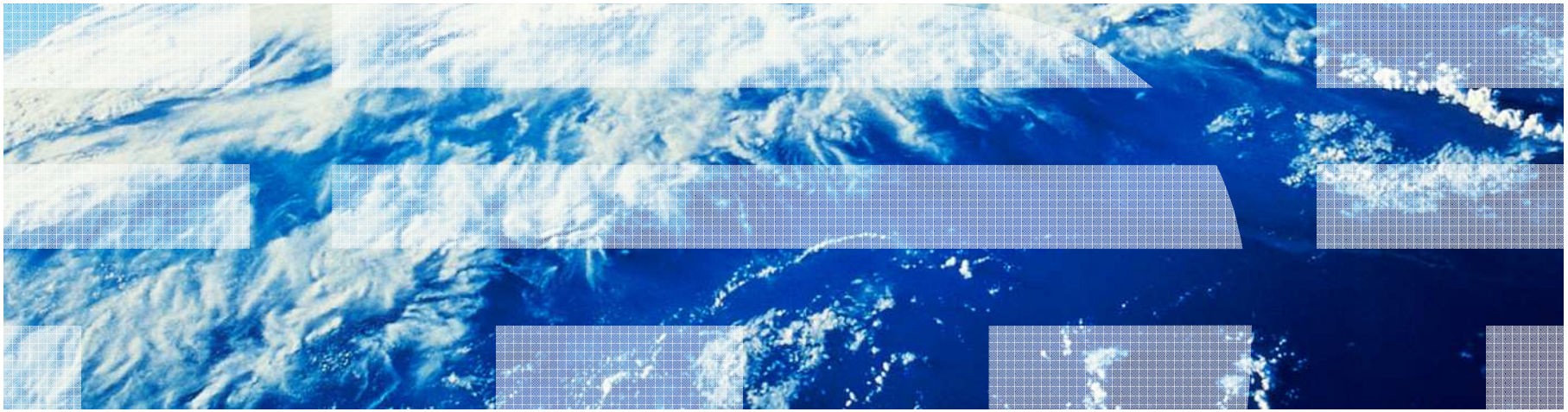
Introduction to REXX Workshop

John Franciscovich
francisj@us.ibm.com



Trademarks

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

z/VM® z10™ z/Architecture® zEnterprise™

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:

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

The information contained in this document has not been submitted to any formal IBM test and is distributed on an "AS IS" basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

In this document, any references made to an IBM licensed program are not intended to state or imply that only IBM's licensed program may be used; any functionally equivalent program may be used instead.

Any performance data contained in this document was determined in a controlled environment and, therefore, the results which may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environments.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming or services in your country.

Agenda

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

Rexx Overview

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

- Rexx is a procedural, general purpose language
  - Intuitive - easy to use and read
  - Many uses, ranging from:
    - Personal tools and utilities
      - For example, frequently used command sequences
    - Complex applications and licensed programs
  - Available on many IBM and non-IBM platforms

- Rexx is designed to be *interpreted*
  - Each program statement translated and executed as the program runs
  - Programs can also be *compiled* to improve
    - Performance
    - Security
    - Change control

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 Platforms

- IBM Platforms
  - VM
  - TSO/E (z/OS)
  - VSE
  - AIX
  - OS/2
- Object Rexx
  - Object-Oriented Rexx supporting many utilities for a UNIX-type environment, including Linux for System z
- Regina Rexx
  - Rexx interpreter ported to most UNIX platforms, including Linux
- NetRexx
  - Blend of Rexx and Java; compiles into Java classes
- Language concepts are the same on all platforms
  - Minor differences such as file names and structure
  - Operating system-specific tools that support Rexx

(See references page for website information)

Creating Rexx Programs: z/VM

- 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

Executing Rexx Programs: z/VM

- 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

Creating and Executing Rexx Programs: TSO/E

- REXX exec can be a sequential data set or a PDS member
- TSO/E EXEC command to invoke a REXX program or a CLIST
- Three ways to use the EXEC command:
  - Explicit execution:
    EXEC dataset(member) 'parameters' operands
  - Implicit execution:
    membername parameters
  - Extended implicit execution:
    %membername parameters

- Search includes:

  //SYSEXEC DD concatenation
  then
  //SYSPROC DD concatenation for **membername** on the command line

Lab Exercises: What to Expect…

1. Run an existing Rexx program to create temporary disk space

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 program to obtain z/VM CP level information
   (issues commands and Diagnose 8)

