本稿は Linux Japan 誌 2000 年 10 月号に掲載された記事に補筆修正したものです.

Linux で電卓

コンピュータはもともと電子計算機ですから，計算は大得意のはずです．前回 awk を高級プログラマブル電卓として使ってみましたが，研究費の計算などには，プログラマブルでなく加減乗除ができるだけで十分です．しかも，ヨドバシカメラの店員さんが持っているような，キーも数字が大きいのが使いやすいですね．今回は，電卓をテーマに探検してみましょう．

いつもならば CUI から話しをするのですが，それらはプログラミング言語に近いので，むしろ電卓としては低級な X11 上のものを先に取り上げます．

xcalc

xcalc は標準配布の X クライアントで Athena Widget を使っています．この際，見栄えを良くする (ボタンなどの 3D 化) ためにライブラリを libXaw3d に変えてしまいましょう．次のように，libXaw.so.6.1 を libXaw3d.so.6.1 からのリンクにすれば良いだけのことです．すなわち，

```
# cd /usr/X11R6/lib/
# mv libXaw.so.6.1 hidden.libXaw.so.6.1
# ln -fs libXaw3d.so.6.1 libXaw.so.6.1
# /sbin/ldconfig
```

として X を起動しなおせば，Athena Widget をリンクした X クライアントが 3D 化されます (変わらない部品もあります).

オプション -rpn

科学技術用計算機 xcalc を何もオプションを付けないで起動しますと，図 1 のようになります．使い方は実在の電卓と同じです．つまり 2 + 3 を計算するには

[2] [+] [3] [=]

と順にマウスでキー入力すればよいのです．ところで，このキー入力に「待てよ」と逡巡した貴方は不惑を過ぎている方に違いないです．何をいってるんだと思う人は

```
$ xcalc -rpn
```

と起動してみてください，図 2のような横長の電卓が現れます.

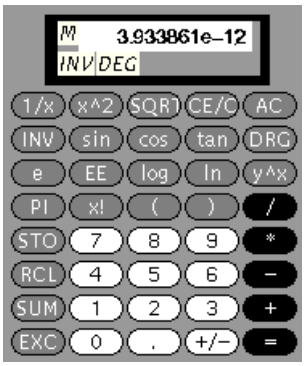

図 1　オプションなしで起動した xcalc.

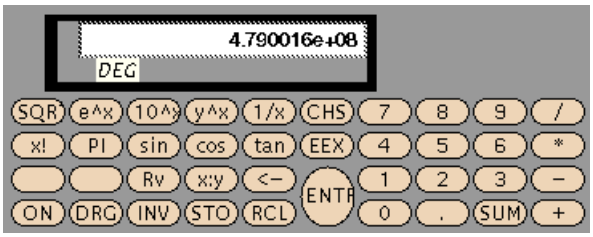

図 2　オプション -prn を付けてで起動した xcalc.

この電卓では 2 + 3 を計算するには

[2] [CR] [3] [+]

とします．この記法は逆ポーランド法と呼ばれるもので，大変効率の良いアルゴリズムとなっています．2+3 では，判然としませんが，6 ∗ (2 + 3) を計算する場合のキー入力をを両者で比較してみると，効率の良さがわかるでしょう．普通の記法では

[6] [*] [(] [2] [+] [3] [)] [=]

と括弧を使わざるを得ませんが，逆ポーランド法では

[6] [CR] [2] [CR] [3] [+] [*]

で済みます．常に先頭スタックの内容が表示されており，2 項演算の結果がすぐに先頭スタックに入りますから，[=] が必要ありません．なによりも，括弧を使わないので，優先順に従った 2 項演算を実行させながら計算をすすめるように記す必要があり，結果的には計算機にとって効率が良くなるのです．メモリが高価だった頃，高級プログラム電卓においても，この人間にとっては不自然な記法が，一次的な記憶場所を少なくすることができるという理由で採用されていたのです.

