Course Exercises Guide

z/OS REXX Programming

Course code EZ52G ERC 1.0

IBM Training

July 2020 edition

Notices

This information was developed for products and services offered in the US.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Contents

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

AIX®	BookManager®	CICS®
CICSPlex®	DataPower®	DB™
Db2 Connect™	Db2®	Distributed Relational Database Architecture™
Domino®	DS8000®	ECKD™
eServer™	Express®	FICON®
FlashCopy®	GDPS®	Global Technology Services®
HiperSockets™	Hiperspace™	HyperSwap®
IBM Systems Director Active Energy Manager™	IBM z Systems®	IBM z®
IBM zHyperWrite™	IBM z13®	IBM z13s™
IBM z14	IMS®	Language Environment®
Lotus®	MQSeries®	MVS™
Notes®	OS/390®	Parallel Sysplex®
Power®	PowerVM®	POWER9®
PR/SM™	Processor Resource/Systems Manager™	pureXML®
RACF®	Redbooks®	Resource Link®
Resource Measurement Facility™	RMF™	S/390®
Smarter Cities®	Smarter Planet®	System Storage®
System z®	System z10®	System z9®
System/390®	Tivoli®	VTAM®
WebSphere®	z Systems®	z/Architecture®
z/OS®	z/VM®	z/VSE®
zEnterprise®	z10™	z13®
z13s™	z9®	400®
z14	z14 ZR1	z15

Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

VMware is a registered trademark or trademark of VMware, Inc. or its subsidiaries in the United States and/or other jurisdictions.

Evolution® is a trademark or registered trademark of Kenexa, an IBM Company.

Social® is a trademark or registered trademark of TWC Product and Technology, LLC, an IBM Company.

Other product and service names might be trademarks of IBM or other companies.

Exercises description

Exercise instructions - This section contains what it is you are to accomplish. There are no definitive details on how to perform the tasks. You are given the opportunity to work through the exercise given what you learned in the unit presentation, utilizing the unit Student Notebook, your past experience, pertinent reference material, and maybe a little intuition.

You will gain most from this course by effort put into the laboratory exercises. The aim should not be to get the exercises right in the minimum time. Rather, as your aim is to learn, you should experiment and try new ideas. Surprisingly, the more mistakes you make, the more you will learn. If in doubt, try it out. Also, use the manuals, the online documentation, the public domain program REXXTRY.

Do not worry if you cannot finish all of the exercises during the lab period. Many of the labs are marked as optional extras to emphasize this point. If you are not an experienced programmer, you might only be able to complete one or two of the cases in each lab. That is okay. The goal of the cases is to prompt you to think through the problem, to evaluate how REXX can be used to solve the problem, and to apply the concepts you have learned as best you can. Also, you will be given a set of solutions and you will have the opportunity to download your own solutions on the last day of the course.

While you are doing the exercises, let your imagination be your guide. Feel free to go beyond the problem statement and add your own enhancements to the programs.

The solutions

We do not regard it as cheating to look at the solutions. Try to do the exercises without looking at the solutions, but there is no point in wasting time. If you get stuck, look at the solutions.

Remember that the solutions are not the only answer to the problem. They are just one way to do the exercise, and at the knowledge level of that part of the course when the exercise is presented. In some cases, the same exercise is done at different stages in the course, with different solutions. We do this to illustrate the advantages of particular programming techniques.

You will find the solutions in the data set called
`'D80WW.ES52V5.SOLUTION.EXEC.'`

Some of the advanced and optional exercises use additional resources that we do not expect you to create, during such a short course as this. Those resources are in the partitioned data set called
`'D80WW.ES52V5.RESOURCE.EXEC'`.

There are some additional data sets that you will allocate and use in the *REXX complier, REXX in batch, and MVS console commands* (batch REXX)

and *System REXX and external environments* exercises (SYSREXX System REXX):

- D80WW.ES52V5.REXXJCL (existing)
- D80WW.ES52V5.REXXLIB (existing)
- D80WW.ES52V5.SAXREXEC (existing)
- TSOCH££.REXCEXEC
- TSOCH££.REXXJCL
- TSOCH££.REXXLIB
- TSOCH££REXXMOD
- TSOCH££.REXXOBJ
- TSOCH££.REXXOUT

General hints

These hints apply to all REXX programming, such as you will do after this course. Come back and look at these hints as you progress with the labs. They will make more sense as you do the labs.

1. At any time, while coding a program, use REXXTRY to work out a method of doing something in REXX.
2. This is a general method of writing a program, that applies to all procedural programming languages:
   a. Write down what your program has to do and how it will do it.
   b. Expand this functional description of the program into a flow diagram with pseudo-code.
   c. Keep expanding the pseudo-code until you have written the program.

TSO user IDs for the course

In the first lab you will log on to the TSO user ID supplied to you for this course. One user ID has been supplied for each student.

The user IDs are of the format TSOX%nn where "%nn" is a letter followed by two numbers. Your assigned ID is in the Course Lab Kit.

For all systems, the usual RACF rules apply, and you might have to change the password before you can log on. Note your new password. IBM has installed a standard RACF exit to enforce additional password rules. The password you create must conform to the IBM internal rules. These are:

- The password must be 8 characters in length.
- The password must contain at least one numeric character.
- The password must contain at least one alphabetic character.
- The first and last position of the password cannot be numeric.
- The new password must not contain four consecutive characters of your current password.
- The password must not contain more than two consecutive repeating characters.

- The password must not contain your user ID.

Procedure for logging on to TSO/E

Once you are logged on to TSO/E, remember this rule: whenever you see three asterisks ("***") at the end of all data on the screen, all you can do is read the screen and press enter to display the next screen of data. Anything else you type will be ignored.

Optional exercises

There are some optional parts to some of the exercises and also some optional exercises in this exercise guide (**exercises 12-19**). Once you have completed the required exercises you can work on the optional exercises. They might help to round out the hands-on experience for a related unit.

Exercise 1. Preparation of the TSO environment

Overview

A major issue of TSO/E is the allocation environment. You will learn more about this later in the course. The usual requirement in running REXX programs (normally called “execs”) is that they are members of a partitioned data set in the //SYSEXEC DD concatenation. Your execs will be placed into a data set called <tsoid>.ES52.EXEC, where <tsoid> is your TSO/E logon user ID. This data set already exists on the system. There are many different ways to arrange that one of your own personal libraries is added to a TSO/E concatenation at log on time, and your own site will have its own way of doing this. The method described here is very convenient and works at this site.

Objectives

At the end of this exercise, you should be able to:

- Log on to TSO/E
- Modify your ISPF environment
- Allocate data sets to be able to execute REXX execs

Introduction

In this exercise, you will log on to a TSO/E system and customize your execution environment.

Exercise instructions

Part 1: Logging on and customizing ISPF

__ 1. Your assigned ID is in the Course Lab Kit, which also provides instructions for logging on to the system.

__ 2. Go to ISPF Option 6.

__ 3. At the ISPF Option 6 panel, enter the following command:
`EXEC 'D80WW.ES52V5.RESOURCE.EXEC(SETUP)' EXEC`

This allocates your `ES52.EXEC` data set and the '`D80WW.ES52V5.RESOURCE.EXEC`' data set to the //SYSEXEC DD concatenation. Doing this will allow you to implicitly execute REXX execs from these data sets.

You *must* execute this same command each time you logon to TSO/E.

__ 4. Exit from ISPF to save the EXEC command you just entered at option 6, so that you will not have to completely retype the command the next time you logon.

__ 5. Reenter ISPF, and go to Option 0 (Settings):

```
Log/List Function keys Colors Environ Workstation Identifier Help
 ------------------------------------------------------------------------
                                ISPF Settings
 Command ===> ____________________________________________________________
                                                                More:   +
 Options                                  Print Graphics
   Enter "/" to select option               Family printer type 2
 _  Command line at bottom                  Device name . . . . _______      /
Panel display CUA mode                 Aspect ratio  . . . 0
 _  Long message in pop-up
 _  Tab to action bar choices
 /  Tab to point-and-shoot fields       General
 /  Restore TEST/TRACE options            Input field pad . . N
 _  Session Manager mode                  Command delimiter . ;
 /  Jump from leader dots
 _  Edit PRINTDS Command
 /  Always show split line
    _  Enable EURO sign

                                                                    Member
list options                                                     Enter "/" to
select option                        / Scroll member list
_  Allow empty member list                                          _  Allow
empty member list (nomatch)
```

__ 6. On the left side of the main Settings panel, customize your ISPF settings to work the way that you want. Here are some suggestions:

__ a. Place the command line at the top or bottom of the screen, as you want, by adding or removing a slash on the “Command line at bottom” line.

__ b. Leave the “Long message in pop-up” line blank in order to have the long message display on one line of the screen instead of in a pop-up window.

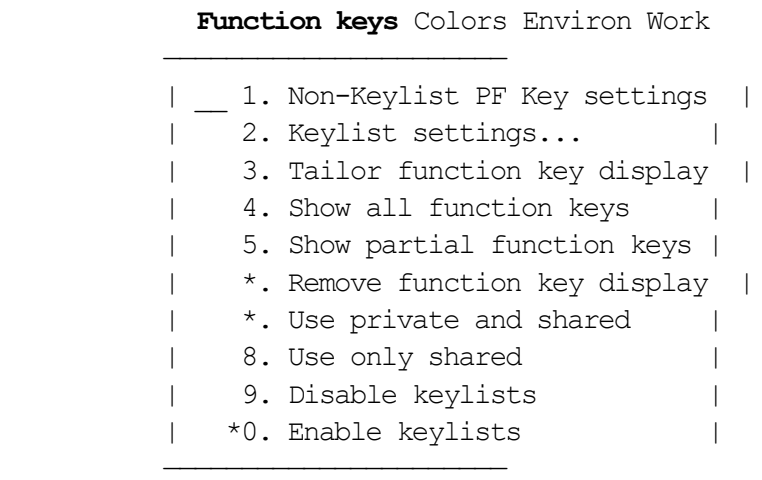
__ c. Leave the “Tab to action bar choices” line blank if you do not want the cursor to jump to the action bar when you press the “Home” key.

