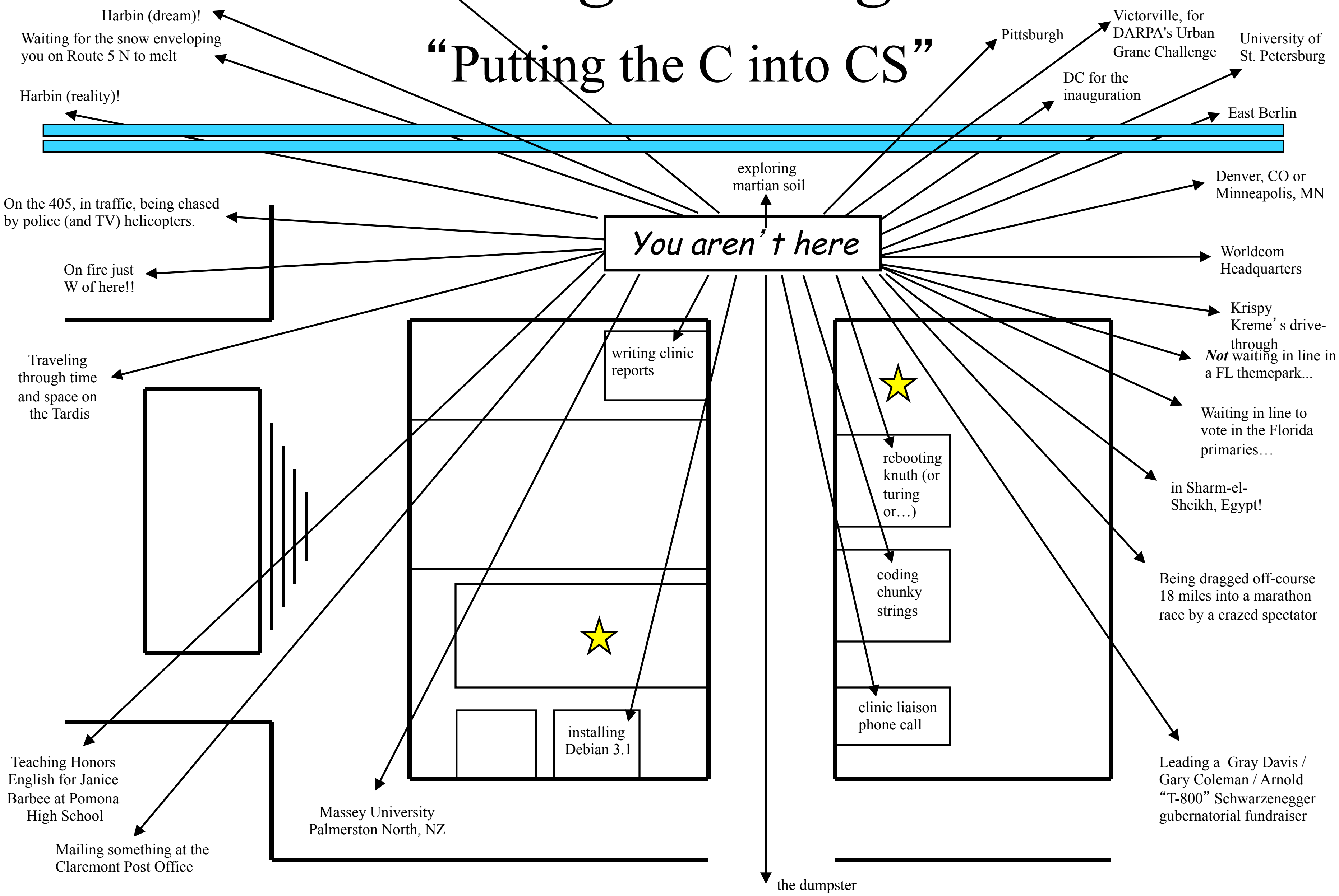
Welcome to *Programming Practicum*

"Putting the C into CS"



Introductions…!

Zach Dodds

Office **Olin B163**

Email **dodds@cs.hmc.edu**

not afraid of stuffed animals!

