Rosetta Pearls

Walter Pachl 3 March 2024

There is a huge collection of programs written in many languages

Available in https://rosettacode.org/wiki/Rosetta_Code

There one can find many REXX programs written by the late Gerard Schildberger with whom I had many “discussions” regarding the

upward compatibility when going from “classic” REXX to ooRexx.

His programs use several features of REXX no longer supported by ooRexx such as variable names like $, @, and #, assignments of the

Form x=;, the Upper instruction, etc. The formatting of his programs is sometimes also strange if not appalling.

It is unfortunate that a newcomer interested in REXX or a professional trying to reuse such a program are confronted with this (mess).

In the following I shall present a few of these programs pointing at the problems I see and ways to improve them.

You can start digging for pearls with this link

https://rosettacode.org/wiki/Category:Programming_Tasks

which shows

Jump: 1 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Pages in category "Programming Tasks"

The following 200 pages are in this category, out of 1,266 total.

(previous page) (next page)

1

- 100 doors
- 100 prisoners
- 15 puzzle game

- Bifid cipher
- Bin given limits
- Binary digits
- Binary search
- Binary strings

Let us start with a finger exercise.

FASTA format

You are encouraged to solve this task according to the task description, using any language you may know.

In bioinformatics, long character strings are often encoded in a format called FASTA.

A FASTA file can contain several strings, each identified by a name marked by a > (greater than) character at the beginning of the line.

Gerard's REXX program is

```
/*REXX program reads a (bio-informational)  FASTA  file  and  displays the contents.    */
parse arg iFID .                                 /*iFID:  the input file to be read.    */
if iFID==''  then iFID='FASTA.IN'                /*Not specified?  Then use the default.*/
name=                                            /*the name of an output file (so far). */
$=                                               /*the value of the output file's stuff.*/
     do  while  lines(iFID)\==0                  /*process the  FASTA  file contents.   */
     x=strip( linein(iFID), 'T')                 /*read a line (a record) from the file,*/
                                                 /*--------- and strip trailing blanks. */
     if left(x, 1)=='>'  then do
                              if $\==''  then say name':'  $
                              name=substr(x,  2)
                              $=
                              end
                         else $=$ || x
     end   /*j*/                                 /* [?]  show output of last file used. */
if $\==''  then say name':'  $                    /*stick a fork in it,  we're all done. */
```

Replacing the $ with d and adding '' to the assignment $=, which is now d='' lets me run this program with ooRexx.

The input file

```
>Rosetta_Example_1
THERECANBENOSPACE
>Rosetta_Example_2
THERECANBESEVERAL
LINESBUTTHEYALLMUST
BECONCATENATED
```

Shows this expected output

```
Rosetta_Example_1: THERECANBENOSPACE
Rosetta_Example_2: THERECANBESEVERALLINESBUTTHEYALLMUSTBECONCATENATED
```

Notice the very strange indentations!

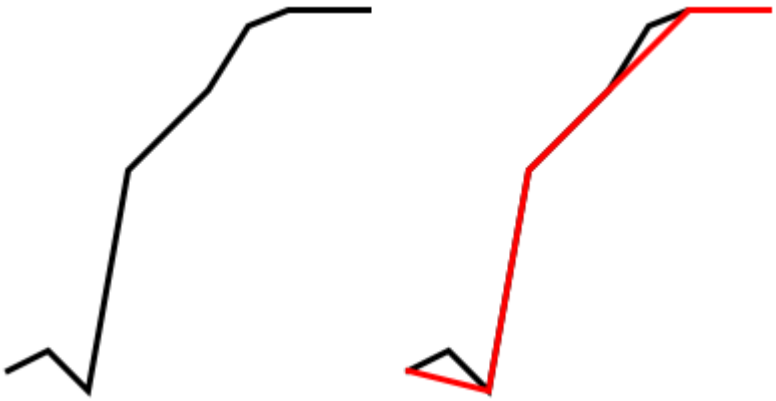
Ramer-Douglas-Peucker (RDP) line simplification algorithm

