2006 年 4 月 5 日
(2007 年 12 月 14 日改訂)

AWK に関する基礎知識

新潟工科大学 情報電子工学科 竹野茂治

1 AWK とは

AWK[1] 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ます。

- プログラムの記述が C 言語に非常に似ている[2]
- Unix には普通標準で入っていて、MS–Windows などで動作するものもある (もちろんフリーで)
- 行単位のフィルタとして動作するし、C ライクなインタプリタとしても使える
- 正規表現や連想配列など、C にはない機能もある
- 関数定義もできるので、ある意味でのライブラリも作れる

ただし、GUI プログラムを作ったりすることはできませんので、MS–Windows で実行する場合は、全てコマンドプロンプト (または DOS プロンプト) で実行することになります。

2 AWK の実行環境

1 節に書いたように、Unix では AWK は普通は標準でインストールされていて[3]、すぐに利用できます。

現在は、昔の Unix についていた AWK よりも GNU 版の AWK(gawk と呼ばれることもあります) の方がポピュラーですが、Unix によっては既にインストールされている AWK が実は gawk である場合もあります。

gawk については、以下をご覧ください。

[1] 作者である V. Aho, P.J. Weinberger, B.W. Kernighan の頭文字をとっていて「おーく」と読まれることが多いようです。
[2] C の作者でもある B.W. Kernighan が作者の一人ですから当然ですね。
[3] /usr/bin/awk などとしてインストールされていると思います。

- Gawk `http://www.gnu.org/software/gawk/gawk.html`

gawk には MS–Windows など、他の OS で動作するものもあります。以下に、MS–Windows 上で動作する gawk をいくつか紹介します。

- gawk 3.0.6 日本語対応版
  `http://www.hinadori.atnifty.com/~wills/program.html`
- gawk 3.1.5 日本語対応版
  `http://www.kt.rim.or.jp/~kbk/`
- UnxUtils (GNU utilities for Win32) に含まれる gawk
  `http://unxutils.sourceforge.net/`
- MSYS (Minimal SYStem) に含まれる gawk
  `http://www.mingw.org/msys.shtml`
  (cf. `http://www.interq.or.jp/japan/s-imai/tcltk/msys_mingw.html`)
- Cygwin (MS–Windows) に含まれる gawk
  `http://sources.redhat.com/cygwin/`

後半 3 つは、AWK だけではなくその他のいくつかの Unix 標準のツール群からなるパッケージですので、Unix と MS–Windows を使う人はそのどれかを持っていると便利だと思います。

以降は、gawk、または伝統的な awk を拡張した nawk[4]を想定して話を進めていきます。

3 AWK スクリプトの構造

AWK に対するプログラムファイルを通常はスクリプトと呼びます。1 行位のスクリプトであれば、ファイルにせずとも直接 AWK の実行時にそれをコマンドライン上で与えることもできます。

スクリプトは、基本的に「関数定義部分」と「実行部分」に分かれます。個々の関数定義は

```
function 関数名 (引数 ...){ (関数本体) }
```

のように書きますし、実行部分は

[4] /usr/bin/nawk などとしてインストールされているかもしれません。

```
（パターン）{（実行部分本体）}
```

のように書きます。関数定義、実行部分は、スクリプト内部に複数書くことができますし、実行部分本体、関数本体の部分 (中かっこの中味) は複数行の命令を書くこともできます。

AWK を行単位のフィルタとして使う場合、例えば

```
% awk -f script.awk < file.in > file.out
```

のようにして、スクリプト (この例では script.awk) 以外に、処理させる対象である入力 (この例では file.in、file.out がその結果の出力) を与えるのですが、その入力に対して実行は以下のように行われます。

1. 「パターン」が “BEGIN” である実行部分がまず実行される
2. 入力を 1 行読みこみ、それに対して各実行部分を順に適用するが、「パターン」(= 条件) に合う実行部分のみが実行される
3. それを入力の最後の行まで繰り返す
4. 「パターン」が “END” である実行部分が最後に実行される