__ d. Place a slash on the “Tab to point-and-shoot fields” line so that you can tab to those fields on various panels.

__ e. Place a slash on the “Jump from leader dots” line so that you can execute the jump function from any field within an ISPF panel, not just from ISPF arrows (==>).

__ f. Place a slash on the “Always show split line” line so that a line of dots will appear between screen images when you are in split screen mode.

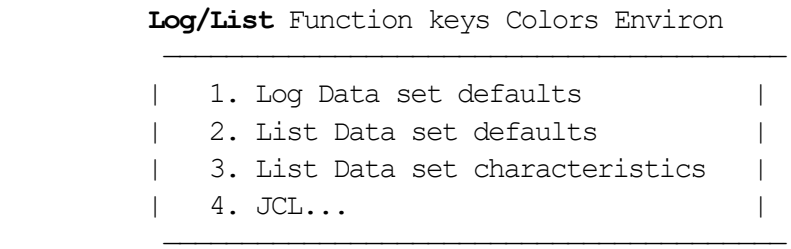
__ 7. Move the cursor up to the Function Keys action bar item and press Enter to display the drop down menu:



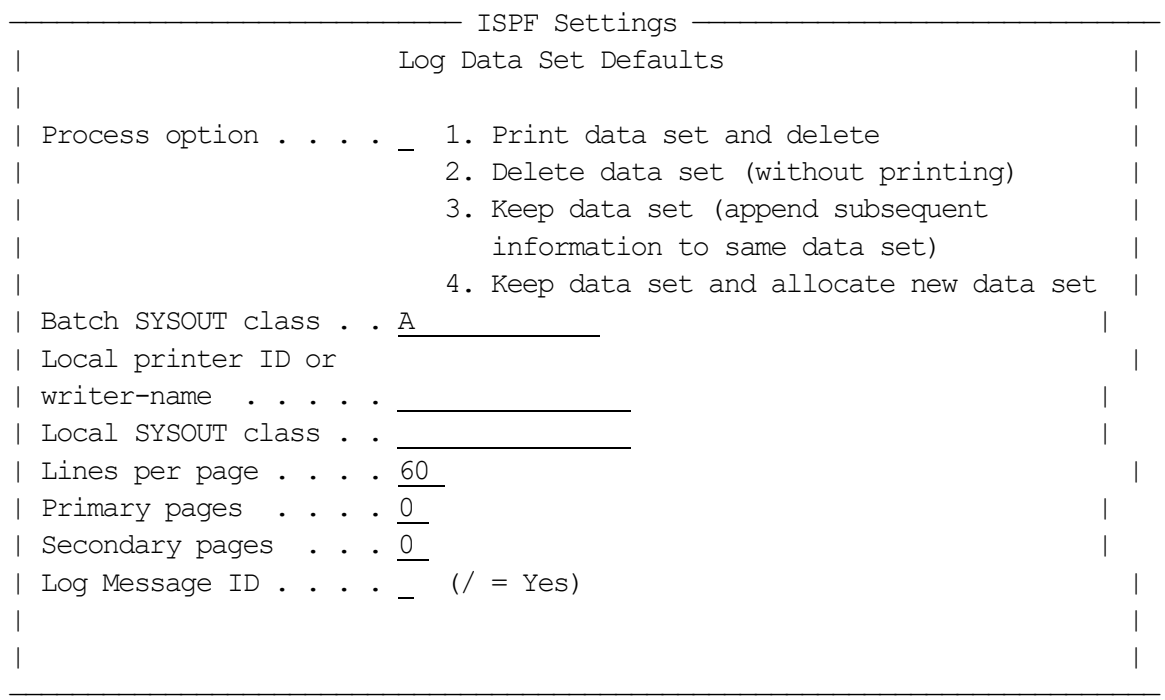
__ 8. Select option **9**, “Disable keylists” and press Enter, so that you will have the same PF key settings on each panel you visit.

__ 9. If you want to remove the PF key definitions from the bottom of the screen, enter the command **PFSHOW OFF**.

__ 10. Move the cursor up to the Log/List action bar item and press Enter to display the drop down menu:



__ 11. Select item **1**, Log Data set defaults, and press Enter. You should see the following window:



__ 12. Select Process option **2**, Delete data set, so that the Log data set will automatically be deleted when you exit from ISPF. You can also specify “Primary pages” and “Secondary pages” of 0 so that the log data set is not even allocated.

__ 13. Press PF3 to go back to the Settings panel.

__ 14. Press PF3 again, to return to the ISPF Primary Options Menu.

__ 15. Enter the command **RETP** at the command line. You should now see a window that looks like this:

```
┌──────────── Retrieve ────────────┐
│ Options  Help                    │
│ ──────────────────────────────── │
│     ISPF Retrieve Panel          │
│                                  │
│ Select the command               │
│ to be retrieved                  │
│                  More:     +     │
│     1.  =0                       │
│     2.  LOG                      │
│     3.  =e.8                     │
│     4.  exec 'd80ww.ES52V5.>     │
│     5.  6                        │
│     6.                           │
│     7.                           │
│     8.                           │
│     9.                           │
│     10.                          │
│     11.                          │
│     12.                          │
│     13.                          │
└──────────────────────────────────┘
```

__ 16. This window contains a list of the previously entered commands. When you press PF12 to retrieve old commands to the command line, they come from this list. It is easier to retype commands of very few characters than to press PF12 many times to recall the command. Also, short commands clutter up this list. Move the cursor up to select the Options action bar item from this panel and press Enter to display the drop down menu:

```
┌──────────────────────────────────────────────────────────────────┐
│   1. Set minimum number of characters saved in retrieve stack    │
│   2. Select cursor position for retrieve                         │
│   3. Exit                                                        │
└──────────────────────────────────────────────────────────────────┘
```

__ 17. Select option **1**, Set minimum number of characters saved in retrieve stack, and press Enter.

__ 18. Change the minimum number of characters saved to **6** and press Enter.

__ 19. Press PF3 to go back to the **Primary Option Menu**. Now, only commands of 6 characters or more will be saved in the retrieve stack.

__ 20. Enter ISPF command '**swapbar**' to activate the swap bar at the bottom of your screen, for an easier navigation between the different ISPF panels that you can activate with the '**start**' command.

Part 2: Copying members to ES52.EXEC

__ 21. Go to Option **2**, the ISPF Editor.

__ 22. At the ISPF Library section of the Edit panel, type in your `ES52.EXEC` data set name, as shown:

```
   ISPF Library:
      Project . . . userid__
      Group . . . . ES52____
      Type  . . . . EXEC____

 Member . . . ________
```

Where *`userid`* is your TSO logon user ID.

__ 23. This data set is currently empty. Type in a new member name called **`RXGET`** in the Member field of the ISPF Library, and press enter. This should give you a blank screen.

__ 24. At the command line, type the command **`COPY`** and press Enter. This should take you to the “Edit - Copy” panel.

__ 25. At the “**From Other Partitioned or Sequential Data Set:**” field, type the following data set and member name, and press Enter:

```
      From Other Partitioned or Sequential Data Set:
   Data Set Name . . . 'D80WW.ES52V5.RESOURCE.EXEC(RXGET)'__________
   Volume Serial . . . ______    (If not cataloged)
```

This should copy the RXGET member from the RESOURCE data set to your data set.

__ 26. Repeat steps 23 through 25 to copy the member SAMPDS from the RESOURCE data set to your `ES52.EXEC` data set. The RXGET member calls the SAMPDS member, so you need to copy both members to your data set.

__ 27. The RXGET member is actually an ISPF Edit macro, which copies members from the ‘`D80WW.ES52V5.RESOURCE.EXEC`' data set to your data set. You will be using this tool from time to time this week in class.

__ 28. Test this member to make sure it is working. Return to the Edit Entry Panel, and type a new member name into the **ISPF Library Member** field, called **LA**. Press Enter to get the blank Edit screen.

__ 29. At the Edit command line, type the command **`RXGET`**. The LA member should be copied from the RESOURCE data set to your data set by using the `RXGET` edit macro.

__ 30. If everything is working properly, then exit from ISPF and logoff from TSO/E.

End of exercise

Exercise 2. REXX basics

Overview

This exercise provides an opportunity to use the REXXTRY exec (which the student will copy to their own data set) and the Say instruction to evaluate the result of assignments and concatenation of variables.

Part 2 of this exercise will allow the student to write a REXX exec to perform simple arithmetic and I/O operations.

Objectives

At the end of this exercise, you should be able to:

- Use REXXTRY and the Say instruction
- Understand the concatenation and abuttal operators
- Read from and write to the terminal
- Perform simple arithmetic calculations

Introduction

There will be hands-on exercises following most of the lectures. They are intended to get you to think about what was covered during the lecture, and to test yourself with regard to your understanding of the lecture materials. Some of these exercises will use the REXXTRY exec. No solutions will be given, as REXXTRY provides you with the solutions as you do the exercises. It is most important that you understand everything that you see and do during these exercises. Do not treat them as a race, but rather as an opportunity to learn. You are encouraged to try additional examples beyond those in the exercises.

Most of the steps in this exercise consist of directions to enter information into REXXTRY while it runs under TSO. Therefore, the instruction “Enter the following line and press Enter” has been omitted from most of these instructions.

In the second part of the exercise, you will write your own REXX exec to perform some simple arithmetic calculations. This forms the base math program, which will be enhanced in later exercises. Be sure to complete this part of the exercise, as later exercises depend on this one.

Exercise instructions

Part 1: Basic concepts

__ 1. Log on to your TSO user ID. Do not forget to run your SETUP exec by entering the command:
```
EXEC 'D80WW.ES52V5.RESOURCE.EXEC(SETUP)' EXEC
```
from ISPF Option 6.

__ 2. Go to ISPF Option 2.

__ 3. Create a new member called **REXXTRY**. When you get to the blank edit screen, type **`RXGET`** at the command line to copy the REXXTRY member from the RESOURCE data set to your data set.

__ 4. There are four Say instructions at about line 27 of the REXXTRY exec. Modify one or more of these Say instructions to make it unique, so that you know when you run this exec you will be running your own copy of the exec, and not the one from the RESOURCE data set.

