CS 3423
Systems Programming

Regular Expression

and

sed (stream editor) utility

Outline

- Regular Expression (RE)
- sed introduction
- Selecting lines using sed
- The ’s’ commands
- sed scripts

Regular Expression

- A regular expression (regex or re for short) is a special text string for describing a search pattern.
- Some utilities/programs that use them:
  - **vi**, **ed**, **sed**, and **emacs**
  - **awk**, **Perl** and **Python**
  - **grep**, **egrep**, **fgrep**
  - **compilers**

Definition

- R is a regular expression if it is:
  1. **a** for some *a* in the alphabet ∑, standing for the language {a}
  2. ε, standing for the language {ε}
  3. Ø, standing for the empty language
  4. R_1+R_2 where R_1 and R_2 are regular expressions, and + signifies union (sometimes | is used)
  5. R_1R_2 where R_1 and R_2 are regular expressions and this signifies concatenation
  6. R* where R is a regular expression and signifies closure
  7. (R) where R is a regular expression, then a parenthesized R is also a regular expression

RE examples

- L(**001**) = {001}
- L(**0+10***) = { 0, 1, 10, 100, 1000, 10000, … }
- L(**0*10***) = {1, 01, 10, 010, 0010, …}   i.e. {w | w has exactly a single 1}
- L(∑∑)* = {w | w is a string of even length}
- L((**0(0+1**))*) = { ε, 00, 01, 0000, 0001, 0100, 0101, …}
- L((**0+ε)(1+ ε**)) = {ε, 0, 1, 01}
- L(1Ø) = Ø   ; concatenating the empty set to any set yields the empty set.
- Rε = R
- R+Ø = R

- Note that R+ε may or may not equal R (we are adding ε to the language)
- Note that RØ will only equal R if R itself is the empty set.

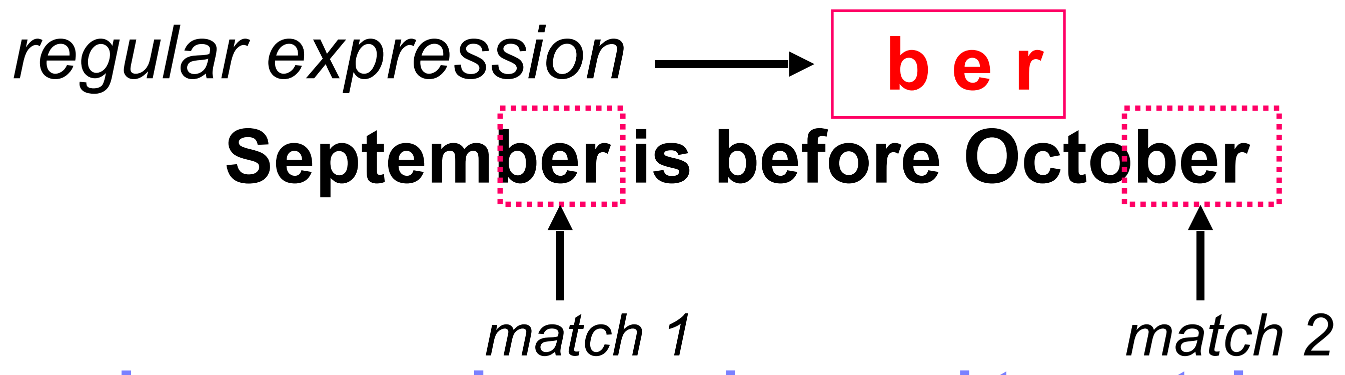
Regular Expression Quick Guide

```
^        Matches the beginning of a line
$        Matches the end of the line
.        Matches any character
*        Repeats a character 0 or more times
*?       Repeats a character 0 or more times (non-greedy)
+        Repeats a character 1 or more times
+?       Repeats a character 1 or more times (non-greedy)
[aeiou]  Matches a single character in the listed set
[^XYZ]   Matches a single character not in the listed set
[a-z0-9] The set of characters can include a range
(        Indicates where string extraction is to start
)        Indicates where string extraction is to end
```

