Introduction to z/VM Rexx Hands-on Lab

Updated with answers to lab exercises

Sessions 9167-9168
SHARE 113
Denver, Colorado
August 2009

John Franciscovich
IBM

Phil Smith III
Voltage Security, Inc.

IBM Systems

Trademarks

The following are trademarks of the International Business Machines Corporation in the United States, other countries, or both.

Not all common law marks used by IBM are listed on this page. Failure of a mark to appear does not mean that IBM does not use the mark nor does it mean that the product is not actively marketed or is not significant within its relevant market.

Those trademarks followed by ® are registered trademarks of IBM in the United States; all others are trademarks or common law marks of IBM in the United States.

For a complete list of IBM Trademarks, see www.ibm.com/legal/copytrade.shtml:

*, AS/400®, e business(logo)®, DBE, ESCO, eServer, FICON, IBM®, IBM (logo)®, iSeries®, MVS, OS/390®, pSeries®, RS/6000®, S/30, VM/ESA®, VSE/ESA, WebSphere®, xSeries®, z/OS®, zSeries®, z/VM®, System i, System i5, System p, System p5, System x, System z, System z9®, BladeCenter®

The following are trademarks or registered trademarks of other companies.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
UNIX is a registered trademark of The Open Group in the United States and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.
IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.

* All other products may be trademarks or registered trademarks of their respective companies.

Notes:
Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.
IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.
All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.
This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.
All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.
Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.
Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

Disclaimer

Agenda

- Rexx Overview and Related Topics
- Creating and Executing Rexx Programs
- Rexx Language
  - Basic Syntax
  - Strings, Operators, Expressions
  - Tracing, Parsing
  - Control Constructs
  - Subroutines & Functions
  - Issuing Commands and use of Pipelines
- Lab Exercises

Rexx Overview

- **RE**structured e**X**tended e**X**ecutor

- Rexx is a procedural, general purpose language available on many platforms
  - Intuitive
  - Easy to use and read
  - Language concepts are the same on all platforms
    - Minor differences such as file names and structure
    - Operating system-specific tools that support Rexx

Rexx Overview (cont.)

- Few restrictions on program format
  - Indentation
  - 1 or more clauses on a line
  - /* comments can be anywhere and any length */
  - *Implied* semicolon delimiters at end of lines
  - Comma (,) as a continuation character

- Natural data typing
  - Meaning of data depends entirely on their usage

Rexx Overview (cont.)

- Dynamic Scoping
  - ▶ Efficiently interpreted because minimal look-ahead is needed
  - ▶ Meaning of an instruction is only affected by the instructions already executed

- Nothing to Declare !
  - ▶ May document and initialize variables, but…
  - ▶ Implicit declarations take place during execution
  - ▶ **labels:** are the only true declarations

Rexx Overview (cont.)

- The Rexx Evolution…
  - Rexx Sockets API
    - Function package for writing socket applications
  - Object Rexx
    - Object-Oriented Rexx supporting many utilities for a UNIX-type environment, including Linux for System z
  - NetRexx
    - Blend of Rexx and Java; compiles into Java classes
  - Regina Rexx
    - Rexx interpreter ported to most UNIX platforms, including Linux
  - Etc……
- See references page for website information

Creating Rexx Programs

- Create a file with filetype of EXEC using XEDIT, the CMS editor

```
XEDIT  myrexx exec a
```

- Rexx programs begin with a comment line:

```
/*  beginning of program  */          /* Rexx */
```

- Can be run uncompiled and interpreted, or compiled with the Rexx compiler
  - Improved Performance
  - Security

Executing Rexx Programs

- Search order
  - Same for both compiled and interpreted execs
  - Loaded and started through CMS EXEC handler
  - Normal CMS Command search order:

