Welcome to CSCI 235

Tiziana Ligorio

Today's Plan

Welcome

Logistics (Rules of the game)

What is CSCI 235?

Why Software Engineering?

Acknowledgments

This course was designed with input from many great resources other than the required textbook

Many thanks for materials and inspiration to

Tiziana Ligorio
Simon Ayzman
Susan Epstein
Keith Schwarz
Ioannis Stamos
Stewart Weiss

Logistics
(The rules of the game)

Course Webpage

Syllabus, Programming Guidelines

Your first assignment.
MUST READ!!!

Linux Lab

Programming Projects

Gradescope

Communication and Help

Course Webpage

maryash.github.io/235/index.html

Visit regularly for:

- Tentative Schedule and Announcements
- Lecture Notes
- Programming Projects
- Study Questions

Programming Projects

7 **Programming Projects**

They build upon each other - **don't fall behind!!!**

All projects submitted on **Gradescope**

If you haven't done so already, login to **Gradescope ASAP**

MUST USE YOUR CORRECT EMPL ID

Projects due by 11pm on the due date

MUST READ: Programming Guidelines on course webpage

Gradescope

To be used for submission of ALL programming projects

Check your email and follow invitation instructions

Gadescope is not for Debugging!!!

Plagiarism

- Submitting to Gradescope does not mean no one will look at your code
- **We check for plagiarism** and **report ALL cases to the office of student affairs**
- Sadly, last semester we reported several cases

Communication and Help

Let us hear from you!

If you find a typo or mistake let me know!!!

If you don’t understand something ask!!!

In class, in lab session, or in tutoring session

If you have concerns about something other than course content, talk to me:

Email or by appointment on BB Collabrate

Questions Etiquette

Different prior exposure to the material

All questions are good questions!!!

Friendly and collegial environment - we are here to help!

Be Proactive if you want Help!

- If everyone show up for help on the project due date it will not be possible to give help to all!

- Please be proactive, start early and plan ahead!

Introducing
Lecture Activity

7% of final grade

Attendance and must show an adequate attempt

Work out a new problem 

What is CSCI 235?

Programming => Software Analysis and Design

Expected professional and responsible conduct

Think like a Computer Scientist:

Design and maintain complex programs

- Software Engineering, Abstraction, OOP

Design and represent data and its management

- Abstract Data Types

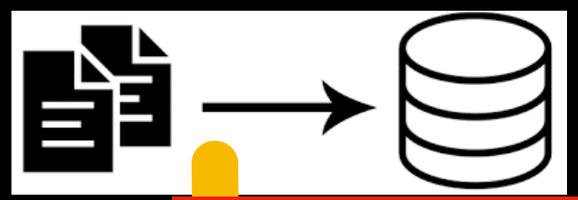
Implement data representation and operations

- Data Structures
- Algorithms

Understand Algorithm Efficiency

“It’s all just bits and bytes…”


http://



STRUCTURE


1010101010101010
0101010101010101
1001100110011001

Increasing software size

Society keeps digitizing more aspects of life

Software keeps getting bigger

Size and interaction of software systems ever increasing

Exciting!!!

Daunting for software engineers

Typical software size / lines of code

~1 - 10	Hello world
~100	Most STL queue implementations
~1,000	Typical Computer Science curriculum term project
~10,000	Intensive team project
~100,000	Most Linux command line utilities
~1,000,000	Linux g++ compiler
~10,000,000	Mozilla Firefox
~50,000,000	Microsoft's Windows
~2,000,000,000	Google (search, maps, docs, gmail, …)

Illustrative example, may not be up to date

Problems with large software

Every bit counts!

A single incorrect bit may result in:

- negative instead of positive int
- pointer past the end of an array
- unsorted rather than sorted vector
- …

Program performs unexpectedly

Problems with large software

Every bit counts!

A single incorrect bit may result in:

- negative instead of positive int
- pointer past the end of an array
- unsorted rather than sorted vector
- …

Program performs unexpectedly

Problems with large software

Two lines of code **interact** if they manipulate same data

```
int x = 5;    // if I change the x to my_var
cout << x;    // I must change it here too
```

Assume n lines of code

Any line may interact with any number of other lines

$n(n-1)/2 = (n^2-n)/2$ possible interactions

With 10 lines of code there may be

45 interactions

Unlikely but it gives you an idea of how bad it can get

Problems with large software

Assume line 1 shares same data with lines 6 and 23.

