CSCI 334:
Principles of Programming Languages

Lecture 22-1: Wrap up

Instructor: Dan Barowy

Williams

Topics

Why OO matters

Turing tarpits

Why PL matters

How to give a good talk

SCS

Your to-dos

1. Project checkpoint #3, “mostly working,” **due Sunday 12/11**.
2. Final project due **Sunday 12/18**.

Announcements

Amy Babay, University of Pittsburgh

Friday, Dec 9 @ 2:35pm (last colloquium of 2022!)
Computer Science Colloquium – Wege TCL 123
Toward Intrusion-Tolerant Critical Infrastructure

As critical infrastructure systems are becoming increasingly exposed to malicious attacks, it is crucial to ensure that they can withstand sophisticated attacks while continuing to operate correctly and at their expected level of performance.

In this talk, I will present our work on making intrusion-tolerant critical infrastructure systems possible and practical. I will start by discussing our Spire system, the first Supervisory Control and Data Acquisition (SCADA) system for the power grid that is resilient to both system-level compromises and sophisticated network-level attacks.

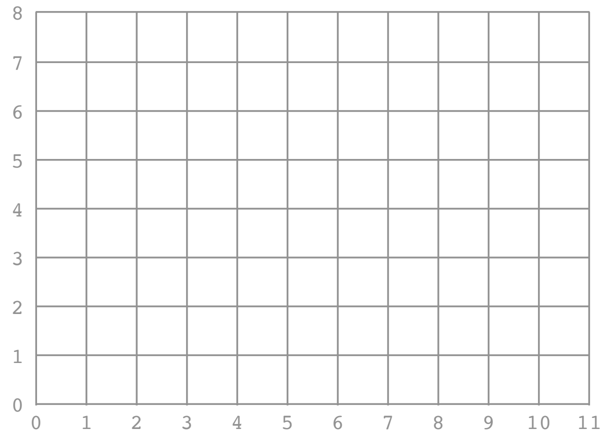
Then, I will present our recent work offering a practical deployment path for Spire and similar BFT-based systems through a new model for “intrusion tolerance as a service”. The intrusion-tolerance-as-a-service model enables critical infrastructure operators to gain the resilience benefits of intrusion tolerance, while offloading significant parts of the system management to a service provider. Critically for practical acceptance, our work shows how these benefits can be achieved without requiring critical infrastructure operators to expose confidential or proprietary data and algorithms to the service provider.

Ingalls Test for Extensibility

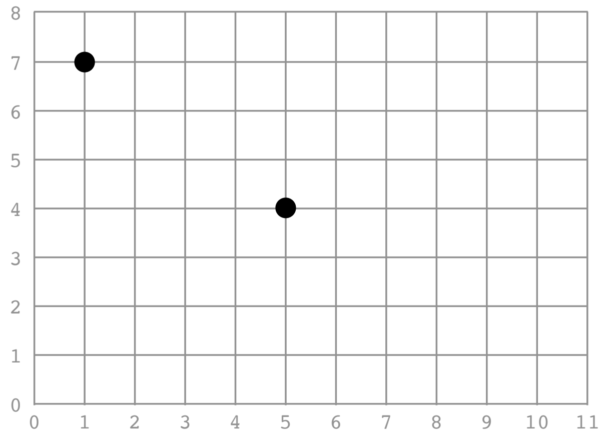
i.e., the “rectangle test”

- The test is about the ability to extend software *after* it has already been designed and written.