    EXECs, synonyms, MODULEs…

- Invocation
  - Invoke as a CMS command or EXEC:

    myexec -or- **exec myexec**

    - Implied exec (IMPEX) settings control whether exec files are treated as commands
      - SET IMPEX ON|OFF (default is ON)
      - QUERY IMPEX

Helpful Hints for our Exercises

- List Files on A-disk:

  FILELIST * * A or… **LISTFILE * * A**

- XEDIT a file
  - from command line:

    Xedit Filename Filetype Filemode

  - from prefix area on Filelist Screen, PF11 or :

```
x   PROFILE  XEDIT    A1 V      75      74      1  09/17/07 15:48:18
```

- Prefix area commands within the file:

  a add (insert) a single line to the file
  d delete a line (**d5** deletes 5 lines)
  m move a line (**f** following or **p** preceding)
  c copy a line (**f** following or **p** preceding)
  mm…mm block move, **dd…dd** block del, **cc…cc** block copy

Helpful Hints for our Exercises (cont.)

- Screen execution modes
  - **CP Read**
    - CP is waiting for a command
  - **VM Read**
    - CMS is waiting for a command
  - **Running**
    - System is ready for commands or is working on some
  - **More …**
    - More information than can fit on the screen is waiting to be displayed)
      - Clear screen manually or let CP clear after x seconds determined by TERM command setting
  - **Holding**
    - Waiting for you to clear screen manually
  - **Not Accepted**
    - Too many commands in buffer; wait for executing command to complete)

Lab Exercises: What to Expect…

1. Update an existing Rexx program to format a string
2. Write a program to accept an input argument, prompt for data, and display results
3. Trace and Debug existing Rexx programs
4. Write a sort program using stems and various control constructs
5. Write a program using a subroutine to issue CMS commands and Pipes to query accessed disks
6. Write a program to obtain z/VM CP level information

Logging on to the z/VM Lab System

- 3270 Session
- Userids and Passwords

Rexx Language Syntax

- Case **I**nsensitivity

  Denver is the same as **denver**

  - specific support for upper and lower case is provided
  - cases in quoted strings are respected

- *All* Rexx programs must begin with a comment

```
/* This is a comment */
```

- Long lines are common
  - Continuation with commas

```
say 'This text is continued ',
          'on the next line'
```

  - May wrap as a long single line *(but don't do this)*

```
say 'This text is continued
         on the next line'
```

Rexx Strings

- Literal strings: Groups of characters inside single or double quotation marks

```
"Try a game of blackjack",'and beat the odds!'
```

- Two " or ' indicates a " or ' in the string

```
'Guess the dealer''s top card'
"The dealer""s card is an Ace"
```

- Hexadecimal strings: Hex digits (0-9,a-f,A-F) grouped in pairs:

  `'123 45'x` is the same as `'01 23 45'x`

- Binary strings: Binary digits (0 or 1) grouped in quads:

  `'10000 10101010'b` is the same as `'0001 0000 1010 1010'b`

Input and Output

- **say [expression]**
  - writes output to the user's terminal

```
say 'Five Euros equals ' ,
5 * 1.30 'USD'
```

- **pull**
  - prompts for input from the user's terminal

```
pull rate
say 'Five Euros equals' 5 * rate 'USD'
```

- **parse arg**
  - collects arguments passed to a Rexx Program
    - Invoke program: **`EXAMP input1 dataX moreData`**

```
parse arg A1 A2 A3
say A1 A2 A3
```

    - Result:

```
input1 dataX moreData
```

Exercise 1: Syntax and Strings

- Adjust STRING EXEC to provide the following output:

```
"      T'
was a dark and stormy night. Ne'er
before, in all their days, had th
e hackers seen a
program so complex …      "
```

Exercise 1: Syntax and Strings - Answer

```
/* STRING EXEC - Syntax and Strings */

say '"     T'''
say "was a dark and stormy night. Ne'er"
say 'before, in all their days, had th'
say 'e hackers seen a'
say "program so complex ...     """
```

