ReXX
Restructured eXtended eXecutor

Contacto
Lic. José N Castro <jncastro@gylgroup.com>
Lic. Aldo Fernández Villalba <avillalba@gylgroup.com>

1. Conceptos básicos de ReXX

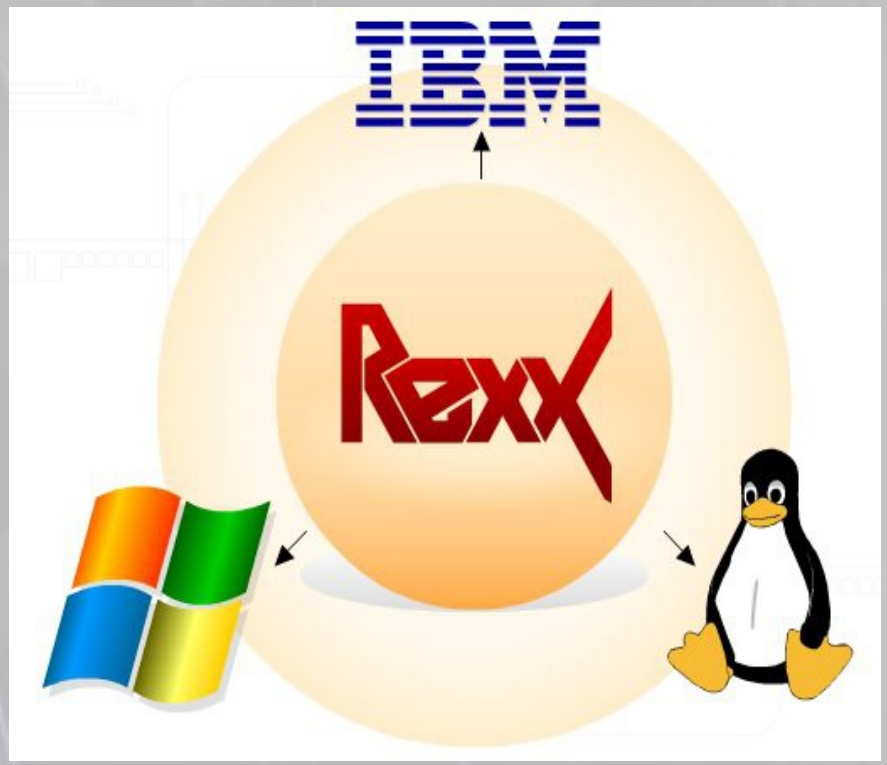
ReXX fue desarrollado por **Michael Cowlishaw**, un «*System Programmer*» del laboratorio de IBM, para el sistema IBM VM/SP.

Según el propio Mike, la idea nació el 20 de marzo de 1979.

Inicialmente destinado a reemplazar el lenguaje «*EXEC2*», una herramienta del mainframe para automatizar y complementar procesos, ReXX demostró que tenía potencial para trascender esas tareas, y se fue convirtiendo en un auténtico lenguaje de programación.

ReXX a través del tiempo

REXX para VM (*1983*)
REXX para TSO/E (*1988*)
REXX Compiler (*1989*)
REXX en AS/400 (*1990*)
REXX en OS/2 (*1990*)
REXX en AIX/6000 (*1991*)
REXX en CICS/ESA (*1992*)
REXX/imc para Unix (*1992*)
Regina REXX para Windows y Linux (*1992*)



Con el lanzamiento del estándar ***ANSI*** para ReXX en 1996, el lenguaje ReXX ahora está claramente definido.

- Lenguaje interpretado
- El intérprete es un componente integrado de IBM TSO
- Hay compiladores disponibles, pero no se usan mucho.
- Puede ejecutar comandos para diferentes entornos del mainframe, incluidos editores de texto.
- Permite aplicar prácticas de programación estructurada
- La codificación es de formato libre
- No se requiere declaración previa de las variables.
- Insensible a mayúsculas y minúsculas
- Incluye funciones integradas
- Incluye capacidades de rastreo de errores
- Gran capacidad para manipular palabras y cadenas de caracteres

2. Ejecutar un ReXX en MVS/TSO

Forma explícita: Tipeando «EXEC» or «EX» seguido por el nombre de la biblioteca que contiene el programa ReXX y el nombre del programa:

Explicito desde la linea de comandos de TSO (o =6):

```
exec 'hlq.rexx(rx20115)'
```

Explicito desde la linea de comandos de ISPF

```
tso exec 'hlq.rexx(rx20115)'
```

Si se necesita enviar **parámetros** al programa, hay que ejecutar el ReXX de forma explícita incluyendo la opción de parametros:

Explicito desde la linea de comandos de TSO:

```
exec 'hlq.rexx(rx20115)' 'Parametros'
```

Forma implícita: No se ingresa «EXEC», sino sólo el nombre del programa. Si TSO no lo reconoce como un comando, del sistema, buscará en los particionados en SYSEXEC.

```
Command==> rx20115
```

Forma implícita extendida: Con un signo de porcentaje seguido del nombre del programa. Asi, TSO solo busca en SYSEXEC. Esta forma es más rápida porque el TSO no intenta buscar comandos del sistema.

```
Command==> %rx20115
```

Para que sean reconocidas en la **ejecución implícita**, los programas ReXX deben almacenarse en un archivo PDS concatenado al DD SYSEXEC en el perfil del TSO. Pueden asignarse dinámicamente mediante el comando ALLOCATE o como parte del procedimiento LOGON.

```
  Display  Filter  View  Print  Options  Help
--------------------------------------------------------------------------------
SDSF OUTPUT DISPLAY B034165  TSU07409  DSID     3 LINE  CHARS 'SYSEXEC' FOUND
COMMAND INPUT ===> _                                          SCROLL ===> CSR
     118 XXSYSEXEC   DD DSN=ISP.SISPEXEC,DISP=SHR
     119 XX          DD DSN=QMF.V101.SDSQEXCE,DISP=SHR
     120 XX          DD DSN=SYS1.SBPXEXEC,DISP=SHR
     121 XX          DD DSN=MQM.V701.SCSQEXEC,DISP=SHR
     122 XX          DD DSN=SYS1.SEDGEXE1,DISP=SHR
     123 XX          DD DSN=FMN.V121.SFMNEXEC,DISP=SHR
         XX*************************************************************100
     124 XXSYSTCPD   DD DSN=TCPIP.ZLAB.PARM(TCPOSA),DISP=SHR
     125 XXSYSFTPD   DD DSN=TCPIP.ZLAB.PARM(FTPDATA),DISP=SHR
```

Normalmente no tendrás autoridad para modificar tu perfil de TSO, pero puedes agregar temporalmente un PDS a SYSEXEC.

```
RECONCAT SYSEXEC DSNAME('xxxx.xxxx.REXXLIB') LAST
```

Una vez que hayas terminado de trabajar con ReXX, opcionalmente podrías cancelar la asignación y liberar la biblioteca PDS:

```
RECONCAT SYSEXEC EXCLUDE('xxxx.xxxx.REXXLIB') FORCE
```

Veamos si esto funciona...

1) Desde la terminal de TSO, ingresar a la opción «3.2», y crear un archivo tipo *PDS LIBRARY*:

KC03###.ZOS2023.REXXLIB

2) Usando el editor de ISPF, generar un elemento con el nombre «*HOLAREXX*» dentro del PDS, con el código del programa ReXX para la prueba:

```
/* REXX */
Say "Hola ReXX";
```

3) Desde la linea de comandos de TSO (=6) ejecutar el programa con la forma explicita.

Desde un proceso «*Batch*»

```
//IRXJCL EXEC PGM=IRXJCL,PARM='<rexx-name> <parametros>'
```

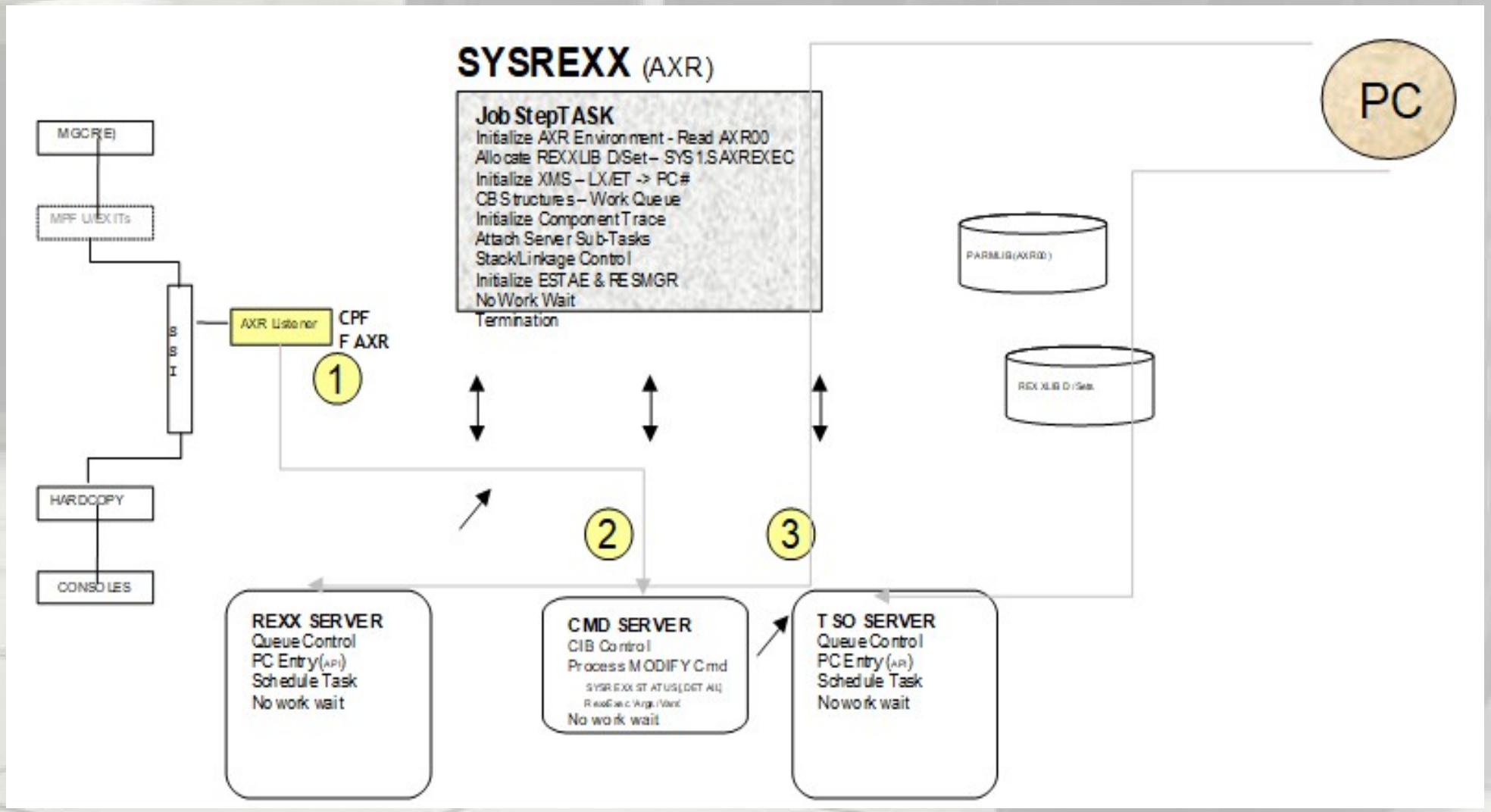
Dentro del editor de ISPF

Las macros escritas en ReXX pueden ser ejecutadas desde la linea de comandos del editor ISPF, o asignándolas a una tecla PF, o como comandos de linea.

