Rexx Language Coding Techniques Part 1

Tracy Dean in IBM

June 2022



Agenda

- Part 1
 - Rexx products
 - External environments and interfaces
 - Instructions, functions, and subroutines
 - Variable visibility
 - Parsing
- > Part 2
 - Rexx compound variables vs. data stack
 - I/O
 - Troubleshooting
 - Programming style and techniques
 - Other Rexx products and projects
- Additional material included in hand-out, not covered in session

Rexx Interpreter and Libraries

- A procedural language
- Indicator to operating system that it's Rexx first line contains:

```
/* Comment (if any) */
```

- The Interpreter executes (interprets) Rexx code "line by line"
 - Included in all z/OS and z/VM releases
- A Rexx library is required to execute compiled programs
 - Compiled Rexx is not an LE language
- Two Rexx library choices:
 - (Runtime) Library a priced IBM product
 - Alternate library a free IBM download
 - Uses the native system's Rexx interpreter
- At execution, compiled Rexx will use whichever library is available

The Rexx Products

- IBM Compiler for Rexx on zSeries Release 4
 - z/VM, z/OS: product number 5695-013
- IBM Library for Rexx on zSeries Release 4
 - z/VM, z/OS: product number 5695-014
- > z/VSE
 - Part of operating system
- IBM Alternate Library for Rexx on zSeries Release 4
 - Included in z/OS base operating system
 - Free download for z/VM (and z/OS)
 - http://www.ibm.com/software/awdtools/rexx/rexxzseries/altlibrary.html
- Rexx Interpreter
 - Included in all z/OS and z/VM releases

Why Use a Rexx Compiler?

- Program performance
 - Known value propagation
 - Assign constants at compile time
 - Common sub-expression elimination
 - stem.i processing
- Source code protection
 - Source code not in deliverables
- Improved productivity and quality
 - Syntax checks all code statements
 - Trace S provides limited syntax checking
 - Flags missing END statements
 - Does not catch syntax errors in If Then statements regarding value comparisons
 - Source and cross reference listings
- Compiler control directives
 - %include, %page, %copyright, %stub, %sysdate, %systime, %testhalt

Rexx External Environments

External Environments

- ADDRESS instruction is used to define the external environment to receive host commands
 - For example, to set TSO/E as the environment to receive commands

Address TSO

- Several host command environments available in z/OS
- A few host command environments available in z/VM

Host Command Environments in z/OS

TSO

- Used to run TSO/E commands like ALLOCATE and TRANSMIT
- Only available to Rexx running in a TSO/E address space
- The default environment in a TSO/E address space
- Example:

```
Address TSO "ALLOC FI(INDD) DA('USERID.SOURCE') SHR"
```

MVS

- Use to run a subset of TSO/E commands like EXECIO
- The default environment in a non-TSO/E address space
- Example:

```
Address MVS "EXECIO * DISKR MYINDD (FINIS STEM MYVAR"
```

Many more

Other z/OS Host Command Environments

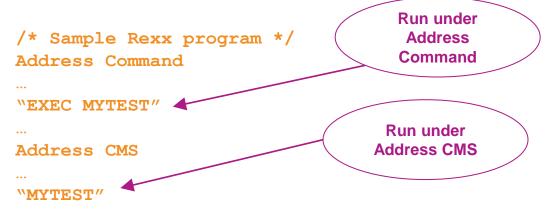
- ISPF services
- ISPF edit macros
- CONSOLE
- LINK, LINKMVS, LINKPGM, ATTACH, ATTCHMVS, ATTCHPGM
- SYSCALL
- SDSF
- DSNREXX

Host Command Environments in z/VM

- CMS (default)
 - Commands treated as if entered on the CMS command line
 - Translation of parameter list
 - Uppercasing and tokenizing
 - Same search order as CMS command line
- COMMAND
 - Basic CMS CMSCALL command resolution
 - No translation of parameter list
 - No uppercasing of tokenized parameter lists
 - To call an EXEC, prefix the command with the word EXEC
 - To send a command to CP, use the prefix CP
- CPICOMM, CPIRR, OPENVM
- Generally, best practice is to use "Address Command" at the top of Rexx EXECs that will be run in CMS environment

