Introduction to CLIST and REXX

Lab Book

Document Revision History

Date	Revision No.	Author	Summary of Changes
17-Dec-2008	1.0	Sudhir Karhadkar	Original
11-Jun-2009	1.1	CLS Team	Review
Mar-2013	2	Rajita D	Revamped

Table of Contents

Getting Started

Overview

This lab book is a guided tour for learning REXX. It comprises solved examples and 'To Do' assignments. You can follow the steps provided in the solved examples and work out the 'To Do' assignments given.

Setup Checklist for REXX

Here is what is expected on your machine in order for the lab to work.

Minimum System Requirements

- Intel Pentium 90 or higher
- Microsoft Windows 95, 98, or NT 4.0, 2k, XP.
- Memory: 32MB of RAM (64MB or more recommended)
- Internet Explorer 6.0 or higher

Please ensure that the following is done:

- PASSPORT PC-TO-HOST (mainframe terminal simulator) is installed
- Connectivity to the Mainframe
- Availability of REXX on the mainframe

Instructions

- Create a directory by your emp-code in drive D. In this directory, create and edit your source REXX programs, then upload it to the mainframe.

Learning More (Bibliography)

- IBM REXX User's Guide
- IBM ISPF Edit and Edit Macros Manual
- IBM ISPF Services Guide
- IBM ISPF Dialog Developer's Guide and Reference

Lab 1. Coding and Running a REXX Exec

Goals	• Understand how to write simple REXX Exec. • Learn to run Exec – Explicitly and Implicitly • Passing data to an Exec
Time	60 minutes

1.1: Simple REXX Exec

First create a PDS *Loginid*.REXX.EXEC. Specifications: RECFM FB, LRECL 80, Block-size as 800.

The source REXX programs i.e. exec are stored as members in this PDS.

Solution:

Step 1: Write the following code as a member in the above PDS.

Loginid.REXX.EXEC(REXX1)

```
/* REXX */
SAY 'Hello World !'
EXIT
```

Example 1: Simple REXX code

1.2: How to run REXX Exec – Explicitly

Step 1: Go to the COMMAND (Option 6) of ISPF.

Step 2: Type (on Option 6 screen) -- EXEC 'dsrp050.rexx.exec(rexx1)' EXEC

OR

From the COMMAND line of any ISPF panel (say Edit Entry Panel), Type –
TSO EXEC rexx.exec(rexx1) EXEC

Output is:

```
Hello World !
```

Example 2: Output

1.3A: Displaying string literal

Step 1: Type the following code. Check how the strings are displayed when enclosed in single quotes, double quotes, and not enclosed in quotes.
Let the member name be dsrp050.rexx.exec(rexx1A)

```
/****REXX****/
SAY 'This is a REXX literal string.'
SAY "This is a REXX literal string."
SAY This is a REXX literal string.
EXIT
```

Example 3: REXX literal string

Step 2: Run the Exec explicitly as explained above.

Observe the output and note your findings.

1.3B: Displaying string literal with apostrophe

Step 1: Type the following code. Check how the strings with apostrophe are displayed – apostrophe or two single quotes are used.
Let the member name be dsrp050.rexx.exec(rexx1B)

```
/****REXX****/
SAY “This isn't a CLIST instruction.”
SAY 'This isn''t a CLIST instruction.'
EXIT
```

Example 4: Sample code

Step 2: Run the Exec explicitly.

Observe the output.

1.4A: Format: Continuing an instruction

Step 1: Type the following code. Check how the instruction is continued to the next line by using comma.
Let the member name be dsrp050.rexx.exec(rexx1C)

```
/****REXX****/
SAY 'This is an extended',
```



```
     'REXX literal string.'
EXIT
```

Example 5: Sample code

Step 2: Run the Exec explicitly.

Observe the output.

Output is:

```
This is an extended REXX literal string.
```

Example 6: Output

A space is added between “extended” and “REXX”.