Con el servidor SYSREXX

El servidor SYSREXX ocupa el address space «AXR». Los requerimientos a SYSREXX ingresan desde el «AXR SSI Listener» o la interfase AXREXX, o desde una PC directamente al servidor.

System ReXX es un componente para z/OS 9.1 y superiores.



Hola ReXX en modo «*Batch*»

```
//IRXJCL   EXEC PGM=IRXJCL,PARM='HOLAREXX'
//SYSEXEC  DD DISP=SHR,DSN=KC03###.ZOS2023.REXXLIB
//SYSTSIN  DD DUMMY      Entrada para Pull
//SYSTSPRT DD SYSOUT=*   Salida para Say
//
```

3. Sintaxis de ReXX

Formato de las instrucciones

- Los **comentarios** comienzan con /* y terminan con */
- Un **punto y coma** indica el final de la instrucción, pero es opcional para las instrucciones de una sola línea
- Las instrucciones deben separarse con **punto y coma** cuando se colocan en la misma línea
- La **coma** al final de una línea se usa para continuar una instrucción en la linea siguiente
- Los **espacios y tabulados** que se incluyan para mejorar la legibilidad no afectan la ejecución del programa
- Las cadenas de literales se pueden encerrar entre **comillas simples** o **dobles**

Formato de las instrucciones

- **Variables**: Los nombres de variables son literales de hasta 250 caracteres alfanuméricos más los caracteres `.!?_`
- Los literales no pueden comenzar con un punto «`.`»
- El carácter punto «`.`» tiene un uso especial para variables compuestas, no debería utilizarse en otros casos
- Un literal seguido de ***dos puntos*** «*`nombre_etiqueta:`*» indica un nombre de párrafo, una función, o una sub rutina

Operadores matemáticos

+ (suma) − (resta) / (división) * (multiplicación)

% División entera

// Resto de la división entera

** Potencia (puede indicarse un exponente positivo o negativo)

() Agrupar operaciones

Operadores de textos

|| Concatenatenar dos literales

Operadores lógicos

= (igual a...) > (mayor que...) < (menor que...)

\= (no igual a...) ¬= (no igual a...) <> (no igual a...)

<= (menor o igual que...) >= (mayor o igual que...)

& (...y...) | (...o...) \ (no...) ¬ (no...)

Instrucción de asignación de valor

```
variable = valor
```

```
a  = (10.5 + x) * y;  /* Operacion matematica *
b  = -34.5645e23;     /* Notacion exponencial *
t  = “Por favor, ingrese 'Enter'”;  /* Texto *
t1 = 'A1';            /* 'A1' en caracteres *
t2 = 'C1F1'x;         /* 'A1' en forma hexadecimal */
t3 = '11000001'b;     /* 'A' en forma binaria     */
```

Variables especiales

Se generan automáticamente durante el procesamiento de un programa REXX. No tienen un valor inicial, pero el programa puede alterarlas. Hay tres variables especiales:

RESULT: Contiene un valor retornado por un programa ReXX o subrutina

RC: Contiene el código de retorno de una llamada a un comando externo

SIGL: El número de línea del programa que causó una transferencia de control en una trampa de condición de error.

Mostrar datos en la pantalla

La instrucción «***SAY***» muestra en la pantalla de la terminal TSO un texto. Si es ejecutado en modo «*Batch*» la salida se envía a la DD SYSTSPRT

```
SAY list_of_elements
```

Los elementos de la lista a mostrar se separan por un espacio. Si se requiere que no haya espacio entre ellos se pueden concatenar:

```
X = 10;
y = 'th.';
SAY x 'is the' x || y 'number.';
```

Mostrará:

```
10 is the 10th. number.
```

A trabajar en el teclado ...

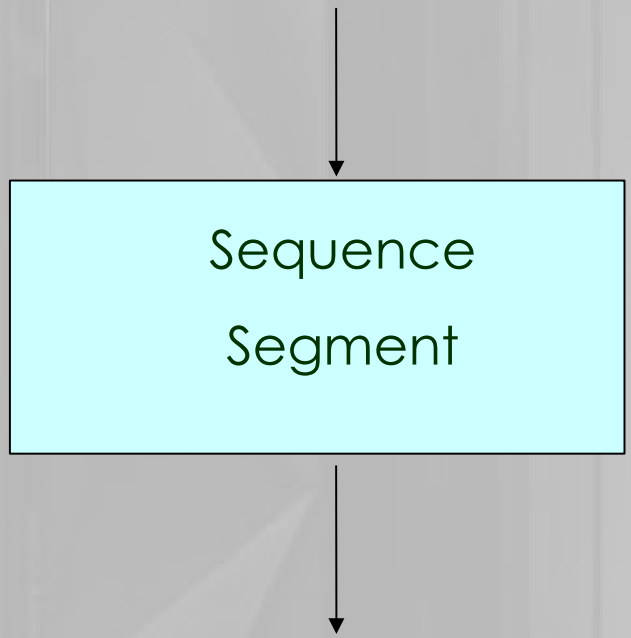
```
/* REXX */
/*  Practicamos asignar valores y mostrar resultados  */

num = ...;          /* Un numero cualquiera  */
mat = ...;          /* Una operacion matematica  */
txt = ...;          /* Un 'texto'  */
hex = ...;          /* Un valor en Hexadecimal 'hh'x */
bin = ...;          /* Un valor en Binario  'nnnn'b  */

Say 'El valor de num es:' num;
...
Return;
```

4. Estructuras de control

Segmento secuencia

Segmento de secuencia de instrucciones.

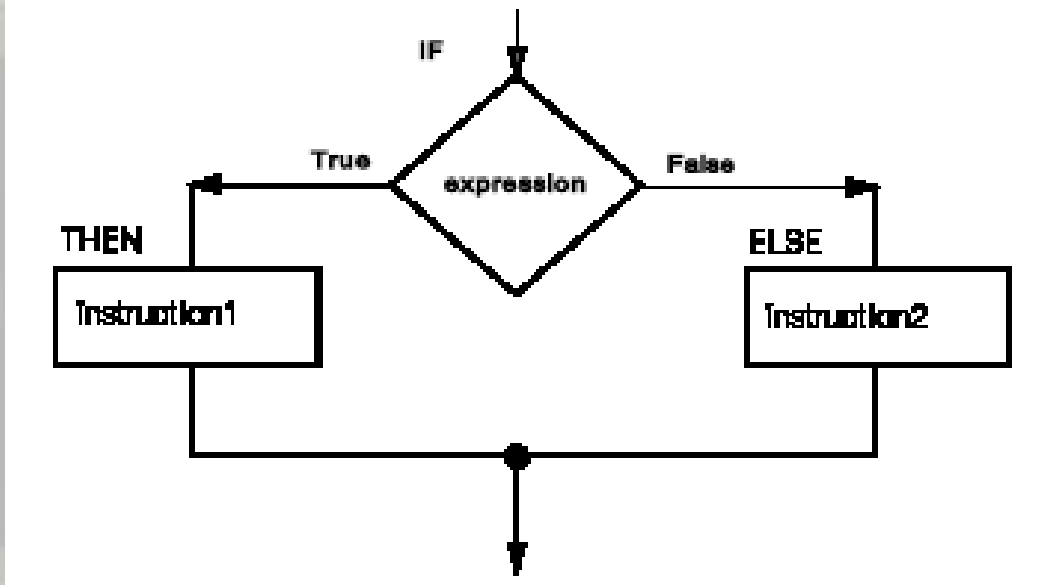
```
Do;
Instrucciones;
End;
```

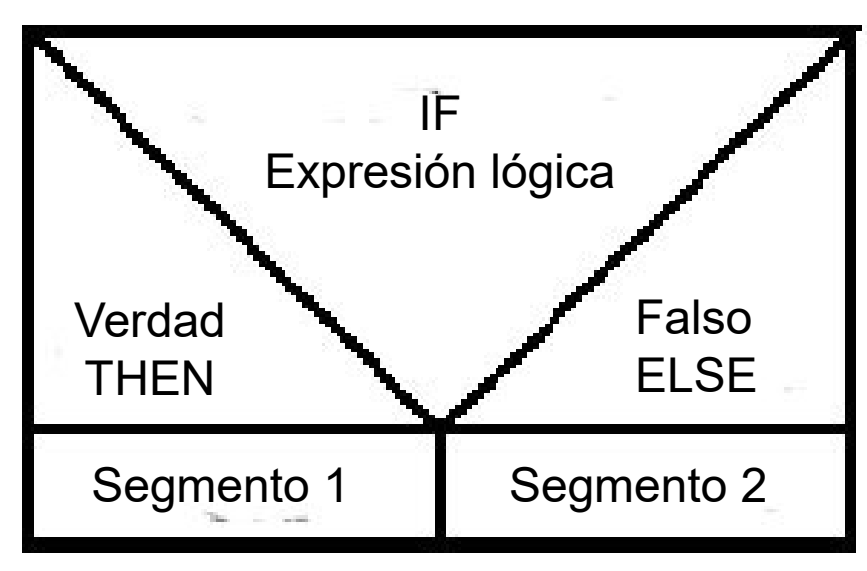


Segmento de instrucciones nulo (vacío).

```
Nop;
```

Segmento selección





Diagramación tradicional y método «*Nassi-Shneiderman*»

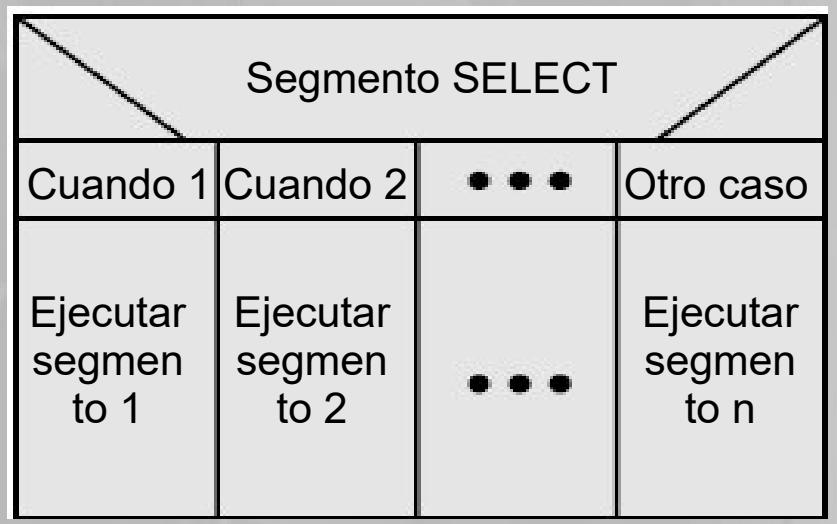
```
IF logical-expresion THEN
   Segmento-1
ELSE
   Segmento-2
```

Segmento de selección (IF).