Multiple Methods to Specify External Environment

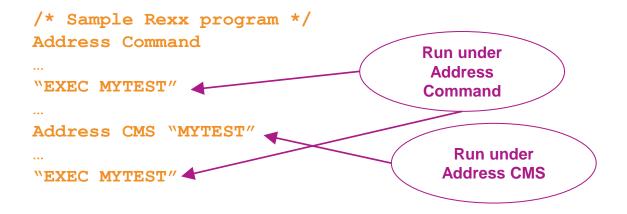
- Initial value, later changed:
 - MYTEST is another Rexx EXEC I'm calling from this program:



All future commands are treated as Address CMS unless specified otherwise

Multiple Methods to Specify External Environment

- Initial value used as default
 - All calls requiring a different value have Address statement
 - MYTEST is another Rexx EXEC I'm calling from this program:



Instructions, Functions, and Subroutines

Instructions vs Functions vs Subroutines

- Keyword instruction
 - One or more clauses
 - First word is a keyword that identifies the instruction

```
Arg, Do, If, Parse, ...
```

- Instruction
 - Statement that performs an assignment of a value to a variable

```
counter = 1
```

- Function
 - Must return a single result string (i.e. often on the right side of equal sign)
 - Built-in provided as part of the Rexx language
 - Internal create your own within the same program
 - External create your own outside this program
- Subroutine
 - Called (similar to a function) but may not return data
 - Returns data in special variable: Result

Functions

- Must return a single result string
 - Often on the right side of an equal sign
- Built-in functions too many to list, so a few examples
 - Absolute value of a number

```
total = -3
newtotal = Abs(total)
→ 3
```

Left justify a string

```
fullname = Left('Tracy Dean',20)

→ 'Tracy Dean'
```

Determine the type of data

```
If Datatype(amtowed,N)=1
  Then amtowed = amtowed * 1.1
  Else Say 'Amount owed is invalid'
```

Today's date or day of the week
 today = Date()

```
→ 25 October 2021
```

```
todaydow = Date('W')
→ Friday
```

Find a string within another string

```
startcol = Pos('day','Tuesday')
  → 5
startcol = Pos('x','Tuesday')
  → 0
```

Subroutines

- Multi-step task to execute multiple times
 - Write once, use multiple times
 - Make code easier to read
- Call a subroutine, pass and return variables

```
Call CalcInterest amountborrowed amountdue = Result
```

Value from Return instruction in special variable called "Result"

- Define a subroutine
 - After Exit instruction of main program
 - Start with name of subroutine followed by colon
 - End with Return instruction

```
CalcInterest:
Parse Arg principal
...
total = principal + (principal * rate)
Return total
```

Declaration and Visibility of Variables

Variable Declaration

- Rexx is a procedural language
- Variables are not declared
 - Initial value is same as variable name in uppercase
 - Seen as a string unless perform mathematical operation
 - Requires the value at the time be valid for the operation
 - Type of data assigned to a variable can change within a program
 - Valid:

```
total = 'Here is some text'
Say 'Here is the original total:' total
total = 5+3
newtotal = 1.5 * total
Say 'Here is the new total:' newtotal
```

Output:

```
Here is the original total: Here is some text Here is the new total: 12.0
```

Visibility of Variables

- Variables can be visible throughout a program
 - Visible within Functions and Subroutines you create within the program
 - No need to pass, declare or expose them
 - Not visible in Procedures unless specifically exposed
- Programming practice
 - Functions, Subroutines, and Procedures use different variable names for reusability
 - Pass the value of variables on the call
 - Function, Subroutine or Procedure will parse the value and assign to its own variables
 - Pass values back to main program via Return statement
 - For subroutines and procedures, values are visible to calling routine via Result variable

Visibility of Variables in Functions and Subroutines

- Using existing variables
 - In a function

```
principal = 100
interest = 0.10
totaldue = CalcTotalDue()
...
Exit
...
CalcTotalDue:
total = principal*(1 + interest)
Return total
```

