02-201, Fall 2015, Carl Kingsford

Lecture 0: Introduction to the class

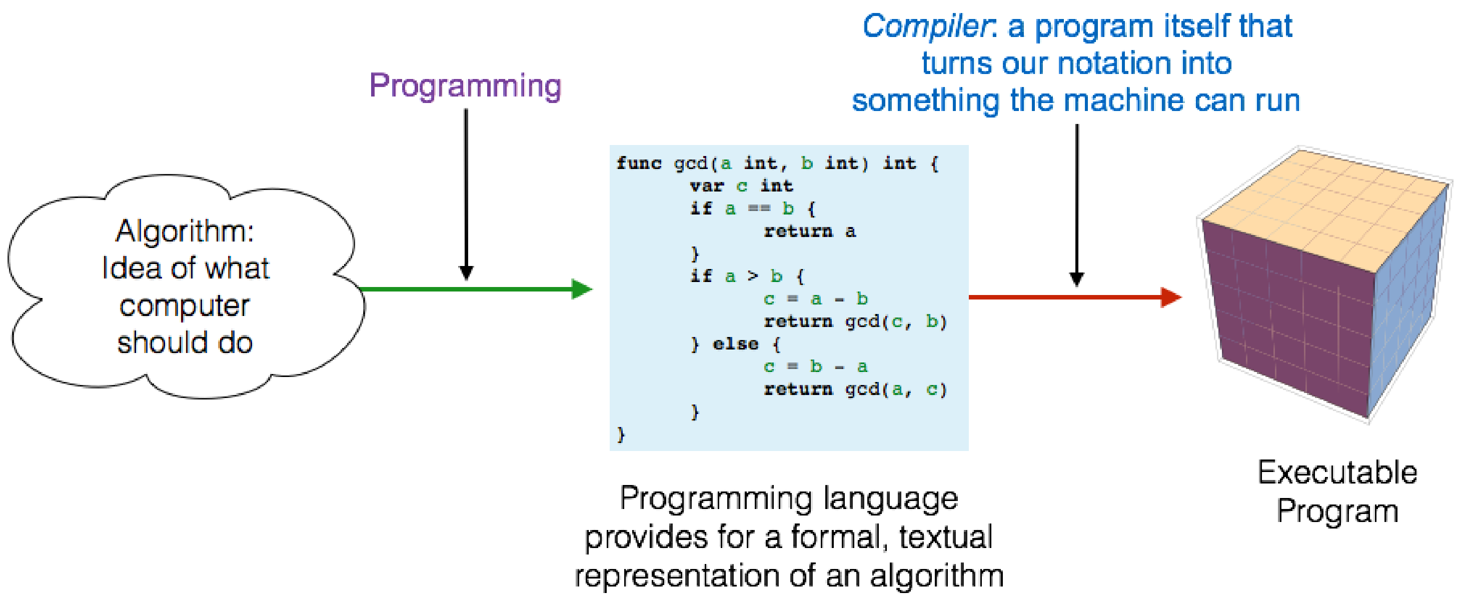
1. What is programming?

Programming is clearly, correctly telling a computer what to do.

A computer is a very complex, simple machine:

- It's complex because it is made up of millions or billions of parts all engineered to work together
- It's simple because it can only do relatively simple operations in exactly the way we tell it

Programming is the way we control this machine.


Programming
Compiler: a program itself that turns our notation into something the machine can run
Algorithm: Idea of what computer should do
Executable Program
Programming language provides for a formal, textual representation of an algorithm

```
func gcd(a int, b int) int {
        var c int
        if a == b {
                return a
        }
        if a > b {
                c = a - b
                return gcd(c, b)
        } else {
                c = b - a
                return gcd(a, c)
        }
}
```

> Example: Computing $\sum_{i=0}^{n} \frac{1}{i^2}$ with a Go program at: http://play.golang.org

Learning to program is like learning a new language

We write programs in "programming languages", which have rules of spelling, grammar, and style.