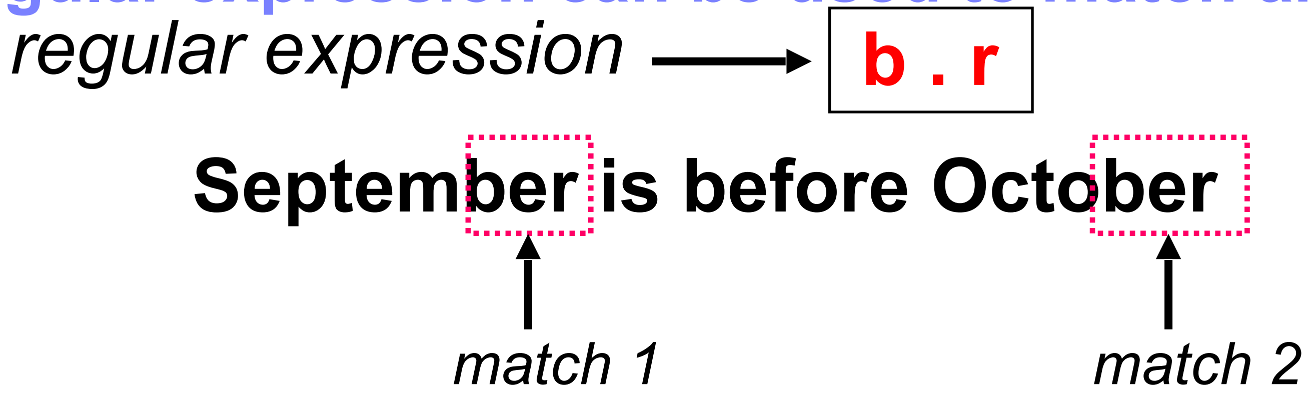
\b	Matches a word boundary, that is, the position between a word and a space. For example, er\b matches the er in "never" but not the er in verb.
\B	Matches a nonword boundary. ea*r\B matches the ear in never early.
\d	Matches a digit character. Equivalent to [0-9].
\D	Matches a nondigit character. Equivalent to [^0-9].
\n	Matches a newline character.
\r	Matches a carriage return character.
\s	Matches any white space including space, tab, form-feed, etc. Equivalent to [\f\n\r\t\v].
\S	Matches any nonwhite space character. Equivalent to [^ \f\n\r\t\v].
\t	Matches a tab character.
\v	Matches a vertical tab character.
\w	Matches any word character including underscore. Equivalent to [A-Za-z0-9_].
\W	Matches any nonword character. Equivalent to [^A-Za-z0-9_].

Matches

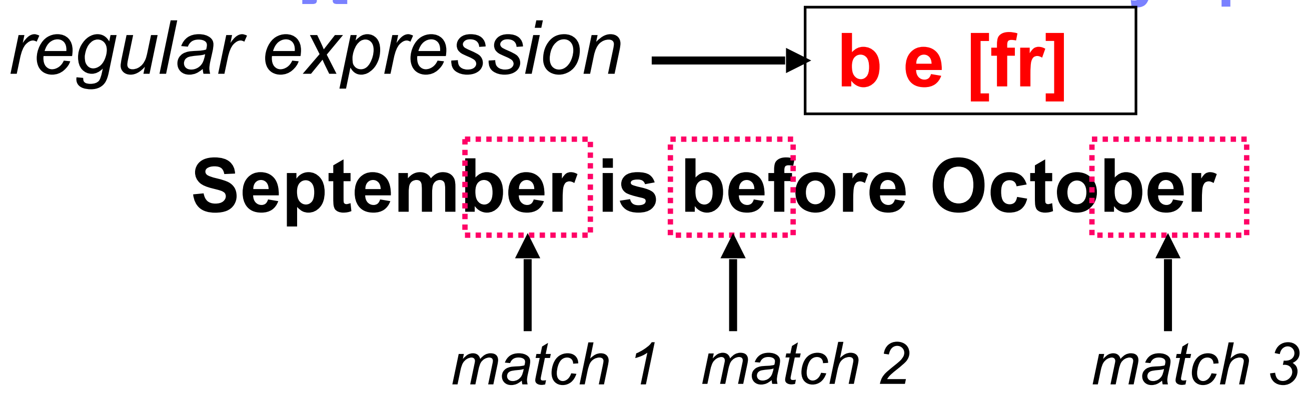
- A regular expression can match a string in more than one place.



- The `.` regular expression can be used to match any character.



- Character classes `[]` can be used to match any specific set of characters.



- Other examples of character classes:
  - [aeiou] will match any of the characters a, e, i, o, or u
  - [kK]ate will match kate or Kate

- Ranges can also be specified in character classes
  - [1-9] is the same as [123456789]
  - [abcde] is equivalent to [a-e]

- You can also combine multiple ranges
  - [abcde123456789] is equivalent to [a-e1-9]

- Note that the - character has a special meaning in a character class but only if it is used within a range
  - [-123] would match the characters -, 1, 2, or 3

^ , $, +, * and ?