fan of *low-level* AI

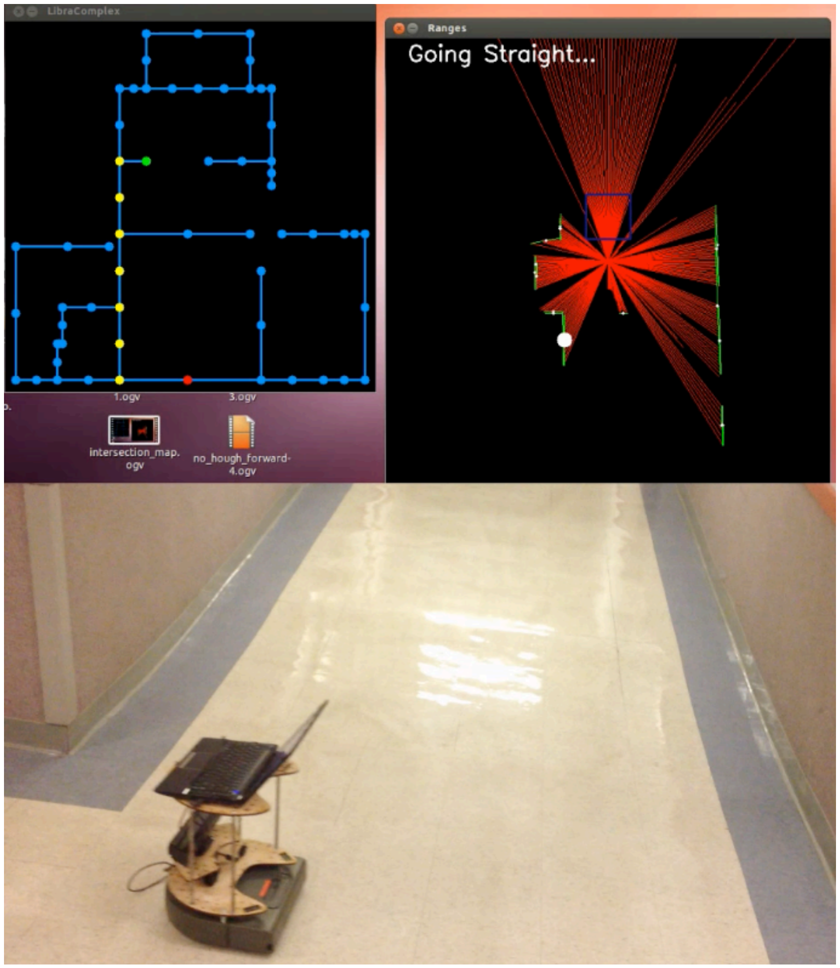
taker of *low-quality* photos

Starbucks triumph-er!

and not good at selfies…

How I spent my summer vacation...

programming robots



Or, more precisely, cheering for many other folks programming robots!

visiting important landmarks!

My selfie-taking has gotten worse over the past couple of months!

What is this course about?

practicing algorithmic/programming skills

first half…

until early November

trying out technologies/projects of interest

second half…

after early November, *if you'd like*

trying out technologies/projects of interest

after early November, *if you'd like*

Alums: What do you feel you ***didn't get*** @ HMC CS?

Will Scott

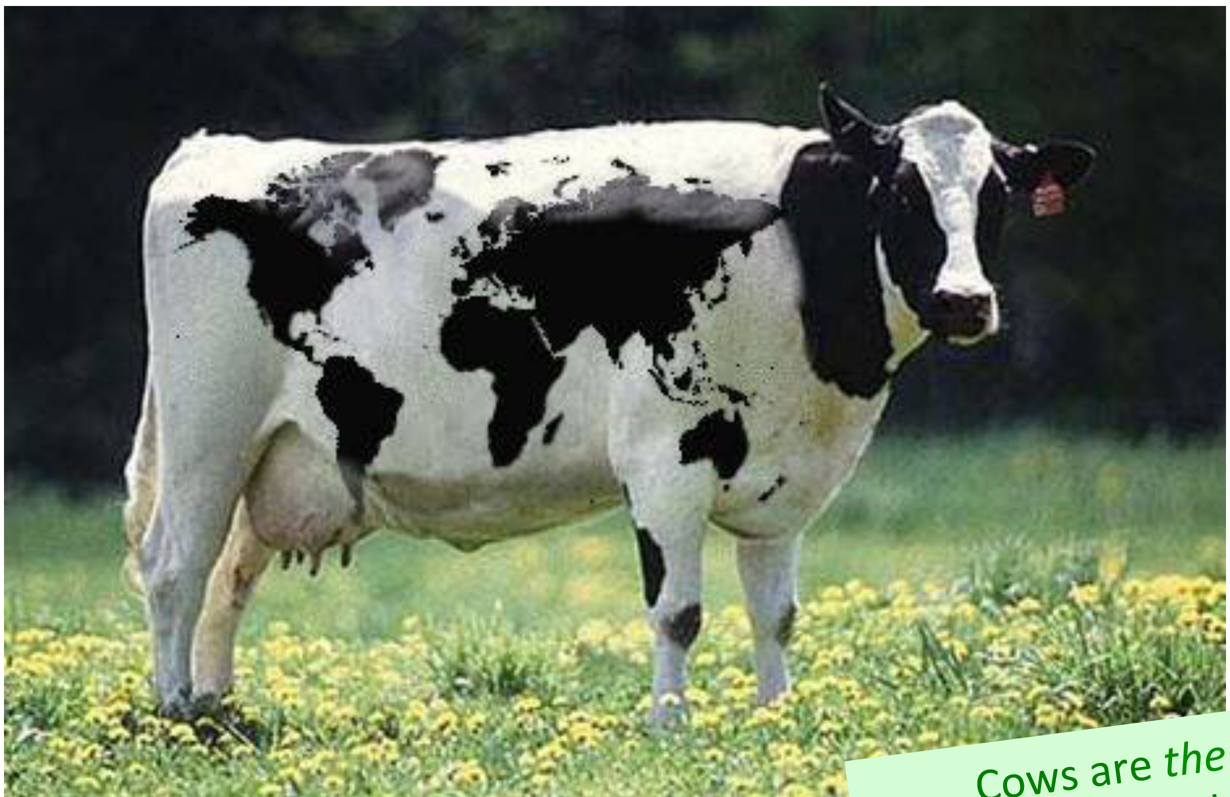
trying out technologies/projects of interest

