I117（A1）スクリプト言語 AWK

知念

北陸先端科学技術大学院大学 情報科学研究科
School of Information Science,
Japan Advanced Institute of Science and Technology

スクリプト

明解な定義はないが...
コンパイルが不要なプログラミング言語でかかれたプログラムを指す
スクリプト向けの言語を「スクリプト言語」と呼ぶ
歴史的には以下のようなものがある

```
shell, sed, awk, perl, REXX, tcl, ruby
```

他にも多数存在して、今後も増え続ける

awk

テキスト処理のツールで、スクリプトで処理を記述
名前は作者 Aho, Weinberger, Kernighan の頭文字

- sed, grep 等 UNIX の伝統的なツール群や C 言語との共通部分が多い
- フィールド指向
  $1 で第一フィールドを示す、$0 は入力行全体
- パターンとアクションの組みで処理を記述

```
a>2008{gsub(/the/,"The"); print $0}
/2008/{print $1, $3}
```

awk *(cont.)*

- パターンは式か正規表現
- 特別なパターンとして BEGIN と END が用意されている

- 多くの処理系で利用可能
  - しかもインストール済であることが多い

awk *(cont.)*

テキストファイルの処理は一般的に以下のような流れ

```
前処理
while( 1行取り込み ) {
    if(パターン1) 処理1
    if(パターン2) 処理2
    ...
    if(パターンN) 処理N
}
後処理
```

awk *(cont.)*

awk ではこの動きの while ループを展開して記述する

```
パターン1{処理1}
パターン2{処理2}
...
パターンN{処理N}
BEGIN{処理}
END{処理}
```

BEGIN や END は前／後処理を指す特別なパターン
パターン1 から パターンN は記述順に評価される
パターンなしで処理だけ書くといつでも評価される

実行例

文字 c を含む行を出力

- shell の各種評価を回避するため、記述はクォートで囲む

```
% ls | awk '/c/{print}'
03c
2dict.c
OSL_PIPE_16647_SingleOfficeIPC_d946fb5d5a5
SunWS_cache
...(略)
```

引数なしの print は入力行全体($0) を出力

実行例 *(cont.)*

C言語のファイル名(.c で終る行)を出力

```
% ls | awk '/\.c$/{print}'
2dict.c
a.c
libcsv.c
libidict.c
libsdict.c
report2-1.c
report2-2.c
stailb.c
```

実行例 *(cont.)*

C言語のファイル名(.c で終る行)を .o に変換して出力

```
%ls|awk '/\.c$/{sub(/c$/,"o");print}'
2dict.o
a.o
libcsv.o
libidict.o
libsdict.o
report2-1.o
report2-2.o
stailb.o
```

実行例 *(cont.)*

(末尾以外も)文字c を o に変換 (pr で3段にした)

```
% ls | awk '/\.c$/{sub(/c/,"o");print}'
2diot.c      libidiot.c  report2-2.o
a.o          libsdiot.c  stailb.o
libosv.c     report2-1.o
% ls | awk '/\.c$/{gsub(/c/,"o");print}'
2diot.o      libidiot.o  report2-2.o
a.o          libsdiot.o  stailb.o
libosv.o     report2-1.o
```

sub は最初だけ、gsub は全ての一致箇所を変換

実行例 *(cont.)*

```
% cat data-a
john    32      45,12
jack    13      71,43
joel    81      44,52
% awk '{print $2}' data-a
32
13
81
% awk '{s+=$2}END{print s}' data-a
126
```

実行例 *(cont.)*

フィールド区切り文字はスペース(0x20)とタブ(0x09)
オプション -F で変更可能

```
% awk -F, '{print $2}' data-a
12
43
52
% awk -F, '{s+=$2}END{print s}' data-a
107
```

実行例 *(cont.)*

フィールド区切り文字は変数 FS に格納されているので、直接書き換えても良い

```
% awk 'BEGIN{FS=","}{print $2}' data-a
12
43
52
```