The **Ramer–Douglas–Peucker** algorithm is a line simplification algorithm for reducing the number of points used to define its shape.

```
/*REXX program uses the  Ramer-Douglas-Peucker (RDP)  line simplification algorithm  for*/
/*---------------------------- reducing the number of points used to define its shape. */
parse arg epsilon pts                           /*obtain optional arguments from the CL*/
if epsilon='' | epsilon="," then epsilon= 1     /*Not specified?  Then use the default.*/
if pts=''  then pts= '(0,0) (1,0.1) (2,-0.1) (3,5) (4,6) (5,7) (6,8.1) (7,9) (8,9) (9,9)'
pts= space(pts)                                 /*elide all superfluous blanks.        */
           say '  error threshold: '   epsilon  /*echo the error threshold to the term.*/
           say ' points specified: '   pts      /*  "   "    shape points   "  "    "  */
$= RDP(pts)                                     /*invoke Ramer-Douglas-Peucker function*/
           say 'points simplified: '   rez($)   /*display points with () ---? terminal.*/
exit 0                                          /*stick a fork in it,  we're all done. */
/*-------------------------------------------------------------------------------------*/
bld: parse arg _; #= words(_); dMax=-#; idx=1; do j=1  for #; @.j= word(_, j); end; return
px:  parse arg _;          return word( translate(_, , ','),  1)   /*obtain the X coörd.*/
py:  parse arg _;          return word( translate(_, , ','),  2)   /*   "    "  Y   "   */
reb: parse arg a,b,,_;                  do k=a  to b;  _= _ @.k;    end;   return strip(_)
rez: parse arg z,_;   do k=1  for words(z); _= _ '('word(z, k)") "; end;   return strip(_)
/*-------------------------------------------------------------------------------------*/
RDP: procedure expose epsilon;    call bld  space( translate(arg(1), , ')(][}{') )
     L= px(@.#) - px(@.1)
     H= py(@.#) - py(@.1)                       /* [?] find point IDX with max distance*/
                        do i=2  to #-1
                        d= abs(H*px(@.i) - L*py(@.i) + px(@.#)*py(@.1) - py(@.#)*px(@.1))
                        if d>dMax  then do;   idx= i;   dMax= d
                                        end
                        end   /*i*/            /* [?]  D is the perpendicular distance*/

     if dMax>epsilon  then do;   r= RDP( reb(1, idx) )
                                 return subword(r, 1, words(r) - 1)     RDP( reb(idx, #) )
                           end
     return @.1  @.#
```

```
dlp: Procedure Expose epsilon
  Parse Arg pl                  /* list of points as string          */
  plc=pl
  Do i=1 By 1 While plc<>''     /* list of points                    */
    End
  end=i-1                       /* points 2 to end must be looked at */
  dmax=0                        /* initialize maximum distance       */
  index=0
  Do i=2 To end-1
    d=distpg(p.i,p.1,p.end)     /* distance of p.i from the base line*/
    If d>dmax Then Do           /* largest one so far                */
      index=i                   /* p.index is candidate for survival */
      dmax=d
      End
    End
  If dmax>epsilon Then Do       /* significant distance              */
    rla=dlp(subword(pl,1,index))
    rlb=dlp(subword(pl,index,end))
    rl=subword(rla,1,words(rla)-1) rlb
    End
  Else                          /* no point with d>epsilon           */
    rl=word(pl,1) word(pl,end)
  Return rl
```



As part of the presentation I wanted to show how the RDP algorithm works on regular polygon with 24 corners.
Alas I couldn't get the DOS screen to the attendants, so here is what you would have seen.

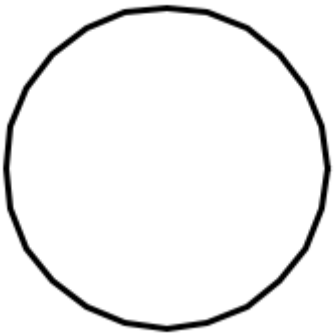
The radius of the polygon is 60
Rdp24_m 1 is the original line with epsilon=m
Rdp24_m 2 is the simplified line

Below are the pictures produced by this process:
Compute the corners' coordinates
Invoke rdp to get the simplified pronz´t lisz
Produce an input file fpr ma draw progra,
Invoke rexx draw with that file

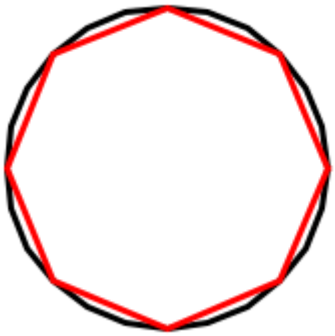
Draw a black (S=schwarz) polygon
Show the picture, save it, and sleep fpr 4 seconds
Draw a red polygon

Y S 80.0 0.0 77.3 20.7 69.3 40.0 56.6 56.6 ...
SS 3
Y R 80.0 0.0 56.6 56.6 0.0 80.0 -56.6 56.6 ...

