述できる。

式$_1$
`while(`式$_2$`) {` 文`;` 式$_3$ `}`

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

4.3.5 空文

単独のセミコロンは**空文**を表す。空文は何もしないが、構文上ひとつの文と見なされる。右のような場合によく用いられる。このプログラムは “?” というフィールドをどこかに含むデータ行を表示する。

```
{  for (i = 1; i <= NF && $i != "?"; i++)
           ;
   if (i <= NF)
       print
}
```

4.3.6 break 文

while 文、for 文の繰り返しの途中から抜け出すのに使われる。これらの繰り返しが多重になっていた場合はひとつ外側に飛び出す。

右のプログラムは break 文を用いて上記の空文の例プログラムを記述しなおしたものである。

```
{  for (i = 1; i <= NF; i++)
       if ( $i == "?" ) break
   if (i <= NF)
       print
}
```

4.3.7 print 文

print 文は以下のようにカンマ (“,”) で区切られた複数の式を 1 行に表示する。

`print` 式$_1$, 式$_2$, . . . , 式$_n$

カンマで区切られた式と式の間には空白が入れられる。

`print`

は、`print $0` の省略形である。空白行を表示するには `print ""` のようにする。

print 文はまた、

`print` (式$_1$, 式$_2$, . . . , 式$_n$)

のように括弧でくくってもよい。式の中に関係演算子がある場合には括弧が必要である。

4.3.8 printf 文

printf 文は以下のどちらかの形式で記述される。ただし、式の中に関係演算子がある場合には括弧が必要である。

`printf` 書式, 式$_1$, 式$_2$, . . . , 式$_n$
`printf` (書式, 式$_1$, 式$_2$, . . . , 式$_n$)

書式とは、そのまま表示される文字列と、与えられた式をどのように表示するか指定する特別な文字列を組み合わせたものである。式の表示の仕方を指定する部分は “%” で示される。

例えば、衛星データについて英語名と半径を表示する場合を考えよう。これまでは以下のようにしていた。

```
{ print $3, $5 }
```

しかし、print 文では縦方向が揃わず、出力結果はあまり見やすくない。そこで、printf 文を用いて以下のようにしてみよう。

```
{ printf "名前:%-12s 半径:%5d¥n", $3, $5 }
```

ここで、`%-12s` という部分は`$3` の出力形式の指定で、文字列として 12 文字分の幅を取り、左寄せ（右を空白で埋める）で出力するということを表している。`%5d` は`$5` の出力形式の指定で、数字として 5 文字分の幅を取り、右寄せで出力することを表している。また、`¥n` とあるのは改行文字の指定（→ **4.2.1**）である。printf 文では 1 行を書き出すためにはこのように改行文字を明示的に記述しなければならない。これら以外の文字列、“名前:”、“半径:” はそのまま出力される。

書式指定は “%” から始まり、以下に示すような書式制御文字で終わる。

`d` 整数　　`f` 実数　　`s` 文字列

“%” とこれらの間に修飾子と呼ばれるものを含んでいてもよい。

%-幅. 桁数 `f`

-、幅、. 桁数はどれがあっても、どれがなくてもよいが、出現する順序はこの順番でなければならない。-は出力を左寄せにする。これを指定しなければ右寄せにされる。幅を指定すると、出力がその幅になるように適宜空白が詰められる。桁数は、`f` と共に用いた場合は小数点以下の桁数の指定となる。`s` と共に用いた場合は、対応する式（文字列）の先頭からその桁数分だけの文字を出力することを表す。

printf 文の出力の中に%を含めたい場合には%%とすればよい。

なお、全角文字を出力する場合、幅、桁数の計算では 1 文字が半角文字 2 文字分に相当する。

printf 文と同様な書式を用いて文字列を作成する関数に `sprintf` がある。この関数は printf 文で出力される文字列を、関数の値として返すものと考えればよい。

練習問題 4

4.1 アイドルの体重の平均を計算するプログラムを作成しなさい。結果は小数点以下 2 桁までを表示すること。

4.2 アイドルデータの中の “?” というフィールドの総数を数えて表示するプログラムを作成しなさい。

4.4 配列

AWK には他の言語にはないさまざまな特徴が備わっているが、以下で説明する配列も特色ある機能であり[11]、さまざまに応用して非常に便利に使うことができる。

4.4.1 配列の使用方法

配列はユーザ定義の変数（→ **4.2.4**）と同様の規則によって名前が付けられる。また、使用された時点で存在するようになるため、特別な定義や宣言は不要である。

