Intro to Rexx
Hands-On Workshop

2015 VM Workshop
Binghamton University
Binghamton, NY

Chip Davis
chip@aresti.com

Topics

- A whirlwind intro to Rexx

- A couple of quick demos

- Some hands-on problems

- Yes, suggested solutions will be given

Rexx Statements

Statements are analyzed in this order:

Null

```
/* comment */
 [or blank line]
```

Label

```
symbol:
```

```
    Get_Args:        exit:
```

Assignment

```
symbol = expression
```

```
    foo = Length(bar'/'baz) + 2
```
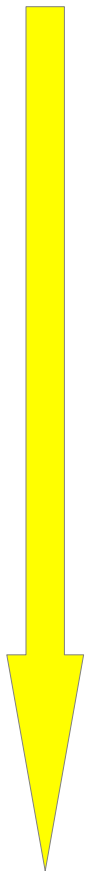
Instruction

```
keyword [expression]
```

```
    If, Do, Say, Parse, Else, ...
```

Command

```
expression
```

```
    'CP QUERY' spool splid
```

Rexx Symbols

Used as variables and labels

Allowed Characters: [**A** – **Z**] [**a** – **z**] [**0** – **9**] **_** **.** **?** **!**

    (some platforms allow extra non-ANSI characters)

Format: Must NOT start with **.** or [**0** – **9**]

Mixed case allowed: Yes

Case-sensitive: No, automatically uppercased

Max Symbol Length: 250 characters

    A *symbol* used as a **variable** may reference a string up to 16 MB long.
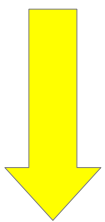
Rexx Expressions

- Any sensible combination of:

  <u>Constants</u>   <u>Variables</u>   <u>Operators</u>   <u>Functions</u>

- Blanks may be used between them for readability

- Evaluation order:

  - from inner to outer nested parentheses

  - by operator precedence

  - from left to right

- Evaluation results in a single character string

- A zero-length string is valid and called the "null string"

- Conversion to internal numeric form is done only when operation calls for it (e.g. decimal arithmetic)

Rexx Constants

- Stored as character strings

  - conversion to numeric form only when required

- Length limited to: 16,777,215 chars/decimal digits

- String   (must be enclosed in single- or double-quotes)

  `'Susie "Q"'    ''   ""        "Don't look"`

- Numeric

  `2.54          –.088        6.626e–34`

- Hex

  `'F8'x        'D0A'X        "c3c8 c9d7"x`

- Binary

  `'11111000'b   '11'B     "1111 0011"b`

Rexx Variables

- Simple: *symbol*

  - No '.'s in *symbol* (default value is *SYMBOL*)

  x15   !0_1?   Last_Matching_Value   foo

- Compound: *symbol.tail[.tail]...*

  - '.'s separate *stem* and *tails*

  - *tail* may be a numeric constant or simple variable

  matrix.12   tax.form.st   vm.mascot

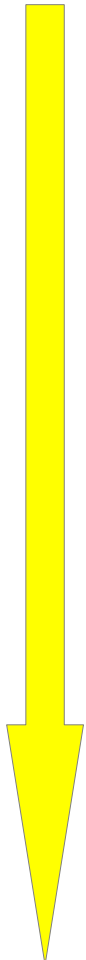 - All tail variable values are substituted, then the resulting *derived name* is used to access the value

  MATRIX.12   TAX.1022.NC   VM.EdgarBear

  - (Re)set ALL *stem.* variables with: *stem. = expr*

  matrix. = 0   line. = ''   Qin. = x/y+3

Rexx Operators

A single *expression* may contain any or all operators!

Prefix (monadic)	+ (pos) – (neg) \ (not)
Power (decimal)	**
Multiplicative (decimal)	* / % //
Additive (decimal)	+ –
Concatenation	|| [*abbutal*] [*blank*]
Comparison - normal	= < > <= >=
- strict	== << >> <<= >==
Logical AND	&
Logical OR/XOR	| &&

```
Say 'February has' 28 + (yr // 4 = 0) 'days'!
```
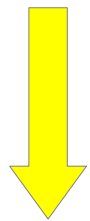
Rexx Functions

symbol([*expr1*],[*expr2*],[...],[*exprn*])

- Invokes a subprocedure which returns a string that replaces the function invocation

  - Expression evaluation continues

- Search Order:

  - Internal to the running program (*symbol:*)

  - Built-in, Platform Extensions, Function Packages

  - External Rexx program (*symbol* EXEC)

```
If \DataType(vaddr,'X') Then Say "Invalid device address"
Say Center(' Monthly Report ',78,'-')
days_between = Abs(Date('B',date1,'U') - Date('B',date2,'U'))
button = "Perl is just Rexx with bad syntax"
Say '"Rexx" is word' WordPos(button, 'Rexx') "of" button"."
Say hex1 '-' hex2 '=' D2X( X2D(hexadr1) - X2D(hexadr2))
If Random(1) Then Say "Heads" ; Else Say "Tails"
If SourceLine(2)='/*Test*/' Then Call Trace '?R'
```

