REXX

BY
Rajith Rajan

History

- Developed By Mike Cowlishaw of IBM.
- Replaced the languages EXEC and EXEC 2
- In 1996 ANSI published a standard for REXX

Introduction –What Is?

- REXX -REstructured eXtended eXecutor (REXX)
- Command level Language.
  - CLIST(Command LIST)
  - REXX

Introduction –What Is?

- REXX -REstructured eXtended eXecutor (REXX)
- Command level Language.
  - CLIST(Command LIST)
  - REXX

Difference b/w CLIST/REXX

- Clist is specially developed for TSO and support only in TSO whereas REXX is used across platform.
- CLIST code is executed by interpreter whereas REXX code is executed by interpreter or as a compiled load module.
- Source code protection is possible in REXX.
- REXX easier to learn compared to CLIST.

System DD & Purpose

DD	Purpose
ISPPLIB	Panel
ISPSLIB	Skeletons
ISPTLIB & ISPTABL	Tables
ISPMLIB	Messages
ISPLLIB	Load
SYSPROC	System CLIST & REXX
SYSEXEC	System REXX
SYSUEXEC	User REXX
SYSUPROC	System CLIST & REXX
SYSHELP	Help Dataset
SYSPROF	Profile Dataset

ISRDDN

- TSO ISRDDN list all the dataset allocated at that point of time.
- Alternatively can use LISTALC.
- Can do many action including edit/browse /view/free/member list/compress

Introduction –Features

- Ease of Use
- Free Format
- Convenient Built-in Functions
- Debugging Capabilities
- Interpreted Language
- Extensive Parsing Capabilities

Writing Rexx Programs

- /* Rexx Program */
- Say “This is my first Rexx Program”

Executing Rexx Programs

- Explicit Ways:

  1. Type EX against member name in the PDS.
  2. TSO EX ‘userid.project.group(mem)’ ‘arg1 arg2’ EXEC

- Impicit Ways:

  1. TSO %mem1 arg1 arg2
  2. TSO ALLOC FI(SYSUEXEC) DA(‘usrid.rexx.dataset’) shr
     TSO ALTLIB ACTIVATE USER(EXEC)
  3. TSO ALTLIB ACTIVATE APPLICATION(EXEC) DA(‘user.tools’)

Writing Rexx –Layout Rules

- There really are very few rules when it comes to coding REXX.
- The system presumes that each line is a new instruction unless you state you are continuing on another line by the use of a comma (we will see this in a minute).
- If you want two instructions on one line you will need to put a semi-colon between them.
- You can indent your instructions as much as you like; this is very useful when you are coding loops.
- You can also have as many blank lines as you like. These can be used to section your code and make it far more readable.

Variable & Stem

- Varible names may contain

  1. A -Z, a -z, 0 -9, !, ?, #, $, @ & _
  2. 0 -9 can not be used for the first character
  3. All lowercase chars are converted to uppercase
  4. Variables are initialized with their name!

- Compound Variables
  1. This type of variable identified by a period in the name.
  2. A compound symbol permits the substitution of variables within its name when you refer to it.
  3. It contain two part STEM and the TAIL
  4. Length before and after substitution cant exceed 250 char.

Operators

OPERATOR	Meaning
>,<,>=,<=	Same meaning as in other language
>>,<<,==	Strict operators
|,&,&&	OR,AND,XOR Respectively
||	Abuttal (Concatenation without space)
*,/,+,-,=	Same meaning as in other language
**	Power Operator. Eg 3**2
//	Returns remainder
%	Integer division
/=,<>, ¬=,\=,><	Boolean Operators.
Prefix \,¬	Logical NOT

Operator Precedence

Operator	Description
+,-,¬,\	Prefix operators
**	Power
*,/,%,//	Multiply and Divide
+,-	add and subtract
||	Concatenation operator
=,>,<	comparison operators
==,>>,<<	Strictly operator
\=,¬=	Not Equal to
><,<>	Greater than or less than
\> ¬>	Not Greater than
\< ¬<	Not Less than
\==,¬==	Strictly Not Equal to
\>> ¬>>	Strictly Not Greater than
\<< ¬<<	Strictly Not less than
>= >>=	Greater than or equal to
<= <<=	Less than or equal to
&	And
|,&&	Or, exclusive or

Precedence

Conditions Statements

- IF-ELSE stmt
- Select stmt

If Else Statement

IF *expression* THEN
instruction
[ELSE]
[instruction]

```
/* REXX IF program */
A = 10 ; B = 12
IF A = B THEN SAY A "IS EQUAL TO" B
ELSE SAY A "IS NOT EQUAL TO" B
------------------------------
10 IS NOT EQUAL TO 12
```

