CS 420 – Compilers

An Introduction to Regular Expression
grep and regex in Linux/Unix Environment

Dr. Chen-Yeou (Charles) Yu

- Why Regular Expression?
- What is Regular Expression (RE)?
- How RE works?
- “grep” command with RE
- “egrep” command with RE
- “grep” for backreferences
- Practical Regex Examples
- The spec of “grep”
- “grep” examples
- Quick Reference

Why Regular Expressions?

- Allow you to search for text in files
- Allow you to search for text based on some other program's output (by using pipeline)
- Allow you to use regular expression along with other powerful tools
  - grep, egrep
  - sed
  - awk
  - shell script

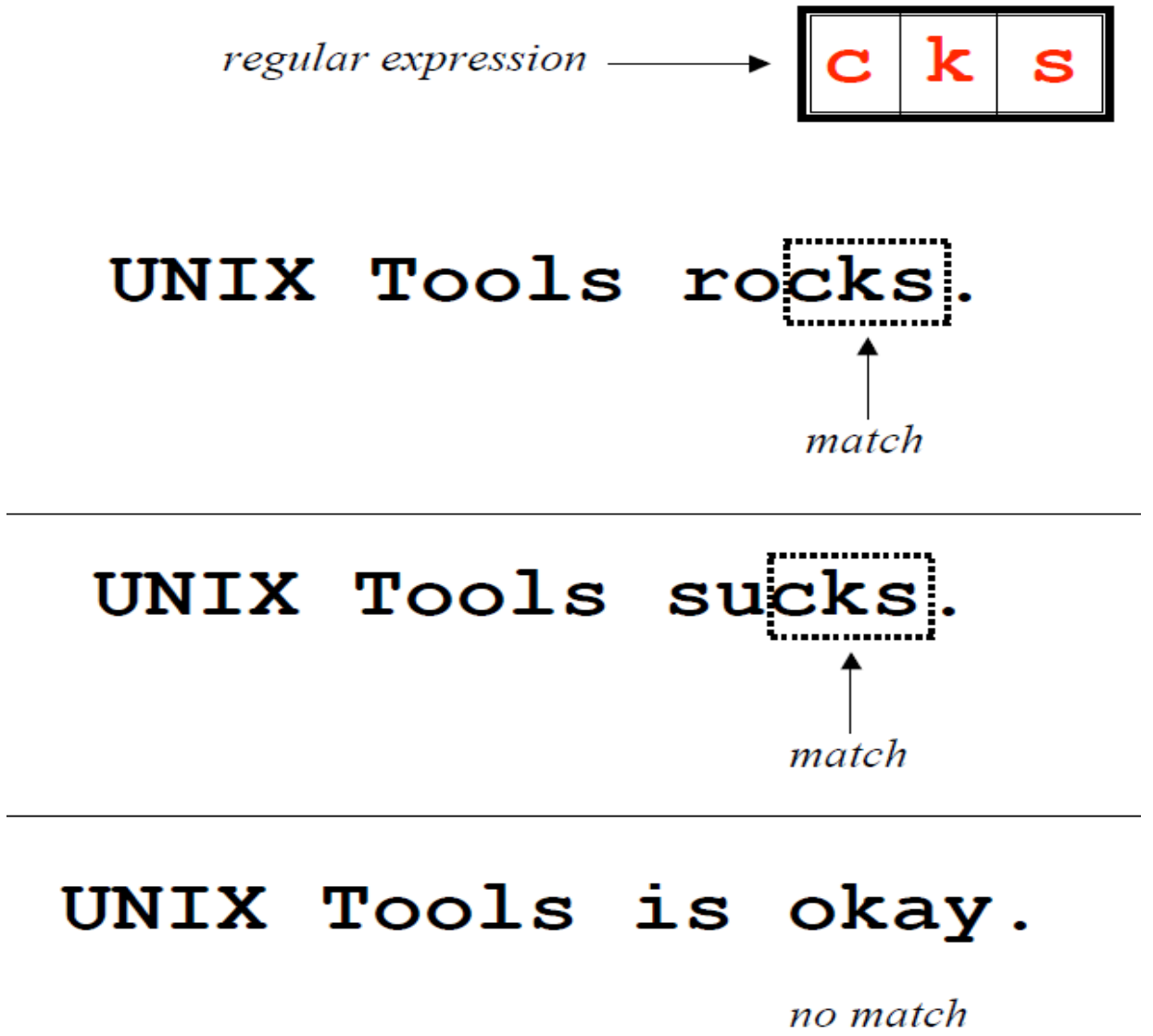
What is Regular Expression?