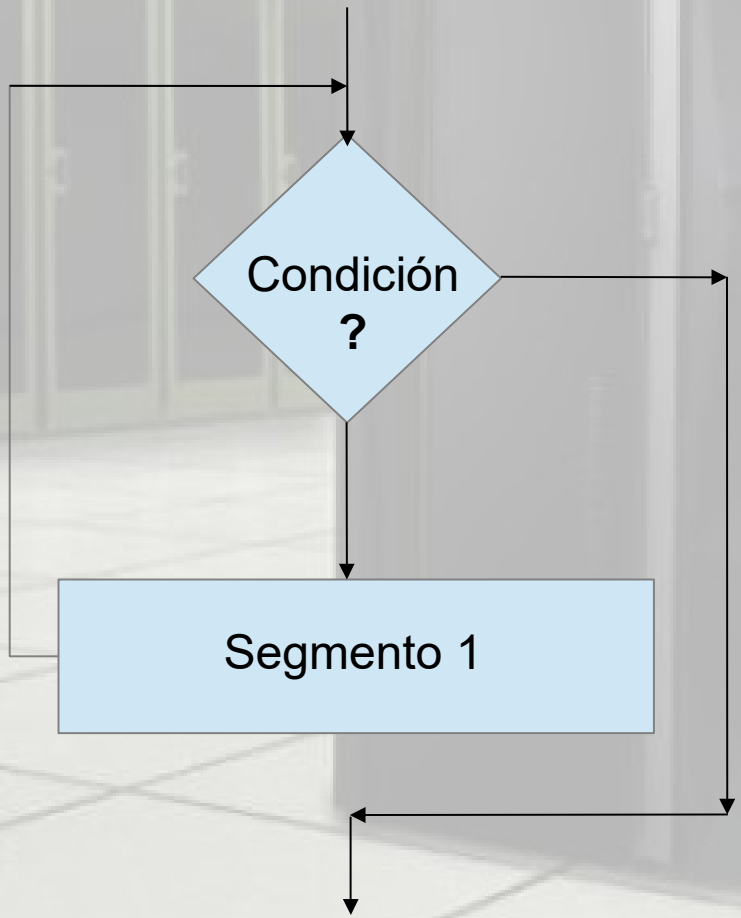
```
/* REXX Segmento de seleccion con IF  */
XA = 2;
XB = 5;

If XA > XB Then
  Say 'A =' XA 'es mayor que B =' XB;
Else
  Do;
  Say 'A =' XA 'no es mayor que B =' XB 'Corrigiendo';
  AUX  = XB;
  XB   = XA;
  XA   = AUX;
  Say 'Ahora si A =' XA 'es mayor o igual a B =' XB;
  End;
```

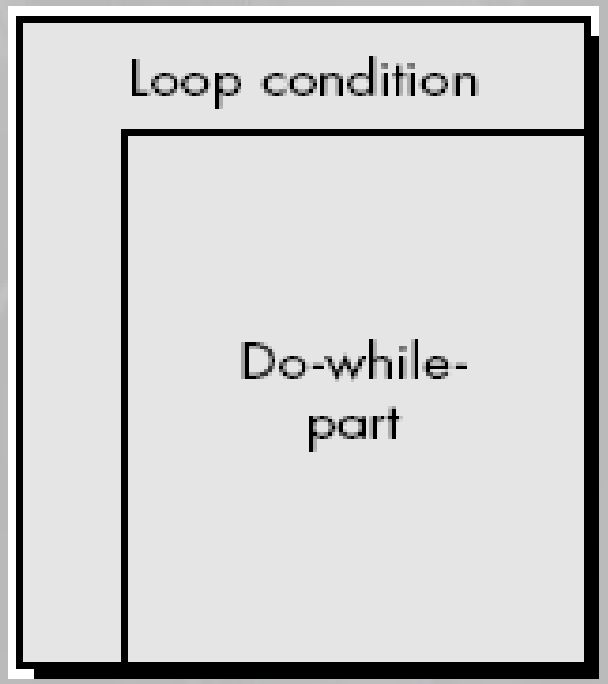
Segmentos de selección (SELECT).

```
/* REXX Segmento de seleccion con SELECT  */
A = 2;
B = 5;

Select
  When A > B Then
    Say 'A es mayor que B';

  When A < B Then
    Say 'B es mayor que A';

  Otherwise
    Say 'A y B son iguales';
End;
```

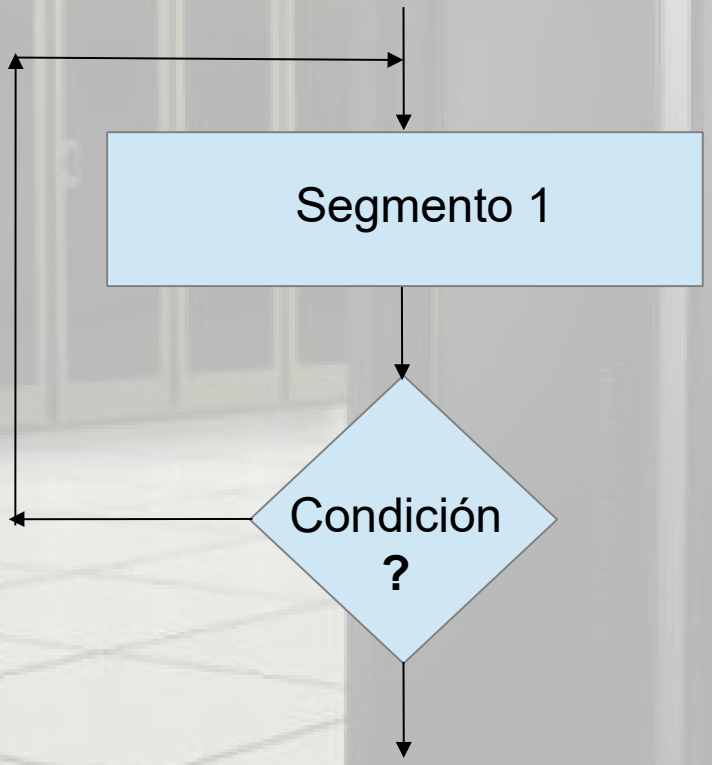


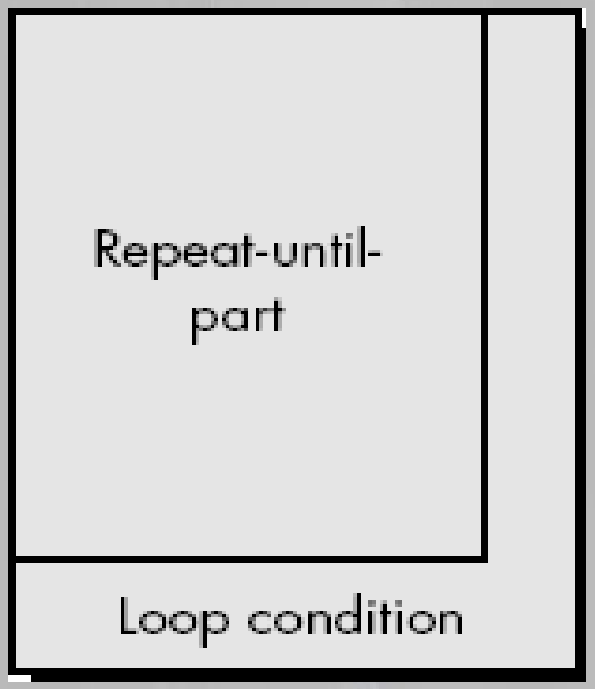
Segmento iteracion While





Segmento iteracion con Until





«DO» como Segmento de iteración

```
DO WHILE / UNTIL expresion-logica
  Segmento-1
END

DO FOREVER / nn
  Segmento-2
END

DO var=expr TO expr BY expr
  Segmento-3
END
```

Alteradores de la secuencia

```
LEAVE / ITERATE
```

Practiquemos para ver la diferencia...

Iteración (DO WHILE)

```
/* REXX */
I = 5;

Do While I < 5
  Say "I =" I;
  I = I + 1;
End;
```

Iteración (DO UNTIL)

```
/* REXX */
I = 5;

Do Until I >= 5
  Say "I =" I;
  I = I + 1;
End;
```


Universidad Nacional de La Matanza
GROUP
5. Parsing

La instrucción **PARSE** analiza un texto y lo separa en diferentes variables.

```
PARSE [UPPER] {VAR | ARG | PULL} template;
```

UPPER: (opcional) Convertir a mayúsculas

{ARG | PULL | VAR}: Fuente de los datos a analizar

Template: Patrones que indican el orden de palabras, los números de columnas, o los marcadores de posición, que se usarán para separar las partes del texto en variables.

Analizando y separando el contenido de una variable de texto en «*palabras*» separadas por espacios:

```
text_var = "one two three four ***";
PARSE VAR text_var w1 w2 el_resto;
SAY w1; SAY w2; SAY el_resto;
```

la salida será:

```
one
two
three four ***
```

Se puede usar un **punto** como variable ficticia para saltar palabras del texto que no interesan.

```
PARSE VAR text_var w1 w2 .;
```

Análisis y separación utilizando patrones posicionales:

```
text_var = "one two three four five";
PARSE VAR text_var 5 x .;
SAY x;
```

la salida será:

```
two
```

Análisis y separación del texto utilizando «*Place-holders*» (Marcadores de posición):

```
PARSE VAR text_var "three" x .;
SAY x;
```

la salida será:

```
four
```

Cuando se utilizan patrones posicionales o «*Place-holders*», el texto es primeramente dividido en sub textos, y recién después se realiza la separación en variables

```
text_var = "one two three four five";
PARSE VAR text_var "three" x;
```

Primero separa el texto en dos:

`"one two "` y `" four five"`

Hay que tener en cuenta esto para evitar posibles resultados erroneos.

Un parámetro posicional puede tener dos formatos:

- **Posicionamiento absoluto**. Indica una posición específica dentro del texto. No lleva signo o puede llevar un signo =

```
variable1 11 variable2 =21 variable3
```

  El número 11 se refiere a la posición 11 en el texto; 21 a la posición 21

- **Posicionamiento relativo**. Indica un desplazamiento desde el puntero previo. Se precede con + ó -. «+0» también es válido

```
variable1 +10 variable2 + 10 variable3
```

  El desplazamiento +10 se refiere a la posición 11 en el texto, y el siguiente + 10 a la posición 21.