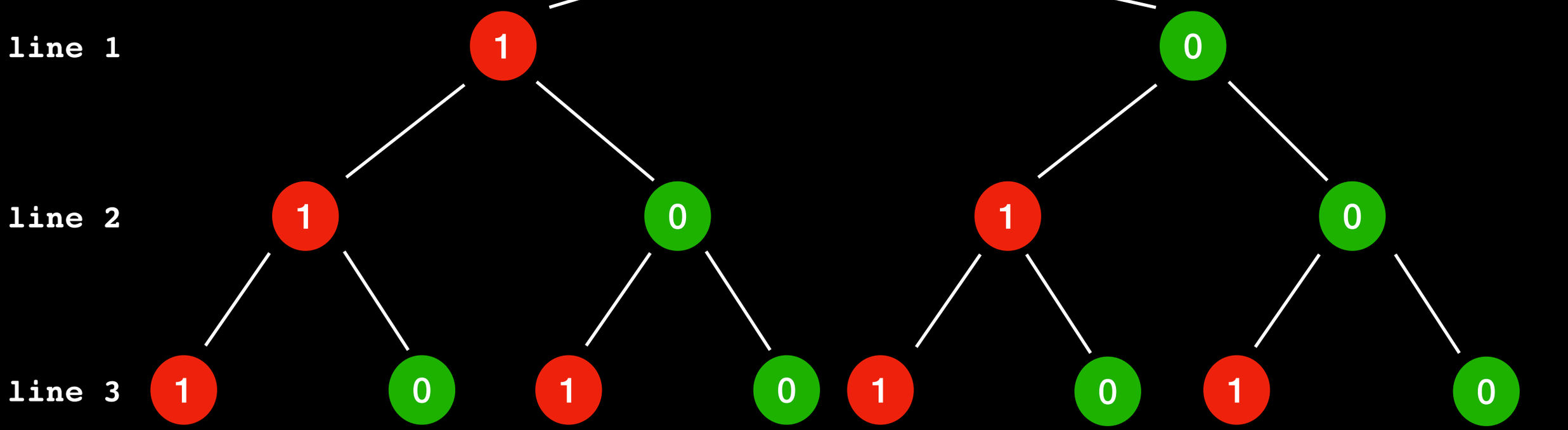
So in actuality the three lines all together form an interaction

If we think of subsets of lines of code interacting (sharing the same data) . . .

How many possible subsets?

Again unlikely all possible subsets will interact but it gives you an idea of why you'd want to control it

for n lines of code



2^3

line n ... 2^n

Every path down the tree is an interaction among one possible subset of lines of code

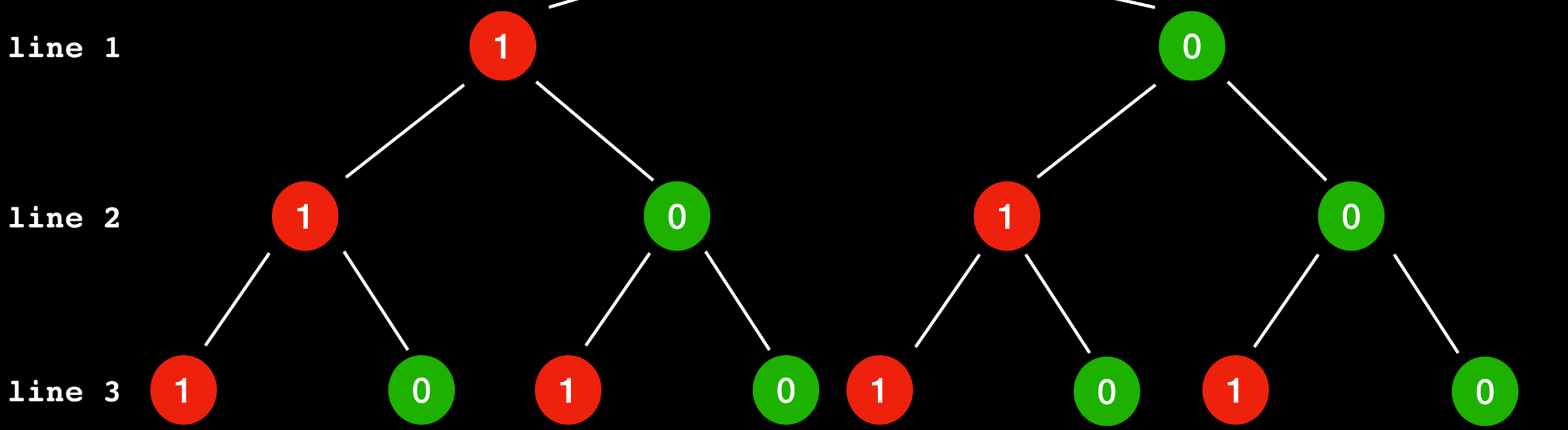
for n lines of code

line 1

line 2

line 3

line n



2^3

. . . 2^n

Every path down the tree is an interaction among one possible subset of lines of code