- A regular expression (*regex*) describes a set of possible input strings.
- *Regular expressions* descend from a fundamental concept in Computer Science called *finite automata* theory

How RE works?

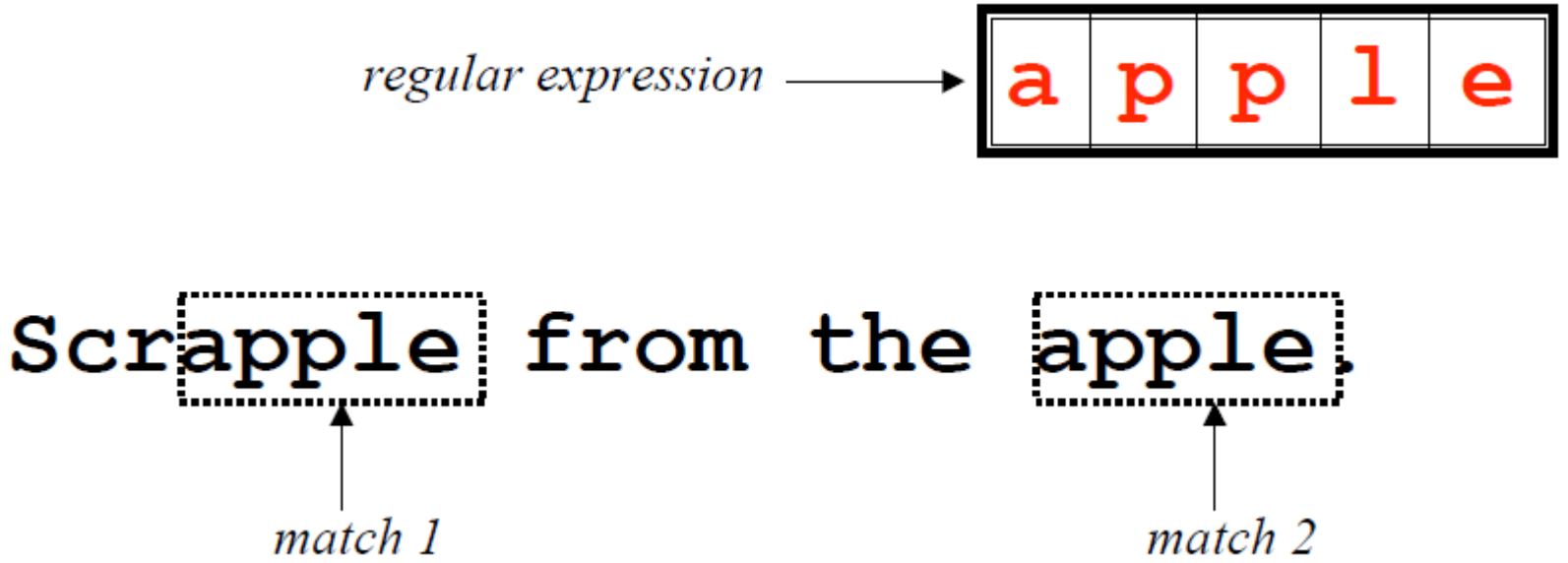
- In one sentence.
  - The input string ***matches*** the RE (like a pattern) if it contains the substring.

How RE works?


regular expression
c k s
UNIX Tools rocks.
match
UNIX Tools sucks.
match
UNIX Tools is okay.
no match

How RE works?

- A RE can match a string in **more than one place**.

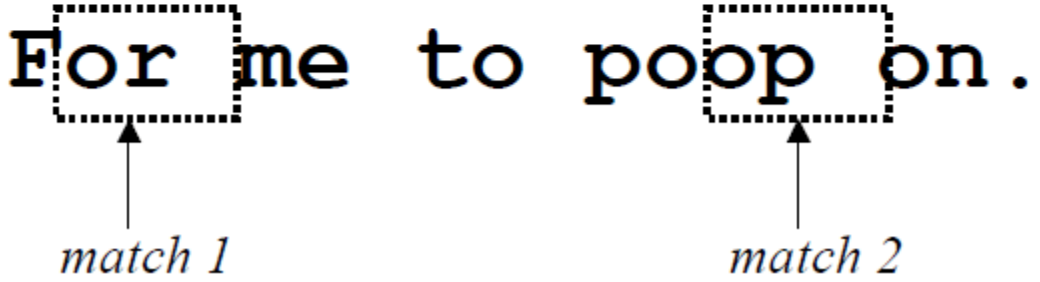


How RE works?