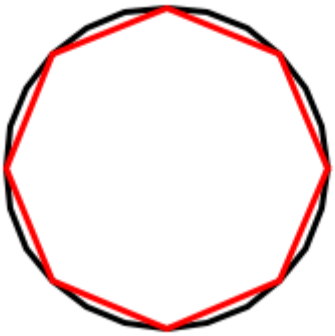
rdp24_6 1



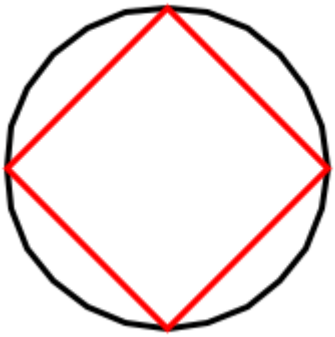
rdp24_6 2



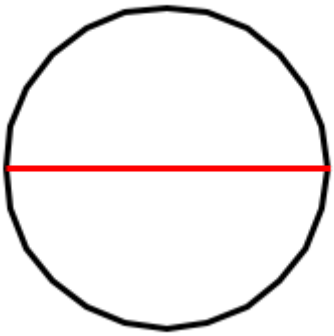
rdp24_12 2



rdp24_30 2



rdp24_80 2



Permutations

==={{header|REXX}}==

This program could be simplified quite a bit if the "things" were just restricted to numbers (numerals), but that would make it specific to numbers and not "things" or objects.

Formatted program looks like a landscape

```
.permSet: procedure expose $. @. between x y;       parse arg ?
          if ?>y  then do;  _= @.1;      do j=2  for y-1
                                         _= _  ||  between  ||  @.j
                                         end   /*j*/
                            say _
                       end
                  else do q=1  for x             /*build the  permutation  recursively. */
                         do k=1 for ?-1;  if @.k==$.q  then iterate q
                         end   /*k*/
                       @.?= $.q;         call .permSet ?+1
                       end     /*q*/
```

Running this program works fine with Regina

```
K:\_B\P>regina permrc
123
132
213
231
312
321
But fails with ooRexx
K:\_B\P>rexx permrc
    15 *-*            @
Error 13 running K:\_B\P\permrc.rex line 15:  Invalid character in program.
Error 13.1:  Incorrect character in program "@" ('40'X).
```

Other problems are
The use of $ as variable name
Omission of the expression in assignments @.=; sep=
Use of the Upper instruction upper @abcU

These 2 lines
@abc = 'abcdefghijklmnopqrstuvwxyz'; @abcU=@abc; upper @abcU
@abcS = @abcU || @abc; @0abcS= 123456789 || @abcS

Can easily be simplified to

xabc = 'abcdefghijklmnopqrstuvwxyz'
xabcS = 123456789 || translate(xabc)||xabc;

Two years ago
(25 January 2022? Walterpachl talk contribs? m 212,777 bytes +2,632? add ooRexx)
I rewrote this program and stored it under ooRexx on rosetta,rc (although it uses no oo feature)
I omitted the feature to separate the elements by a specified string
But added a help function invoked by using a question mark as argument,
Of course, this program uses better variable names.

```
K:\_B\P>rexx perm ?
rexx perm            -> Permutations of 1 2 3
rexx perm 2          -> Permutations of 1 2
rexx perm 2 4        -> Permutations of 1 2 3 4 in 2 positions
rexx perm 2 a b c d  -> Permutations of a b c d in 2 positions
```

By default my things are the digits 1 through 9 followed by the letters A through F followed by the numbers 16, 17, etc, as needed.
Rexx perm 4 14 > p4_14.txt results in this file of 24026 lines:

1 2 3 4
1 2 3 5
…
E D C A
E D C B
24024 Permutations
0.399000 seconds

I'll come back to this later,

Prime Conspiracy

Prime numbers, it seems, have decided preferences about the final digits of the primes that immediately follow them.