1.4B: Format: Continuing an instruction

Step 1: Type the following code. Check how the instruction is continued on next line using comma.
Let the member name be dsrp050.rexx.exec(rexx1D)

```
/****REXX****/
SAY 'This is',
     'a string.'
SAY 'This is' 'a string.‘
EXIT
```

Example 7: Sample code

Step 2: Run the Exec explicitly.

Observe the output and note your findings.

1.4C: Format: Continuing an instruction

Step 1: Type the following code. Check how to continue a literal string without adding a space.
Let the member name be dsrp050.rexx.exec(rexx1E)

```
/****REXX****/
SAY 'This is a string that is bro'||,
      'ken in an awkward place.‘
EXIT
```

Example 8: Sample code

Step 2: Run the Exec explicitly.

Observe the output and verify it is as follows.

Output is:

```
This is a string that is broken in an awkward place.
```

Example 9: Output

1.4D: Try another program.

Step 1: Type the following code. Check how to continue a literal string without adding a space.
Let the member name be dsrp050.rexx.exec(rexx1F)

```
/****REXX****/
SAY 'This is' ||,
      'a string.'
SAY 'This is' || 'a string.'
EXIT
```

Example 10: Sample code

Step 2: Run the Exec explicitly.

Observe the output and verify if it is as follows.

Output is:

```
This isa string.
```

Example 11: Output

1.4E: Invoking TSO command

Step 1: Type the following code. REXX exec can invoke TSO command; it can accept data from user using PULL instruction. This program executes TIME command of TSO. Let the member name be dsrp050.rexx.exec(rexx1G).

```
/****REXX****/
/*  This Exec accepts time and uses TIME command  */
Game1:
SAY 'What time is it?'
PULL usertime            /* Put the user's responses into
                         a variable called usertime  */
IF usertime = '' '' THEN       /* User did not enter the time */
  SAY "O.K. Game's over."
ELSE
   DO
     SAY "The computer says:"
     /* TSO System */  TIME  /* command */
  END
EXIT
```

Example 12: TSO command

Step 2: Run the Exec explicitly.

Observe the output.

1.4F: (To do)

Write a REXX exec which accepts three numbers from the user. Display the sum of the numbers. The exec should prompt the user to enter the number. (To do)

1.4G: (To do)

Write a REXX exec which accepts the user name and displays welcome message for the user. (To do)

1.5A: How to run REXX Exec – Implicitly

The PDS containing REXX exec *Loginid*.REXX.EXEC needs to be allocated to ddname SYSEXEC.

Step 1: Create the member SETUP in *Loginid*.REXX.EXEC as follows.

Loginid.REXX.EXEC(SETUP)

```
"EXECUTIL SEARCHDD(yes)"
"ALLOC FILE(SYSEXEC) DATASET(REXX.EXEC) SHR REUSE"

IF RC = 0 THEN
  SAY 'ALLOCATION TO SYSEXEC COMPLETED.'
ELSE
  SAY 'ALLOCATION TO SYSEXEC FAILED.'
```

Example 13: Sample code

Step 2: Run this exec explicitly.

Now, the system exec library SYSEXEC will be searched when execs are implicitly invoked. This happens due to the statement EXECUTIL SEARCHDD(yes).

Step 3: Go to the COMMAND (Option 6) of ISPF.

Step 4: Type (on Option 6 screen) -- rexx1

OR

From the COMMAND line of any ISPF panel (say Edit Entry Panel), Type – **TSO** rexx1

To reduce the search time for an exec that is executed implicitly and to differentiate it from a TSO/E command, precede the member name with a %:

TSO %rexx1 or
%rexx1 (from Option 6)

1.5B: Run all previous examples implicitly. Note the difference.

1.6: Interpreting Error Messages

While running an exec that contains an error, an error message is displayed with corresponding line of exec

Step 1: Type the following code (which contains an error).

```
/* REXX */
SAY 'ENTER NAME'
PULL NM              /* GET THE NAME
SAY 'YOUR NAME: ' NM
```

Example 14: Sample code

Step 2: Execute the exec either explicitly or implicitly. Analyze the error message.