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

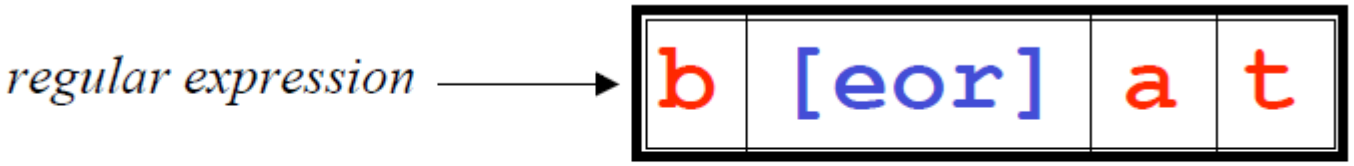
(the **single** “dot”)

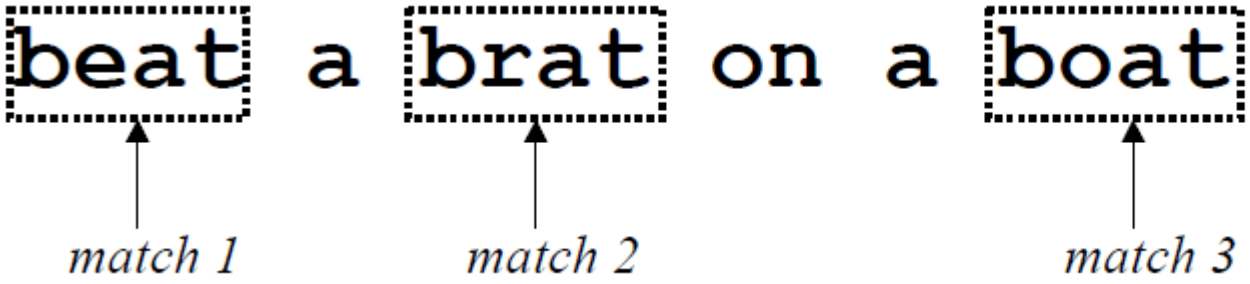
The pattern needs to have 3 characters. The 1st one is an “o” and the 2nd one is any character. The 3rd one is a space.



How RE works?

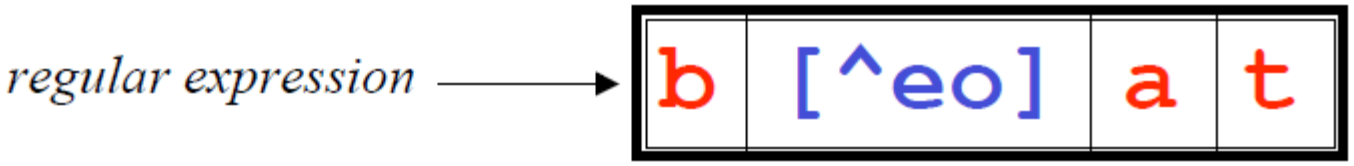
- **Character Classes**
  - It is denoted by []
  - This can be used to match any specific **set** of characters

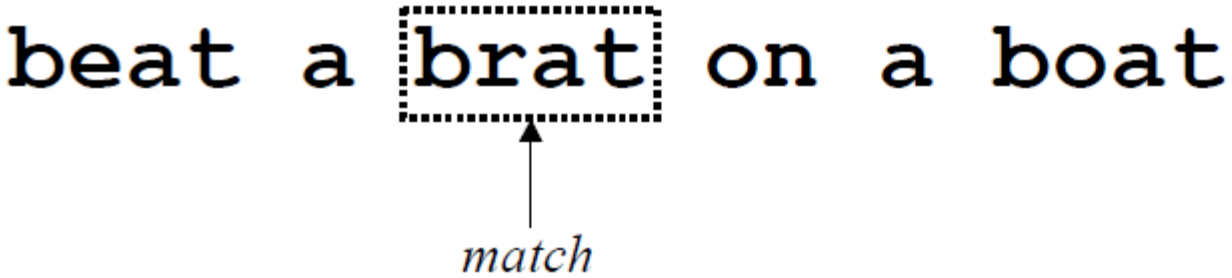




How RE works?

