2016 年 06 月 03 日

計算機実習 III (2016 年度)
第 7 回: AWK その 1

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

目 次

1 AWK の概要 1
2 簡易インタプリタとしての AWK の実行方法 2
3 AWK の基本文法 3
4 printf と print 6
コラム: AWK の作者と歴史 8

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型プログラミング言語で、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 では標準装備、MS–Windows で動作するフリーの AWK もある。
- テキストデータに対する行単位のフィルタとして動作するが、C ライクな簡易インタプリタとしても使える (特に GNU 版の AWK は)。
- 正規表現や連想配列、文字列のフィールド分割、文字列処理関数など、C 言語にはない機能もある。
- 関数を自作することもでき、サブルーチンのライブラリ化もある意味で可能。

本実習では、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 (計算機実習室にはインストール済み)。なお、必要なのはそこに含まれる `gawk.exe` という小さい実行ファイル一つだけなので、USB メモリで簡単に持ち運ぶことができる。

- 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、「test1.awk」のような「.awk」という拡張子を持つ名前のファイルに保存する。otbedit で AWK スクリプトを編集する場合は、「.awk」の拡張子のファイルを読み込むか、または画面左上にある AWK というアイコンで AWK モードにする。

2 簡易インタプリタとしての AWK の実行方法

AWK スクリプトの行フィルタとしての使い方は第 9 回以降で紹介することとし、しばらくは簡易インタプリタとしての以下のような構造のスクリプトを使用する。

```
BEGIN {
    [プログラム本体]
}
```

gawk にこの形式のスクリプトを処理させると、[プログラム本体] 部分を実行し、それが終了すると gawk も終了する。

このスクリプトを実行させるには、そのスクリプトの名前が例えば test1.awk であれば、コマンドプロンプトで

```
Z:¥> gawk -f test1.awk
```

とする。スクリプトがカレントディレクトリにない場合は、

```
Z:¥> gawk -f script¥test1.awk
```

のように置き場所のパス (相対パスでも絶対パスでもよい) を指定して実行するか、先にカレントディレクトリをそのディレクトリに移動してから実行する:

```
Z:¥> cd script
Z:¥script> gawk -f test1.awk
```

gawk の -f オプションは、その後ろに gawk に処理させる AWK スクリプトを指定する。本実習では gawk のオプションは -f, -F, -v 以外にはほとんど使用しないが、gawk オプションの一覧は、

```
Z:¥> gawk --help
```

で表示できる (が、オプションの説明はない)。また、gawk のバージョンや copyright は

```
Z:¥> gawk --version
```

で表示できる。なお、gawk は MS–Windows の標準コマンドではないため、help gawk ではヘルプは表示できない。

3 AWK の基本文法

インタプリタ形式の AWK スクリプトの簡単な例:

```
BEGIN {
    x = 3; y += 2
    printf "%d¥n", x + y
    for (j=1; j<=5; j+=2) z += j
    printf "%3d¥n", z
    if (x+y < z) s = "x + y < z"
    else s = "x + y >= z"
    printf "%s¥n", s
}
```

このスクリプトからわかるように、AWK スクリプトの文法は C 言語と非常に近いが違うところもある。基本的な文法について説明する。

1. 変数
   - 変数宣言は不要。
   - 変数値は数値か文字列で、数値はすべて実数 (整数型はない)。また、’ ’ で囲む文字型もなく、1 文字でも文字列として " " で囲む。
   - 数値を表わす文字列は、文字列としても、数値としても使える。例えば「s = "12345"; x = s + 5」とすると x には数値の 12350 が代入される。
   - 明示的に初期化されていない変数は、数値としては 0 が、文字列としては空文字列が初期値となる。
   - 配列は、C 言語と同様の形式で使えるが、添字は 0 から始める必要はない。配列の値も実数でも文字列でもよい。
2. 書式
   - # から行末までは無視されるのでコメントとして使える。逆に、/* */ や // はコメントにはならない。

- 1 行に複数の文を書く場合は、それらを ; で区切る必要がある。逆に C 言語とは違い、行末に ; をつける必要はない (つけても問題はない)。
- 複数の文のグループ化は C 言語同様 { } で囲む。
- C 言語同様、空白、タブ、改行はほぼ任意に書けるが、文の途中で改行する場合は、その改行の直前 (行末) に ¥ が必要になる場合がある。例えば、

```
BEGIN { for (j=1; j<=5; j++) printf "%d %d¥n", j, 2*j }
```

を、改行を入れて以下のように書いた場合、

```
BEGIN {
    for (j=1; j<=5; j++)
        printf ¥
        "%d %d¥n" ¥
        , j,
        2*j
}
```