「パターン」が “BEGIN” である実行部分のみのスクリプト、すなわち、

```
BEGIN{（実行部分本体）}
```

か、または

```
BEGIN{（実行部分本体）}

function 関数名（引数 ...){（関数本体）}

function 関数名（引数 ...){（関数本体）}
...
```

のような形である場合、それを

```
% awk -f file.awk
```

のようにすれば、入力なしで BEGIN 内の部分のみが実行され、すなわち簡単なインタプリタとして実行することになります。

MS–Windows 環境では、コマンドプロンプトを立ち上げて、その上で

```
> gawk -f file.awk
```

のようにします。これを実行する場合、gawk.exe があるディレクトリ (フォルダ) 内にスクリプトファイルも置いてそこで実行しまうのが簡単ですが、gawk にパスを通せば任意の場所で実行できるので、AWK スクリプト専用のフォルダを作って整理することもできます。

4 AWK の文法の概略

AWK は C 言語と似た文法で書けますが、違うところもあります。

- C 言語同様、実行単位は文で、複数の文は でグループ化できる
- C 言語同様、文の最後に ; をつけるが、C とは違い行末ならば ; は不要
- 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に \ (MS–Windows の場合は円マーク) を書く必要がある
- if 文、while 文、do while 文、for 文は C 言語同様に使える
- 変数宣言は不要
- 変数の値は数値か文字列で、整数型はなく、数値型は全て実数型、文字型はなく、文字列自体が変数の値となる
- 数値を表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
- 配列の添字は数値 (負の値も実数も可)、文字列等任意の値が使える (連想配列)
- コメントは # から行末まで

また、C と同様に printf も使えますが、() は通常省略できます。

以下にいくつか簡単な例を上げますが、C 言語の入門書に書かれていそうな例題を AWK で解いてみることにします。

例 1

「Hello World」を出力

```
BEGIN{ printf "Hello world.\n" }
```

AWK には、単に文字列を 1 行改行をつけて出力する print という関数もあるので、それを使えば、

```
BEGIN{ print "Hello world." }
```

とすることもできます。なお、print は %d などの書式指定は受けつけません。

C だと、例えば以下のように書きます。

```
#include <stdio.h>
int main(void)
{
    printf("Hello world.\n");
    return 0;
}
```

比較すればわかりますが、AWK の方は C で (多分意味もわからずに) 書いている #include や main() return 等が不要です。

例 2

1 から 100 までの和、1^2 から 100^2 までの和を出力

```
BEGIN{
  for(j=1;j<=100;j++){ x+=j; y+=j*j }
  printf "和=%d, 自乗和=%d\n",x,y
}
```

x,y の初期値が設定してありませんが、その場合は 0 が初期値になります。最後の出力部分は、単に数字だけ欲しいなら

```
print x,y
```

としてしまうことも可能です。

C だと例えば以下のようになります。

```
#include <stdio.h>
int main(void)
{
    int j,x=0,y=0;

    for(j=1;j<=100;j++){ x+=j; y+=j*j; }
    printf("和=%d, 自乗和=%d\n",x,y);
    return 0;
}
```

このように、

- C とほぼ同じように書ける
- ヘッダーファイルや変数宣言などのおまじない部分が不要
- コンパイル作業はいらず、そのまま実行できる

という点で AWK が優位であることがわかると思います。

5 フィールド要素等

AWK を行単位のフィルタとして使う場合、実行部分では以下のような定義済み変数が使用できます。

- `$0`: その行全体からなる文字列を意味する
- `$1`,`$2`,...: その行の第 1 フィールド、第 2 フィールド,... を意味する
- `NF`: その行のフィールド数を意味する
- `NR`: その行が何行目であるかを意味する

「フィールド」というのは、区切り文字 (デフォルトではスペースやタブ) で分けた場合のその行の各構成要素を意味し、例えば入力行が

```
1 23 456 "ab cde"
```

だった場合、自動的に、

```
NF=5, $1="1", $2="23", $3="456", $4+="\"ab" ("ab の 3 文字の文
字列), $5="cde\"" (cde" の 4 文字の文字列)
```