5. Write a program to locate an available CMS file mode and define a temporary disk at that file mode
   (uses a subroutine, Pipelines, stems, and various control constructs)

Helpful Hints for 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 EXEC 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)

Logging on to the z/VM Lab System

- 3270 Session
- Userids and Passwords
  - PIPUSR01 – PIPUSR25
- Password same as userid

Exercise 1: Create Temp Disk Space

1. Logon to your VM lab userid
2. Issue command **QUERY DISK** to see which disks are accessed
3. Run existing exec **GETTEMP *mode*** to
   - create a temporary disk at filemode *mode*
   - copy existing EXEC programs from a-disk to new temp disk
4. Issue **QUERY DISK** again – notice new disk at *mode*
5. Issue command **FILELIST * * *mode***
6. Run **GETTEMP** again with mode ***a***
7. Issue **QUERY DISK** again – notice new disk at mode ***a***
8. **LOGOFF**

Exercise 1: Create Temp Disk Space

```
/* Get Temporary disk space */

/* File mode of temporary disk is input argument */

 parse upper arg fmode rest
 If (fmode = '') | (rest ¬= '') then
  Do
    say ''
    say 'ERROR:  Input parm is FILEMODE.'
    say ''
    exit 4
  End

 'CP DETACH 555'              /* Get rid of old disk */
 'CP DEFINE T3390 555 2'      /* Define 2 cylinders of temp space */

 queue 1                      /* Answer YES to FORMAT prompt   */
 queue TMP555                 /* Disk label is TMP555          */
 'FORMAT 555 'fmode           /* Format the disk for CMS files */

 If (fmode = 'A') Then        /* If input mode is "A" move A disk to B */
   Do
     Parse Value Diag(8,'QUERY 'UserId()) With thisuser .
     'access VMSYSU:'thisuser'. b/a'
     frommode = 'b'
   End
 Else frommode = 'a'

'COPYFILE * exec ' frommode '= =' fmode /* COPY existing EXEC files
                                        to new temp disk          */
 exit 0
```

Exercise 1: Create Temp Disk Space - Answer

```
query disk

LABEL  VDEV M  STAT   CYL TYPE BLKSZ   FILES  BLKS USED-(%) BLKS LEFT  BLK TOTAL
-      DIR  A   R/W     -    - 4096       44             -          -          -
MNT190 190  S   R/O   115 3390 4096      694      14562-70       6138      20700
MNT19E 19E  Y/S R/O   355 3390 4096     1875      49995-78      13905      63900

gettemp z

HCPDTV040E Device 0555 does not exist
DASD 0555 DEFINED
DMSFOR603R FORMAT will erase all files on disk Z(555). Do you wish to continue?
Enter 1 (YES) or 0 (NO).
DMSFOR605R Enter disk label:
DMSFOR733I Formatting disk Z
DMSFOR732I 2 cylinders formatted on Z(555)

query disk

LABEL  VDEV M  STAT   CYL TYPE BLKSZ   FILES  BLKS USED-(%) BLKS LEFT  BLK TOTAL
-      DIR  A   R/W     -    - 4096       44             -          -          -
MNT190 190  S   R/O   115 3390 4096      694      14562-70       6138      20700
MNT19E 19E  Y/S R/O   355 3390 4096     1875      49995-78      13905      63900
TMP555 555  Z   R/W     2 3390 4096       19         60-17        300        360
```

Exercise 1: Create Temp Disk Space – Answer..

```
gettemp a

DASD 0555 DETACHED
DASD 0555 DEFINED
DMSFOR603R FORMAT will erase all files on disk A(555). Do you wish to continue?
Enter 1 (YES) or 0 (NO).
DMSFOR605R Enter disk label:
DMSFOR733I Formatting disk A
DMSFOR732I 2 cylinders formatted on A(555)
B (VMSYSU:PIPUSR00.) R/O

query disk

LABEL  VDEV M  STAT   CYL TYPE BLKSZ   FILES  BLKS USED-(%) BLKS LEFT  BLK TOTAL
TMP555 555  A   R/W      2 3390 4096       19         60-17        300        360
-      DIR  B/A R/O      -    - 4096       44             -          -          -
MNT190 190  S   R/O    115 3390 4096      694      14562-70       6138      20700
MNT19E 19E  Y/S R/O    355 3390 4096     1875      49995-78      13905      63900
```