Operators & Expressions

- String Expressions

  (blank) "Mile" "High" --> "Mile High"

  || "Rock"||"ies"   --> "Rockies"

  (abuttal) **abc** = "Rock"

  abc"ies"   --> "Rockies"

- Arithmetic Expressions

  + - * / % (int division) // (remainder)

  ** (power) Prefix - Prefix+

Operators & Expressions

- Comparative Expressions
  - Normal **= \= <> >< > < >= <=**
    - comparison is case sensitive
    - leading/trailing blanks removed before compare
    - shorter strings padded with blanks on right
- Strict **== \== >> << >>= \<< <<= \>>**
    - comparison is case sensitive
    - if 2 strings = except one is shorter, the shorter string is less than the longer string
- Logical Expressions

  & | &&

  \ (preceding expression)

Note: the "not" sign and backslash " \ " are synonymous

Numbers

- A Rexx character string that includes 1 or more decimal digits with an optional decimal point
  - May have leading and trailing blanks
  - Optional sign + or -
  - An "E" specifies exponential notation
    - Be careful with device addresses such as 1E00 (use quotes)

- Precision in calculations may be controlled by the NUMERIC DIGITS instruction
  - Default is 9 digits

- Examples (could also be enclosed in quotes):

12 **-17.9** **+ 7.9E5**

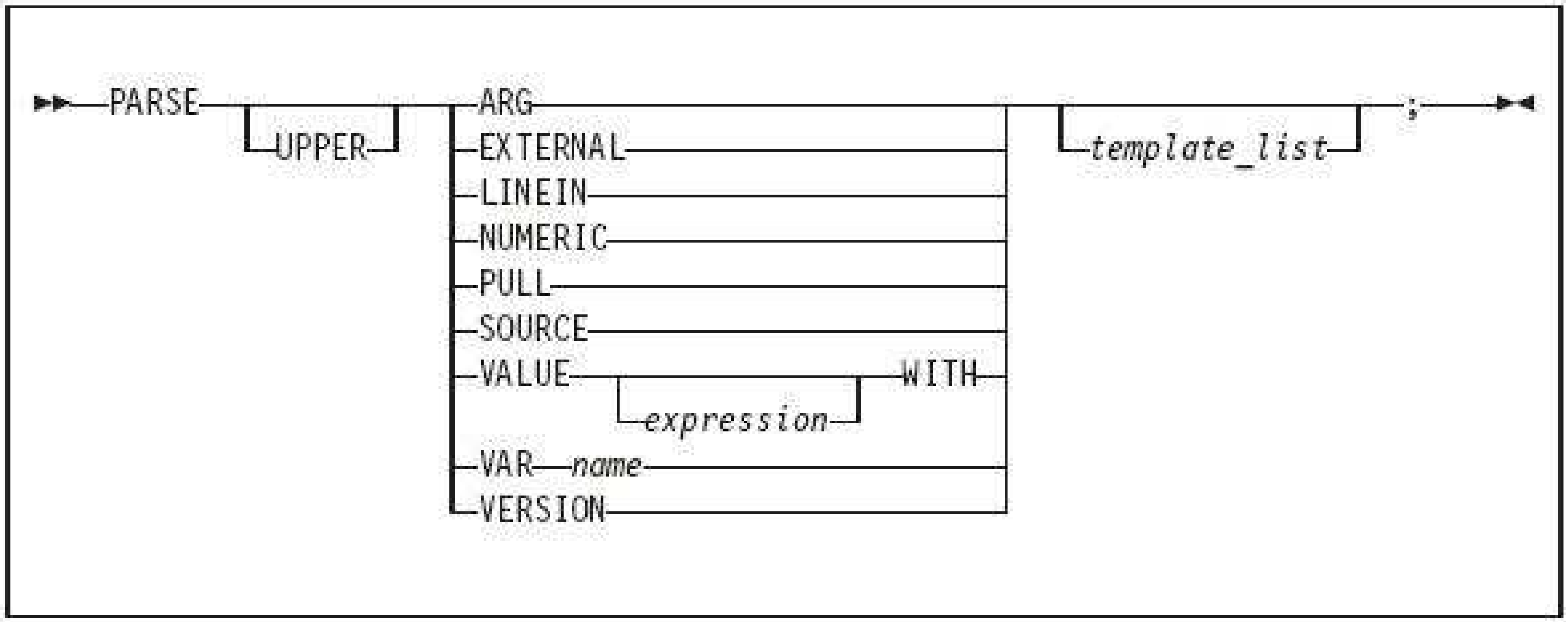
Variables

- Data known by a unique name whose value may change
- Variable names
  - **NOT** case sensitive
  - **Cannot** begin with a digit 0-9
- Defined by assignment (give it a value)

population = 184627

- Variables with no assigned value will have the uppercase variable name as its initial value
- Special variables: **rc, result, sigl**
  - may be set automatically during program execution

Parsing Strings



- Parse Arg – takes data passed into exec or internal routine
  - (see example on “Input and Output” chart)
- Parse Var – parses variable into other variable(s)

Parsing Strings...

- Assigns data to variables using parsing rules

```
str1 = '04-08 May 2009'
parse var str1 w1 w2 w3
```

  - w1 = 04-08
  - w2 = May
  - w3 = 2009

```
parse upper var str1 w1 . w2
```

  - w1 = 04-08
  - w2 = 2009