- Anchors are used to match at the beginning or end of a line (or both).
  - ^ means beginning of the line
  - $ means end of the line

regular expression → **^Sep**

September is before October, September is after August.

match

regular expression → **b e [fr] $**

September is before October

match

regular expression → **b e r?**

September is before October

match *match* *match*

regular expression → **ya*y**

yaaaaaaaay! *match*

yy! *match*

regular expression → **ya+y**

yaaaaaaaay! *match*

yy! *not a match*

Repeating ranges, subexpressions

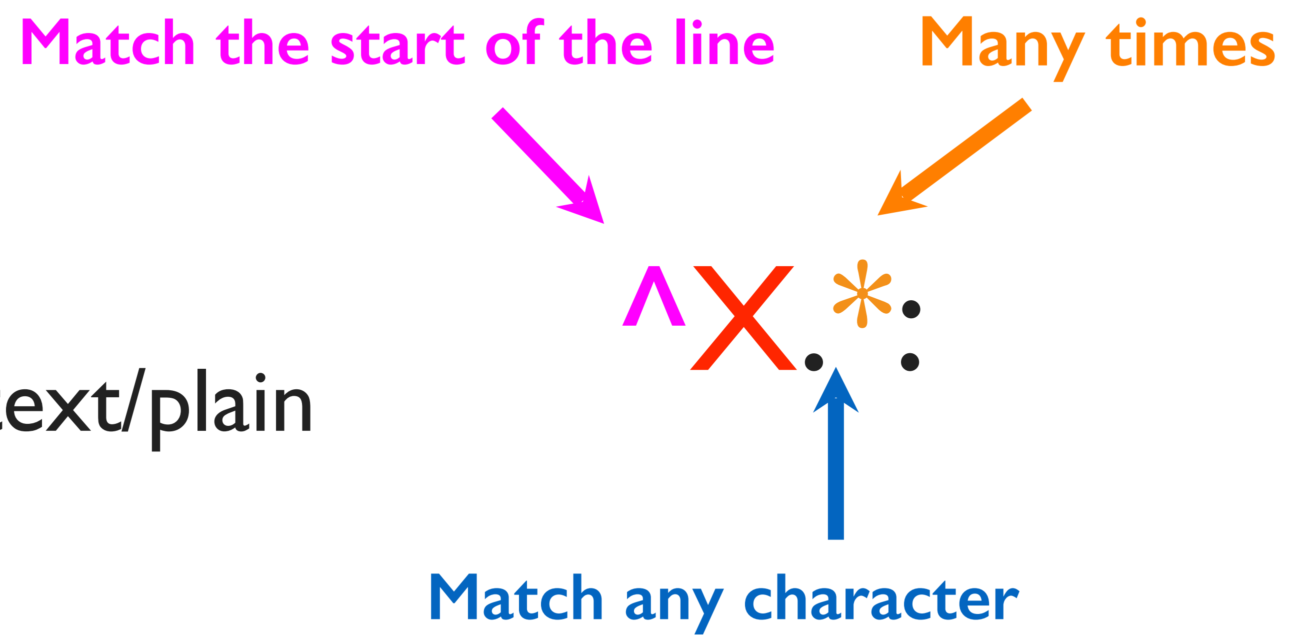
- Ranges can also be specified, `{n,m}` notation can specify a range of repetitions for the immediately preceding regex
  - `{n}` means exactly *n* occurrences
  - `{n,}` means at least *n* occurrences
  - `{n,m}` means at least *n* occurrences but no more than *m* occurrences
- Example:
  - `.{0,}` same as `.*`
  - `a{2,}` same as `aaa*`
- If you want to group part of an expression so that `*` applies to more than just the previous character, use `( )` notation
- Subexpresssions are treated like a single character
  - `a*` matches `0` or more occurrences of `a`
  - `abc*` matches `ab`, `abc`, `abcc`, `abccc`, …
  - `(abc)*` matches `abc`, `abcabc`, `abcabcabc`, …
  - `(abc){2,3}` matches `abcabc` or `abcabcabc`
- `What if you want to search` 'a*b*' ?
  - a*b*? this will match zero or more 'a's followed by zero or more 'b's, **not what we want!**
  - 'a*b*' will fix this - now the asterisks are treated as regular characters.

