誰にでも書ける #! /bin/nawk -f 講座
第 4 回
「関数の引数と変数」

安岡孝一

yasuoka : root さん、 root さん。
root : 何だい？
yasuoka : 今日は nawk の関数定義について、みっちり教えてくれる約束だったじゃないですか。
root : そういえば、そうだったね。じゃ、次のスクリプトをあげよう。名前は ugrep だ。

```
#! /bin/nawk -f
# "ugrep" Version 1.0
BEGIN{
  if(ARGC==1){
    system("echo 'Usage: ugrep exp [file]' >&2");
    exit(1);
  }
  e=ARGV[1];
  for(i=1;i<ARGC;i++)
    ARGV[i]=ARGV[i+1];
  if(--ARGC==1)
    ARGV[ARGC++]="-";
}
{
  f=t="";
  ugrep($0);
  if(f)
    printf("%s\n%s\n",$0,t);
}
function ugrep(s){
  if(match(s,e)){
    for(i=1;i<RSTART;i++)
      t=t " ";
    for(i=0;i<RLENGTH;i++)
      t=t "^";
    f=1;
    if(RLENGTH>0&&e!~/^\^/)
      ugrep(substr(s,RSTART+RLENGTH));
  }
}
```

（間）

root : 出来たかい？
yasuoka : はい。でもこれ、どうやって使うんですか？
root : egrep と同じなんだけど、マッチしたところに^ を付けてくれるんだ。ちょっと試してみてごらん。
yasuoka : はい。

```
~/bin% who | ugrep yasuoka （ぽこ）
yasuoka  console Apr  7 14:22
^^^^^^^
~/bin% who （ぽこ）
yasuoka  console Apr  7 14:22
hama     ttyh0   Apr  7 13:08
takahash ttyh1   Apr  6 13:14
~/bin% who | ugrep : （ぽこ）
yasuoka  console Apr  7 14:22
                              ^
hama     ttyh0   Apr  7 13:08
                                ^
takahash ttyh1   Apr  6 13:14
                                ^
~/bin%
```

ふーん、なるほど。

```
~/bin% ugrep ugrep ugrep （ぽこ）
# "ugrep" Version 1.0
   ^^^^^
    system("echo 'Usage: ugrep exp [file]' >&2");
                           ^^^^^
  ugrep($0);
  ^^^^^
function ugrep(s){
         ^^^^^
      ugrep(substr(s,RSTART+RLENGTH));
      ^^^^^
~/bin% 
```

正規表現も使えるんですか？

root ： うん、使えるよ。

yasuoka ： じゃーっと。

```
~/bin% ugrep 'u.*p' ugrep （ぽこ）
# "ugrep" Version 1.0
   ^^^^^
    system("echo 'Usage: ugrep exp [file]' >&2");
                           ^^^^^^^^^
  ugrep($0);
  ^^^^^
function ugrep(s){
 ^^^^^^^^^^^^
      ugrep(substr(s,RSTART+RLENGTH));
      ^^^^^
~/bin% who | ugrep '^[^ ]+' （ぽこ）
yasuoka  console Apr  7 14:22
^^^^^^^
hama     ttyh0   Apr  7 13:08
^^^^
takahash ttyh1   Apr  6 13:14
^^^^^^^^
~/bin% 
```

うーん、なかなか。

```
~/bin% who | ugrep h （ぽこ）
hama     ttyh0   Apr  7 13:08
^           ^
takahash ttyh1   Apr  6 13:14
    ^  ^    ^
~/bin% 
```

あ、複数の場所にもマッチするとは、よくできてますね。どういう仕掛けなんですか？

root ： そうだな、仕掛けを説明する前に、まず 21 行目の match を説明しておこうか。

match(文字列,/正規表現/)
　文字列の何文字目から正規表現がマッチしたかを返すと同時に、その値を RSTART にセットし、マッチした文字数を RLENGTH にセットする。なければ 0 を返す。
match(文字列,文字列)
　右文字列を正規表現とみなし、他は上に同じ。

yasuoka ： index の正規表現版ってことですか？

root ： まあそういうことかな。マッチした文字数が RLENGTH に入るけどね。

yasuoka ： わかりました。

```
~/bin% cat ugrep （ぽこ）
#! /bin/nawk -f
# "ugrep" Version 1.0
BEGIN{
  if(ARGC==1){
    system("echo 'Usage: ugrep exp [file]' >&2");
    exit(1);
  }
  e=ARGV[1];
  for(i=1;i<ARGC;i++)
    ARGV[i]=ARGV[i+1];
  if(--ARGC==1)
    ARGV[ARGC++]="-";
```

```
}
{
  f=t="";
  ugrep($0);
  if(f)
    printf("%s\n%s\n",$0,t);
}
function ugrep(s){
  if(match(s,e)){
    for(i=1;i<RSTART;i++)
      t=t " ";
    for(i=0;i<RLENGTH;i++)
      t=t "^";
    f=1;
    if(RLENGTH>0&&e!~/^\^/)
      ugrep(substr(s,RSTART+RLENGTH));
  }
}
~/bin% ■
```