Un puntero posicional es como un punto de detención no inclusivo, por ejemplo 1 + 10: Posición 11, no incluida. Si es negativo retrocede en el texto.

Dos ejemplos para practicar...

```
/* -- Puntero posicional relativo -- */
reg = 'Justo     Liborio   Quebracho';
Parse var reg apellido +10 nombre +10 seudonimo;
Say seudonimo 'es el seudonimo de' nombre apellido;


/* -- Retroceder a una posición anterior -- */
txt = 'astronomers';
          /*    st       a        r         s  */
Parse var txt 2 v1 +2 -3 v2 +1 +2 v3 +1 +6 v4;
Say txt 'study' v1 || v2 || v3 || v4;
```

Análisis y separación del texto utilizando «*Placeholders*» (Marcadores de posición):

```
fecha = '06/10/2023 10:25:00 en Argentina';
Parse var fecha dia '/' mes '/' anio ' ' hora ':' ,
      min ':' seg .;
Say 'Dia' dia 'del mes' mes 'del Año' anio     ,
     '; Hora' hora || '.' || min || '.' || seg;
```

Una variable como «*place-holder*»

Para que ReXX reconozca una variable como «*place-holder*» se la encierra entre paréntesis:

```
texto  = 'Jorge L. Borges';
patern = '. ';
Parse var texto nombre inicial (patern) apellido;
Say apellido || ',' nombre inicial;
```

Una variable como «*place-holder*»

La variable puede también tomarse de un carácter incluído en el mismo texto y extraído previamente en la misma instrucción:

```
fecha  = '21/10/2023';
Parse var fecha dia 3 delim +1 mes (delim) anio;
Say 'Dia' dia 'del mes' mes 'del Año' anio  ,
    '- Delimitador:' delim;
```

Una variable como puntero posicional

Si un signo igual precede al paréntesis izquierdo, entonces el valor de la variable se trata como un puntero posicional absoluto o, si es un signo más o menos, como un puntero posicional relativo:

```
datos  = '12 29 .....Pedro B. PalaciosAlmafuerte';
Parse var datos pos1 pos2 6 =(pos1) nombre ,
        =(pos2) seudonimo;
Say 'Nombre:' nombre '- Seudonimo:' seudonimo;
```

NOTA: *¿Por qué se necesita el patrón posicional 6 en la plantilla? Recordá que el análisis de palabras ocurre después de que ReXX divide el texto en sub textos usando los punteros. Por lo tanto, el puntero posicional =(pos1) no se puede interpretar correctamente como =12 hasta que ReXX haya cortado el texto en la columna 6, y haya asignado los valores 12 y 29 a pos1 y pos2, respectivamente.*

Practiquemos ahora con argumentos...

```
/* REXX */
/* Separa items de la lista de parametros   */
PARSE ARG a b c .;
Say "a =" a "- b =" b "- c =" c;
```

Ejecucion en TSO, modo explicito pasando parametros como argumento:

```
exec 'KC03###.ZOS2023.REXX(nombre)' 'one two three'
```

Resultado esperado:

```
a = one - b = two - c = three
```

6. Variables compuestas

Arreglos de datos en ReXX (Opción 1)

Las variables compuestas son una técnica para agrupar variables. Esta función se usa con frecuencia para administrar vectores y matrices de datos en ReXX.

Una variable compuesta comienza con un «*stem*» (raíz), que es un nombre de variable válido, seguido de un punto. Los caracteres después del primer punto se llaman «*tail*» (cola).

```
my_variable.my_index
+--stem---+ +-tail-+
```

Arreglos de datos en ReXX (Opción 1)

Asignar un valor a un elemento de un vector:

```
my_table.0 = "elemento cero";
```

En el momento de la ejecución, cada variable se reemplaza por su valor actual para generar el nombre de la variable derivada. Esta es una expresión equivalente:

```
i = 0;
my_table.i = "elemento cero";
```

Asignar un valor a un elemento de una matriz de 3 dimensiones:

```
ind = 4;
my_matrix.3.ind.5 = "Esto es coordenadas (3,4,5)";
```

Arreglos de datos en ReXX (Opción 1)

Si se asigna un valor a la «*raíz*», todas las posibles variables compuestas que comienzan con esa raíz también contendrán por defecto ese mismo valor.

```
my_variable. = "Valor inicial";
+--stem---+
```

Vamos a la práctica...

```
/* REXX - Variables compuestas */

NAME. = "Nadie"; /* Asignar el valor por defecto */

A = 2;  /* Asignar valores a las variables de cola */
B = 3;

NAME.A = "Dario"; /* Esto se convierte en NAME.2 */
NAME.B = "Juan";  /* Esto se convierte en NAME.3 */

SAY NAME.1 NAME.2 NAME.3; /* a NAME.1 no se asigno valor */
Return;
```

7. Subrutinas y funciones

Una función es una porción de código, escrito por separado, que devuelve un valor.

Al referenciar la función se utilizan paréntesis, para enviarle uno o más parámetros o argumentos.

```
/* REXX - Ejemplo de una funcion */
Say "El cuadrado de 3 es:" square(3);
Say "El cuadrado de 5 es:" square(5);
Say "El cuadrado de 9 es:" square(9);
Exit;  /* Terminar el programa es necesario aqui */

/* funcion que calcula el cuadrado de un numero */
square:
  Arg IN;
  Return IN*IN;
```

Una subrutina es similar a una función, excepto que no devuelve un valor. Se la referencia con la instrucción «**call**».

```
/* REXX - Ejemplo de una subrutina */
Call box "Una oracion en una caja";
Exit; /* Terminar el programa es necesario aqui */

box:  /* Imprimir el argumento dentro de una caja */
  Arg texto;  /* ARG convierte a mayusculas */
  Say "+------------------------------+";
  Say "|"centre(texto,32)"|";
  Say "+------------------------------+";
  Return;
```

La instrucción **PROCEDURE** asegura que se genere un ámbito de variables propio para una función o subrutina. Esto habilita funciones recursivas. Debe ser la primera instrucción dentro de la subrutina.

```
/* REXX - Llamada a una funcion recursiva */
X = 4;
Say x"! =" Factorial(X);
Exit; /* Terminar el programa */

Factorial: Procedure;  /* Calcula factorial */
  Arg N;               /* por llamadas recursivas */
  If N = 0 then
     Return 1;
  Return Factorial(N - 1) * N;
```

Para proteger todas las La variables en un ámbito interno de una función o subrutina, pero dejar expuestas algunas, se utiliza la instrucción PROCEDURE con el parámetro EXPOSE.

```
/* REXX - Protege las variables pero no todas */
NUMBER1 = 10;
Call Subroutine;
Say NUMBER1 NUMBER2; /* Muestra: 7  NUMBER2 */
Exit;

Subroutine:
  Procedure expose NUMBER1;
  NUMBER1 = 7;  /* Esta variable es del ambito global */
  NUMBER2 = 5;  /* Esta es interna de la subrutina */
  Return;
```

Funciones «*built-in*» propias del lenguaje. Puede haber variaciones entre diferentes versiones de ReXX y en distintas plataformas.

ABS(number) Devuelve el valor absoluto de un número, como un valor sin signo.

TRUNC(number [,n]**)** Devuelve la parte entera de un número con n posiciones decimales. "n" por defecto es "0", sin decimales.

CENTER(string,length [,pad]**)** Devuelve un texto centrado en un largo dado y rellena con los caracteres «*pad*», si se especifica, o con espacios.

DATE([option]**)** Devuelve la fecha actual en el formato especificado por «*option*».

Arreglos de datos en ReXX (Opción 2)

Otra posible opción, aunque limitada, para manejar vectores de datos en ReXX son las funciones «*Built-in*» que permiten direccionar palabras, o sea, conjuntos de caracteres separados por uno o más espacios:

`WORD(texto,n)` Retorna la n-ava palabra del texto.

`WORDINDEX(texto,n)` Retorna la posición del primer caracter de la n-ava palabra del texto.

`WORDLENGTH(texto,n)` Retorna el largo de la n-ava palabra del texto.

`WORDPOS(texto,'x' [,n])` Retorna el número de la primera palabra en el texto igual a **`'x'`**. Opcionalmente comenzando a buscar en la n-ava palabra.

`WORDS(texto)` Retorna la cantidad de palabras que contiene el texto.

Subrutinas y funciones codificadas externamente

Puede codificarse una subrutina, a ser llamada con «**CALL**», o una funcion, referenciada con «***funcion(param)***», en un código en la biblioteca PDS separado del programa ReXX que la referenciará.

Esto habilitaría para construir nuevas funciones que funcionarían como «*Built-in*»

La biblioteca que contenga la función o subrutina debe estar en la lista de bibliotecas concatenadas en la DD SYSEXEC.

En el caso de la ejecución explicita desde TSO, si la subrutina o funcion está en la misma biblioteca del programa la encuentra igualmente.

Subrutinas y funciones codificadas externamente

- El nombre del member en el PDS debe ser el mismo nombre con que se invoca a la subrutina o función
- El literal con el nombre de la subrutina o función no es necesario, ReXX la asocia por el nombre del member en el PDS
- No acepta la instrucción **PROCEDURE**
- A diferencia de cuando se codifica in-line, la subrutina o función siempre tiene un ámbito local de variables
- Los parámetros son pasados correctamente y recibidos con la instrucción **ARG**
- La variable predefinida **RESULT** contendrá el valor de retorno.

```
/* REXX */
/* Llama a una subrutina codificada externamente */
NUMBER1 = 0;
P1 = 'A';     /* Variable de entorno global */
P2 = 'B';     /* Variable de entorno global */
Call ExtSubR P1 P2;
Say RESULT;                    /* Muestra: A <> B          */
Say NUMBER1 NUMBER2;           /* Muestra: 0 NUMBER2 */
Say 'P1 =' P1 '; P2 =' P2; /* Muestra: P1 = A ; P2 = B   */
Exit;
```

```
/* REXX */
/* Subrutina KC03###.ZOS2023.REXXLIB(EXTSUBR) */
Parse Arg X1 X2 .;
R1 = X1 '<>' X2;     /* R1 Variable de entorno local */
P1 = 'Null';         /* Variable de entorno local */
P2 = 'Null';         /* Variable de entorno local */
NUMBER1 = 7;         /* Variable de entorno local */
NUMBER2 = 5;         /* Variable de entorno local */
Return R1;
```

```
/* REXX */
/* Ejemplo de una Funcion codificada externamente */
RET = Box("Un texto en una caja");
Say 'Return =' RET;
Exit;
/* Muestra:
     +--------------------------------+
     |      UN TEXTO EN UNA CAJA      |
     +--------------------------------+
     Return = Ok.
*/
```

```
/* REXX */
/* Funcion KC03###.ZOS2023.REXXLIB(BOX) */
Arg TEXTO;       /* ARG convierte a mayusculas */
 Say "+--------------------------------+";
 Say "|"centre(texto,32)"|";
 Say "+--------------------------------+";
Return 'Ok.';
```

```
/* REXX */
/* Llamada a una funcion externa recursiva */

X = 4;
Say x"! =" Factorl(X);
Exit;
```

```
/* REXX */
/* Funcion KC03###.ZOS2023.REXXLIB(FACTORL) */
Arg N;
If N = 0 then
   Return 1;
Return Factorl(N - 1) * N;
```

Documentación pública de IBM

z/OS TSO/E REXX Reference

Describe la estructura y la sintaxis general del lenguaje REXX, incluidas expresiones y operadores, instrucciones y funciones.