Rexx Language Syntax

- Case **In**sensitivity

  Atlanta is the same as **a**tlanta

  - 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`

Operators & Expressions

- String Expressions

(blank) "CNN" "Center" --> "CNN Center"

|| "At"||"lanta" --> "Atlanta"

(abuttal) **abc** = "At"

abc"lanta" --> "Atlanta"

- Arithmetic Expressions

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

** (power) Prefix - Prefix+

Input and Output

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

```
say 'Five Euros equals ' ,
5 * 1.40 'USD'
```

- **pull**
  - prompts for input from the user

```
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
```

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):

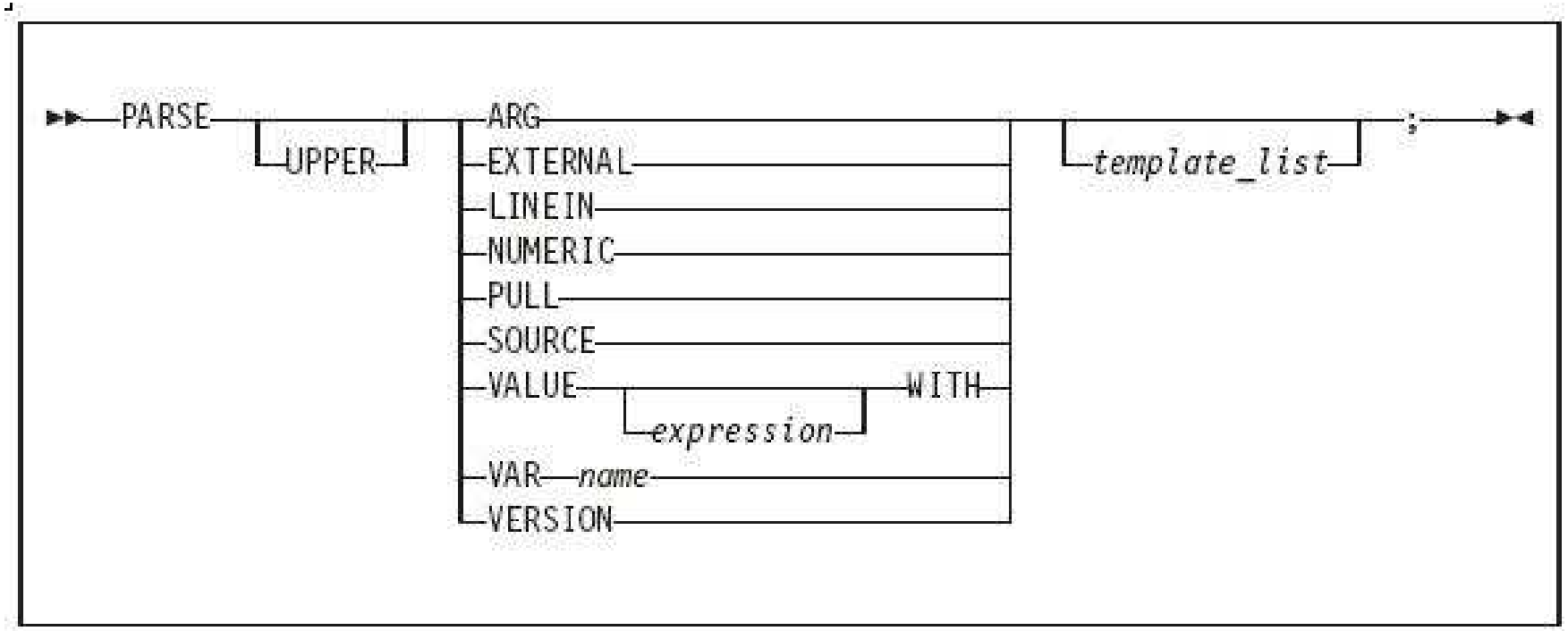
12 -17.9 + 7.9E5

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 = 'March 11-16, 2012'
parse var str1 w1 w2 w3
```

  - w1 = March
  - w2 = 11-16,
  - w3 = 2012

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

  - w1 = MARCH
  - w2 = 2012

```
parse var str1 w1 w2
```

  - w1 = March
  - w2 = 11-16, 2012

Parsing Strings…

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

