

Bio 595

Computers in Biomedical Research Fall 2003 Slides for Monday, November 10

A few DOS commands

CD	(or CHDIR) change directory
CLS	clear screen
DEL	delete specified file
EXIT	quit (return to Windows environment)
MKDIR	make a new directory
PATH	displays or sets a search path
REN	(or RENAME) rename a directory or file
TYPE	display contents of a text file

Regular expressions

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

The symbol is equivalent to `[\t\n\r]` (character class, enclosed in square brackets). Space is typed preceding first `\`. Here, `\t` is for tab, `\n` for newline, `\f` for formfeed, and `\r` for carriage return.

Similar regular expression already used: the `s///` command, as in `s/C/G/g`. Here `C` is a regular expression.

Also, in `if ($motif =~ /^ \s* $ /) {` testing for whitespace. Here the regular expression is `/^ \s* $ /` this says to match a string that, from the beginning (the `^`) is zero or more (indicated by the `*`) whitespace characters (indicated by the `\s` until the end of the string is reached (indicated by the `$`).

Regular expressions

Search for the motif: `if ($protein =~ /$motif/) {`
Here the binding operator `=~` searches for the regular expression stored as variable `$motif` in the protein `$protein`. Here we're interpolating the value of a variable into a string.

In `ex10`, try any regular expression:

<code>A{D}S)V</code>	search for an <code>A</code> , followed by a <code>D</code> or an <code>S</code> , followed by a <code>V</code>
<code>KND*E(2,)</code>	search for <code>K</code> , <code>N</code> , zero or more <code>D</code> s, 2 or more <code>E</code> s (here <code>(2,)</code> means two or more)
<code>EE.*EE</code>	search for two <code>E</code> s, followed by anything, then two <code>E</code> s Here period stands for any character except a newline and the <code>*</code> says zero or more characters

New Perl routines

Counting nucleotides:

Questions asked about DNA: coding or noncoding?
Contain a regulatory element? How many of each of the four nucleotides are present? In some species coding regions have a specific nucleotide bias.

This will use new features of Perl:

Exploding a string

Looking at specific locations in strings

Iterating over an array

Iterating over the length of a string (using `substr` function)

Exploding strings into arrays

Separate out each letter, so each becomes its own scalar value in the array. This is the inverse of "join", which takes an array of strings and makes a single scalar value from them.

Get DNA from a file, make it a single string of sequence data; join the data in multiple lines, clean up the whitespace, convert it back to a (clean" array (of single nucleotides): see `ex11`.

Details about ex11

`@DNA = split(' ', $DNA);`
this explodes the string `$DNA` into an array of single characters `@DNA`. (compare to “join”)
Calling ‘split’ with an empty string as the first argument causes the string to explode into individual characters; the first argument with this command can be any regular expression.

Initialization: here scalar variables are used to store numbers (`$count_of_A = 0`; etc.)

Perl doesn't require declaration of variable type, like integer, string

Note ex12: Perl's automatic sensing of strings, numbers in calculations

ex11 details, continued

The ‘foreach’ loop here works on the elements of an array

Each time through the loop, the scalar variable `$base` is set to the next element of the array

The loop checks for each base and increments the count for each base found.

Ways to increment a counter:

```
++$count;
$count++;
$count = $count + 1;
$count += 1;
```

Alternate coding of the loop

```
if (/A/) {
    ++$count_of_A;
} elsif (/C/) {
    ++$count_of_C;
} elsif (/G/) {
    ++$count_of_G;
} elsif (/T/) {
    ++$count_of_T;
} else {
    print "!!!! Error - I don't recognize this base: ";
    print $base;
    print "\n";
    ++$errors;
}
}
```

ex11 details, continued

This version of the loop, `foreach(@DNA) {` doesn't have a scalar value.

in this loop, if no scalar variable is specified to hold scalars being read from the array, Perl saves them as a special variable `$_`.

(Other Perl functions use this variable.)

Here Perl assumes `$_ =~ /A/`; and so the statement “print” prints out the value of this variable `$_`

Examining characters in a string

Operating on strings:

It isn't necessary to explode a string to look at each character.

Can be expensive in memory if the data file is large (e.g. genomic)

Note ex13: a second program to examine bases in a string of DNA

Unless (`-e $dna_filename`) }

This is an example of a file test operator.

Files can be checked for several attributes – size, permission, file type.

‘for’ loop

The ‘for’ loop here is equivalent to the earlier ‘while’ loop.
`for ($position = 0; $position < length $DNA ; ++$position) {`

(block of code)

is equivalent to this ‘while’ loop:

```
$position = 0;
while( $position < length $DNA ) {
    (block of code)
    ++$position;
}
```

Counting bases

Notice counting of elements in a string;
String has a length (measured by length \$DNA)
First element is number 0, last element is 1 less than the length

The 'substr' function:

```
$base = substr($DNA, $position, 1);  
just like the Mid$ function we used in VB.
```

Writing to a file

Example ex14 also counts nucleotides and writes results to a file

```
$outputfile = "countbase";  
unless ( open(COUNTBASE, ">$outputfile") ) {  
    print "Cannot open file \"$outputfile\" to write to!\n\n";  
    exit;  
}  
print COUNTBASE "A=$a C=$c G=$g T=$t errors=$e\n";  
close(COUNTBASE);
```

Counting bases – another way

This version of counting bases is more concise than previous versions

Here the while loop is: while(\$dna =~ /a/ig)(\$a++)

Conditional test in parentheses: \$dna =~ /a/ig

here the 'i' says case-insensitive match (A or a), globally the (\$a++) increments the 'a' counter.

Faster, shorter still: use the 'tr' transliteration function:

```
$a = ($dna =~ tr/Aa//);
```

here, the 'tr' function returns the count of specified characters; since set of characters is empty, it doesn't actually change any characters

Because 'tr' doesn't accept character classes, it won't count errors not explicitly stated. But you could do this:

```
$basecount = ($dna =~ tr/ACGTacgt//);
```

```
$nonbase = (length $dna) - $basecount
```

Subroutines

A subroutine is a named block of code; you call the subroutine and pass values (parameters) to this block; the subroutine then reports back its results

Subroutines allow programs to be written in modular form. With defined inputs and outputs, these subroutines, once tested and validated, don't need to be "reinvented" every time the action of the subroutine is needed in a program. New programs often are created as a collection of existing subroutines, called as needed in a short "main" program.

Often, subroutines are drawn from existing libraries written by others and available from the web.

Writing a subroutine

Pass data as arguments, collect the return value(s) of the subroutine.

Give the subroutine a name (a descriptive one is best)

The name is followed by argument(s) in parentheses.

Example: suppose you write a subroutine to add the sequence ACGT to a DNA sequence called \$dna, and the resulting string will be saved as \$longer_dna. The call looks like this:

```
longer_dna = addACGT($dna);
```

The actual subroutine looks like this:

```
Sub addACGT {  
    my($dna) = @_;  
    $dna .= 'ACGT';  
    Return $dna;  
}
```

Subroutines

```
#!/usr/bin/perl -w  
# ex14:program with subroutine to append ACGT to DNA  
# The original DNA  
$dna = 'CGACGTCTTCTCAGGCGA';  
# The call to the subroutine "addACGT".  
# The argument being passed in is $dna; the result  
# is saved in $longer_dna  
$longer_dna = addACGT($dna);  
print "I added ACGT to $dna and got  
$longer_dna\n\n";  
exit;  
# Here is the definition for subroutine "addACGT"  
sub addACGT {  
    my($dna) = @_;  
    $dna .= 'ACGT';  
    return $dna;  
}
```