What is Classic Rexx?

Walter Pachl, March 2015

For a long time I was (and still am) happy that ooRexx lets me use all the programs I used on the host (under VM/CMS or MVS/TSO) also on my PC (Windows of all sorts).

The only difference that I had to consider was input/output. Under TSO, I did not have the stream I/O functions and my ooRexx did not have EXECIO. Allegedly both were available in one form or another but I never resolved this issue. I solved this "problem", however, by coding different paths for the two environments I had to deal with. Something like:

```
If g.0host Then Do
  "ALLOC FI(IN) DA('...') SHR REUSE"
  "EXECIO * DISKR IN (STEM L. FINIS"
  End
Else Do
  Do ri=1 By 1 While lines(fid)>0
    l.ri=linein(fid)
    End
  l.0=ri-1
  End
Do i=1 To l.0
  /* process l.i */
  End
```

The logic had to be changed for huge data sets. The simpler way of using "EXECIO 1 DISKR (STEM V." proved to be too slow to be useful even for small data sets.

I don't recall how I came to know rosettacode.org but it was then that I noticed that many if not most of the programs found there in the REXX category could not be run using ooRexx.

There were three language elements that their author used (and still uses):

- Variables by the name of @, $, or #
- Assignments of the form x=; i.e., without righthand side
- The UPPER instruction to translate a string to uppercase

This author's position was and is that ooRexx cannot run Classic Rexx programs. Period.
When the REXXLA home page states that "Upwardly compatible with classic Rexx" this is true to a large extent but not entirely.

I wrote a program in 'classic' Rexx that is utterly incompatible with ooRexx.

```
/* REXX ---------------------------------------------------------------
* Use all kinds of old REXX features
*--------------------------------------------------------------------*/
Parse Version version

Select
  When pos('370',version)>0 Then Do
    oid='OLDREXX TEXT A'
    'ERASE' oid
    End
  When left(version,11)='REXX-ooRexx' Then Do
    oid='z:\oldrexx.txt'
    Call SysfileDelete oid
    End
  When left(version,11)='REXX-Regina' Then Do
    Say version
    oid='z:\oldrexx.reg'
    'erase' oid
    End
  Otherwise Do
    Say 'Unknown version:' version
    Exit
    End
  End
Call o version

Call oh 'Use of @#$¢ as or in symbols'
@='Klammeraffe'; Call o '@='@
#='Kanalgitter'; Call o '#='#
$='Dollar     '; Call o '$='$
¢='Cent       '; Call o '¢='¢

Call oh 'a= as a short form of a=""'
a=
Call o 'a='a'<'

Call oh 'Multi-line strings (extending a string over line
boundaries)'
s='First
 Line'
Call o 's='s
```

```
Call oh 'the Upper instruction'
Upper s
Call o 's='s

Call oh 'Bifs: externals, find, index, justify, linesize'
Call o "externals()         ="externals()         ;
Call o "find('abc d ef gh,'d  ef')  ="find('abc d ef gh','d
ef')  ;
Call o "index('abcdef','c') ="index('abcdef','c') ;
Call o "justify('abc def',9)="justify('abc def',9);
Call o "center('abc def',9) ="center('abc def',9) ;
Call o "linesize()          ="linesize()          ;

Call oh '/= and /== as alternatives to \= or \=='
x=13
Call o "x="||x
Call o "x/=13 -->"||(x/=13)
Call o "x/==' 13 '-->"||(x/==' 13 ')
Exit
oh:Call lineout oid,copies('-',62)
o: Say arg(1)
   Return lineout(oid,arg(1))
```

