誰にでも使える dc 講座
第 2 回
「dc のスクリプトはパラメータをとれない」

安岡孝一

yasuoka ： root さん、前回の oct2dec が出来たんですけど

```
~/bin% cat oct2dec （ぽこ）
#! /bin/sh
# "oct2dec" Version 1.1

echo 8 i ${1-0} p | dc
exit 0
~/bin% oct2dec 123 （ぽこ）
83
~/bin%
```

こんなもんでしょうか？

root ： 8 進数から 10 進数への変換だったね。うん、それでいいと思うよ。

yasuoka ： ところで root さん。

root ： 何だい？

yasuoka ： dc のスクリプトってないんですか？

root ： あるよ。じゃあ、この oct2dec を dc のスクリプトに書き換えてみようか。

```
[ [ ] ; exec dc $0 #] c
8 i [octal? ] P ? p q
```

yasuoka ： えらく短いんですね。

root ： まあね。しかも 1 行目の [[ ] ; exec dc $0 #] c は、このスクリプトが dc のスクリプトだっていう宣言みたいなもんだから、実質的には 2 行目の 8 i [octal?] P ? p q だけだ。もう出来たかい？

yasuoka ： はい。実行してみますね。

```
~/bin% oct2dec 123 （ぽこ）
octal?
```

あれ？ 何か訊いてきましたよ。

root ： dc のスクリプトには、パラメータを読む機能がないんだ。だからこのスクリプトでは、値を訊くようにしてる。適当な 8 進数を入力してごらん。

yasuoka ： じゃあ、パラメータに書いたのと同じ値で。

```
octal? 123 （ぽこ）
83
~/bin%
```

どういう仕掛けなんですか？

root ： まず最初の 8 i はわかるね。スタックに

```
8
```

を積んで、次にそれを取り除いて、入力を 8 進数とみなす。

yasuoka ： はい。

root ： 次の [octal?] だけど、これはスタックに

```
octal?
```

っていう文字列を積むんだ。

> [文字列]
> 文字列をスタックに積む。文字列中の文字は何であってもよいが、[を含む場合はそれと同数の] を含んでいなければならない。

yasuoka ： 文字列も扱えるんですね。

root ： 扱えるとは言っても、複数の文字列をくっつけたり、文字列の一部を取り出したりはできないけどね。で、次の P で octal? を表示して、スタックから取り除く。

> P
> スタックの一番上の文字列を取り出し、標準出力に出力する。改行コードは付加されない。

root ： さらに次の? だけど、これは標準入力から 1 行読み込んで、それを dc の命令として実行するんだ。

> ?
> 標準入力から 1 行分の文字列を読み込み、それを実行する。

root ： さっきの場合だと 123 を入力したから、それが実行されて、スタックには

```
83
```

が積まれたわけだ。その後、 p で 83 が改行コード付きで表示されて、 q で終了。

yasuoka : ?っていうのは、どんな命令でも実行できるんですか？

root : うん。試しに q を入力してごらん。

yasuoka : はい。

```
~/bin% oct2dec （ぽこ）
octal? ■
```

ここで q ですか？

```
octal? q （ぽこ）
~/bin% ■
```

あ、何も表示せずに終了した。

root : そういうこと。

yasuoka : ところで root さん。

root : 何だい？

yasuoka : dc には繰り返しはないんですか？

root : 繰り返す方法はある。この oct2dec を改造してみようか。

```
[ [ ] ; exec dc $0 #] c
8 i [[octal? ] P ? p s. d x] d x [bye.
] P q
```

（間）

yasuoka : 出来ました。

root : じゃ、実行してごらん。

yasuoka : はい。

```
~/bin% oct2dec （ぽこ）
octal? ■
```

root : で、さっきと同じように、何か 8 進数を入力する。

yasuoka : まずは 123 を入力します。

```
octal? 123 （ぽこ）
83
octal? ■
```

あ、また訊いてきた。

root : 別の値を入れてごらん。

yasuoka : それじゃ次は 20 にします。

```
octal? 20 （ぽこ）
16
octal? ■
```

終了するにはどうするんですか？

root : q で終了するはずだ。

yasuoka : わかりました、 q ですね。

```
octal? q （ぽこ）
bye.
~/bin% ■
```

bye. が表示されて、終了っと。どういう仕掛けなんですか？

root : 1 行目の [[ ] ; exec dc $0 #] c は、このスクリプトが dc のスクリプトだっていう宣言で、 2 行目の最初の 8 i は入力を 8 進数とみなす。これは、さっきと同じだね。次の [[octal?] P ? p s. d x] は、スタックに

```
[octal? ] P ? p s. d x
```