実行例 *(cont.)*

処理をファイルに記述して実行することも可能

```
% cat a.awk
BEGIN{ FS="," }
END{ print s }
{ s+=$2 }
% awk -f a.awk data-a
107
```

パターン

```
% awk '/jack/{print}{print}' data-a
john    32          45,12
jack    13          71,43
jack    13          71,43
joel    81          44,52
```

jack が2回処理される— jackは2回合致

```
% awk '/jack/{next}{print}' data-a
john    32          45,12
joel    81          44,52
```

jack は出力されない— jack はスキップされる

パターン *(cont.)*

next は以下のようなイメージ

```
前処理なし
while( 1行取り込み ) {
     if( $0 にjackを含む ) {
          goto next-line;  }
     { print $0;}
next-line:
}
後処理なし
```

C 言語では continue に相当

パターン *(cont.)*

範囲指定も可能、以下 jack から joel まで

```
% awk '/jack/,/joel/{print}' data-a
jack     13          71,43
joel     81          44,52
```

出力フィールド

出力フィールド区切り文字は空白(0x20)
変数 OFS に格納されている

```
% awk ’{print $2,$1}’ data-a
32 john
13 jack
81 joel
% awk ’BEGIN{OFS=","}{print $2,$1}’ data-a
32,john
13,jack
81,joel
```

出力フィールド *(cont.)*

出力レコード区切り文字は改行(0x0a)
変数 ORS に格納されている

```
% awk 'BEGIN{ORS="\\\\\\n"}{print $2,$1}' d
32 john\\
13 jack\\
81 joel\\
```

```
% awk 'BEGIN{ORS=":"}END{ORS="";\
? print"\n"}{print $2,$1}' data-a
32 john:13 jack:81 joel:
```

連想配列(辞書)

配列の添字は文字列でも良い

```
% awk '{c[$1]=$2}\
? END{print c["jack"]}' data-a
13
% awk '{c[$1]=$2}END{\
? for(i in c)print i,c[i]}' data-a
jack 13
john 32
joel 81
```

for の in 演算子は配列の添字を使うため

連想配列(辞書) *(cont.)*

ただし、添字は整列されていないため、整列したい場合は外部で支援する必要がある

```
% awk '{c[$1]=$2}END{\
? for(i in c)print i,c[i]}' data-a |sort
jack 13
joel 81
john 32
```

連想配列(辞書) *(cont.)*

配列に含まれるかどうかの判定は演算子 in を使う
例:ファイルの属性を数え、新出の際にその旨表示する

```
NF>2{ if(!($1 in c)) { print "new",$1 } c[$1]++ }
END{ for(i in c) { print i,c[i] } }
```

```
% ls -l / | awk -f c.awk | pr -2 -t -w40
new drwxr-xr-x       dr-xr-xr-x 6
new lrwxrwxrwx       drwxrwxrwt 1
new dr-xr-xr-x       lrwxrwxrwx 4
new drwx------       drwx------ 2
new -rw-------       drwxr-xr-x 13
new drwxrwxrwt       -rw------- 1
```

連想配列(辞書) *(cont.)*

以下の記述でも同じ効果が得られるが...

```
NF>2{ if(c[$1]==0) { print "new",$1 } c[$1]++ }
END{ for(i in c) { print i,c[i] } }
```

- c[$1] の式を評価すると c[$1] に初期値が代入されてしまう
- 内容が 0 になる状況には対応できない
- 定義済かどうかの確認は in演算子を使う

関数

function で関数を定義

```
% cat b.awk
function max(a,b) {
    return a>b ? a : b
}
{ s = max(s,$2) }
END { print s }
% awk -F',' -f b.awk data-a
52
% awk -f b.awk data-a
81
```

参考文献

A.V.エイホ, B.W.カーニハン, P.J.ワインバーガー「プログラミング言語AWK」トッパン,ISBN 4-8101-8008-5, 1989