The output when run under CMS (to which I got recently access to, thanks to Frank Hessler) looks as follows:

```
REXX370 4.02 01 Dec 1998
--------------------------------------------------------------
Use of @#$¢ as or in symbols
@=Klammeraffe
#=Kanalgitter
$=Dollar
¢=Cent
--------------------------------------------------------------
a= as a short form of a=""
a=<
--------------------------------------------------------------
Multi-line strings (extending a string over line boundaries)
s=First Line
--------------------------------------------------------------
the Upper instruction
s=FIRST LINE
--------------------------------------------------------------
Bifs: externals, find, index, justify, linesize
externals()         =0
find('abc d ef gh,'d  ef')  =2
index('abcdef','c') =3
justify('abc def',9)=abc   def
center('abc def',9) = abc def
linesize()          =80
```

```
-------------------------------------------------------------------
/= and /== as alternatives to \= or \==
x=13
x/=13 -->0
x/==' 13 '-->1

When running this program with ooRexx, I had to eliminate most of
the lines due to syntax errors:

------------------------------------------------- z:\2a.txt
    10 *-* @
Error 13 running Z:\oldrexx.rex line 10:  Invalid character in
program
Error 13.1:  Incorrect character in program "@" ('40'X)
------------------------------------------------- z:\2b.txt
    16 *-* a=
Error 35 running Z:\oldrexxo.rex line 16:  Invalid expression
Error 35.918:  Missing expression following assignment instruction
------------------------------------------------- z:\2c.txt
    19 *-* s='First
Error 6 running Z:\oldrexxo.rex line 19:  Unmatched "/*" or quote
Error 6.2:  Unmatched single quote (')
------------------------------------------------- z:\2d.txt
    38 *-* Call o "x/=13 -->"||(x/=13)
Error 35 running Z:\oldrexxo.rex line 38:  Invalid expression
Error 35.1:  Incorrect expression detected at "/="
------------------------------------------------- z:\2e.txt
Z:\OLDREXX konnte nicht gefunden werden
Der Befehl "UPPER" ist entweder falsch geschrieben oder
konnte nicht gefunden werden.
    28 *-* Call o "externals()         ="externals()         ;
Error 43 running Z:\oldrexxo.rex line 28:  Routine not found
Error 43.1:  Could not find routine "EXTERNALS"
------------------------------------------------- z:\2f.txt
```

The only statement that was accepted was the invocation of the CENTER builtin function:

```
REXX-ooRexx_4.2.0(MT)_32-bit 6.04 22 Feb 2014
Bifs: externals, find, index, justify, center, linesize
center('abc def',9) = abc def
```

When run with Regina, the output looks as follows.
First a number of Syntax errors that I eliminated one by one.

```
Error 13 running "Z:\oldrexx.rex", line 30: Invalid character in
program
Error 13.1: Invalid character in program "('a2'X)"

Error 6 running "Z:\oldrexxr.rex", line 38: Unmatched "/*" or
```

```
quote
Error 6.2: Unmatched single quote (')

Error 35 running "Z:\oldrexxr.rex", line 59: Invalid expression
Error 35.1: Invalid expression detected at "="
```

After this the program ran successfully to completion:

```
Der Befehl "SYSFILEDELETE" ist entweder falsch geschrieben oder
konnte nicht gefunden werden.
Der Befehl "EXTERNALS" ist entweder falsch geschrieben oder
konnte nicht gefunden werden.
Der Befehl "LINESIZE" ist entweder falsch geschrieben oder
konnte nicht gefunden werden.

REXX-Regina_3.8.2(MT) 5.00 22 Jun 2014
-------------------------------------------------------------------
Use of @#$¢ as or in symbols
@=Klammeraffe
#=Kanalgitter
$=Dollar
-------------------------------------------------------------------
a= as a short form of a=""
a=<
-------------------------------------------------------------------
the Upper instruction
s=FIRST LINE
-------------------------------------------------------------------
Bifs: externals, find, index, justify, linesize
externals()          =
find('abc d ef gh,'d  ef')  =2
index('abcdef','c') =3
justify('abc def',9)=abc   def
center('abc def',9) = abc def
linesize()           =
-------------------------------------------------------------------
/= and /== as alternatives to \= or \==
x=13
```

