Welcome!

- **today**
  - administrative stuff
  - course overview
  - regular expressions and grep

- **check out the course web page (CS, not Blackboard) and Piazza**
  - notes, readings and assignments are posted only on the web page; monitor the web page and Piazza every day
  - Assignment 1 is posted; due 10:00 pm Thursday Feb 14
  - initial project information is posted (more on Thursday)

- **please do the survey if you haven't already**

People

Christopher Moretti

Allison Chang '18

Jace Lu

Lance Goodridge '17

Very Tentative Outline

week 1	regular expressions, grep; project info
week 2	scripting: shell, AWK, Python
week 3	web technology: HTTP, CSS, Javascript
week 4	client libraries and server frameworks
week 5	user interfaces; phone apps
week 6	databases; software engineering
	(spring break)
week 7	networks
week 8	advanced C++, Java; Go
week 9	APIs, design patterns
week 10	XML, JSON, REST, DSLs
week 11	??
week 12	??
	(start of reading period)
May 8-9?	project presentations
May 12	project submission (2 days before Dean's Date)

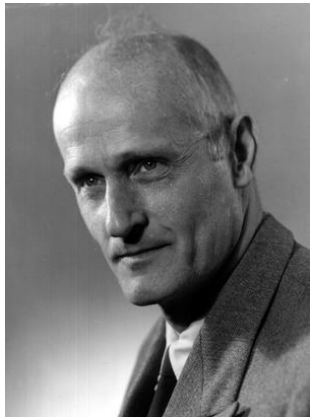
House rules

- please turn cell phones off
- please don't use your laptop, tablet, phone, ...
  - it distracts you
  - it distracts your neighbors
  - it distracts me
- please don't snore (sleeping is ok)
- please sit towards the front, not in the back
- please stay away if you're sick ! ! !

- please ask questions about anything at any time

Regular expressions and grep

- **regular expressions**
  - based on ideas from automata theory pioneered by Stephen Kleene *34
  - notation
  - mechanization
  - pervasive in Unix tools
  - in all scripting languages, often as part of the syntax
  - in general-purpose languages, as libraries
  - basic implementation is remarkably simple
  - efficient implementation requires good theory and good practice



- **grep is the prototypical tool**
  - written by Ken Thompson @ Bell Labs ~1972

Grep regular expressions

`c` any character matches itself, except for
metacharacters `. [ ] ^ $ * \`

$r_1 r_2$ matches r_1 followed by r_2

`.` matches any single character

`[...]` matches one of the characters in set ...
shorthand like a-z or 0-9 includes any character in the range

`[^...]` matches one of the characters not in set
[^0-9] matches non-digit

`^` matches beginning of line when ^ begins pattern
no special meaning elsewhere in pattern

`$` matches end of line when $ ends pattern
no special meaning elsewhere in pattern

`*` any regular expression followed by * matches 0 or more

`\c` matches c unless c is `( )` or digit

`\(...\)` tagged regular expression that matches ...
the matched strings are available as \1, \2, etc.

Examples of matching

`xy`	*xy* anywhere in string
`^xy`	*xy* at beginning of string
`xy$`	*xy* at end of string
`^xy$`	string that contains only *xy*
`^`	matches any string, even empty
`^$`	empty string
`.`	non-empty string, i.e., at least 1 char
`xy.$`	*xy* plus any char at end of string
`xy\.$`	*xy*. at end of string
`\\xy\\`	*xy*\ anywhere in string
`[xX]y`	*xy* or *Xy* anywhere in string
`xy[0-9]`	*xy* followed by one digit
`xy[^0-9]`	*xy* followed by a non-digit
`xy[0-9][^0-9]`	*xy* followed by digit, then non-digit
`xy1.*xy2`	*xy1* then any text then *xy2*
`^xy1.*xy2$`	*xy1* at beginning and *xy2* at end

"Regular expressions" are not always regular

- there's a precise definition but lots of casual usage

- R: c R_1R_2 $R_1 | R_2$ (R) R*
  - equivalent to a finite automaton
  - this is what egrep provides

- shorthands like [A-Z], \d, [:alnum:], etc., don't change properties
  - can't count, can't recognize repeated strings, ...

