2012 年 05 月 25 日

計算機実習 IV (2012 年度)
第 6 回: AWK その 1

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

目 次

1 AWK の概要

AWK (おーく) は、スクリプト言語と呼ばれるインタプリタで、次のような特徴を持っている。

- プログラムの文法、書式は C 言語に非常に近い
- Unix には標準に装備されていて、MS–Windows で動作するフリーの AWK もある。
- AWK は行単位のフィルタであるが、(特に GNU 版の AWK は) C ライクな簡易インタプリタとしても使える
- 正規表現や連想配列、文字列のフィールド分割など、C にはない機能もある
- 関数定義もでき、サブルーチンのライブラリ化も可能

本講義では、以下で公開されている GNU 版の AWK インタプリタ (コマンド名 gawk) を使用する

- マルチバイト対応版の GNU AWK 3.1.5 (Win32 バイナリ)
  http://www.vector.co.jp/soft/dl/win95/util/se376460.html

AWK プログラム (**スクリプト** と呼ぶ) は、短かくて 1 行で済むものならばファイルに保存しなくても

```
Z:> gawk "1 行スクリプト"
```

のように二重引用符 " " で囲んで直接指定できるが[1]、ファイルに保存する場合は test.awk のような名前のファイルに保存し、

```
Z:> gawk -f test.awk
```

のように -f オプションの後ろにスクリプトファイル名を指定して実行する。なお、AWK はインタプリタなので、C のようなコンパイルの作業は必要ない。

otbedit で AWK スクリプトを編集する場合は、.awk という拡張子でスクリプトを保存したものを読み込ませるか、または上にある AWK というアイコンで AWK モードにすることができる[2]

課題 6-1. gawk --help を実行し GNU AWK のヘルプを表示させてみよ。ヘルプが長い場合は more にパイプで渡せばよい。

課題 6-2. gawk --version で GNU AWK のバージョンを確認せよ。

課題 6-3. otbedit 上でアイコンで AWK モードにして、後のページにあるスクリプトの一部分を入力してキーワードが色付けされることなどを確認せよ。

AWK スクリプトの行フィルタとしての構造は次回以降で紹介し、今回は AWK の言語仕様や書式を確認するために、以下のような特別な構造のスクリプトのみを書くことにする:

```
BEGIN {
    (プログラム本体)
}
```

これは、C 言語で言えば、入力のない main() だけのプログラムのようなもので、gawk にこの形式のスクリプトを処理させると、処理後に直ちに終了する。

まず最も基本的な関数として、以下の 2 つを紹介する。

[1] 1 行のスクリプトでもかなり色んなことができてしまうが、このような使い方を愛称で「一行野郎」と呼ぶことがある。

[2] otbedit の AWK モードではファイルは .awk で保存される。

- printf : C 言語と同様、%d, %s, %f などの書式指定により変数値の表示が可能な文字列出力関数。
- print : printf とは違い、% の変数値の表示機能は持たず、指定した文字列をそのまま出力 (複数指定した場合は空白区切りで出力) し、最後に改行する関数 (printf は ¥n をつけないと改行しない)。

なお、C 言語とは違い、両者とも ( ) は必要ない[3]。また、1 行スクリプトの二重引用符 (") 内で " を使うときは、外側の引用符の終りとみなされないよう ¥" と書く必要がある。

課題 6-4. printf と print を使って、「こんにちは。」を 2 行表示する AWK スクリプト kadai6-4.awk を作成せよ。

課題 6-5. 「printf "%d¥n", 123+456;」と「print "%d¥n", 123+456;」の結果を比較してみよ。

2 AWK の文法

AWK の文法は C 言語と非常に近いが、違うところもある。

1. 変数
   - 変数宣言は不要。
   - 変数値は数値か文字列で、数値には整数型はなく、すべて実数型。文字型はなく、すべて文字列。
   - 文字列は " " で囲んで指定する。
   - 数値を表わす文字列は、文字列としても使えるし、数値としても使える。明示的に数値に変換する場合は、後ろに +0 をつければよい。
   - 明示的に初期化されていない変数は、数値としては 0 を、文字列としては空文字列が初期値となる。
2. 書式
   - コメントは # から行末まで。