- The [^] syntax can be applied onto character classes
- This means, the character classes can be negated with the [^] syntax
- In this example, this means, except ‘e’ and ‘o’, I can match any other characters!
- (More examples are on the next page!)





How RE works?

- **[aeiou]** will match **any** of the characters: **a**, **e**, **i**, **o**, **u**
- **[mM]ohri** will match **mohri** or **Mohri**
- **[1-9]** is the same as **[123456789]**
- **[abcde]** is equivalent to **[a-e]** ← **The range in this way would be very flexible!**
- **[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
  - **i.e. [-123]** will lead to a match of characters **-**, **1**, **2**, or **3**

How RE works?

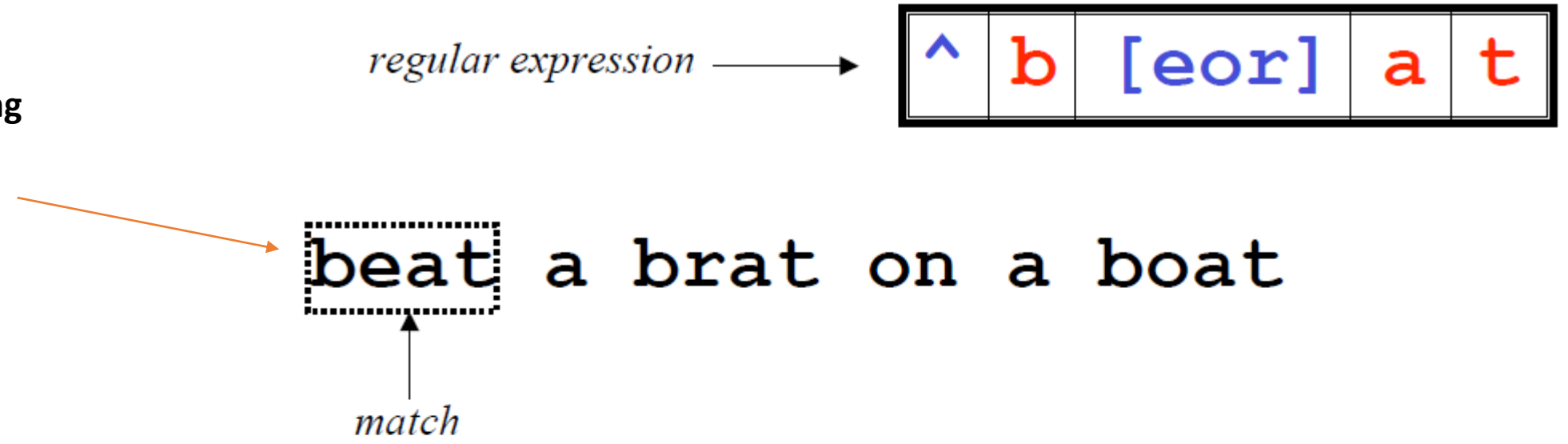
- Named Character Classes
  - Commonly used character classes can be referred to by name (*alpha*, *lower, upper, alnum*, *digit*, *punct*, *cntrl*)
  - This [:::] is going to match a single colon ":"
  - Syntax `[:name:]`
    - `[a-zA-Z]` `[[:alpha:]]`
    - `[a-zA-Z0-9]` `[[:alnum:]]`
    - `[45a-z]` `[45[:lower:]]`

How RE works?

