誰にでも使える dc 講座
第 3 回
「スタックだったり配列だったり」

安岡孝一

yasuoka : root さん、 root さん。
root : 何だい？
yasuoka : 前回の宿題が出来たんですけど

```
~/bin% cat factorial（ぽこ）
[ [ ] ; exec dc $0 #] c
[la lb * sb la 1 - d sa 1 <!] s!
[[number? ] P ? sa 1 sb l! x lb p s. d x] d x q
~/bin% ■
```

こんなもんでしょうか？
root : 変数名を！にするとは、なかなかやるな。やってみせてくれるかい？
yasuoka : はい。

```
~/bin% factorial（ぽこ）
number? 20（ぽこ）
2432902008176640000
number? 30（ぽこ）
265252859812191058636308480000000
number? q（ぽこ）
~/bin% ■
```

たぶん正しいはずですけど。
root : 0 の階乗は？
yasuoka : え？

```
~/bin% factorial（ぽこ）
number? 0（ぽこ）
0
number? ■
```

あら。どうしてでしょう？
root : 繰り返しが後判定になってるからね。変数 a が 0 でも、最低 1 回は実行してしまうのが間違いだ。
yasuoka : どうしたらいいんでしょう？
root : l! x を la 1 <! とでもしたらいいんじゃないかな。

```
[ [ ] ; exec dc $0 #] c
[la * la 1 - d sa 1 <!] s!
[[number? ] P ? sa 1 la 1 <! p s. d x] d x q
```

（間）

root : ついでに変数 b は使わないようにしてみたけど、もう出来たかい？
yasuoka : はい。

```
~/bin% factorial（ぽこ）
number? 0（ぽこ）
1
number? ■
```

うーむ、なかなかやりますね。 3 行目の sa と la の間で積まれてる 1 が、変数 b の代わりですか？
root : そういうこと。
yasuoka : ふーん。

```
number? 1（ぽこ）
1
number? q（ぽこ）
~/bin% ■
```

ところで root さん。
root : 何だい？
yasuoka : 数を表示する時って、必ず改行コードが付くんですか？
root : どういう意味だい？
yasuoka : 例えばこの factorial だったら、 0 を入力した時には 0!=1 とか表示できないのかなぁ、と思ったんですけど。
root : 20 の時には 20!=2432902008176640000って風に？
yasuoka : そうです。
root : それは数の代わりに文字を表示するしかないな。
yasuoka : どうするんですか？

root ： 例えば 20 を表示したいなら、2っていう文字と 0っていう文字を連続して表示する。

yasuoka ： うーん。数の桁数を知る方法はありますか？

root ： Z かな。

> Z
>
> スタックの一番上の数を取り出し、その桁数をスタックに積む。
>
> X
>
> スタックの一番上の数を取り出し、その小数部の桁数をスタックに積む。

（間）

yasuoka ： root さん、こんなもんでどうでしょう？

```
~/bin% cat factorial （ぽこ）
[ [ ] ; exec dc $0 #] c
[d 0 <1 s. [0] P] s0
[d 1 <2 s. [1] P q] s1
[d 2 <3 s. [2] P 3 Q] s2
[d 3 <4 s. [3] P 4 Q] s3
[d 4 <5 s. [4] P 5 Q] s4
[d 5 <6 s. [5] P 6 Q] s5
[d 6 <7 s. [6] P 7 Q] s6
[d 7 <8 s. [7] P 8 Q] s7
[8 <9 [8] P 9 Q] s8
[[9] P 10 Q] s9
[la 10 * 10 lb ^ / 10 % l0 x lb 1 - d sb 0 <P] sP
[la * la 1 - d sa 1 <!] s!
[[number? ] P ? sa la Z sb lP x [!=] P
1 la 1 <! p s. d x] d x q
~/bin% factorial （ぽこ）
number? 20 （ぽこ）
20!=2432902008176640000
number? 0 （ぽこ）
0!=1
number? q （ぽこ）
~/bin% 
```

root ： 0 から 9 まで、それぞれの文字を出力する文字列を、変数 0 から変数 9 に入れたんだね。でもこういう場合は、配列を使った方がたぶん楽だと思う。

yasuoka ： 配列？

> :変数
>
> スタックの一番上の値と、その次の数を取り出し、変数の「数」番目の要素に値を代入する。
>
> ;変数
>
> スタックの一番上の数を取り出し、変数の「数」番目の要素の値をスタックに積む。
>
> いずれも変数は配列であるとみなされる。要素を表す「数」は、0 以上 1024 未満の整数でなければならない。