[3] ( ) をつけても問題なく動作する。

- C 言語同様、文単位で実行されるが、複数の文は { } で囲んでグループ化できる。
- C 言語同様、空白は任意に書けるが、改行後も文が続く場合は、その改行の手前 (行末) に ¥ が必要になる場合がある。
- 1 行に複数の文を書く場合は ; で区切る[4]

3. 演算子、構文
   - 四則演算、比較演算、論理演算子、代入演算子、3 項演算子などの演算子は、ほぼ C 言語と同じものが使える
   - x^y (累乗) は x^y と書ける (負の値の非整数乗は不可)
   - 文字列を空白を空けて 2 つ並べるとそれが連結された文字列を表す
   - if 文、while 文、do while 文、for 文も C 言語と同じものが使える
   - switch 文は AWK にはない
   - 条件文での真偽は、数値としては 0、文字列としては空文字が偽で、それ以外が真となる。

簡単な例を紹介する。

```
BEGIN {
    x = 3; y += 2;
    print x + y;
    for (j=1; j<=5; j+=2)
        z += j;
    print z;
    if (x+y < z)
        print "x + y < z"
    else
        print "x + y >= z"
}
```

というスクリプトを gawk に処理させると、「8」「9」「$x + y < z$」を表示する。y は明示的に初期化されていないが、その場合初期値は 0 になる。行末のセミコロンは必ずしも必要はない。

また、

[4]文末に ; を書けるところは C 言語に似ているが、C とは違い、行末には ; をつける必要がない。しかし、C 言語と同様にすべての文末に ; をつけても問題はないので、C 言語と同じ書き方をする方がいいかもしれない。

```
BEGIN {
    s1 = "2012"; s2 = "15001"; # 2 つの文字列
    s3 = s1 s2; # 2 つの文字列を連結
    print s3;
    # 文字列としての比較（辞書順）
    if (s1 < s2) print "s1 < s2"
    else print "s1 >= s2"
    # 数値としての比較（+0 をつけることで数値化される）
    if (s1+0 < s2+0) print "s1 < s2"
    else print "s1 >= s2"
}
```

というスクリプトを gawk に処理させると、「201215001」「s1 >= s2」「s1 < s2」を表示する。

課題 6-6. 2012 と 366 の和、差、積、商、整数商の余りを表示する AWK スクリプト kadai6-6.awk を作成せよ。

課題 6-7. for 文を使って 1 から 100 までの和を計算して表示する AWK スクリプト kadai6-7.awk を作成せよ。

課題 6-8. for 文を使って $1^2+2^2+3^2+\cdots+100^2$ を計算して表示する AWK スクリプト kadai6-8.awk を作成せよ。

課題 6-9. 「`copy test1.txt t001.txt`」、「`copy test2.txt t002.txt`」、...「`copy test20.txt t020.txt`」の 20 個のコマンドを実行するバッチファイル kadai6-9.bat を出力する AWK スクリプト kadai6-9.awk を作成せよ。

AWK でも C 言語と同様の形式の配列が利用できるが、配列の値は普通の変数と同じで数値か文字列である。また配列の添字は、C 言語とは違い 0 から始める必要はない。

実は、AWK の配列は C 言語とは大きく違っているため「連想配列」と呼ばれているのであるが、これについては後で紹介する。多次元配列も C 言語とは形式が異なる。これも後で紹介する。

課題 6-10. a[1] から a[100] の 100 個の配列に 150.0 以上 170.0 未満の乱数を代入し、その平均値、およびそれぞれの 2 乗の値の平均値を計算する AWK スクリプト kadai6-10.awk を作成せよ。