I was surprised, however, that the invocation of 'builtin functions' that are not supported caused only a warning message and returned a null string. When I asked Mark Hessling about that behavior, he advised me to use

```
regina -oNOEXT_COMMANDS_AS_FUNCS oldrexx.rex
```

and this cured, or rather showed the problem (after installing Regina 3.9)

```
    49 +++ Call o "externals()          ="externals()
Error 43 running "Z:\oldrexxr.rex", line 49: Routine not found
```

```
Error 43.1: Could not find routine "EXTERNALS"
```

This reminds me of a 'problem' with the Rexx compiler. Since the compiler performs a thorough syntax check, it is impossible to get an executable for a program that has a syntax error. The Interpreter would interpret the program until that error is actually encountered in the execution path. Customers compiling their programs would thus discover 'hidden' faults, i.e., errors that were not encountered because they were in paths not normally or never taken.

Another member of the Rexx community, Les Koehler, did some research on that matter and came up with a matrix that I am allowed to show you here.

Matrix © Les Koehler

Let me now comment on some of the differences. Consider this little program:

```
/* REXX */
Parse Version version
If pos('370',version)>0 Then
  oid=left(version,7) 'TEXT A'
Else
  oid=left(version,11)'.txt'

Say 'oid='oid
Call head 'stem assignment'
a.='a'; a.3='3'
b.='b'; b.3='*'
Call o 'a. a.3:' a. a.3
Call o 'b. b.3:' b. b.3
b.=a.
Call o 'b. b.3:' b. b.3 '(after b.=a.)'

Call head 'random'
Call o ''
x=random(,,13)
ol=''
Do i=1 To 10
  ol=ol random(9)
  End
Call o ol

Call head 'single line comment'
v=17--3
Call o 'v='v

Call head 'justify'
s=' a b c d e f '
Call o justify(s,8,'+')
```

```
Say 'Output is in' oid
Exit
o:    Say arg(1)
      Return lineout(oid,arg(1))

head:
head: Call o '.'
      Return o(arg(1))
```

You get three different sets of output:

```
REXX370 4.02 01 Dec 1998   REXX-Regina_3.8.2(MT) 5.00   REXX-oRexx_4.2.0(MT)_32-
bit
.                          .                          .
stem assignment            stem assignment            stem assignment
a. a.3: a 3                a. a.3: a 3                a. a.3: a 3
b. b.3: b *                b. b.3: b *                b. b.3: b *
b. b.3: a a (after b.=a.)  b. b.3: a a (after b.=a.)  b. b.3: a 3
                                                      (after b.=a.)
.                          .                          .
random                     random                     random   (Note 1)
 6 3 9 1 9 3 7 6 2 6        0 0 2 4 3 3 1 2 0 6        0 2 4 8 1 7 0 6 6 8
.                          .                          .
single line comment        single line comment        single line comment
v=20                       v=17   (Note 2)            v=17
.                          .                          .
justify                    justify                    justify
a++b+c+d                   a+b+c+d+                   a+b++c+d (Note 3)
```

1) The algorithm for RANDOM is neither standardized nor the same in the Implementations shown here. MFC left it deliberately open in order not to preclude the use of a 'better' algorithm.
2) single line comments can be disabled in Regina by using this command regina -oNOSINGLE_LINE_COMMENTS pgmname
3) JUSTIFY is no bif in ooRexx. The result shown comes from my translation of MFC's assembler algorithm to Rexx. I could not get the algorithm used in Rexx/370 since this is IBM Confidential.