```
parse var str1 w1 w2
```

  - w1 = 04-08
  - w2 = May 2009

Parsing Strings…

- Default token delimiter is a blank
  - ▶ May be changed on Parse statement

```
str1 = '04-08*May*2009'
parse var str1 w1 '*' w2 '*' w3
```

- w1 = 04-08
- w2 = May
- w3 = 2009

Exercise 2: Say, Pull, & Passing Parameters

- Assume a card deck with suits of Hearts, Diamonds, Clubs, and Spades
- Write a Rexx program to:
  - **pass in** 1 of the 4 suits as an argument
  - **prompt** for a number from 2-10
  - **display** the number and the suit in the format:

```
'Your card is a 10 of Hearts'
```

- **Run** the program with different suits and numbers

Exercise 2: Say, Pull, & Passing Parameters - Answer

```
/*  */
parse arg suit
say 'Enter a number from 2-10:'
pull num
say 'Your card is a 'num' of ' suit
```

Tracing

- trace All
- trace Commands
- trace Error
- trace Failure
- trace Intermediates
- trace Labels
- trace Normal
- trace Off
- trace Results
- trace Scan

- output identifier tags:

`*-*`	source of a single clause
`>>>`	result of expression
`>.>`	value assigned to placehldr
`+++`	error messages

- prefixes if TRACE Intermediates in effect:

`>C>`	data is compound variable
`>F>`	data is result of func call
`>L>`	data is a literal
`>O>`	data is result of operation on 2 terms
`>P>`	data is result of prefix op
`>V>`	data is contents of variable

Tracing (cont.)

- Prefix Options **!** and **?** modify tracing and execution

  ? controls interactive debugging

  TRACE ?Results

  ! inhibits host command execution

  TRACE !C causes command to be traced but not processed

- CMS command **SET EXECTRAC ON** allows you to switch tracing on without modifying the program

- **TS** and **TE** immed commands turn tracing on/off asynchronously

Tracing - Example

- Program

```
/* Trace Sample Program */
Trace Intermediates
number = 1/7
say number
```

- Output

```
    3  *-* number = 1/7
   >L>    "1"
   >L>    "7"
   >O>    "0.142857143"
   4  *-* say number
   >V>    "0.142857143"
0.142857143
```

Exercise 3: Tracing and Debugging

The following Rexx Programs are on your VM A-disk:

