2016 年 06 月 10 日

計算機実習 III (2016 年度)
第 8 回: AWK その 2

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

目 次

1 AWK の数学関数

AWK の数式で利用できる基本的な数学関数を表 1 に紹介する。

関数	意味
`sin(x)`	x のサインの値 (x はラジアン)
`cos(x)`	x のコサインの値 (x はラジアン)
`atan2(y,x)`	$\tan\theta = y/x$ となる θ の値 ($-\pi \leq \theta \leq \pi$)
`exp(x)`	e^x ($e = 2.71828\ldots$)
`log(x)`	$\log_e x$ ($= x$ の自然対数)
`sqrt(x)`	$\sqrt{x}$
`int(x)`	x の整数部分

表 1: AWK の標準的な数学関数

`atan2(y,x)` の値は、正確には座標 (x, y) の極座標での偏角になる。三角関数は、角の単位はラジアンなので、x の単位が度である場合は、

```
BEGIN { x=15; pi=3.1415926535; print sin(x*pi/180) }
```

のように $\pi/180$ 倍して変換する必要がある。なお、円周率 π は AWK では定義されていないので、上のように実際の値として定義するか、または `atan2()` を利用して「`pi=atan2(0,-1)`」とする。

AWK に用意されている数学関数は C 言語に比べれば少ないが、以下の公式を用いればいくつかの関数は補うことができる。

$$\begin{aligned}
\tan x &= \frac{\sin x}{\cos x} \quad (\cos x \neq 0 \text{ のとき}), & \sin^{-1} x &= \mathrm{atan2}(x, \sqrt{1-x^2}) \\
\log_a x &= \frac{\log x}{\log a} \quad (a \neq 1,\ a > 0 \text{ のとき}), & \cos^{-1} x &= \mathrm{atan2}(\sqrt{1-x^2}, x) \\
\sqrt[n]{x} &= x^{1/n} \quad (x > 0,\ n \geq 2), & \tan^{-1} x &= \mathrm{atan2}(x, 1)
\end{aligned}$$

例:

```
BEGIN { pi = atan2(0, -1)
    print "簡易三角関数表"; print "度 cos   sin   tan"
    for (j=0; j<90; j++) {
        printf "%2d", j
        t = j*pi/180; x = cos(t); y = sin(t); z = y/x
        printf " %5.3f %5.3f %5.3f¥n", x, y, z }
    printf "%2d %5.3f %5.3f %5s¥n", 90, 0.0, 1.0, "--" }
```

注意:

- C 言語では `1/3` と書くと 0 となる (整数の割り算) が、AWK はすべて実数計算なので 0.333... となる。整数の割り算値を得るには `int(x/y)` のようにする。
- 実数 x を、小数部分を四捨五入して整数にするには「`int(x+0.5)`」のようにすればよい。
- `int(x)` は、x が正でも負でもその整数部分を返すので、例えば、

  `int(3.4)`$= 3$, `int(3.0)`$= 3$, `int(-3.4)`$= -3$, `int(-3.0)`$= -3$

  となり、C 言語の `floor(x)` ($= x$ 以下の最大整数) や `ceil(x)` ($= x$ 以上の最小整数) とはいずれも異なる。
- x が整数 (正確には小数部分が 0 の実数) かどうかは、「`int(x) == x`」という条件式で判別できる。

2 AWK の乱数生成関数

AWK にも乱数を生成する関数があるが、これは整数ではなく、実数の乱数を生成する。

関数	意味
rand()	0.0 以上 1.0 未満の一様疑似乱数を返す
srand()	現在時刻に基づく rand() の初期値を設定

表 2: AWK の乱数関数

rand() には引数はなく、呼び出す度に違う値 (疑似乱数列) が返る。ただし rand() の実行前に、その乱数列の初期値を決める srand() を実行しておかないと毎回同じ乱数列が発生してしまう。逆に srand() は rand() を使う度に実行するのではなく、プログラムの中で一回だけ実行する。例:

```
BEGIN { srand()
    for (j=1; j<=100; j++) printf "%.7f¥n", rand() }
```

このスクリプトは、100 個の 0.0 以上 1.0 未満 (小数以下 7 桁) の乱数列を出力するが、最初の srand() のため、このスクリプトを実行する度に異なる 100 個の乱数列が出力される。もし srand() を入れないと、毎回同じ 100 個の乱数が並ぶことになる。

バッチファイルのときのように、0 以上 n 未満の整数の乱数を作りたい場合は、rand() の値を n 倍し、その整数部分を int() で取得すればよい。例:

```
BEGIN { srand(); print "4 回ジャンケン"
    s[0] = "グー"; s[1] = "チョキ"; s[2] = "パー"
    for (j=1; j<=4; j++) {
        x = int(rand()*3) # 1/3 ずつの確率で x は 0,1,2
        print s[x] } }
```