This conspiracy among prime numbers seems, at first glance, to violate a longstanding assumption in number theory: that prime numbers behave much like random numbers.
(original authors from Stanford University): Kannan Soundararajan and Robert Lemke Oliver

The REXX program uses the Sieve of Eratosthenes to mark the composite numbers
And looks then for the successor of each prime.
Its output starts as follows:

```
For  1000000  primes used in this study:
digit  1 --? 1  has a count of:   42853,  frequency of:   4.2853%.
digit  1 --? 3  has a count of:   77475,  frequency of:   7.7475%.
digit  1 --? 7  has a count of:   79453,  frequency of:   7.9453%.
digit  1 --? 9  has a count of:   50153,  frequency of:   5.0153%.

digit  2 --? 3  has a count of:       1,  frequency of:   0.0001%.

etc.
```

I wrote a little program to show this as a matrix;

Analysis of 1000000 prime numbers shows

```
i followed by 1       3       5       7       9
1             4.2853% 7.7475%         7.9453% 5.0153%
2                     0.0001%
3             5.8255% 3.9668% 0.0001% 7.2828% 7.9358%
5                                     0.0001%
7             6.4230% 6.8595%         3.9603% 7.7586%
9             8.4596% 6.4371%         5.8130% 4.2843%
```

Then I tried to run the program for 2000000 prime numbers

```
i followed by 1        3        5        7        9
1             4.2920%  7.8383%           7.7701%  5.0285%
2                      0.0001%
3             5.7801%  4.0007%  0.2619%  7.1285%  7.7660%
5                                0.2619%
7             6.3258%  6.7724%           3.9913%  7.8519%
9             8.5310%  6.3258%           5.7896%  4.2846%
```

And there are suddenly many primes ending with 5!!
The problem is in the estimate for the upper limit used for the sieve of Eratosthenes

H= N* (2**max(4, (w%2+1))) /*used as a rough limit for the sieve. */

np=2000001
i=32052371 largest prime
h=32000000

What would be a better / correct estimate for H?

Happy Numbers

A happy number is defined by the following process:
Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals **1** (where it will stay) or it loops endlessly in a cycle which does not include **1**.

The REXX solution is again strangely formatted and uses @. as variable name. Replacing @ by, say, a
suffices to run his program, with ooRexx.

```
/*REXX program  computes  and  displays  a  specified  amount  of   happy   numbers.    */
parse arg limit .                                  /*obtain optional argument from the CL.*/
if limit=='' | limit=="," then limit=8             /*Not specified?  Then use the default.*/
haps=0                                             /*count of the happy numbers  (so far).*/

  do n=1  while haps<limit;    @.=0;  q=n          /*search the integers starting at unity*/
        do  until q==1                             /*determine if   Q   is a happy number.*/
        s=0                                        /*prepare to add squares of digits.    */
                do j=1  for length(q)              /*sum the squares of the decimal digits*/
                s=s + substr(q, j, 1) **2          /*add the square  of  a  decimal digit.*/
                end   /*j*/

        if @.s  then iterate n                     /*if already summed,   Q   is unhappy. */
        @.s=1;  q=s                                /*mark the sum as found;   try  Q  sum.*/
        end   /*until*/
  say n                                            /*display the number    (N  is happy). */
  haps=haps+1                                      /*bump the  count  of  happy numbers.  */
  end          /*n*/
                                                   /*stick a fork in it,  we're all done. */
```

```
K:\_B\HN>rexx hnrca
     6 *-*     @
Error 13 running K:\_B\HN\hnrca.rex line 6:  Invalid character in program.
Error 13.1:  Incorrect character in program "@" ('40'X).
```

Gerard added a second program that allows to show a range of happy numbers.
It starts off with a cryptic argument analysis-

```
/*REXX program computes and displays a specified range of happy numbers. */
parse arg L H .                                    /*obtain optional arguments from the CL*/
if L=='' | L=="," then L=8                         /*Not specified?  Then use the default.*/
if H=='' | H=="," then do;  H=L; L=1;  end         /*use a range for the displaying of #s.*/
            do i=0  to 9;  #.i=i**2;  end /*i*/   /*build a squared decimal digit table. */
@.=0;   @.1=1;       !.=@.;    !.2=1;    !.4=1   /*sparse array:   @=happy,  !=unhappy. */
haps=0                                             /*count of the happy numbers  (so far).*/
```

After the ususal replacements (@->a, $->d, etc.) the program loops because of this little incompatibility: @.=0; @.1=1; !.=@.; is not the same as !.=0, because ooRexx copies the entite stem.and makes 1 an unhappy number.