PostScript 言語は逆ポーランド法に類する記法を採用しています．ghostscript で確かめてみましょう.

```
$ gs -DNODISPLAY -q
GS> 6 2 3 add mul ==
30
GS>quit
```

CR が要りませんから，効率の良さが非常にはっきりとしますね．しかし，この言語を直接書こうという気はあまり起こらないでしょう．

カスタマイズで大きな数字

いきなり脱線してしまいました．xcalc 自身の話しに戻りましょう．マウスで画面のボタンを押すには疲れますね，画面上のボタンが小さい場合はなおさらです．実は，画面のボタンしか使えないのではなく，キーボードから入力することができます (関数もほぼ頭文字で命令できます．sin なら s，tan なら t，arcsin すなわち INV-sin なら i,s といった調子)．したがって，画面上のボタンは小さくても構いません．問題は，標準のリソース指定 (/usr/lib/X11/app-defaults/XCalc) では表示用のフォントが小さいという点でしょう．そこで，大きさが可変のフォント (Type1 や TrueType) を利用して大きな数字が表示されるようにカスタマイズしましょう．フォントの扱いは各配布系で非常に異なります．ここでは，2000 年 5 月号の本連載記事を参考にして，gs の Type1 が使えるよう設定したと仮定します．すると helvetica が任意の大きさで使えるようになっているはずです (xfontsel で確認してください)．そこで，

```
big.ti.background: OliveDrab
big*bevel.background: orange2
big*bevel.horizDistance: 4
big*bevel.screen.vertDistance: 4
big*bevel.screen.*.Font: \
  -*-helvetica-medium-r-normal--12--*-*-*-*-*-*
big*bevel.screen.*.background: gray10
big*bevel.screen.*.foreground: aquamarine
big*bevel.screen.*.horizDistance: 4
big*bevel.screen.*.shadowWidth: 0
big*bevel.screen.LCD.horizDistance: 0
big*bevel.screen.LCD.Font: \
  -*-helvetica-bold-r-normal--36--*-*-*-*-*-*
big*bevel.screen.LCD.width: 236
big*bevel.screen.M.vertDistance: 46
big*bevel.screen.INV.fromHoriz: M
big*bevel.screen.INV.fromVert: LCD
big*bevel.screen.INV.vertDistance: 0

big*Font: \
  -*-helvetica-bold-r-normal--14--*-*-*-*-*-*
big*ShapeStyle: rectangle
big*Command.height: 24
big*Command.width: 51
big*Command.shadowWidth: 3
big*button22.background: ivory
big*button23.background: ivory
big*button24.background: ivory
big*button27.background: ivory
big*button28.background: ivory
big*button29.background: ivory
big*button32.background: ivory
big*button33.background: ivory
big*button34.background: ivory
big*button37.background: ivory
big*button38.background: ivory
big*button39.background: ivory
big*button20.background: gray30
big*button25.background: gray30
big*button30.background: gray30
big*button35.background: gray30
big*button40.background: gray30

big*hp.background: OliveDrab
big*hp*Command.background: gray60
big*hp*Command.foreground: gold
```

のように big という名前の X クライアントに関する記述を~/.Xdefaults に追加しますと，

```
$ xcalc -name big
```

で，図 3 のように，大きな電卓が現れます．libXaw3d を使うように変えていれば，shapeStyle を rectangle に指定するとボタンが立体化され，だいぶ見栄えも向上します．また，逆ポーランド法電卓の方もカスタマ

図 3　大きなフォントを使うようにカスタマイズした xcalc. libXaw3d を使うとボタンが立体的になる．

イズの結果を眺めてみてください．

```
$ xcalc -rpn -name big
```

bc

続いて CUI のものを紹介します．お金の計算や，単発の関数計算などは xcalc で十分ですが，繰り返しや

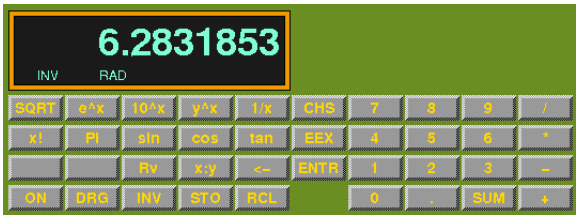

図 4　カスタマイズされた逆ポーランド電卓.

