2012 年 06 月 08 日

計算機実習 IV (2012 年度)
第 8 回: AWK その 3

目 次

1 正規表現 1
2 match() と substr() 5
3 文字列処理関数 6
コラム: gawk の国際化 9

1 正規表現

正規表現 (regular expression) は、特別な意味を持つ制御文字を使うことで複数の文字列のパターンを表し得る文字列であり、主に文字列や行の検索に用いる。コマンドプロンプトのワイルドカードに似ているが、意味が違うところに注意が必要[1]。

表 1 の文字は、正規表現では特別な意味を持つ (**メタ文字** と呼ばれる) が、それ以外の文字は、文字通りの意味を表す。

以下に例を示す。

- `2.3` : “213”, “2a3”, “2X3” などを表す
- `12?3` : “13”, “123” のいずれか
- `12*3` : “13”, “123”, “1223”, “12223” など
- `12+3` : “123”, “1223”, “12223” など (上の “13” 以外)
- `12.*3` : “12” から始まり 3 で終わる任意の文字列

課題 8-1. 正規表現「`20..12...`」が表す文字列を説明せよ。

[1] AWK 以外にも正規表現をサポートするスクリプト言語は多い。実装はそれぞれで多少異なるが、大半は AWK の正規表現をほぼ含んでいる。

メタ文字	意味
.	任意の 1 文字を表す
?	直前の文字が 0 個または 1 個あること表す
*	直前の文字が 0 個以上続くことを表す
+	直前の文字が 1 個以上続くことを表す
¥	エスケープ文字
[ ]	[ ] 内の文字のいずれか 1 文字を表す
[^]	[^] 内の文字以外のいずれか 1 文字を表す
( )	( ) 内の正規表現をグループ化する
|	OR (主に ( ) 内で用いる)
^	文字列の先頭を表す (文字自体は表さない)
$	文字列の末尾を表す (文字自体は表さない)

表 1: メタ文字一覧

課題 8-2. 正規表現「AB+C?D」が表す文字列を説明せよ。

課題 8-3. 正規表現「a?b*c.d」が表す文字列を説明せよ。

¥ (エスケープ文字) は、主に以下の形式で使用する[2]。

1. メタ文字の前について特別な意味を消す (例: ¥* で “*” を表す)
2. n, t などの前について、改行やタブ文字などを表す

[ ], [^] 内には半角文字列を並べて示すが、- によって範囲指定することも可能である。以下に例を示す。

- [axf]d : “ad”, “xd”, “fd” のいずれかを表す
- [01][A-C] : “0A”, “0B”, “0C”, “1A”, “1B”, “1C” のいずれか
- [0-9A-Za-z] : アルファベットか数字のいずれか 1 文字
- [^sf]printf : “sprintf”, “fprintf” 以外の “printf” 文字列

課題 8-4. 正規表現「20[01][0-9]」が表す文字列を説明せよ。

課題 8-5. 正規表現「-?[1-9][0-9]」が表す文字列を説明せよ。

[2]文字の 8 進コード、16 進コードを表現するのに使うこともあるが、説明は省略する。

課題 8-6. 正規表現「`0x[0-9a-f][0-9a-f]`」が表す文字列を説明せよ。

[ ], [^] は 1 文字を意味するので、この後ろに *, +, ? をつけることができるが、( ) を用いると、ある正規表現のまとまりを 1 文字と考えて *, +, ? をつけることができるようになる。以下に例を示す。

- `(01)+` : “01”, “0101”, “010101” などを表す
- `[01]+` : 0 と 1 だけからなる 1 文字以上の文字列全体

[ ] ではいくつかの文字から選んだ 1 文字を表せるが、1 文字とは限らない文字列の中から選択したい場合は | を使用する。これは通常 ( ) 内で使用する。

- `(ABC|12)` : “ABC”, “12” のいずれかを表す
- `.*¥.(ttf|TTF)` : “.ttf” か “.TTF” で終わる文字列

課題 8-7. 正規表現「`[-+]?[1-9][0-9]+`」が表す文字列を説明せよ。

課題 8-8. 正規表現「`[A-Z][a-z]+ [A-Z]+`」が表す文字列を説明せよ。

課題 8-9. 正規表現「`[Aa](bcd|BCD)e[Ff]?`」が表す文字列を説明せよ。

課題 8-10. 正規表現「新潟(工科|産業|経営)?大学」が表す文字列を説明せよ。

正規表現は、AWK では / / で囲んで使用するが、これは、以下のマッチ演算子を使って、条件文として使用できる。

- `s ~ /`正規表現`/` : 文字列 s が正規表現にマッチすれば (それが表す文字列を含んでいれば) 真、そうでなければ偽。
- `s !~ /`正規表現`/` : 上と逆の意味。

