Lecture #18 – Perl Part II

- Global special arrays and hashes

  @ARGV = command line args
  %ENV = environment variables

  Example:

```
$count = 1;

foreach (@ARGV)
{
	print "Arg $count = $_\n";
	$count++;
}
```

  Example:

```
foreach $key (sort keys %ENV)
{
	print "$key = $ENV{$key}\n";
}
```

- Global special file handles

  ARGV = file handle to iterate over command line arguments
  STDERR
  STDIN
  STDOUT

  Example:

```
$count = 1;

while ($line = <ARGV>)
{
	print "Arg $count = $line\n";
	$count++;
}
```

- String operator

  Concatenation operator "." is used to add strings together:

```
print 'abc' . 'def';          #  prints abcdef
print $a . $b;               # prints value of a followed by value of b
```

  Binary "x" is repetition operator:

```
print '-' x 80;              # prints row of 80 dashes
@ones = (1) x 80;           # list of 80 1's
```

- Perl regular expressions

  Perl recognizes most of the regexp syntax that we have discussed earlier, and includes some additional syntax of its own:

\b	"word" boundary
\B	not a "word" boundary
\w	any single character classified as a "word" character (alphanumeric and "_")
\W	any single non-"word" character
\s	any whitespace character (space, tab, or newline)
\S	any non-whitespace character
\d	any digit character (i.e. [0-9])
\D	any non-digit character
\A	match only at beginning of string
\Z	match at end of string, on or before newline at end
\z	match at end of string

  Examples:

/\d\d:\d\d:\d\d/	time format hh:mm:ss
/[\d\s]/	any single digit or whitespace char
/\w\W\w/	word char, then non-word char, then word char
/[a-z]+\s+\d*/	lowercase word, some whitespace, possibly some digits
/\d{4}/	4 digit number

- Pattern matching operators

m/pattern/[flag] match

where flag is:

g	match globally (i.e. find all occurrences)
i	case insensitive
m	treat string as multiple lines
s	treat string as single line
x	use extended regular expressions

Example:

```
if ($shire =~ m/Baggins/) {          #  same as if ($shire =~ /Baggins/)
        print "$shire matches Baggins";
}

if ( m/Baggins/) {
        print "$_ matches Baggins";
}
```

Example:

```
$string = "password=abc verbose=9 score=1";
%h = $string =~ /(\w+)=(\w+)/g;

foreach $value (sort keys %h)
{
        print "$value = $h{$value}\n";
}
```

s/pattern/replacement/[flag] substitution

Example:

```
$t = "This is a test";
$t =~ s/a/an/;
$t =~ s/test/exam/g;
print "$t\n";                              #   This is an exam

$count = $t =~ s/exam/interrogation/;
print "$t\n";                              #  This is an interrogation
print "$count\n";                          #  1
```

Example:

```
$t = "16";

$t =~ s/([0-9]+)/nn $1 nn/;                 # replace num with “nn <num> nn”
$t =~ s/([0-9]+)/sprintf("%#x", $1)/ge;     # translate num to hex

print “$t\n”;                               # nn 0x10 nn
```

tr/pattern1/pattern2/[flag] transpose

where flag is:

c	complement pattern1
d	delete found be unreplaced char
s	squash duplicate replaced char

Example:

```
$t = “this is a test”;

$t =~ tr/a-z/A-Z/;
print “$t\n”;                               # THIS IS A TEST
```

- Advanced String Matching

Example:

```
# Create an Array using the directory listing
@dir_array  = `ls -l`;

print "Here is the directory again:\n";
print @dir_array, "\n";


print "Here are the perl programs:\n";

$pattern = '\s+(\w+\.+pl)\s';  #Define a pattern using "regular expressions"

# Meaning "\s+" - at least one or more spaces or tabs
#       "\w+" - at least one or more alpha-numeric characters
#       "\.+" - a period or dot
#       "pl" - the proper "pl" extender
#       "\s" - a trailing space
```

```
$j=0;

for ($i=0; $i <= $#dir_array; $i++)          # Loop through all lines
{
        if ($dir_array[$i] =~ $pattern)
        {
                print $1, "\n";
                $perlprogs[$j] = $1;
                $j++;
        }
}

print "The program names are also stored in an array: ";
$, = ", ";                                   #  Make OFS a comma
print @perlprogs;
print "\n";
```

