2016 年 07 月 15 日

# 計算機実習 III (2016 年度)
# 第 12 回: AWK その 6

(http://takeno.iee.niit.ac.jp/%7Eshige/
math/lecture/comp4/comp4.html)

## 目 次



## 1 連想配列

AWK の配列の添字は、前に紹介したように 0 から始める必要はないが、実は負の数でも、さらに文字列でさえも添字に使うことができる。この性質により、AWK の配列は **連想配列** と呼ばれる。例:

```
BEGIN { a["One"] = 1; a["Two"] = 2 }
```

日本語化された gawk では日本語文字列も添字に使える。例:

```
BEGIN { week["日"] = "Sun"; week["月"] = "Mon"; ...
```

なお、このような配列の設定は、一つ一つ代入するより `split()` を使う方が良い。`split()` は添字を 1 からの連番にするので、以下のように 2 つの一時的な配列と `for` 文を利用すれば、上のような配列を構成できる。

```
BEGIN { split("日 月 火 水 木 金 土", tmpa)
    split("Sun Mon Tue Wed Thu Fri Sat", tmpb)
    for (j=1; j<=7; j++) week[tmpa[j]] = tmpb[j] }
```

こうすることで、例えば「week[tmpa[1]] = tmpb[1]」により「week["日"] = "Sun"」と設定されることになる。

また、以下のように各行に学生ひとりずつの情報が、学籍番号、学科、学年、性別、姓、名、身長、体重、の順に書かれているデータファイル student.txt があるとき、

```
201411234 機械 1 男 工科 太郎 171 68
201212091 情報 3 男 柏崎 次郎 165 81
201113123 環境 4 女 新潟 花子 158 46
201314253 建築 2 男 藤橋 匠 178 75
.........
```

このデータファイルに対して、

```
{ N[$4] ++; h[$4] += $7; w[$4] += $8 }
```

という AWK スクリプトで処理すると、例えば student.txt の 1 行目では、

```
N["男"] ++; h["男"] += 171; w["男"] += 68
```

のような作業が行われることになるので、配列 N[], h[], w[] の添字は "男" か "女" の文字列で、データ行をすべて読み終えた後では、

- N["男"] = 男性の人数、 N["女"] = 女性の人数
- h["男"] = 男性の身長の合計、h["女"] = 女性の身長の合計
- w["男"] = 男性の体重の合計、w["女"] = 女性の体重の合計

が保存されることになる。よって、上のスクリプトに END ブロックの処理を追加して、

```
{ N[$4] ++; h[$4] += $7; w[$4] += $8 }
END { print "男性の身長の平均 = ", h["男"]/N["男"]
    print "女性の体重の平均 = ", w["女"]/N["女"] }
```

とすれば平均値の出力が行える。このように、ある項目の値にいくつかの種類があるときは、連想配列を利用することで、種類毎の集計が容易に行える。

なお、これは添字を文字列にしなくても、場合分けにより 1, 2 のような添字にして、

```
{ if ($4 == "男") ind = 1; else ind = 2
    N[ind] ++; h[ind] += $7; w[ind] += $8 }
END { print "男性の身長の平均 = ", h[1]/N[1]
    print "女性の体重の平均 = ", w[2]/N[2] }
```

とすることもできる (C 言語では連想配列は使えないので、逆にこうしないといけない) が、$4 の値を直接添字に使う方が楽だし、より直感的でわかりやすい。

データから度数分布表を作成するときにも連想配列を用いることができる。例えば、student.txt の身長のデータから 10cm 区切り、すなわち 150cm から 159cm まで、160cm から 169cm まで、等の人数の度数分布表を作るには、

```
{ x = int($7/10) * 10; htable[x] ++ }
```

のようにすればよい。この「int($7/10) * 10」により、例えば 173cm は

```
int(173/10) * 10 = int(17.3) * 10 = 17 * 10 = 170
```

のように 1 の位を 0 にした数字に変換される。最後に表を作るために配列 htable[] の添字 (x) の値の最小値 (x1) と最大値 (x2) を保存する処理と、END ブロックの処理も追加すれば、例えば以下のようになる:

```
{ x = int($7/10) * 10; htable[x] ++;
    if (NR == 1) x1 = x2 = x; else if (x < x1) x1 = x;
    else if (x > x2) x2 = x }
END { for (x = x1; x <= x2; x += 10)
    printf "%3d ~ %3d : %d¥n", x, x + 9, htable[x] }
```

これにより、例えば、

```
150 ~ 159 : 2
160 ~ 169 : 8
.........
```

のように各範囲の人数が表示される。

このスクリプトでは、htable[] に各階級の人数を保存しているが、++ によるインクリメントの代わりに「htable[x] = htable[x] "#"」のように人数分「#」をつなげた文字列を作ってそれを最後に %s で文字列として表示させれば、

```
150 ~ 159 : ##
160 ~ 169 : ########
.........
```