- REXXEX3A.EXEC
- REXXEX3B.EXEC

There is something wrong with each program

- Using the TRACE instruction, debug each problem
- Fix the code so that it functions properly

Exercise 3: Tracing and Debugging – Answer A

Trace Intermediate output:

```
 6 *-* string1 = "Rexx" 'Lab'
        >L>    "Rexx"
        >L>    "Lab"
        >O>    "Rexx Lab"
      7 *-* say string11
        >L>    "STRING11"
STRING11
      9 +++ string2 = "Exerc"||"ise'say string2
Error 6 running REXXTR5A EXEC, line 9: Unmatched "/*" or quote
```

Exercise 3: Tracing and Debugging – Answer A

Corrected Rexx Program:

```
Trace I

string1 = "Rexx" 'Lab'
say string1

string2 = "Exerc"||"ise"
say string2
```

Result:

```
 6 *-* string1 = "Rexx" 'Lab'
      >L>   "Rexx"
      >L>   "Lab"
      >O>   "Rexx Lab"
     7 *-* say string1
      >V>   "Rexx Lab"
Rexx Lab
     9 *-* string2 = "Exerc"||"ise"
      >L>   "Exerc"
      >L>   "ise"
      >O>   "Exercise"
    10 *-* say string2
      >V>   "Exercise"
Exercise
```

Exercise 3: Tracing and Debugging – Answer B

Trace Intermediate output:

```
  7 *-* Var1 = "25 35 71"
      >L>   "25 35 71"
    8 *-* /* INPUT: Three positive integers */
    9 *-* /* OUTPUT: The average of these three values */
   12 *-* parse arg w1 . w2 w3
      >>>   ""
      >.>   ""
      >>>   ""
      >>>   ""
   13 *-* w3 = 25
      >L>   "25"
   15 *-* $average = (w1 + w2 + w3) // 3
      >V>   ""
      >V>   ""
   15 +++ $average = (w1 + w2 + w3) // 3
Error 41 running REXXTR5B EXEC, line 15: Bad arithmetic conversion
```

Exercise 3: Tracing and Debugging – Answer B

Corrected Rexx Program:

```
Trace I
Var1 = "25 35 71"

/* INPUT: Three positive integers */
/* OUTPUT: The average of these three values */
parse var var1 w1 w2 w3
/* w3 = 25 */

$average = (w1 + w2 + w3) / 3
say "The average value of these numbers is" $average "."
```

Exercise 3: Tracing and Debugging – Answer B

Result:

```
    7 *-* Var1 = "25 35 71"
      >L>   "25 35 71"
     8 *-* /* INPUT: Three positive integers */
     9 *-* /* OUTPUT: The average of these three values */
    12 *-* parse var var1 w1 w2 w3
      >>>   "25"
      >>>   "35"
      >>>   "71"
    13 *-* /* w3 = 25 */
    15 *-* $average = (w1 + w2 + w3) / 3
      >V>   "25"
      >V>   "35"
      >O>   "60"
      >V>   "71"
      >O>   "131"
      >L>   "3"
      >O>   "43.6666667"
    16 *-* say "The average value of these numbers is" $average "."
      >L>   "The average value of these numbers is"
      >V>   "43.6666667"
      >O>   "The average value of these numbers is 43.6666667"
      >L>   "."
      >O>   "The average value of these numbers is 43.6666667 ."
The average value of these numbers is 43.6666667 .
```

Control Constructs – DO…END

DO … END can be used to create a code block

```
if wins > losses then
   do
     say 'Congratulations!'
     say 'You have won!'
   end
else say 'Sorry, you have lost'
```

Control Constructs - Selection

```
if wins > losses then say 'you have won'
                 else say 'you have lost'

select
  when wins > losses then say 'winner'
  when losses > wins then say 'loser'
  otherwise say ‘even’
end


select
  when wins > losses then say 'winner'
  when losses > wins then say 'loser'
  otherwise NOP
end
```