after early November, *if you'd like*

Alums: What do you feel you ***didn't get*** @ HMC CS?

What is the first ½ about?

practicing algorithmic/programming skills



Bessie!

Cows are *the* ***global*** theme of CS189's problems.

Example

elevator.py
elevator.java
elevator.cc

```
Space Elevator

The cows are going to space! They plan to achieve orbit by building
a sort of space elevator: a giant tower of blocks. They have K (1
<= K <= 400) different types of blocks with which to build the
tower.  Each block of type i has height h_i (1 <= h_i <= 100) and
is available in quantity c_i (1 <= c_i <= 10). Due to possible
damage caused by cosmic rays, no part of a block of type i can
exceed a maximum altitude a_i (1 <= a_i <= 40000).

Help the cows build the tallest space elevator possible by stacking
blocks on top of each other according to the rules.

PROBLEM NAME: elevator.X
```

Example

elevator.py
elevator.java
elevator.cc

```
Space Elevator

The cows are going to space! They plan to achieve orbit by building
a sort of space elevator: a giant tower of blocks. They have K (1
<= K <= 400) different types of blocks with which to build the
tower.  Each block of type i has height h_i (1 <= h_i <= 100) and
is available in quantity c_i (1 <= c_i <= 10). Due to possible
damage caused by cosmic rays, no part of a block of type i can
exceed a maximum altitude a_i (1 <= a_i <= 40000).

Help the cows build the tallest space elevator possible by stacking
blocks on top of each other according to the rules.

PROBLEM NAME: elevator.X
```

Input

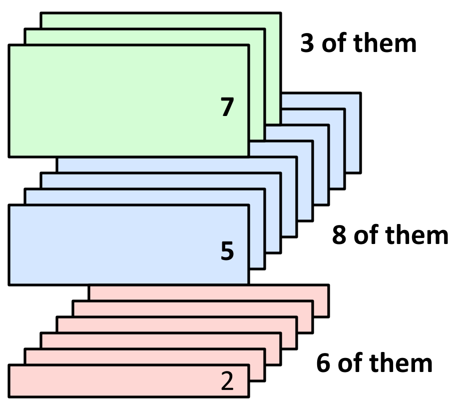
of blocktypes

```
3
7  40  3
5  23  8
2  52  6
```

block height

max altitude

quantity (count)



Example

elevator.py
elevator.java
elevator.cc

```
Space Elevator

The cows are going to space! They plan to achieve orbit by building
a sort of space elevator: a giant tower of blocks. They have K (1
<= K <= 400) different types of blocks with which to build the
tower.  Each block of type i has height h_i (1 <= h_i <= 100) and
is available in quantity c_i (1 <= c_i <= 10). Due to possible
damage caused by cosmic rays, no part of a block of type i can
exceed a maximum altitude a_i (1 <= a_i <= 40000).

Help the cows build the tallest space elevator possible by stacking
blocks on top of each other according to the rules.

PROBLEM NAME: elevator.X
```

Input

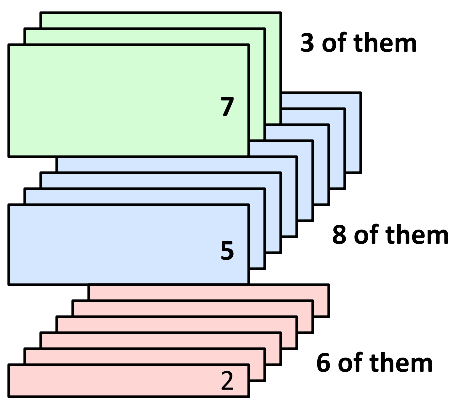
of blocktypes

```
3
7  40  3
5  23  8
2  52  6
```

block height

max altitude

quantity (count)

