Week 04 Tutorial Questions

1. What is the difference in value/type of the following Perl expressions:

   a. `"a"` vs `'a'`
   b. `"A"` vs `A`
   c. `"abc"` vs `'abc'`
   d. `"it\'s"` vs `'it\'s'`
   e. `42` vs `"42"`
   f. `"3"` vs `"3.0"`
   g. `"$2.50"` vs `'$2.50'`

2. Write a Perl program, `nargs.pl` which prints how many arguments it has been given. For example:

```
$ ./nargs.pl the quick brown fox
4
```

3. Write a Perl program, `devowel.pl` which filters any vowels from its input. For example:

```
$ ./devowel.plThe quick brown fox
jumped over the lazy dog.
Ctrl-D
Th qck brwn fx
jmpd vr th lzy dg.
```

4. Write a simple version of the `head` command in Perl, that accepts an optional command line argument in the form *–n*, where *n* is a number, and displays the first *n* lines from its standard input. If the *–n* option is not used, then the program simply displays the first ten lines from its standard input.

   Examples of use:

```
$ perl head.pl <file2        # display first ten lines of file2
...
$ perl head.pl -10 <file2    # same as previous command
...
$ perl head.pl -5 <file2     # display first five lines of file2
...
```

5. Modify the `head` program from the previous question so that, as well as handling an optional *–n* argument to specify how many lines, it also handles multiple files on the command line and displays the first *n* lines from each file, separating them by a line of the form `==> FileName <===`.

   Examples of use:

```
$ perl head.pl file1 file2 file3  # display first ten lines of each file
...
$ perl head.pl -3 file1 file2   # display first three lines of each file
...
```

6. Write a simple version of the `grep` command, that takes a regular expression as its first command line argument and then prints all lines in the standard input (or named files) that contain this pattern.

   Examples of use:

```
$ perl mygrep.pl 'a.*c' file1 file2 file3     # all lines containing a...c
...
$ perl mygrep.pl '[0-9]+' file1 file2 file3   # all lines containing numbers
...
$ perl mygrep.pl '^The' <file1          # all lines starting with "The"
...
```

7. Modify the `grep` command from the previous question so that accepts a –v command line option to reverse the sense of the test (i.e. display only lines that do *not* match the pattern). It should continue with its original behaviour if no –v is specified.

Revision questions

The following questions are primarily intended for revision, either this week or later in session. Your tutor may still choose to cover some of these questions, time permitting.

1. The following programs are all Perl versions of the `cat` program. Each of them either reads from standard input (if there are no command line arguments) or treats each command line argument as a file name, opens the file, and reads it. The final one shows just how concise Perl code can be. You may find the ideas in these programs useful in helping you solve the problems below.

```
#!/usr/bin/perl -w

# First Perl version of cat
# Verbose, but shows exactly what's happening

# if no args, read from stdin
if (@ARGV == 0) {
    while ($line = <STDIN>) {
        # note: line still has \n
        print $line;
    }
} else {
    foreach $file (@ARGV) {
        open my $input, '<', $file or die "Can not open $file: $!\n";
        while ($line = <$input>) {
            print $line;
        }
        close $input;
    }
}
```

```
#!/usr/bin/perl -w

# Second Perl version of cat
# More concise, by using special variable $_

if (@ARGV == 0) {
    while (<STDIN>) {
        print;
    }
} else {
    foreach $file (@ARGV) {
        open my $input, '<', $file or die "Can not open $file: $!\n";
        while (<$input>) {
            print;
        }
        close $input;
    }
}
```

```
#!/usr/bin/perl -w

# Third Perl version of cat
# places input into an array

if (@ARGV == 0) {
    @lines = <STDIN>;
    print @lines;
} else {
    foreach $file (@ARGV) {
        open my $input, '<', $file or die "Can not open $file: $!\n";
        @lines = <$input>;
        print @lines;
        close $input;
    }
}
```

```
#!/usr/bin/perl -w

# Fourth Perl version of cat
# More concise, but makes filtering difficult

if (@ARGV == 0) {
    print <STDIN>;
} else {
    foreach $file (@ARGV) {
        open my $input, '<', $file or die "Can not open $file: $!\n";
        print <$input>;
        close $input;
    }
}
```

```
# Other versions of cat
# Make use of the fact that <> has a special meaning
# – if no command line arguments, read standard input
# – otherwise, open each argument as a file and read it
# Very concise, but ...
# – you'll need to put up with Perl's error messages
# – you treat all files as a single stream ... which means
#   – you can't distinguish which file each line comes from
#   – there is no scope for doing things at file boundaries
while (<>) { print; }
#or
@lines = <>;
print @lines;
#or
print <>;
```

   Write a new version of `cat` so that it accepts a –n command line argument and then prints a line number at the start of each line in a field of width 6, followed by two spaces, followed by the text of the line. The numbers should constantly increase over all of the input files (i.e. don't start renumbering at the start of each file). The program always reads from its standard input.

   Example of output:

```
$ perl cat -n myFile
     1  This is the first line of my file
     2  This is the second line of my file
     3  This is the third line of my file
        ...
  1000  This is the thousandth line of my file
```

2. Modify the `cat` program from the previous question so that it also accepts a –v command line option to display *all* characters in the file in printable form. In particular, end of lines should be shown by a `$` symbol (useful for finding trailing whitespace in lines) and all control characters (ascii code less than 32) should be shown as *^X* (where *X* is the printable character obtained by adding the code for 'A' to the control character code). So, for example, tabs (ascii code 9) should display as `^I`.
   Hint: the `chr` and `ord` functions might be useful. Try

```
$ perldoc -f ord
```

   for info about functions such as these.

   Example of output:

```
$ perl cat -v < myFile
This file contains a tabbed list:$
^I– point 1$
^I– point 2$
^I– point 3$
And this line has trailing spaces   $
which would otherwise be invisible.$
```

3. Write a version of the `tac` command in Perl, that accepts a list of filenames and displays the lines from each file in reverse order.