“|”

- alternation character: `|` - - matching one or another subexpression
  - `(T|Fl)ow` will match Tow or Flow
  - `^(From|Subject):` will match the From and Subject lines of a typical email message
    - It matches a beginning of line followed by either the characters From or Subject followed by a ‘:’

Wild-card characters

X-S123: CMU Sieve 2.3
X-DSPAM: Innocent
X-DSPAM: 0.8475
X-Content-Type-Message-Body: text/plain



Tuning the match:

X-S123: CMU Sieve 2.3
X-DSPAM: Innocent
X-DSPAM: 0.8475
X-Content-Type-Message-Body: text/plain

^X.*[0-9]+:

Some examples

- Variable names in C
  - `[a-zA-Z_][a-zA-Z_0-9]*`
- Dollar amount with optional cents
  - `\$[0-9]+(\.[0-9][0-9])?`
- Time of day
  - `(1[012]|[1-9]):[0-5][0-9] (am|pm)`
- HTML headers <h1> <H1> <h2> …
  - `<[hH][1-4]>`

- http://www.night-ray.com/regex.pdf

sed

Introduction

- sed: **s**tream **ed**itor
- Used for editing files automatically.
- Non-interactive editor: won’t modify the file, all the output is just printed out.

```
sed [options] 'command' file(s)

sed [options] -f scriptfile file(s)
```

Selecting a Line

```
$ cat a.txt
line 1
line 2
line 3
line 4
line 5
line 6
```

```
Select the second line:

$ sed -n ‘2p’ a.txt
line 2
$ sed '2!d' a.txt
line 2
```

```
$ sed 2p a.txt
line 1
line 2
line 2
line 3
line 4
line 5
line 6
```

-n: Suppress the default output (in which each line, after it is examined for editing, is written to standard output). Only lines explicitly selected for output are written.

Selecting Lines

```
$ cat a.txt
line 1
line 2
line 3
line 4
line 5
line 6
```

```
Select multiple lines:

$ sed -n '2,4p' a.txt
line 2
line 3
line 4
$ sed '2,4!d' a.txt
line 2
line 3
line 4
```

```
-e: multiple commands

$ sed -n -e '1,2p' -e '4p' a.txt
line 1
line 2
line 4
```

Deleting lines

```
$ cat a.txt
line 1
line 2
line 3
line 4
line 5
line 6
```

Deleting the third line:

```
$ sed '3d' a.txt
line 1
line 2
line 4
line 5
line 6
```

Deleting all the lines with ‘2’ in it:

```
$ sed '/2/'d a.txt
line 1
line 3
line 4
line 5
line 6
```

Deleting from the third line to the end:

```
$ sed '3,$d' a.txt
line 1
line 2
```

Deleting the last line:

```
$ sed '$d' a.txt
line 1
line 2
line 3
line 4
line 5
```

’s’ command

```
$ cat test.txt
#!/bin/bash
 function hello{
 echo "hello";
}
hello;
$ sed 's/hello/hi/g' test.txt
#!/bin/bash
 function hi{
 echo "hi";
}
hi;
$ cat test.txt
#!/bin/bash
 function hello{
 echo "hello";
}
hello;
```

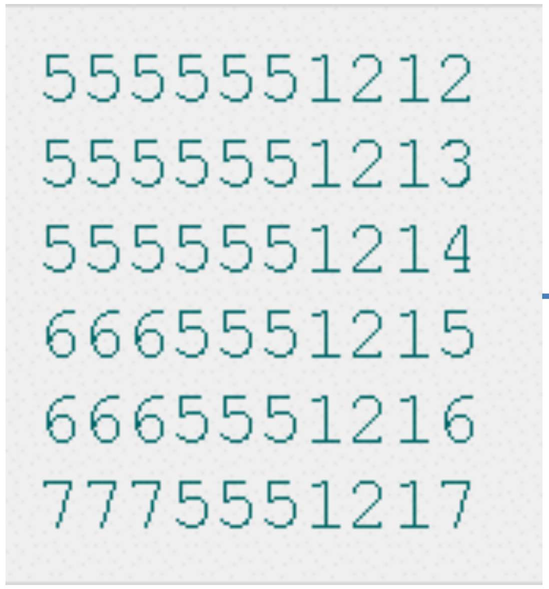
```
$ sed -n 's/hello/hi/p' test.txt
 function hi{
 echo "hi";
hi;
```

```
$ sed -n 's/hello/&hi/p' test.txt
 function hellohi{
 echo "hellohi";
hellohi;
```