なお、/ / 内で / 文字を使う場合は、¥/ としなければならない。

この場合、表 1 の ^ や $ を使うことで、「含まれる」ではなく、正規表現で「始まる」文字列や、正規表現で「終わる」文字列などを指定できる。以下に例を示す。

- `s ~ /[0-9]/` : 文字列 s が数字を含む場合に真

- `s !~ /[0-9]/` : 文字列 s が数字を含まない場合に真
- `s ~ /^[0-9]/` : 文字列 s が数字から始まる場合に真
- `s ~ /[0-9]$/` : 文字列 s が数字で終わる場合に真
- `s ~ /^[0-9]+$/` : 文字列 s が数字しか含まない場合に真

条件文の左辺とマッチ演算子を省略した場合、例えば

```
if (/[0-9]+/)
```

と書いた場合は、`$0`、すなわち入力行全体が対象となり、

```
if ($0 ~ /[0-9]+/)
```

と同じことになる。この否定

```
if ($0 !~ /[0-9]+/)
```

も、`$0` を省略して

```
if (!/[0-9]+/)
```

と書くことができる。

いくつか 1 行スクリプトの例を示す。

- `!/^$/` : 空行以外を表示
- `/^20[01][0-9]120[0-9][0-9]/` : 学籍番号で始まる行のみを表示
- `/(情報電子工|機械制御システム工|環境科|建築) 学科/`
  学科名が含まれる行のみを表示

課題 8-11. `dir C:¥Windows¥Fonts` の出力から `.ttf` か `.TTF` で終わる行 (TrueType フォント) のみを表示する 1 行スクリプトを書け。

課題 8-12. `dir C:¥Windows¥Fonts` の出力から日付で始まる行 (ファイルを表す行) のみを表示する 1 行スクリプトを書け。

2 match() と substr()

正規表現にマッチした部分文字列を取り出すための関数として match() と substr() が用意されている。

- match(str, reg) : 文字列 str が正規表現 reg にマッチした場合はマッチした場所を表す正の値を、マッチしなかった場合は 0 を返す。

これだけだと、str ~ reg とほぼ同じであるが、match() はさらに、どの部分文字列にマッチしたかを示す値をシステム変数の RSTART と RLENGTH にセットする。

- RSTART : マッチした部分文字列の開始位置 (= match() の返り値)
- RLENGTH : マッチした部分文字列の長さ

例えば、

```
s = "12ab34cd"
match(s, /[a-z]+/) # 小文字アルファベットの連続にマッチ
```

とすると RSTART は 3 に、RLENGTH は 2 にセットされる。

なお、正規表現にマッチする部分文字列が複数ありうる場合は、最初に現れて、最も長くなるものが対象となる。例えば、

```
s = "2ab2ab2ab"
match(s, /a.*2/) # a で始まり 2 で終わる文字列にマッチ
```

とすると "ab2ab2" という文字列にマッチするので、RSTART は 2 に、RLENGTH は 6 になる。

課題 8-13. s="http://takeno.iee.niit.ac.jp/~shige/" に対して、match(s,/¥/¥/.*¥//) とした場合の RSTART, RLENGTH は何になるか確認せよ。

この RSTART, RLENGTH の値は substr() と組み合わせることで威力を発揮する。

- substr(str, ind [, len])
  文字列 str の ind 番目から長さ len の文字列を返す (文字列値関数)。ただし、len を省略して substr(str, ind) とした場合は、str の最後までの文字列を返す。

例えば、

```
s1 = "abcdefghi"
s2 = substr(s1, 3, 4)
s3 = substr(s1, 7)
```

とすると、s2 は "cdef"、s3 は "ghi" となる。

課題 8-14. s="201512001,170cm,65kg" に対して、substr(s, 9+2, 5), substr(s, 9+1+5+2) がそれぞれ返す文字列は何になるか確認せよ。