My version of argument analysis, using reasonable variable names, would be

```
Parse Arg low high                        /* obtain range of happy numbers       */
If low='?' Then Call help
If high='' Then
  Parse Value 1 low With low high
…
help:
  Say 'rexx hno n compute and show the first n happy numbers'
  Say 'rexx hno low high      show happy numbers from index low to high'
  Exit
```

```
K:\_B\HN>rexx hno 8
1 7 10 13 19 23 28 31
31 0.001000

K:\_B\HN>rexx hno 1000 1002
6899 6904 6917
6917 0.020000
```

For this task there is also a program in the ooRexx category created by Rick McGuire!
It’s beautiful but alas 67% slower for the One Millionth happy number.

```
K:\_B\HN>rexx hno 1000000 1000000
7105849
7105849 81.791000

K:\_B\HN>rexx hnrick 1000000
7105849
7105849 136.628000
```

Temperature Conversion

There are quite a number of temperature scales. For this task we will concentrate on four of the perhaps best-known ones: Kelvin, Celsius, Fahrenheit, and Rankine.

Task

Write code that accepts a value of kelvin, converts it to values of the three other scales, and prints the result.

Gerard wrote two programs doing some of and MUCH more than that.

He allows to specify several source temperatures and specific target scales.

Instead of the four scales he supports eight or, in yet another version even 58 (!) scales

His output looks, for example, like this:

```
K:\_B\TC>rexx tcgso 0 to C,10C to re,88f
-------------------------------------------------------  0 to C
                                 -273.15            Celsius
-------------------------------------------------------  10C to re
                                   8                Reaumur
-------------------------------------------------------  88f
                                  31.11111111       Celsius
                                 103.33333333       Delisle
                                  88                Fahrenheit
                                 304.26111111       kelvins
                                  10.26666667       Newton
                                 547.67             Rankine
                                  24.88888889       Reaumur
                                  15.7962963        Romer
```

As usual, two problems are found in his program:
Use of $ as function name and use of the Upper instruction.

Another problem found is this:

```
K:\_B\TC>rexx tcgso -400C To K
-------------------------------------------------------  -400C To K
                                 -126.85            kelvins
```

A temperature below absolute zero should not be accepted!

Looking at the program, the processing of the given arguments is again rather cryptic;

```
/*REXX program  converts  temperatures  for a number (8)  of  temperature scales.      */
numeric digits 120                          /*be able to support some huge numbers.*/
parse arg tList                             /*get the specified temperature list.  */

  do  until  tList=''                       /*process the list of temperatures.    */
  parse  var tList  x  ','  tList           /*temps are separated by commas.       */
  x= translate(x, '((', "[{")               /*support other grouping symbols.      */
  x= space(x);      parse var x z '('       /*handle any comments  (if any).       */
```

```
parse upper  var  z  z   ' TO '  ! .           /*separate the  TO  option from number.*/
if !==''  then != 'ALL';   all= !=='ALL'       /*allow specification of "TO" opt*/
if z==''     then call serr "no arguments were specified."                /*oops-ay. */
_= verify(z, '+-.0123456789')                 /*list of valid numeral/number thingys.*/
n= z
if _\==0  then do
               if _==1  then call serr 'illegal temperature:'   z
               n= left(z,  _ - 1)             /*pick off the number  (hopefully).    */
               u= strip( substr(z, _) )       /*pick off the  temperature  unit.     */
               end
          else u= 'k'                         /*assume kelvin as per task requirement*/
```

The syntax for each argument in the list is apparently

Number [scale1] [to scale2]

Scale 1 defaults to KELVIN

If to scale2 is omitted, the given temperature is converted to all target scales.