Rexx Subprocedures

- Subprocedure does not know or care how it was invoked
- Invoke as:
    - Function: $symbol(expr1,expr2,...,exprn)$
    - Subroutine: CALL $symbol\ expr1,expr2,...,exprn$
- Within a subprocedure:
    - Get arguments   PARSE ARG $arg1, arg2, ...$
    - Return a value   RETURN $expr$
- Returned value will:
    - Function: Replace the function invocation
    - Subroutine: Replace the value in RESULT variable

Rexx Subprocedures

```
         ...
Call MyTip meal,20
Say "You should leave" result
         -or-
      ...
Say "You should leave" MyTip(meal,20)
      ...
MyTip: Parse Arg tab, pct
      tip = tab / (100 / pct)
      Return '$'Format(tip,,2)

-------------------------------------------------

x = Length('This is preferred')
         -or-
Call Length('This is dumb')
x = result
```

Rexx Stream I/O

- CMS has separate Read & Write pointers

- Stream I/O functions

  - Stream()   Housekeeping

  - LineIn() / LineOut()   Read/Write a line

  - CharIn() / CharOut()   Read/Write characters

  - Lines() / Chars()   Return count of remaining

```
myfile = 'TEST DATA A'
line42 = LineIn(myfile,42)
line43 = LineIn(myfile)
stat = LineOut(myfile,'This is the new last line')
Say "There are still" Lines(myfile) "left to read."
```

Read File into Stem Array

```
Parse Value Stream(myfile,'C','OPEN READ') ,
     With status extra
If status \= 'READY:' Then [...]

line. = ''
Do i = 1 While Lines(fileid) > 0
  line.i = LineIn(fileid)
End i
line.0 = i - 1

/* Now display it on the screen */
Do j = 1 To line.0
  Say 'Line' j':' line.j
End j
```

Trace [?]*type*

- Executes statement, then displays source and trace lines

- Many *type*s but you'll only need these three:

    - Trace Off   No trace output generated

    - Trace Results   One trace line per line of code

    - Trace Intermediates   More answer than you have
        question...

```
42 *-* Return '$'Format(tip,,2)
   >>>   "$2.75"
```

- Interactive tracing: Trace ?*type*

    - Pauses for input after tracing an instruction

    - Anything entered at pause point will be executed as
        if it were at that line in the program

    - Last instruction traced may be re-executed (!)

Trace Identifiers

`*-*`	Rexx instruction as coded in program
`>>>`	String result of executing instruction
`>.>`	String ignored in Parse template
`>C>`	Derived name of compound variable
`>F>`	String returned from a Function
`>L>`	Literal string encountered
`>O>`	Result of a dyadic operation
`>P>`	Result of a monadic (prefix) operation
`>V>`	String retrieved from a variable
`+++`	Trace message

If - Then - Else

```
IF   cond_expr
THEN   statement
[ELSE  statement]
```

- *cond_expr* must evaluate to **0**(false) or **1**(true)

- THEN and ELSE may be followed by <u>one</u> statement (which may be NOP)

- Multiple statements may be grouped by enclosing them in a Do - End block

```
If Length(data) <= lrecl
   Then line.next = data
   Else Do
      Call Error lrecl, data
      Exit 99
      End
```

Iterated Do-Loops

Do *count_expr*
   [statements]
End

```
scale = ''
Do lrecl % 5 + (lrecl // 5 > 0)
  scale = scale"----+"
End
```

Do *ndx_var = beg_expr* [To *end_expr*] [By *incr_expr*]
   [statements]
End *ndx_var*

```
merge. = ''
Do oddndx = 1 To Lines(file1)*2-1 By 2
  merge.oddndx = LineIn(file1)
End oddndx
```

Conditional Do-Loops

Do While *cond_expr*   ⬅   *cond_expr* evaluated here
   [statements]
End

```
Do i = 1 While rec.i \= ''
  lrc = Lineout(outfile,rec.i)
End i
```

Do Until *cond_expr*
   [statements]
End   ⬅   *cond_expr* evaluated <u>here</u>

```
Do Until rec = 'EOF'
  Say "Enter record:"
  Parse Pull rec
  lrc = LineOut(outfile,rec)
End
```

Leave & Iterate

- LEAVE [*ndx_var*]   Terminates loop and continues with the instruction after the END

- ITERATE [*ndx_var*]   Skips to the END instruction and returns to the DO instruction to continue from the top of the loop

- If *ndx_var* specified, applies to DO *ndx_var* = ... loop

    - Otherwise, applies to current loop

```
comp. = '' /* Copy non-comment lines to comp. array */
j = 0
Do i = 1 to lines.0
  If line.i = '' Then Leave i        /* at EOF, done */
  If Left(line.i,1) = '*'    /* Don't copy this line */
    Then Iterate i
  j = j + 1
  comp.j = line.i
End i
comp.0 = j
```