のように度数分布表の代わりに簡易のヒストグラムを表示することもできる。

# 2 配列用の for 文

AWK の for 文には、C 言語と同じ

```
for ([初期設定]; [条件]; [増分設定])
```

の形式以外に、主に連想配列用に

```
for ([変数] in [配列名])
```

という形式があり、[変数] を [配列名] の配列の添字全部を動かして実行することができるようになっている。例えば、

```
BEGIN { a[10]="A"; a[13]="D"; a[15]="F"
    for (j in a) printf "a[%s] = %s¥n", j, a[j] }
```

とすると、2 行目の j は、配列 a[] の添字である 10, 13, 15 を 1 回ずつ取り、よってこれを実行すると

```
a[13] = D
a[15] = F
a[10] = A
```

のように表示される。なお、この例のように、for 文のこの形式では、j に代入される添字の順番は、必ずしも小さい順にはならないことに注意。

例えば、1 節の student.txt のデータに対し、

```
{ N[$2] ++; h[$2] += $7 }
END { for (ka in N) print ka, h[ka]/N[ka] }
```

とすると、N[ ], h[ ] は "機械" 等の学科名を添字とする配列となり、それぞれ学科毎の人数と身長の合計が入っているので、END ブロックの for 文ではその学科名とその学科の身長の平均値を、すべての学科に対して表示することになる。for 文のループ変数 ka は、この場合 N[ ] の添字である "機械"、"情報" などのすべての学科列を行きわたる。

この形式の for 文は、連想配列の添字があらかじめ容易には想定できない場合に便利である。例えば、データ hi-sports.txt に

```
新潟高校 陸上 体操 水泳 ...
新潟中央高校 陸上 ソフトテニス テニス ...
新潟南高校 ワンダーフォーゲル ダンス 野球 テニス ...
.........
```

のように、1 列目が高校名、2 列目以降はその高校にあるすべての運動部名が続いている場合、運動部名毎に、その運動部を持つ高校の数と高校名の一覧を作成するには次のようにすればよい。

```
{ for (j=2; j<=NF; j++) {
    N[$j] ++; sclist[$j] = sclist[$j] " " $1 } }
END { for (name in N)
    print name, ":", N[name], "校、" sclist[name] }
```

通常ブロックで処理する各行では、運動部名 ($j) を添字として高校数を保存する配列 N[ ] と高校名リストを保存する sclist[ ] を作成、更新している。文字列を単に空白を間に置いて書き並べれば文字列が連結されるので、「sclist[$j] = sclist[$j] " " $1」により文字列 sclist[$j] の最後に 1 列目の高校名が空白を空けてあらたに追加される。なお、このデータは各行の列数も行毎に違うので、2 列目以降の各列に対する処理を行う for 文は、2 から NF (= その行の列数) まで、とする必要がある。

そして END ブロックでは、「for (name in N)」により、配列 N[ ] の添字全部、すなわち存在するすべての運動部名を動く name という変数を使うループ処理になっていて、内部では name の値である運動部名、文字「:」、高校数 N[name]、文字列「校、」を出力し、最後に高校一覧 sclist[name] を出力する。

また、この形式の for 文を利用すると、簡単に「逆引きの配列」を作ることができる。例えば、この節の最初の例の配列 a[] に対して

```
BEGIN { a[10]="A"; a[13]="D"; a[15]="F"
    for (j in a) b[a[j]] = j
    for (j in b) printf "b[¥"%s¥"] = %s¥n", j, b[j] }
```

とすると、この 2 行目の for 文により a[] の添字と値を逆にした逆引き配列 b[] が作られ、

```
b["A"] = 10
b["D"] = 13
b["F"] = 15
```

のように表示される。

また、delete という命令で不要な配列の要素を削除することもできる。例えば、上の配列 b[] に対して、

```
delete b["D"]
```

とすれば、b[] は要素が 2 つの配列となる。

# 3 多次元配列

C 言語の a[1][2] のような多次元配列も AWK で利用できる。ただし、C 言語とは違い、「a[1, 2]」のように、一つの [ ] の中で複数の添字をカンマ (,) で区切って指定する。

例えば、1 節のデータファイル student.txt に対して 2 次元配列を利用して、

```
{ N[$3, $4] ++; h[$3, $4] += $7 }
```

とすれば、例えば student.txt の 1 行目では

```
N[1, "男"] ++; h[1, "男"] += 171
```

のような処理が行われ、学年と性別の 2 種類の情報を添字とする 2 次元配列が作られる。このように、添字の一方が数値で、他方が文字列でも構わない。

上のスクリプトに END ブロックの処理を追加して、

```
{ N[$3, $4] ++; h[$3, $4] += $7 }
END { for (j=1; j<=4; j++) {
    if (N[j, "男"] > 0)
        printf "%d年男 %fcm¥n", j, h[j, "男"]/N[j, "男"]
    if (N[j, "女"] > 0)
        printf "%d年女 %fcm¥n", j, h[j, "女"]/N[j, "女"] }}
```