入力を促すプログラムはできません．このようなような高級で複雑な計算はやはりプログラミング言語が必要となります．前回の awk もいいのですが，ここでは計算に特化したものを紹介しましょう．ずばり GNU の bc[1]W[3] です．現在の開発担当は Philip A. Nelson(phil@cs.wwu.edu) 氏となっています．

言語の概要

手始めに対話的に動かしてみましょう．オプション `-l` で数学関数を有効にし，`-q` で起動時の挨拶を抑止します．

```
$ bc -lq
6*(2+3) CR
30
s(1) CR
.84147098480789650665
s(1)^2+c(1)^2 CR
.99999999999999999999
quit CR
$
```

計算式を入力 (CR) すると，結果が表示されるといった，そっけないユーザーインターフェースがいかにもです．表 1 に組み込み関数を示します．なぜか n 次ベッセル関数が入っているところがマニアックですね．

表 1　bc の組み込み関数

`e(x)`	指数
`s(x)`	sine
`c(x)`	cosine
`a(x)`	arctangent
`l(x)`	自然対数
`j(n,x)`	n 次ベッセル
`sqrt(x)`	平方根
`scale(x)`	小数点以下の有効桁数
`length(x)`	全有効桁数
`read()`	標準入力からの読み込み

繰り返し計算

予め計算式をプログラムしておいて，実行時にはデータを入力するだけという使い方は，実験のデータをその場で確かめる場合起こる状況です．例えば，ピタゴラスの定理を使って，平面上の距離 r を式

$$r = \sqrt{x^2 + y^2}$$

により x, y 成分の測定結果から求める場合を考えます．予めスクリプトを書いて実行させましょう．大体，次のような感じになると思います．リテラル文字の比較ができないので，“文字 q のキー入力で終了する” というような書き方ができません．わざわざユーザー関数を定義してみました．戻り値が式の場合には必ず括弧でくくる必要があります．

```
define hypot(x,y) {
  return (sqrt(x^2+y^2));
}

cont = 1
while (1) {
  print "x = ? ";
  x = read ();
  print "y = ? ";
  y = read ();
  print "r = "; hypot(x,y);
  print "\n 継続 ? (yes=1/no=0)";
  cont = read();
  if (cont == 0) break;
}
quit
```

hypot.bc とでも名付けて保存してください．実行例は

```
$ bc -lq hypot.bc
x = ? 3
y = ? 4
5.00000000000000000000

継続 ? (Yes=1/No=0)0
$
```

のようになります．ところで，有効桁数が多いことに気づきましたか？ C 言語の倍精度が有効数字 15 桁であるにもかかわらず bc ではデフォルトで 20 桁もあります．驚くことはありません bc は任意精度計算言語なのですから．小数点以下の有効数字は scale で定義できます．関係式 $\pi = 4 * \arctan(1)$ を使って 200 桁の π を求めてみましょう．一行当たりの表示文字数は環境変数 `BC_LINE_LENGTH` で指定します．

```
$ bc -lq
4*a(1) CR
3.14159265358979323844
scale = 200 CR
4*a(1) CR
3.14159265358979323846264338327950288419\
71693993751058209749445923078164062862 08\
99862803482534211706798214808651328230 66\
47093844609550582231725359408128481117 45\
02841027019385211055596446229489549303 81\
96
```

シェルからの呼び出し

bash 自身が扱える数値は整数だけですが,

```
$ echo $[6*(2+3)]
30
$ echo $[2-32000]
-31998
```

のように算術式を評価させることもある程度できます. しかし，実数を与えると syntax error となりますから科学技術計算には全く不向きです．計算機ツールを対話的に使わず，パイプで値を受渡しする工夫をしてみましょう．実数演算の結果を得るのは，適当なツールなどにコマンド文字列をパイプで流し込むことで可能となります．ツールは awk でも gnuplot でも良いのですが,「print」コマンドを記述しなくて済むので，bc が最も簡素です．2 の平方根を計算する場合を例として示しますと,

```
$ echo 'sqrt(2)' |bc -l
1.41421356237309504880
$ echo 'print sqrt(2)' |gnuplot
1.4142135623731
$ awk 'BEGIN{print sqrt(2)}'
1.41421
```

となり，やはり bc がすっきりしていていいです．もっとキー入力を少なくしたいならば，実用的かどうか判りませんが,

```
function sqrt {
  echo "sqrt($1)" |bc -l
}

function mul {
  echo "$1 * $2" |bc -l
}
```

のようにシェルのユーザー関数を定義すれば，コマンドライン上であたかも直接計算が可能のようにみせかけることができます.

```
$ sqrt 2
1.41421356237309504880
$ mul 8457.6 3.1415
26569.55040
```

あるいは，コマンドライン専用の計算機 upn なども慣れれば便利かもしれません．ただし，記法は演算子が被演算子の後ろに置かれる (後置法) タイプです.

```
$ upn 2 3 div
      0.666666666667
```

条件分岐命令が組み込まれていたりして，楽しめそうです．作者は Jörg Weule (weule@ac.org) 氏であるとマニュアルにあります．Sunsite の pub/Linux/apps/math/calc/ からソースを入手できます.