__ 5. From the edit command line, type: **`SAVE;TSO REXXTRY`**

__ 6. From this point on, enter the following instructions interactively while REXXTRY runs. Do not add these lines to the REXXTRY exec! Observe the results:

__ a. `/* Rexx beats Java */`

__ b. `Say REXX beats java`

__ c. `Say Rexx 'beats' java`

__ d. `Say Rexx 'beats' java`

__ e. `a = Rexx`

__ f. `b = eats`

__ g. `c = java`

__ h. `Say a b c`

__ i. `Say a b b c`

__ j. `Say a 'b' b c`

__ k. `Say a 'b' || b c`

__ l. `Say a 'b'b c`

What happened?
Why?_____

__ m. `Say a b c`

__ n. `a = "Rexx" ; b = "eats" ; c = "Java beans for breakfast"`

__ o. `Say a b c`

__ p. `Say a || b || c`

__ q. `Say a || b || "fresh" c`

__ r. Say 5 + 9 * 33 - 2 / (7 + (5*45))

__ 7. Perform some other arithmetic calculations. You can use REXXTRY as a calculator in this fashion.

__ a. Say 'c1'x

__ b. Say 'c2'x

__ c. Say '1100 0001'b

__ d. Say 'C1'x

__ e. Say '1100 0011'b

__ f. Say 'C3'x

__ g. Trace R

__ 8. Continue with your own experiments until you get tired of the trace output.

__ a. Trace N

__ b. 55555

Did you get a message like:

```
IKJ56621I INVALID COMMAND NAME SYNTAX
```

or

```
INVALID COMMAND NAME SYNTAX
```

Which do you think is more informative? What do you think would cause you to get one form or the other?

__ 9. Exit from REXXTRY.

Part 2: Simple MATH exec

__ 10. Create a new member in your ES52.EXEC data set called **MATH1**.

__ 11. Write a REXX exec to do the following:

- Prompt the user to enter two numbers.
- Receive the two numbers into your exec from the keyboard.
- For any two numbers entered by the user, display the numbers, and then perform addition, subtraction, multiplication, and all three types of division on the numbers.

For example, if the user were to type the numbers 25 and 4 as input to the exec, then the exec output might be:

```
Please enter two numbers:
25 4 <==(the user enters this line)
You entered 25 and 4.
25 + 4 = 29
25 - 4 = 21
25 * 4 = 100
25 / 4 = 6.25
25 divided by 4 is 6 with a remainder of 1
```

Be sure that your exec works with *any* two numbers, not just the ones in the example above.

End of exercise

Exercise 3. REXX programming concepts

Overview

This exercise provides an opportunity to use the REXXTRY exec to experiment with the function of more REXX instructions.

In Part 2 of this exercise the student will use some of these new instructions to enhance the MATH1 exec that was written in the *REXX basics* exercise.

In Part 3 of this exercise the student will write a different type of math exec, where individual calculations will be performed.

Objectives

At the end of this exercise, you should be able to:

- Understand how the Parse Arg instruction works
- Use the If-Then-Else instruction
- Use the Select-When-Then-Otherwise instruction

Introduction

This exercise once again uses REXXTRY to experiment and test the usage of various concepts covered in the lecture. Parts 2 and 3 allow the students to apply the lessons learned to their own REXX code.

Exercise instructions

Part 1: More basic concepts

As in the previous exercise, use REXXTRY for these instructions, but this time, you will pass some arguments to REXXTRY from the command line. The instructions that follow here are just the input to your REXXTRY session.

__ 1. Be sure to start your REXXTRY session from ISPF Option 6, not the ISPF Edit panel. The Edit panel automatically converts the entire input line to upper case, while Option 6 does not.

__ a. `REXXTRY chives dill fennel sage thyme`

__ b. `Say 555 = 555`

__ c. `Say 555 = 556`

__ d. `Say 555 = 555.0000000001`

__ e. `a = 1 ; b = 2 ; c = 3`

__ f. `Say a b c`

__ g. `Say a = a`

__ h. `Say a = b`

__ i. `Say (a + b) = c`

__ j. `answer = " YES "`

__ k. `Say answer = "YES"`

__ l. `Say answer == "YES"`

__ m. `Say answer = "yes"`

__ n. `If answer = 'YES' Then Say 'The answer is yes'; Else Say 'The answer is no'`

__ 2. Reenter the previous instruction using other tests in the If instruction.

__ a. `Parse Upper Arg parms`

__ b. `Say parms`

__ c. `Parse Arg parms`

__ d. `Say parms`

Did you notice any difference between the previous two ways of getting the data into your exec?

__ e. `Parse Upper Arg a b c d e f`

__ f. `Say a`

__ g. `Say b`

__ h. `Say c`

__ i. `Say d`

__ j. Say e

__ k. Say f

__ l. Parse Arg fulllist

__ m. Parse Var fulllist item fulllist

__ n. Say item

__ o. Say fulllist

__ 3. Repeat sub-steps L to O until you understand what is happening here.

Part 2: Enhancement to MATH1 exec (MATH2)

__ 4. Create a new member in your ES52.EXEC data set called **MATH2**.

__ 5. Copy the MATH1 exec into the MATH2 member.

__ 6. Add the following enhancements to this exec:

- If two numbers are provided as command line arguments, then use them and do not request input from the user. If two numbers are not provided as command line arguments, prompt the user for the two numbers.
- If the second number is zero, then do not perform division.

  For example, if the user were to type the numbers 21 and 0 as input to the exec, then the exec output might be:

  ```
  math2 21 0 <==(the user enters this line)
  You entered 21 and 0.
  21 + 0 = 21
  21 - 0 = 21
  21 * 0 = 0
  The second number is 0: division cannot be done.
  ```

  Be sure that your exec works with *any* two numbers, not just the ones in the example above.

Part 3: MYMATH exec

__ 7. Create a new member in your ES52.EXEC data set called **MYMATH**.

__ 8. Write a REXX exec to perform the following:

__ a. Ask the user whether they want to add, subtract, multiply, or divide.

__ b. Receive the answer from the user.

__ c. Ask the user to enter two numbers.

__ d. Receive the two numbers from the user.

__ e. Perform the requested arithmetic and display the output.

For example, if the user wants to multiply the numbers 7 and 8, the exec output might be:

```
tso mymath  <==(the user enters this line)
Do you wish to add, subtract, multiply, or divide?
multiply    <==(the user enters this line)
Please enter two numbers:
7 8         <==(the user enters this line)
7 * 8 = 56
```

Be sure the exec works with *all four* types of arithmetic and *any* two numbers. Also, be sure to guard against division by zero.

End of exercise

Exercise 4. REXX loops

Overview

Part 1 of this exercise once again enhances the MATH exec. It allows the student to practice coding looping structures in a REXX exec. In Part 2, the student will build enhancements to the MYMATH exec.

Objectives

At the end of this exercise, you should be able to:

- Understand how to code loops

Introduction

This exercise builds upon the MATH2 and MYMATH execs that were written in the last exercise. It adds looping structures to these execs.

Exercise instructions

Part 1: MATH3 exec

__ 1. Create a new member in your ES52.EXEC data set called **MATH3**. Copy the MATH2 exec to your new member.

__ 2. Add the following enhancements to MATH3:

- Ensure that two arguments are received from the user, either from the command line or by prompting the user. Continue to prompt the user until two arguments are received.
- After displaying the arithmetic calculations, allow the user to enter two more numbers for further calculations. Make sure that you provide an option for the user to exit from the exec.

  For example, the exec output might look like:

```
tso math3  <==(the user enters this line)
Please enter two numbers:
10         <==(the user enters this line)
Please enter two numbers:
10 5       <==(the user enters this line)
You entered 10 and 5.
10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
10 / 5 = 2
10 divided by 5 is 2 with a remainder of 0
Please enter two numbers:
15 0       <==(the user enters this line)
You entered 15 and 0.
15 + 0 = 15
15 - 0 = 15
15 * 0 = 0
Second number is 0: division cannot be done.
Please enter two numbers:
exit       <==(the user enters this line)
Goodbye.
```

  Be sure that your exec works with *any* two numbers, not just the ones in the example above.

Part 2: MYMATH2 exec

__ 3. Create a new member in your ES52.EXEC data set called **MYMATH2**.

__ 4. Copy the MYMATH member to MYMATH2 and add the following enhancements:

- If the user does not type an appropriate type of calculation to be performed (add, subtract, multiply, or divide), display an error message and request the calculation type again.

- After displaying the requested arithmetic calculation, allow the user to request another calculation to be performed. Ask the user for the next type of calculation, and ask for two more numbers. Make sure that you provide an option for the user to exit from the exec.

  For example, the exec output might look like:

```
tso mymath2  <==(the user enters this line)
Do you wish to add, subtract, multiply, divide, or exit?
multiply        <==(the user enters this line)
Please enter two numbers:
7 8          <==(the user enters this line)
7 * 8 = 56
Do you wish to add, subtract, multiply, divide, or exit?
foo             <==(the user enters this line)
'FOO' is not an appropriate answer.
Do you wish to add, subtract, multiply, divide, or exit?
subtract        <==(the user enters this line)
Please enter two numbers:
81 49           <==(the user enters this line)
81 - 49 = 32
Do you wish to add, subtract, multiply, divide, or exit?
exit            <==(the user enters this line)
Goodbye.
```

End of exercise

Exercise 5. REXX functions (part 1)

Overview

This exercise provides an opportunity to use several REXX functions.

Objectives

At the end of this exercise, you should be able to:

- Use common REXX functions

Exercise instructions

Part 1: FUNLAB

__ 1. Create a new member in your `ES52.EXEC` data set called **FUNLAB**. At the command line type the command:

```
RXGET FUNCS
```

This will copy the FUNCS member from the RESOURCE data set to your `ES52.EXEC` data set.

__ 2. In your new FUNLAB member, there are several questions which require you to code functions to accomplish the required tasks.

End of exercise

Exercise 6. REXX functions (part 2)

Overview

In this exercise the student will use functions to enhance the MATH3 and MYMATH2 execs that were written in the *REXX loops* exercise.