Processing of scales is so sophísticated that this is a valid

Input K:_B\TC>rexx tcgso -10.3degreeskels to degreedescale can be (degree[s]abbrev[s] and

there needn't be a blank between the number and the scale

the code shows how any known misspelling can be recognized:

```
when abbrev('CENTIGRADE' , yU)    |,
     abbrev('CENTRIGRADE', yU)    |,                    /* 50% misspelled.*/
     abbrev('CETIGRADE'  , yU)    |,                    /* 50% misspelled.*/
     abbrev('CENTINGRADE', yU)    |,
     abbrev('CENTESIMAL' , yU)    |,
     abbrev('CELCIU'     , yU)    |,                    /* 82% misspelled.*/
     abbrev('CELCIOU'    , yU)    |,                    /*  4% misspelled.*/
     etc.
```

I have no idea how the percentage of the misspelling is computed!

In the second program one finds expressions for the Dalton temperature

```
F= 273.15 * pow(273.15 / 273.15, n / 100) * 1.8 - 459.67
 a = (1e||(-digits()%2)-digits()%20) /*minimum number for Dalton temperature*/
eV = (F + 459.67) / 20888.1        /*compute the number of electron volts.*/
If K>a then dalt=(100*ln(k/273.15)/ln(373.15/273.15))
     else dalt='-infinity'
```

Since Classic REXX has neither logarithm nor power functions, Gerard provides them.
On this page, they are, however, compressed in an unusable block of code

```
↵?:    parse arg y; if not\== then do; if noS\=="" then if left(y, noL)==noS then return 0; if noE\== then if right(y,
noL)==noE then return 0; end;   if all | y==! then return 1;  return 0↵e:    e =
2.718281828459045235360287471352662497757247093699959574966967627724076630353547594571382178525166 4
return e            /*112 useful decimal digits. */↵isInt: return datatype(arg(1), 'W')        /*is the argument a whole
number (integer)? */↵exp:   procedure; parse arg x; ix=x%1; if abs(x-ix)>.5 then ix=ix+sign(x); x=x-ix; z=1; _=1;
w=z; do j=1; _=_*x/j; z=(z+_)/1; if z==w then leave; w=z; end; if z\==0 then z= z * e()**ix;     return z/1↵ln:
procedure; parse arg x; call e; ig=x>1.5; is=1-2*(ig\==1); ii=0; xx=x;          return ln..()↵ln..:  do while ig & xx>1.5 |
\ig & xx<.5;_=e;do k=-1;iz=xx*_**-is;if k>=0 & (ig & iz<1 | \ig & iz>.5) then leave;_=_*_;izz=iz;end;xx=izz; ii=ii+is*2**k;
end; x=x*e**-ii-1; z=0;_=-1;p=z;do k=1; _=-_*x;z=z+_/k; if z=p then leave;p=z; end; return z+ii↵pow:   procedure;
parse arg x,y; if y=0 then return 1; if x=0 then return 0; if isInt(y) then return x**y; if isInt(1/y) then return root(x,1/
y,f);  return pow.()↵pow.:  if abs(y//1)=.5 then return sqrt(x)**sign(y)*x**(y%1); return exp(y*ln(x))↵root:  procedure;
parse arg x 1 ox,y 1 oy; if x=0 | y=1 then return x; if isInt(y) then return rooti(x,y); _=sqrt(x); if y<0 then _=1/_;
return _↵rooti: x=abs(x); y=abs(y); a= digits() + 5; g=rootIg(); m= y-1; d=5;  do until d==a; d=min(d+d, a); numeric
digits d; o=0;  do until o=g; o=g; g=format( (m*g**y+x) /y/g**m, , d-2); end; end;  _= g * sign(ox); if oy<0 then _= 1/_;
return _↵rootIg: numeric form;parse value format(x,2,1,,0) 'E0' with ? 'E' _ .;          return (? / y'E'_ % y) +
(x>1)↵s:    if arg(1)==1 then return arg(3);    return word(arg(2) 's',1)       /*pluralizer.*/↵serr:  say;     say
'***error***';     say;      say arg(1);      say;       exit 13↵</lang>
```

A bug hidden in the conversion program is shown by this test

0 FAHRENHEIT TO DALTON

-21.5729765 Dalton

-21.5729765 DALTON TO FAHRENHEIT

32 Fahrenheit

The formula shown above

F= 273.15 * pow(273.15 / 273.15, n / 100) * 1.8 - 459.67

is actually incorrect and should be replaced with

F=pow(e(),(ln(373.15/273.15)*n/100))*9*273.15/5-459.67

Which is the inverse of

dalt=(100*ln(k/273.15)/ln(373.15/273.15))