CHANGESTR

I was very surprised and somewhat annoyed by the syntax of the newly introduced CHANGESTR builtin function, Why oh why is it so different from the good old TRANSLATE builtin function?

```
/* TRANSLATE returns the characters of its first argument with each
   character either unchanged or translated to another character.    */
Say "translate('aa bb cc','a','b')  ="translate('aa bb cc','a','b')
/* CHANGESTR replaces all occurrences of the first argument within
   the second argument, replacing them with the third argument.      */
Say "'CHANGESTR'('b','aa bb cc','a')="'CHANGESTR'('b','aa bb cc','a')
Say "changestr('aa bb cc','a','b')  ="changestr('aa bb cc','a','b')
```

```
Exit
/*--------------------------------------------------------------------
translate('aa bb cc','a','b')  =aa aa cc
'CHANGESTR'('b','aa bb cc','a')=aa aa cc
changestr('aa bb cc','a','b')  =aa aa cc
*--------------------------------------------------------------------*/
changestr: Procedure
Parse Arg string,new,old
Return 'CHANGESTR'(old,string,new)
/* or Return 'CHANGESTR'(arg(3),arg(1),arg(2)) */
```

JUSTIFY

You can see that CMS and Regina insert plus characters differently. Actually I consider Regina's result incorrect: there shouldn't be a plus at the end! I was told that justify was dropped in ooRexx because of proportional fonts. I think that argument could also be used to drop center.

RESULT

Discussions on rosettacode showed that some early REXX implementations set (or drop) the special variable RESULT not only upon Return from a called subroutine but also when returning to a function invocation.
Consider this piece of code:

```
squ=square(3)
say squ result
...
squ: Return arg(1)**2
```

While this redundancy was certainly not intended, it may have been caused by implementers blindly implementing what can be read in TRL1 and TRL2:

If a subroutine is being executed (see the CALL instruction, on page 43) then the expression (if any) is evaluated, active control constructs are terminated, control passes back to the caller, and the special variable RESULT is set to the value of the expression. If no expression was specified, the variable RESULT is dropped (becomes uninitialized). The various settings saved at the time of the CALL (tracing, addresses, etc.) are restored.
If a function is being executed, then the action taken is the same, except that an expression must be specified on the RETURN instruction. The

result of the expression is then used in the original expression at the point where the function was invoked. See the description of functions on page 77 for more details.

While preparing this paper I discovered another RETURN peculiarity: While end of source implies an EXIT for most REXX implementations, Regina differs by implying a RETURN. If the end of source is encountered at the end of an internal subroutine, this results in returning to the point of invocation instead of leaving the program.

To end on a somewhat funny (or rather sad) note, I'll tell you a little story.

When I had to change to a new development environment (an ... MVS/TSO) I found that there was no Rexx support for developers. It was not possible to employ a private REXX library by concatenating it in the search sequence. My attempts to solve this deficiency involved my sending a list of most useful ISPF edit macros to the responsible manager.

The ongoing email conversation ended with the following note:

Hallo Herr Pachl.
Ich kann in Ihrer Liste kein Makro erkennen, ohne dass nicht programmiert werden kann. In der Tat arbeiten bereits 1000 Entwickler der ... seit Jahren mit dem System, offenbar geht es also.

Mit freundlichen Grüßen,

or in plain English:

Hello Mr. Pachl.
In your list, I cannot see a macro that is necessary fro writing programs. Indeed, a thousand developers of ... have been working with the system. Apparently it's possible.

Regards,

Fortunately an old friend and colleague from the IBM Lab times (Karl Vago) devised a way to dynamically allocate the REXX library at the beginning of each session. Entering that command after logon gave me

back the power of my ISPF macros.

rosettacode.org

A short story from rosettacode.org. We find two solutions to compute the greatest common divisor in the REXX category.
Version 1 must be adapted for ooRexx by replacing the symbol $ and extending the incomplete assignment ($=;)

```
/*--------------------------------------------------------------------
gcd: procedure; $=;   do i=1 for arg();  $=$ arg(i);  end    /*arg list.*/
parse var $ x z .;  if x=0 then x=z;  x=abs(x)  /*handle special 0 case.*/
   do j=2  to words($);   y=abs(word($,j));       if y=0  then iterate
     do until _==0;  _=x//y;  x=y;  y=_;  end   /*?-- the heavy lifting.*/
   end   /*j*/
return x
*--------------------------------------------------------------------*/
gcd1: procedure; D=''; do i=1 for arg(); D=D arg(i);  end    /*arg list.*/
parse var D x z .;  if x=0 then x=z;  x=abs(x)  /*handle special 0 case.*/
   do j=2  to words(D);   y=abs(word(D,j));       if y=0  then iterate
     do until _==0;  _=x//y;  x=y;  y=_;  end   /*?-- the heavy lifting.*/
   end   /*j*/
return x
```