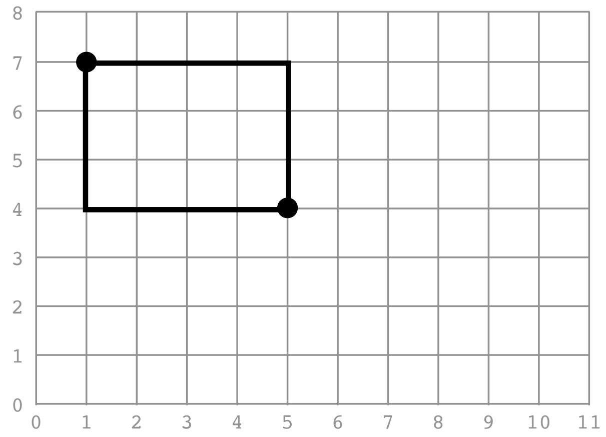
Ingalls Test for Extensibility

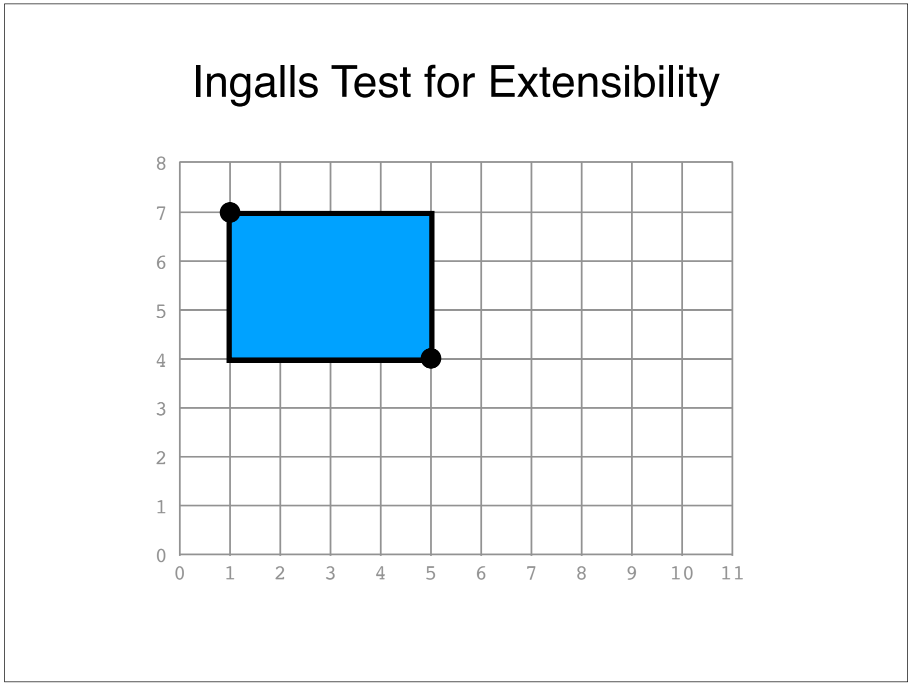


Ingalls Test for Extensibility

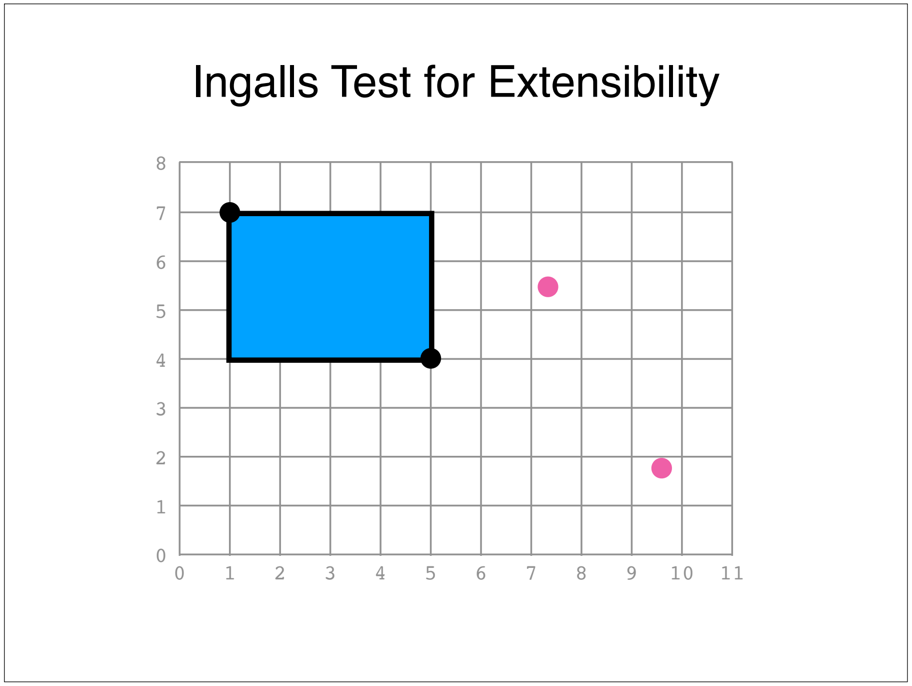


Ingalls Test for Extensibility

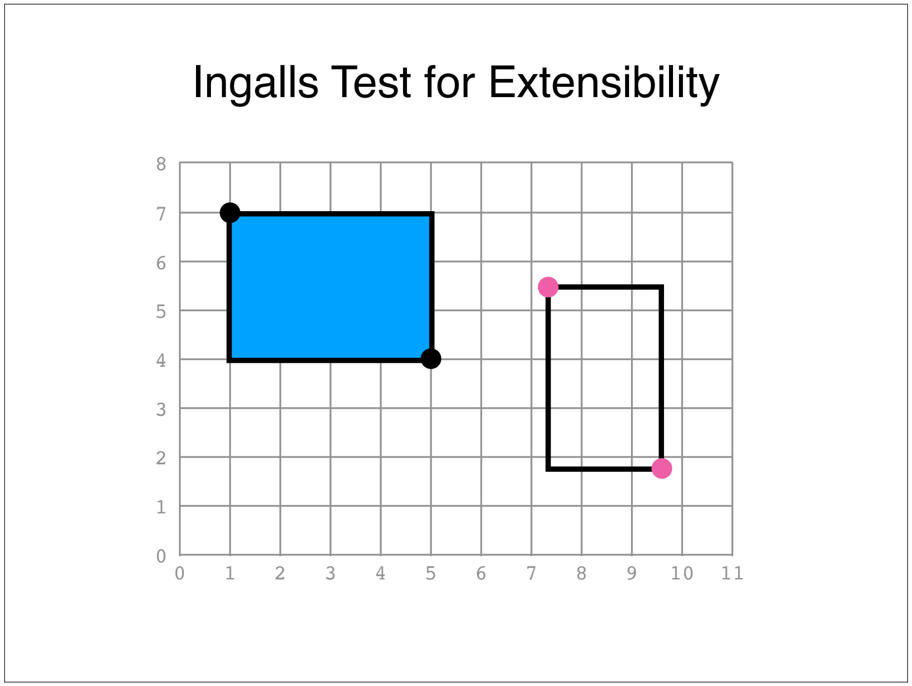