Example (Regular Expression matching):

```
sub print_array                              # Print the full contents of the Array
{
        for ($i=0; $i<=$#strings;$i++)
        {
                print $strings[$i], "\n";
        }
        print "\n\n";
}

sub grep_pattern                             # Print strings which contain the pattern
{
        print "Searching for: $pattern\n";

        foreach (@strings)
        {
                print "$_\n" if /$pattern/;
        }
        print "\n\n";
}

### Setting up the Array of strings

@strings = ("Two, 4, 6, Eight", "Perl is cryptic", "Perl is great");

@strings[3..6] = ("1, Three", "Five, 7", "Write in Perl", "Programmer's heaven");
print_array;
```

```
## Find the word "Perl"
$pattern = 'Perl';
grep_pattern;

## Find "Perl" at the beginning of a line
$pattern = '^Perl';
grep_pattern;

## Find sentences that contain an "i"
$pattern = 'i';
grep_pattern;

## Find words starting in "i", i.e. a space preceeds the letter
$pattern = '\si';
grep_pattern;

## Find strings containing a digit
$pattern = '\d';
grep_pattern;

## Search for a digit followed by some stuff
$pattern = '\d+.+';
grep_pattern;

## Find strings with a digit at the end of a line
$pattern = '\d+$';
grep_pattern;

## Search for a digit, possible stuff in between, and another digit
$pattern = '\d.*\d';
grep_pattern;

## Find four-letter words, i.e. four characters offset by word boundaries
$pattern = '\b\w{4}\b';
grep_pattern;

## Sentences with three words, three word fields separated by white space
$pattern = '\w+\s+\w+\s+\w+';
grep_pattern;

## Find sentences with two "e" letters, and possible stuff between
$pattern = 'e.*e';
grep_pattern;
```

Marking Regular Expression Sub-strings and Using Substitution

```
## Substitute "Pascal" for "Perl" words at the beginning of a line
print "Substituting first Perl words.\n";
foreach(@strings)
{
        s/^Perl/Pascal/g;
}
print_array;

## Find five-letter words and replace with "Amazing"
$pattern = '\b\w{5}\b';
print "Searching for: $pattern\n";
foreach(@strings)
{
        s/$pattern/Amazing/;
}
print_array;

## Replace any "Perl" words at the end of a line with "Cobol"
print "Substituting Final Perl \n";
foreach(@strings)
{
        s/Perl$/Cobol/;
}
print_array;

## Delete any apostrophes followed by an "s"
print "Substituting null strings\n";
foreach(@strings)
{
        s/\'s//;  # Replace with null string
}
print_array;

## Search for two digits in same line, and switch their positions
print "Tagging Parts and Switching Places\n";
foreach(@strings)
{
        $pattern = '(\d)(.*)(\d)';

        if (/$pattern/)
        {
                print "Grabbed pattern: $pattern   \$1 = $1   \$2 = $2   \$3 = $3\n";
                s/$pattern/$3$2$1/;
        }
}
```

- File and I/O

  Example (reading user input from the keyboard):

```
print "Enter a file name:";
chomp($fname = <STDIN>);          # chomp removes the newline from the input
```

  Example (reading a file - Slurping):

```
open (FPTR,$fname) || die "Can't Open File: $fname\n";

@filestuff = <FPTR>;              #Read the file into an array

print "The number of lines in this file is ",$#filestuff + 1,"\n";
print @filestuff;

close (FPTR);
```

  Note: This method is BAD for big files since we need RAM for entire file.

  Example (primitive file copy – convert to uppercase):

```
open (FPTR,$fname) || die "Can't Open File: $fname\n";
open (OUTFILE, ">upcase.txt") || die "Can't open output file.\n";

while (<FPTR>)
{
        tr/a-z/A-Z/;

        print OUTFILE, $_;
}

close(FPTR);
close(OUTFILE);
```

- Built-in Functions

  There are many built-in functions many resembling functions provided in C or one of the other scripting languages. See man pages, or reference material for a list