Objectives

At the end of this exercise, you should be able to:

- Use common REXX functions

Exercise instructions

Part 1: MATH4 exec

__ 1. Create a new member in your ES52.EXEC data set called **MATH4**.

__ 2. Copy the MATH3 exec into the MATH4 member.

__ 3. Add the following enhancements to this exec:

- Make sure that the input arguments, whether from the command line or the keyboard, are both numbers. Continue to prompt the user for values until two numbers are received. Do not forget to retain the ability for the user to terminate the program.
- Add the date and time to the output display.

  For example, the exec output might be:

```
tso math4     <==(the user enters this line)
Please enter two numbers:
1 2 3 4       <==(the user enters this line)
Please enter two numbers:
sss aaa bb jhsdfk <==(the user enters this line)
Please enter two numbers:
1             <==(the user enters this line)
Please enter two numbers:
25 4          <==(the user enters this line)
At HH:MM:SS on YY/MM/DD you entered 25 and 4.
25 + 4 = 29
25 - 4 = 21
25 * 4 = 100
25 / 4 = 6.25
25 divided by 4 is 6 with a remainder of 1
Please enter two numbers:
exit          <==(the user enters this line)
Goodbye.
```

  Be sure that your exec works with *any* two numbers, not just the ones in the example above.

Part 2: MYMATH3 exec

__ 4. Create a new member in your ES52.EXEC data set called **MYMATH3**. Copy the MYMATH2 member to MYMATH3.

__ 5. Add the following enhancements to MYMATH3:

__ a. When you prompt the user for the type of arithmetic (add, subtract, multiply, divide), allow the user to enter an abbreviation instead of the full word. There is a function you can use to handle this.

__ b. When you ask for the two numbers, make sure you receive two numbers before you try to do arithmetic. If the values received are not numbers, display an error message and allow the user to retype the numbers again.

For example, your exec output might look like this:

```
mymath3 <==(the user enters this line)
Do you wish to add, subtract, multiply, divide, or exit?
M       <==(the user enters this line)
Please enter two numbers:
7 8     <==(the user enters this line)
7 * 8 = 56
Do you wish to add, subtract, multiply, divide, or exit?
r       <==(the user enters this line)
'R' is not an appropriate answer.
Do you wish to add, subtract, multiply, divide, or exit?
sub     <==(the user enters this line)
Please enter two numbers:
14 asdf <==(the user enters this line)
You did not enter two numbers.
Please enter two numbers:
14 56   <==(the user enters this line)
14 - 56 = -42
Do you wish to add, subtract, multiply, divide, or exit?
exit    <==(the user enters this line)
Goodbye.
```

End of exercise

Exercise 7. REXX functions or subroutines

Overview

This exercise will provide an opportunity for the students to work more with REXX built-in functions. The second part of the exercise will allow the students to create their own functions or subroutines.

Objectives

At the end of this exercise, you should be able to:

- Write a function or subroutine

Introduction

In the first part of this exercise students will create a new exec that adds a string of numbers together and returns the sum.

In Part 2 of the exercise, students will use this exec as a subroutine to another exec that they will write to calculate the average of a string of numbers.

Exercise instructions

Part 1: SUMMIT exec

__ 1. Create a new member in your `ES52.EXEC` data set called **SUMMIT**. Write a REXX exec to perform the following:

__ a. Accept a series of numbers, separated by spaces, as an input argument.

__ b. Add the numbers together, and display the sum to the screen.

__ 2. The exec should work with any number of input numbers.

For example, if you invoke the SUMMIT exec with the following command:

```
tso summit 1 3 5 11 12.5
```

then the exec output might look like this:

```
The sum of 1 3 5 11 12.5 is 32.5
```

Enhancement:

Make sure that each value in the input string is a number before you try to add it.

Part 2: AVG exec

__ 3. Modify your SUMMIT exec so that it can be called as either a function or a subroutine. The exec will:

__ a. Receive a string of numbers as an input argument.

__ b. Return the sum of the input string of numbers to the calling exec.

__ 4. Create a new member in your `ES52.EXEC` data set called **AVG**. Write an exec that will do the following:

__ a. Accept an input string of numbers, either comma or blank delimited.

__ b. Translate the commas to blanks.

__ c. Call the SUMMIT exec, passing the (now) blank delimited string of numbers.

__ d. Receive the sum of the input string as the returned value from the SUMMIT exec.

__ e. Calculate the average of the input string of numbers.

__ f. Display the input string, along with the average in an output message.

For example, if the following command is entered:

```
tso avg 1 2,3 4,5
```

The exec output will look like:

```
The average of 1 2 3 4 5 is 3
```

Be sure that your exec works with *any* input string of numbers, not just the ones in the example above.

Enhancement:

Modify the output from the AVG exec so that it looks like this:

```
The average of 1,2,3,4 and 5 is 3.
```

End of exercise

Exercise 8. Coding error recovery routines

Overview

This exercise will provide an opportunity for the students to write recovery code which will enhance the quality of your execs.

Objectives

At the end of this exercise, you should be able to:

- Code basic error recovery instructions
- Understand the error recovery requirements of an exec

Introduction

The recovery routines you add to your execs will provide a basic level of quality for your execs.

Exercise instructions

Part 1: Error handling code

1. Go back to your MATH2 and MYMATH execs and place the REXX keyword instruction
   ```
   Signal On Syntax
   ```
   at the beginning of each exec.
2. Write a Syntax error handling routine in MATH2 and MYMATH. Test these routines to make sure they work properly.
3. From now on, add a Signal On Novalue instruction and appropriate recovery code to each exec that you write.

Enhancement:

Add Signal instructions and proper error routines to some of the other execs you have already written.

End of exercise

Exercise 9. Compound variables, data stack, and executing host commands

Overview

Part 1 of this exercise provides students an opportunity to use the REXXTRY exec to practice the use of compound variables and the data stack.

Part 2 of this exercise will allow students to use REXXTRY to execute host commands and capture their responses.

In part 3 students will write a REXX exec to issue host commands, and capture and manipulate the responses.

Objectives

At the end of the exercise, you should be able to:

- Use compound variables
- Use and manage the data stack
- Execute host commands from a REXX exec
- Capture and modify the output from a host command

Introduction

As with the *REXX basics* and *REXX programming concept* exercises, the first two parts of this exercise consist of directions to enter information into REXXTRY while it runs under TSO/E. Therefore, the instruction “Enter the following line and press Enter” has been omitted from most of these instructions.

In the third part of the exercise, you will write your own REXX exec to execute a TSO/E host command. You will capture the output from this command and modify parts of it for display on the terminal screen.

Exercise instructions

Part 1: Basic concepts

__ 1. Log on to your TSO/E user ID. Do not forget to run your SETUP exec by entering the command:
```
EXEC 'D80WW.ES52V5.RESOURCE.EXEC(SETUP)' EXEC
```
from ISPF Option 6.

__ 2. There is a utility exec in the RESOURCE data set called **SS2**. It will show you an image of your data stack. You will use this exec to see what the stack looks like at several points in the exercise.

__ 3. Bring up the REXXTRY exec. From this point on, enter the following instructions interactively while REXXTRY runs. Do not add these lines to the REXXTRY exec! Observe the results:

```
__ a. Call OUTTRAP 'line.'
__ b. 'avg 1 5 9 17 23 8'
__ c. 'math2 25 14'
__ d. Call OUTTRAP 'OFF'
__ e. Say line.0
__ f. Say line.1
__ g. Say line.2
```

__ 4. Repeat for as many times as line.0 indicated.

```
__ a.  Do i = 1 To line.0; Say line.i; End i
__ b.  Push 'The rain'
__ c.  Push 'in Spain'
__ d.  ss2
__ e.  Say Queued()
__ f.  'MAKEBUF'
__ g.  Push 'Stays mainly'
__ h.  Push 'in the plain'
__ i.  ss2
__ j.  Say Queued()
__ k.  Parse Pull x; Say x
__ l.  ss2
__ m.  Parse Pull x; Say x
__ n.  ss2
__ o.  Say Queued()
```

__ p. Parse Pull x; Say x

__ q. ss2

__ r. Say Queued()

__ s. Parse Upper Pull x; Say x

__ t. Say Queued()

__ u. Queue 'The rain'

__ v. Say Queued()

__ w. Queue 'in Spain'

__ x. 'MAKEBUF'

__ y. Say Queued()

__ z. Queue 'Stays mainly'

__ aa. Say Queued()

__ ab. Queue 'in the plain'

__ ac. ss2

__ ad. Say Queued()

__ ae. 'DROPBUF'

__ af. ss2

__ ag. Do Queued();Parse Pull x;Say x;End

__ ah. Say Queued()

__ ai. Push 'We have met'

__ aj. Push 'the enemy'

__ ak. ss2

__ al. 'NEWSTACK'

__ am. ss2

__ an. Queue 'and he'

__ ao. 'MAKEBUF'

__ ap. Queue 'is us'

__ aq. 'MAKEBUF'

__ ar. ss2

__ as. 'DROPBUF 1'

__ at. ss2

__ au. 'DELSTACK'

__ av. ss2

__ aw. Do Queued();Parse Pull;End

__ ax. ss2

__ 5. Make sure Stack 1 is empty. Exit from REXXTRY.

Part 2: Executing host commands

__ 6. Bring up REXXTRY and enter the following instructions interactively while REXXTRY runs. Do not add these lines to the REXXTRY exec! Observe the results:

__ a. 'LISTCAT'

__ b. Call OUTTRAP 'rec.'

__ c. 'LISTCAT'

__ d. Call OUTTRAP 'OFF'

__ e. Say rec.0

__ f. Say rec.1

__ g. Say rec.2

__ h. Say rec.3

__ i. period = Pos('.',rec.2)

__ j. nxtpd = Pos('.',rec.2,period + 1)

__ k. hqual = Substr(rec.2,1,period - 1)

__ l. mqual = Substr(rec.2,period + 1,nxtpd - 1 - period)

__ m. Say "The high level qualifier of" rec.2 "is" hqual

__ n. Say "The next qualifier is" mqual

__ 7. Exit from REXXTRY.

Part 3: MYTIME exec

__ 8. Create a new member in your ES52.EXEC data set called MYTIME.

__ 9. Write a REXX exec that will perform the following:

__ a. Capture the output from the TSO/E `TIME` command.

__ b. Display the current time of day, the amount of time the user's session has been in the system, the amount of CPU time consumed, and the number of service units consumed, in separate messages.

