Quiz #6 Topics

- cut
- paste
- sort
- tail
- head

A note about quotes in UNIX

```bash
% set a=ls
% echo a
% echo $a
% set b="$a"
% echo $b
% echo "$a"
% echo $b
% echo "$a"
% echo $b
```

Output of `pnmfile`

```bash
% more test_data.pgm
P2
3 3
255
1 2 3
4 5 6
7 8 9
% pnmfile test_data.pgm
test_data.pgm: PGM plain, 3 by 3 maxval 255
```

Setting the result of `pnmfile` into a shell variable

```bash
% set b=`pnmfile test_data.pgm`
% echo $b
test_data.pgm: PGM plain, 3 by 3 maxval 255
% echo $b[4]
3
% echo $b[6]
3
```

Internally Documenting Your IDL Routines

```bash
setenv IDL_DIR /cis/common/rsi/idl_5
setenv IDL_PATH
 \$IDL_DIR/lib:\$IDL_DIR/examples:
 \$IDL_DIR/lib/idl
\dirs/common/rsi/idl_5:\dirs/common/lib/idl:
~:/lib/idl
```
### doc_library example

;+  ; The following routine computes the mean  
; of an arbitrary sized array  
;  ; $Header$  
;  ; $Log$  
; ;  
; function mean,x  
 ...  

### Executing doc_library

IDL> doc_library  
Name of procedure or * for all: mean  
Enter 1 for printer, 0 for terminal: 0  

----- Documentation for  
/cis/staff/rvrpci/lib/idl/statistics/mean.pro -----  
The following routine computes the mean  
of an arbitrary sized array

### Executing doc_library

$Header:  
/nfs/cis/staff/rvrpci/lib/idl/statistics  
/RCS/mean.pro,v 1.1 1998/01/13 15:44:47 rvrpci Exp $  
$Log: mean.pro,v $  
Revision 1.1 1998/01/13 15:44:47 rvrpci  
Initial revision

### Debugging in IDL

### Basic IDL Debugging Commands

breakpoint  
.step (.s)  
.stepover (.so)  
.continue  
.return  
.out

### mean.pro

10  ;-  
11  ;  
12 function mean,x  
13 n = n_elements(x)  
14 answer = sum(x)/double(n)  
15 return, answer  
16 end
Breakpoints

IDL> breakpoint, 'mean.pro', 12
IDL> help/breakpoint
Breakpoints:
<table>
<thead>
<tr>
<th>Index</th>
<th>Module</th>
<th>Line</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MEAN</td>
<td>12</td>
<td>mean.pro</td>
</tr>
</tbody>
</table>

OR

IDL> breakpoint, /clear, 0

OR

IDL> breakpoint, /clear, 'mean.pro', 12

.step

• Used after a breakpoint is reached
• Single step through commands
• If statement is a function or procedure
  – Will enter into it

.stepover

• Similar to .step
• Executes functions and procedures to completion without stopping inside the function

.return

• Continues execution until a return statement is encountered.
• Useful in checking the return value of an inner (nested) function in a function composition
• e.g.

  \[
  \text{sqrt(variance(x))}
  \]

.out

• Continues execution out of the current routine.
• Useful in leaving a routine that you stepped into, but have determined that all is working properly and want to return to the calling routine.

mean.pro

```idl
10  ;-
11  ;
12  function mean, x
13    n = n_elements(x)
14    answer = sum(x)/double(n)
15    return, answer
16  end
```
More Widget Examples

Display PGM Widget Program

- Illustrates the use of the pickfile function
- How to create a draw widget for image display
- How to access the draw widget and display the image

The QUIT Button

pro main_event, event
end
pro quit_event, event
    Widget_Control, event.top, /destroy
end
pro quit_button
    base=Widget_Base(/row)
    button=Widget_Button( base, value='Quit', event_pro='quit_event')
    Widget_Control, base, /realize
    Xmanager, 'quit_button', base, event_handler=$
        'main_event'
end

Components of Display PGM Widget

- read_pgm procedure
- pickfile function
- size function
- wset procedure
- tv procedure
- widget_control procedure
- widget_draw function
- xmanager procedure