で、どういう仕掛けなんですか？

root ： じゃあとりあえず

```
~/bin% echo who are you | ugrep .o. (ぽこ)
who are you
 ^^^     ^^^
~/bin% ■
```

を例にして、動作を説明してみようか。

yasuoka ： はい。

root ： まず BEGIN が実行されて、4 行目から 7 行目までは、ちゃんとパラメータが指定してあるかどうかのチェック。ここでは ARGC が 2 だから、4 行目の if は成立しない。さらに 8 行目で変数 e に「.o.」っていう文字列が代入される。

yasuoka ： はい。

root ： 9 行目と 10 行目で ARGV をシフトして、11 行目で ARGC を 1 減らす。ただし、もし ARGC が 1 になっちゃったなら、最初のパラメータを - にして、ARGC を 2 にする。ここはいいね？

yasuoka ： はい。でもどうしてパラメータをシフトするんですか？

root ： 第 2 パラメータから後を、ファイル名として扱うためだよ。nawk は BEGIN が済んだ後は、ARGV[1] から ARGV[ARGC-1] までをファイル名とみなして、順番に入力をおこなうからね。でも ARGC が 1 だと入力がおこなわれないから、その時には ARGV[1] を - にして、標準入力から入力をおこなうようにしてるんだ。

yasuoka ： うーん、そういうことですか。

root ： で、who are you が入力されて、14 行目に行く。15 行目で f と t にヌルが代入された後、16 行目で関数 ugrep が呼び出される。

yasuoka ： はい。

root ： で、実行は 20 行目に移って、引数 s に「who are you」が代入されて実行開始。

yasuoka ： ちょっと待って下さい。引数ってなんですか？

root ： 関数の定義に出てくる変数のことだよ。この ugrepっていう関数では s だけだね。nawk では引数は関数の中でだけ使える変数になるんだ。

yasuoka ： どういう意味ですか？

root ： 例えば、この ugrep 関数の中で、sっていう変数に代入をおこなっても、それは外の sっていう変数には何の影響も及ぼさない。けれどそれ以外の変数に代入すると、それは外の変数にも影響が及ぶんだよ。

yasuoka ： 大体わかりますけど…。

root ： じゃあ、もう少し続けてみよう。

yasuoka ： はい。

root ： 21 行目の match(s,e) では、s は「who are you」で、e は「.o.」だから、2 文字目から 3 文字分マッチする。ここはいいね？

yasuoka ： えっと、o を中心に、h、o、空白、のところにマッチするんですね。

root ： そういうこと。で、match(s,e) の結果は 2 になるから、if が成立する。同時に RSTART に 2、RLENGTH に 3 が代入される。

yasuoka ： 2 文字目から 3 文字分マッチしたからですね。

root ： そう。それから 22 行目から 25 行目で、変数 t に空白が 1 つと^ が 3 つ追加される。t は元々ヌルだったから、結果は「 ^^^」になる。ここまではいいね？

yasuoka ： 変数 t が元々ヌルだったっていうのは？

root ： さっき 15 行目でヌルを代入しただろ？

yasuoka ： ああ、そうでしたね。

root : で、27 行目の if は、RLENGTH が 3 で e が「.o.」だから成立して、最後から 3 行目で関数 ugrep が呼び出される。substr(s,2+3) で呼び出されるから、今度は引数 s には「are you」が代入されて、21 行目から実行開始。

yasuoka : 関数の中から関数を呼び出せるんですか？ しかも、自分自身を。

root : もちろん呼び出せる。

yasuoka : すごい。

root : 21 行目の match(s,e) では、s は「are you」で、e は「.o.」だから、5 文字目から 3 文字分マッチする。それで RSTART には 5、RLENGTH には 3 が代入される。22 行目から 25 行目では、t に空白が 4 つと^ が 3 つ追加されて「 ^^^ ^^^」になる。ここまではいいね？

yasuoka : はい。

root : さらに 27 行目の if が成立して、28 行目でまた関数 ugrep が呼び出される。今度は substr(s,5+3) だから、20 行目で引数 s にはヌルが代入されて、実行開始。でも 21 行目で match(s,e) が 0 だから if が成立しなくて、最後の行まで行って、関数からリターンする。

yasuoka : return がありませんけど？

root : return がなくても、関数定義の最後まで行ったら、自動的にリターンするんだ。

yasuoka : そうなんですか。

root : で、さっき ugrep を呼び出した 28 行目の直後に戻って、そこから最後の行までは何もなくて、またリターン。

yasuoka : はい。

root : また 28 行目の直後に戻って、最後の行まで何もなくて、またリターン。今度は 16 行目の直後に戻る。17 行目では、f が 1 なので if が成立して、18 行目で

```
who are you
 ^^^    ^^^
```

