REXX - part 2

Prof. Chris GauthierDickey

COMP 2400 - Unix Tools

UNIVERSITY OF DENVER

Functions and Subroutines

- Rexx has them both
  - A function must return a value
  - A subroutine may return a value
- Rexx uses 'CALL' to call a function/subroutine
  - call foo a, b, c

UNIVERSITY OF DENVER

Returns

- Rexx uses 'RETURN' to return a value
  - return 2+x
- Rexx also sets a special variable called 'RESULT' that is set when you call 'return'

```
Call FOO

Say result

exit
FOO:
   Say 'Running FOO'
   RETURN 5
```

Calls are like GOTOs

- Notice that one must protect all of your functions with an exit
  - Otherwise the parser will run your code--your labels are just goto labels
- With procedures, it's an error to not exit before the procedure definitions

UNIVERSITY OF DENVER

Procedures

- Procedures are more like 'normal' functions you may be used to
  - Variables outside the procedure are hidden
  - You can expose those external variables using 'expose'
  - You must still exit before your procedure definitions
- Procedures can use recursion

UNIVERSITY OF DENVER

An Example

- Notice, the ; lets us put more than one command on a line

```
j=1; x.1='a'
call foo

say j k m

exit

foo:procedure expose j k x.j
  say j k x.j
  k=7; m=3
return
```

- This will output "1 K a" followed by "1 7 M"
- expose lets the function see and modify j, k and x.1 (since j=1)

Rexx built-in functions

- max(a,b,c,d,...): lets you find the max of any list of numbers
- substr(name,1,1): lets you find a substring of a string
- time(): returns the time
- userid(): returns the current user ID
- length(str): returns the length of a string
- word(str, n): returns the nth word of a string str
- words(str): returns the number of words in a string
- random(min,max,seed): returns a psuedo-random value

More Rexx built-ins

- Rexx has about 70 built-in functions
- Do a 'man rexx' to find out what they all are!

UNIVERSITY OF DENVER

Another example

- Functions can take up to 10 arguments
- We get each argument by arg(1), arg(2), ... arg(10)
- arg() gives the number of arguments

```
result=sum(1,2,3,4,5)

say 'The sum is:' result

exit
sum:procedure
  r=0
  DO i = 1 to arg()
    r=r+arg(i)
  END
  return r
```

UNIVERSITY OF DENVER

Rexx's Stack

- Rexx has a multi-purpose stack/queue-like structure that all programs can use
- The stack can be passed between programs
- Choose a discipline and stick to it: either FIFO or LIFO!
- Don't touch other people's stack/queue!

Rexx's Stack

- We access the stack with
  - PUSH: pushes something to the beginning of the stack
  - PULL: removes something from the beginning of the stack
  - QUEUE: puts something on the end of the stack

An Example

- The stack is unnamed
- Notice that when we pull, we pull to a variable

```
#!/usr/bin/regina
push 1
push 2
push 3

say queued() 'items on the stack'

pull t
say 't is' t
```

Compound Variables

- Compound variables are simply variables with a "." that lets you assign to parts of it
  - a.x=1, a.y=2, a.b.x=3
- Strange things you can do with it:

```
#!/usr/bin/rexx

a.b=5

say a.b

c=b

a.c=6

say a.b
```

Stems

- Stems are a way to refer to all sub-parts of a compound variable
- Imagine: a.x=3, a.y=2, a.z=1
- You can set them all to zero using: a.=0

More Parsing

- Parsing lets you break apart user input, strings, and arguments
- PARSE UPPER PULL fname lname
  - This function calls parse, tells it to uppercase everything, and uses pull to read from the keyboard
  - The results are split into fname and lname

We can parse strings

- PARSE VALUE 'hello world' WITH w1 w2
  - This will parse from the value 'hello world' and put the results in w1 w2
- s='hello world'; PARSE VAR s w1 w2
  - Will parse on a variable called 's'
- We can also specify the matching pattern:
  - s='10:23:35'; parse var s h ':' m ':' s

More parsing

- We can specify variables for the patterns, we just have to enclose them in ()
  - k=':'; t=10:23:35; parse var t h (k) m (k) s
- Parsing from a position:
  - parse value '123456' with 4 k; say k
    - The result will be 456
  - parse value '1234567' with 4 k +2 m
    - The result will be 45 followed by 67