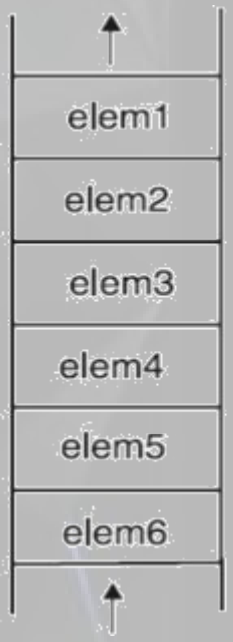
https://www-40.ibm.com/servers/resourcelink/svc00100.nsf/pages/zOSV2R3sa320972/$file/ikja300_v

8. Pilas y colas de datos

«*Pilas*» y «*Colas*» de datos

Stack

LIFO



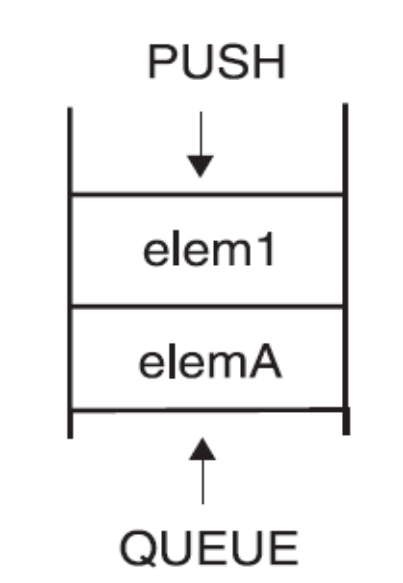
Queue

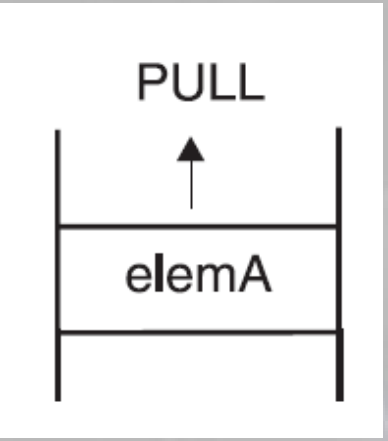
FIFO

Manejo de pilas y colas de datos en ReXX

El manejo de colas de datos de Rexx combina técnicas *LIFO* (**`Push`**) y *FIFO* (**`Queue`**) para agregar elementos.

La extracción de elementos se produce únicamente desde la parte superior de la pila (**`Pull`**):

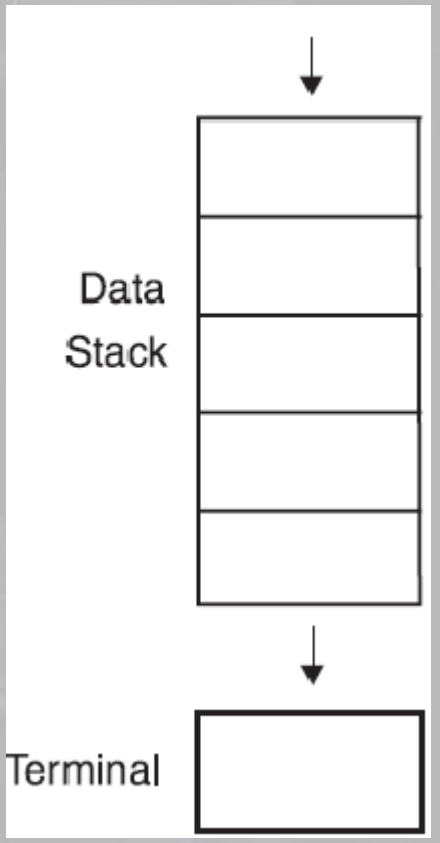




Manejo de pilas y colas de datos

Cuando usa la instrucción **`Pull`** desde ReXX, pero la pila de datos está vacía, la información se recupera:

- Desde el teclado de la terminal bajo TSO
- Desde la `//SYSTSIN DD` si se está ejecutando en modo «*batch*»



Manejo de pilas y colas de datos en ReXX

Para poder controlar las colas de datos, existe una función «*built-in*» que retorna la cantidad de filas, o elementos, que contiene una cola:

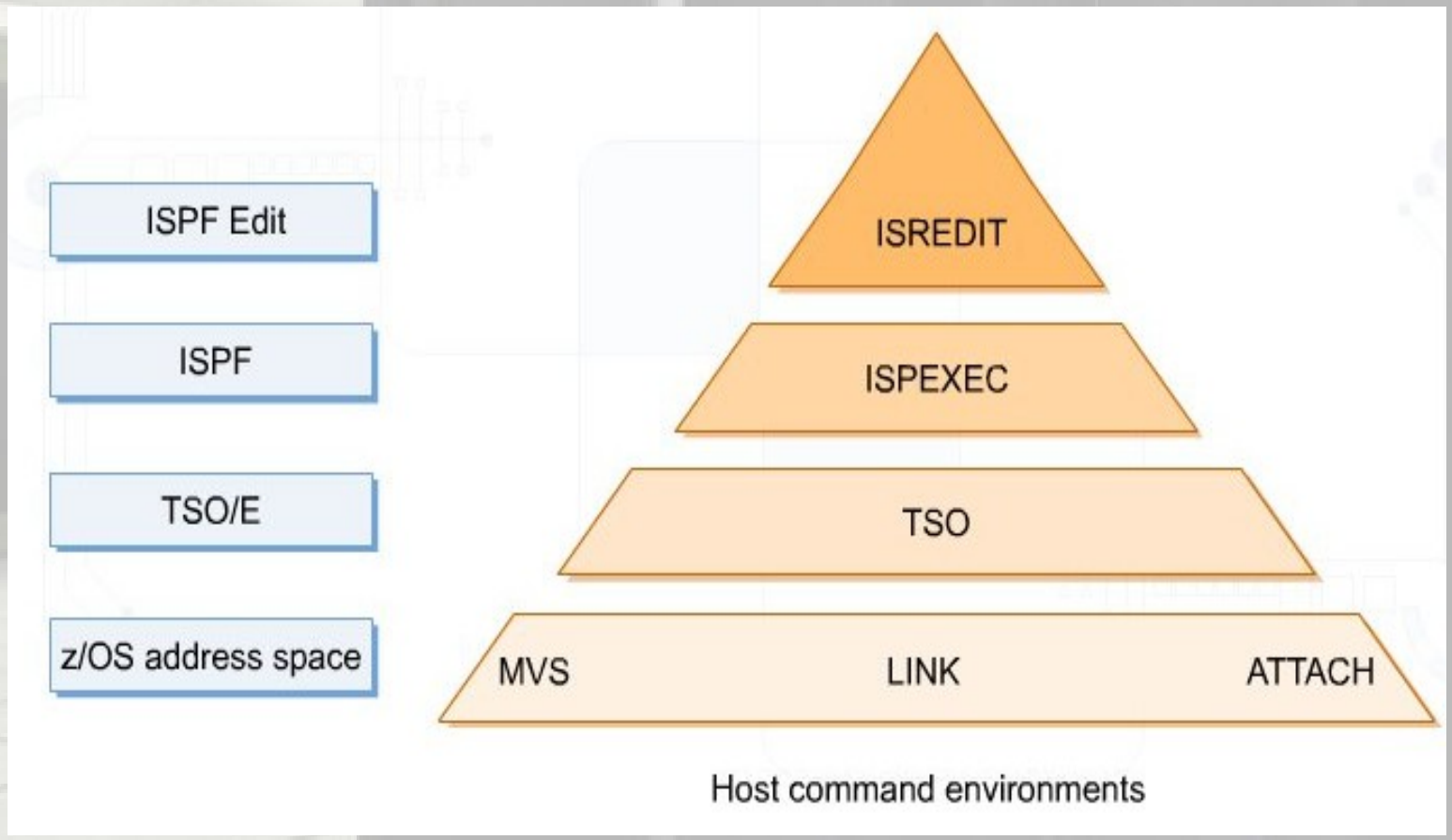
```
IF QUEUED() = 0 THEN
  DO;
  ...   /*  La cola de datos esta vacia  */
  END;
ELSE
  DO;
  ...   /*  Procesar los datos disponibles  */
  END;
```

Conversemos con la terminal...

Utilizar la cola de datos para leer desde el teclado de la terminal:

```
/* REXX */
/* Parse desde la terminal */
NN_NAME = 'Nomen Nescio';    /* NN: Nombre desconocido */

Say 'Por favor, ingresa tu nombre:'
Pull NEW_NAME;

If NEW_NAME = '' then
  NEW_NAME =  NN_NAME;

Say 'Hola,' NEW_NAME;
Exit;
```

9. Interfases con el entorno


ISPF Edit
ISPF
TSO/E
z/OS address space
ISREDIT
ISPEXEC
TSO
MVS
LINK
ATTACH
Host command environments

La instrucción «**ADRESS**» establece de forma temporaria o permanente a que interfase se enviarán los comandos:

```
ADDRESS environment ["command"];
```

Sin un comando, se establece un enrutamiento permanente al entorno indicado, que durará hasta que se indique otra cosa.

Algunos de los entornos disponibles en z/OS:

```
ADDRESS 'TSO';      /* Target: TSO/E host environment (Default) */
ADDRESS 'MVS';      /* Target: MVS host environment (Default sin TSO) */
ADDRESS 'ISPEXEC'; /* Target: ISPF from TSO/E  */
ADDRESS 'CONSOLE'; /* Target: MVS Console commands environment  */
ADDRESS 'ISREDIT'; /* Target: ISPF edit macros  */
ADDRESS 'LU62';     /* Target: APPC/MVS based on the SNA LU 6.2  */
```

10. Lectura de archivos en z/OS

Cuando se ejecuta ReXX desde TSO, los archivos que se leerán deben asignarse dinámicamente primero para poder hacer referencia a ellos:

```
"ALLOC DA('hlq.input.file')  F(DD001) SHR";
```

Para liberar los recursos asignados de TSO cuando se terminó de utilizarlos:

```
"FREE F(DD001)";
"FREE ALL";
```

Cuando se ejecuta la instrucción de lectura de TSO «**EXECIO**», las líneas requeridas son leídas del archivo y se **encolan** en la pila de datos.

