誰にでも書ける #! /bin/awk -f 講座
第 3 回
「タルィフの転逆び再」

安岡孝一

yasuoka : root さん、 root さん。

root : 何だい？

yasuoka : 今日は awk の while を、教えてくれるっていう約束だったじゃないですか。

root : ああ、そうだったね。じゃ、何か例をあげようか。前に sed で書いた、入力の中の yasuoka を YASUOKA に変換するスクリプト、あれを見せてくれるかい？

yasuoka : はい。

```
/home/yasuoka/bin% cat yasuoka （ぽこ）
#! /bin/sed -f
/ "yasuoka" Version 1.0 /{}
s/yasuoka/YASUOKA/g
/home/yasuoka/bin% ■
```

root : うん、これだこれだ。これを awk のスクリプトに書き換えてみようか。

```
#! /bin/awk -f
# "yasuoka" Version 2.0
{
  t=$0;
  while(i=index(t,"yasuoka"))
    t=substr(t,1,i-1) "YASUOKA" substr(t,i+7);
  printf("%s\n",t);
}
```

（間）

yasuoka : できました。

```
/home/yasuoka/bin% who （ぽこ）
yasuoka  console Mar 17 15:00
takahash ttyp0   Mar 17 11:17   (kinkaku)
ochi     ttyp1   Mar 17 15:09   (daikaku)
/home/yasuoka/bin% who | yasuoka （ぽこ）
YASUOKA  console Mar 17 15:00
takahash ttyp0   Mar 17 11:17   (kinkaku)
ochi     ttyp1   Mar 17 15:09   (daikaku)
/home/yasuoka/bin% echo yasuoka yasuoka | yasuoka （ぽこ）
YASUOKA YASUOKA
/home/yasuoka/bin% ■
```

うまくいってますね。5 行目の whileって、中の条件が真の間は後のコマンドを繰り返す、って意味ですか？

root : うん、そう。

```
while(条件)
  命令;
      条件が真である間、命令を繰り返し実行する。条件は if に同じ。
```

yasuoka : すると i=index(t,"yasuoka") が真の間 t=substr(t,1,i-1) "YASUOKA" substr(t,i+7) を繰り返すってことだから…。 i=index(t,"yasuoka")ってどういう時に真なんだろ。

root : index(t,"yasuoka") が 0 以外の時、つまり t の中に yasuokaって文字列が含まれてる時だよ。 i=index(t,"yasuoka") は index(t,"yasuoka") の値をそのまま返すからね。しかも i には、 t の何文字目に yasuoka があったかが代入される。

yasuoka : ああ、その i の値を t=substr(t,1,i-1) "YASUOKA" substr(t,i+7) で使ってるんですね。 substr(t,1,i-1) は t の中の yasuoka の直前まで、 substr(t,i+7) は t の中の yasuoka の直後から、その間に "YASUOKA"だから、 t の中の yasuoka のところが YASUOKA に置き換わったのが、また t に代入される。

root : そういうこと。

yasuoka : で、 t の中に yasuoka がなくなるまでそれを繰り返す、っと。なかなか。

```
/home/yasuoka/bin% yasuoka yasuoka （ぽこ）
#! /bin/awk -f
# "YASUOKA" Version 2.0
{
```

```
    t=$0;
    while(i=index(t,"YASUOKA"))
      t=substr(t,1,i-1) "YASUOKA" substr(t,i+7);
    printf("%s\n",t);
  }
  /home/yasuoka/bin% ■
```

while の後には、命令は 1 つしか書けないんですか？

root ： いや、 if と同じで、 { } を使えば複数書ける。

yasuoka ： そうですか。じゃあ…。

（間）

yasuoka ： root さん、 root さん。

root ： 何だい。

yasuoka ： 前に sed で書いた revってスクリプトがあったでしょう？

```
/home/yasuoka/bin% cat rev~ （ぽこ）
#! /bin/sed -f
/ "rev" Version 1.0 /{}
h
G
:loop
s/^\(.*\).\(\n\)\(.\)/\3\1\2/
/\n./bloop
s/\n//
/home/yasuoka/bin% ■
```

root ： ああ、行の左右をひっくり返すやつか。

yasuoka ： はい。それを awk で書き直したんですけど

```
/home/yasuoka/bin% cat rev （ぽこ）
#! /bin/awk -f
# "rev" Version 2.0
{
  i=length($0);
  while(i>0){
    printf("%s",substr($0,i,1));
    i--;
  }
  printf("\n");
}
/home/yasuoka/bin% ■
```