3 行目、4 行目の行末の ¥ は、それを取ってしまうと次の行まで文がつながっているとみなされずにエラーになるので省略できない。逆に 2 行目、5 行目の行末には ¥ は必要ない (つけても問題はない)。2 行目の for 文は ( ) の後ろに実行文が続くことが期待されているし、5 行目は , で終わっているので 6 行目までこの文が続くと認識されるためである。

なお、行末に ¥ を置く場合は、¥ と行末の間に空白を入れてはいけない[1]。

3. 演算子、構文

- 四則演算 (+, -, *, /, %, ++, --)、比較演算 (<, >, <=, >=, ==, !=)、論理演算子 (&&, ||)、代入演算子、(+=, -=, *=, /=, %= 等)、3 項演算子 (A?B:C) などの演算子は、ほぼ C 言語と同じものが使える。
- 整数も実数とみなされるので、C 言語とは違い「5/3」は 1 でなく 1.6666... になるが、% (剰余計算) は例外で、「5%3」は 2 になる。
- 整数の定数には、先頭に 0x をつけた 16 進数整数表記、先頭に 0 をつけた 8 進整数表記も使える。
- 累乗 x^y は「x^y」と書けるが、$x < 0$ の場合は整数乗 (y が整数) しか許されない ($x \geq 0$ なら y は非整数も可)。
- if 文、while 文、do while 文、for 文も C 言語と同じものが使えるが、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、それ以外はすべて真となる。

[1] 普通空白や改行は見えないのでミスしがちだが、otbedit では表示されるので便利。

- printf 関数は、上の例のようにほぼ C 言語と同じ書式であるが、書式文字列や変数をまとめて ( ) で囲む必要はない。詳しくは 4 節で説明する。

簡単なスクリプトの例をいくつか紹介する。

```
BEGIN { for (j=1; j<=150; j++)
    printf "ren file%d.txt file%04d.txt¥n", j, j }
```

これは、「ren file1.txt file0001.txt」のように、ファイル名の数字部分に 0 を追加して 4 桁の番号に変換するためのコマンド列を 150 行表示してくれる [2]。

```
BEGIN { x0 = 1; x1 = 20
    printf " x、2 乗、平方根 (%d <= x <= %d)¥n", x0, x1
    for (x=x0; x<=x1; x++)
        printf "%2d、%4d、%5.2f¥n", x, x^2, x^(1/2)
}
```

このスクリプトは、平方と平方根の簡単な数表を出力する。

```
BEGIN { a[0] = "日"; a[1] = "月"; a[2] = "火"
    a[3] = "水"; a[4] = "木"; a[5] = "金"; a[6] = "土"
    for (j=1; j<=30; j++)
        printf "2016 年 6 月 %d 日 (%s)¥n", j, a[(j + 2)%7] }
```

これは、6 月の日付一覧を表示する。配列で曜日を定義しているが、その出力で使われている添字の「(j + 2)%7」は、7 での割り算の余りにより 0 から 6 までを繰り返す数列になり、2 を足すことでそれが 3 から始まるようにしている (1 日が水曜)。

```
BEGIN { N = 27; maxloop = 10000; printf "0: %d¥n", N
    for (j=1; j<=maxloop && N>1; j++) {
        N = (N%2 == 0) ? N/2: 3*N + 1
        printf "%d: %d¥n", j, N }
}
```

これは、ある数列を計算するもので、N の更新で 3 項演算子を使っているが、N が偶数なら N には $N/2$ が、N が奇数なら $3N+1$ が代入される。簡単な条件分岐や、条件で値を変えるものは、if 文を使わなくても 3 項演算子で書くことができる。

[2] 実際にこの出力を出力リダイレクトを用いてファイルに保存して、バッチファイルとして実行させることもできる: 逆にバッチファイルの for 文でこの作業を行うのは難しい。

4 printf と print

AWK での出力 (表示) は、書式付き出力命令 printf と、書式無し出力命令 print で行う。

命令	意味
printf "[書式]", [値], [値], ...	書式に従って値を表示
print [値], [値], ...	値を順に表示して改行

表 1: printf, print

注意:

- print には書式指定はなく、指定した値を (複数指定した場合は) 空白区切りで表示し、最後に改行する (printf は ¥n を指定しないと改行しない)。
- 両者とも命令の後ろの部分を ( ) で囲む必要はないが、「printf("Hello.¥n")」のようにかっこをつけても問題はない。
- 空行を出力するには「printf "¥n"」か「print ""」とする。

printf の書式文字列は、C 言語とほぼ共通であるが、少し復習する。

書式文字列には、通常の文字列の他に、%d や %f などの値の書式化指定、¥t (タブ), ¥n (改行) などの特殊文字、%% (% 自体)、¥¥ (¥ 自体) が使用できる。そして書式文字列の後に、値の書式化指定の個数分の値をカンマ (,) 区切りで書き並べる。