となります。$ の後ろには定数だけでなく、変数や式を書くこともできて、

```
j=3; k=$j; m=$(NF-1);
```

のような使い方をすることもできます。

6 AWK の関数

AWK では、自分で関数を定義することもできますが、デフォルトで使える関数も色々あります。テキスト処理フィルタという性格から文字列処理関数が主に用意されていますが、数学関数なども用意されています。

数学関数には以下のようなものがあります。

- `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 ($-\pi < \theta \leq \pi$)
- `exp(x)` $= e^x$
- `log(x)` $= \log_e x$ (自然対数の値)
- `sqrt(x)` $= \sqrt{x}$
- `int(x)` $= x$ の整数部分
- `rand()` $= 0$ 以上 1 未満の一様な乱数値 (呼び出す度に値が違う)
- `srand(x)` $=$ rand() の初期値を決めるパラメータ (種) を設定

int(x) は、C での floor(x) (x 以下の最大の整数) や ceil(x) (x 以上の最小の整数) とは違い、`int(1.5)`$=1$, `int(`-1.5`)`$=-1$ のようになります。

rand() は小数の値をランダムに返しますが、srand() を使わなければ、AWK の実行の度に同じ列を繰り返し生成することになりますので、srand(x) を使って初期値を変動させます。

しかし、AWK の実行の度に x (整数) が同じならば、やはり同じ乱数列が生成されてしまいますので、C ではプログラム実行時の時刻を使ってそれが常に変化するようにします。

AWK でも、x を省略して単に srand() とすると、現在の時刻 (正確には 1970 年 1 月 1 日 00:00:00 からの秒数) を種にセットします。

文字列処理関数には以下のようなものがあります。

- `index(s,c)` = 文字列 s の中に含まれる文字列 c の先頭位置 (s の中に c が含まれていなければ 0 を返す)
- `length(s)` = 文字列 s の長さ (C での strlen())
- `substr(s,n,len)` = 文字列 s の n 番目の文字から len 文字分の文字列 (len を省略した場合は最後まで)
- `sub(r,s,c)` = 文字列 c の正規表現 r に最初にマッチした部分を文字列 s で置きかえる (c を省略した場合、現在の入力行が c になる)
- `gsub(r,s,c)` = 文字列 c の正規表現 r にマッチした部分を全部文字列 s で置きかえる (c を省略した場合、現在の入力行が c になる)
- `match(s,r)` = 文字列 s で正規表現 r にマッチした部分文字列の先頭位置を返す (予約済み変数 RSTART に先頭位置を、RLENGTH に部分文字列の長さをセットする)
- `printf(f,...)` = C 言語の printf(f,...) とほぼ同じ (() は通常省略できる)
- `print(s)` = `printf("%s\n",s)` とほぼ同じ (() は通常省略できる)
- `sprintf(f,...)` = printf(f,...) で出力されるべき文字列を返す (C 言語の sprintf() に似たもの)
- `split(s,a,r)` = 文字列 s を正規表現 r を区切りとして分割し、配列 a に、a[1],a[2],... と順に収める (r を省略した場合はホワイトスペース区切りとし、分割個数を返す)
- `tolower(s)` = 文字列 s に含まれるアルファベットの大文字は小文字に変換したものを返す
- `toupper(s)` = 文字列 s に含まれるアルファベットの小文字は大文字に変換したものを返す

AWK では文字列の連結も容易に行なえ、並べて書けば連結、','を入れて書けば空白一つを入れて連結、となります。よって、

```
a = "1" "234","5 6","7" " 89";
```

とすると a は "1234 5 6 7 89" となります。

その他に、以下のような関数もあります。

- delete a[j] = 配列の要素 a[j] を削除
- systime() = 1970 年 1 月 1 日 00:00:00 からの秒数
- strftime(f,n) = 1970 年 1 月 1 日 00:00:00 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
- system(s) = 文字列 s をコマンドラインとして OS 上で実行し、その終了コードを返す

参考文献

[1] 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」、新紀元社 (2004) (元版は 1989)

[2] D.Dougherty、A.Robbins (福崎俊博訳)、「sed & awk プログラミング 改訂版」、オライリー・ジャパン (1997)

[3] 志村拓 鷲北賢 西村克信、「AWK を 256 倍使うための本」、アスキー出版 (1993)