For example, if the user is logged on to the user ID TSOCH00, the exec output might look like:

```
The current time is 04:11:00 PM.
 TSOCH00 has been logged on for 01:57:41
 The session has used 00:00:02 CPU time
 and 23525 service units.
```

End of exercise

Exercise 10.REXX I/O

✓Overview

This exercise will provide an opportunity for the students to work with the I/O facilities in REXX, the EXECIO command, and the REXX stream I/O functions.

Objectives

At the end of this exercise, you should be able to:

- Use the TSO/E EXECIO command to perform I/O from a REXX exec
- Use the REXX stream I/O functions to perform I/O from a REXX exec

Introduction

In the first part of this exercise students will create a new exec that uses the TSO/E EXECIO command to read from a data set and display the data on the screen in two different ways.

In Part 2 of the exercise, students will use the REXX Stream I/O functions to read and write data to and from data sets and the terminal screen.

Exercise instructions

Part 1: EXECIO command

1. Create a new member in your `ES52.EXEC` data set called **LISTIT1**. Write a REXX exec to perform the following:
   a. Accept a sequential data set name or partitioned data set and member name as input.
   b. Verify that the data set exists by using the SYSDSN() function, and issue an error message if it does not.
   c. Read the data set records into compound variables by using the `EXECIO` command.
   d. Display the records on the screen with Say instructions.
2. Create another new member in your `ES52.EXEC` data set called **LISTIT2**. Copy the LISTIT1 member to LISTIT2, and make the following modifications:
   a. Read the data set into the data stack by using the `EXECIO` command.
   b. Display the records on the screen by using the `EXECIO` command.

Hint

To allocate the terminal screen as the output location, code an asterisk (*) as the data set name in the TSO/E `ALLOCATE` command: `'ALLOC FI(OUTPUT) DA(*) SHR REUSE'`

Part 2: Stream I/O functions

3. Create a new member in your `ES52.EXEC` data set called **LISTIT3**. Write a REXX exec to perform the following:
   a. Accept a sequential data set name or partitioned data set and member name as input.
   b. Verify that the data set exists with the SYSDSN() function, and issue an error message if it does not.
   c. Read the records from the data set with the Linein() function.
   d. Write the records from the data set to the terminal screen with the Lineout() function.
4. Create a new member in your `ES52.EXEC` data set called **LISTIT4**. Write an exec that will do the following:
   a. Prompt the user to enter lines of input data from the keyboard.
   b. Read the lines of data into the REXX exec with the Linein() function.
   c. Use the Lineout() function to write the data typed by the user to a new member of your `ES52.EXEC` data set called **STRMIO**.

d. Close the input and output streams when done.

End of exercise

Exercise 11. REXX compiler, REXX in batch, and MVS console commands

Overview

Part 1 of this exercise provides students an opportunity to use the REXX compiler to create compiled exec (CEXEC) code.

Part 2 and 3 of this exercise will allow students to execute REXX execs in batch jobs.

In part 4 students will write a REXX exec to issue MVS console DISPLAY commands, and capture and manipulate the responses.

Objectives

At the end of this exercise, you should be able to:

- Invoke the REXX compiler and create CEXEC code
- Execute REXX execs in the background by using JCL
- Execute MVS console commands from within a REXX exec

Exercise instructions

Part 1: REXX compiler

1. Frist of all, logoff and Log on to your TSO user ID.
2. Do not forget to run your SETUP exec by entering the command:
   EXEC 'D80WW.ES52V5.RESOURCE.EXEC(SETUP)' EXEC
   from ISPF Option 6.
3. Set your ISPF prefix to your TSO userid with ISPF Option 6 command:
   PROF PREF(TSOCHxx)
4. Enter command TSO PROF to display and verify that it was set successfully.

```
IKJ56688I CHAR(0)  LINE(0)    PROMPT   INTERCOM   NOPAUSE MSGID   NOMODE  NOWTP
MSG NORECOVER PREFIX(TSOCH95)  PLANGUAGE(ENU) SLANGUAGE(ENU) VARSTORAGE(LOW)
 IKJ56689I DEFAULT LINE/CHARACTER DELETE CHARACTERS IN EFFECT FOR THIS TERMINAL
 ***
```

5. Create a new member in your `ES52.EXEC` data set called **FACTRL**. When you get the blank editor screen type **RXGET** at the command line to copy this member from the RESOURCE data set to your data set.
6. Jump to ISPF option 3.4. At the “DSNAME LEVEL” line, type your user ID (TSOCHxx). Press enter to display a list of your data sets.
7. Tab the cursor down the left margin until it is at the line with your `ES52.EXEC` data sets. Enter the “**M**” command to display the members of this data set.
8. At the member list, move the cursor down the left margin until you reach the line with the new FACTRL member.
9. Type the command:
   REXXC
   next to the FACTRL member and press enter. This should invoke the REXX compiler against this member.

```
REXXC starting
Compiling     'TSOCH95.ES52.EXEC(FACTRL)'
PRINT output  'TSOCH95.ES52.FACTRL.LIST'
CEXEC output ***
Compiler return code is 4
 ***
```

10. When the compiler is done, press PF3 to return to the data set list.

    Where is the compiled exec data set?
    TSOXA13.ES52.CEXEC

    Where is the listing data set?
    TSOXA13.ES52.FACTRL.LIST

11. Refresh your data set listing by either typing the **REFRESH** command at the command line or by exiting and reentering the data set list. Your new data sets should show up now.

12. Browse the `.CEXEC` data set. There should be a member called **FACTRL**. If you browse this member you will see that it contains machine code, but not like a regular load module. This is the CEXEC code.

13. Browse the listing data set.

14. When you reach the source code listing, there are three columns to the left of the source code. They are labeled "**If**", "**Do**", and "**Sel**".
    What do you think "Sel" stands for?

    _____

    What are these three columns for?_____

15. Go to ISPF option 6.

16. In the *REXX basics* exercise you created a member called **LA**. Compile this member by typing the following command:
    `REXXC ES52.EXEC(LA) XREF`

17. When the compiler is finished, jump back to option 3.4 and list your data sets again.

18. Browse the `.CEXEC` data set. There should now be two members in this data set, **FACTRL** and **LA**.

19. Browse the listing data set for the LA compile.

20. Note the cross reference listing that is included in this listing data set.

21. Execute the LA member from ISPF Option 6 using the TSO/E **`EXEC`** command. It will produce a listing of the data sets that are allocated to your TSO/E session, along with their ddnames.

```
EXEC 'TSOCHxx.ES52.CEXEC(LA)'
```

Part 2: REXX in batch

22. Create a new member in your `ES52.EXEC` data set called **JCL1**. At the blank Edit screen command line type the command:
    `BATCHRX`
    This will copy a set of JCL statements from the RESOURCE data set to your data set.

23. Modify the JCL stream to execute the program IKJEFT1A.

24. Add a **`PARM=`** parameter to the EXEC statement to execute the REXXTRY exec.

25. Under the **//SYSTSIN DD *** statement, add the following lines:

```
Say "Hello World!"
Say Hello World
math3 25 14
12 73
exit
LISTALC STATUS
QSTACK
Say rc
```

26. Submit the job. Go to SDSF to view the output. SDSF is accessed via option SD on the primary menu.

```
SD SDSF          System Display and Search Facility
```

27. Under SDSF type ST, and scroll down (PF8) to select the last job that is displayed to view its output.

28. Go back to the JCL1 member and change the program name on the EXEC statement from IKJEFT1A to IRXJCL. Do not make any other changes.

29. Submit this job, and view the output in SDSF. What is different? _____

_____

Why?

_____

Part 3: Compile and link in batch

Before you start this part, you will have to allocate some data sets in which we will copy some sample jobs. If you look at `D80WW.ES52V5.RESOURCE.EXEC(ALLOCLIB)`, you will see how this REXX exec allocates some unique user data sets (REXXLIB, REXCEXEC, REXXOBJ, REXXOUT, REXXMOD).

30. Goto ISPF option 6 TSO, and execute this REXX exec.

```
Enter TSO or Workstation commands below:

 ===> ex 'D80WW.ES52V5.RESOURCE.EXEC(ALLOCLIB)'
```

You should receive the following messages:

```
rc for  'TSOCH££.REXXJCL' allocation is  0
rc for  'TSOCH££.REXXLIB' allocation is  0
rc for  'TSOCH££.REXCEXEC' allocation is  0
rc for  'TSOCH££.REXXOBJ' allocation is  0
rc for  'TSOCH££.REXXOUT' allocation is  0
rc for  'TSOCH££.REXXMOD' allocation is  0
NONVSAM ------- TSOCH££.REXCEXEC
     IN-CAT --- ICFCAT.MVS100.UCAT.STUD1
NONVSAM ------- TSOCH££.REXXJCL
     IN-CAT --- ICFCAT.MVS100.UCAT.STUD1
NONVSAM ------- TSOCH££.REXXLIB
     IN-CAT --- ICFCAT.MVS100.UCAT.STUD1
NONVSAM ------- TSOCH££.REXXMOD
     IN-CAT --- ICFCAT.MVS100.UCAT.STUD1
NONVSAM ------- TSOCH££.REXXOBJ
     IN-CAT --- ICFCAT.MVS100.UCAT.STUD1
NONVSAM ------- TSOCH££.REXXOUT
     IN-CAT --- ICFCAT.MVS100.UCAT.STUD1
NONVSAM ------- TSOCH££.TEMPDATA
     IN-CAT --- ICFCAT.MVS100.UCAT.STUD1
***
```

Now you will populate the data sets with some sample REXX execs and jobs, with a sample REXX exec that is in `D80WW.ES52V5.RESOURCE.EXEC(`**`ES52LABS`**`)`. You can browse its contents, then when you are ready, execute it under ISPF 6 TSO.

```
Enter TSO or Workstation commands below:


===> ex 'D80WW.ES52V5.RESOURCE.EXEC(ES52LABS)'
```

You should receive:

```
Populating sysrexx lab datasets...
TSOCH££.REXXLIB...
TSOCH££.REXXJCL...
job submitted to populate REXXLIB and REXXJCL....
***
```

Your data sets `TSOCH££.REXXLIB` and `TSOCH££.REXXJCL` have been populated with some sample REXX execs and JCL members.