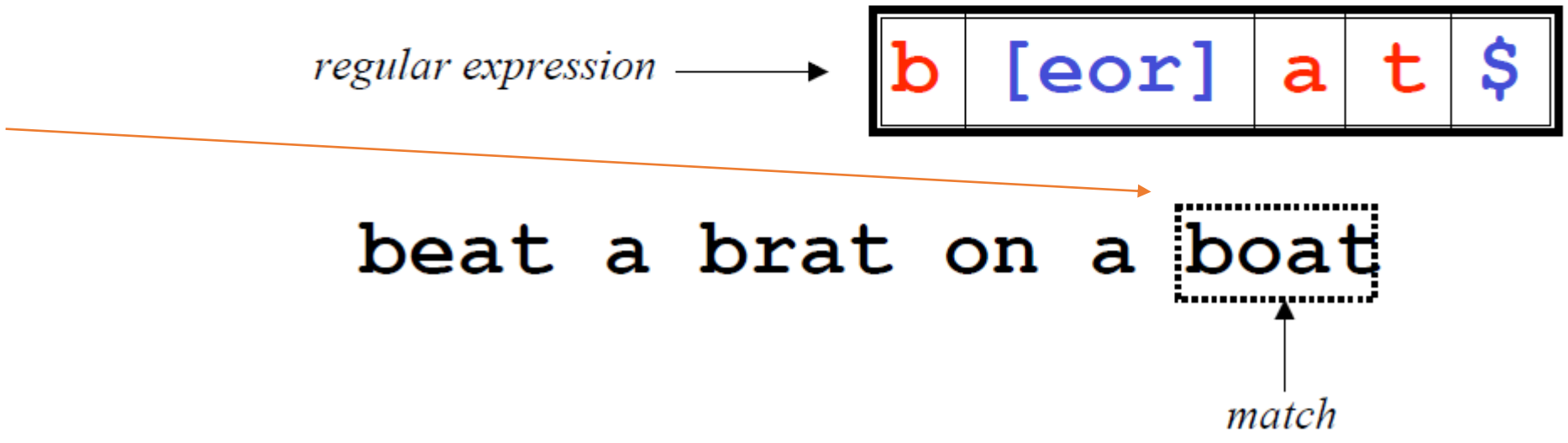
- Anchors
  - Anchors are used to match at the beginning or end of a line (or both).
  - **‘^’** means **beginning of the line**
    - Note that, this is not the ‘negate’ for character class ‘[…]’ any more!
  - ‘$’ means **end of the line**

How RE works?

Matches the beginning of the line for '^'

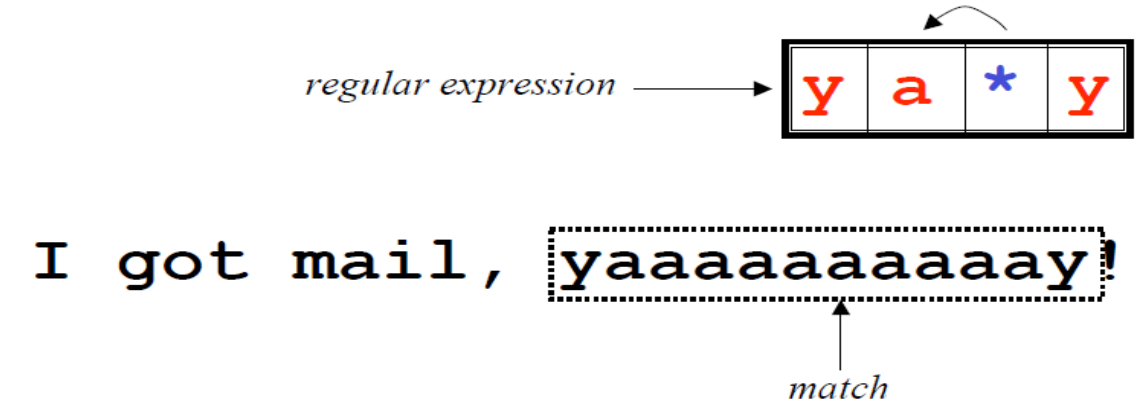


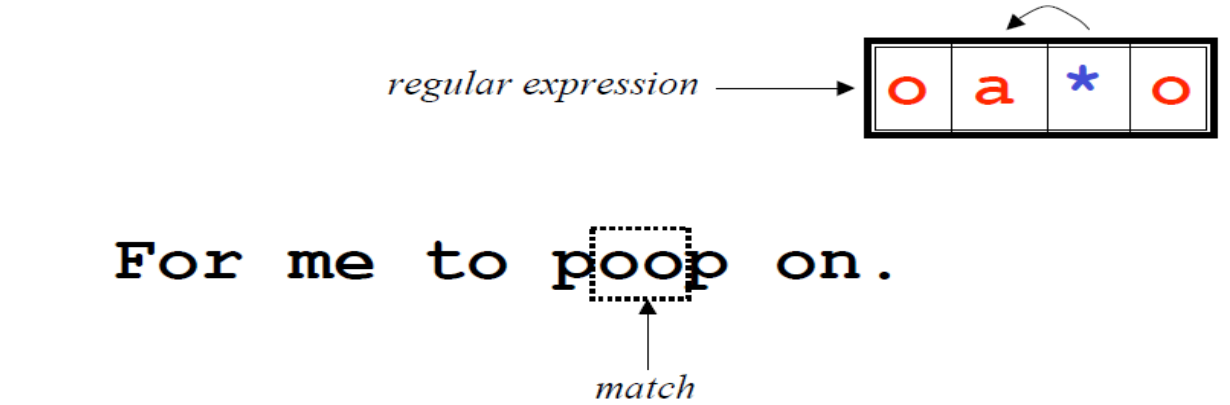
Matches the end of the line for '$'



How RE works?