Ceci n'est pas une pipe -> SUBJECT ne VERB pas DIRECT_OBJECT

Programming is like writing: you are striving to express what you want the computer to do as clearly, correctly, and concisely as possible. Both your English writing and your programming should be precise and concise.

Just as with writing, you write, revise, edit, refine in a quest for correctness, clarity, and style.

We have to start by learning the "grammar" and "spelling" rules of our progamming language.

Programming is a notation system for describing processes.

Calculus is a notation system for describing certain types of mathematics.

$$f(t) = \int_0^t \frac{1}{\ln(t)} dt$$

There's a lot of notation here that you had to learn and that now probably seems invisible to you:

- "f(t)" denotes a function application on variable t.
- The "=" sign means you are giving a definition for function f: what its value is for parameter t.
- "ln(t)" denotes calling a particular function (the natural logarithm).
- The $\frac{x}{y}$ notation means division.
- The $\int_0^t \dots dt$ means to take integrate over t (the variable in the denominator) from from 0 to t (the variable passed into t).

There are some subtleties to this notation: on the righthand side, the 3 occurrences of t mean 3 different things. The "dt" defines a variable over which you are integrating. That variable appears inside the ln(t). Finally, the t in the range of the integral is the t defined by the parameter to funciton f.

You had to learn all of this for calculus. Programming involves learning a different notation to describe things a computer can do:

```go
func gcd(a int, b int) int {
    var c int
    if a == b {
        return a
    }
    if a > b {
        c = a - b
        return gcd(c, b)
    } else {
        c = b - a
        return gcd(a, c)
    }
}
```

You don't need to understand this yet, but it specifies Euclid's algorithm for computing the greatest common divisor.

A fairly large part of this course is mastering programming notation.

Programming a way of thinking clearly about processes.

Programming is much more than the notation. In fact, there have been thousands of different notations invented.

But most of these notations all involve the same concepts:

- **Making choices:** Do A if B otherwise to C.
  (Go to the store if it's raining, otherwise go running.)
- **Repeating things:** Do A 100 times. Do A until B.
  (Run around the track until you are tired.)
- **Definining new operations in terms of existing ones:** When I say "Do A", you should do B,C,D,E.
  (When I say "Go to the store", you should "drive there, and buy cookies, milk, cake, and chocolate.")

A large part of this class will be learning to think about these kinds of concepts.

Programming is a craft.

Programming is like carpentry: it takes practice and skill to do it right.

As with cabinets, there are good programs and bad programs. A goal of this course is for you to learn to write good programs.

Programming is great practice for thinking logically and linearly.

Writing a program requires you to think carefully about a problem in a "linear way" (A then B then C, etc.) rather than vague intuition.

Writing a program is like solving a sudoku puzzle — programming tests (and builds!) your powers of concentration and logical thinking — but programming is better than sudoku since at the end you have something useful instead of a square of numbers. Programming also pays better as a career that being a sudoku solver.

Programming is fun.

By the end of this class, you will be able to get a computer to draw pictures, answer mathematical questions, design protein structures, analyze English text, and do many other tasks.

Specifically, the course projects will include programs to:

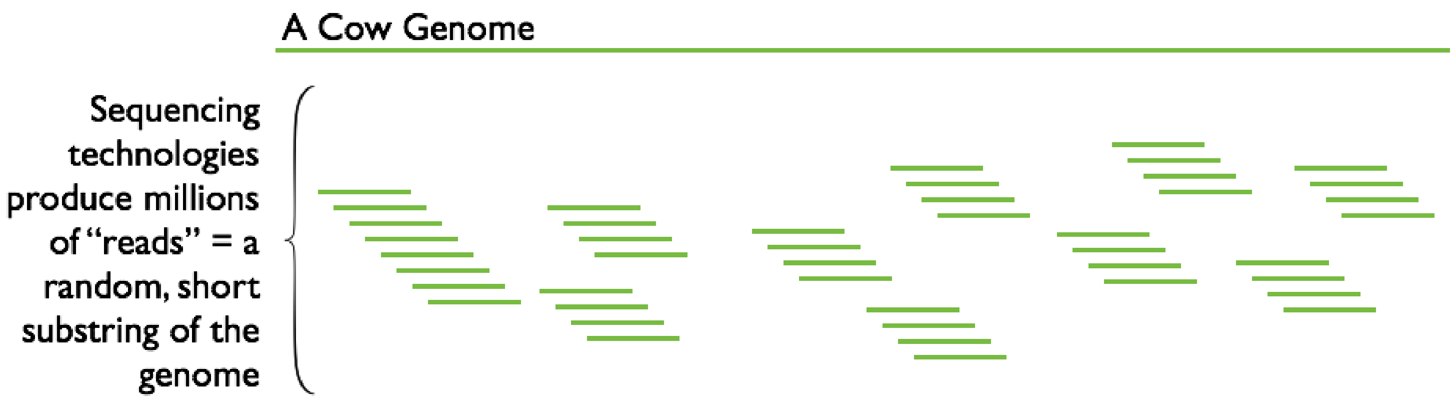
- Model the growth of a population
- Model diffusion over a surface
- Run a simple model of the universe
- Simulate evolution of competing species
- Simulate English text and DNA
- Draw plots of data
- Perform simple encryption
- Design protein structures

All by just writing some letters into a text file.

It's the ultimate virtual lego system: by writing code, you put together virtual machines.

2. The use of computers in science

Genome sequencing:

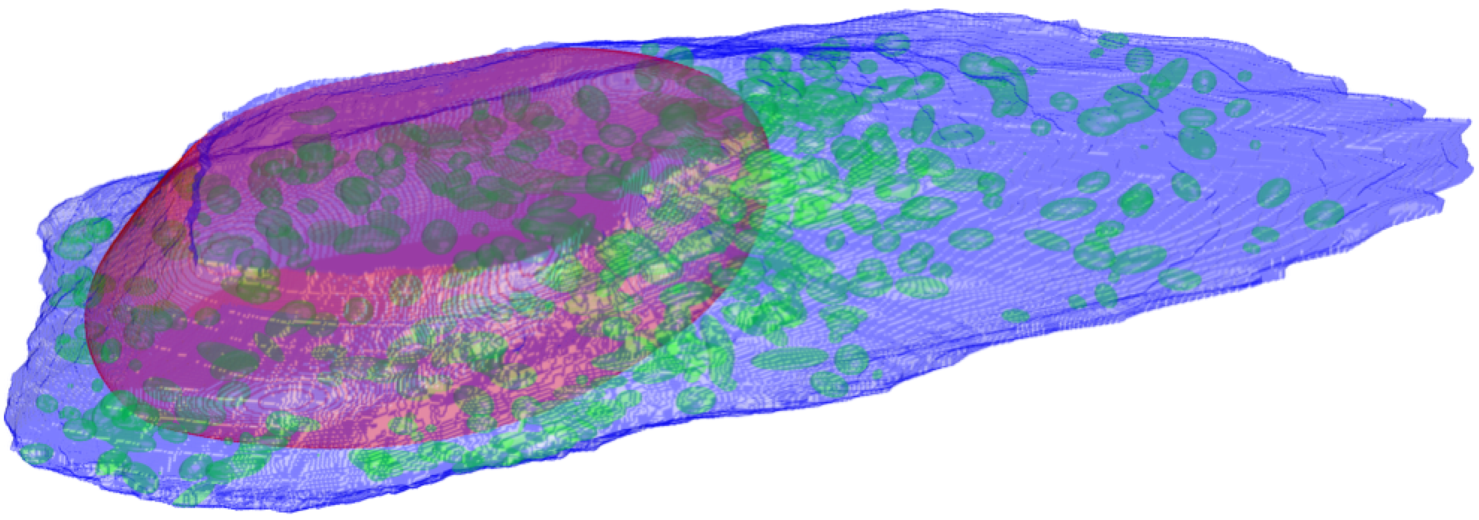

A Cow Genome
Sequencing technologies produce millions of "reads" = a random, short substring of the genome

- DNA sequencing technologies produce millions of short (30-200 nucleotide) reads of a genome. Computers are needed to piece those segments into a complete genome.