Version 2 is most elegant albeit handling only two arguments. It is somewhat slower than version 1

```
GCD2: Procedure
/*********************************************************************
* Recursive procedure as shown in PL/I
*********************************************************************/
Parse Arg a,b
If b = 0 Then Return abs(a)
Return GCD2(b,a//b)
```

Finally I derived Version 3 from Version 1 by avoiding the transformation of the argument list. This is about twice as fast as Version 1.

```
gcd3: procedure
x=abs(arg(1))
do j=2 to arg()
  y=abs(arg(j))
  If y<>0 Then Do
    do until z==0
      z=x//y
      x=y
      y=z
      end
    end
  end
return x
```

A performance comparison of these versions and some reasoning about the differences you can find on
http://rosettacode.org/wiki/Least_common_multiple#versions_1_and_2_compared

A fairly complete list of REXX implementations you can find on
http://rosettacode.org/wiki/Category:REXX

Three of the entries there are of particular interest:

"T/REXX, a REXX compiler for CMS is an implementation of classic REXX written by Lundin and Woodruff (according to Wikipedia).

When I first saw this one, Lundin and Woodruff were said to be the authors of the VM/CMS compiler. This was based on a reference on http://en.wikipedia.org/wiki/Rexx#See_also, which says:
Lundin, Leigh; Woodruff, Mark (1987-04-23). "T/REXX, a REXX compiler for CMS". U.S. Copyright Office (Washington, DC: Independent Intelligence Incorporated) (TXu000295377).

VM/CMS REXX is an IBM implementation of classic REXX that was first implemented in the early 1980s. A license is required to use this product as well as the operating system that it runs under. This was the original implementation of REXX written by Mike Cowlishaw of IBM (circa 1979).

VM/CMS REXX compiler is an IBM implementation of classic REXX, a license is required to use this product as well as the operating system it runs under.

The Regina Reference Manual contains a list of builtin functions implemented or described in CMS, Regina, AREXX, and the ANSI Standard. This document lists also the operating systems where one can use Regina.

As for me, I have been using ooRexx and Regina only on Microsoft Windows 7.

Some of my other activities on rosettacode.org

http://rosettacode.org/wiki/Trigonometric_functions#ooRexx
shows an implementation of rxMathCalc in ooRexx for arbitrary precision (advised by Rony Flatscher).

http://rosettacode.org/wiki/Arithmetic/Rational#PL.2FI
shows an implementation of fraction arithmetic in PL/I (using complex numbers to represent fractions)

http://rosettacode.org/wiki/Tree_traversal#REXX
shows tree traversal algorithms in REXX

http://rosettacode.org/wiki/Conway%27s_Game_of_Life#REXX
shows two versions of Conway's Game of Life

http://rosettacode.org/wiki/Draw_a_clock#ooRexx
shows the algorithm for a wall clock that I developed with lots of help by the late Mark Miesfeld.

http://rosettacode.org/wiki/Category_talk:OoRexx
shows very long “discussions” on compatibility and syntax highlighting which were the first battlefields where I met Gerard Schildberger, the author of many if not most REXX solutions on rosettacode.

http://rosettacode.org/wiki/Rosetta_Code:Village_Pump/Dialects#REXX.2C_ooRexx.2C_and_others
was my first attempt to document the differences between REXX/370 and ooRexx with respect to compatibility.