- Repetition
  - The * is used to define **zero or more occurrences** of the ***single RE*** **preceding** it.





How RE works?

- Ranges can also be specified
  - **{ }** notation can specify a range of repetitions for the immediately preceding regex
  - **{*n*}** means exactly *n* occurrences
  - **{*n*, }** means at least *n* occurrences (Note! The 2nd parameter is an open wildcard!)
  - **{*n*, *m*}** means **at least *n* occurrences but no more than *m* occurrences** n also be specified
- Example:
  - **.{0, }** same as **.* (the content in the curly braces is to modify the 'dot')**
  - **a{2, }** same as **aaa* (a single 'a' repeated by 2 more occurrences, but the 3rd 'a' can be null, or more repeated 'a')**

How RE works?

- Subexpressions (purpose: to group a **part** of an expression)
  - If you want the * or { } applies to more than just the previous character, use the ( ) notation
  - Sub-expressions should be treated like a single character!
    - **a*** matches 0 or more occurrences of **a**
    - **abc*** matches **ab**, **abc**, **abcc**, **abccc**, …
    - But!
    - **(abc)*** matches: epsilon, **abc**, **abcabc**, **abcabcabc**, …
    - **(abc){2,3}** matches **abcabc** or **abcabcabc**
      - **{2, 3} is used to modify the occurrences of (abc)**

“grep” command with RE

- grep comes from the **ed** (Unix text editor) search command “**g**lobal **r**egular **e**xpression **p**rint” or g/re/p
- Now, it is a useful command that it was written as a standalone utility
- Two other variants, ***egrep*** and ***fgrep*** that comprise the *grep* family
- *grep* is the answer to the moments when you want to know what the file that contains a specific phrase but you can’t remember its name

“grep” command with RE

- Family Differences
  - **grep** - uses regular expressions for pattern matching
  - **fgrep** - file grep, does not use regular expressions, only matches fixed strings but can get search strings from a file
  - **egrep** - extended grep, uses a more powerful set of regular expressions but does not support backreferencing, generally the fastest member of the grep family
  - **agrep** – approximate grep; not standard

“grep” command with RE

- Syntax
  - Regular expression concepts we have seen so far are common to **grep** and **egrep**
  - grep and egrep have different syntax
    - **grep**: BREs (Basic)
    - **egrep**: EREs (Enhanced features we will discuss)
  - Major syntax differences:
    - **grep**: **\(** and **\)**, **\{** and **\}**
    - **egrep**: **(** and **)**, **{** and **}**

“grep” command with RE

- Escaping Special Characters
  - Even though we are single quoting our regexs so the shell won’t interpret the special characters, some characters are special **to grep** (eg * and .)
  - To get literal characters, we *escape* the character with a **** (backslash)
  - Suppose we want to search for the character sequence ‘a*b*’
    - The exactly matching of the ‘a star’ followed by a ‘b star’
  - Unless we do something special, 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

“egrep” command with RE

- egrep: An Alternation
  - Regex also provides an alternation character ‘**|**’ for matching one or another subexpression
    - **(T|Fl) an** will match ‘Tan’ or ‘Flan’
    - **^(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 ‘:’**
    - **At(ten|nine)tion** then matches “Attention” or “Atninetion”
    - **But, Atten|ninetion is different from the thing we mentioned above**

“egrep” command with RE

- Repetition Shorthand
  - The * (star) has already been seen to specify zero or more occurrences of the immediately preceding character
  - **+,** the plus, means “one or more”
  - **abc+d** will match ‘abcd’, ‘abccd’, or ‘abccccccd’ but will not match ‘abd’
  - Equivalent to **{1, }**

“egrep” command with RE

- Repetition Shorthand (cont.)
  - The ‘**?**’ (question mark) specifies **an optional character**, the single character that immediately precedes it
  - ? Is used to describe a **single** character, either existing or not
  - **July?** will match ‘Jul’ or ‘July’
    - Equivalent to **{0,1}**
    - Also equivalent to **(Jul|July)**
  - The ***, **?**, and **+** are known as *quantifiers* because they specify the quantity of a match
  - Quantifiers can also be used with subexpressions.
    - **(a*c)+** will match ‘c’, ‘ac’, ‘aac’ or ‘aacaacac’ but will not match ‘a’ or a epsilon

“grep” for backreferences

- Sometimes it is handy to be able to refer to a match that was made earlier in a regex
  - i.e. We want to get the last match if there are many
- This is done using *backreferences*
  - *n* is the backreference specifier, where *n* is the number
- This is the most mysterious and the most powerful part (back references) what the RE can do to help the grep
- Remember “egrep” cannot support backreferences, right?