The code has an error. The exec runs until it detects the error, a missing */ at the end of the comment. The exec ends with an error message as given below.

```
ENTER NAME
3 +++ PULL NM              /* GET THE NAME
SAY 'YOUR NAME: ' NM
IRX0006I Error running REXX1E, line 3: Unmatched "/*" or quote
***
```

Example 15: Error message

The first line of error message begins with the line number of the statement where the error was detected, followed by three pluses (+++) and the contents of the statement. The second line of error message begins with the message number followed by a message containing the exec name, line where the error was found, and an explanation of the error.

1.7A: Passing data to an exec: PULL instruction

The exec displays the message to the user to enter the data. Then PULL instruction accepts the data from the user into the variables.
Please refer to the assignments 1.4E, 1.4F, 1.4G.

1.7B: Passing data to an exec: PULL instruction for multiple values

The PULL instruction can extract multiple values on a single statement.

Step 1: Type the following code.

```
/* REXX */
SAY 'ENTER TWO NUMBERS'
PULL N1 N2
SUM = N1 + N2
SAY 'THE SUM IS ' SUM'.'
```

Example 16: Sample code

Step 2: Run the exec either explicitly or implicitly.
Output is:

```
ENTER TWO NUMBERS
40 50
 THE SUM IS  90.
 ***
```

Example 17: Output

1.7C: Specifying input values (numbers) when invoking the exec

Step 1: Type the following code.

```
/* REXX */
ARG N1 N2
SUM = N1 + N2
SAY 'THE SUM IS ' SUM'.'
```

Example 18: Sample code

Step 2: Execute this exec.
Explicit: EXEC rexx.exec(rexx17C) '50 40' EXEC
Implicit: rexx17c 50 40

These values are called *arguments*.

Output is:

```
        THE SUM IS  90.
        ***
```

Example 19: Output

1.7D: Specifying input values (string) when invoking the exec

Step 1: Type the following code. It uses ARG instruction.

```
/*  REXX  */
ARG NM1 NM2
SAY 'WELCOME ' NM1 " " NM2
```

Example 20: Sample code

Step 2: Execute this exec.
Implicit: rexx17d Sachin Tendulkar

Output is:

```
WELCOME  SACHIN   TENDULKAR
         ***
```

Example 21: Output

Observe that the name is displayed in capitals, and not as entered by the user.

1.7E: Specifying input values (string) when invoking the exec

Step 1: Type the following code. It uses PARSE ARG instruction.

```
/*  REXX  */
PARSE ARG NM1 NM2
SAY 'WELCOME ' NM1 " " NM2
```

Example 22: Sample code

Step 2: Execute this exec.
Implicit: rexx17e Sachin Tendulkar

Output is:

```
WELCOME  Sachin   Tendulkar
         ***
```

Example 23: Output

Observe that the name is displayed as entered by the user.

Lab 2. Operators

Goals	• Learn how Arithmetic, Comparison and Logical operators work • Understand Operator priority
Time	45 minutes

2.1: Using operators

Step 1: Type the following code.

```
/****REXX ****/
PARSE ARG season snowing broken_leg
IF ((season = 'winter') | (snowing ='yes')) & (broken_leg ='no')  THEN
                    SAY 'Go skiing.'
ELSE
                    SAY 'Stay home.‘
EXIT
```

Example 24: Sample code

Step 2: Run the exec. Arguments passed are "spring yes no".
Run the exec. Arguments passed are "spring no no".
Run the exec with different combinations of arguments.