が表示される。これで実行は終りだね。

yasuoka : なかなかすごいですね。

```
~/bin% nawk '{printf("%2d %s\n",NR,$0);}' ugrep （ぽこ）
 1 #! /bin/nawk -f
 2 # "ugrep" Version 1.0
 3 BEGIN{
 4   if(ARGC==1){
 5     system("echo 'Usage: ugrep exp [file]' >&2");
 6     exit(1);
 7   }
 8   e=ARGV[1];
 9   for(i=1;i<ARGC;i++)
10     ARGV[i]=ARGV[i+1];
11   if(--ARGC==1)
12     ARGV[ARGC++]="-";
13 }
14 {
15   f=t="";
16   ugrep($0);
17   if(f)
18     printf("%s\n%s\n",$0,t);
19 }
20 function ugrep(s){
21   if(match(s,e)){
22     for(i=1;i<RSTART;i++)
23       t=t " ";
24     for(i=0;i<RLENGTH;i++)
25       t=t "^";
26     f=1;
27     if(RLENGTH>0&&e!~/^\^/)
28       ugrep(substr(s,RSTART+RLENGTH));
29   }
30 }
~/bin% 
```

あ、27 行目で e の最初の文字が^ かどうか判定してるのは、どうしてですか？

root : ugrep では^ は行頭っていう意味にしたいんだけど、nawk の正規表現では^ は文字列の先頭だからね。これじゃうまくないので、^ の処理は一回だけするようになってるんだ。

yasuoka : うーん、わかったような、わからないような。

root : それじゃあ

```
~/bin% echo who are you | ugrep '^.' (ぽこ)
who are you
^
~/bin%
```

の動作を、自分で考えてごらん。

yasuoka : えっと、まず 4 行目から 7 行目に飛んで、8 行目で変数 e に「^.」って文字列が代入される。これでいいんですよね。

root : そういうこと。

yasuoka : で、 ARGV をシフトする代わりに ARGV[1] に "-"を代入して、15 行目で f と t にヌルを代入して、16 行目で関数 ugrep を呼び出す。引数 s に代入されるのは「who are you」と。

root : それからそれから？

yasuoka : 21 行目の match(s,e) では、 s は「who are you」で、 e は「^.」だから、 1 文字目から 1 文字分マッチする。あ、ここなんですね。

root : まあね。でも真髄は、もうちょっとしてからわかるよ。

yasuoka : そうなんですか？ これで RSTART は 1、 RLENGTH は 1 になるから、 t には空白が 0 個と^ が 1 個追加されて「^」になる。で、さらに最後から 3 行目で、関数 ugrep を呼ぶんですよね。今度は引数は「ho are you」で。

root : いや違う。その 1 行前の if が成立しないんだ。 e が「^.」で^ から始まってるからね。

yasuoka : あ、ここが真髄なんですね。

root : よくわかったね。

yasuoka : そういうことでしたか。で、最後の行でリターンして 16 行目の直後に戻ってくる。

root : そうそう。

yasuoka : f は 1 だから 17 行目の if が成立して、 18 行目の printf で

```
who are you
^
```

が表示されるっと。

root : よくできました。

yasuoka : へへー。

root : ただ実は、まだこれでも行頭の判定は弱くて、例えば (^.) とか書かれると、もうお手上げなんだよ。

yasuoka : え、そうなんですか？

```
~/bin% echo who are you | ugrep '(^.)' (ぽこ)
who are you
^^^^^^^^^^^
~/bin%
```

確かに変ですね。それじゃあ、 27 行目の判定を e!~/^\(*\^/ にしたら、いいんじゃないですか？

root : それでも (a|^.) は防げない。

yasuoka : あ、そうか。それなら e!~/^\^|[|(]\^/ は、どうですか？

root : それだと、今度は [(^] が引っかかってしまう。

yasuoka : うーん、難しいなぁ。無理なのかなぁ。

```
~/bin% echo who are you | ugrep '.$' (ぽこ)
who are you
          ^
~/bin%
```

$ は特別な処理は要らないんですか？

root : 文字列を前から順番に処理していくからね。 $ があったら、末尾にマッチしてそれでおしまい。ただ、 nawk の古いヴァージョンには match にバグがあるのがあって、例えば今の echo who are you | ugrep '.$' だと

```
who are you
          ^^
```

になってしまうのがあるらしい。 $ にマッチすると、 RLENGTH が 1 多くなってしまうんだそうだ。

yasuoka : そういうのがあるんですか。

root : まあ、単なるバグだけどね。さて、これで nawk について一通り説明し終わった。どうだい？ ちゃんと nawk のスクリプトが書けそうかい？

yasuoka : まだまだだとは思いますけど、頑張ってみます。本当にどうもありがとうございました。