誰にでも使える dc 講座
第 1 回
「expr は 10 桁以上の数を計算できない」

安岡孝一

yasuoka : root さん、 root さん。
root : 何だい？
yasuoka : 階乗を計算するコマンドを作ったんですけど、結果が何か変なんです。
root : 変って？
yasuoka : えっとですね

```
~/bin% cat factorial （ぽこ）
#! /bin/sh
# "factorial" Version 1.0

N=${1-0}
ANS=1

while [ $N -gt 1 ]
do ANS=‘expr $ANS ’*’ $N‘
   N=‘expr $N - 1‘
done

echo $ANS
exit 0
~/bin% ■
```

こういうシェル・スクリプトなんですけど、 10 の階乗はちゃんと計算できるのに

```
~/bin% factorial 10 （ぽこ）
3628800
~/bin% ■
```

20 の階乗は変な値になっちゃうんです。

```
~/bin% factorial 20 （ぽこ）
-2102132736
~/bin% ■
```

どこが間違ってるんでしょう？
root : expr のせいだな。 expr は 10 桁以上の数は計算できないからね。
yasuoka : じゃあ、 20 の階乗を計算したい時は、どうしたらいいんでしょう？
root : dc を使う。立ち上げてごらん。
yasuoka : dc ですか？

```
~/bin% dc （ぽこ）
■
```

root : まずは 5 の階乗をやろうか。２ ３ * ４ * ５ * p してごらん。
yasuoka : ２ ３ * ４ * ５ * p ですね。

```
２ ３ * ４ * ５ * p （ぽこ）
120
■
```

あ、 5 の階乗だ。どういう仕掛けなんですか？
root : まず覚えておかなきゃいけないのは、 dc での計算は、全てスタックを介しておこなわれるってこと。このスタックってのは、数とかを積んでおくための内部的な場所で、基本的には一番上にあるものしかアクセスできない。 dc が起動された時には、スタックは空だ。
yasuoka : はい。
root : それから、 dc の中で使えるのは dc の命令だけだ。 dc の命令は、 1 行にいくつ書いてもいい。
yasuoka : はい、わかりました。
root : で、実際の dc の命令だけど、まずはこの２ ３ * ４ * ５ * p を例に話そう。最初の２は、 2っていう数をスタックに積む、っていう命令だ。

数
数をスタックに積む。最初の文字が _ である場合には、負の数を表す。

root : 次の３も、 3 をスタックに積む、っていう命令だから、これでスタックは

3
2

っていう風になる。

yasuoka ： はい。
root ： 次の * は、スタックの一番上の 2 つをかけあわせて、その結果を代わりにスタックに積む、っていう命令だから、これでスタックは

6

になる。

+	スタックの一番上と、その次の数を取り出し、その和をスタックに積む。
-	スタックの一番上と、その次の数を取り出し、下の数から上の数を引いた差をスタックに積む。
*	スタックの一番上と、その次の数を取り出し、その積をスタックに積む。
/	スタックの一番上と、その次の数を取り出し、下の数を上の数で割った商をスタックに積む。
%	スタックの一番上と、その次の数を取り出し、下の数を上の数で割った余りをスタックに積む。
^	スタックの一番上と、その次の数を取り出し、下の数を上の数で冪乗した結果をスタックに積む。

yasuoka ： * が、かけ算の命令なんですね。
root ： そう。その次の 4 でスタックは

4
6

になり、次の * で

24

さらに 5 で

5
24

そして * で

120

となる。ここまではいいね？
yasuoka ： はい。よくわかります。
root ： そして最後の p だけど、これはスタックの一番上の数を表示する命令だ。これで 120 が表示される。

p	スタックの一番上の数を、改行コード付きで標準出力に出力する。スタックは変化しない。

root ： p はスタックの一番上を取り出したりはしないから、今もスタックには

120

が積まれたままだ。
yasuoka ： じゃあ、さらに 6 * を実行したら、 6 の階乗が得られるんですね。
root ： やってごらん。
yasuoka ： はい。

```
6 *（ぽこ）
```

あれ、何も起こらない。
root ： p してごらん。
yasuoka ： あ、そうでしたね。

```
p（ぽこ）
720
```

うーん、すごい。数は 1 桁だけなんですか？
root ： いや、整数なら何桁でも書けるし、何桁でも計算できる。それから小数も使えるよ。
yasuoka ： とすると

```
7 * 8 * 9 * 10 * p（ぽこ）
3628800
```