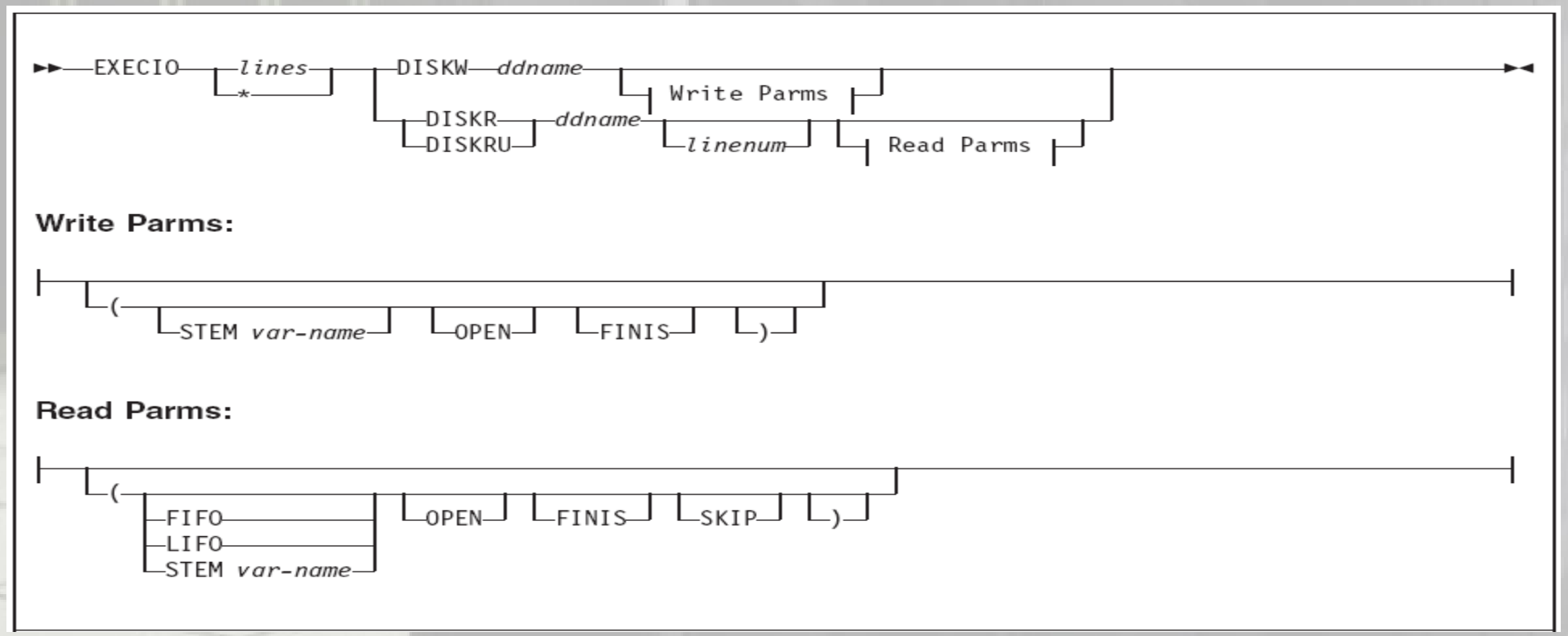
Luego se usa la instrucción ReXX «**PULL**» para recuperar los datos desde la pila:

```
"EXECIO 1 DISKR DD001";
Pull LINEA;
```

O bien, los datos se pueden analizar y separar en variables en este punto:

```
Parse Pull ITEM1 ITEM2 .;
```

La instrucción de lectura/escritura «**EXECIO**» puede leer tanto desde archivos PDS como archivos secuenciales desde TSO o «*Batch*».



Para ver en detalle este comando, consulta el capítulo 10, del manual «*z/OS TSO/E REXX Reference*».

Esta lógica que se sugiere aquí es solo una de las múltiples formas posibles de organizar un bucle para leer un archivo secuencial:

```
Do forever;
   'Execio 1 Diskr DDD01';  /* Leer una linea */
   If RC <> 0  then         /* Si no leyo el registro */
      Do;
      /* --- Ultimos pasos en el fin del archivo --- */
      Leave;    /* Abandonar el bucle infinito */
      End;

   Pull RECORD;  /* Recuperar el registro de la pila */
   /* --- Procesar el registro leido --- */
End;
```

Trabajo práctico...

ReXX-1 Leer archivo en z/OS

11. Grabar archivos en z/OS

Cuando se genera un archivo de salida desde TSO, deben especificarse algunos parámetros adicionales. Minimamente:

```
"Alloc da('hlq.input.file')  F(DD001) Lrecl(100) New";
```

Y hay otros parametros disponibles:

```
SPACE(2) TRACKS DSORG(PS) RECFM(F) BLKSIZE(n) ...
```

La variable especial ***RC*** devolverá el resultado de la operación para realizar alguna acción correctiva o terminar el programa:

```
If RC <> 0  then
  "Alloc da('hlq.input.file') F(DD001) Lrecl(100) Old";
```

Consulta el formato del comando ALLOC en el
z/OS TSO/E Command Reference

Para grabar registros en un archivo, las líneas deben ponerse en cola en la pila, usando los métodos de «**stack**» o «**queue**».

El comando «**EXECIO**» recuperará los registros «*First Out*», o sea, de la parte superior de la pila, a menos que especifique algo diferente

```
Push  "Segunda linea a grabar";
Push  "Primera linea a grabar";
Queue "Ultima linea a grabar";
"EXECIO 3 DISKW DD002 (FINIS";
```

El parámetro FINIS cierra el dataset luego de grabar y permite que el archivo se libere al finalizar

```
"Free F(DD002)";
```

Trabajo práctico...

ReXX-2 Grabar un archivo en z/OS

12. Paneles ISPF con ReXX

```
------------------   Cuentas de clientes   ------------------

COMANDO ===>


    Seleccione una opcion de las siguientes :

      1 Mostrar detalles de clientes

      2 Eliminar un cliente

      3 Agregar un cliente

      4 Modificar una cuenta de cliente


              Seleccion : 1
```

```
)BODY
+
+-----------------   Cuentas de clientes   -----------------
+COMANDO ===>_ZCMD
+    Seleccione una opcion de las siguientes :
+    %1+ Mostrar detalles de clientes
+    %2+ Eliminar un cliente
+    %3+ Agregar un cliente
+    %4+ Modificar una cuenta de cliente
+
+              Seleccion : _CHOICE+
+
+   %&MSG1
+   %&MSG2
)INIT
    .CURSOR = CHOICE
)PROC
    VER (&CHOICE,LIST,1,2,3,4)
    VER (&CHOICE,NB)
)END
```

Los paneles de ISPF se almacenan como elementos un el archivo Particionado. Al definir la biblioteca para los paneles ISPF deben respetarse algunos parámetros para obtener un resultado correcto:

- La biblioteca debe definirse como un dataset Particionado PDS o PDS/E (tipo LIBRARY)
- El formato es Fijo Bloqueado, con una longitud de registro de 80 bytes, o más
- Configurar la biblioteca sin numeración automática, comando NUM OFF en el editor de ISPF
- Si, por alguna razón se generó númeración automática en las últimas posiciones de las lineas, hay que eliminarlas para evitar errores en el panel

Las secciones y parámetros que definen la estructura de un panel son:

)BODY Identifica la sección que será visible en la terminal.
- \+ Inicio de un campo protegido
- % Inicio de un campo protegido resaltado
- & Comienzo de una variable protegida.
- _ Comienzo del campo de entrada con variable

)INIT Indica los detalles de configuración.
- .CURSOR Variable especial, ubicación del cursor

)PROC Sección que se ejecuta después de presionar <*Enter*>
- VER Opciones de verificación:
  - LIST Comparar con una lista
  - NB Verificar que un valor no sea un espacio en blanco

)END Fin del panel

Para trabajar con un panel ISPF, se utiliza la instrucción «**ADDRESS**» para que el entorno «*ISP Exec*» muestre el panel.

La biblioteca PDS donde se definió el panel, debe estar definida por la **DD ISPPLIB**.

```
Address ISPEXEC "LIBDEF ISPPLIB DATASET ID('hlq.paneles') STACK";
Address ISPEXEC "DISPLAY PANEL(panel001)";
```

Estas instrucciones definen la biblioteca de paneles y muestran el panel en la terminal. Luego, ISPF esperará a que se ingrese «*Intro*». El control volverá entonces al programa.

Si se presiona la tecla «F3», se enviará un código de retorno 8 al programa. La tecla «F3» es el estándar para terminar en Mainframe.

```
Address ISPEXEC "LIBDEF ISPPLIB DATASET" ,
                "ID('KC03###.ZOS2023.PANELES')"
MSG1 = ""; MSG2 = "";

Do Forever
  Address ISPEXEC "DISPLAY PANEL(SAMP01P)";
  If RC = 8  then
    Do;
    MSG1 = "Gracias por probar!";
    MSG2 = "«Enter» para terminar";
    Address ISPEXEC "DISPLAY PANEL(SAMP01P)";
    Leave; /* -- Sale del ciclo -- */
    End;
  Else
    MSG1 = "Se tipeo la opcion:" CHOICE;
End;
Exit;
```

Trabajo práctico...

TP3 Panel ISPF para el sistema K9

13. ISPF Table Services

El ISPF Table Services

- **Table Service** es otro de los servicios que tiene el ISPF disponibles
- Las tablas del «*ISPF Table Services*», ***NO son*** bases de datos
- Se pueden usar para almacenar datos permanentes, y resultan útiles debido a su facilidad de uso
- Hay varios componentes del servicio de tablas disponibles en la «*Application library ISPTLIB*». Algunos afectan a las filas de la tabla para agregar filas, eliminarlas, etc, y otros afectan a toda una tabla para crearla, abrirla, cerrarla, ordenarla, etc.
- Sólo se puede procesar **una fila** de una tabla a la vez
- Las tablas de ISPF se mantienen y acceden en memoria
- Pueden ser guardadas como un miembro en un dataset particionado definido previamente como FB con LRECL=80, y vinculado al TSO con la DD ISPTABL

Servicios que afectan una tabla entera

TBCREATE Crea una nueva tabla y la abre para su procesamiento

TBOPEN Abre una tabla existente y la carga en memoria para su procesamiento

TBEND Cierra una tabla sin guardarla

TBCLOSE Cierra una tabla y opcionalmente guarda una copia permanente

TBERASE Elimina una tabla permanente del disco

TBSAVE Guarda una copia en disco de una tabla sin cerrarla

TBSORT Ordena las filas de una tabla

TBSTATS Obtiene el estado actual de una tabla

Servicios que afectan las filas de una tabla

TBADD Agrega una nueva fila a la tabla

TBDELETE Elimina una fila de la tabla

TBTOP Establece el puntero (CRP) en TOP, o sea antes de la primera fila

TBBOTTOM Establece el puntero (CRP) en la última fila y recupera la fila

TBSKIP Avanza una fila dentro de la tabla y recupera los datos de la fila

TBGET Recupera una fila específica de la tabla según una clave

TBMOD Actualiza una fila existente en la tabla o agrega una nueva fila a la tabla.