Display PGM Widget Definition

pro simple_image_display
    base = Widget_Base{ column=1 }
    button=Widget_Button( base, $value='Display PGM Image'
    window1 = Widget_Draw(base, xsize=256, $ysize=256 )
    Widget_Control, window1, /realize
    Widget_Control, window1, get_value=window_id
    Widget_Control, base, set_uvalue=window_id
    Xmanager, 'simple_image_display', base, Event_handler='simple_image_display_event'
end
Widget Event Handler

```plaintext
pro simple_image_display_event, event
    widget_control, event.id, get_value=widget_value
    case widget_value of
        'Display PGM Image': begin
            image1_file = pickfile()
            read_ppm, image1_file, image1_data
            image_size = size(image1_data)
            x_size = image_size(1) & y_size = image_size(2)
            widget_control, event.top,
            get_uvalue=window_id
            wset, window_id & tv, image1_data
        end
        ELSE:
        endcase
    end
```

Pickfile Dialog Box

Resulting Widget Image Display

Mystery Images

Given an image of unknown origin

- Dimensions of the image?
  - Most importantly the number of samples
- Aspect ratio of the image?
- Number of bytes per pixel?
- Number of bands of the image?
- Number of bits per pixel?
- External or internal compression applied
  - gzip
  - tiff or jpeg

Tools at your disposal for dissecting the image

- file
- anytopnm
- more
- ls -l
- bc -l
- od
- dd
- rawtopgm or cat
- xv
Can you guess the dimensions of the image?
- Try to guess the number of rows (width) first.
  - Possible guesses
    - 256
    - 512
    - $2^n$
    - $2^n + 2^{n-1}$

Can you guess the aspect ratio of the image?
- `wc -c` or `ls -l`
  - Determine how many bytes are in the image
- Square root of the image size
  - Multiple of a perfect square
- Assume that the image is greyscale & square
  - Use `rawtopgm`

```
% rawtopgm 512 512 image.raw > image.pgm
```

Square Gray Scale Image
- Used `rawtopgm` to create a pgm file and then use `xv` to display

A convenient little trick
- If you are just experimenting with a mystery image

```
% rawtopgm 512 512 image.raw | xv -
```

Does it have a header?
- Not a perfect square
  - Go ahead and try to display it as if it were a square image to see if it has a header.

Square Grey Scale image with a header
- This structure means you have a header in the file that is causing a uniform shift
- You can fix this by using `dd`
Square image no header
• Your guess on the width of the image is short by one pixel

Square image no header
• Your guess on the width of the image is long by one pixel

Square image no header
• You probably guessed a little too low on the width of this image
• (off by 6 pixels)

Square Image with no header
• Your probably guessed a little too high on the width of the image
• (off by 6 pixels)

A color image displayed as greyscale image
• Correct dimensions – 256x256x3 BIP
• Incorrectly assumed – 443x443 image with header
• N.B.
  196608/3=256^2
• Indicates a multiband image

Multiband Assumption
• Once multiband image is assumed
  – Need to determine interleaving
  – Can be deduced by displaying first band
  – This assumes you have a guess of image dimension
• Extract the first band out using `dd`
First band of color image file assumed to be BSQ

- Assume a square image for first band
- Color image file obviously not BSQ

Square color image displayed as a greyscale (3:1 aspect ratio)

- Original image 3x256x256
- Displayed it as a greyscale 768x256
- To get insight into interleaving structure.
- BIP file would look like the left image

Square color image displayed as greyscale (3:1 aspect ratio)

- This is how a BIL image would behave
- What we have here is the individual bands side by side

Square color image displayed as greyscale (3:1 aspect ratio)

- This is how a BSQ file would behave

Multibyte pixel image

- Need to be careful about the endian of the machine on which the image was written
- Can use `dd` to swap bytes for short integer images (2-bytes per pixel) or the `swap_endian` function in IDL

Mystery Image Examples

- Check Webpage for sources of examples.
- An example will be given for the midterm exam.