# Lecture # 5 – Regular Expressions (Chapter 3)

- Regular Expressions

  A regular expression (regexp) defines a set of one or more strings of characters.

  Used in many utilities - ed, vi, emacs, grep, awk, sed

  Regular expressions are often enclosed in a pair of delimiters - (Usually forward slash)

- Simple Strings can be regular expressions

  Examples: /day/ day, today, Monday
  /or not/ or not, poor nothing

  $ grep “day” testfile
  $ grep “or not” testfile

- Regular Expression Metacharacters

  1. Period (.) matches any single character

     /.alk/ will **talk**, may **balk**
     /.ing/ **sing**ing, **ping**, before **ing**lenook

     $ grep ‘.alk’ myfile

  2. Square Brackets [] = character class, matches any one character from class

     /[bB]ill/ bill, Bill
     /t[aeiou].k/ **talk**ative, s**tink**, **teak**, **tank**er
     /number [6-9]/ number<space><6 or 7 or 8 or 9>

     Caret (^) within brackets means any character not in the set

     /[^a-zA-Z]/ any character that is not a letter (i.e. 1, 7, @, !)

     $ grep ‘[^A-Z]’ myfile
     $ grep ‘[aAbB]’ myfile

3. Asterisk (*) = zero or more occurrences of the preceding character

| | |
|---|---|
| /ab*c/ | single 'a' followed by zero or more 'b' followed by single 'c'<br>**ac**, **abc**, **abbc**, debbca**abbbc** |
| /ab.*c/ | 'ab' followed by zero or more characters then a 'c'<br>abc, abxc, ab45c, xa**b 756.345 x c**at |
| /t.*ing/ | ting, thing, I thought of going |
| /[a-zA-Z]*/ | any string without numbers or punctuation |

4. Caret (^) = beginning of line anchor

| | |
|---|---|
| /^T/ | Line beginning with T |
| /^[0-9]/ | Line beginning with a digit |

5. Dollar ($) = end of line anchor

| | |
|---|---|
| /:$/ | Line ending with ':' |
| /^Test$/ | Line with Test by itself |

6. Backslash (\) = quoting special characters

| | |
|---|---|
| /end\./ | The **end.** |
| /\[5\]/ | it was five **[5]** |
| /and\/or/ | and/or (Note: most allow alternate delimiters, i.e. –and/or-) |

- In general, regular expressions match the longest possible string from left of line

| | |
|---|---|
| /(.*)/ | matches longest string between ( and ) |
| /[^)]*)/ | matches shortest string with ( and ) |

- Bracketing Expressions

1. The Replacement String

   Ampersand (&) takes the value of a matched string

   :s/[0-9][0-9]*/NN&NN/ matches a number and puts NN before and after

   Why have [0-9] followed by [0-9]* ?

2. Quoted Digit

Quoted Digit (\n) takes on the value of the string that the bracketed expression beginning with the nth matched

Last name, first name and we want to change to first name last name
:1,$s/\([^,]*\), \(.*\)/\2 \1/

1,$ means all lines s means substitute
Take everything up to a comma = \1, all else \2 and then switch them around

- Extended regular expressions

Plus (+) matches one or more characters
Question (?) matches zero or one characters
Pipe (|) means or

Examples:

| | |
|---|---|
| /ab+c/ | "a" followed by one or more "b" followed by a "c" |
| /ab?c/ | "a" optionally followed by a "b", then followed by a "c" |
| /(ab)+c/ | one or more occurrences of "ab" followed by a "c" |
| /ab\|ac/ | either "ab" or "ac" (i.e. /a[bc]/) |
| /^Exit\|^Quit/ | lines that start with "Exit" or "Quit" |

# Using Grep (Chapter 4)

- Origins of the grep utility

  Grep can be traced back to the "ex" editor (predecessor to "vi").

  The "ex" command:

  | | |
  |---|---|
  | :/pattern/p | print the first line matching the pattern |
  | :g/pattern/p | print all lines matching the pattern. |

  Since the pattern is often a regular expression, this could also be written:

  :g/RE/p

- Common options for grep

  | | |
  |---|---|
  | -c | displays a count of matching lines rather than displaying the actual lines |
  | -h | does not display filenames |
  | -i | ignores case of letters when making comparisons |
  | -l | list only the names of files with matching lines |
  | -n | precedes each line by its relative line number |
  | -s | work silently (i.e. display nothing but error messages) |
  | -v | invert the search (i.e. display only lines that do not match) |

- Exit status

  - pattern is found in at least one line
  - pattern not found
  - file not found

- Examples of grep with regexp (using data file from p. 86)

  Search for the string "NW" in the file "datafile":

  ```
  $ grep NW datafile
  ```

  Search for the string "NW" in all files starting with "d" in the current directory:

  ```
  $ grep NW d*
  ```

  Print all lines that begin with a "n":

  ```
  $ grep '^n' datafile
  ```

  Print all lines that end with a "4":

  ```
  $ grep '4$' datafile
  ```

Search for "TB Savage":

```
$ grep 'TB Savage' datafile
```

Search for 5 followed by a literal period followed by any character:

```
$ grep '5\..' datafile
```

Print all lines containing one non-digit:

```
$ grep '[^0-9]' datafile
```

Print all lines with at least 9 consecutive lowercase letters:

```
$ grep '[a-z]\{9\}' datafile
```

- Examples with options

Print all lines that begin with "south" including line numbers:

```
$ grep –n '^south' datafile
```

Print all lines with "pat" regardless of case:

```
$ grep –i "pat" datafile
```

Print all lines that do not contain 'Suan Chin':

```
$ grep –v 'Suan Chin' datafile
```

Print just the filenames of files containing lines with "SE":

```
$ grep –l 'SE' *
```

Print the number of lines containing "west":

```
$ grep –c 'west' datafile
```

- egrep (extended grep)

Similar to regular grep, except it uses extended regular expressions.

Includes the following meta characters:

| | |
|---|---|
| + | matches one or more of the preceding characters |
| ? | matches zero or one of the preceding characters |
| a \| b | matches either a or b |
| () | groups characters |

Examples:

Print lines that contain NW or EA:

```
$ egrep 'NW|EA' datafile
```

Print lines that contain one or more 3's:

```
$ egrep '3+' datafile
```

Print lines that have a '2' with or without a period and a digit:

```
$ egrep '2\.?[0-9]' datafile
```

Print lines with one or more occurrences of "no":

```
$ egrep '(no)+' datafile
```

- GNU grep

GNU grep (i.e. the one that comes with Linux) has similar functionality as UNIX grep, and some additional improvements.

GNU grep combines the functionality of grep and egrep (i.e. grep –E).

GNU grep supports POSIX character classes:

| | |
|---|---|
| [:alnum:] | alphanumeric char |
| [:alpha:] | alphabetic char |
| [:cntrl:] | control char |
| [:digit:] | numeric char |
| [:lower:] | lowercase char |
| [:upper:] | uppercase char |
| [:punct:] | punctuation char |
| [:space:] | whitespace |

Examples:

Print lines with a space, a period, a digit, and a space.

```
$ grep '[[:space:]]\.[[:digit:]][[:space:]]' datafile
```

GNU grep supports additional options:

| | |
|---|---|
| -A # | Prints # lines of trailing context after matching lines |
| -B # | Prints # lines of leading context before matching lines |
| -L | Print just the names of all files where the pattern does not match |
| -m # | Stop reading a file after # of matching lines |