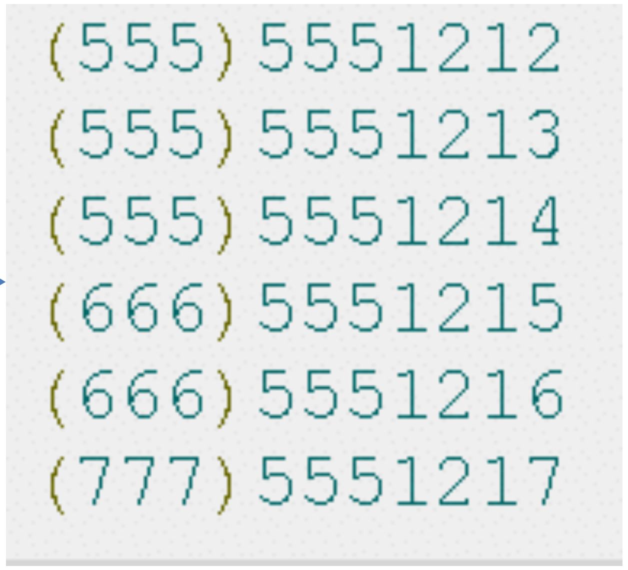
```
$ sed -n 's/\(he\)llo/\1lp/p' test.txt
 function help{
 echo "help";
help;
```

& referring

- $ cat phone



```
5555551212
5555551213
5555551214
6665551215
6665551216
7775551217
```



```
(555)5551212
(555)5551213
(555)5551214
(666)5551215
(666)5551216
(777)5551217
```

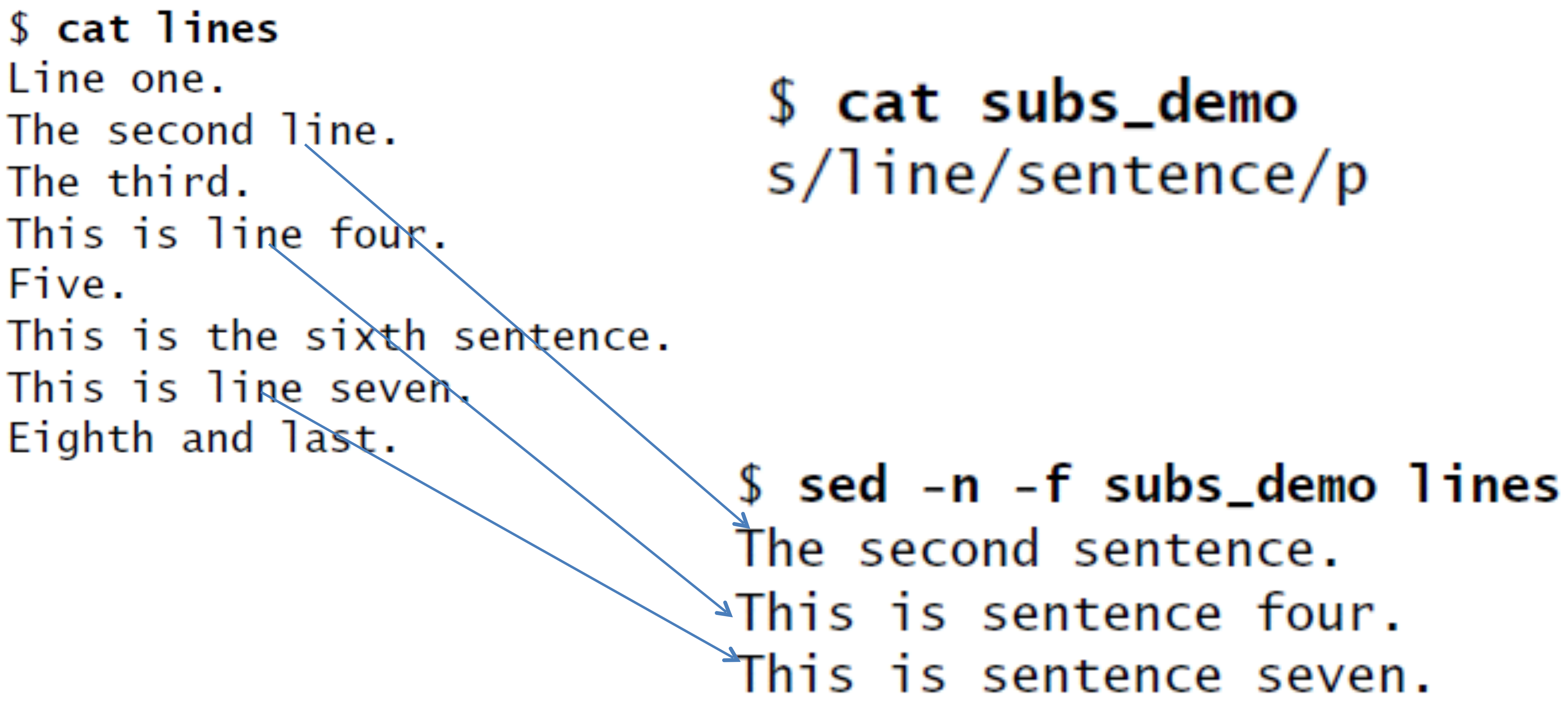
sed 's/^[0-9][0-9][0-9]/(&)/' phone

sed 's/^[0-9]\{3\}/(&)/' phone

Matches the regular expression 3 times

sed script

- # is followed by the comments. e.g., #!/bin/sed -f



```
$ cat lines
Line one.
The second line.
The third.
This is line four.
Five.
This is the sixth sentence.
This is line seven.
Eighth and last.
```