のようにすれば、学年、性別毎の身長の平均が表示される。なお、if を使っているのは 0 割算エラーを避けるためである。

多次元配列は、実際には、添字の部分のカンマをシステム変数 SUBSEP が指す文字列に変え、数字は文字列に変換し、それらをすべて連結した文字列 (例えば N[1, "男"] の場合、"1" と SUBSEP と "男" をつなげた文字列) を添字とする「1 次元」の連想配列として実現されている。

システム変数 SUBSEP の値は、デフォルトでは ASCII 文字列ではないもの、すなわち連想配列の添字としては普通は使われないものが設定されていて、逆にそれを目印として後で分離することができるようになっている。

for 文の in を用いて多次元配列の要素を参照する場合は、多次元配列は実際には 1 次元配列として実現されているので、一つの for で 1 次元配列のように参照する必要がある。また、その添字は多次元の添字が連結された文字列だが、関数 split() で SUBSEP を区切りに指定して切り分けることで、元の多次元の添字に分離できる。

例えば、上のスクリプトの END ブロックを、「for (s in h)」のようなループで書き直すと、以下のようになる。

```
{ N[$3, $4] ++; h[$3, $4] += $7 }
END { for (s in h) {
    split(s, a, SUBSEP) # a[1], a[2] に学年、性別が復帰
    print a[1] "年" a[2], h[s]/N[s] "cm" }}
```

ループ変数の s は「"1" SUBSEP "男"」等の文字列であるが、split(s, a, SUBSEP) により a[1] に 1 が、a[2] に "男" が代入され、それにより多次元配列の添字をバラバラに利用できるようになる。

## コラム: 改行コードと日本語コード

OS 毎にテキストファイルの「改行」を表す文字コードは違いがあり、表 1 のようになっている。

| OS | 改行コード | 特殊文字表現 | バイト数 |
|---|---|---|---|
| MS–Windows | 0x0d 0x0a | ¥r¥n | 2 バイト |
| 旧 Mac など | 0x0d | ¥r | 1 バイト |
| Unix, MacOS X | 0x0a | ¥n | 1 バイト |

表 1: 改行コード

MS–Windows のテキストファイルでは改行コードを 2 バイトで表しているが、そのため MS–Windows 上の C 言語の printf() などでは、書式文字列に書かれた Unix 由来の改行文字 ¥n を、¥r¥n に変換してから出力している。また、MS–Windows の C 言語の fopen() には、"r", "w" の後ろに b をつけた "rb", "wb" (バイナリモード) が存在するが、これも同じ目的であり、"b" がついていない場合 (テキストモード) では上の逆の変換 (「¥r¥n」→「¥n」) をするが、"b" をつけた場合はその変換をしない。

しかし、Unix では改行文字は ¥n 1 バイトなので、そのような処理は不要で、fopen() の "b" もない (必要ない)。

使用している OS とテキストデータの改行コードが合っていないと、テキストファイルが改行されずに長い 1 行のデータとして表示されてしまったり、改行文字が制御文字として表示されてしまったりする。

同様に、日本語を表現するコードも OS などにより少しずつ違いがあり、それらが混在して使用されることが文字化けの一つの原因になっている。日本語は 1 バイト (0 ～ 255) では表現できないので、通常 2 バイト以上を組み合わせて表現されているが、現在よく使われている日本語コードはほぼ 4 種類ある。

| コード名 | 符号化方式名 | バイト数 | 利用環境 |
|---|---|---|---|
| 7 ビット JIS | ISO-2022-JP | 2 バイト/文字 以上 | 主にインターネット |
| 日本語 EUC | EUC-JP | 2 バイト/文字 | Unix |
| シフト JIS | Shift_JIS | 2 バイト/文字 | 旧 Mac, MS–Windows |
| ユニコード | UTF-8 | 主に 3 バイト/文字 | 最近多方面で |

表 2: 日本語コード

シフト JIS は「MS–漢字コード」や「cp932」などとも呼ばれ、MS–Windows の前の MS–DOS が日本に導入される際に作られた 2 バイトコードである。

ユニコードは、各国で別々に 8 ビット領域や 2 バイト文字などを使用していた状況を変え、4 バイトコード一つで世界中すべての文字を表現しようというプロジェクトであるが、その符号化方式として最も普及しているのが UTF-8 である。UTF-8 では大半の日本語文字は 3 バイトで表現されるため、他の文字コードに比べ日本語テキストファイルのデータサイズはやや大きくなる。

今後も日本語コードの混在の状況はしばらく続くと思われるので、それらがどういうものであるか、どのように対応すればいいかは、知っておく必要があるだろう。

改行コード、日本語コードを変換するフリーソフトなども色々なものが広く出回っているので、多人数でテキストデータを共有する場合、送られたメールが文字化けして読めない場合などに備えて、いずれかを導入し使えるようにしておくと便利だろうと思う。