っていう文字列を積む。ここまではいいね？

yasuoka : はい。

root : 次の d だけど、これはスタックの一番上にあるものをコピーして、それをさらにスタックに積むんだ。この結果、スタックは

```
[octal? ] P ? p s. d x
[octal? ] P ? p s. d x
```

っていう風になる。

```
d
        スタックの一番上の値と同じものを、スタックに積む。
```

yasuoka : d は文字列のコピーしかできないんですか？

root : いや、スタックに積んであるものなら、何でもコピーできる。

yasuoka : そうなんですか。

root : それから、 2 行目の最後から 7 文字目の x で、スタックの一番上の文字列を実行する。

x 　　スタックの一番上の文字列を取り出し、実行する。

root ： スタックに積まれた文字列を実行できる、っていう機能が、繰り返しを可能にしてるんだ。

yasuoka ： どうしてですか？

root ： 順に追っていってみようか。今の x で、スタックは

```
[octal? ] P ? p s. d x
```

になって、取り出された [octal?] P ? p s. d x の実行が開始される。最初の [octal?] で、スタックは

```
octal?
[octal? ] P ? p s. d x
```

になり、次の P で octal? が表示されて

```
[octal? ] P ? p s. d x
```

になる。ここまではいいね？

yasuoka ： はい。

root ： 次の? で標準入力を 1 行読み込んで実行。さっきの場合は 123 だったから、スタックは

```
83
[octal? ] P ? p s. d x
```

になって、次の p で 83 が改行コード付きで表示されるけど、スタックは

```
83
[octal? ] P ? p s. d x
```

のままだ。

yasuoka ： p はスタックを変化させないんでしたね。

root ： そう。その次の s. は、実は細かい説明が必要なんだけど、ここではスタックの一番上の値を取り除くっていう意味だから、スタックは

```
[octal? ] P ? p s. d x
```

になる。さらに d で

```
[octal? ] P ? p s. d x
[octal? ] P ? p s. d x
```

になって、 x で

```
[octal? ] P ? p s. d x
```

になると同時に、 [octal?] P ? p s. d x の実行が開始される。これで繰り返しが実現できたわけだ。

yasuoka ： 同じ文字列を何度もコピーして実行する、っていうのが繰り返しになってるんですね。

root ： そういうこと。しかも x は、文字列の末尾にある時以外は、実行がすんだらその続きの部分に戻ってくるんだ。

yasuoka ： どういう意味ですか？

root ： octal? に対して q を実行したらどうなった？

yasuoka ： bye. が表示されて終了しました。

root ： それが 2 行目の最後なんだ。2 行目から 3 行目にかけての [bye.改] で、スタックは

```
bye.改
[octal? ] P ? p s. d x
```

になって、次の P で改行コード付きの bye. が表示された後、 q で終了。改ってのは改行コードね。これは、 2 行目の最後から 7 文字目の x の続きだろ？

yasuoka ： そうですけど、でも qってのは、実行終了って意味じゃなかったんですか？

root ： 文字列の実行中の時以外はね。

Q 　　スタックの一番上の数を取り出し、「数」段の文字列の実行から飛び出す。 q 　　2 Q に同じ。ただし、文字列の実行が 1 段以下（文字列の実行中でない場合を含む）の場合は、実行を終了する。

root ： でも、文字列を実行してる時は、 q はシェルの break 2 みたいなもので、実行中の文字列を 2 段分、飛び出すんだよ。この場合は、？で入力された文字列の実行から飛び出して、さらに [octal?] P ? p s. d x の実行から飛び出して、 2 行目の x の直後に戻ってくるんだ。

yasuoka ： うーむ、そういうことなんですか。

```
~/bin% cat oct2dec（ぽこ）
[ [ ] ; exec dc $0 #] c
```

```
8 i [[octal? ] P ? p s. d x] d x [bye.
] P q
~/bin% oct2dec（ぽこ）
octal? 2 Q（ぽこ）
bye.
~/bin%
```

root さん、これ、入力が何もなかった時は終了する、っていう風にできませんか？

root ： それは無理だな。? は空行が入力された場合は、標準入力をもう 1 行読み取りに行くからね。

yasuoka ： だったら仕方ないですね。それじゃ前回の

```
~/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%
```

を dc のスクリプトにできませんか？

root ： 階乗を求めるんだったね。うーん。

```
[ [ ] ; exec dc $0 #] c
[number? ] P ? sa 1 sb [q] sc
[la 2 >c la lb * sb la 1 - sa d x] d x
lb p q
```