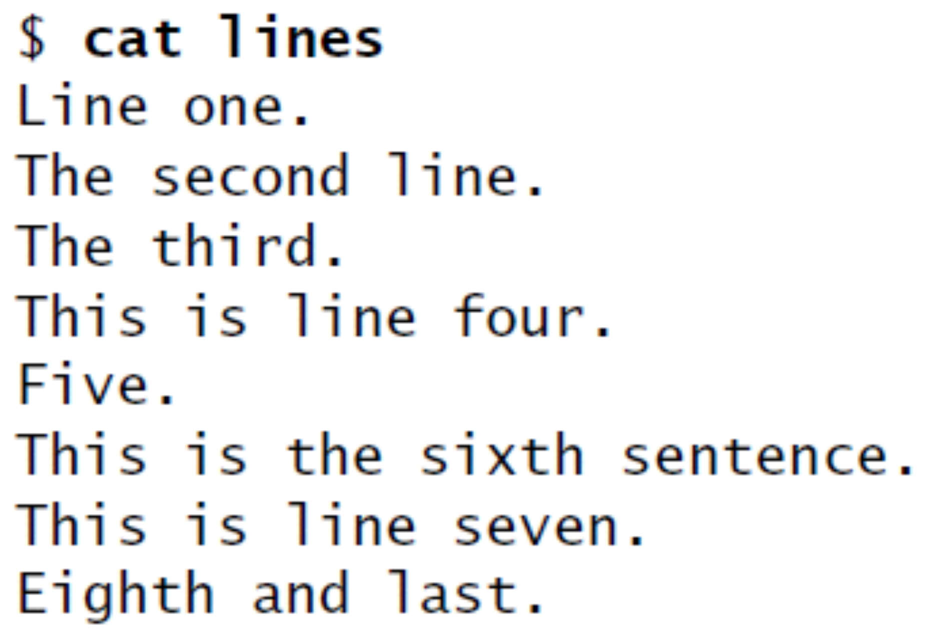
```
$ cat subs_demo
s/line/sentence/p
```

```
$ sed -n -f subs_demo lines
The second sentence.
This is sentence four.
This is sentence seven.
```