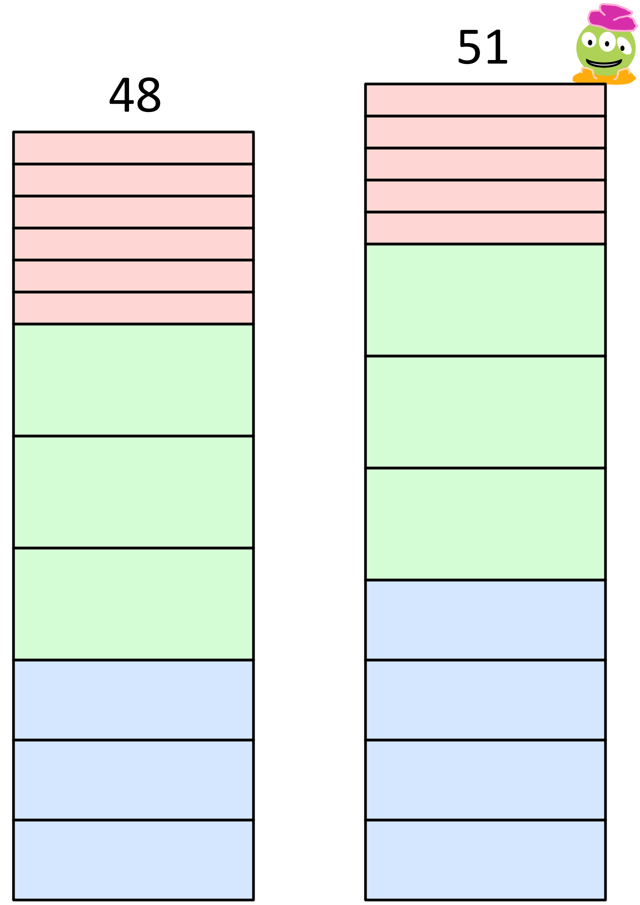


Output

48

The height of the tallest tower possible

What's wrong with this tower?



practicing algorithmic/programming skills

What

Algorithm analysis and insight

Program design and implementation

optimizing ***coding*** time,
as well as ***running*** time

Why

ACM programming contest

Hands-on practice with algorithms and techniques

Familiarizing with your choice of language/libraries

"reasonable"

Research/prototype programming

Technical interview questions…

Unofficial course name: CS -70

Class Meetings

first half

alternating format

discussion sessions

- problem and program analysis
- discussion of strategy and coding tips
- deciding on functional decomposition, data structures, language facilities, and algorithms to use in teams of 2-3
- short time to work on at least 1 problem

lab sessions

- more extended team problem-solving practice: coming to the problems "cold"
- these problems count for ***each*** member of the group

- sometimes new problems, other times with known ones
- ~5-6 problems given out per week…

Course Organization

Sep 10	Welcome discussion! and DP problems ~ **5 problems**
Sep 17	Lab session ~ **5 problems**
Sep 24	Discussion session on graph problems ~ **5 problems**
Oct 1	Lab session on graph problems ~ **5 problems**
Oct 8	***Guest speaker*** Don Chamberlin, author/inventor of SQL = **2 problems**
Oct 16	Discussion session on geometry problems ~ **5 problems**
Oct 22	Lab & ACM qualifying contest ~ **6 problems**
Oct 29	Discussion session on something new!! ~ **5 problems**
Nov 5	Final meeting: *project opportunities*
Nov 9	(approximate) **ACM Regional contest** (in Riverside...)

Rest of the term: *projects or problems*
-- *you choose* --

≥ 42 problems total
You may submit problems
until the end of exams…

part – but only *part* – of the motivation for CS 189:

ACM programming contest

2012-13 Final Standings

Rank	Team ID	Team Name	Solved	Penalties	Score
1	acm170	USC Trojans	9	11	24:59:34
2	acm107	Caltech A	8	3	17:25:48
3	acm151	UCLA Flyaway	6	2	7:23:47
4	acm122	UCSD Load, Spin, Pull	6	0	10:31:29
5	acm124	UCSD kamehb	6	5	11:49:16
6	acm121	HMC Squared	5	0	9[illegible]
7	acm168	USC Cardinal	5	2	10[illegible]
8	acm152	UCLA HeroesIII	5	2	11:38:56
9	acm157	UCI constructors	5	3	11:54:00
10	acm123	UCSD bumaga	4	1	5:20:39
11	acm109	Caltech D	4	3	7:43:22
12	acm119	HMC J	4	4	8:02:39
13	acm154	UCSB alpha	4	2	8:47:20
14	acm106	Caltech 1	4	3	9:05:09
15	acm158	UCI instances	4	2	9:27:40
16	acm111	CSUF-B	4	1	9:40:47
17	acm117	CSULB Undeclared Identifiers	4	2	10:22:22
18	acm108	Caltech C	4	3	10:39:47
19	acm118	HMC Escher	4	1	10:46:15
20	acm129	UCR Raphael	4	0	11:37:09

USC advanced to the finals in 2011 and 2012...


75 teams...
Jackson!