dc

GNU の bc のソースには dc という逆ぽーランド法の任意精度計算機が含まれています（数学関数は組み込まれていません）．例えば $6*(2+3)$ を計算させるには,

```
$ dc
6 2 3 + * p
30
q
$
```

とします．dc には結果を表示する 「print」コマンド p があります．なぜなら，bc の場合と違って改行は単なる区切りにすぎないからです．この例では任意精度がはっきりしませんから，小数点以下の桁を設定するコマンド k を使ってみましょう.

```
$ dc
30 k
123 63 /
p
1.95238095238095238095238095238 0
```

うーん，やはり辛いかも.

複素数

文系では学校の数学の授業以外にお目にかかることもない複素数ですが，理系では，まれに計算しなければならない場合があります．しかしこの計算はなかなかの難物でして，プログラミングしようにも C 言語では自分でライブラリを書く (教科書的な問題として良く取り上げられますね) か，標準的に扱うことができる C++ を使うしか手がありません．したがって，複素数の計算機は結構ありがたいです．もっともあまり

種類はありません．X なら xrpn という逆ポーランド記法のもの，CUI では... 簡単なものがありません (総合数学ツールになってしまう)．強いて使うとすれば，gnuplot でしょうか (^^;

xrpn[2] W3

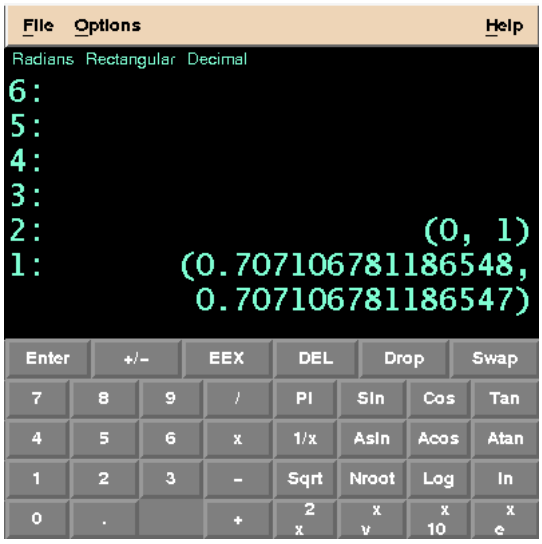

図 5　複素数も扱える逆ポーランド電卓.

作者は Paul Wilkins(paul.wilkins@analog.com) 氏です．Motif または Lesstif が必要ですが，ライブラリがある場合には，Sunsite から入手したソースを展開して，make 一発でした．複素数をどのように入力するかだけ説明しましょう．例えば，純虚数 $i = (0,1)$ を生成するには，0 と 1 を入力して (stack 1 が 1，statck 2 が 0 となるはず)，cplx コマンドを入力します．stack 1 に $(0,1)$ と表示されることを確かめてください．dup コマンドで statck 2 に $(0,1)$ を複製したあと，sqrt を実行した結果を図 5 に示します．最終桁の数値が違っているのはご愛敬．同一作者によって gtk で動くようにした grpn [3]W3もあります．

gnuplot

筆者愛用の Gnuplot は複素数の演算もできます．ただし括弧が変わっていて $\{0,1\}$ のように表します．純虚数 i の平方根を求めてみましょう．

```
$ echo 'print sqrt({0,1})' |gnuplot
{0.707106781186548, 0.707106781186547}
```

最終桁がやはり違ってしまいました，クワバラクワバラ．さて問題です，$\sqrt{n+i}$ $(n = 0,1,...,50)$ を計算するにはどうしたらよいでしょう．これまでの流れから当然 gnuplot を呼び出すシェルスクリプトです．

```
#!/bin/sh
for n in `seq 1 50`; do
  echo -n "sqrt{$n,1} = " ;
  echo "print sqrt({$n,1})" |gnuplot ;
done
```

整数列を発生するコマンド seq が効いてますね．これを知らないと，ちょっと面倒です．seq の詳細は “`seq --help`” してください．

参考文献

[1] GNU bc の公式ページ．W3
`http://www.gnu.org/manual/bc`

[2] Xrpn は ibiblio にあります．W3
`http://http://ibiblio.org/pub/Linux/apps/math/calc/`

[3] Grpn の公開ページ．W3
`http://lashwhip.com/grpn.html`