With all that insight, I rewrote the program to the following “specification”:

```
K:\_B\TC>rexx tcw ?
use as command:
rexx tcw fromtemp [fromscale] [TO toscale | all],...
or as function
tcw(fromtemp [fromscale] [TO toscale])
```

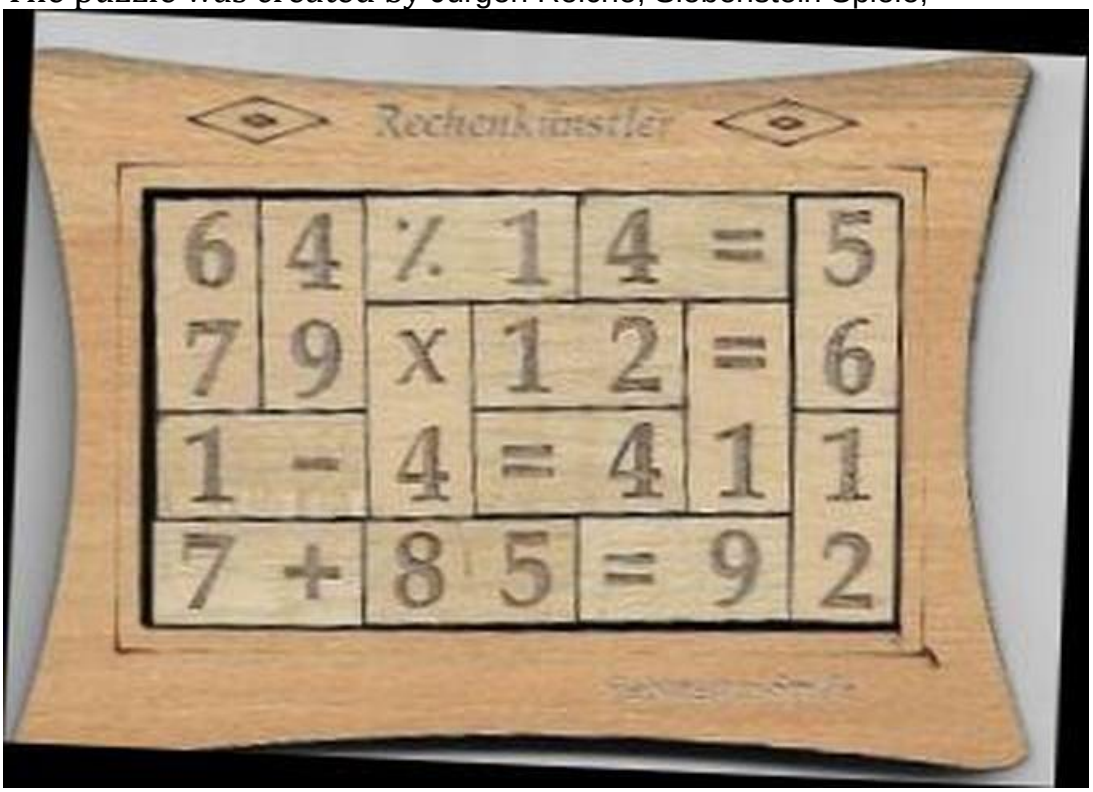
The output shows the full names of the specified scales

```
K:\_B\TC>rexx tcw -21.5729765       Dalton to f
-21.5729765 Dalton to f
-21.5729765 DALTON TO FAHRENHEIT
          0               Fahrenheit
```

```
K:\_B\TC>rexx tcw 0 c
0 c
0 CELSIUS TO all
        273.15           Absolute
         51.5000436      Amonton
          3.66666181     Barnsdorf
          0              Beaumuir
          0.49996164     Benart
          5.9999667      Bergen
          0              Brisson
          0              Celsius
         13.4999834      Cimento
```

The Christmas Puzzle

A friend of mine showed me a puzzle he got for Christmas.
It consists of a frame containing 14 (remember 14?) stones as shown here:
The puzzle was created by Jürgen Reiche, Siebenstein Spiele,



The stones should be rearranged such that each row contains a correct equation like the one in the last row.
The challenge was to use REXX for finding the solution(s).

Using Rony Flatscher's JDOR interface I printed myself the puzzle on paper, cut it into pieces and could start to experiment.