PLEASE!
No Cell Phones
No Web Surfing
No Gaming
No E-Mailing
NO ONLINE or POCKET TEXT TRANSLATORS
DELL

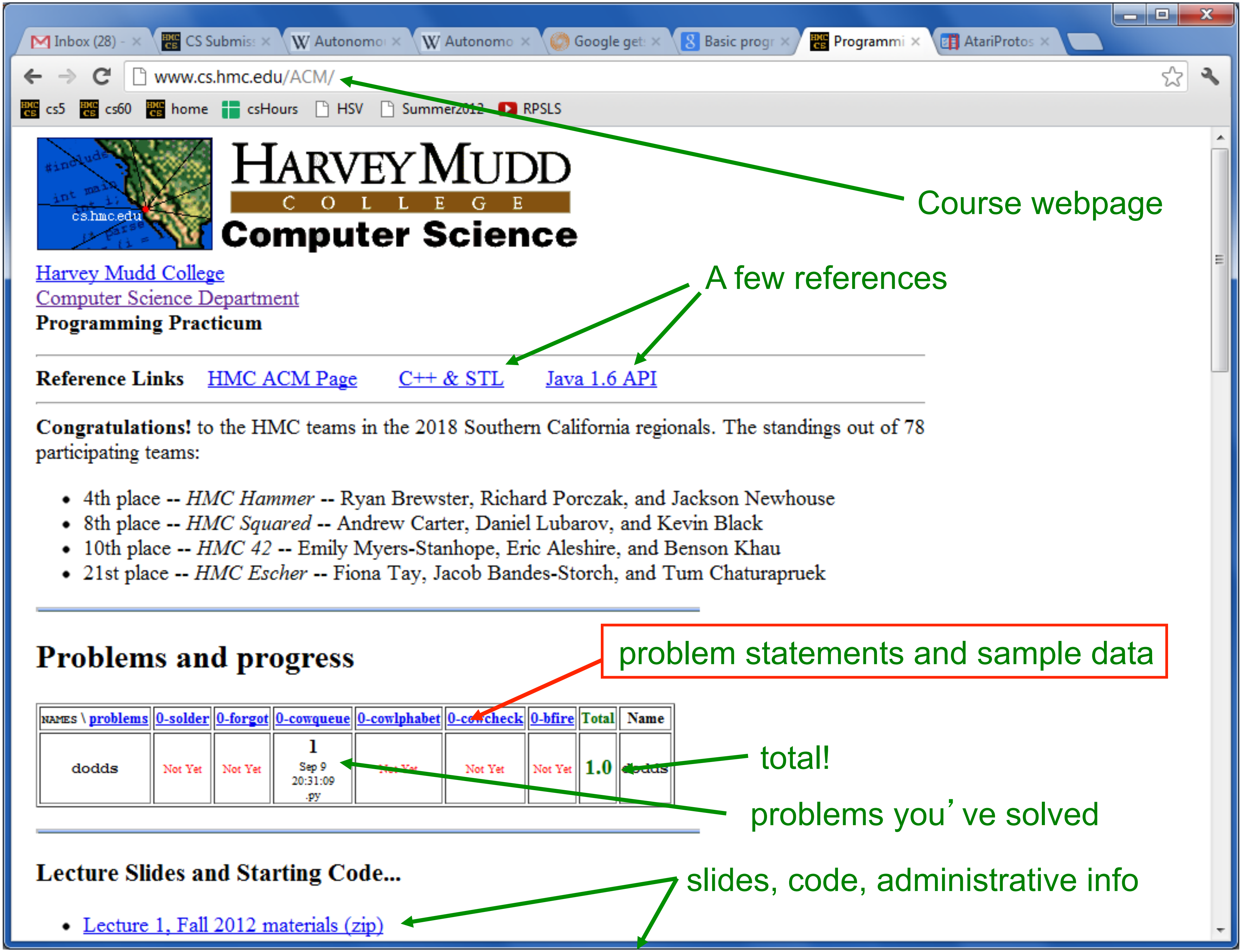
Benson!

Fluxx…


active
watching!


active
watching!

active
watching!



Harvey Mudd College
Computer Science Department
Programming Practicum

Reference Links HMC ACM Page C++ & STL Java 1.6 API

Congratulations! to the HMC teams in the 2018 Southern California regionals. The standings out of 78 participating teams:

- 4th place -- *HMC Hammer* -- Ryan Brewster, Richard Porczak, and Jackson Newhouse
- 8th place -- *HMC Squared* -- Andrew Carter, Daniel Lubarov, and Kevin Black
- 10th place -- *HMC 42* -- Emily Myers-Stanhope, Eric Aleshire, and Benson Khau
- 21st place -- *HMC Escher* -- Fiona Tay, Jacob Bandes-Storch, and Tum Chaturapruek

Problems and progress

NAMES \ problems	0-solder	0-forgot	0-cowqueue	0-cowlphabet	0-cowcheck	0-bfire	Total	Name
dodds	Not Yet	Not Yet	1 Sep 9 20:31:09 .py	Not Yet	Not Yet	Not Yet	1.0	dodds

Lecture Slides and Starting Code...