Control Constructs – DO Loops

```
do forever
  say 'You will get tired of this'
end


do 3
  say "Roll, Roll, Roll the dice"
end


do i=1 to 50 by 1
  say i
end
```

More DO Loops

```
i=30
do until i > 21      /* Evaluate after DO executes */
  i=i+5
end
say i           ──→35


i=30
do while i < 21      /* Evaluate before DO executes */
  i=i+5
end
say i           ──→30
```

Iterate and Leave

- **Iterate** causes a branch to end of control construct

```
 do i=1 to 4
    if i=2 then iterate
   say i
end                    /* displays 1, 3, 4  */
```

- **Leave** exits the control construct

```
 do i=1 to 4
   say i
   if i=3 then leave
end                    /* displays 1, 2, 3  */
```

Symbols and Stems

- Constant symbol starts with a digit (0-9) or period:

  77   .123   12E5

- Simple symbol does not start with a digit nor contains periods:

  ABC   ?3

- Compound symbol contains at least one period, and at least 2 other characters
  - ► **Stem** (up to 1st period), followed by **tail**

  ABC.3   Array.i   Total.name   x.y.z

Built-In Functions

- Parentheses always needed in function calls even if no arguments are required
- Some commonly used functions:

▶ `ABS(-1.674)`	→	`1.674`
▶ `C2D('a')`	→	`129`
▶ `DATE('U')`	→	`'09/19/07'`
▶ `DELSTR('abcde',3,2)`	→	`'abe'`
▶ `D2X(129,2)`	→	`'81'`
▶ `LENGTH('abcdef')`	→	`6`
▶ `POS('day','Wednesday')`	→	`7`
▶ `RIGHT('12',4,'0')`	→	`'0012'`
▶ `SUBSTR('abc',2)`	→	`'bc'`
▶ `WORDS('are we done yet?')`	→	`4`
▶ `WORDPOS('the','now is the time')`	→	`3`

Exercise 4: Sorting Cards - using stems and control constructs

- Write the program CARDSORT EXEC

- Set a variable called rank in your program to represent the possible card values and order:

  rank= '2 3 4 5 6 7 8 9 10 J Q K A'

- Cardsort takes an argument of 2 or more (up to 13) words representing the values of playing cards
  - ►**HINT:** you may want to parse the input args into a stem variable

- Sort the input values in descending order

Exercise 4: Sorting Cards - Answer

```
/* */
rank='2 3 4 5 6 7 8 9 10 J Q K A'
parse upper arg hand             /* get input args      */
num=words(hand)                  /* count how many      */
do i=1 to num
  parse var hand item.i hand     /* place in stem       */
end
do i=1 to num                    /* loop through stems  */
  do j=i+1 to num                /* compare against rank*/
    if wordpos(item.j, rank) > wordpos(item.i,rank)
       then do                   /* sort them           */
             temp=item.j
         item.j=item.i
         item.i=temp      /*after loop item.i has > num*/
       end
    end j
    hand = hand item.i
end
say hand
```

Subroutines & Procedures

- **CALL** instruction is used to invoke a routine
  - ► May be an internal routine, built-in function, or external routine
- May optionally return a result

```
RETURN expression
```

  - ► variable **result** contains the result of the expression
- Parameters may be passed to the called routine

```
CALL My_Routine  parm1
```

…which is functionally equivalent to the clause:

```
NewData = My_Routine(parm1)
```

- Variables are global for subroutines, but not known to procedures unless passed in or EXPOSE option used

Subroutine Example: Returning a Value

```
/* subroutine call example */
x = 5
y = 10
Call Calc x y                   /* call subroutine Calc */
If result > 50 Then
  say "Perimeter is larger than 50"
Else
  say "Perimeter is smaller than 50"
exit

Calc:                          /* begin subroutine     */
Parse Arg len width            /* input args           */
return 2*len + 2*width         /* calculate perimeter  */
                               /* ...and return it     */
```