Lecture Activity

Draw a **very small** square on the leftmost bottom corner

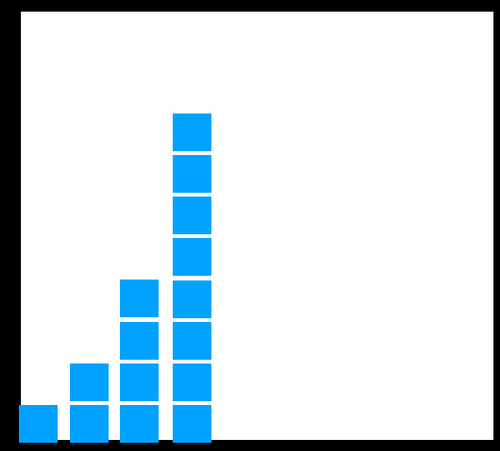
Next to it, double it (2 squares one on top of other)

Next to it, double it (4 squares one on top of other)

Next to it double it , …

Keep going…

How quickly do you run off the top of the page?



Problems with large software

How folding paper can get you to the moon:

https://www.youtube.com/watch?v=AmFMJC45f1Q

Watch this home if we don’t have time at the end of lecture

Problems with large software

How do you go about modifying code with many interactions?

Larger software has greater likelihood of error

More difficult to debug and modify

Control software interaction!!!

Software with **many interactions is bad**!!!!

Write small units of code

Minimize Interactions/Coupling!!!

Enforce strict rules on how code interacts

How? We will consider:

- Principles of Software Engineering
- Object Oriented Programming

What is Software Engineering?

"The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software"

IEEE *Standard Glossary of Software Engineering Terminology*

Big Ideas of Software Engineering

Modularity

Modifiability/Extensibility

Ease of Use

Fail-Safe Programming

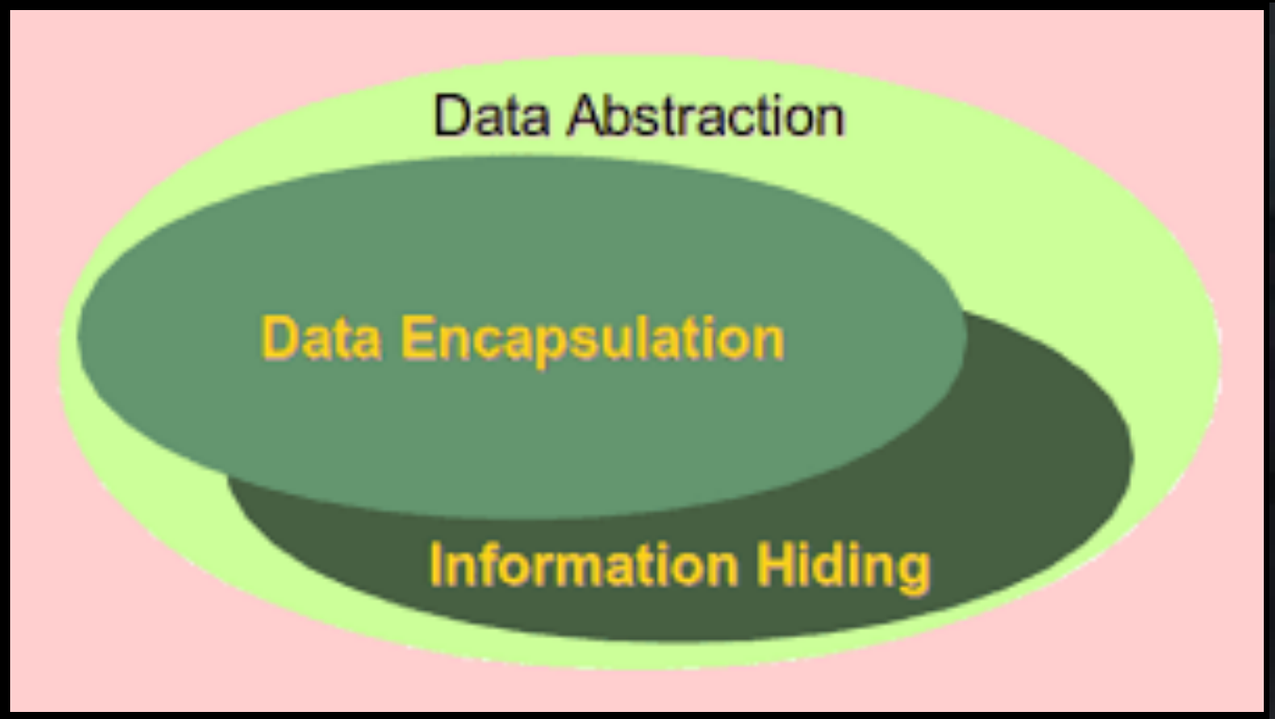
Debugging

Testing

We will come back to these throughout the course

APPENDIX B

Object Oriented Programming



Next Time

Abstraction and OOP