- Lecture 1, Fall 2012 materials (zip)

Course webpage

A few references

problem statements and sample data

total!

problems you' ve solved

slides, code, administrative info

Grading

CS 189 is graded by default ... (it' s possible to take it P/F, too)
though not for CS elective credit…

Coding Guidelines

- problems can be done *any time* during the semester
- discussion of algorithms always OK
- coding should be *within teams of 1-3*
- you may use any references *except* others' solutions or partial solutions…
- use **/cs/ACM/acmSubmit <file>** to submit on **knuth**

# Solved (out of 42)	Assessment
43+	pretty much impossible!
28-42	A
23-27	A-
20-22	B+
17-19	B
14-16	B-
9-13	C range
≤ 9	< D range or less

Details

Problems are worth 150% if

you can work in teams of up to 3 people

- You solve them during the 4:15 - 5:30 lab sessions
- … which extend to about ***11pm*** at night.

Language Choice?

Any ***reasonable*** language is OK; keep in mind that the ACM competition allows only Java, C, and C++.

Other "standard" languages for CS189 (so far):

C#, Python, Ruby, Perl, PHP, Haskell, Lua, Clojure, Lisp

additions will also be considered…

This week's problems

Notes, starting code, slides, etc. ...

- Lecture 1, cowqueue code examples (zip)
- Fall '13 Lecture 1 slides

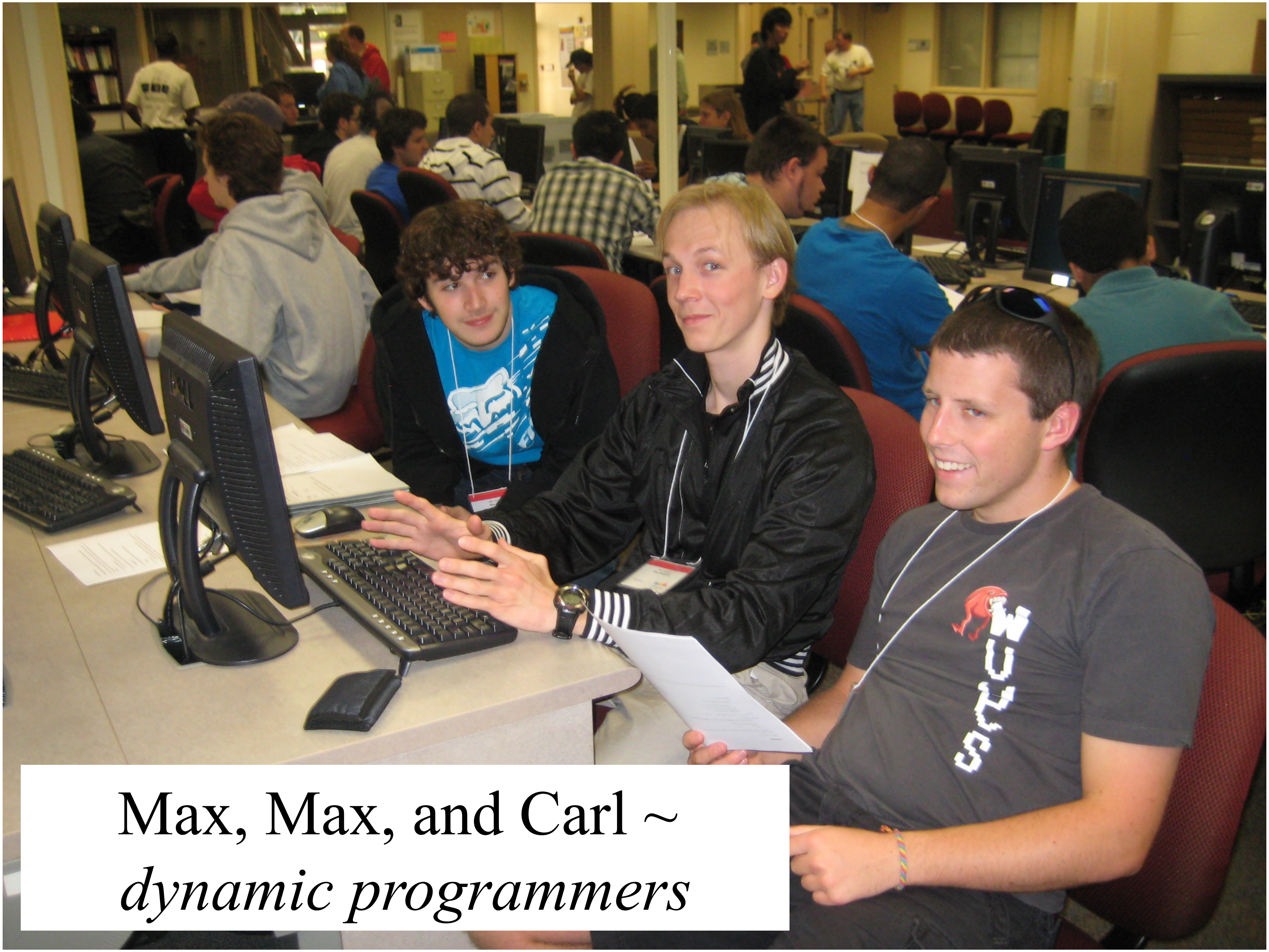
Problems and progress

NAMES \ problems	0-smount	0-lazy	0-elevator	0-cowqueue	0-cowcash	0-ave	Total	Name
dodds	Not Yet	Not Yet	Not Yet	1.5 Sep 9 16:19:24 .py	Not Yet	Not Yet	1.5	dodds

New to CS189? Start with this problem!

Part of the challenge is deciding *which* problem to tackle...

Some of this week's problems have a "dynamic programming" theme...



Max, Max, and Carl ~
dynamic programmers

Dynamic Programming

Many problems can be solved recursively...

... but with lots of ***repeated*** recursive calls!

These problems can be solved ***quickly*** with

(1) **Memoization**, or

(2) **Dynamic programming**

Idea: *just don't repeat the repeated calls!*

The `cowqueue` problem

Input

`ABACB`
`AABC`

Cow label sequence #1 (s1)

Cow label sequence #2 (s2)

Output

`3`

The number of the *longest common subsequence* bewteen s1 and s2.

In this case, the longest common subsequence is