If Else Statement *with Do & END*

```
/* REXX pgm showing DO-END blocks */
A = 76 ; B = 76
SAY "A=" A "B=" B
IF A = B THEN DO
   SAY “AND THEY ARE THE SAME“
END
ELSE DO
   SAY "AND THEY ARE DIFFERENT“
END
```

```
A= 76 B= 76
AND THEY ARE THE SAME
```

Select Statement

```
SELECT
  WHEN *expression THEN*
       *instruction*
  OTHERWISE
       *[instruction]*
END
```

Select Statement Ex

```
/* REXX The SELECT, WHEN, OTHERWISE and NOP
  instructions */
A = "120"
SELECT
  WHEN A = "100" THEN NOP
  WHEN A = "200" THEN SAY "A IS 200"
  WHEN A = "300" THEN DO
        SAY "DO A BLOCK OF CODE IF ITS 300 " END
  OTHERWISE SAY "NONE ABOVE ARE TRUE"
END
```

Do Loop

```
DO [repetitor] [conditional]
 instruction
END
repetitor:
  name=expri TO exprt BY exprb FOR exprf
  FOREVER
  exprr
conditional:
  WHILE exprw
  UNTIL expru
```

Do Loop - Simple DO Group

If you specify neither repetitor nor conditional, the construct merely groups a number of instructions together. These are processed one time.

Do Loop - Repetitive DO Loops

If a DO instruction has a repetitor phrase or a conditional phrase or both, the group of instructions forms a **repetitive DO loop.**

1. **Simple Repetitive Loops**

   A simple repetitive loop is a repetitive DO loop in which the repetitor phrase is an expression that evaluates to a count of the iterations.

ex

```
DO 5
      say 'Hello'
END
```

Do Loop - Repetitive DO Loops

2. **Forever Loop**

   Logically the loop repeat forever, the termination of the loop has to be coded the instruction part of the loop.

ex

```
DO Forever
     t=TIME('R')
say hello
t=TIME('E')
     IF t > 5 then leave
END
```

Do Loop - Repetitive DO Loops

3. **Controlled Repetitive Loops**

   The controlled form specifies a control variable that is assigned an initial value (expri + 0) before the first execution of the instruction list. The variable is then stepped (expri + exprb) before the second and subsequent times that the instruction list is processed while the end condition (determined by the result of exprt) is not met

ex

```
DO I=3 TO -2 BY -1
say i
END
```

Do Loop - Repetitive DO Loops

3. **Controlled Repetitive Loops**

ex

```
    I=0.3
    DO Y=I TO I+4 BY 0.7
        say Y
    END
----------------------------------------------------------------
    DO Y=0.3 TO 4.3 BY 0.7 FOR 3
        say Y
    END
----------------------------------------------------------------
    Do K=1 to 10
        ...
    End k /* Checks that this is the END for K loop */
```

