

Bio 595

Computers in Biomedical Research

Fall 2003

Slides for Wednesday, November 5

Next on the agenda

How to:

Search for motifs in DNA or proteins

Interact with the user through the keyboard

Write data to a file

Use loops

Use basic regular expressions

Perform conditional tests

Operate on strings and arrays: how to examine sequence data in detail

Flow control in Perl

So far we've only used linear execution of statements, from top to bottom.

As in VB, there are loops, or conditional statements, available in Perl.

Conditional statements: if, if – else, unless

These evaluate to a true-false value (logical 1 or 0)

Equality of numbers is tested by double equals (==) operator. (Single equals are used for assignment of a value to a variable.)

Example of conditional statement

```
if ($x == 1) {  
    print "$x equals 1\n\n";  
}
```

If \$x has the current value of 1, will print "\$x equals 1"

Or:

```
if (x) {  
    Print "$x evaluates to true\n\n";  
}
```

this produces output of "\$x evaluates to true" if \$x is 1 (logical true) but produces no output if \$x is 0 (false)

The order in statements can put evaluation before the "if":

Print "\$x equals 1\n\n" if (x == 1); this also works.

if – else statement

This does one thing if true, or something else if false:

```
if ($x == 1) {  
    print "$x equals 1\n\n";  
}  
if ($x == 0) {  
    print "$x equals 0\n\n";  
} else {  
    print "$x does not equal 0\n\n";  
}
```

If \$x equals 1 will print out "\$x does not equal 0"

unless

This is the opposite of “if” statement; if condition is true, no action is taken; if condition evaluates to false, the statements are executed.

Example (here again let’s assume \$x is equal to 1):

```
Unless ($x == 0) {  
    Print “$x does not equal 0\n\n”;  
}
```

will produce the output “\$x does not equal 0”.

Conditional tests; blocks of code

Conditional tests can also test for inequality (`!=`), greater than (`>`) or less than (`<`)

Test for string equality by using the “`eq`” operator.

Statements within curly braces are called a “block” of code.

Note braces must match and number of left, right brackets used must be equal in number.

Conditional test example

```
#!/usr/bin/perl -w
# if-elsif-else demonstration
$word = 'MNIDDKL';
# if-elsif-else conditionals
if ($word eq 'QSTVSGE') {
    print "QSTVSGE\n"
} elsif ($word eq 'MRQQDMISHDEL') {
    print "MRQQDMISHDEL\n";
} elsif ( $word eq 'MNIDDKL' ) {
    print "MNIDDKL – the magic word!\n"
} else {
    print "Is \"$word\" a peptide? Not sure.\n"
}
exit;
```

Note in last example

The `\` “ in the else block’s print statement allows a double-quote within a double-quoted string. Here the backslash tells Perl to treat the following “ as the sign itself, not as the marker for the end of the string.

`eq` operator: this operator checks for the equality of two strings.

Loops

These let you repeatedly execute a block of statements. Loops include “while”, “for”, and “foreach”

“while” loop

```
#!/usr/bin/perl -w
# Read protein sequence data from a file
# the Filename of the file containing the protein sequence data
$proteinfilename = 'NM_021964fragment.pep';
# First we need to open the file; if the attempt to open the file fails,
# print an error message and exit the program
unless (open(PROTEINFILE, $proteinfilename) ) {
    print "Could not open the file $proteinfilename!\n";
exit;
}
# Read the protein sequence data from the file in a 'while' loop,
# printing out each line as it is read.
while( $protein = <PROTEINFILE> ) {
    print "Here is the next line of the file:\n"}
    print $protein;
}
# now close the file.
close PROTEINFILE;
exit;
```

Note in the last example:

The variable \$protein is assigned each time through the loop.

The test each time here is whether the assignment succeeds in reading another line.

If there is another line to read in, the assignment occurs and the conditional statement is true, so the block with the two print statements is executed.

If conditional is false, program skips the print block and quits the “while” loop, then closes and quits.

“open” accesses OS

“open” asks for file from operating system (Windows)

Check for success of “open” command (example in “unless”)

“open” command gives true result if file is successfully opened

Formatting statements

Examples of three equivalent formats:

Perl cares only about order of syntactical elements, not how they’re laid out in lines.

Formats 1 and 2 make clear the extent of blocks within statements; line up the curly brackets. Format 3 is no format at all, and is harder to read.

Three equivalent formats

```
While ($x) {  
  If ($y) {  
    Print "conditions met!\n";  
  }  
}
```

```
While ($x)  
{  
  If ($y)  
  {  
    Print "conditions met!\n";  
  }  
}
```

```
While ($x) {If ($y) {Print "conditions met!\n";}}
```

Finding motifs

This is a common procedure in bioinformatics, to find the presence of short segments of DNA or protein, for example, regulatory elements, or peptide segments conserved across many species.

(see PROCITE website for protein motifs at

<http://www.expasy.ch/prosite/>

String-searching capability for finding motifs. The next example reads protein data from file, puts all sequence data into 1 string for easy searching, and then looks for motifs entered via the keyboard.

Explanation of some program details

```
@protein = <PROTEINFILE>;
```

Here a filehandle and the angle bracket input operator are used to read in data from an opened file into an array.

```
$proteinfilename = <STDIN>;
```

The same syntax is used to get input from the user via the keyboard. Here, a special file handle called STDIN (standard input) is used; in the first use of this statement, it gets the filename from the user. Here the variable used to save the input is a scalar, so only one line will be read in (that's fine in this case).

Chomp

One more step is needed before the filename is used:

When the user types in a filename, a newline is sent when the enter (return) key is struck; that newline is entered as part of the variable. It must be removed or the name won't work.

“chomp” removes newlines (actually CRs and LFs) from the end of a string.

Do-until

Here the loop is executed once before the conditional text is performed.

If the user input is a null string, the loop is terminated.

“join”

```
$protein = join( ' ', @protein);
```

Protein sequence data is often broken into short segments for convenience in printout or display. However, CR LFs will cause problems in character searches.

Here, a null string is placed between joined elements.

Join is like a concatenation operator.

Join collapses an array of data into a single scalar string.

Program details, continued

```
$protein =~ s/ \ / g;
```

Regular expressions are ways of matching one or more strings using wildcard-like operators. Regular expressions can be a single word or more complex descriptions. In this case this would include any spaces or tabs.

Here we substitute any of a set of whitespace characters (\s with nothing); operation is done globally.

The “metasymbol” \s matches space, tab, newline, carriage return, formfeed.