2.2: Using comparison and logical operators

Step 1: Type the following code**.**

```
/****REXX ****/

SAY (4>2) & (a=a)       /* true, so result is 1*/
SAY (2>4) & (a=a)       /* false, so result is 0*/
SAY (4>2) | (5=3)       /* at least one is true, so result is 1*/
SAY (2>4) | (5=3)       /* neither one is true, so result is 0*/
SAY (4>2) && (5=3)      /* only  one is true, so result is 1*/
SAY (2<4) && (5=5)      /* both are true, so result is 0*/
SAY \0                  /* opposite of zero, so result is 1*/
SAY \(4>2)              /* opposite of true, so result is 0*/ EXIT
```

Example 25: Sample code

Step 2: Run the exec and observe the output.

2.3: Using concatenation operators

Step 1: Type the following code.

```
/****REXX ****/
sport = 'base'
equipment = 'ball'
column = ‘     ’
cost = 5
SAY  sport||equipment  column  '$' cost
EXIT
```

Example 26: Sample code

Step 2: Run the exec and observe the output.

Lab 3. Control Flow Constructs

Goals	• Learn how to use IF..THEN..ELSE and nested IF .. statements • Learn how to code simple loops and conditional loops • Understand LEAVE instruction
Time	90 minutes

3.1: IF .. THEN .. ELSE

Step 1: Type the following code. Accept the variables “weather” and “tenniscourt” in the following exec as command line arguments.

```
        IF weather = ‘fine’ THEN
          DO
                SAY 'What a lovely day!'
                IF tenniscourt = ‘free’ THEN
                      SAY 'Shall we play tennis?'
                ELSE NOP
          END
        ELSE
          SAY 'Shall we take our raincoats?‘
```

Example 27: Sample code

Step 2: Run the exec with different values of arguments.

3.2: Loop using control variable

Step 1: Type the following code.

```
        /****REXX****/
        DO number = 1 TO 10  BY 2
                SAY 'Loop '  number
                SAY 'Hello ..!!'
        END
        SAY ‘Came out of loop when number reached ' number
EXIT
```

Example 28: Sample code

Step 2: Run the exec and observe the output.

3.3: Leave instruction

Step 1: Type the following code.

```
        /****REXX****/
        DO number = 1 TO 5
                SAY 'Loop '  number
                 IF number > 2 THEN
                    LEAVE
                 ELSE
                    SAY 'Hello ..!!'
        END
        SAY ‘Came out of loop when number reached ' number
EXIT
```

Example 29: Sample code

Step 2: Run the exec and observe the output.

3.4: Nested DO Loops

Step 1: Type the following code.

```
        /****REXX****/
        DO outer = 1 TO 2
    DO inner = 1 TO 2
                IF inner > 1 THEN
                   LEAVE inner
                ELSE
                   SAY 'HIP'
    END
    SAY 'HURRAH'
       END
EXIT
```

Example 30: Sample code

Step 2: Run the exec and observe the output.

3.5: (To do)

Write a loop as DO NUM = 10 To 8 …… END. Put a SAY instruction inside the loop. Observe what happens to the loop. Display the value of NUM after the loop. Note your conclusion.

3.6: (To do)

Accept a password from the user. If it equals to “PATNI123”, then display “Password is correct”, else display “Wrong password”. If user enters incorrect password three times, then display “Not allowed to continue”. Otherwise for correct password, display the number of attempt, as First/Second/Third, as the case may be.

3.7: (To do)

Write two nested loops. The outer loop should be DO outer = 1 To 3 ….. END. The inner loop should be DO inner = 1 TO 3 ….. END. For every iteration of the outer loop, only one iteration of the inner loop should be performed. Control variable value for inner loop has to be DO inner = 1 TO 3. Put appropriate SAY instructions in the loops.

3.8: (To do)

Accept “employee-name” and “salary” from the user. Compute the tax as follows.

If the salary is less than 50001, then tax is zero
If the salary is greater than 50000and less than 60001,
    then the tax is 10% of (salary – 50000)
If the salary is greater than 60000 and less than 100001,
    then the tax is 1000 + 20% of (salary – 60000)
If the salary is greater than 100000, then the tax is 9000 + 30% of (salary – 100000)

Form a string of name, salary, tax, and then display the string.