Modelling spatial relationships between organelles:

CellOrganizer analyzes images of cells to produce models of:

- cell shape
- nuclear shape
- chromatin texture
- vesicular organelle size, shape and position
- microtubule distribution

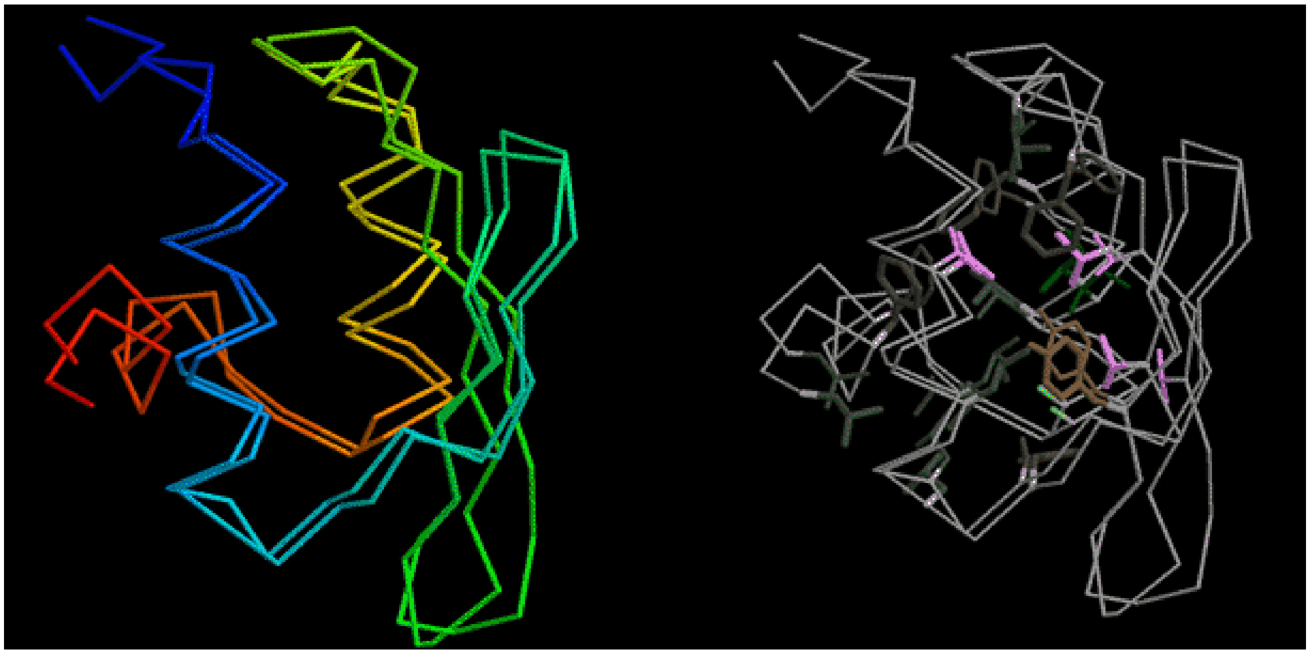


(http://www.cellorganizer.org)

Predicting the shape of proteins:

A protein is a linear change of amino acids. Typically, the sequence of amino acids is encoded in a gene. The chain folds up into some functional shape.