`ABC` or **`AAB`**
though the problem doesn't require knowing these.

LCS problem

`s1 = "ABACB"` Input `s2 = "AABC"`

i1 i2

Output

LCS(i1, i2) = length of longest common subsequence of s1 up to i1 and s2 up to i2

Strategy

(1) Write a solution recursively.
(2) Then, don't make any call more than once!

LCS problem

`s1 = "ABACB"` Input `s2 = "AABC"`

i1 i2

length of longest common subsequence
of s1 up to i1 and s2 up to i2

LCS(i1, i2):

if s1[i1] == s2[i2]: return 1 + LCS(i1-1, i2-1)

if the same character, count it!

else: return max(LCS(i1-1, i2), LCS(i1, i2-1))

otherwise, lose both ends and take the better result

LCS code

`s1 = "ABACB"` Input `s2 = "AABC"`

i1 i2

cowqueue_recursive.py – /Users/zdodds/Desktop/cowqueue_recursive.py

```
import sys
sys.setrecursionlimit(100000)

def LCS( i1, i2 ):
 """ classic LCS """

 if i1 < 0 or i2 < 0: return 0

 if s1[i1] == s2[i2]:
     return 1 + LCS(i1 - 1, i2 - 1)
 else:
     return max(LCS(i1 - 1, i2), LCS(i1, i2 - 1))


if __name__ == "__main__":

 s1 = raw_input(); L1 = len(s1)
 s2 = raw_input(); L2 = len(s2)

 result = LCS( L1-1, L2-1 )

 print result
```

LCS idea

`s1 = "ABACB"` Input `s2 = "AABC"`

i1 i2

string2 s2[:i2]

string1 s1[:i1]

	⊘	A	AA	AAB	AABC
⊘					
A					
AB					
ABA					
ABAC					
ABACB					LCS(4,3)

LCS idea

`s1 = "ABACB"`   Input   `s2 = "AABC"`

i1   i2

string2 s2[:i2]

string1 s1[:i1]

	∅	**A**	**AA**	**AAB**	**AABC**
∅					
A					
AB					
ABA					
ABAC					LCS(3,3)
ABACB				LCS(4,2)	LCS(4,3)

LCS idea

`s1 = "ABACB"` Input `s2 = "AABC"`

i1 i2

string2 s2[:i2]

string1 s1[:i1]

	∅	A	AA	AAB	AABC
∅					
A					
AB					
ABA				LCS(2,2)	
ABAC			LCS(3,1)		LCS(3,3)
ABACB				LCS(4,2)	LCS(4,3)

LCS idea

`s1 = "ABACB"` Input `s2 = "AABC"`

i1 i2

string2 s2[:i2]

string1 s1[:i1]

	$\emptyset$	A	AA	AAB	AABC
$\emptyset$					
A					
AB				LCS(1,2)	
ABA			LCS(2,1)	LCS(2,2)	
ABAC		LCS(3,0)	LCS(3,1)		LCS(3,3)
ABACB				LCS(4,2)	LCS(4,3)

LCS idea

s1 = "ABACB" Input **s2 = "AABC"**

i1 i2

string2 s2[:i2]

string1 s1[:i1]

	⊘	A	AA	AAB	AABC
⊘	LCS(-1,-1)	LCS(-1,0)			
A		LCS(0,0)	LCS(0,1)		
AB	LCS(1,-1)	LCS(1,0)		LCS(1,2)	
ABA		LCS(2,0)	LCS(2,1)	LCS(2,2)	
ABAC	LCS(3,-1)	LCS(3,0)	LCS(3,1)		LCS(3,3)
ABACB				LCS(4,2)	LCS(4,3)

