

Chapter 2

REXX STATEMENTS

*Get on the
Fast Track!*



TM

**SYS-ED/
Computer
Education
Techniques, Inc.**

Objectives

You will learn:

- Variables.
- REXX expressions.
- Concatenation.
- Conditional programming and flow of control.
- Condition traps.

1 Variables

1.1 Variable Name

Variable names may consist of the following characters:

A-Z	a-z								
0-9	@	#	\$?	!	.	_		

The first character cannot be a period or digit.

The maximum size for a variable name is 250 characters.

Variables need to be used properly in order to code an efficient REXX program.

The use of special names, such as RC, SIGL, or RESULT, as variable names will result in errors and unpredictable results.

Example:

This assignment would be confusing.

```
data = data
```

1.2 Simple Variable

A simple variable contains a single data item; its value is set with commands.

- A variable can contain numbers or character strings.
- Variables do not have to be declared.

1.3 Compound Variables

With the REXX programming language, a compound variable typically is referred to as an array; it consists of a list of elements identified by a single name.

A specific value can be established with an assignment statement.

Example:

```
monthtbl.2 = 345  
monthtbl.mon = amt
```

Multi dimensional tables also can be used.

Compound variables often are used with DO loops or repetitive loops.

Example:

```
daily_stats.month.day = amt
```

A stem can be used to initialize the entire array.

Example:

```
monthtbl. = 0
```

2 Expressions

Expressions can be used within a REXX clause consisting of:

Constants	Operators	Comparisons operators	Logical operators
-----------	-----------	-----------------------	-------------------

2.1 Constants

These constants are used to perform arithmetic in REXX.

Whole number	Decimal number	Floating point number	Signed number
--------------	----------------	-----------------------	---------------

2.2 Operators

These operators are used in REXX expressions:

+	Add.
-	Subtract.
*	Multiply.
/	Divide.
%	Divide and return a whole number.
//	Divide and return the remainder only.
**	Exponentiation.

2.3 Comparison Operators

These comparison operators are used in REXX expressions:

=	Equal.
==	Strictly equal.
\=	Not equal.
\==	Not strictly equal.
>	Greater than.
<	Less than.
>=	Greater than or equal.
<=	Less than or equal.
\<	Not less than.
\>	Not greater than.

Strictly equal and not strictly equal requires that all characters including the case, type, and number of spaces be identical.

Example:

```
IF salary > 200000
& age < 25
& sex = 'F' THEN
SAY 'Are you single?'
```

Order of Operation

The precedence rules adhere to the following order of operation:

-+	Prefix Operators.
**	Exponentiation.
* / % //	Multiplication and Division.
+ -	Addition and Subtraction.

Example:

```
/*****REXX*****/
PARSE ARG amount tax
total = amount + (amount * tax)      /* total including taxes */

SAY 'Your total bill is ' total
```

2.4 Logical Operators

Logical expressions return a value of 0 or 1.

The logical operators are:

&	AND
	OR
&&	Exclusive OR
¬	Logical NOT

Example:

```
IF overtime > 100 & evaluation = 'E' THEN
    salary = salary * 1.10
else
    salary = salary * .90

counter = 1
SAY counter = 0          /* What is displayed */
```

3 Concatenation

There are different ways of combining data; the operation is known as concatenation.

The data can be expressions, strings, variables and constants.

Concatenation of is used to format output.

The following operators are available:

Blank	Concatenate and place a single space in between: Example: <code>SAY fld1 fld2</code>
	Concatenate without spaces in between: Example: <code>SAY (10/2) (2*2)</code>
Abuttal	Concatenate and place no blanks in between: Example: <code>SAY amt 'DB'</code>

4 IF/THEN

The IF statement allows instructions to be executed conditionally. Nested IF statements also are permitted.

- When the entire IF is placed on a single line, the ELSE must be separated with a semicolon.
- When more than one instruction is to be executed, use the DO statement.

The format of the IF statement is:

```
IF expression
  THEN
    instruction
  [ELSE
    instruction]

IF expression THEN instruction[; ELSE instruction]
```

Example:

```
IF salary > 1000000 THEN
  DO
    SAY 'Do you want to go to lunch? Its your treat'
    PULL ans
  END
```

A NOP: No operation instruction is a dummy instruction. The NOP instruction can be used when no operations are necessary.

Example:

```
IF weather = 'fine' THEN
  DO
    SAY "What a great day "
    IF POOL = 'open' THEN
      SAY "Let's go for a dip"
    ELSE NOP
  END
ELSE
  SAY "Need your raincoat?"
```

5 SELECT Statement

The SELECT statement is used for evaluating a number of choices. It is easier to maintain and program than a series of nested statements.

The first WHEN statement that has a true value is executed. After it executes the first true WHEN, all other WHEN statements are ignored.

The OTHERWISE is executed if none of the WHENs are true.

If more than one instruction is to be executed, then the DO/END statement should be used. The exception is the OTHERWISE which does not require a DO or END. OTHERWISE would be the required clause.

The format of the SELECT statement is:

```
SELECT
    WHEN expression THEN instruction(s)
    WHEN expression THEN instruction(s)
    .
    .
    OTHERWISE
        instruction(s)
END
```

Example:

```
SELECT
    WHEN month = 4 | month = 6 | month = 9 | month = 11
        THEN days = 30
    WHEN month = 2
        THEN days = 28
    OTHERWISE
        days = 31
END
```

5.1 SELECT Statement Walkthrough

Code a REXX program using the SELECT statement to perform the following steps:

- Obtain from the operator:
 - # of years programming experience
 - Operators age, sex, and name

- Display one of the following messages:

<u>Condition</u>	<u>Message</u>
EXPERIENCE <1 YEAR	YOU ARE A BABY PROGRAMMER
EXPERIENCE BETWEEN 1 AND 3 YEARS AND FEMALE	HELLO, MS _____
EXPERIENCE BETWEEN 1 AND 3 YEARS AND MALE	HELLO, MR _____
MORE THAN 3 YEARS EXPERIENCE AND FEMALE	HER HIGHNESS _____
MORE THAN 3 YEARS EXPERIENCE AND MALE	YOUR GREATNESS _____

- Fill in the underscores with the operator's name.