__ 31. To make sure that your REXXLIB will be allocated as an active CLIST/REXX data set, we will use the TSO **ALTLIB** function. Look at member TSOCH££.REXXLIB(**ALTLIB**), and see how we will achieve that.

__ 32. Now goto ISPF option 6 (TSO) and enter the above REXX exec:

```
===> ex 'TSOCH££.REXXLIB(ALTLIB)'
```

You should be back to the ISPF primary menu.

__ 33. To ensure that your REXXLIB is activated, enter TSO command:

TSO LISTA ST

At the three asterisks (***) prompt, press enter several times until you see:

```
TSOCH££.REXXLIB
  SYS00204 KEEP
TSOCH££.REXXLIB
  SYS00205 KEEP
D80WW.ES52V5.RESOURCE.EXEC
  ISP11402 KEEP,KEEP
D80WW.ES52V5.RESOURCE.EXEC
  ISP11403 KEEP,KEEP
D80WW.ES52V5.RESOURCE.EXEC
  ISP11404 KEEP,KEEP
D80WW.ES52V5.RESOURCE.EXEC
  ISP11405 KEEP,KEEP
TSOCH££.REXXMOD
  ISP11406 KEEP,KEEP
REXX.SEAGLPA
           KEEP,KEEP
```

This is the evidence that the `ALTLIB ACT` command has been activated: now you can execute the REXX execs in this data set by calling them directly, without the need to specify the data set name.

Example:

```
TSO SAYHELLO (or SAYHELLO under opt6)
```

instead of:

```
tso ex 'TSOCH££.REXXLIB(SAYHELLO)' (or ex 'TSOCH££.REXXLIB(SAYHELLO)'
under opt6)
```

Important

All the remaining steps in this exercise which involve ISPF editing and macros must be done from this new ISPF primary menu.

If, at any time, you exit from this ISPF primary panel, you will lose your active ALTLIB, so you will have to reactivate it by reexecuting the `TSOCH50.REXXLIB(`**`ALTLIB`**`)` REXX exec.

__ 34. Try it: goto option 6, and enter: **sayhello** and **sayhelo1**.

```
sayhello
+----------------------+
| hello TSOCH££ !      |
+----------------------+

***
sayhelo1
+----------------------+
| hello TSOCH££ !      |
| hello TSOCH££ !      |
+----------------------+

***
```

__ 35. Look at the content of both REXX execs in TSOCH££.REXXLIB. There is one small difference.

```
(Say '| hello' SYSVAR(SYSUID) '! )
```

Now we will invoke both REXX execs in batch mode.

__ 36. Edit JCL member TSOCH££.REXXJCL(REXXEX1); it invokes SAYHELLO with program IKJEFT01; to add a jobcard we have provided an edit macro, in TSOCH££.REXXLIB(**JC**). Because the REXXLIB data set is an active ALTLIB, you can invoke the edit macro directly while in edit mode.

__ 37. While in edit mode in TSOCH££.REXXJCL(REXXEX1), type '**JC**' to invoke this edit macro whose purpose is to add some JCL cards (jobcard + comments).

```
EDIT       TSOCH££.REXXJCL(REXXEX1) - 01.05              Columns 0
Command ===> jc                                            Scroll
****** ***************************** Top of Data ******************
000013 //*
000014 //*-------------------------------------------------------------
000015 //* purpose
000016 //*  Execute a source rexx program (not compiled): SAYHELLO
000018 //* with IKJEFT01
000023 //*-------------------------------------------------------------
000024 //ZT000        EXEC PGM=IKJEFT01,PARM='SAYHELLO'
000026 //SYSEXEC      DD DISP=SHR,DSN=TSOCH££.REXXLIB
000027 //SYSTSPRT     DD SYSOUT=*
000028 //SYSPRINT     DD SYSOUT=*
000029 //SYSTSIN      DD DUMMY
000030 //*  end of job
```

The result is:

```
EDIT       TSOCH££.REXXJCL(REXXEX1) - 01.06               Columns
Command ===>                                                Scrol
****** **************************** Top of Data ******************
000001 //TSOCH££W JOB (TSOCH££),'TSOCH££',CLASS=A,MSGCLASS=Q,
000002 //         NOTIFY=TSOCH££,REGION=0M,MSGLEVEL=1
000003 //*
000004 //*
000005 //*       -DOC- TSOCH££     22 Apr 2016     12:20:51
000006 //*       -LIB- TSOCH££.REXXJCL(REXXEX1)
000007 //*
000008 //*       -PURPOSE -_____________________________________
000009 //*                 _____________________________________
000010 //*                 _____________________________________
000011 //*
000012 //*
000013 //*
000014 //*-------------------------------------------------------------
000015 //* purpose
000016 //*  Execute a source rexx program (not compiled): SAYHELLO
000018 //* with IKJEFT01
000023 //*-------------------------------------------------------------
000024 //ZT000        EXEC PGM=IKJEFT01,PARM='SAYHELLO'
000026 //SYSEXEC      DD DISP=SHR,DSN=TSOCH££.REXXLIB
000027 //SYSTSPRT     DD SYSOUT=*
000028 //SYSPRINT     DD SYSOUT=*
000029 //SYSTSIN      DD DUMMY
000030 //*  end of job
```


__ 38. Perform a global edit change of TSOCH££ to your actual user ID: You can either do a '**C TSOCH££ TSOCH££ all**' or invoke edit macro '**CHANGES**' in your REXXLIB which does the same.

```
EDIT       TSOCH££.REXXJCL(REXXEX1) - 01.06              Columns 0
Command ===> changes                                       Scroll
****** ***************************** Top of Data ******************
000001 //TSOCH££W JOB (TSOCH££),'TSOCH££',CLASS=A,MSGCLASS=Q,
000002 //         NOTIFY=TSOCH££,REGION=0M,MSGLEVEL=1
000003 //*
000004 //*
000005 //*       -DOC- TSOCH££     22 Apr 2016     12:20:51
000006 //*       -LIB- TSOCH££.REXXJCL(REXXEX1)
000007 //*
000008 //*       -PURPOSE -_____________________________________
000009 //*                 _____________________________________
000010 //*                 _____________________________________
000011 //*
000012 //*
000013 //*
000014 //*-------------------------------------------------------
000015 //* purpose
000016 //*  Execute a source rexx program (not compiled): SAYHELLO
000018 //* with IKJEFT01
000023 //*-------------------------------------------------------
000024 //ZT000        EXEC PGM=IKJEFT01,PARM='SAYHELLO'
==CHG> //SYSEXEC     DD DISP=SHR,DSN=TSOCH££.REXXLIB
000027 //SYSTSPRT    DD SYSOUT=*
```

__ 39. Now submit the job which execute REXX exec SAYHELLO In batch, using program IKJEFT01.

__ 40. Goto SDSF (option 'SD' on the primary menu) to see your job output: type **ST** under SDSF; you should see a list of jobs starting with your user ID; the last one (bottom) on the list was the one you just submitted; select it (**S**) to view its output; goto the bottom, you should see:

```
+-----------------------+
| hello TSOCH££ !       |
+-----------------------+
```

__ 41. Repeat steps 37 to 41('JC' then 'CHANGES', and submit) with job TSOCH££.REXXJCL(REXXEX2) which executes REXX exec SAYHELO1.

The output should be:

```
+-----------------------+
| hello TSOCH££ !       |
| hello TSOCH££ !       |
+-----------------------+
```


__ 42. Repeat the same with members REXXEXE1 and REXXEXE2, which also invoke both REXX execs in batch mode, but through IRXJCL and not IKJEFT01. Instead of running IKJEFT01, you can also run IRXJCL – this is the "*REXX in batch*" module.

__ 43. Any difference?

REXXEXE2 will fail with:

```
IRX0043I Error running SAYHELO1, line 7: Routine not found
7 +++ Say '| hello' SYSVAR(SYSUID) '!        |'
```

IRXJCL runs in an MVS address space, and the default host command environment is MVS. The TSO, ISPEXEC, ISREDIT, and CONSOLE host command environments are not available when using IRXJCL or running in an MVS address space. The SYSVAR() function in SAYHELO1 is a TSO/E REXX function. It is not available in IRXJCL.

Now you are going to compile those two REXX execs.

__ 44. Edit member `TSOCH££.REXXJCL(`**`REXXCCM1`**`)` to compile SAYHELLO. Enter edit macro '**`JC`**' to add a jobcard, and '**`CHANGES`**' to change ££ to your actual user ID suffix.

