

**Bio 595**

**Computers in Biomedical Research**

**Fall 2003**

**Slides for Wednesday, November 19**

# Subroutine modules and libraries

**Build your own collection of reusable subroutines.**

**Save the files in one or more subdirectories, then employ the Perl function 'use', which reads the subroutine file.**

**But note: the last line in your subroutine must, however, be '1;' or it won't work.**

**Your 'use' function might look something like this:**

**use lib 'c:/Perl/MySubLib'; (this gives path to directory)**

**use mysub1; (this is the subroutine you want to use)**

# The Perl debugger

**Note the `-w` suffix on the first line of your Perl programs provides reporting of many (e.g. syntax) errors; equivalent to `'use warnings;'`**

**Simple strategy to find errors: use `'print'` statements to show intermediate results.**

**Shorthand debugger notation:**

**a 48 print "\$i \$k\n"**

**Says to print values of variables `$i` and `$k` before line 48.**

**Use `ex18` as a sample program to debug: it's designed to report on the sequence data found after two specified bases are located.**

# The debugger

To start the debugger, type ‘-d’ switch to Perl on the command line:

```
perl -d ex18.txt
```

To stop the debugger, use ‘q’ or ^D.

Get help through ‘h’ or ‘h h’; more help still from ‘man perldebug’.

Page through the commands with | h (prints out a page at a time; hit spacebar to move to next page).

```
DB<1> h h
```

List/search source lines:

**l [ln|sub]** List source code

**- or .** List previous/current line

**v [line]** View around line

**f filename** View source in file

**/pattern/?patt?** Search forw/backw

**M** Show versions of modules

# Debugger commands

## Debugger controls:

- o [...]** Set debugger options
- <[<|{|}]|>[>]** [cmd] Do pre- post-prompt
- ! [N|pat]** Redo a previous command
- H [-num]** Display last num commands
- = [a val]** Define/list an alias

## Control script execution:

- T** Stack trace
- s [expr]** Single step [in expr]
- n [expr]** Next, steps over subs
- <CR/Enter>** Repeat last n or s
- r** Return from subroutine

# Debugger commands

<b>c [ln/sub]</b>	<b>Continue until position</b>
<b>L</b>	<b>List break/watch/actions</b>
<b>t</b>	<b>toggle trace [trace expr]</b>
<b>b [ln]event[sub] [cnd]</b>	<b>set breakpoint (B is delete a/all breakpoints)</b>
<b>a [ln] cmd</b>	<b>Do cmd before line (A is delete all actions)</b>
<b>w expr</b>	<b>Add a Watch expression (W is delete all watches)</b>
<b>h</b>	<b>Get help on command</b>
<b>h h</b>	<b>Complete help page</b>
<b>  [   ]</b>	<b>Send output to pager</b>
<b>q or ^D</b>	<b>Quit</b>
<b>A or W</b>	<b>Delete all actions/watch</b>
<b>! [ ! ] syscmd</b>	<b>Run cmd in a subprocess</b>
<b>R</b>	<b>Attempt a restart</b>

# Debugger commands

## Data examination:

- expr** Execute Perl code, also see: s, n, t expr
- x/n expr** Evals expr in list context, dumps the result or lists method
- p expr** Print expression (uses script's current package)
- S [ [ ! ] pat]** List subroutine names [not] matching pattern
- V [Pk [Vars]]** List Variables in package. Vars can be ~pattern or !pattern.
- X [Vars]** Same as “V current\_package [Vars]”.

For more help, type **h cmd\_letter**, or run **man perldebug** for all docs.

**DB<2>**

# The debugger

**Note that in debug mode, when started, the debugger stops at the first line of code. This shows the line it's about to execute next, not the line it just executed.**

**Proceed to hit n or s: both will execute the line just displayed(Difference: n skips plunges into subroutine calls, while s enters subroutines and single-steps through them as well. After this, hit enter/return to repeat the same command.**

**Try to debug ex18; there are actually two bugs in the program. The output should be "AAGGCGA"**

# Random numbers

**Can be used to simulate mutation of DNA.**

**Computer models to explore evolution, genetic disease, DNA repair mechanisms.**

**We can (1) pick a random line and location in a file, (2) generate random DNA sequences, and (3) model repeated mutation of DNA to simulate effects of mutations accumulating over time during evolution.**

**Random number generator: subroutine to output pseudo-random numbers, which exhibit a uniform distribution of numbers.**

**Pick a “seed” for the subroutine: use the time function.**

# ex19

```
#!/usr/bin/perl -w
# ex19: demonstrating use of a random number generator
# to randomly select words of nonsense sentences.
use strict;
use warnings;
# Declare the variables
my $count;
my $input;
my $number;
my $sentence;
my $story;
# Here are the arrays of parts of sentences:
```

## ex19 continued

```
my @nouns = (  
  'DNA',  
  'A Peptide',  
  'A billion nucleotides',  
  'Several amino acids',  
  'Sulfuric acid',  
  'Professor Paolini',  
  'Joe',  
  'Moe',  
);  
my @verbs = (  
  'mutated to',  
  'was sequenced and identified',  
  'added kerosene',  
  'fragmented',  
  'began to dissolve',  
  'exploded',  
  'was morphed into',  
);  
my @prepositions = (  
  'into a weapon of mass destruction',  
  'over the rainbow',  
  'through centrifugation',  
  'at the beach',  
  'before the weekend',  
  'at the Salk Institute',  
  'in a dream',  
  'around the world',  
);
```

## ex19 continued

```
# Seed the random number generator.
# time|$$ combines the current time with the current process id
# in a somewhat weak attempt to come up with a random seed.
srand(time|$$);
# This do-until loop composes six-sentence "stories".
# until the user types "quit".
do {
    # (Re)set $story to the empty string each time through the loop
    $story = '';
    # Make 6 sentences per story.
    for ($count = 0; $count < 6; $count++) {
        # Notes on the following statements:
        # 1) scalar @array gives the number of elements in the array.
        # 2) rand returns a random number greater than 0 and
        #    less than scalar(@array).
        # 3) int removes the fractional part of a number.
        # 4) . joins two strings together.
        $sentence = $nouns[int(rand(scalar @nouns))]
                    . " "
                    . $verbs[int(rand(scalar @verbs))]
                    . " "
                    . $nouns[int(rand(scalar @nouns))]
                    . " "
                    . $prepositions[int(rand(scalar @prepositions))]
                    . '. ';
        $story .= $sentence;
    }
    # Print the story.
    print "\n", $story, "\n";
    # Get user input.
    print "\nType \"quit\" to quit, or press Enter to continue: ";
    $input = <STDIN>;
    # Exit loop at user's request
} until($input =~ /^s*q/i);
exit;
```