（間）

yasuoka ： 出来ました。

```
~/bin% factorial 20（ぽこ）
number?
```

パラメータはやっぱり読めないんですね。

root ： dc のスクリプトの宿命だね。

yasuoka ： 仕方ないのかなぁ。

```
number? 20（ぽこ）
2432902008176640000
~/bin%
```

あれ？ 繰り返さないんですね。どういう仕掛けかな。

```
~/bin% cat factorial（ぽこ）
[ [ ] ; exec dc $0 #] c
[number? ] P ? sa 1 sb [q] sc
[la 2 >c la lb * sb la 1 - sa d x] d x
lb p q
~/bin%
```

1 行目は dc のスクリプトっていう意味ですよね。2 行目は

```
number?
```

をスタックに積んで、P でそれを表示して、? で 1 行入力して、 saって何ですか？

root ： スタックの一番上の値を取り出して、それを aっていう変数に代入する。

s変数 　　スタックの一番上の値を取り出し、変数に代入する。 l変数 　　変数の値をスタックに積む。

root ： dc では変数名は 1 文字だけだ。

yasuoka ： じゃあ変数名に TEMP とかは使えないんですね。

root ： そう。その代わりどんな文字でも使える。. でも # でもスペースでも。

yasuoka ： するとここでは、? で入力された値を変数 a に、1 を変数 b に、 qっていう文字列を変数 c に、それぞれ代入してるんですね。で、次の 3 行目では

```
la 2 >c la lb * sb la 1 - sa d x
```

をスタックに積んで、それを d でコピーして x で実行っと。 la で、変数 a を読み出してスタックに積む。その上に 2 が積まれて、>cって何ですか？

root ： la 2 >c で、変数 a の値が 2 未満だったら、変数 c に入った文字列を実行するっていう意味だ。

=変数
　スタックの一番上と、その次の数を取り出し、等しければ変数中の文字列を実行する。
!=変数
　スタックの一番上と、その次の数を取り出し、等しくなければ変数中の文字列を実行する。
>変数
　スタックの一番上と、その次の数を取り出し、上の数が下の数より大きければ、変数中の文字列を実行する。
!<変数
　スタックの一番上と、その次の数を取り出し、上の数が下の数以上ならば、変数中の文字列を実行する。
<変数
　スタックの一番上と、その次の数を取り出し、上の数が下の数未満ならば、変数中の文字列を実行する。
!>変数
　スタックの一番上と、その次の数を取り出し、上の数が下の数以下ならば、変数中の文字列を実行する。

root ： この場合は変数 c に入ってる文字列は q だから、結局、変数 a が 2 未満なら、 la 2 >c la lb * sb la 1 - sa d x の実行から飛び出して、 3 行目の末尾の x の後に行く。

yasuoka ： うーん、繰り返しの飛び出しみたいですね。

root ： 飛び出しっていうよりは、終了条件が先判定の繰り返しそのものだな。

yasuoka ： 変数 a が 2 未満になるまで、 la lb * sb と la 1 - sa を繰り返すわけですね。 la lb * sb は変数 b を変数 a 倍してるし、 la 1 - sa は変数 a のデクリメントですから、まさに階乗の計算そのものですね。

root ： そういうこと。

yasuoka ： で、 4 行目の lb p で結果を表示して、 q で終了っと。でもこれ、さっきの oct2dec みたいに、 q が入力されるまで何回でも繰り返すようにはできないんですか？

root ： できるよ。

```
[ [ ] ; exec dc $0 #] c
[[number? ] P ? sa 1 sb [q] sc
[la 2 >c la lb * sb la 1 - sa d x] d x s.
lb p s. d x] d x q
```

yasuoka ： いきなり [number?] P ? sa 1 sb [q] sc改[la 2 >c la lb * sb la 1 - sa d x] d x s.改lb p s. d xっていう文字列をスタックに積んで、それをコピーして実行開始ですか。すごく大胆ですね。

root ： 仕掛けはわかるかい？

yasuoka ： ええ、だいたいわかります。でも、ところどころで s. が挿入されてるのはなぜですか？

root ： スタックの要らない値を捨てるためだよ。

yasuoka ： そういえば oct2dec にも

```
~/bin% cat oct2dec（ぽこ）
[ [ ] ; exec dc $0 #] c
8 i [[octal? ] P ? p s. d x] d x [bye.
] P q
~/bin%
```

s. がありましたけど、これも同じ理由ですか？

root ： そうだよ。 s. は、スタックの一番上の値を取り出して .っていう変数に代入する、っていう意味だけど、大抵はスタックの一番上の値を捨てたい時に s. を使うんだ。 dc にはスタックの一番上を捨てるための命令がないからね。

yasuoka ： ふーん、そういうことですか。でも、 [q] sc は 1 回だけ実行すればいいような気がするんですけど。

root ： そう言われればそうだ。けどむしろ la lb * sb la 1 - sa を変数に入れておいた方が、すっきりするかもしれないな。次回までの宿題にしていいかい？

yasuoka ： はい、わかりました。どうもありがとうございました。