root ： dc を立ち上げてごらん。

yasuoka ： はい。

```
~/bin% dc （ぽこ）
```

root ： そこで [Triton改] 1 :a して。

yasuoka ： [Triton改] 1 :a ですね。

```
[Triton （ぽこ）
] 1 :a （ぽこ）
```

root ： さらに [Nereid改] 2 :a して。

yasuoka ： [Nereid改] 2 :aっと。

```
[Nereid （ぽこ）
] 2 :a （ぽこ）
```

他もするんですか？

root ： いや、それは今はいいから、次に 1 ;a して、さらに P してごらん。

yasuoka ： 1 ;a P ですね。

```
1 ;a P （ぽこ）
Triton
```

あ、そういうことですか。すると

```
2 ;a P（ぽこ）
Nereid
```

やっぱり。じゃあ

```
[Naiad（ぽこ）
] 3 :a [Thalassa（ぽこ）
] 4 :a [Despoena（ぽこ）
] 5 :a [Galatea（ぽこ）
] 6 :a [Larissa（ぽこ）
] 7 :a [Proteus（ぽこ）
] 8 :a（ぽこ）
```

としておけば

```
4 ;a P（ぽこ）
Thalassa
```

となるわけですね。

root : そういうこと。さらにすごいのは、配列をそのままコピーできることだ。la sb してごらん。

yasuoka : la sb ですか？

```
la sb（ぽこ）
```

これでどうなったんですか？

root : ためしに変数 b の 4 番目の要素を読み出して、表示してごらん。

yasuoka : 4 ;b P ですね。

```
4 ;b P（ぽこ）
Thalassa
```

あれ？

```
4 ;a P（ぽこ）
Thalassa
```

コピーされてる。どういうことですか？

root : 変数 a が配列として使われてる時は、 la は配列全体を読み出してスタックに積むんだよ。それを sb で変数 b に代入したんだ。

yasuoka : ふーん、なかなか。

root : ただし、配列のコピーにはバグがあるみたいで、コピーした後にどっちかの要素に代入すると、動作がおかしくなることがある。

yasuoka : じゃあ、あんまり使えませんね。

```
q（ぽこ）
~/bin%
```

root : そういうことだ。

（間）

yasuoka : 出来ました。

```
~/bin% cat factorial（ぽこ）
[ [ ] ; exec dc $0 #] c
[0] 0 :p [1] 1 :p [2] 2 :p [3] 3 :p [4] 4 :p
[5] 5 :p [6] 6 :p [7] 7 :p [8] 8 :p [9] 9 :p
[la 10 * 10 lb ^ / 10 % ;p P lb 1 - d sb 0 <P] sP
[la * la 1 - d sa 1 <!] s!
[[number? ] P ? sa la Z sb lP x [!=] P
1 la 1 <! p s. d x] d x q
~/bin% factorial（ぽこ）
number? 25（ぽこ）
25!=15511210043330985984000000
number? 15（ぽこ）
15!=1307674368000
number? 0（ぽこ）
0!=1
number? q（ぽこ）
~/bin%
```

こんなもんですか？

root : これで正解だけど、変数 P の演算が複雑過ぎるから、できればこう変更したいところだな。

```
[ [ ] ; exec dc $0 #] c
[0] 0 :p [1] 1 :p [2] 2 :p [3] 3 :p [4] 4 :p
[5] 5 :p [6] 6 :p [7] 7 :p [8] 8 :p [9] 9 :p
[d SA 10 / d 0 <P LA 10 % ;p P] sP
[la * la 1 - d sa 1 <!] s!
[[number? ] P ? d sa lP x s. [!=] P
1 la 1 <! p s. d x] d x q
```

root : Z も使わなくてすむしね。

yasuoka : でも代わりに、 SA とか LA とかがあるじゃないですか。これ何ですか？

S変数
　　スタックの一番上の値を取り出し、変数に積む。
L変数
　　変数の一番上の値を取り出し、スタックに積む。
いずれも変数はスタックであるとみなされる。

root : dc では変数は、普通の変数とみなされたり、配列とみなされたり、スタックとみなされたりするんだよ。普通の変数とみなすための命令が s と l、配列とみなすための命令が: と;、そしてスタックとみなすための命令が S と L なんだ。

yasuoka : そうなんですか。

root : で、 SA は、スタックの一番上の値を取り出してスタック変数 A に積む、っていう命令だし、 LA は、スタック変数 A の一番上の値を取り出してスタックに積む、っていう命令だ。そうだな、実際に 6 行目の? d sa lP x がどう動くか説明した方が、わかりやすいかな。

yasuoka : お願いします。

root : 例えば、入力された値が 20 だったとすると、この d sa lP でスタックは

```
d SA 10 / d 0 <P LA 10 % ;p P
20
[number? ] P ? d sa lP x s. [!=] P改1 la 1 <! p s. d x
```