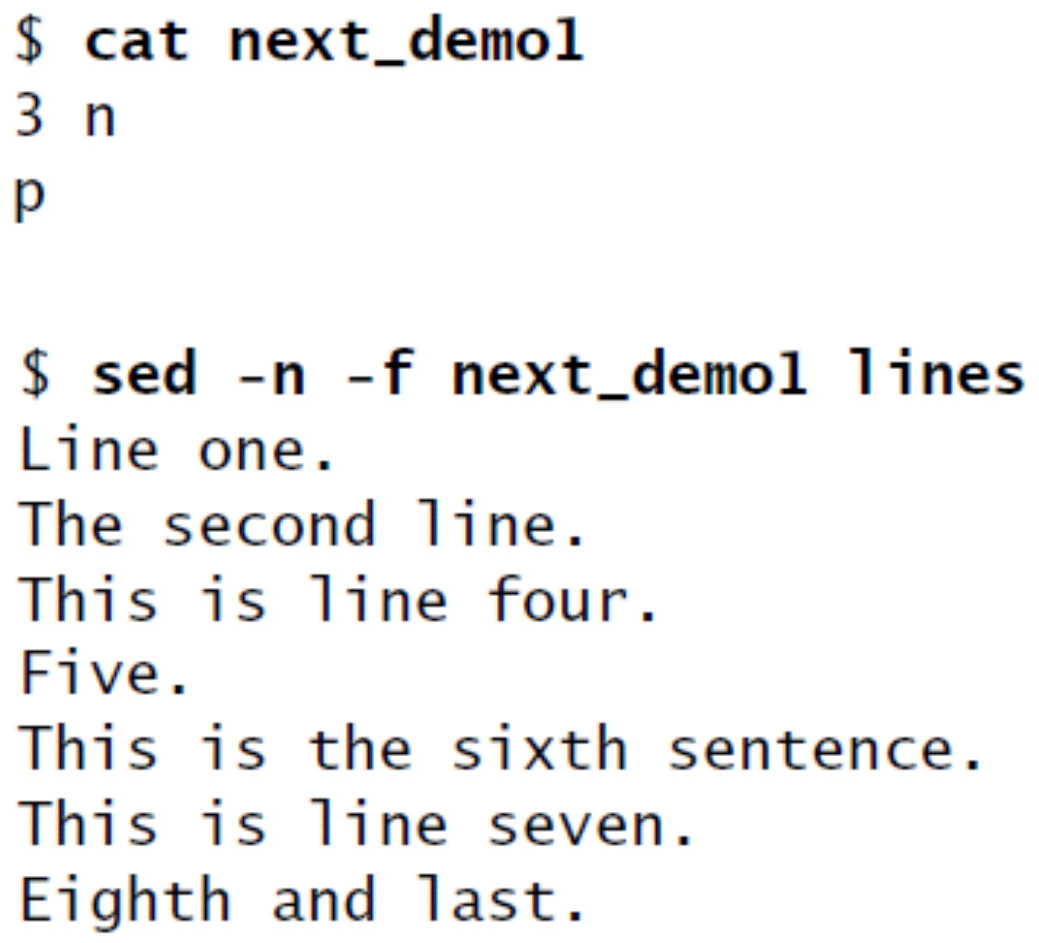
sed -n 's/line/sentence/p' lines

Instruction-Next (n)

- Reads the next input line, and starts processing the new line with the next instruction



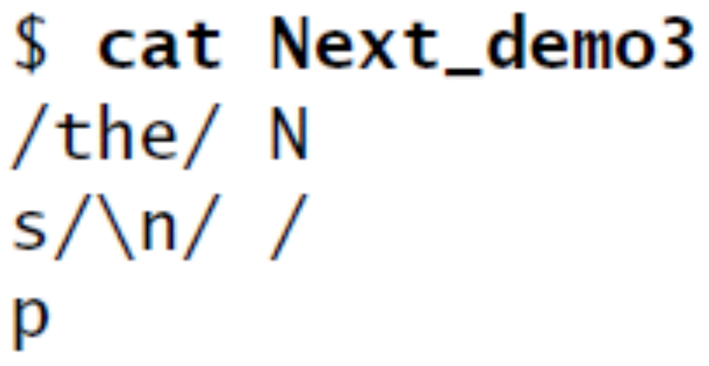
```
$ cat lines
Line one.
The second line.
The third.
This is line four.
Five.
This is the sixth sentence.
This is line seven.
Eighth and last.
```



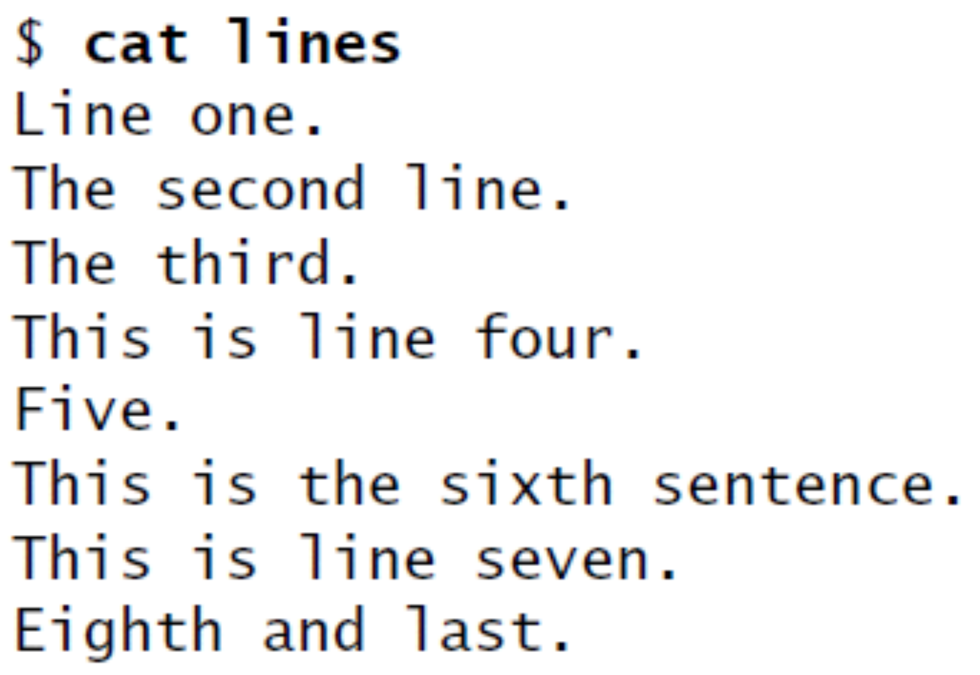
```
$ cat next_demo1
3 n
p

$ sed -n -f next_demo1 lines
Line one.
The second line.
This is line four.
Five.
This is the sixth sentence.
This is line seven.
Eighth and last.
```