__ 45. Submit the job; you should expect RC=**0**.

__ 46. Repeat this step with `TSOCH££.REXXJCL(`**`REXXCCM2`**`)` to compile SAYHELO1.

__ 47. Both execs have been compiled and generated in a CEXEC data set: **`TSOCH££.REXCEXEC`**.

__ 48. Use ISPF 3.4 to browse its content.

```
Command ===>
           Name     Prompt       Size
_________ SAYHELLO
_________ SAYHELO1
          **End**
```

__ 49. Select those members to browse them: the content is not readable.

```
BROWSE    TSOCH££.REXCEXEC(SAYHELLO)              Line 0000000000 Col 001 080
 Command ===>                                          Scroll ===> PAGE
******************************** Top of Data ********************************
å..ÐEXECPROCEAGRTPRC Compiled REXX   4.0   22 Apr 2016 13:58:55 MVS REXXC370 3.4
8 23 Dec 1999 LVL PK04822 TSOCH££.REXXLIB(SAYHELLO)
    ... ...Ù°.}..¬ì^&u.§j.^.å.&Ö.. .&.jØ.²å.&Q.-....å0&S 0..i0...ö.õì0&yå}é..ÃjØ
.²åØé. .é. ...i......Âh-.. ....Âì0{.á\0..& ....Âåø&. .&...q.}..Ú..............
...ì...&.......°...Q...Y...........Ø...°... ...ì..... .......^.............^
...¯...H...M...\...Ö...8...È...°...ì...4...ì...........m...½.......}...µ...Ð...ü
.......D.........................".Ø...µ..... ... ...°...}.......}...«...-
...½.............-...h.......h.......h.......h.......q... ...Q.......Q.......0
...Y...Q...%...ç...è...µ.......µ.......µ.......µ.......½.....................
 øNYì0AMá\0. øO.ì0{¯á\0. øNÀì0{{á\0. øNøì0{{á\0. øN<ì0{µá\0Ç Ø©ç øN ì0{½á\0 ì0{{
á\0.ìøµ^ì0{4á\0. øNøì0{{á\0. øNÀì0{{á\0.ìø^<.'oØø.ì0{0á\0.....................
.....................+------------------------+.............| hello....
```

__ 50. Enter '**ex**' in front of the member names to execute the REXX execs.

```
BROWSE             TSOCH££.REXCEXEC
 Command ===>
             Name      Prompt        Size   Cr
 ex_______ SAYHELLO
 _________ SAYHELO1
           **End**

REXXC370 3.48 23 Dec 1999
+-----------------------+
| hello TSOCH££ !       |
+-----------------------+

***
```

Now, try to invoke the compiled REXX exec in batch mode:

__ 51. Edit TSOCH££.REXXJCL(**REXXCEX1**), enter edit macro '**JC**' to add a jobcard, and '**CHANGES**' to change ££ to your actual user ID suffix. See how the compiled version is invoked with **'EX 'TSOCH££.REXCEXEC(SAYHELLO)''**.

__ 52. Submit the job; you should expect RC=**0**.

__ 53. Repeat the same with TSOCH££.REXXJCL(**REXXCEX2**) for exec SAYHELO1.

Now we are going to compile and link the REXX execs, and create an executable load module.

__ 54. First of all, edit the **PROCEDURE TO COMPLILE/LINK AND CATALOG A REXX PROGRAM** in member TSOCH££.REXXJCL(**ZREXXCL**), enter edit macro '**CHANGES**' to change ££ to your actual user ID suffix, and save it away (PF3).

__ 55. Edit TSOCH££.REXXJCL(**REXXCL1**) to compile and link SAYHELLO, enter edit macro '**JC**' to add a jobcard, and '**CHANGES**' to change ££ to your actual user ID suffix. '

__ 56. Submit the job; you should expect RC=**0**.

__ 57. Repeat the same with TSOCH££.REXXJCL(**REXXCL2**) for exec SAYHELO1.

__ 58. Submit the job; you should expect RC=**0**.

__ 59. If you use ISPF 3.4 to look at data set TSOCH££.REXXMOD, you will see that two load modules have been linkedited.

```
BROWSE             TSOCH££.REXXMOD
Command ===>
            Name      Prompt        Alias-o
_________ SAYHELLO
_________ SAYHELO1
          **End**
```

__ 60. You can invoke the programs with the '**call**' line command.

```
BROWSE             TSOCH££.REXXMOD
 Command ===>
             Name      Prompt
 _call____ SAYHELLO
 _________ SAYHELO1
           **End**
REXXC370 3.48 23 Dec 1999

+-----------------------+
| hello TSOCH££ !       |
+-----------------------+

***
```

Now we are going to execute those programs in batch mode.

__ 61. Edit member `TSOCH££.REXXJCL(`**`XREXXCL1`**`)`, and see how it is different with a linked REXX exec: We call it with PGM= and a STEPLIB/JOBLIB containing the program (loadmodule).

```
//* purpose                                                          *
//*  . Execute a rexx program compiled/linked (cataloged): SAYHELLO  *
//*                                                                   *
//*-------------------------------------------------------------------*
//JOBLIB       DD DISP=SHR,DSN=TSOCH££.REXXMOD
//             DD DISP=SHR,DSN=REXX.SEAGLPA
//*-------------------------------------------------------------------*
//ZT000        EXEC PGM=SAYHELLO
//SYSTSPRT     DD SYSOUT=*
//*-------------------------------------------------------------------*
```

__ 62. Enter edit macro '**JC**' to add a jobcard, and '**CHANGES**' to change ££ to your actual user ID suffix.

__ 63. Submit the job; you should expect RC=**0**.

```
REXXC370 3.48 23 Dec 1999

+-----------------------+
| hello TSOCH££ !       |
+-----------------------+
```

Repeat the same with `TSOCH££.REXXJCL(`**`XREXXCL2`**`)` for exec SAYHELO1.

Was the execution successful? No, you should get RC 3659.

```
EAGREX4300E Error 43 running compiled ?, line 9: Routine not found
+----------------------+
| hello TSOCH££ !      |
     9 +++
```

Because we are trying to execute a REXX exec which invokes TSO/E services in a non TSO environment (PGM=) it fails.

__ 64. Instead, try to invoke this REXX loadmodule using IKJEFT01 and STEPLIB: Edit member `TSOCH££.REXXJCL(XREXXCL4)`, enter edit macro '**`JC`**' to add a jobcard, and '**`CHANGES`**' to change ££ to your actual user ID suffix.

__ 65. Submit the job; you should expect RC=**0**.

```
REXXC370 3.48 23 Dec 1999

+----------------------+
| hello TSOCH££ !      |
| hello TSOCH££ !      |
+----------------------+
```

The same should also work with `TSOCH££.REXXJCL(`**`XREXXCL3`**`)` for exec SAYHELLO.

Part 4: MVS console commands

__ 66. Create a new member in your `ES52.EXEC` data set called **CONS**. Write a REXX exec to perform the following:

__ a. Define a console profile so that solicited messages are not displayed at the console screen. Store up to 200 messages in the message table.

__ b. Enable your TSO/E console session.

__ c. Switch to the CONSOLE host command environment.

__ d. Set a CART value for each command you are going to enter.

__ e. Execute the following MVS commands:

```
DISPLAY ASM
 DISPLAY IPLINFO
 DISPLAY TS,TSOCHxx
```

(where *`TSOCHxx`* is your TSO/E user ID)

__ f. End the console session and switch back to the TSO/E host command environment.

__ g. Receive the output from each command into compound variables.

__ h. Display the following information:

- The current time of day.
- The time and date the system was last IPLed.
- The percent that the PLPA data set is full and the data set name.

- The address space ID, CPU time, and elapsed time of your TSO session.

  For example, the output from this exec might look like:

```
It is now hh:mm:ss
 The system was last IPLed at hh:mm:ss on mm/dd/yy
 The PLPA data set, SYS1.xxxx.PLPA, is xx% full.
 TSOCHxx is running in Address Space ID nnnn,
 and has been logged on for xx hours, yy minutes, and
 zz seconds, using xx.yyy CPU seconds.
```

Note

Your TSO userid may not be allowed to execute a CONSOLE command, due to RACF missing privileges, in which case you will receive the following error messages, and the EXEC will not execute successfully.

```
IKJ55303I THE CONSOLE COMMAND HAS TERMINATED.+
 IKJ55303I AN ERROR OCCURRED DURING CONSOLE INITIALIZATION.  THE MCSOPER
RETURN
CODE WAS X'00000004' AND THE REASON CODE WAS X'00000000'.
```

End of exercise

Exercise 12.Parsing data (optional)

Overview

This exercise will provide an opportunity for the students to work with the REXX Parse instruction.

Objectives

At the end of this exercise, you should be able to:

- Use the REXX Parse instruction to parse data in a variety of different ways

Introduction

In the first part of this exercise students will use the REXXTRY exec to test various types of data parsing by using the Parse instruction.

In Part 2 of the exercise, students will copy the PARS exec into their own data set, calling the new member PARSLAB. They will then modify the PARSLAB exec to achieve the wanted results according to instructions within the exec, much like the *REXX functions* exercise (FUNLAB).

Exercise instructions

Part 1: REXXTRY

__ 1. Bring up the REXXTRY exec and enter the following instructions to test various types of parsing:

```
__ a. Quote = 'Experience is the best teacher.'
__ b. Parse Var quote word1 word2 word3
__ c. Say word1
__ d. Say word2
__ e. Say word3
__ f. Parse Var quote word1 word2 word3 word4 word5 word6
__ g. Say word1
__ h. Say word2
__ i. Say word3
__ j. Say word4
__ k. Say word5
__ l. Say word6
__ m. Parse Var quote word1 word2 15 word3 word4
__ n. Say word1
__ o. Say word2
__ p. Say word3
__ q. Say word4
__ r. Parse Var quote 15 v1 +16 =12 v2 +2 1 v3 +10
__ s. Say v1
__ t. Say v2
__ u. Say v3
__ v. Parse Var quote 1 v1 +11 v2 +6 v3 -4 v4
__ w. Say v1
__ x. Say v2
__ y. Say v3
__ z. Say v4
__ aa.delim = 7
__ ab.Parse Var quote 1 v1 (delim) v2
__ ac.Say v1
```


__ ad. `Say v2`

__ ae. `Parse Var quote 1 v1 =(delim) v2 +6 v3`

__ af. `Say v1`

__ ag. `Say v2`

__ ah. `Say v3`

Part 2: PARSLAB exec

__ 2. Create a new member in your `ES52.EXEC` data set called **PARSLAB**. At the command line type the command:

`RXGET PARS`

This will copy the PARS member from the RESOURCE data set to your `ES52.EXEC` data set.

__ 3. In your new PARSLAB member, there are several questions which require you to code parsing templates to accomplish the required tasks.

Part 3: Wrapup

This is a five-part exercise that builds upon everything you have learned so far.

Solutions can be found in **`D80WW.ES52V5.SOLUTION.EXEC`**.

__ 4. Exercise 1:

__ a. For this exercise create member **EX111** in your REXX library.

- **Goal:** Display the name of the current system name and level of z/OS.
- **Hint:** Use the **`Say`** command and the MVSVAR function.