これで 10 の階乗っと。 100 ÷ 3 はどうなるのかな？

```
100 3 / p（ぽこ）
33
```

あれ？ 整数の割算しかできないんですか？

root ： 基本的にはね。小数点以下何桁ほしい？

yasuoka ： じゃあ大きく 20 桁！

root ： だったら 20 k だ。

yasuoka ： 20 k ですね。

```
20 k（ぽこ）
```

これでどうなったんですか？

root ： さっきの 100 3 / p をもう一度やってごらん。

yasuoka ： はい。

```
100 3 / p（ぽこ）
33.33333333333333333333
```

うーん、なかなか。

k スタックの一番上の数を取り出し、それを *、 /、 %、 ^、 v での小数部の有効桁数とみなす。 K 現在の有効桁数をスタックに積む。

yasuoka ： v って何ですか？

root ： 平方根。 2 v p とでもしてごらん。

v スタックの一番上の数を取り出し、その平方根をスタックに積む。

yasuoka ： 2 の平方根ですね。

```
2 v p（ぽこ）
1.41421356237309504880
```

でも ^ があるなら、 v なんてなくても、 2 0.5 ^ で 2 の平方根が計算できるんじゃないですか？

root ： 冪乗は冪数の方に小数は書けないんだよ。

yasuoka ： そうなんですか。すると立方根とかは無理なんですね。ところで、スタックには数は 2 個しか積めないんですか？

root ： いや、 100 個くらいは積めるはずだよ。今だってスタック上には

1.41421356237309504880
33.33333333333333333333
33
3628800

の 4 つが積んであるはずだ。

yasuoka ： スタックに数が何個積んであるか、調べる方法はないんですか？

root ： z p してごらん。

z 現在、スタック上に値が何個あるかを、スタックに積む。

yasuoka ： z がそういう命令なんですか？

```
z p（ぽこ）
4
```

root ： そういうこと。ただし今、スタック上には

4
1.41421356237309504880
33.33333333333333333333
33
3628800

の 5 つが積んであるわけだけどね。

yasuoka ： スタック上の数は取り除けないんですか？

root ： スタックを空にしたいなら、 c かな。

c スタック上の値を全て取り除き、スタックを空にする。

root ： さて、そろそろ dc を終了して、さっきの factorial を作り直してごらん。終了は q だ。

yasuoka ： はい。

```
q（ぽこ）
~/bin% 
```

（間）

yasuoka ： 出来ました。こんなもんですか？

```
~/bin% cat factorial（ぽこ）
#! /bin/sh
# "factorial" Version 1.1

N=${1-0}
( echo 1
  while [ $N -gt 1 ]
  do echo $N '*'
     N=`expr $N - 1`
  done
  echo p q
) | dc
exit 0
~/bin% factorial 5（ぽこ）
120
~/bin% factorial 20（ぽこ）
2432902008176640000
~/bin% 
```

root ： うん、正しいみたいだね。どういう仕掛けなんだい？

yasuoka ： えっと、例えば `factorial 5` だったら

```
1
5 *
4 *
3 *
2 *
p q
```

っていう命令を dc に入力します。

root ： それで正解。

yasuoka ： はー、よかった。ところで、 root さん、もう 1 つ見てほしいものがあるんですけど。

```
~/bin% cat oct2dec（ぽこ）
#! /bin/sh
# "oct2dec" Version 1.0

echo $1 |
( echo 0
  sed 's/\(.\)/8 * \1 + /g'
  echo p q
) | dc
exit 0
~/bin% 
```

root ： 8 進数の 10 進数への変換かな？ ちょっとやってみせてよ。

yasuoka ： 大当たりです。

```
~/bin% oct2dec 123（ぽこ）
83
~/bin% 
```

root ： どういう仕掛けなんだい？

yasuoka ： 今の `oct2dec 123` だったら

```
0
8 * 1 + 8 * 2 + 8 * 3 +
p q
```

っていう命令を dc に入力します。

root ： うーん、確かにそれでも悪くないんだけど、実は `8 i 123 p` で、簡単にできちゃうんだよ。

`i` スタックの一番上の数を取り出し、それを「数」での位取り基数とみなす。
`I` 現在の「数」での位取り基数をスタックに積む。

yasuoka ： `8 i 123 p` ですか？

```
~/bin% dc（ぽこ）
8 i 123 p（ぽこ）
83
```

あ、本当だ。この 8 i ってどういう意味なんですか？

root ： これ以降、 123 とかの「数」を 8 進数とみなす、っていう意味だよ。だから今、スタックには 123 じゃなくて

83

が積まれてるんだ。 10 + p してごらん。

yasuoka ： 10 を加えるんですね。

```
10 + p（ぽこ）
91
```

あれ？

root ： 10 じゃないよ。 8 進数での 10 は 8 を表すから、 10 の実行でスタックは

8
83

となって、次の + で

91

になったんだ。

yasuoka ： そういうことですか。ところで i って、何進数にでもできるんですか？

root ： 2 進数から 16 進数まで。数字として使えるのは、 0、 1、 2、 3、 4、 5、 6、 7、 8、 9、 A、 B、 C、 D、 E、 F の 16 種類だ。小文字はダメ。

yasuoka ： 表示の方は 10 進数以外は使えないんですか？

root ： 20 o としてごらん。

o 　　スタックの一番上の数を取り出し、それを p での位取り基数とみなす。 O 　　現在の p での位取り基数をスタックに積む。

yasuoka ： えっと、 20 は 16 だから

```
20 o（ぽこ）
```

これで表示が 16 進数になったんですね。

root ： そう。今、スタックには

91

があるはずだから p してごらん。

yasuoka ： はい。

```
p（ぽこ）
5B
```

うーむ、なかなか。入力の方を 10 進数に戻すには 12 i ですか？

root ： それでもいいけど、 A i の方がより一般的だな。どんな位取りの時でも、 A は 10 をスタックに積むからね。

yasuoka ： そうなんですか。

```
A i（ぽこ）
```

10 進数に戻ったはずだから、 100 を積んでみてと。

```
100 p（ぽこ）
64
```

あ、 o の方も直さなきゃ。

```
A o（ぽこ）
p（ぽこ）
100
```

よし、 OK。じゃあ、 oct2dec を作り直そうかな。

```
q（ぽこ）
~/bin%
```

root ： 今日はもう時間がないから、 oct2dec の作り直しは、次回までの宿題にしよう。それで大丈夫だろう、 yasuoka くん？

yasuoka ： はい、大丈夫だと思います。どうもありがとうございました。