“grep” for backreferences

- For example, if we want to find if the ***first*** word of a line is the ***same*** as the ***last***:

  The `\([[:alpha:]]\{1,\}\)` matches 1 or more letters

  - If we plug the explanation we mentioned above to the RE below, that means the **word**, at the beginning of a line with a “^”!

```
^\([[:alpha:]]\{1,\}\) .* \1$
```

  - Check the slide #12, what is this [[:alpha:]]
  - Check the slide #8, the single dot means any char, that can repeat several times, for .*
  - Check the slide #13 for dollar sign, $, means the end of the line
- I want to get the first word of the line, and use that as a pattern to **match** the 1st last capturing group (word) to the end of this line → 1st word is the same as last word

“grep” for backreferences

- We won’t go through too detail because it is “overly” mysterious.
- It needs at least a month to explain and the content would be like a book that kind of thickness!
- Just getting you the smell how it looks like!

Practical Regex Examples

- Variable names in C
  - **[a-zA-Z_][a-zA-Z_0-9]***
- HTML headers <h1> <H1> <h2> …
  - **<[hH][1-4]>**
- Dollar amount with optional cents (i.e. $1.25)
  - **\$[0-9]+(\.[0-9][0-9])?**

The spec of “grep”

grep [-hilnv] [-e expression] [filename]

egrep [-hilnv] [-e expression] [-f filename] [expression] [filename]

fgrep [-hilnxv] [-e string] [-f filename] [string] [filename]

- **-h** Do not display filenames
- **-i** Ignore case
- **-l** List only filenames containing matching lines
- **-n** Precede each matching line with its line number
- **-v** Negate matches
- **-x** Match whole line only (*fgrep* only)
- **-e** *expression* Specify expression as option
- **-f** *filename* Take the regular expression (egrep) or a list of strings (fgrep) from *filename*

“grep” examples

- grep 'men' GrepFile
- grep 'fo*' GrepFile
- egrep 'fo+' GrepFile
- Find all lines with signed numbers, limit the searching to xxxx.c file

```
$ egrep '[-+][0-9]+\.?[0-9]*' *.c
bsearch. c: return -1;
compile. c: strchr("+1-2*3", t-> op)[1] - '0', dst,
convert. c: Print integers in a given base 2-16 (default 10)
convert. c: sscanf( argv[ i+1], "% d", &base);
strcmp. c: return -1;
strcmp. c: return +1;
```

- It starts with – or +, followed by [0-9]+, then the “dot” which is optional. Followed by [0-9] for several times

“grep” examples

- How many words have 3 characters, with one letter apart?
  - egrep u.u.u GrepFile
    - cumulus
  - egrep a.a.a GrepFile | wc -l
    - 54
    - “-l” means, how many “Lines”

Quick Reference

x	Ordinary characters match themselves (NEWLINES and metacharacters excluded)	*fgrep, grep, egrep*
xyz	Ordinary strings match themselves	
\m	Matches literal character *m*	
^	Start of line	
$	End of line	
.	Any single character	
[xy^$x]	Any of x, y, ^, $, or z	*grep, egrep*
[^xy^$z]	Any one character other than x, y, ^, $, or z	
[a-z]	Any single character in given range	
r*	zero or more occurrences of regex r	
r1r2	Matches r1 followed by r2	
\(r\)	Tagged regular expression, matches r	
\n	Set to what matched the *n*th tagged expression (n = 1-9)	*grep*
\{n,m\}	Repetition	
r+	One or more occurrences of r	
r?	Zero or one occurrences of r	
r1|r2	Either r1 or r2	
(r1|r2)r3	Either r1r3 or r2r3	
(r1|r2)*	Zero or more occurrences of r1|r2, e.g., r1, r1r1, r2r1, r1r1r2r1,…)	*egrep*
{n,m}	Repetition	

Quick Reference