TBPUT Actualiza una fila en la tabla si existe y si las claves coinciden

TBSCAN Busca en una tabla una fila que coincida con una lista de variables de "argumento" y recupera la fila

Estructura y características de una tabla

```
TBCREATE table-name [KEYS (key-name-list)]
        [NAMES (name-list)]
        [WRITE|NOWRITE] [REPLACE]
```

table-name puede tener de 1 a 8 caracteres alfanuméricos de longitud y debe comenzar con uno alfabetico

key-name-list Lista de columnas que serán claves de la tabla. Antes de agregar una fila, el Table Services determina si algún campo clave (**KEYS**) de la nueva fila duplica el campo clave de una fila existente. Si es así, no se inserta el registro.

name-list Nombre de las columnas de la tabla que no son claves

WRITE Indica que la tabla es permanente, y se guardará mediante TBSAVE o TBCLOSE. Es la opción predeterminada.

NOWRITE Especifica que la tabla es sólo para uso temporal y será eliminada por TBEND o TBCLOSE

REPLACE Si la tabla existe la reemplaza y termina con RC = 4

z/OS ISPF ISPF Reference Summary

Sintaxis y parámetros de los servicios de ISPF.

https://www-40.ibm.com/servers/resourcelink/svc00100.nsf/pages/zOSV2R4sc193624/$file/f54rs00_v

z/OS ISPF Developer's Guide and Reference

Guía de los servicios de ISPF, paneles, table services (*Pag 61*), y varios otros.

https://www-40.ibm.com/servers/resourcelink/svc00100.nsf/pages/zOSV2R3sc193619/$file/f54dg00_v

Antes de acceder a los servicios de ISPF Table Services es necesario indicar el nombre la biblioteca, el dataset particionado en el cual se guardarán las tablas:

```
/* --- Si la tabla va a ser creada o modificada --- */
"LIBDEF ISPTABL DATASET ID('KC03###.ZOS2023.TABLAS')";

/* --- Si la tabla va a solo leida --- */
"LIBDEF ISPTLIB DATASET ID('KC03###.ZOS2023.TABLAS')";
```

Cada servicio de ISPF que se utilice deja un código de retorno en la variable ReXX ***RC*** que puede ser utilizado para controlar el estado.

Códigos de retorno de **TBCREATE**:

- `0` Terminación normal
- `4` Terminación normal. La tabla existe y se indicó REPLACE
- `8` La tabla ya existe y no se indicó REPLACE
- `12` La tabla está siendo usada
- `16` Modo «write» y la tabla existe en la biblioteca de entrada
- `20` Error grave

Códigos de retorno de **TBOPEN**:

- `0` Terminación normal
- `8` La tabla no existe
- `12` La tabla está siendo usada
- `16` No se indicó una biblioteca de entrada
- `20` Error grave

El servicio **TBSTATS** permite consultar el estado de una tabla. Devuelve hasta tres códigos de estado en las variables que se indique:

```
"TBSTATS tabla STATUS1(var1) STATUS2(var2) STATUS3(var3)"
```

Status 1:

- 1 La tabla existe en la biblioteca de entrada
- 2 La tabla no existe en la biblioteca de entrada
- 3 La biblioteca de entrada no fue vinculada

Respuestas del servicio **TBSTATS**

Status 2:

- 1 La tabla no fue abierta
- 2 La tabla está abierta en modo «nowrite»
- 3 La tabla está abierta en modo «write»
- 4 La tabla está abierta en modo «shared nowrite»
- 5 La tabla está abierta en modo «shared write»

Status 3:

- 1 La tabla está disponible en modo «write»
- 2 La tabla no está disponible en modo «write»

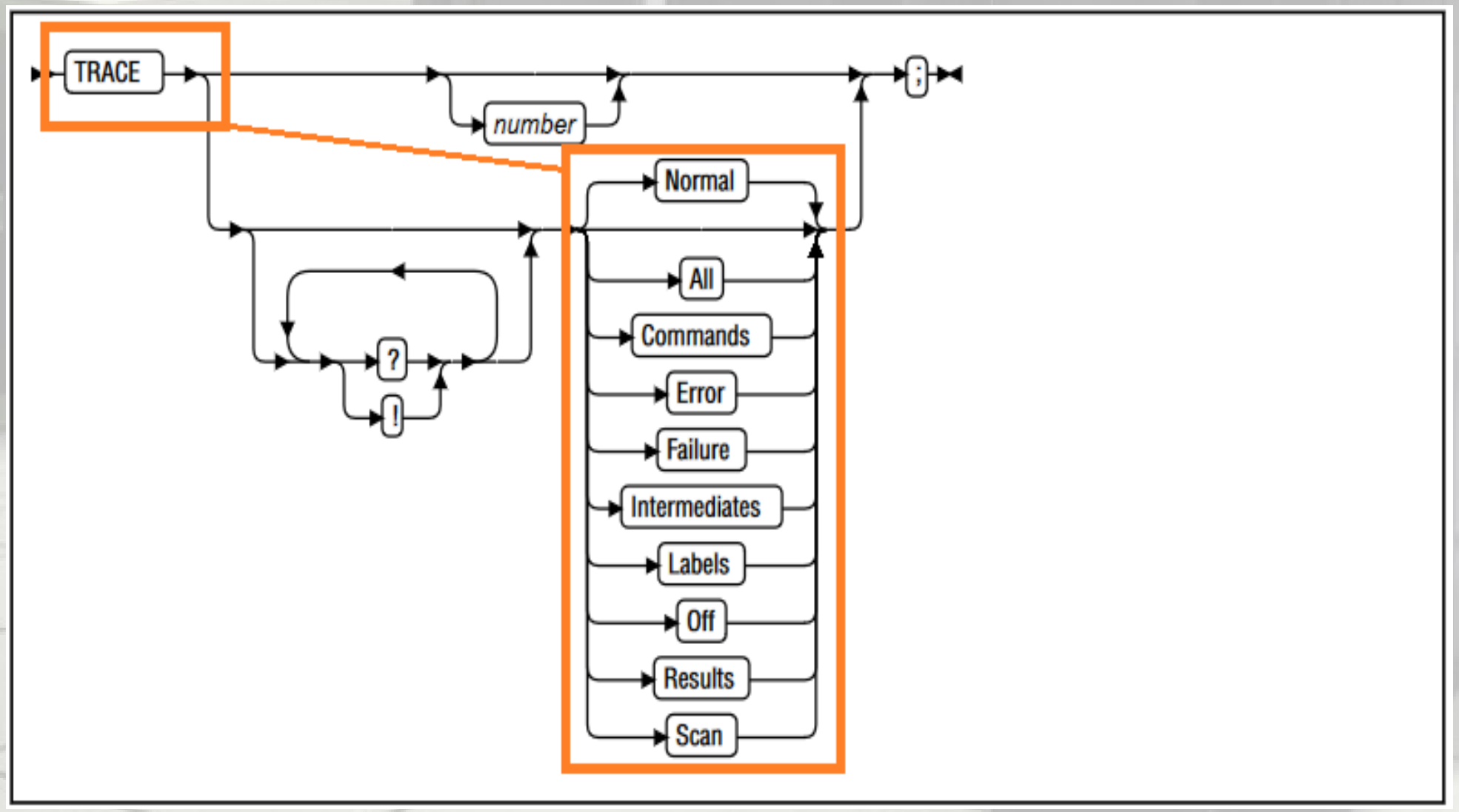
```
/* REXX */
/* ------------------------------------------------------------
   Practica con el sistema de ISPF Table Services
   Asegurate de haber generado el dataset '...TABLAS'
   Formato PO, FB, Lrecl=80
 ------------------------------------------------------------ */
Address ISPEXEC;
"LIBDEF ISPTABL DATASET ID ('KC03###.ZOS2023.TABLAS')";
"TBCREATE REXXTAB1 KEYS(RNO)"  ,
          "NAMES(NOMBRE, CALIFIC, ESTADO) WRITE REPLACE";
Say 'TBCreate RC=' RC;

Say 'Cantidad de filas a insertar?';
Pull I;
```

```
Do J = 1 to I by 1;  /* --- Repetir el loop I veces --- */
   Say 'Ingrese un Nombre y una calificacion (0 a 5)'
   Pull INOMBRE ICALIFIC .;

   If ICALIFIC < '0' | ICALIFIC > '5'  then
      IESTADO = 'Calificacion invalida';
   Else
      If ICALIFIC > '3' then
         IESTADO = 'Aprobado';
      Else
         IESTADO = 'Reprobado';
   /* --- Prepara las columnas e inserta la fila --- */
   RNO      = J;
   NOMBRE   = Strip(INOMBRE);
   CALIFIC = Strip(ICALIFIC);
   ESTADO   = Strip(IESTADO);
   "TBADD REXXTAB1";

   If RC <> 0 then
      Say 'Error al insertar la fila con clave' RNO 'RC=' RC;
End;
```

```
/* --- Muestra todas las filas insertadas --- */
Do J = 1 to I by 1;
   "TBTOP REXXTAB1";
   RNO  = J;
   "TBGET REXXTAB1";
   Say 'El estado de' NOMBRE 'es:' ESTADO;
End;

"TBCLOSE REXXTAB1";
Exit;
```

14. Depuración de errores

El comando TRACE...



El comando TRACE...

```
Trace <parametro>;  /*  Esta es la forma mas usual  */
```

Normal: Rastrea cualquier comando que resulte en un código de retorno negativo (con error) después de la ejecución, junto con el código de retorno del comando. *Esta es la configuración predeterminada*.

Results: Rastrea todas las instrucciones antes de la ejecución. Muestra los resultados de comandos. También muestra los valores asignados durante las instrucciones *PULL*, *ARG* y *PARSE*. *Esta es la configuración que se sugiere para una depuración general.*

All: Rastrea, es decir muestra, todas las instrucciones antes de su ejecución.

Off: No rastrea nada.

El comando TRACE...

Las lineas en el diagnóstico del «*trace*» son precedidas por un código de tres caracteres que permite identificar a que clase de rastreo corresponde.