Issuing Commands from Rexx

- Issuing commands is a way to send a message or request to some unit external to the Rexx program

- Environment is selected by default on entry to a Rexx program
  - ▶ **ADDRESS** instruction can change the active environment
  - ▶ **ADDRESS( )** built-in function used to get name of the currently selected environment

Issuing z/VM Commands from Rexx

- CMS commands issued as a quoted string:
  - ▶ **'STATE PROFILE EXEC'**

- Use DIAG function to issue CP commands with Diagnose x'08‘
  - ▶ **DIAG(8,'QUERY CPLEVEL')**
  - ▶ Can be an expression as part of a longer statement

Issuing Commands – z/VM Example

```
Address CMS             /* send cmds to CMS */
 'STATE PROFILE EXEC'



 If RC=0 Then           /* file found */
    'COPY PROFILE EXEC A TEMP = ='


Parse Value diag(8,'QUERY CPLEVEL') With queryout
```

Issuing Commands – TSO Example

```
"CONSOLE ACTIVATE"
...

ADDRESS CONSOLE  /* change environment to CONSOLE for all commands */
"mvs_cmd"
...
"mvs_cmd"

ADDRESS TSO tso_cmd    /* change environment to TSO for one command */
...
"mvs_cmd"

ADDRESS TSO    /* change environment to TSO for all commands */
"tso_cmd"
...

"CONSOLE DEACTIVATE"
```

Manipulating Files

- Input and output managed as streams
  - Default (terminal input and display)
  - File or dataset
  - Reader
  - Punch
  - Printer
  - Program stack

Manipulating Files – Input (z/VM)

- Input with function call LINEIN
  - LINEIN(name,line,count)
  - answer = LINEIN(name,line,count)
  - CALL LINEIN name,line,count
    - Name
      - name of input stream
    - Line
      - line number to be read
      - Default is current position in stream
    - Count
      - 1 – read 1 line and advance the read position (default)
      - 0 – read no lines, set read position at beginning of specified line

Manipulating Files – Output (z/VM)

- Output with function call LINEOUT
  - LINEOUT(name,string,line)
  - answer = LINEOUT(name,“Sample line”,line)
  - CALL LINEOUT name,string,line
    - String
      - line of data to be written
    - Line
      - line number to write
    - Name
      - name of input stream
  - Output stream opened automatically on first LINEOUT call, closed implicitly at end of program (unless closed explicitly first)

Manipulating Files – TSO

- EXECIO Command
  - Read or write information to a dataset
  - Update an existing dataset
  - EXECIO * DISKR MYINDD (FINIS **STEM MYVAR**
    - *
      - entire dataset (# of lines)
    - DISKR
      - read
    - MYINDD
      - dataset name
    - FINIS
      - close dataset after reading
    - STEM MYVAR
      - place contents of dataset into stem variable MYVAR
  - EXECIO is also a CMS command, usable from z/VM Rexx

Using Pipelines with Rexx

- PIPE is a command that accepts *stage* commands as operands
  - Stages separated by a character called a *stage separator*
    - Default char is vertical bar **|** **(x'4F')**

- Allows you to combine programs so the output of one serves as input to the next
  - Like pipes used for plumbing: data flows through programs like water through pipes!

- User-written stages are Rexx programs
  - Reads in data, works on it, places it back into pipe

Using Pipelines with Rexx - Examples

- Invoking from CMS command line:

```
pipe < profile exec | count lines | console
```

- Invoking from an Exec:

```
/*  Count number of lines in exec */
'PIPE < profile exec | count lines| console'
exit
```

  - /* or…on multiple lines */

```
'PIPE < profile exec',
         '| count lines',
         '| console'
```

Using Pipelines with Rexx - Examples

- Invoking commands and putting output in a stem:

```
'pipe',
'CMS LISTFILE * EXEC A',      /* issue cmd   */
  '| STEM response.'          /* stem output */


do i = 1 to response.0
      say response.i
end
```