課題 8-15. 各行の 1 列目が 9 桁の学籍番号であるデータに対し、その下 4 桁を出力する AWK スクリプト kadai8-15.awk を作成せよ。

match() と substr() を組み合わせると、正規表現にマッチした部分を簡単に取り出せる。例えば、

```
if (match($0, /[0-9]+/)) {
    s1 = substr($0, 1, RSTART-1);
    s2 = substr($0, RSTART, RLENGTH);
    s3 = substr($0, RSTART+RLENGTH);
}
```

とすると、$0 (= 入力 1 行全体) の [0-9]+ にマッチした部分文字列が s2 に保存され、s2 よりも前の部分文字列が s1 に、s2 よりも後ろの部分文字列が s3 に保存される。

課題 8-16. 入力データの各行内に最初に現れるアルファベット文字列を取り出す AWK スクリプト kadai8-16.awk を作成せよ。

3 文字列処理関数

AWK には、match(), substr() 以外にも以下のような文字列処理関数がある。

- length(s) : 文字列 s の長さを返す。
- index(s1, s2) : 文字列 s1 の中に部分文字列 s2 が最初に現れる場所を返す (なければ 0 を返す)

- `sprintf(format, ...)` : `printf(format, ...)` が画面に表示する文字列を値として返す (文字列値関数)
- `tolower(s)` : 文字列 s の中のアルファベットの大文字をすべて小文字に変えた文字列を返す (文字列値関数)
- `toupper(s)` : 文字列 s の中のアルファベットの小文字をすべて大文字に変えた文字列を返す (文字列値関数)
- `sub(r, s1 [, s2])`
  文字列 s2 (省略した場合は $0 が対象) 内の正規表現 r に最初にマッチする部分を s1 に置換し (直接 s2 を書き換える)、置換した部分文字列の個数を返す
- `gsub(r, s1 [, s2])`
  `sub(r, s1 [, s2])` と同様の置換を行うが、最初の一つだけでなく r にマッチする部分文字列すべてを s1 に置換する (`sub()` は最初の一つだけ)
- `split(s, h [, r])`
  文字列 s を、区切り正規表現 r (省略した場合はフィールドの区切りと同じ) で切り分けて、各項目を配列 h[1] から h[N] までに保存し、切り分けた単語数 N を返す

これらのいくつか、特に正規表現に関するものは C 言語には標準的には実装されてはおらず、このような便利な文字列処理が手軽に行えるところが AWK の大きな魅力の一つであろう[3]。

課題 8-17. 列が 2 つ以上あり、かつ 1 列目の文字列の長さが 3 文字以上の行を出力する 1 行スクリプトを書け。

例えば、

```
Z:> gawk "{ gsub(/,/, ¥"¥"); print }" test1.dat
```

は、データ test1.dat に含まれるカンマ (`,`) をすべて削除して出力する。`sub()`, `gsub()` の置換文字列に空文字列 (`""`) を指定すると空文字列への置換が行われ、結果として部分文字列の削除となる。

[3] GNU AWK にはさらに `gensub()`, `asort()`, `strtonum()`, ... 等の関数も追加されているし、`match()` の仕様も拡張されている。

なお、sub(), gsub() は元の文字列を直接書き換えてしまうので、元の文字列を保存して別に使いたい場合は、あらかじめ文字列を別の変数に保存しておく必要がある。

課題 8-18. 入力データの「、」をすべて「, 」(カンマと空白) に変換し、「。」をすべて「. 」(ピリオドと空白) に変換する 1 行スクリプトを書け。

課題 8-19. 入力データ内の「か」「き」「く」「け」「こ」をそれぞれ「ka」「ki」「ku」「ke」「ko」に変換し、「きゃ」「きゅ」「きょ」をそれぞれ「kya」「kyu」「kyo」に変換する AWK スクリプト kadai8-19.awk を作成せよ。

課題 8-20. 2 つ以上続く空白をすべてひとつの空白に変換する 1 行スクリプトを書け。

以下は実際には 1 行で記述する。

```
Z:> dir C:¥ | gawk "/<DIR>/{ split($0, h, /[ /:]+/);
    print h[4],h[5],h[2],h[3] }"
```

これは、dir C:¥ の出力のうち、文字列 <DIR> が含まれるもの (すなわちディレクトリに対する出力) に対し、その出力を空白かスラッシュ (/) かコロン (:) のいずれかを区切りに切り分け、それを配列 h に保存し、その 2 番目から 5 番目までの要素を並べかえて表示している。dir のディレクトリに対する出力は、

```
2011/12/26  12:48   <DIR>       Documents and Settings
```