Both are valid for functions and subroutines

- Creating new variables
 - In a subroutine

```
loan = 100
rate = 0.10
Call CalcTotalDue(loan rate)
totaldue = Result
...
Exit
...
CalcTotalDue:
Parse Arg principal interest .
total = principal*(1 + interest)
Return total
```

More common to use separate variables in the function or subroutine

Does not effect variable

of same name

in main program

Visibility of Variables in Procedures

No visibility of variables from main program unless specifically requested – most common

```
Variable used for
                                 something else in
total = 855 ◀
                                   the program
                                           Value from Return
loan = 100
                                             statement in
rate = 0.10
                                             Procedure
Call CalcTotalDue(loan rate)
Say 'This is your total due: 'Result
Say 'Principal & interest:' principal interest
                                    Indicates no exposure of
Exit
                                  variables from main program
CalcTotalDue: Procedure
                                            Value of anything
Parse Arg principal interest .
                                             here becomes
total = principal*(1 + interest)
                                            value of "Result"
                                                variable
Return total
```

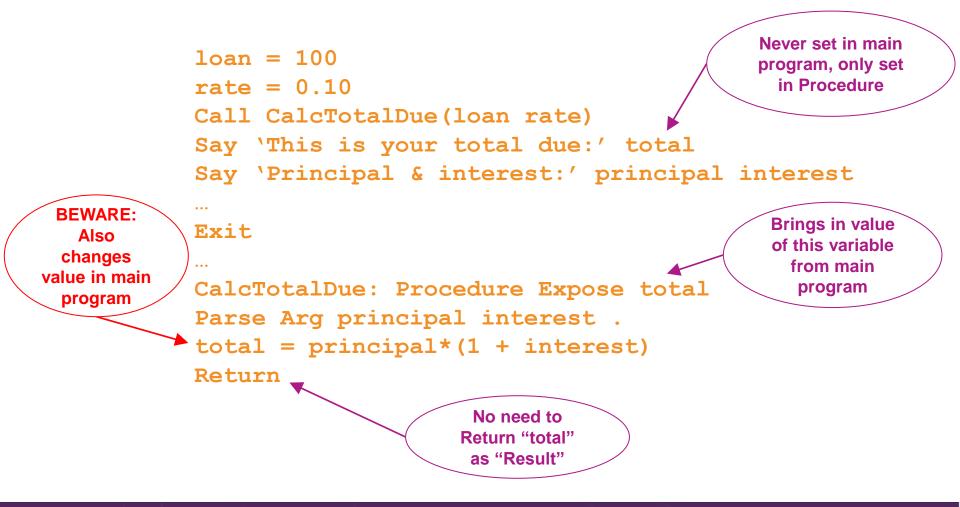
Visibility of Variables in Procedures

No visibility of variables from main program unless specifically requested

specifically requested Variable used for something else in the program Output: This is your total due: 110.00 Principal & interest: PRINCIPAL INTEREST est Doe effect of same name in main program CalcTotalDue: Procedure Value of anything Parse Arg principal interest . here becomes total = principal*(1 + interest) value of "Result" variable Return total