LCS idea

`s1 = "ABACB"` Input `s2 = "AABC"`

i1 i2

string2 s2[:i2]

string1 s1[:i1]

	∅	**A**	**AA**	**AAB**	**AABC**
∅	LCS(-1,-1)	LCS(-1,0)			
A		LCS(0,0)	LCS(0,1)		
AB	LCS(1,-1)	LCS(1,0)		LCS(1,2)	
ABA		LCS(2,0)	LCS(2,1)	LCS(2,2)	
ABAC	LCS(3,-1)	LCS(3,0)	LCS(3,1)		LCS(3,3)
ABACB				LCS(4,2)	LCS(4,3)

Collisions!

LCS, memoized

Put results in a dictionary.
Look up instead of recomputing.

```
# This is the "memoizing" dictionary of all distinct calls.
# Each distinct call is made only once and stored here.

D = {}

def LCS( i1, i2 ):
 """ classic LCS """

 if i1 < 0 or i2 < 0: return 0           # base cases

 if (i1,i2) in D: return D[ (i1,i2) ]   # already done!

 if s1[i1] == s2[i2]:
     result = 1 + LCS(i1-1, i2-1)
 else:
     result =  max( LCS(i1-1, i2), LCS(i1, i2-1) )

 D[ (i1,i2) ] = result                   # memo-ize it!

 return result                           # before returning


if __name__ == "__main__":

 s1 = raw_input(); L1 = len(s1)
 s2 = raw_input(); L2 = len(s2)

 result = LCS( L1-1, L2-1 )

 print result
```

Python *function decorators*

```
import sys; sys.setrecursionlimit(100000)

class memoize:
  def __init__(self, function):
    self.function = function
    self.memoized = {}

  def __call__(self, *args):
    try:
      return self.memoized[args]
    except KeyError:
      self.memoized[args] = self.function(*args)
      return self.memoized[args]
```

Python's "function decorator" syntax!

```
@memoize
def LCS( i1, i2 ):    # slow, recursive f'n here
```

LCS, DP'ed

Compute the table of results, bottom-up!

`s1 = "ABACB"` Input `s2 = "AABC"`

i1 i2

string2 s2[:i2]

string1 s1[:i1]

	⊘	A	AA	AAB	AABC
⊘					
A					
AB					
ABA					
ABAC					
ABACB					

LCS, DP'ed

Compute the table of results, bottom-up!

```
s1 = "ABACB"
```

Input

```
s2 = "AABC"
```

i1

i2

string2 s2[:i2]

string1 s1[:i1]

	∅	A	AA	AAB	AABC
∅					
A					
AB					
ABA					
ABAC					
ABACB					

```
if __name__ == "__main__":

 s1 = raw_input(); L1 = len(s1)
 s2 = raw_input(); L2 = len(s2)

 DP = [ [0]*(L2+2) for i1 in range(L1+2) ]

 for i1 in range(L1):
   for i2 in range(L2):
     if s1[i1] == s2[i2]: DP[i1][i2] = 1 + DP[i1-1][i2-1]
     else: DP[i1][i2] = max( DP[i1][i2-1], DP[i1-1][i2] )

 result = DP[L1-1][L2-1]

 #for row in DP:
 #  print row

 print result
```

Jotto!

A word-guessing game similar to mastermind...

Sophs	JRs	SRs	POM-CMC-SCR-PTZ	*other*
diner ?	**diner ?**	**diner ?**	**diner ?**	**diner 2**

This term's first class to guess another's word earns 1 problem...

This term's last class to have its word guessed earns 1 problem...

Recent-past Jotto finale:

Sophs (Win)	Jrs	Srs	Others
icily 0	icily 0	icily 1	icily 1
strep 2	strep 2	strep 2	strep 1
spork 1	spork 3	spork 0	spork 0
spend 2	spend 2	spend 2	spend 2
peeps 2	peeps 1	peeps 2	peeps 1
furls 1	furls 1	furls 0	furls 1
Ghost 2	Ghost 1	Ghost 1	Ghost 0
Tanks 2	Tanks 1	Tanks 2	Tanks 1
Gecko 2	Gecko 1	Gecko 1	Gecko 1

Quine 5

Try 1-2 of these tonight!

Notes, starting code, slides, etc. ...

- Lecture 1, cowqueue code examples (zip)
- Fall '13 Lecture 1 slides

Problems and progress

NAMES \ problems	0-smount	0-lazy	0-elevator	0-cowqueue	0-cowcash	0-ave	Total	Name
dodds	Not Yet	Not Yet	Not Yet	1.5 Sep 9 16:19:24 .py	Not Yet	Not Yet	1.5	dodds

Poster time!