__ b. Save the REXX program and then run it by issuing from the ISPF Command Prompt TSO %EX1 or split the screen and option ISPF option 6 and issue **`%EX111`**.

__ 5. Exercise 2:

__ a. For this exercise create member **EX112** in your REXX Library.

- **Goal:** Issue any TSO command from within a REXX Program. For example, `LISTD 'SYS2.PARMLIB'`.
- **Hint:** Use ARG or PARSE ARG to get the wanted TSO command and options from the command line.
- **Note:** A variable will be processed before it is executed.

__ b. Save the REXX program and run it.

__ 6. Exercise 3:

__ a. For this exercise create member **EX113** in your REXX Library.

- **Goal:** The same as Step 5. Exercise 2, but this time capture the output and display it inside a loop.
- **Hint:** Use OUTTRAP and a DO END loop.

__ b. Save the REXX program and run it.

__ 7. Exercise 4:

__ a. For this exercise create member **EX114** in your REXX Library.

- **Goal:** The same as Step 6. Exercise 3, but this time instead of displaying the results by using a loop we will use the ISPF Browse service.
- **Hint:** Use TSO `Allocate` and `Free`. Use `EXECIO` to write the trapped results. Use Address ISPEXEC Browse to view it.

__ b. Save the REXX program and run it.

__ 8. Exercise 5:

__ a. For this exercise create member **EX115** in your REXX Library.

- **Goal:** Test the existence of a data set passed to the program and display information about it.
- **Hint:** Save the status (from SYSDSN) in a variable to test and then use LISTDSI for more information.

__ b. Save the REXX program and run it.

End of exercise

Exercise 13.LISTDD (optional)

Overview

This optional exercise allows the student to practice capturing and manipulating TSO/E command output.

Objectives

At the end of this exercise, you should be able to:

- Trap TSO/E command output and use compound variables

Exercise instructions

Part 1: LISTDD exec

__ 1. Create a new member in your `ES52.EXEC` data set called **LISTDD**. Write a REXX exec that will format the display of ddnames and data set names from the `LISTALC STATUS` command.

__ 2. Trap the output from the `LISTALC STATUS` command.

__ 3. Reformat the output so as to make it more readable. Present the output in two columns, one for ddnames, and the other for data set names.

__ 4. Ensure that the columns align neatly (there is a function for that).

__ 5. Give the columns headings.

Hint

It is important to study with care the output of the LISTALC STATUS command.

Enhancements:

__ 6. Allow for one optional argument on the command line.

__ 7. If the argument is a dollar sign ($) then prompt for a specific ddname, and display the data set concatenation for only that ddname and no others.

End of exercise

Exercise 14.MAVG (optional)

Overview

This optional exercise allows the student to practice calling one exec from another.

Objectives

At the end of this exercise, you should be able to:

- Understand how to invoke a REXX exec from another REXX exec

Exercise instructions

Part 1: MAVG exec

__ 1. Create a new member in your `ES52.EXEC` data set called **MAVG**. Write a REXX exec that will perform the following:

__ a. Accept a series of numeric inputs, either comma or blank delimited, separated by colons (:). For each set of input numbers MAVG will invoke the AVG exec, passing the string of numbers. AVG will then display the average of each set of numbers.

For example, if the user enters the command:

```
MAVG 1,2,3:4 5:6 3,1,2
```

The output should look like:

```
The average of 1,2 and 3 is 2.
 The average of 4 and 5 is 4.5.
 The average of 6,3,1 and 2 is 3.
```

End of exercise

Exercise 15.MPROB (optional)

Overview

This optional exercise allows the student to practice calling one exec from another.

Objectives

At the end of this exercise, you should be able to:

- Understand how to invoke a REXX exec from another REXX exec

Exercise instructions

Part 1: MPROB exec

__ 1. Create a new member in your `ES52.EXEC` data set called **MPROB**. Write a REXX exec that will perform the following:

__ a. Accept a series of different inputs, either comma or blank delimited, separated by colons (:). The first word of each input string is the name of the exec MPROB is to invoke, and the rest of the input string are the arguments for that exec.

For example, if the user enters the command:

```
MPROB AVG 1,2,3:SUMMIT 4 5 8 3:MATH1 6 3
```

Then the output should look like:

```
The average of 1,2 and 3 is 2.
 The sum of 4 5 8 3 is 20
 You entered 6 and 3
 6 + 3 = 9
 6 - 3 = 3
 6 * 3 = 18
 6 / 3 = 2
 6 divided by 3 is 2 with a remainder of 0
```

End of exercise

Exercise 16.PUTID (optional)

Overview

This optional exercise allows the student to create an ISPF edit macro.

Objectives

At the end of this exercise, you should be able to:

- Write an ISPF Edit macro

Exercise instructions

Part 1: PUTID macro

__ 1. Create a new member in your `ES52.EXEC` data set called **PUTID**. Write a REXX exec that will place a comment on the first line of a member being edited, including the word REXX and the name of the member:

__ a. Using an ISPF Edit service to find the name of the member being edited.

__ b. Build a comment line including the word REXX and the member name, like:

`/* Rexx exec <execname> */` (where *execname* is the member name)

__ c. Add this line as the first line of the member.

Hint

There is an ISPF Edit macro command called `LINE_AFTER` that inserts data in the specified place in the member being edited.

Hint

The ISPF Edit macro command `MEMBER` will return the current member name.

Hint

Edit macros are invoked differently from normal execs: normal execs are TSO/E commands and are invoked from the edit command line by prefixing them with the edit command TSO. Edit macros are entered as ISPF edit commands without the TSO prefix. They may only be executed from the Edit panel. Do not forget to indicate to ISPF that this is an Edit macro by using the ISPF Edit macro command MACRO!

End of exercise

Exercise 17.QUERYDS (optional)

Overview

This optional exercise allows the student to gain an understanding of the LISTDSI() function.

Objectives

At the end of this exercise, you should be able to:

- Use the LISTDSI() function in a REXX exec

Exercise instructions

Part 1: QUERYDS exec

__ 1. Create a new member in your `ES52.EXEC` data set called **QUERYDS**. Write a REXX exec that will accept a data set name as an input argument and display the following information about that data set:

- __ a. Record format (RECFM)
- __ b. Logical record length (LRECL)
- __ c. Last referenced date
- __ d. Allocation space and units of allocation
- __ e. Space allocations used
- __ f. Data class, management class, storage class
- __ g. Data set type (sequential, PDS, PDSE, other)
- __ h. If a sequential data set, display the first ten lines of data

Hint

The LISTDSI() function is described in the *TSO/E REXX Reference* manual.

End of exercise

Exercise 18.CALCUL8R (optional)

Overview

This optional exercise allows the student to get practice using the Interpret instruction.

Objectives

At the end of this exercise, you should be able to:

- Use the REXX Interpret instruction

Exercise instructions

Part 1: CALCUL8R exec

__ 1. Create a new member in your `ES52.EXEC` data set called **CALCUL8R**. Write a REXX exec that will accept an arithmetic expression of any length and compute an answer. The exec will:

__ a. Accept as input an arithmetic expression in algebraic form, using the REXX arithmetic operators (**, *, /, %, //, +, -).

__ b. Continue to prompt the user for input if none is given.

__ c. Display appropriate error messages for any invalid input (like dividing by zero or using letters as input values).

__ d. After displaying the output for the calculation, continue to prompt the user for more arithmetic until an appropriate termination option is entered.

For example, the exec output might look like:

```
calcul8r        <==(The user types this line)
Please enter an arithmetic expression or 'EXIT':
42 / 7 + 6 * 4 <==(The user types this line)
42 / 7 + 6 * 4 = 30
 Please enter an arithmetic expression or 'EXIT':
a * 17          <==(The user types this line)
Error processing arithmetic expression: a * 17
Perhaps you didn't enter numeric values or proper arithmetic
operators?
 Try again.

Please enter an arithmetic expression or 'EXIT':
exit            <==(The user types this line)
Goodbye
```

End of exercise

Exercise 19.GETJNAME (optional)

Overview

This optional exercise allows the student to get some practice using the TSO/E REXX Storage() function and the data conversion functions.

Objectives

At the end of this exercise, you should be able to:

- Use the TSO/E REXX Storage() function to follow MVS control block chains
- Use the REXX data conversion functions

Exercise instructions

Part 1: GETJNAME exec

__ 1. Create a new member in your `ES52.EXEC` data set called **GETJNAME**. Write a REXX exec that will display your job name (address space name), which is pointed to by the address space control block (ASCB) in storage.

__ a. Location ‘10'x contains the address of the Communications Vector Table (CVT). All searches through MVS control block chains start here.

__ b. Offset 0 in the CVT is the address of the NEW/OLD pointer.

__ c. Offset ‘C'x from the NEW/OLD pointer is the address of the ASCB.

__ d. Offset ‘B0'x in the ASCB is the address of the jobname.

__ e. The jobname field is eight characters in length. This will be the same as your userid for an interactive TSO/E session.

For example, the exec output might look like:

```
getjname    <==(the user types this line)
The CVT address is 00FD4400
The NEW/OLD pointer address is 00000218
The ASCB address is 00FB2480
The JOBNAME address is 00FA4CB0
The job name is XXXXXXX (where XXXXXX is the address space name)
```

Hint

1. A storage address is four bytes long.
2. The address used in the first argument of the storage function must be in hexadecimal, and the length argument must be in decimal.
3. The data returned from the storage function is in character format, that is, no conversion is performed on it by REXX.
4. Remember that REXX only does decimal arithmetic. To calculate the offsets from the start of a control block, you must convert the address to decimal first (C2D()), then add the decimal value of the offset (X2D()), then convert the result back to hexadecimal for input to the next storage function (D2X()).
5. The address found at offset 0 of the CVT will be ‘218'x, or 536 in decimal.
6. When debugging the exec, you can display the hexadecimal value of non-displayable character data by using the C2X() function.
7. ‘C1'x is character data. ‘C1' can be used as a hex value.
8. Like all of the exercises, you can make use of REXXTRY to experiment and understand the process.

End of exercise

IBM Training