- *Muestra el código de la instrucción, es decir, la línea de codigo tal cual está en el programa.*

+++ *Identifica un mensaje de «trace». Puede ser el código de retorno distinto de cero en un comando, un error de sintaxis, u otras cosas.*

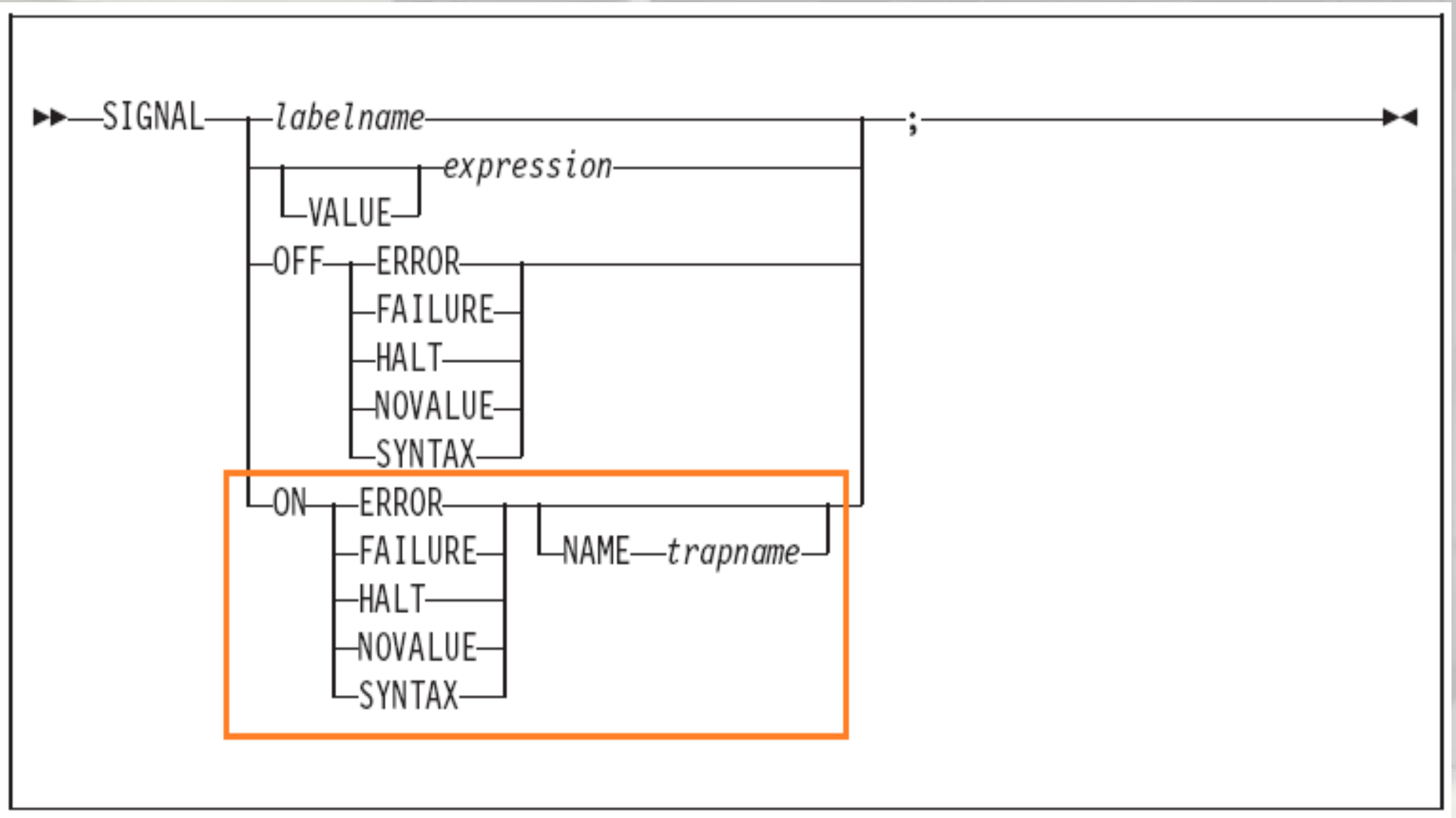
>>> *Identifica el resultado de una expresión (si se indicó «TRACE R»), o el valor asignado a una variable, o el valor devuelto por una subrutina.*

El comando TRACE...

```
TRACE A
"test"
old_name = "Fred"
SAY "Please enter name : "
PARSE PULL tst_name
PARSE VAR tst_name new_name .
IF new_name = old_name THEN DO
   SAY "Hello Fred"
END
ELSE DO
   NOP
END
```

```
 12 *-* "test"
        >>>   "test"
 MISSING DSNAME
     13 *-* old_name = "Fred"
     14 *-* SAY "Please enter name : "
 Please enter name :
     15 *-* PARSE PULL tst_name
Fred
     16 *-* PARSE VAR tst_name new_name .
     17 *-* IF new_name = old_name
        *-*  THEN
        *-*  DO
     18 *-*   SAY "Hello Fred"
 Hello Fred
     19 *-*  END
 ***
```

El comando SIGNAL...



Documentación pública de IBM

z/OS TSO/E REXX User's Guide
Explica como diseñar, escribir y ejecutar programas en ReXX.
https://www-40.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r3sa320982/$file/ikjc300_v2

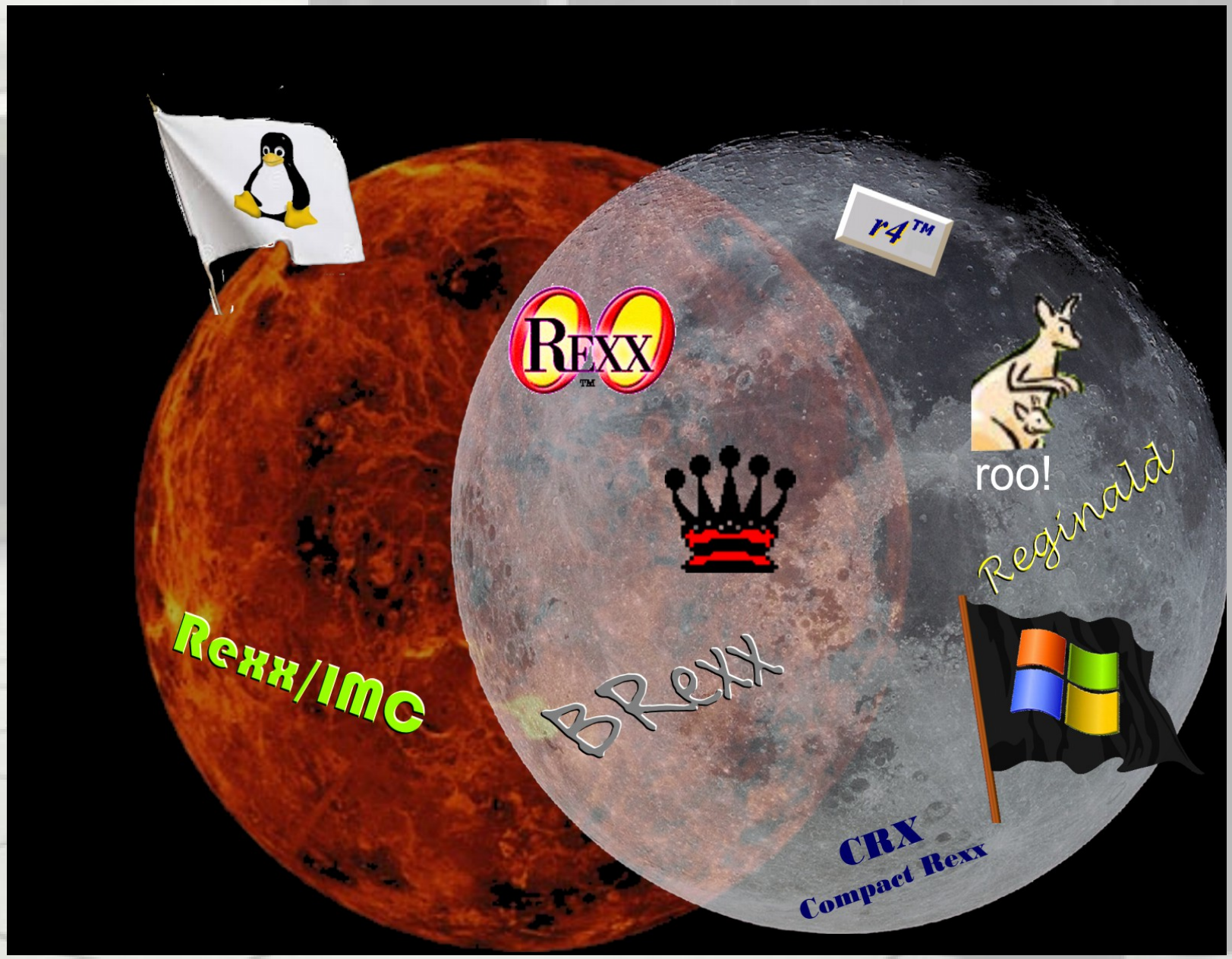
z/OS TSO/E Command Reference
Describe en detalle la sintaxis y la función de los comandos y subcomandos del TSO/E.
https://www-40.ibm.com/servers/resourcelink/svc00100.nsf/pages/zOSV2R3sa320975/$file/ikjc500_v

Documentación pública de IBM

IBM Compiler and Library for REXX on zSeries V1R4
User's Guide and Reference
https://www.ibm.com/docs/es/zos/2.1.0?topic=clrszugr-abstract-compiler-library-rexx-zseries-users-gu

z/OS Information Roadmap
Una guía de la documentación disponible sobre z/OS y los productos asociados.
https://www-40.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r3sa232299/$file/e0zc100_v2

15. ReXX en el mundo PC


r4™
REXX
roo!
Reginald
Rexx/IMC
BRexx
CRX
Compact Rexx

ooRexx:

Open Object Rexx incorpora funciones típicas de un lenguaje orientado a objetos, como subclasificación, polimorfismo y encapsulación de datos. Estas extensiones no reemplazan las funciones clásicas de Rexx ni impiden el desarrollo o la ejecución de programas clásicos de Rexx.

Regina

Intérprete de ReXX de código abierto, libre y gratuito bajo Licencia GNU. Regina es el que más se parece al ReXX de z/OS.

Una característica importante de Regina, que lo hace muy recomendable es la de apuntar a dos objetivos principales declarados:

- *Cumplir al 100 % con la norma ANSI.*
- *Estar disponible en tantas plataformas como sea posible.*

El sitio oficial de ReXX

The Rexx Language Association

Una organización independiente, sin fines de lucro, dedicada a promover el uso y la comprensión del lenguaje de programación Rexx.

Uno de sus administradores es *Michael Cowlishaw*. Este sitio contiene una copia del sitio original de Mike.

Desde este sitio se puede acceder a páginas confiables para descarga de los interpretes gratuitos, con licencia GNU como *OORexx*, *Regina* y otros, para diferentes plataformas. También contiene enlaces y ejemplos de utilitarios diversos que son aportes de colaboradores.

https://www.rexxla.org/

¿Dudas, Preguntas?

Contactos

José N Castro <jncastro@gylgroup.com>

Aldo Fernández Villalba <avillalba@gylgroup.com>