- can have subsets that do less (coming up)

- can do much more than pure REs:
  - supersets (back-referencing in grep)
  - libraries that are Turing-complete (Java, Python, etc.)
  - extra-lingual processing (commandline arguments like grep –i –v --color)

egrep: fancier regular expressions

`r+`	one or more occurrences of r
`r?`	zero or one occurrences of r
r_1`|`r_2	r_1 or r_2
`(r)`	r (grouping)


Al Aho *67

grammar:

`r:` `c` `.` `^` `$` `[ccc]` `[^ccc]`

`r*` `r+` `r?`

r_1 r_2

r_1`|`r_2

`(r)`

precedence:

`*` `+` `?` higher than concatenation, which is higher than |

```
([0-9]+\.?[0-9]*|\.[0-9]+)([Ee][-+]?[0-9]+)?
```

Python RE's

^	Matches beginning of line.
$	Matches end of line.
.	Matches any single character except newline. Using m option allows it to match newline as well.
[...]	Matches any single character in brackets.
[^...]	Matches any single character not in brackets
re*	Matches 0 or more occurrences of preceding expression.
re+	Matches 1 or more occurrence of preceding expression.
re?	Matches 0 or 1 occurrence of preceding expression.
re{n}	Matches exactly n number of occurrences of preceding expression.
re{n,}	Matches n or more occurrences of preceding expression.
re{n, m}	Matches at least n and at most m occurrences of preceding expression.
a|b	Matches either a or b.
(re)	Groups regular expressions and remembers matched text.
(?imx)	Temporarily toggles on i, m, or x options within a regular expression. If in parentheses, only that area is affected.
(?-imx)	Temporarily toggles off i, m, or x options within a regular expression. If in parentheses, only that area is affected.
(?: re)	Groups regular expressions without remembering matched text.
(?imx: re)	Temporarily toggles on i, m, or x options within parentheses.
(?-imx: re)	Temporarily toggles off i, m, or x options within parentheses.
(?#...)	Comment.
(?= re)	Specifies position using a pattern. Doesn't have a range.
(?! re)	Specifies position using pattern negation. Doesn't have a range.
(?> re)	Matches independent pattern without backtracking.
\w	Matches word characters.
\W	Matches nonword characters.
\s	Matches whitespace. Equivalent to [\t\n\r\f].
\S	Matches nonwhitespace.
\d	Matches digits. Equivalent to [0-9].
\D	Matches nondigits.
\A	Matches beginning of string.
\Z	Matches end of string. If a newline exists, it matches just before newline.
\z	Matches end of string.
\G	Matches point where last match finished.
\b	Matches word boundaries when outside brackets. Matches backspace (0x08) when inside brackets.
\B	Matches nonword boundaries.
\n, \t, etc.	Matches newlines, carriage returns, tabs, etc.
\1...\9	Matches nth grouped subexpression.
\10	Matches nth grouped subexpression if it matched already. Otherwise refers to the octal representation of a character code.

The grep family

- **grep**
- **egrep**
  - fancier regular expressions, trades compile time and space for run time
- **fgrep**
  - parallel search for many fixed strings
- **agrep**
  - "approximate" grep: search with errors permitted
- **relatives that use similar regular expressions**
  - ed — original Unix editor
  - sed — stream editor
  - vi, emacs, sam, ... — editors
  - lex, flex — lexical analyzer generator
  - awk, perl, python, … — all scripting languages
  - Java, C# ... — libraries in mainstream languages
- **simpler variants**
  - filename "wild cards" in Unix and other shells (assignment 1 this year)
  - "LIKE" operator in SQL, Visual Basic, etc.

Important ideas from regexprs & grep

- tools: let the machine do the work
  - good packaging matters
- notation: makes it easy to say what to do
  - may organize or define implementation
- hacking can make a program faster, sometimes, usually at the price of more complexity
- a better algorithm can make a program go a lot faster
- don't worry about performance if it doesn't matter (and it often doesn't)
- when it does,
  - use the right algorithm
  - use the compiler's optimization
  - code tune, as a last resort