Do Loop – Conditional DO Loops

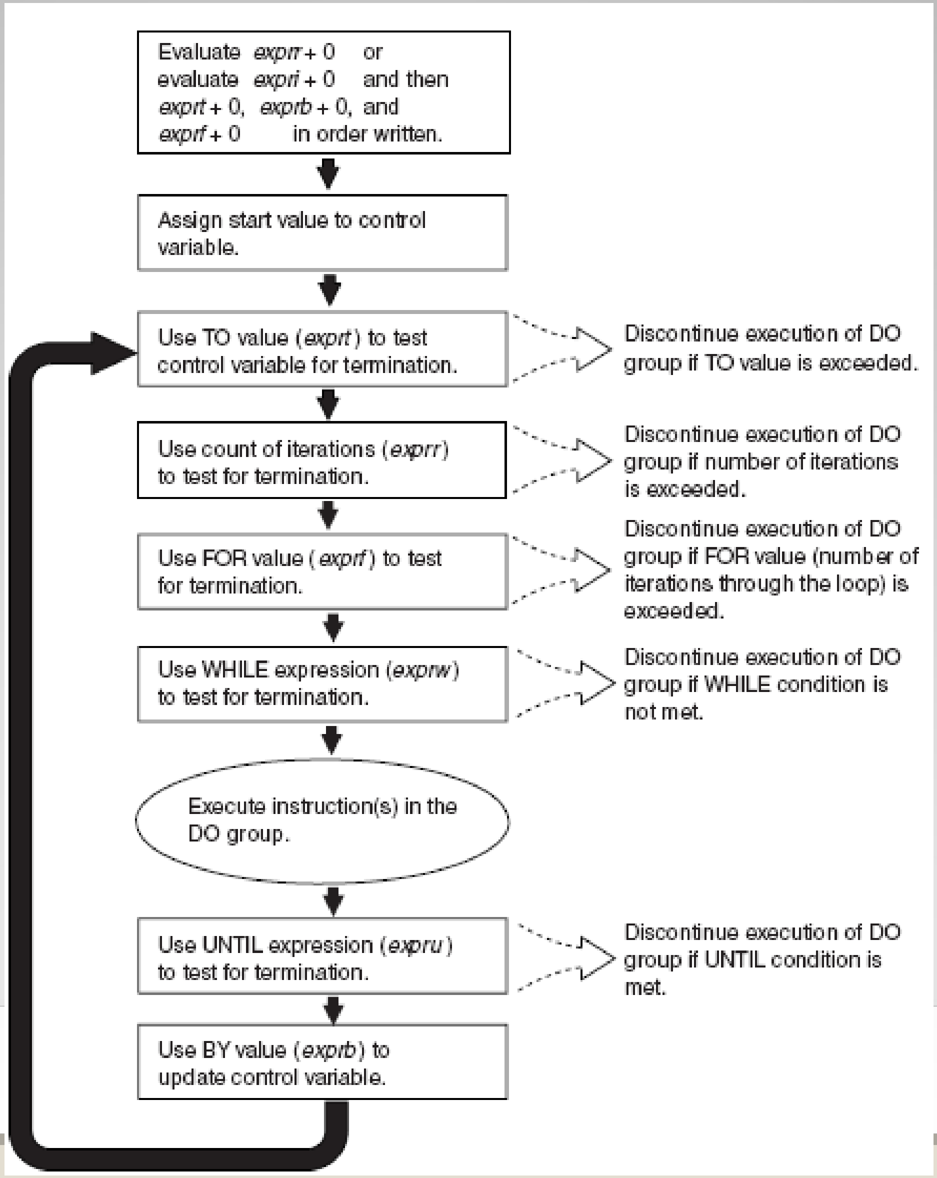
- **Conditional Phrases (WHILE and UNTIL)**

  A conditional phrase can modify the iteration of a repetitive DO loop. It may cause the termination of a loop. It can follow any of the forms of *repetitor*

```
DO I=1 TO 10 BY 2 UNTIL i>6
say i
END

DO I=1 TO 10 BY 2 WHILE i<6
say i
END
```

Flow of a DO Loop


Evaluate *exprr* + 0 or evaluate *expri* + 0 and then *exprt* + 0, *exprb* + 0, and *exprf* + 0 in order written.
Assign start value to control variable.
Use TO value (*exprt*) to test control variable for termination.
Discontinue execution of DO group if TO value is exceeded.
Use count of iterations (*exprr*) to test for termination.
Discontinue execution of DO group if number of iterations is exceeded.
Use FOR value (*exprf*) to test for termination.
Discontinue execution of DO group if FOR value (number of iterations through the loop) is exceeded.
Use WHILE expression (*exprw*) to test for termination.
Discontinue execution of DO group if WHILE condition is not met.
Execute instruction(s) in the DO group.
Use UNTIL expression (*expru*) to test for termination.
Discontinue execution of DO group if UNTIL condition is met.
Use BY value (*exprb*) to update control variable.

Do Loop - ITERATE

- When a ITERATE instruction is encountered execution of the group of instructions stops, and control is passed to the DO instruction.

  ITERATE [*ctrl-var]*

ex

```
DO i=1 TO 4
  IF i=2 THEN ITERATE
  say i
END
```

Do Loop - LEAVE

- LEAVE causes an immediate exit from one or more repetitive DO loops

LEAVE [*ctrl-var]*

ex

```
DO i=1 TO 4
   IF i=2 THEN LEAVE
   say i
END
```

Loops–Interpret

```
/* REXX program to show the power of INTERPRET */
A = "SAY 'Input REXX instructions or END'"
INTERPRET A
DO FOREVER
PULL Input
IF Input = "END" THEN EXIT 0
INTERPRET Input ; END
-----------------------------
Input REXX instructions or END
A = 15
B = 20
SAY A * B
300
END
```

CLIST

- Free format language
- Should be coded in capital letters.
- Continuation symbols are + & -.
- Comments Start with '/*' End with '*/'
- Variable name can be of 256 character.