になって、次の x で

```
20
[number? ] P ? d sa lP x s. [!=] P改1 la 1 <! p s. d x
```

になって、 d SA 10 / d 0 <P LA 10 % ;p P の実行が始まる。

yasuoka : はい、そこまではわかります。

root : 最初の d でスタックは

```
20
20
[number? ] P ? d sa lP x s. [!=] P改1 la 1 <! p s. d x
```

になって、次の SA でスタックは

```
20
[number? ] P ? d sa lP x s. [!=] P改1 la 1 <! p s. d x
```

になると同時に、スタック変数 A に

```
20
```

が積まれる。それから 10 / d 0 でスタックは

```
0
2
2
[number? ] P ? d sa lP x s. [!=] P改1 la 1 <! p s. d x
```

になって、次の <P が成立するから、スタックは

```
2
[number? ] P ? d sa lP x s. [!=] P改1 la 1 <! p s. d x
```

になって、変数 P に入ってる文字列 d SA 10 / d 0 <P LA 10 % ;p P の実行が始まる。

yasuoka : はい。

root : 再び d SA で、スタックは

```
2
[number? ] P ? d sa lP x s. [!=] P改1 la 1 <! p s. d x
```

になって、変数 A に

```
2
20
```

が積まれる。ここがミソだ。

yasuoka : ミソって？

root : 続けようか。 10 / d 0 でスタックは

```
0
0
0
[number? ] P ? d sa lP x s. [!=] P改1 la 1 <! p s. d x
```

になるから、今度は <P は成立しなくて、スタックは

```
0
[number? ] P ? d sa lP x s. [!=] P改1 la 1 <! p s. d x
```

になる。次の LA で、スタック変数 A の一番上の 2 が取り出されるから、スタックは

```
2
0
[number? ] P ? d sa lP x s. [!=] P改1 la 1 <! p s. d x
```

になって、変数 A は

```
20
```

になる。その後は 10 % ;p で文字列に変換されて、スタックは

```
2
0
[number? ] P ? d sa lP x s. [!=] P改1 la 1 <! p s. d x
```

になり、 P で 2 が表示されて、スタックは

```
0
[number? ] P ? d sa lP x s. [!=] P改1 la 1 <! p s. d x
```

になる。

yasuoka : それからどうなるんですか？

root : 文字列の実行が終了するから、 <P の直後に戻る。 LA でスタック変数 A は空になって、スタックは

```
20
0
[number? ] P ? d sa lP x s. [!=] P改1 la 1 <! p s. d x
```

になり、 10 % ;p P で 0 が表示されて、スタックは

```
0
[number? ] P ? d sa lP x s. [!=] P改1 la 1 <! p s. d x
```

で、 6 行目の lP x の直後に戻る。次の s. でスタックは

```
[number? ] P ? d sa lP x s. [!=] P改1 la 1 <! p s. d x
```

になるから、後は元の通り。

yasuoka : うーん、そういう仕掛けなんですか。よくわかりました。スタック変数を l で読み出すと、スタック変数全体が読み出されるんですか？

root : いや、そうはならなくって、スタック変数の一番上の値が読み出される。 L との違いは、スタック変数自身は変化しないことだ。

yasuoka : じゃあ s は？

root : スタック変数に s すると、一番上の値に代入される。例えば、スタック変数 A が

```
2
1
```

の時に 8 sA を実行すると、変数 A は

```
8
1
```

になる。

yasuoka : 要するに s と l は、スタック変数の一番上を普通の変数だと思ってアクセスするんですね。

root : そういうこと。スタック変数に配列を積む、っていう風な使い方も実は可能だ。どう使うかはちょっと思いつかないけどね。

yasuoka : ふーん。そういえば、 dc ではコメントは書けないんですか？

root : 書けない。どうしてもコメントみたいなものを書きたい時は

```
[ "factorial" Version 3.2 ] s.
```

とかいう風に、意味のないコマンドを書くしかないな。実行がちょっと遅くなるけど。さて、もうこんな時間だ。 dc についてはここまでにしようか。もう dc のスクリプトが書けるようになっただろう？

yasuoka : まだまだだと思いますけど、これから頑張ってみます。どうもありがとうございました。