6	4	8	1	4	=	5
7	9	*	1	2	=	6
1	-	4	=	4	1	1
7	+	8	5	=	9	2

Each row can contain three vertical stones and two horizontal stones, one vertical stone and three horizontal stones, or even five vertical stones and one horizontal stone.

I number the 14 stones with 1 to 6 for vertical and 7 to 16 for horizontal stones.

Step 1

From the permutations of 4 from 14 elements I select those which contain one vertical stone (numbered 1 to 6)

From the permutations of 5 from 14 elements I select those which contain three vertical stone (numbered 1 to 6)

Each combination is checked for correct syntax:
Number operator number equal number or
Number equal number operator number
And whether the resulting equation is true.

Step 2

We combine the various rows so that the combination of two correct rows can be used at the bottom or on top of the solution.

Step 3

We look for combinations of the results from step 2 that click together,

Of course, I tested the programs by starting with valid solutions, cutting them into 6+8 pieces, and running the programs.

Some "Solutions"

```
92/4=23     17+4=21
21=17+4     11=22/2
16*3=48     27-4=23
56-55=1     33*3=99
```

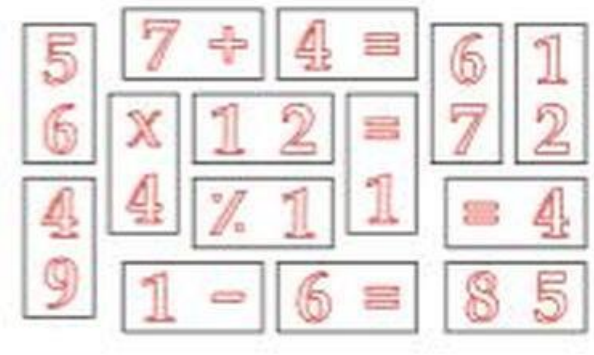
Some ways to cut them into pieces

```
12aabb5     1778899     1778899     1778845
1267745     13aa456     13aa4bb     1399645
996cc43     23bb456     23cc456     23aa6bb
88ddee3     2ccddee     2ddee56     2ccddee
```

Everything worked nicely!

Unfortunately, this splendid process did not find a solution for the given puzzle.

So I wrote an email to the puzzle's creator asking for a statement that the puzzle can be solved and, if possible, the solution.
Hours later I got a mail with the solution and the problem was found.
My friend had apparently reverted the original 6= stone to the =9 stone
Shown above.



Note that this is the only stone where this can happen.
When I see numbers that win a Lotto prize on Television,
6 and 9 are, therefore, accompanied by a period.

As to the process described above, the findings are as follows

Step 1 delivers only 2 possible equations with 4 stones:

```
o 4
m 91-6=85
u

o   4
m 6=91-85
u
```

4932 bad syntax
2436 syntax ok
1863 equation incorrect
2 correct equations

Many more (187) possible equations with 4 stones

211968 bad syntax
87360 syntax ok
64478 equation incorrect
187 correct equations

Among them

```
o       =
m 57+4=61
u 6    7

o
m 57+4=61
u 6    72
  -------
```

Among the output from Step 2: these two can be combined to the only solution

```
57+4=61
6*12=72
 4  1              57+4=61
         click to  6*12=72
                   44/11=4
 *  =              91-6=85
44/11=4
91-6=85
```

Thank you for your attention!

Work after the presentation

After the presentation I did what had to be done:
I replaced three of my "pearls" by better code

Here you can see recent changes

https://rosettacode.org/wiki/Special:RecentChanges?hidebots=1&limit=50&days=7&enhanced=1&urlversion=2

that's one of my three

20:01 Prime conspiracy? diffhist +358? Walterpachl talk contribs (??{{header|REXX}}: rewritten to be readable)

https://rosettacode.org/wiki/FASTA_format#REXX
https://rosettacode.org/wiki/Happy_numbers#REXX
https://rosettacode.org/wiki/Prime_conspiracy#REXX

Here you can view the change log and the changes made

View logs for this page (view abuse log)

Diff selection: Mark the radio buttons of the revisions to compare and hit enter or the button at the bottom.

Legend: (cur) = difference with latest revision, (prev) = difference with preceding revision, m = minor edit.

curprev 20:01, 4 March 2024? Walterpachl talk contribs? 140,020 bytes +358? ??{{header|REXX}}: rewritten to be readable undo