から行末までは無視されることに注意。

表示する文字列を 0, 1, 2 の添字を持つ配列 s[] に設定しておいて、0, 1, 2 の値を取るランダムな変数 x を生成し、それを添字とする配列要素 s[x] を表示している。

上は等分の確率の例であるが、if 文で場合分けすれば等分でない確率にすることもできる。

```
BEGIN { srand(); print "4 回ジャンケン";
    for (j=1; j<=4; j++) {
        x = rand()
        if (x < 0.2) print "グー"; # 確率 0.2
        else if (x < 0.7) print "チョキ"; # 確率 0.5
        else print "パー"; # 確率 0.3
    } }
```

3 時刻処理関数

AWK には現在時刻の取得や、日時の書式つき表示用の関数もある (表 3)。

関数	意味
`systime()`	Unix epoch 時から現在時刻までの秒数
`strftime(f {,n})`	Unix epoch 時からの秒数 n (デフォルトは現在時刻[1]) を書式文字列 f に従って書き直した文字列を返す

表 3: 時刻に関する関数

「Unix epoch 時」とは「1970 年 1 月 1 日 00:00:00」を指す。時刻はその時点からの絶対秒数で管理されているので、2 つの時刻の差を取る場合など、数式での計算に便利。例:

```
BEGIN { t = systime() # 最初の時刻
    print strftime("現在時刻: %H:%M:%S")
    do { s = systime() } while (s < t + 5)
    print strftime("現在時刻: %H:%M:%S") }
```

これは、スクリプト開始から 5 秒経つまで `systime()` の値を do-while 文で取得し続け、5 秒後に終了する。この例の `strftime()` のように、2 番目の引数を省略した場合は現在の時刻が対象となる。なお、`strftime()` は文字列を返すだけなので、それを出力する場合は `print`, `printf` が必要となる。

`strftime()` の書式文字列で使える書式化文字列の主なものを表 4 にあげる。

[1]以後、関数の引数に出てくる {,n} のような { } で囲んだ部分は「省略できる引数の部分」を意味し、「デフォルト」がそれを省略した場合の値や動作を意味するものとする。

記号	意味	記号	意味
%Y	西暦 (4 桁)	%H	時 (00–23)
%y	西暦 (下 2 桁)	%M	分 (00–59)
%m	月 (00–12)	%S	秒 (00–59)
%d	日 (00–31)	%I	12 時間制の時 (01–12)
%a	曜日名 (“日”–“土”)	%p	午前/午後

表 4: strftime の書式で使える主な書式化文字列

例:

```
BEGIN { x = systime()
    y = x - 3*24*60*60; z = x - 7*24*60*60
    print strftime("今日: %Y 年 %m 月 %d 日 (%a)")
    print strftime("三日前: %Y 年 %m 月 %d 日 (%a)", y)
    print strftime("一週間前: %Y 年 %m 月 %d 日 (%a)", z) }
```

これは、今日、三日前、一週間前の日付、曜日を、月の変わり目などをまたいでも正しく表示する (自前でやると結構難しい)。

この例のように、strftime() の書式文字列は printf とは違い、% で始まる複数の書式化文字列を使用しても、時刻部分 n はひとつだけ指定する。

4 文字列処理関数

AWK では、文字列も変数値として重要であるが、そのため文字列を処理する関数も色々用意されている。表 5 にその一部を紹介するが、正規表現に関連する AWK 独特の関数については後で取り上げることとする。なお、これらの関数は、いずれも元の文字列 (引数) を変更しない。

この他にも、文字列の連結は、文字列を空白区切りで並べることで行える。例えば「s = "abc" "def" "ghi"」は「s = "abcdefghi"」と同じになる。

この関数を用いた例を少し紹介するが、本格的な使用例は、第 9 回以降で説明する。

substr() は文字列を一部分切り出すのに利用できる。例:

関数	意味
length(s)	s の長さを返す
tolower(s)	s のアルファベットを小文字にした文字列を返す
toupper(s)	s のアルファベットを大文字にした文字列を返す
sprintf(f, ...)	printf() の出力と同じ文字列を返す
index(s1, s2)	文字列 $s1$ 内の $s2$ の先頭位置を返す
substr(s, n {,m})	文字列 s の場所 n から長さ m (デフォルトは最後まで) の部分文字列を返す
split(s, a)	文字列 s を半角空白区切りで切り分けて配列 a に保存し、切り分けた個数を返す

表 5: 文字列に関する関数

```
BEGIN { s = "abcdefghijklmnopqrstuvwxyz"; N = length(s)
    for (j=1; j<=N; j++) {
        c = substr(s, j, 1)
        printf "[%d] %s %s¥n", j, c, toupper(c) }}
```