Exercise 5: MYDISKS EXEC

- Write a Rexx program to show which disks your userid has accessed

1. **Call a subroutine** that:
   - Uses a PIPE to **issue** CMS command **QUERY DISK** and **save** response
   - **Determine** the number of disks accessed
   - **Return** the value to the main routine
2. **Display** the returned number of disks accessed
3. **Display** each of the disks that are accessed
4. **Issue** the CMS command **QUERY DISK** without using a PIPE
5. **Verify** that output from Steps 3 and 4 match

Exercise 6: WHATCP EXEC

- Write Rexx program WHATCP EXEC to show z/VM CP Level information
  - Issue CP command QUERY CPLEVEL to display CP level
  - Use Rexx Diag function to issue QUERY CPLEVEL command
    - Parse command output to display CP Version, Release, and Service level

Exercise 5: MYDISKS EXEC – Answer #1

```
/* Find Number of disks accessed and list them */
Call GetDisks
Say 'This user has' NumDisks 'disks accessed.'
Say '  '

Do i = 1 to Numdisks
   Say DiskList.i
End

Say '  '
ADDRESS CMS
'QUERY DISK'
Exit

/* Subroutine: Get list of disks and return number of disks accessed*/
GetDisks:
   'PIPE',
     'CMS QUERY DISK',
     '| Drop 1',
     '| STEM DiskList.'
    NumDisks = DiskList.0
Return NumDisks
```

Exercise 5: MYDISKS EXEC – Answer #2

```
/* Find Number of disks accessed and list them */
Call GetDisks
Say 'This user has' NumDisks 'disks accessed.'
Say '  '

Do i = 1 to Numdisks
   Say DiskList.i
End

Say '  '
ADDRESS CMS
'QUERY DISK'
Exit
/*Subroutine: Get list of disks and return number of disks accessed*/
GetDisks:
   'PIPE',
     'CMS QUERY DISK',
     '| Drop 1',
     '| STEM DiskList.',
     '| count lines',
     '| var NumDisks'
Return NumDisks
```

Exercise 6: WHATCP – Answer

```
/* Display CP Level information for the z/VM system */
'CP QUERY CPLEVEL'

Parse value diag(8,'QUERY CPLEVEL') with ,
                 . . version . release . . servicelvl .

say 'z/VM Version = ' version
say 'z/VM Release = ' release
say 'Service Level = ' servicelvl
```

For More Information…

- **Websites:**

http://www.ibm.com/software/awdtools/rexx/	Rexx webpage
http://www.ibm.com/software/awdtools/netrexx/	NetRexx
http://www.ibm.com/software/awdtools/obj-rexx/	Object Rexx
http://regina-rexx.sourceforge.net/	Regina Rexx

- **z/VM publications:**
  - Rexx/VM Reference - SC24-6113
  - Rexx/VM User's Guide - SC24-6114
  - website for library downloads: **http://www.vm.ibm.com/library/**

- **z/OS publications:**
  - TSO/E Rexx User's Guide - SC28-1974
  - TSO/E Rexx Reference - SC28-1975
  - website for library downloads: http://www-304.ibm.com/jct03004c/servers/s390/os390/bkserv/r9pdf/tsoe.html

- **Rexx Compiler**
  - Products ordered separately from z/VM:
    - REXX/370 Compiler, 5695-013
    - REXX/370 Library, 5695-014

- **Other books:**

The Rexx Language	ISBN 0-13-780651-5
The Netrexx Language	ISBN 0-13-806332-X

- **List servers:**
  - http://listserv.uark.edu/scripts/wa.exe?A0=ibmvm

Contact Information

John Franciscovich
(607) 429-3574
francisj@us.ibm.com

Phil Smith III
(703) 476-4511
phil@voltage.com