のような形式になっているが、これを split() で「[/:]+」を区切りとして切り分けると、

```
h[1] = "2011", h[2] = "12", h[3] = "26",
h[4] = "12",  h[5] = "48",  h[6] = "<DIR>",
h[7] = "Documents",  h[8] = "and",  h[9] = "Settings"
```

のようになる。

split() は、配列にデータを代入するのにも利用することが多い。例えば、

```
week[0]="日"; week[1]="月"; week[2]="火"; ...
week[6]="土";
```

とする代わりに

```
split("月 火 水 木 金 土", week); week[0]="日";
```

とすれば一つ一つ代入しなくて済む。

課題 8-21. `split()` を使って、配列 `mon[1]` から `mon[12]` に`mon[1]="Jan"` のように英語の月名の最初の 3 文字を代入する AWK スクリプト kadai8-21.awk を作成せよ。

課題 8-22. `split()` を使って、`C:¥dir1¥dir2¥...` のような絶対パスを表す文字列を、ドライブ名 (`C`)、各ディレクトリ (`dir1`, `dir2`, ...) に分けて配列 `h[ ]` に配分する AWK スクリプト kadai8-22.awk を作成せよ。

コラム: gawk の国際化

最近の GNU AWK (3.1.8 等) には、ネットワーク機能と国際化対応などが追加されている。ネットワーク機能は、ネットワークにつながっている他のサーバと TCP/IP で接続して、その入出力データをファイルとの入出力と同様にできるようにするものであり、例えば URL を指定して Web ページを取得しつつ処理したり、メールサーバに直接接続してメールを出したり取得したりすることも可能になる。

一方、国際化対応であるが、gawk のような外国生まれのフリーソフトの多くは日本語のような 2 バイト文字には対応していないことが多い。通常は、日本語に対応していないアプリケーションでは、以下のような問題がある。

- 日本語がメニューに出ない (GUI アプリケーションの場合)
- ヘルプやエラーメッセージなどが日本語で見られない
- データとして日本語が扱えない

そのようなソフトで日本語が使えるようにすることを「日本語対応」とか「日本語化」と呼ぶ。

さらにより多くの言語に対応することを目指すことを「国際化」と呼び、「日本語化」のように特定の言語用にソフトを改良する方向を「地域化」と呼んで区別することもある。現在は、文字コードを国際的にある程度統一して扱える Unicode (UTF-8) という規格ができたこともあり、外国生まれのフリーソフトも「国際化」が進んできている。

gawk の国際化も 3.1.5 位から標準で入っているが、gawk のような CUI ツールの国際化は、主に扱うデータやエラーメッセージへの対応であり、gawk では 多バイト文字を英数字などと同じように「1 文字」とみなす方向で国際化が行われている。

例えば、ASCII データとしては ”abc” と ”日本語” という文字列は、前者は 3 バイト、後者は 2 バイト文字コードでは 6 バイト (UTF-8 では 9 バイト) であるが、昔の国際化されていない gawk では、

```
length("abc") = 3, length("日本語") = 6,
substr("日本語", 3, 2) = "本"
```

であったのに対し、バージョン 3.1.5 等の最近の gawk では、

```
length("abc") = length("日本語") = 3,
substr("日本語", 2, 1) = "本"
```

となる。

ただしこの日本語化対応はバージョンによって多少違いがあり、

```
printf "%6s¥n", "日本"
```

というスクリプトの出力は、バージョン 3.1.5 では、「スペース 2 つ + 日本」が表示される (すなわち「日本」を 4 文字と見ている) が、バージョン 3.1.8 以降では、「スペース 4 つ + 日本」と表示される (すなわち「日本」を 2 文字と見ている)。

全角文字を 1 文字として扱う仕組みは、文字データを日本語か非日本語かを区別せずに扱え、特に 1 バイト文字と 2 バイト文字が混在している文字列の処理の場合に便利であるが、等幅フォントで全角文字を英数半角文字の 2 倍の幅に揃えたい場合などには不都合であったりする。

そのような事情を、欧米人が多い開発側に理解してもらうことは難しく、日本人が欲する方向には改良されにくいことも多いようで、よって、開発メンバーに日本人が参加していないプロジェクトでは、日本人にとってはやや不都合な国際化が起こることもままあり、日本独自の日本語化 (地域化) が必要な場面もまだまだありそうである。