```
str1 = 'March*11-16,*2012'
parse var str1 w1 '*' w2 '*' w3
```

  - w1 = March
  - w2 = 11-16,
  - w3 = 2012

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** - clauses before execution
- **Trace Commands** - commands before execution. If the command has an error, then also displays the return code
- **Trace Error** - any command resulting in an error after execution and the return code
- **Trace Failure/Normal** – default setting, any command with a negative return code after execution, and the return code
- **Trace Intermediates** – Trace All, plus intermediate results during evaluation of expressions and substituted names
- **Trace Labels** - only labels passed during execution
- **Trace Off** - traces nothing and resets options
- **Trace Results** – Trace All, plus results of an evaluated expression and values assigned during PULL, ARG, and PARSE instructions
- **Trace Scan** – Trace All, but without the clauses being processed

Tracing (cont.)

- 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 REXXTR3A EXEC, line 9: Unmatched "/*" or quote
```

Exercise 3: Tracing and Debugging – Answer A

Corrected Rexx Program:

```
Trace I

string1 = "Rexx" 'Lab'
say string1                    /* Was: say string11             */

string2 = "Exerc"||"ise"       /* Was: 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 *-* Nums = "25 35 71"
        >L>   "25 35 71"
      9 *-* parse arg w1 . w2 w3
        >>>   ""
        >.>   ""
        >>>   ""
        >>>   ""
     11 *-* $average = (w1 + w2 + w3) // 3
        >V>   ""
        >V>   ""
     11 +++ $average = (w1 + w2 + w3) // 3
DMSREX476E Error 41 running REXXTR3B EXEC, line 11: Bad arithmetic conversion
```

Exercise 3: Tracing and Debugging – Answer B

Corrected Rexx Program:

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

parse var Nums w1 w2 w3              /* Was: parse arg w1 . w2 w3   */

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

Exercise 3: Tracing and Debugging – Answer B

Result:

```
      7 *-* Nums = "25 35 71"
        >L>   "25 35 71"
      9 *-* parse var Nums w1 w2 w3
        >>>   "25"
        >>>   "35"
        >>>   "71"
     11 *-* $average = (w1 + w2 + w3) / 3
        >V>   "25"
        >V>   "35"
        >O>   "60"
        >V>   "71"
        >O>   "131"
        >L>   "3"
        >O>   "43.6666667"
     12 *-* 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 .
```

Symbols and Stems

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

  77 .123 12E5

- Simple symbol does not start with a digit and does not contain 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

Symbols and Stems…

```
/* Stems as arrays */

  do i=1 to 50 by 1
    array.i = i+5
  end
  say array.25          /* Output: "30" */
  say array.51          /* Output: "ARRAY.51" */


/* Stems as records */

  If attendee.payment == "LATE" then
  do
    say attendee.$fullname
    say attendee.$email
    say attendee.$company.telephone
  end
```

Issuing Commands from Rexx

- CP and CMS commands can be issued as a quoted string:
  - **`'CP QUERY CPLEVEL'`**
  - **`'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

- 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 Commands – z/VM Example

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


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


               /* Save command output in variable */
Parse Value diag(8,'QUERY CPLEVEL') With queryout
```

Issuing Commands – TSO

```
"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"
```

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'

/*  or … on multiple lines  */
  'PIPE < profile exec',
        '| count lines',
        '| console'
```

Using Pipelines with Rexx - Examples

- Invoking commands and parsing output into a stem:

```
'pipe',
'CMS LISTFILE * EXEC A',   /* issue cmd          */
  '| SPECS 1 1 ,           /* parse first word */
  '| STEM response.'       /* save in stem      */



do i = 1 to response.0
   say response.i          /* display file names */
end
```

Exercise 4: 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 4: 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
```

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, Leave, and Exit

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

```
do i=1 to 4
  if i=2 then iterate
  say i
end
```

```
1, 3, 4
```

Leave exits the control construct and continues the REXX program

```
do i=1 to 4
   say i
   if i=3 then leave
end
```

```
1, 2, 3
```

Exit exits the REXX program unconditionally

```
i=1
do forever
   say i
   if i=3 then exit
   i=i+1
end
```

```
1, 2, 3
```

Built-In Functions

```
ABS(-1.674)
/* absolute value */
```
→ 1.674

```
C2D('a')
D2X(129,2)
/* char to decimal, dec to hex*/
```
→ 129
→ '81'

```
DATATYPE('10.5','W')
DATATYPE('12  ')
/* determines if a string matches a provided type */
```
→ '0'
→ 'NUM'