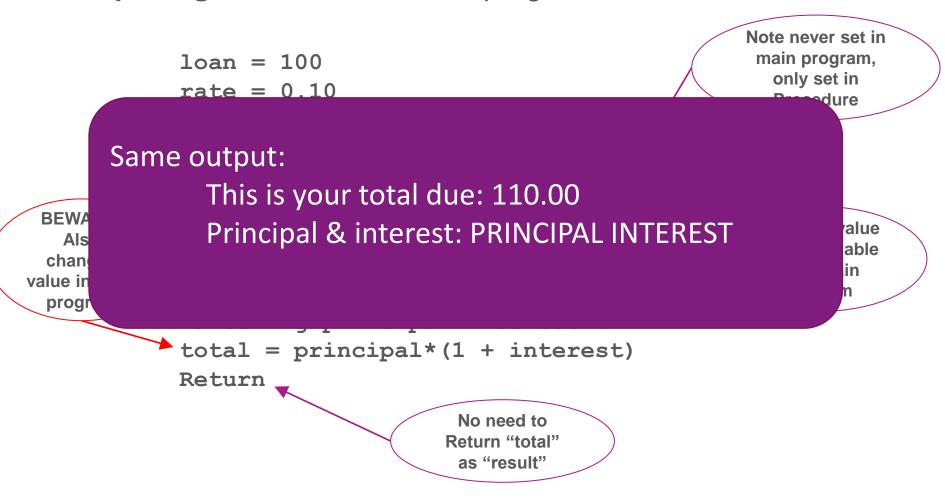
Visibility of Variables in Procedures

Exposing variables from main program to a Procedure



Visibility of Variables in Procedures

Exposing variables from main program to a Procedure



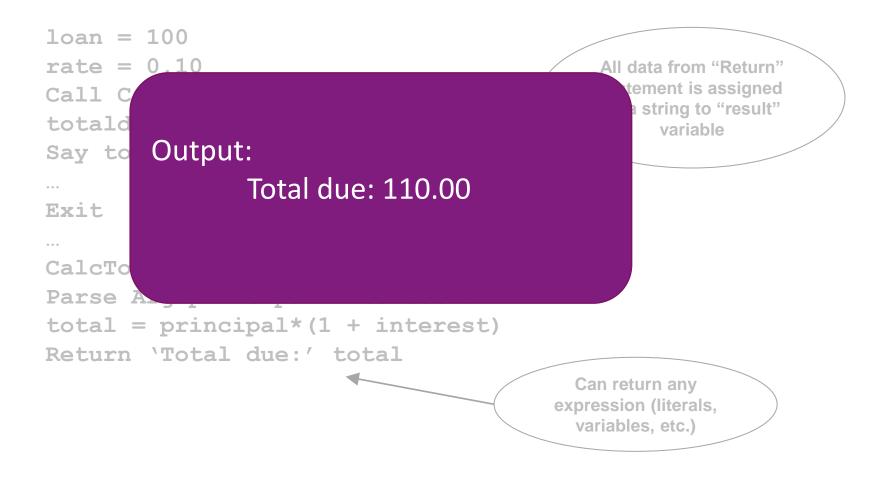
Returning Variables from Functions and Procedures

Returning variables from a Procedure to the main program

```
loan = 100
rate = 0.10
                                                All data from "Return"
                                                statement is assigned
Call CalcTotalDue(loan rate)
                                                as a string to "Result"
totaldue = Result 	
                                                     variable
Say totaldue
Exit
CalcTotalDue: Procedure
Parse Arg principal interest .
total = principal*(1 + interest)
Return 'Total due: ' total
                                              Can return any
                                            expression (literals,
                                              variables, etc.)
```

Returning Variables from Functions and Procedures

Returning variables from a Procedure to the main program



Calling a Subroutine vs External Routine

- From within a Rexx EXEC, call another Rexx EXEC
 - As an external routine

```
'EXEC CALCINTEREST' amountborrowed

If rc=0 Then

Parse Pull amountdue
```

Processed as a variable unless in quotes. Be aware of case sensitivity

As a subroutine

```
Call CalcInterest amountborrowed amountdue = Result
```

Calling a Subroutine vs External Routine

- External calls
 - Receive data
 - Parameters (aka arguments)
 - On the stack
 - Return data
 - In a return code (variable rc)
 - Must be a whole number
 - On the stack



No variable visibility

- Subroutines
 - Receive data
 - Parameters (aka arguments)
 - Visible variables
 - On the stack
 - Return data in
 - Result variable
 - Other visible variables
 - On the stack



Controlled variable visibility

Parsing

Keyword Instruction: Parse

- Parse
 - Allows the use of a template to split a source string into multiple components
- Short forms to some of these instructions exist
 - NOT RECOMMENDED
 - But you may see them in another user's code you must maintain
 - ARG
 - Short form for Parse Upper Arg
 - PULL
 - Short form for Parse Upper Pull