6 DO Loops

There are two categories of REXX loops:

Repetitive loops repeat a certain number of times.	Conditional loops repeat based on a condition.
--	--

Both repetitive and conditional loops are supported by REXX.

The format of the repetitive loops statement is:

```
DO number-times
  instructions
END
```

With a control variable, the loop can be repeated and the variable incremented. The control variable automatically increases by the incr amount. The default is 1.

```
DO numb = start TO end BY incr
  instructions
END
```

Examples:

```
looper = 5
DO looper
  SAY "HI YA ALL"
END
```

```
DO I = 1 to 5
  SAY I
END
```

```
DO I = 1 TO 10 BY 2
  SAY I
END
```

Examples:

```
DO COUNT = 1
```

```
    SAY COUNT
```

```
    IF COUNT = 10 THEN
```

```
        LEAVE
```

```
END
```

```
DO PTR = 12 BY -2 TO 7
```

```
    SAY 'COUNT BACKWARDS ' PTR
```

```
END
```

```
DO COUNT =1 to 50
```

```
    DO CITY=1 to 10
```

```
        say 'GIVE Next City Population.'
```

```
        say 'of state', COUNT
```

```
        Pull CITYPOP
```

```
        COUNTRY.COUNT.CITY=CITYPOP
```

```
    END
```

```
END
```

7 DO FOREVER

DO FOREVER is an infinite loop that can be used to keep executing logic until a null key is entered.

The format of the DO FOREVER statement is:

```
DO FOREVER
    instructions.
END
```

Example:

```
DO FOREVER
    SAY "ENTER DATA SET TO BE DELETED"
    PULL DSN
    IF DSN = '' THEN
        LEAVE
    "DELETE" DSN
END
```

8 LEAVE and ITERATE Instructions

The LEAVE instruction causes an immediate exit from the loop. Control is then transferred to the statement after the END.

The format of the LEAVE instruction is:

LEAVE [name]

The ITERATE instruction passes control to the DO to continue with the next control variable or to continue the loop.

The format of the ITERATE instruction is:

ITERATE [name]

If a control variable is detected it is incremented or decremented, based on the values in the DO.

9 DO WHILE and UNTIL Instructions

The DO WHILE and UNTIL statements are conditional loops repeat based on a condition

The WHILE instruction loop executes the loop while the condition is true; the logic validation is performed at the top of the loop.

The format of the DO WHILE expression is:

```
DO WHILE expression
    instructions
END
```

If the condition is false the first time into the loop, the loop is bypassed. The LEAVE statement is used for exiting from the loop.

The UNTIL loops when a condition is not true; execution of the loop will continue until the condition evaluates to true. Since the check is at the end of the loop. The loop always executes at least once

The format of the DO UNTIL expression is:

```
DO UNTIL expression
    instructions
END
```

DO loops can be nested.

Examples:

```
I = 0
FLAG = 'T'
DO WHILE FLAG = 'T'
    I = I + 1
    IF I = 10 THEN FLAG = 'F'
    SAY I
END
```

```
DO UNTIL pass = password
    SAY "ENTER password"
    PULL pass
END
```

10 EXIT Statement

The EXIT is used to unconditionally terminate a REXX procedure and return to where the procedure was invoked.

The format of the EXIT code is:

```
EXIT code
```

The termination usually returns control back to TSO/ISPF; however it can be invoked and returned to the SELECT services of Dialog Manager or to another procedure.

The EXIT also can return a single value to the invoking procedure.

If the invoking procedure is a REXX routine, the value passed back is in the special variable called RESULT.

Example:

```
DO FOREVER
    SAY 'Do you want to continue (Y/N) '
    PULL response
    IF response = 'N' THEN EXIT
.
.
END
```

11 SIGNAL Statement

The format 1 of the SIGNAL statement can be used to pass control to a label in the program, such as a GOTO type statement.

SIGNAL does not provide the return capabilities of a subroutine or function. Over use of the SIGNAL statement will make program maintenance more difficult.

The format 2 of the SIGNAL statement is very useful. It causes REXX to branch to the label ERROR.

If an error is detected, which occurs when a host statement has a non zero condition code in the program, SIGL will contain the line number of the statement that caused the transfer.

The formats of the SIGNAL statement are:

```
SIGNAL label
SIGNAL ON ERROR NAME label
```

Example:

```
SIGNAL looper
```

```
looper:
```

Example:

```
/* Example of REXX SIGNAL ON ERROR */
SIGNAL ON ERROR
"ALLOC DA(SAMPLE.DATA) DD(INPFILE) SHR"
.
.
.
ERROR:
SAY "An error was detected on line " SIGL
SAY "The Return Code is " RC
```

12 Condition Traps

These condition traps can be used in the SIGNAL command.

ERROR	Error in host instruction. Traps all positive and negative return codes.
FAILURE	Error in host instruction. Traps all negative return codes.
HALT	Attempt to cause an external interrupt. For example, the HI command.
NOVALUE	Unutilized variable.
SYNTAX	Syntax error on REXX interpretation of a command.