Repeat this till the user says no more employees.

Lab 4. Functions and Subroutines

Goals	• Learn how to use built-in and user defined functions • Learn how to code subroutines • Learn how to pass and receive data from/to functions, subroutines
Time	90 minutes

4.1: Coding of a function

Step 1: Type the following code. The function accepts three numbers and returns the maximum of it.

```
/****REXX****/
PARSE ARG number1, number2, number3
IF number1 > number2 THEN
   IF number1 > number3 THEN
      greatest = number1
   ELSE
      greatest = number3
ELSE
   IF number2 > number3 THEN
      greatest = number2
   ELSE
      greatest = number3
RETURN greatest
```

Example 31: Sample code

Step 2: Insert the lines in the beginning to invoke this function. Run it with different integer values.

4.2: Built-in functions

Step 1: Type the following code.

```
/****REXX****/
SAY 'ENTER DATA'
PARSE PULL VAR1
SAY DATATYPE(VAR1)
SAY LENGTH(VAR1)
SAY SUBSTR(VAR1,2,8)
```

Example 32: Sample code

Step 2: Run it with different input values.

4.3: (To do)

Use built-in functions of MAX, MIN, RANDOM, ABS, COMPARE, LENGTH, SUBSTR, DELSTR, USERID.

4.4: Passing data to a subroutine using variables

Step 1: Type the following code.

```
/****REXX****/
number1 = 5
number2 = 10
CALL sub1
SAY answer
EXIT

sub1:
answer = number1 + number2
RETURN
```

Example 33: Sample code

Step 2: Run the exec.

4.5: Passing data to a subroutine: Problems of sharing variables

Step 1: Type the following code.

```
/****REXX****/
number1 = 5
number2 = 10
DO i = 1 TO 5
   CALL sub2
   SAY answer
END
EXIT

sub2:
DO i = 1 TO 5
   answer = number1 + number2
   number1 = number2
   number2 = answer
END
RETURN
```

Example 34: Sample code

Step 2: Run the exec and analyze the output.

4.6: Passing data to a subroutine: PROCEDURE instruction

Step 1: Type the following code.

```
/****REXX****/
number1 = 10
CALL sub3
SAY number1 number2
EXIT


sub3: PROCEDURE
number1 = 7
number2 = 5
RETURN
```

Example 35: Sample code

Step 2: Run the exec and analyze the output. Please note your findings.

4.7: Passing data to a subroutine: PROCEDURE EXPOSE instruction

Step 1: Type the following code.

```
/****REXX****/
number1 = 10
CALL sub4
SAY number1 number2
EXIT

sub4: PROCEDURE EXPOSE number1
number1 = 7
number2 = 5
RETURN
```

Example 36: Sample code

Step 2: Run the exec and analyze the output. Please note your findings.

4.8: Passing data to a subroutine by using arguments

Step 1: Type the following code.

```
/****REXX****/
PARSE ARG long wide
CALL perimeter long, wide
SAY 'The perimeter is ' RESULT
EXIT

perimeter:
ARG length, width
perim = 2 * length + 2 * width
RETURN perim
```

Example 37: Sample code

Step 2: Run the exec accepting parameter values from command line.

4.9: Passing data to a function (To do)

Repeat assignments 4.4 thru 4.8 by using Functions
.

Lab 5. CLIST Assignments

1.1: Write a CLIST program to accept two numbers and display the sum of the two numbers.

1.2: Write a CLIST program to accept a number, check whether it is a prime number, and display appropriate message.

1.3: Write a CLIST program to accept a four digit year, display whether it is a leap year or not. Display appropriate messages for the given conditions.

a. Year accepted not a four digit numeric

b. Year is a leap year

c. Year is not a leap year

1.4: Write a CLIST program to accept AM or PM. If the user enters "AM" the message should be displayed as "Good Morning". If the user enters "PM" the message should be displayed as "Good Evening".

