B211 Internet Computing

Perl String Processing

Learning Objective

- To learn to use Perl to do string processing.

String Processing in Perl

- One of the most powerful features of Perl is it's ability to processing text strings.
- It was originally created to do do string extraction and processing of text documents.
  - The word "Perl" is an acronym for "Practical Extraction and Reporting Language".
- The power of the string processing is tied to it's use of *regular expressions* (REs).

More Regular Expressions

- It is the special characters appearing in REs that gives the string processing its power (and also confusion).
- Some special characters used in REs:

```
.       # Any single character except a newline
^       # The beginning of the line or string
$       # The end of the line or string
*       # Zero or more of the last character
+       # One or more of the last character
?       # Zero or one of the last character
```

More Regular Expressions

- Some example simple matches:

```
t.e      # t followed by anything followed by e
         # This will match         the
         #                         tre
         #                         tle
         # but not        te
         #                tale
^f       # f at the beginning of a line
^ftp     # ftp at the beginning of a line
e$       # e at the end of a line
tle$     # tle at the end of a line
und*     # un followed by zero or more d characters
         # This will match         un
         #                         und
         #                         undd
         #                         unddd (etc)
.*       # Any string without a newline. This is because
         # the . matches anything except a newline and
         # the * means zero or more of these.
^$       # A line with nothing in it.
```


Special Characters

- Some general special characters used in strings:

```
\n       # A newline
\t       # A tab
\s       # Any whitespace character: space, tab, newline, etc
```

- Example usage:

```
print "Hello!\n" ;        # print the word "Hello!" followed by a new line
```

```
if ($sentence =~ /\t/) {          # if there is a tab in $sentence
        print ”There a tab in the sentence!\n" ;
}
```


Substitutions

- As well as identifying regular expressions Perl can make substitutions based on those matches.

- Eg. to replace occurences of “london” with “London” in $sentence:

```
$sentence =~ s/london/London/ ;    # replace first occurrence
$sentence =~ s/london/London/g ;   # replace all occurrences
                                   # - global substitution
$sentence =~ s/london/London/i ;   # ignore case of "london”
                                   # Means will replace "lOnDon",
                                   # "LONDON", "lONdon", etc.
```


Translation

- The `tr` function does a character-by-character translation.

- Eg. to replace ‘a’ with ‘e’, ‘b’ with ‘f’, and ‘c’ with ‘g’ in $sentence:

```
$sentence =~ tr/abc/efg/ ;
```

- Eg. to replace all lower case with upper case in $sentence:

```
$sentence =~ tr/a-z/A-Z/ ;
```

Splitting a string

- A very useful function for doing string processing in Perl is the `split` function.
- Eg.

```
$info = "January:February:March";
@months = split /:/, $info ;
```

@months now has the array (“January”,”February”,”March”)

```
$info = "January February March";
foreach $month (split /\s/, $info) {
    ...
}
```


The $_ special variable

- One of the potential confusion aspects of Perl syntax is that in many situations in Perl, functions defaults to using the $_ variable if no other variable is specified.
- Eg.

```
split(/&/) ;              # is the same as split(/&/,$_);
print ;                   # is the same as print $_ ;
foreach (@food) ...       # is the same as foreach $_ (@food) ...
if (/blah/) ...           # is the same as if ($_ =~ /blah/) ...
```


Printing Strings

- As you saw in the last lecture, you can use print to print any strings or variables. Eg.

```
print "Hello!" ;
print "The value of the variable is $a." ;
```

- An alternate way is to use a label when you want to print multiple lines:

```
print <<END_OF_PRINT ;

  Humpty Dumpty sat on the wall.
  Humpty Dumpty had a great fall.
  Of all the King’s horses and all the King’s men
  Could not put Humpty Dumpty back together again.

END_OF_PRINT
```


Printing Strings

- Another example: printing a HTML document:

```
print <<END_HTML ;

  <html>
    <head>
      <title>A Generated Page<title>
    </head>
    <body>
      <p>I have nothing to say.</p>
    </body>
  <html>

END_HTML
```

- The label you use (eg. `END_OF_PRINT` and `END_HTML` above) must not have spaces before it
  - Therefore, you cannot indent the line with the label.