[11] 連想配列と呼ばれる。

いま、blood という名前を配列の名前としよう。

```
blood["A"] = 1
```

配列はこのように、配列名の後に [ ] で囲まれた式（文字列）を付けたものとして用いる。そのようなひとまとまりが、ひとつの変数と同様にはたらくのである。上の例では blood["A"] が 1 を代入され、それを値とするようになっている。

```
    { blood[$7]++ }
END { print "A", blood["A"]
      print "AB", blood["AB"]
      print "B", blood["B"]
      print "O", blood["O"]
      print "?", blood["?"]
    }
```

この例の"A"のように、[ ] の中の式を配列の**添字**と呼ぶ。文字列 x が配列 a の添字としてすでに使われている場合、x in a の値が真になる（→ **4.2.5**）。

上のプログラムはアイドルデータについて、各血液型がそれぞれ何人いるか調べる。

4.4.2 配列用の for 文

前節のプログラムでは、調査対象が血液型だったので END のアクションに print 文を並べることができたが、では、出身県について同様なことをしようとしたらどうすればよいのだろうか。

ある配列の添字として使われた文字列をすべて取り出し、それらひとつひとつについて同じ処理を繰り返し適用するための文が用意されている。

```
for ( 変数 in 配列 ) 文
```

この for 文では配列の添字として使われた文字列を次々に変数にセットして文を繰り返し実行する。これを用いて、前節のプログラムは右のように簡単に書き直せる。さらに、$7 の部分を$8 に変更するだけで、出身県ごとの人数調査を行うことができる。

```
    { blood[$7]++ }
END { for ( type in blood )
          print type, blood[type]
    }
```

4.5 正規表現

AWK らしさという点で最も重要なのが、先の配列とここで述べる**正規表現**である[12]。正規表現は文字列を指定したり、照合したりするための記法である。

4.5.1 超文字

超文字（メタキャラクタ）とは、正規表現で特別なはたらきをする文字であり、AWK では以下のものを超文字とする。また、これら以外の文字を非超文字ということにする。

```
¥   ^   $   .   [   ]   |   (   )   *   +   ?
```

4.5.2 基本的な正規表現

(1) **非超文字**

A のような非超文字はそれ自体を表す。全角文字も 1 文字として扱うことができる。

(2) **文字列の最初と最後**

^ は文字列の先頭に、$ は文字列の最後にマッチする。

(3) **任意の 1 文字**

. は任意の 1 文字とマッチする。

[12] 正規表現は UNIX に一貫した思想とも言える。

(4) **文字クラス**

[ABC] は A, B, C のいずれかにマッチする。省略して、[A-Z] のようにすることもできる。これは半角の英大文字どれか 1 字にマッチする。[に続く文字が ^ であるような場合、例えば [^0-9] は、数字以外の任意の 1 文字にマッチする。

(5) **引用符付きの超文字**

¥* のように ¥ に引き続く超文字は超文字としては機能せず、その文字（例えば *）自体を表す。

4.5.3 演算子

算術演算と同様に、演算子を使って正規表現を大きくしてゆくことができる。

A|B	選択。A または B にマッチする。
AB	連接。A の直後に B が続く時マッチする。
A*	0 個以上の A、つまり空列、A、AA、AAA、... にマッチする。
A+	1 個以上の A、つまり A、AA、AAA、... にマッチする。
A?	空列、あるいは A にマッチする。
(r)	rが一致するのと同じ文字列にマッチする。

4.5.4 例

いくつかの例を挙げる。

^C	文字列の先頭にある C にマッチ
C$	文字列の最後にある C にマッチ
^...$	任意の 3 文字だけからなる文字列にマッチ
¥.$	文字列の最後にあるピリオドにマッチ
^[ABC]	文字列の最初にある、A, B または C にマッチ
^[^a-z]$	英小文字以外の任意の 1 文字からなる文字列にマッチ
AB*C	AC, ABC, ABBC, ABBBC, ... にマッチ
(AB)+C	ABC, ABABC, ABABABC, ... にマッチ
AB?C	AC、または ABC にマッチ
[a-zA-Z]+	1 文字以上の英文字からなる文字列にマッチ
A(B|C)D	ABD または ACD にマッチ
(¥+|-)?[0-9]+	省略可能な符号を持つ 1 文字以上の数字列

4.5.5 使用方法

正規表現はまず、文字列照合パターンとして使用できる。以下のプログラムは、アイドルデータに対して「○田○子」という名前を含むデータ行を表示する。 ☆

```
/.田.子/
```