見てくれますか？

```
/home/yasuoka/bin% rev rev （ぽこ）
f- kwa/nib/ !#
0.2 noisreV "ver" #
{
;)0$(htgnel=i
{)0>i(elihw
;))1,i,0$(rtsbus,"s%"(ftnirp
;--i
}
;)"n\"(ftnirp
}
/home/yasuoka/bin% ■
```

root ： うーん、 while を使うよりは…。

yasuoka ： while を使うよりは？

root ： for を使った方がいいんじゃないかなぁ。

```
#! /bin/awk -f
# "rev" Version 2.1
{
  for(i=length($0);i>0;i--)
    printf("%s",substr($0,i,1));
  printf("\n");
}
```

yasuoka ： forって？

```
for(命令;条件;命令)
  命令;
      最初の命令を実行した後、条件が真である間、直後の命令と条件の後に書かれた命令とを繰り返す。条件は if に同じ。
```

root : まあようするに、 while の直前の命令と、条件と、ループの最後の命令とをひとまとめに書ける命令だよ。

yasuoka : へーえ。なかなか便利なのがあるんですね。するとこの

```
for(i=length($0);i>0;i--)
  printf("%s",substr($0,i,1));
```

ってのは

```
i=length($0);
while(i>0){
  printf("%s",substr($0,i,1));
  i--;
}
```

と全く同じ動きをするんですか？

root : うん、そう。ループの中に continue がない時には、 for と while はお互いに置き換えられる。

yasuoka : continue がない時って？

```
continue;
      ループの最後にジャンプする。ループを繰り返す条件が成立していれば、再
      度ループを繰り返す。
break;
      ループから飛び出す。
```

root : while の中の continue は、条件のところにジャンプすると考えればいいけど、 for の中の continue は、最後の命令を実行してから条件のチェックにいく。だから for をそのまま while に置き換えると、 continue の時に最後の命令を実行しなくなる、っていう違いが起きる。ま、それ以外は、単純に置き換えればいいけどね。

yasuoka : じゃ、 for(i=length($0);i>0;i--) のループの中に continue があったら、 i-- を実行してから i>0 のチェックにいくんですか？

root : そういうこと。

yasuoka : そうなんですか。 break の時はどうなるんですか？

root : break については、 for でも while でも動作はいっしょだ。単純にループを飛び出す。最後の命令は関係ない。

yasuoka : break で多重ループは飛び出せないんですか？

root : 飛び出せない。飛び出すのは 1 つだけ。

yasuoka : ちょっと不便ですね。

root : そうかな。

yasuoka : ところで root さん。

root : 何だい？

yasuoka : 前に sed で書いた、行を逆順にするスクリプト

```
/home/yasuoka/bin% cat tac（ぽこ）
#! /bin/sed -f
/ "tac" Version 1.1 /{}
1!G
h
$!d
/home/yasuoka/bin% ■
```

これを awk で書いたら、どうなります？

root : これはそんなには難しくなさそうだな。

```
#! /bin/awk -f
# "tac" Version 2.0
{
  buf[NR]=$0;
}
END{
  for(i=NR;i>0;i--)
    printf("%s\n",buf[i]);
}
```

（間）

root : できたかい？

yasuoka : はい。

```
/home/yasuoka/bin% tac tac（ぽこ）
}
    printf("%s\n",buf[i]);
  for(i=NR;i>0;i--)
END{
}
  buf[NR]=$0;
```

```
{
# "tac" Version 2.0
#! /bin/awk -f
/home/yasuoka/bin% 
```

うまくいってるみたい。

```
/home/yasuoka/bin% cat tac （ぽこ）
#! /bin/awk -f
# "tac" Version 2.0
{
  buf[NR]=$0;
}
END{
  for(i=NR;i>0;i--)
    printf("%s\n",buf[i]);
}
/home/yasuoka/bin% 
```

どういうしかけなんですか？

root ： buf[1] に入力の 1 行目を、 buf[2] に 2 行目を、っていう風に順に入れていって、最後に逆順に出力してるんだよ。

yasuoka ： buf[1]って配列ですか？

root ： よくわかったね。 awk では普通の変数以外に配列も使えるんだ。しかも、添字に数以外のものも使える。

yasuoka ： え？ どういう意味ですか？

root ： buf["soeji"] とかも使えるってことだよ。

yasuoka ： へーえ。

root ： て言うか、第 1 回でも話したとおり、 awk では数と文字列の区別はないからね。 buf[1] と buf["1"] は同じものを指すんだ。

yasuoka ： END{の中では NR の値はどうなってるんですか？

root ： 入力の最後の行での NR の値が、そのまま残ってる。

yasuoka ： それなら、これで逆順になりますね。ふーん、なかなか。

root ： さて、あと awk で説明してない大事な命令は 1 つだけだから、それを説明して終わりにしようか。

split(文字列,配列)
　　文字列をフィールド毎に区切り、配列[1]、配列[2]、…に代入する。フィールド数を返す。
split(文字列,配列,文字列)
　　右文字列をフィールドの区切りとみなし、他は上に同じ。右文字列は長さが 1 でなければならない。