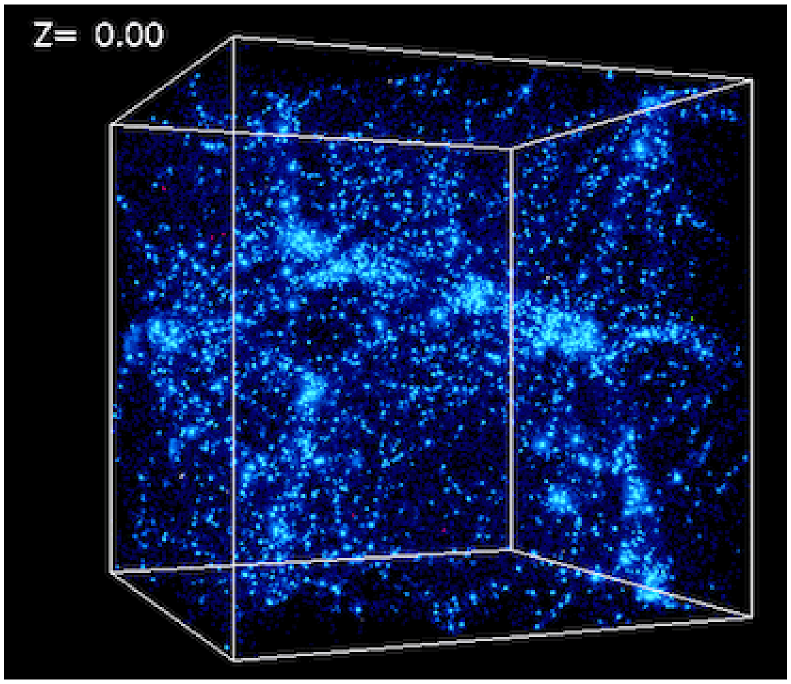
Problem: predict the shape of a protein from its amino acid sequence. The two lines show the real and computationally predicted shape of a particular protein:



(Image by Phil Bradley from https://boinc.bakerlab.org/rah_hires_prediction.php)

Modeling the universe:

Understanding the large-scale structure of the universe and how dark matter and dark energy affect its evolution requires simulations.



(image from http://cosmicweb.uchicago.edu/sims.html).

Goals of the course

1. Master the "Go" notation for programming languges.
2. Master the basic building blocks of algorithms (loops, conditionals, functions).
3. Understand what is going on inside the computer at a working level.
4. Practice writing semi-large programs that do real tasks.
5. Learn how to use programming tools to help you write good programs.
6. See how computation drives scientific progress.

7. See how much fun programming can be!

If you successfully complete the projects of this course, you should be comfortable writing your own programs and perhaps even be able to contribute to a research group's projects. This skill will make you more valuable for internships and in-semester research projects, and it will really help you in later classes.

3. Administration of the course

For a fuller discussion of these and other points, please read the syllabus. Below are some highlights of the syllabus.

Course work:

- **10-15 homework assignments**. (50% of your grade.) These are chosen to be challenging but doable. This is where most of the work and learning should happen! Most of the homeworks will be programming assignments where you will turn in a working program, but there will be several handwritten problem sets.

- **Several quizzes and participation**. (10% of your grade.) Quizzes will be short, announced ahead of time, and are intended so that both the teaching staff and you can gauge where your understanding is lacking and where it is strong.

- **Two midterms and a final**. (40% of your grade in total: 10% for each midterm, and 20% for the final.) These are used to test your knowledge both of programming and non-programming concepts we will cover.

Other policies:

- Laptop use is encouraged! If you are following along with the examples we are working or trying out programming concepts.

- Laptop use for other purposes is **forbidden**. Texting during class is **not allowed**.

- You must submit all assignments on time. There are no late assignments accepted. You should turn in what you have by the deadline, and you will get partial credit. We will drop the lowest assignment grade.

- Quizes cannot be made up if you miss them, except in the case of documented illness or emergency. We will drop the lowest quiz grade.

Academic honesty:

Unless specified otherwise, you must work on all assignments on your own.

- You may discuss programming assignments with classmates in a general way. However, you **must not share or show or see the code of your classmates**. You must write your own code entirely. You can post general coding questions (with code snippets) on the discussion board.

- You may **never** use, look at, study, or copy any answers from previous semesters of this course.

4. Summary

- Programming is a way of telling the computer what to do.
- It is fun and good practice for thinking logically.
- It is central to almost all aspects of modern science.

5. Glossary

- algorithm: a formal specification of a process that is (typically) executed by a computer.
- programming language: a particular notation system for specifying algorithms.
- compiler: a program that reads programming notation and turns it into something the computer can run.