また、次のプログラムはひらがなの名前を持つデータを表示する。 ☆

```
$1 ~ /[あ-ん]+$/
```

次に、文字列関数と組み合わせて使うことができる。match 関数は match(s,r) の形で用いて、文字列 s の中でパターン rが現れる最も左の位置を探す。パターン rには正規表現が使用でき、例えば、

```
match(text, /(S|s)h?iva/)
```

とすると、文字列 text に含まれる Shiva,Siva,shiva,siva のうち、最も左のものを探し、その開始位置を変数 RSTART にセットするとともに値として返す。マッチした長さは変数 RLENGTH にセットされる。

練習問題 5

5.1 衛星の英語名で “o” で終わる名前を持つデータ行を表示しなさい。

5.2 アイドルの名前（姓でなく）の読みのうち、「こみりかゆ」の５文字のどれかの組合せからなるもの（例えば「ゆか」など）のデータ行を表示しなさい。

5.3 上の問題で、名前の読みだけを表示するプログラムを作成しなさい。次に、同じ名前は２回以上表示しないようにしなさい。

5 例題

ここではこれまでに説明した事柄を利用した、小規模なプログラムをいくつか示す。いずれもアイドルデータに対して実際に実行することができる。

5.1 年齢の計算

データでは生年月日が 6 桁の数字で表現されているが、これをもとに年齢の計算を行う。現在の日付は BEGIN の部分に記述しておくとする。誕生日が過ぎていれば「現在の年 − 生まれた年」をそのまま年齢とするが、誕生日がまだ来ていない場合はそこから 1 を引く必要がある。

```
BEGIN { year = 91; mon = 8; day = 25; }
      { y = jsubstr($4,1,2) # 年
        m = jsubstr($4,3,2) # 月
        d = jsubstr($4,5,2) # 日
        old = year - y
        if ( mon < m || ( mon == m && day < d )) old--
        print $1, old "歳"
      }
```

5.2 月別の誕生日の数

各月毎の誕生日の数を数えるプログラムを作成しよう。下のプログラムの 1 行目は何をしているのだろうか。

```
      {   m = 0 + jsubstr($4,3,2)
          ++mon[m]
      }
END {   for ( i=1; i<=12; i++ )
            print i, mon[i]
      }
```

5.3 名前の漢字の頻度

配列を使い、名前の中でどんな漢字がよく使われているかを調べる。ただし、出現回数が 10 回未満のものは出力しないようにしてある。頻度の多い順に並べるには結果をソートすればよい。

```
      {
          n = jlength($1)                  # 名前の文字数
          for (i=1; i<=n; i++) {
```

```
            c = jsubstr($1,i,1)
            if ( c !~ /[あ-んア-ン]/ )  # ひらがな、カタカナは除く
                kanji[c]++
        }
    }
END {
        for ( c in kanji )
            if ( kanji[c] >= 10 )
                printf("%s %4d¥n", c, kanji[c])
    }
```

練習問題 6

6.1 アイドルの誕生年ごとの身長の平均を計算するプログラムを作成しなさい。

6.2 **5.3**のプログラムを変更して、名前（姓でなく）の読みの平仮名の頻度を調べるプログラムを作りなさい。さらに、名前が２文字（「みほ」など）の場合に限った出現回数も併せて調べられるようにしなさい。

6.3 アイドルの名前の読みについて、「りえ」と「りえこ」のように末尾に「こ」をつけると他のアイドルの名前になるような読みが他にもあるか調べるプログラムを作成しなさい。

6 おわりに

AWK を利用した場合、Pascal, C といった言語によってプログラムを作成するよりもはるかに容易に、しかも実用的な処理を行うことができる。もちろん、どのようなデータ処理にも適用できるというわけではないが、特に複雑なデータでない限り AWK は非常に有効である。

ここで述べた機能は基本的なもののみであるが、応用することでかなりいろいろな用途に適用可能であると考えられる。しかし、有用な機能でありながらここでは説明しなかったものもある。さらに興味のある人は以下の書籍が参考になるだろう。

A.V. エイホ, B.W. カーニハン, P.J. ワインバーガー（足立高徳:訳）
「プログラミング言語 AWK」, トッパン.

また、月刊アスキー, 1991 年 6 月号には jgawk 2.11.1 + 2.9 とともに弘前大学教育学部の小山智史氏の作成されたテキストが納められたフロッピーディスクが添付されている。

---

22 Sept. 1991 (Ver.1.1)
ogihara@asama.rd.ecip.osaka-u.ac.jp