3 AWK の標準関数

AWK (gawk) に用意されている標準関数を以下に紹介する (文字列処理関数については、次回以降説明する)。

- `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 $(-\pi \leq \theta \leq \pi)$[5]
- `exp(x)` : e^x
- `log(x)` : $\log_e x$ (x の自然対数)
- `sqrt(x)` : $\sqrt{x}$
- `int(x)` : x の整数部分
- `rand()` : 0.0 以上 1.0 未満の一様疑似乱数 (毎回違う値を返す)
- `srand(x)` : `rand()` の初期値を決めるパラメータ (種) を設定。x を指定しなければ現在の時刻にもとづいた種を設定する。
- `systime()` : Unix epoch 時からの秒数[6]
- `strftime(f [,n])` : Unix epoch 時からの秒数 n を、書式 f に従って書き直した文字列を返す (文字列値関数)。n を省略した場合は、現在の時刻に従う[7]。使える書式文字列の主なものは以下の通り:

%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[8]

[5] よって $\tan^{-1}(y/x)$ (主値) とは異なる場合がある。より正確にいえば、(x, y) の座標を極座標で表現した場合の偏角を意味する。

[6] Unix epoch 時とは 1970 年 1 月 1 日 00:00:00 を指す。

[7] AWK の関数にはこのように引数を省略できるものがいくつかある。

[8] 曜日名、午前/午後、および %B による月名は、英語環境の MS–Windows では、Sunday, AM/PM, January などと表示される。

用意されている数学関数は、C 言語に比べれば少ないが、以下の公式などを用いれば他の関数を補うことができる。

$$\begin{aligned}
\tan x &= \frac{\sin x}{\cos x} \quad (\cos x \neq 0 \text{ のとき}) \\
\log_a x &= \frac{\log x}{\log a} \quad (a \neq 1,\ a > 0 \text{ のとき}) \\
\sqrt[n]{x} &= x^{1/n} \quad (x > 0,\ n \geq 3 \text{ のとき}) \\
\sin^{-1} x &= \mathrm{atan2}(x, \sqrt{1-x^2}) \\
\cos^{-1} x &= \mathrm{atan2}(\sqrt{1-x^2}, x) \\
\tan^{-1} x &= \mathrm{atan2}(x, 1)
\end{aligned}$$

```
BEGIN {
    x = 2;
    printf "x の 2 乗 = %f", x*x;
    printf "x の 1/2 乗 = %f", sqrt(x);
    # または x^(1/2)
    printf "x の 1/3 乗 = %f", x^(1/3);
    printf "log_10(x) = %f", log(x)/log(10);
}
```

なお、C 言語では `1/3` と書くと整数の割り算を意味し 0 となるが、AWK はすべて実数計算なので 0.333... の値を意味する。

三角関数は、角の単位はラジアンなので、x の単位が度の場合は、

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

のようにする必要がある[9]。

課題 6-11. $\sin x$, $\cos x$ の $x = 0°$ から $x = 90°$ までの $1°$ 刻みの表を出力する AWK スクリプト kadai6-11.awk を作成せよ。

課題 6-12. $\log_2 x$, $\log_{10} x$ の $x = 1.0$ から $x = 10.0$ までの 0.1 刻みの表を出力する AWK スクリプト kadai6-12.awk を作成せよ。

[9]円周率は AWK では定義されていない。

課題 6-13. 平均律音階の周波数 $440.0 \times 2.0^{n/12.0}$ (Hz) の数値を、$n = -12$ から $n = 12$ まで出力する AWK スクリプト kadai6-13.awk を作成せよ。

日付は `strftime()` に書式文字列と、必要なら `systime()` を用いた時刻を指定して作成し、それを `print` などで出力させるが、現在の日付 (時刻) を表示させる場合は `strftime()` の 2 つ目の引数は不要。なお、printf とは違い、複数の書式文字列を使用しても、時刻部分 n の指定はひとつでよい。例:

```
BEGIN {
    print strftime("今日は %Y 年 %m 月 %d 日");
    print strftime("2 時間後の時刻は %H:%M:%D",¥
        systime()+60*60*2);
}
```