値の書式化指定の主なものを表 2 に示す。なお、(q) の部分は省略可能なオプション指定 (実際に ( ) は書かない)。

指定	意味	指定	意味
%c	整数を文字に変換	%(q)x	16 進整数 (符号なし)
%(q)s	文字列として表示	%(q)o	8 進整数 (符号なし)
%(q)d	10 進整数 (符号あり)	%(q)f	浮動小数表示
%(q)u	10 進整数 (符号なし)	%(q)e	実数の指数表示

表 2: 値の主な書式化指定

%x を %X にすると 16 進数の a,b,c,d,e,f が大文字の A,B,C,D,E,F になり、%e を %E にすると指数表示の e が大文字の E になる。また、%c で文字列を表示させると先頭文字 1 文字だけが表示される。

表 2 の「q」の省略可能オプションは、

([フラグ])(w)(.p)

の形式で指定する (( ) は省略可能であることを意味)。w, p は 0 以上の整数で、w は結果を最低でも w 幅で出力することを意味し、p は指定により意味が異なり、%d, %u, %x, %o では必要なら頭に 0 をつけて最低でも p 桁の数字を表示、%f, %e では小数点以下の表示桁数 (足りなければ 0 をつけ、越えると四捨五入して詰める、デフォルトでは 6 桁)、%s では最大表示文字数 (越えると途中で切る) を意味する。

[フラグ] には、主に表 3 の 4 種類の文字、またはその組み合わせを使う。

フラグ文字	意味
-	幅指定内に左詰めで出力
+	符号付きで値が正の場合は + を先頭に一つ置く
空白	符号付きで値が正の場合は空白を先頭に一つ置く
0	値文字列が幅指定よりも短い場合左側を 0 で埋める

表 3: 書式化指定の主なフラグ

表 4 に、printf の後ろに書く書式と値、およびその出力結果を、出力幅がわかりやすいように [ ] で囲んだ形の例で示す。なぜそうなるのか一つ一つ確認せよ。

書式と値の例	結果	書式と値の例	結果
"[%f]", 3.14	[3.140000]	"[%c%c]", 0x41, 0x30	[A0]
"[% f]", 3.14	[3.140000]	"[%3s]", "abcde"	[abcde]
"[%9f]", 3.14	[3.140000]	"[%-7s]", "abcde"	[abcde]
"[%-9f]", 3.14	[3.140000]	"[%7.3s]", "abcde"	[abc]
"[%.4f]", 3.14	[3.1400]	"[%06x]", 314	[00013a]
"[%-+9.4f]", 3.14	[+3.1400]	"[%0.6d]", -314.15	[-000314]
"[%10.2e]", 31450	[3.15e+004]	"[%-+8.5d]", 314.99	[+00314]
"[%.1E]", 0.00315	[3.2E-003]	"[%- 8.5d]", -314.99	[-00314]

表 4: 書式化指定の例

やや裏技的であるが、例えば i と n の間に空白を 17 個入れて出力する場合、17 個の空白を実際に書く以外に上の w 幅指定を利用して、「printf "i%18s", "n"」や、「printf "%-18sn", "i"」、「printf "i%17sn", ""」などとする方法があり、これらの方が空白を 17 個書くよりも間違いが少ない。

また、w や p の部分に * を指定した場合は、それらを値部分で指定することを意味し、変数で w や p を制御できるようになる。例えば、「printf "%*d", 3, 5」は、「printf "%3d", 5」と同じ意味になる。

コラム: AWK の作者と歴史

AWK (おーく) の名前の由来は、その 3 人の作者 A.V.Aho (エイホ、コンピュータ科学者)、P.J.Weinberger (ワインバーガー、数学者)、B.W.Kernighan (カーニハン、コンピュータ科学者) の名前の頭文字から来ている。そしてカーニハンは C 言語の作者の一人でもあり、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を持っておらず、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)。本実習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 4.1.3 である (05/26 2016 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。xgawk のように gawk にさらに便利な機能を追加するプロジェクトや、Perl や Python, Ruby のような AWK よりも強力なスクリプト言語もあるが、gawk の軽快さ、持ち運びやすさ、仕様の簡便さは他の言語に十分勝っており、今後も多くの場面で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)