Parse [Upper]

PARSE ARG	*template(s)*	Argument string(s)
PARSE PULL	*template*	External data queue/keyboard
PARSE VAR *symbol template*		String in variable
PARSE SOURCE	*template*	Program metadata
PARSE VALUE *expr* WITH *template*		String value of *expr*

- A *template* is constructed from *variable names*, *patterns*, and *placeholder* '.'s

- If no *patterns*, string is "word parsed":

  - Each blank-stripped word of data string is assigned to each variable L-R

  - If no data for variable, it is assigned the *null string* ("")

  - If data left over, the last variable is assigned the remainder of the string (incl. blanks)

Word Parsing

```
Say "Enter your email address:"
Parse Upper Pull email .
Say "Enter your name:"
Parse Pull name

nums = LineIn(num_file)
Parse Var nums num1 num2 num3 .

'QUERY DISK' md '(LIFO'
Parse Pull . . . stat . . . . . avail .
If stat = 'R/W' & avail > need Then [...]

Parse Source opsys how sfn sft sfm cmd cif
Say "This is" sfn sft "on the" sfm"-disk"
Say "of a" opsys "system. It was invoked"
Say "as a" how "with" cmd". The initial"
Say "command interface was" cif"."
```

Template Patterns

Patterns may be a

- String: `','`   `'0D0A'x`   `"  "`   `'POS='`

- Numeric

  - Absolute: `=12`   `13`   `=9999999`

  - Relative: `+12`   `+0`   `-42`

- Variable containing a pattern

  - String: **(***symbol***)**

  - Absolute: **=(***symbol***)**

  - Relative: **+(***symbol***)**   **−(***symbol***)**

Pattern Parsing

```
Say "Enter your email address:"
Parse Upper Pull user '@' domain '.' tld
Say "Enter your name (Last, First):"
Parse Pull lname', 'fname

Parse Value Date('E') With dd '/' mm '/' yy
c = ':'
Parse Value Time('N') With hr (c) mn (c) sc

nums = LineIn(num_file)
Parse Var nums =2  num1 =7  . ,
               =9  num2 =17 . ,
               =25 num3 =32 .
Parse Var nums =2  num1 +5 . ,
               =9  num2 +8 . ,
               =25 num3 +7 .
Parse Var nums =2  len1 +2 num1 +(len1) ,
               =9  len2 +2 num2 +(len2) ,
               =25 len3 +2 num3 +(len3)
```

Address *interface* [*command*]

- Controls to which interface *command* is sent

- If *command* omitted, sets interface for subsequent cmds

- Many interfaces available - two for CP/CMS commands:
  ADDRESS CMS [*command*]   (default)
  - Full CMS command line hand-holding:
    uppercasing, EXEC lookup, synonyming, abbreviating

  ADDRESS COMMAND [*command*]
  - WYWIWYG - more robust, more explicit, no surprises

  - Must specify *command* in uppercase, preface with 'CP' or
    'EXEC' if not a CMS command/module, no synonyms,
    no abbreviations

- Return code from *command* replaces value in variable RC

```
Address Command 'CP SPOOL' spl 'CLASS' cls
'QUERY DISK R'
If Rc \= 0 Then Call Cmd_Error
```

Retrieving Command Output

- Divert output from screen into stem array

- CMS - use the Stack interface

```
'QUERY SEARCH (STACK'
Do i = 1 To Queued()
   Parse Pull qsline.i
End i
qsline.0 = i - 1
```

- CP - use the Diagnose interface

```
d8out = Diag(8,'CP QUERY NAMES')
Do i = 1 While d8out \= ''
   Parse Var d8out qnline.i '15'x d8out
End i
qnline.0 = i - 1
```

Demos

- SAY EXEC   - Q&D Rexx expression tester
- REXXTRY EXEC - SAY EXEC on steroids
- TIPPER EXEC - How to use Tracing
- CPCMD EXEC   - How to issue commands to CP/CMS
- PI EXEC   - NUMERIC DIGITS 10000 or more

Lab Exercises

- Lab Exercises are in ***problem* LAB D**

- Suggestions:

    - `SCOPY, ENUFF, WC, MAGIC8, DUMPMEM`

    - `MAGIC8` needs to be completed

    - `DUMPMEM` has three bugs

    - Start with `TRACE R` setting

    - Use `REXXTRY` & `SAY EXEC`s to test snippets

    - `RENAME PROFILE SXEDIT D = XEDIT =`

    - Ask for help <u>before</u> you get frustrated

Finally...

- Lab Exercises are in **_problem_ LAB D**

- Price List

Hints, Tips, Nudges	$0.10
Good Answers	.25
Complete Answers	.50
More Than You Want To Know	Free

- Questions?