Instruction- Next (N)

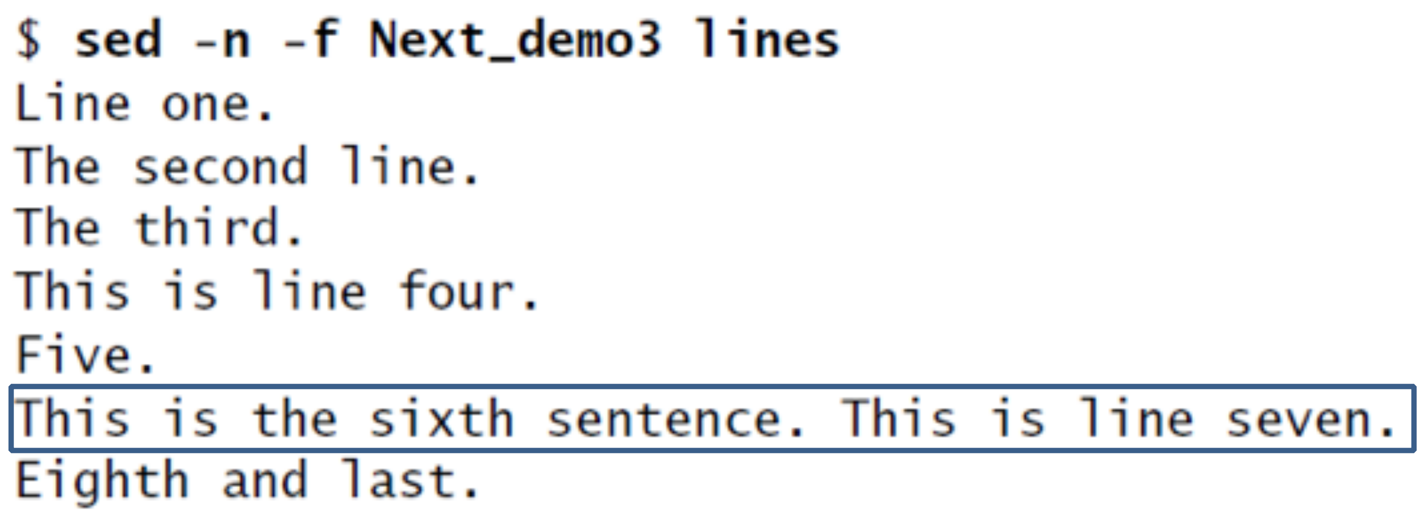
- Reads the next input line, and appends it to the current line. The two lines are separated by an embedded NEWLINE character.



```
$ cat Next_demo3
/the/ N
s/\n/ /
p
```



```
$ cat lines
Line one.
The second line.
The third.
This is line four.
Five.
This is the sixth sentence.
This is line seven.
Eighth and last.
```



```
$ sed -n -f Next_demo3 lines
Line one.
The second line.
The third.
This is line four.
Five.
This is the sixth sentence. This is line seven.
Eighth and last.
```

Instruction-Write (w)

```
$ cat test.txt

#!/bin/bash
 function hello{
 echo "hello";
}
hello;
```

```
$ cat script.txt
s/hello/&/w output
```

```
$ sed -n -f script.txt test.txt
$ ls
output  script.txt   test.txt
$ cat output
 function hello{
 echo "hello";
hello;
```

Another example

This example changes lower case vowels to upper case

```
$ cat test.txt
#!/bin/bash
 function hello{
 echo "hello";
}
hello;
```

```
$ cat script.txt
#!/bin/sed -f
s/a/A/g
s/e/E/g
s/i/I/g
s/o/O/g
s/u/U/g
```

```
/g: global replacement

$ sed -f script.txt test.txt
#!/bIn/bAsh
 fUnctIOn hEllO{
 EchO "hEllO";
}
hEllO;
```

Use sed in a shell

script a.txt:

```
#!/bin/bash
echo -n 'what is the value? '
read value
sed  's/XXX/'$value'/' <<EOF
The value is XXX
EOF
```

```
$ ./a.txt
what is the value? 1234
The value is 1234
```