root ： splitって何か命令みたいだけど、「変数=式」なんかと同じ代入式の一種だと考えた方がいいな。配列への特殊な代入式だ。

yasuoka ： どういうふうに使うんですか？

root ： じゃあ、 split を使った例をひとつあげよう。ファイル名は minicalc だ。

```
#! /bin/awk -f
# "minicalc" Version 1.0
BEGIN{
  printf("? ");
}
{
  if((i=split($0,p,"+"))==0)
    exit;
  for(a=0;i>0;i--){
    j=split(p[i],q,"*");
    for(b=1;j>0;j--){
      k=split(q[j],r,"^");
      for(c=r[k--];k>0;k--)
        c=exp(c*log(r[k]));
      b*=c;
    }
    a+=b;
  }
  printf("%f\n? ",a);
}
```

（間）

yasuoka ： できました。どうやって使うんですか？

root ： まずは minicalc を実行してみてごらん。

yasuoka ： はい。

```
/home/yasuoka/bin% minicalc （ぽこ）
? ■
```

root ： + で足し算、 * で掛け算、^ で冪乗が使えるから、適当に式を入れてみてごらん。あ、演算には優先順位があるよ。冪乗が再優先で、掛け算、足し算の順。

yasuoka ： わかりました。じゃまずは、 2 の 3 の 2 乗乗。

```
? 2^3^2 （ぽこ）
512.000000
? ■
```

なーかなか。割り算はないんですか？

root ： ないけど、負の数が使えるから掛け算と冪乗で代用できるだろ？

yasuoka ： あ、そうか。じゃ、 1 足す 2 割る 9っと。

```
? 1+2*9^-1 （ぽこ）
1.222222
? ■
```

式にカッコは使えないんですか？

root ： 使えない。

yasuoka ： あら、残念。終わるにはどうするんですか？

root ： 単にリターン。

yasuoka ： リターンっと。

```
? （ぽこ）
/home/yasuoka/bin% ■
```

どういうしかけなんですか？

root ： まず 4 行目で、? のプロンプトを出してるのはわかるね？

yasuoka ： はい。

root ： ここで、さっきの 1+2*9^-1 を入力した場合を考えると、 7 行目の split で p[1] には 1 が、 p[2] には 2*9^-1 が代入される。もちろん文字列としてね。で、 9 行目から 18 行目までのループで、 p[1] を計算した結果と p[2] を計算した結果の合計が a に入れられる。

yasuoka ： うーん。

root ： で、 p[2] の方、つまり i が 2 の時を考えると、 10 行目の split で q[1] には 2 が、 q[2] には 9^-1 が代入される。この 2 つの積が、 11 行目から 16 行目までのループで b に入るんだ。

yasuoka ： はい。だんだんわかってきました。

root ： さらに q[2] の方、つまり j が 2 の時を考えると、 12 行目の split で r[1] には 9 が、 r[2] には -1 が代入されるから、 c=exp(-1*log(9)) で c は 9 分の 1 になる。

yasuoka ： ええっと、そうなるのかな。

root ： そうなる。で、それが 2 と掛け合わされて、さらに 1 が加えられて、 19 行目の printf で出力される。同時に? のプロンプトも、また出力してる。

yasuoka ： だいたいわかりました。で、単にリターンを押した時終了するってのは、 8 行目の exit ですか？

root ： うん、そう。単にリターンを押すと i には 0 が代入されるからね。あっと、ここでひとつ注意しておかなきゃいけないな。

yasuoka ： 何ですか？

root ： awk の条件で、代入式と比較の両方を使う時には、代入式の方をカッコでくくっておくこと。ちょうどこの 7 行目の (i=split($0,p,"+")) のようにね。ああそれから、 split にはひとつ問題があるんだった。

yasuoka ： 問題って何ですか？

root ： 左文字列の中に改行コードが入ってると、正常に動かないことがあるんだよ。この minicalc ではそんなことはないから、大丈夫だけどね。

yasuoka ： わかりました。

root ： さて、これで awk についての話は終わりだ。どうだい、役に立ったかい？

yasuoka ： そうですね。これから頑張って役に立たせていこうと思います。

root ： よし、その意気だ。じゃ、 awk についてはここまでにしよう。

yasuoka ： はい。どうもありがとうございました。