Parse Templates

- Simple template
 - Divides the source string into blank-delimited words and assigns them to the variables named in the template
 - The last variable gets the rest of the string exactly as entered

```
datastring = ' Write the blank-delimited string '
Parse Var datastring firstvar secondvar thirdvar fourthvar

firstvar -> 'Write'
secondvar -> 'the'
thirdvar -> 'blank-delimited'
fourthvar -> ' string '
```

Parse Templates – Blank Delimiter

- Simple template
 - A period (aka a dot) is a placeholder in a template
 - A "dummy" variable used to collect unwanted data
 - Notice the consecutive single quotes so the single quote is recognized as part of the string

```
datastring = 'Last one gets what''s left'
Parse Var datastring firstvar . secondvar
firstvar -> "Last"
secondvar -> "gets what's left"
```

Often used at the end of Parse statement to take "the rest of the data"

```
datastring = 'Last one gets what''s left'
Parse Var datastring firstvar secondvar .
firstvar -> "Last"
secondvar -> "one"
```

Causes the last variable to get the last word without leading and trailing blanks

```
datastring = ' Write the blank-delimited string '
Parse Var datastring firstvar secondvar thirdvar fourthvar .
firstvar -> 'Write'
secondvar -> 'the'
thirdvar -> 'blank-delimited'
fourthvar -> 'string'
```

Parse Templates – Literal or Variable Delimiter

- String pattern template
 - A literal or variable string pattern indicating where the source string should be split
 - Assumes blank-delimited if no other pattern specified

```
Literal:

Parse Var datastring firstvar '-' secondvar

Variable:

delim = '-'
Parse Var datastring firstvar (delim) secondvar.

Outcome (the same in both cases):

firstvar -> ' Write the blank'
secondvar -> 'delimited'
```

Parse Templates – Positional Delimiter

- Positional pattern template
 - Use numeric values to identify the character positions at which to split data in the source string
 - An <u>absolute</u> positional pattern is a number or a number preceded by an equal sign.

```
datastring = 'Cowlishaw Mike UK '
Parse Var datastring =1 surname =20 chrname =35 country =46 .

surname -> 'Cowlishaw '
chrname -> 'Mike '
country -> 'UK '
```

- A <u>relative</u> positional pattern is a number preceded by a plus or minus sign
 - Plus or minus indicates movement right or left, respectively, from the last match

```
datastring = 'Cowlishaw Mike UK '
Parse Var datastring =1 surname +19 chrname +15 country +11 .

surname -> 'Cowlishaw '
chrname -> 'Mike '
country -> 'UK '
```

Parse Templates . . .

- Positional pattern template removing blanks
 - Specify an <u>absolute</u> positional pattern
 - Insert periods to strip blanks

```
datastring = 'Cowlishaw Mike UK '
Parse Var datastring =1 surname . =20 chrname . =35 country .

surname -> 'Cowlishaw'
chrname -> 'Mike'
country -> 'UK'
```

If data starts in column 1 and is blank-delimited, this is the same as Parse Var datastring surname chrname country

Warning – won't work if any of the data has more than one "word"

```
datastring = 'Cowlishaw, Jr. Mike UK '
Parse Var datastring =1 surname . =20 chrname . =35 country .

surname -> 'Cowlishaw,'
chrname -> 'Mike'
country -> 'UK'

Blank
delimited
```

Summary

- Part 1
 - Rexx products
 - External environments and interfaces
 - Instructions, functions, and subroutines
 - Variable visibility
 - Parsing
- > Part 2
 - Rexx compound variables vs. data stack
 - I/O
 - Troubleshooting
 - Programming style and techniques
 - Other Rexx products and projects
- Additional material included in hand-out, not covered in session

Tracy Dean
Product Manager, IMS Tools and z/VM Tools
tld1@us.ibm.com





Hindi



Traditional Chinese

감사합니다

Korean

Спасибо

Onaorio

Ndzi khense ngopfu

Tsonga

Gracias

Spanish

Russian



Arabic



Italian

Ke a leboha

Tswana



Thank You

Danke German

Merci

French

Obrigado

Brazilian Portuguese



ありがとうございました

Japanese