```
DATE('U')
/* date function */
```
→ '10/04/11'

```
LENGTH('abcdef')
/* length of the string */
```
→ 6

Built-In Functions

```
POS('day','Wednesday')
```
→ 7

```
/* starting position of substr inside a string */
```

```
RIGHT('12',4,'0')
```
→ '0012'

```
/* pad 12 out to 4 characters with 0's */
```

```
SUBSTR('abcdef',2,3)
```
→ 'bcd'

```
/* obtain substring of 3 characters beginning at second character */
```

```
WORDS('are we done yet?')
```
→ 4

```
/* return number of tokens inside a given string */
```

```
WORDPOS('the','now is the time')
```
→ 3

```
/* return position of a given substring */
/* inside a string                      */
```

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     */
```

Exercise 5: GETTMODE EXEC

- Write Rexx program GETTMODE to locate the first available file mode (A-Z) and create a temporary disk at that file mode

1. **Call a subroutine** that will:
   - Use a PIPE to **issue** CMS command **QUERY SEARCH** to obtain the used modes (file mode is 3rd word of response) – **save in stem**
   - **Build a string** of used modes from the **output stem** of the PIPE
   - **Create a string** of possible file modes (A-Z)
   - **Build a stem** containing the possible file modes
   - **Mark** the used file modes "unavailable" in the list of possible modes
     - *(Hint: blank out the unavailable modes in the list)*
   - **Locate** the first available mode and **return** it to the main program

2. **If a file mode is returned:**
   - **Issue commands** to define and format a temporary disk at the returned mode

     (Hint: Use commands from GETTEMP EXEC)

Exercise 5: GETTMODE EXEC – Answer (1 of 3)

```
/* Get temporary disk space and access it at an available file mode  */

'CP DETACH 555'              /* Get rid of old disk */

/* Call subroutine Findmode to locate the first available file mode.   */
/* Once found, define a temporary disk and format and access it at     */
/* the returned file mode.                                             */

Call Findmode

If rtnmode <> 0 Then
  Say 'Temp disk will be accessed at mode' rtnmode
Else
  Do
    Say 'No Filemodes available for temp disk'
    Exit 8
  End

'CP DEFINE T3390 555 2'      /* Define 2 cylinders of temp space */

queue 1                      /* Answer YES to FORMAT prompt   */
queue TMP555                 /* Disk label is TMP555          */
'FORMAT 555 'rtnmode         /* Format the disk for CMS files */

Exit rc
```

Exercise 5: GETTMODE EXEC – Answer (2 of 3)

```
/* Subroutine Findmode will locate the first available (A-Z) file mode.*/
/* and return it in variable rtnmode. If no file modes are available,  */
/* rtnmode will be set to zero.                                         */
Findmode:
   'PIPE',
     'CMS QUERY SEARCH',
     '| SPEC WORDS 3 1',
     '| STEM usedmode.'

/* Build string of accessed file modes                                */
acc_modes = ''
Do I = 1 TO usedmode.0
   acc_modes = acc_modes || SUBSTR(usedmode.I,1,1)
END

/* Build stem containing all possible file modes                       */
possible_modes = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
Do i = 1 TO 26
   modelist.i = SUBSTR(possible_modes,i,1)
End

/* Remove all accessed file modes from possible file mode list         */
mlength = LENGTH(acc_modes)
Do n = 1 TO mlength
      Do i = 1 TO 26
         If (SUBSTR(acc_modes,n,1) = modelist.i) Then
           Do
              modelist.i = ' '
              Leave
           End
      End
End
```

Exercise 5: GETTMODE EXEC – Answer (3 of 3)

```
/* Locate the first possible file mode that is "available" and          */
/* return it                                                           */

foundmd = 'NO'
Do i = 1 TO 26
   If modelist.i ¬= ' ' Then
     Do
       rtnmode = modelist.i
       foundmd = 'YES'
       Leave
     End
End

/* If no file modes available, return zero                             */

If foundmd = 'NO' Then
   rtnmode = 0

Return
```

For More Information…

- **Websites:**
  - http://www.ibm.com/software/awdtools/rexx/ Rexx webpage
  - http://www.ibm.com/software/awdtools/netrexx/ Netrexx
  - http://www-01.ibm.com/software/awdtools/rexx/opensource.html 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://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/Shelves/IKJOSE10?filter=rexx

- **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