これは、アルファベットの小文字と大文字 (toupper() で変換したもの) を順に表示する。substr(s, j, 1) は、s の j 番目の長さ 1 の文字列、すなわち 1 文字を取り出す。

```
BEGIN { s = "http://takeno.iee.niit.ac.jp/%7Eshige/"
    s = s "math/lecture/comp4/comp4.html"
    t = "lecture"; len = length(t)
    n = index(s, t) # s 内の t の先頭位置
    s1 = substr(s, 1, n - 1) # その前
    s2 = substr(s, n, len)   # t
    s3 = substr(s, n + len)  # その後
    print s1; print s2; print s3 }
```

これは、長い文字列 s (この講義のホームページの URL) の中の部分文字列 t の位置を index() を用いて検索し、s を、その前の部分 s1、その部分 s2、その後の部分 s3 の 3 つに substr() を用いて分離している。最後の substr() は 3 つ目の長さ引数が指定されていないが、その場合は文字列の最後までとなる。

split() はスペース区切りの文字列を配列に切り分けるので、配列を初期化するのに利用できる。例:

```
BEGIN { sj = "睦月 如月 弥生 卯月 皐月 水無月 文月 葉月"
    sj = sj " 長月 神無月 霜月 師走"
    split(sj, mj)
    se = "January February March April May June July"
    se = se " August September October November December"
    N = split(se, me)
    for (j=1; j<=N; j++)
        printf "%2d: %6s %s¥n", j, mj[j], me[j] }
```

2 行目、5 行目は、それぞれ 1 行目、4 行目で書けなかった残りの部分を連結していて長い文字列にしている。そして、その日本語名と英語名の月名を `split()` で `mj[]`, `me[]` に配列化している。`split()` では配列の添字は、1 番から順番に使われ、例えば `mj[1]` が「睦月」、`me[2]` が 「February」となる。「`mj[1]="睦月"`」のように順に 12 個代入するより `split()` を使う方がずっと楽に初期化できる。

`sprintf()` は、書式化した文字列を文字列として利用できる。例えば、長さ 20 の空白文字列を作るには「`s = sprintf("%20s", "")`」でよい。なお `printf` とは違い、`sprintf()` には ( ) が必要。

コラム: スクリプト言語

現在は、実はスクリプト言語全盛といってもいいくらいスクリプト言語がよく使われている。AWK よりもむしろ有名なスクリプト言語に、Perl, PHP, Python, Ruby などがあり、これらは Unix, MS–Windows など多くの環境で動作し、例えば大きな Web サイトの構築などで広く使われている。

MS–Windows 上では、VBScript や JScript, ASP など Microsoft が提供するスクリプト言語もあるし、フリーの MS–Windows 用のスクリプト言語としては、簡単なゲームなどを作るのに使われている HSP や、日本語でプログラミングできる「なでしこ」などがある。

以上はいずれも汎用のスクリプト言語であり、広い目的で使用されるが、Web アプリケーション専用の JavaScript や PHP, Flash アニメーションの記述を行う ActionScript、アプリケーションの GUI インターフェース部分のみを作成する Tcl/Tk、グラフ描画ソフト gnuplot の命令スクリプトなど、特定の用途専用のスクリプト言語もある。バッチファイルもスクリプト言語の一つと言えるだろう。

そもそも「スクリプト」(script) とは「台本」を意味する言葉で、バッチファイルのよ

うに命令を順次実行させるよう並べて書いたようなものをスクリプトと呼び、一般的には以下のような性質を持ったものをスクリプト言語と呼んでいるようである。

- 動作はインタプリタ型 (コンパイルせずに動作させる)
- コンパイル型言語と比べて型宣言などが厳密でない
- コンパイル型言語よりも短い命令で一定の仕事ができる
- 動作速度はあまり早くない

Perl や Python, Ruby は特に広く使われているスクリプト言語であるが、これに共通しているのは以下のような性質であり、そのために好まれているのだろう。

- オープンソースなフリーソフト
- 多くの環境で動作する
- ライブラリを追加することで容易に機能拡張が行える
- 書籍も多く出版されている
- AWK よりは高機能で、C 言語よりは楽に色々なことが行える
- 正規表現をサポートしている
- 行フィルタ作業も行えるし、バッチファイルのようなファイル、ディレクトリ、外部コマンド操作も行える

特に Web アプリケーションの処理は文字列処理が多いので、このような汎用のスクリプト言語が頻繁に利用されている。

C 言語をある程度学んだら、Perl や Ruby などのスクリプト言語をひとつ勉強し使えるようにすると、色々な意味で便利だろうと思う。