課題 6-14. 現在の日時を、「`YYYY-TT-DD HH:MM:SS`」と表示する AWK スクリプト kadai6-14.awk を作成せよ。ここで、YYYY は西暦 4 桁、TT は月 (00–12)、DD は日 (00–31)、HH は時 (00–23)、MM は分 (00–59)、SS は秒 (00–59) とする。

課題 6-15. 明日の日付を、今日がいつであっても正しく「`YYYY-TT-DD`」の形式で表示する AWK スクリプト kadai6-15.awk を作成せよ。

乱数は、`rand()` を呼び出す度に 0 以上 1 未満の実数値の疑似乱数 (一様乱数) が取り出せる。ただし、最初に 1 度 `srand()` を実行しておかないと、毎回同じ乱数列が生成されてしまう。例:

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

if 文の部分は、配列を使えば以下のようにすることもできる。

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

課題 6-16. 0 か 1 を 1/2 ずつの確率でランダムに 10 回表示する AWK スクリプト kadai6-16.awk を作成せよ。

課題 6-17. 1 から 6 までの数を 1/6 ずつの確率でランダムに 10 回表示する AWK スクリプト kadai6-17.awk を作成せよ。

課題 6-18. 0 か 1 を、0 が 3 割、1 が 7 割の割合でランダムに 10 回表示する AWK スクリプト kadai6-18.awk を作成せよ。

コラム: AWK の作者と歴史

AWK (おーく) の名前の由来は、AWK の 3 人の作者 A.V.Aho (エイホ)、P.J.Weinberger (ワインバーガー)、B.W.Kernighan (カーニハン) の名前の頭文字から来ている。エイホはアルゴリズムの有名な本を書いているコンピュータ科学者であり、ワインバーガーは数学者、そしてカーニハンは C 言語の作者の一人としても知られている。AWK の言語仕様が C 言語に似ているのはそのせいであろう。

初期の AWK (1970 年代) はあまり多くの機能は持っておらず、ソースもシンプルなものであったが、AWK の広がりとともに拡張機能への要望が強まり、大きく改良された AWK (nawk) が誕生し (1980 年代)、そしてさらにその仕様を元に GNU 版の AWK である gawk が作られた (1980 年代)。本講義で使用しているのはその GNU 版の gawk (の日本語対応版の MS-Windows 版) であり、現在の gawk の最新版は version 4.0.0 である (01/12 2012 現在)。

gawk は、ソースコードが長くなったり軽快さが失われたりしないように無駄な機能の追加が極力抑えられているが、version 3.1 からはネットワーク接続

機能が追加され、TCP/IP に基づくネットワーク越しのファイルの取得や出力が可能になっている。例えば URL を指定してその HTML ファイルを取得してそれを処理する、ということが gawk 単独で行えるようになっている。

さらに gawk に便利な機能を追加するプロジェクトもあり、XML ファイルの処理機能や高精度計算、データベースへのアクセス機能、gd という画像ライブラリによる画像描画機能などが追加された xgawk が有名である。ただ、gawk には system() 関数やパイプ機能もあり、すでに他のプログラムと連携を取ることは可能なので、AWK にできないことは別のプログラムにやってもらい、AWK では AWK に得意な処理をやる、という形が自然だろう。Perl や Python, Ruby のように外部ライブラリとして機能を自由に追加することもあまり得意ではないので、AWK は主としてはこのシンプルな形のまま使われ続けていくだろうと思う。

AWK の思想は、作者の書いた AWK の本「プログラミング言語 AWK」の序章の冒頭に明快に記されているので、最後にそれを引用する:

> 「コンピュータの利用者は、多くの時間をデータの書式を変更したり、その整合性を検査したり、ある性質を持った項目を探したり、数を足し込んだり、あるいはレポートを出力したりするような、単純で機械的なデータ操作に費している。これらの仕事の多くは機械化されるべきであるが、かといってそれらを処理する特別なプログラムを毎回 C や Pascal のような一般的な言語で書くのはばかげている。awk はこういった仕事を、たいていは 1 行か 2 行で済むようなとても短いプログラムで始末できるように設計されたプログラミング言語である。」

(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」(1989)、アジソン ウェスレイ・トッパン)