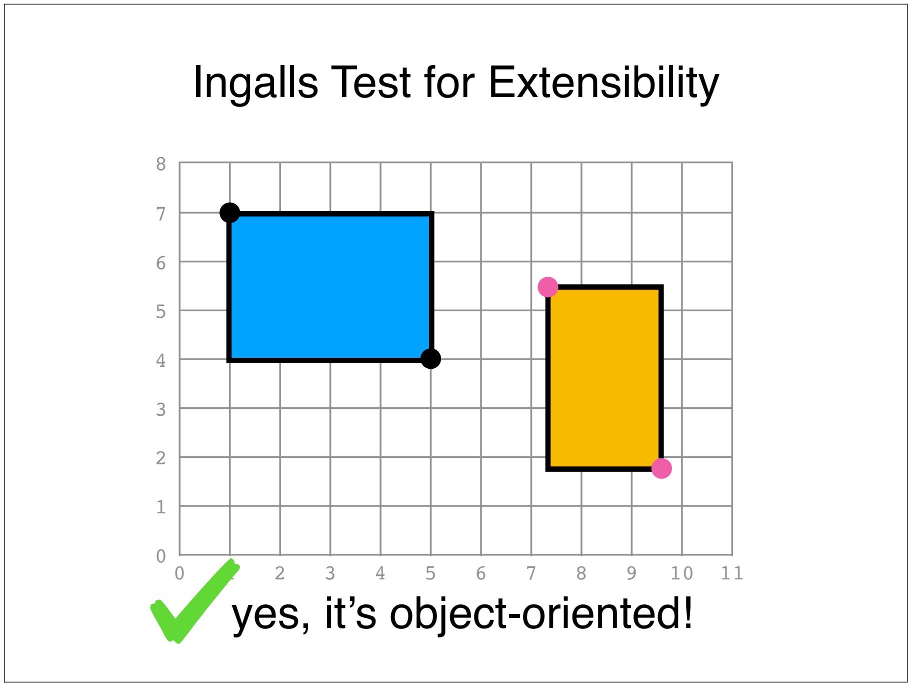
Ingalls Test for Extensibility



Ingalls Test for Extensibility



Ingalls Test for Extensibility



Ingalls Test for Extensibility

✓ yes, it’s object-oriented!

Java, Python, etc. pass the rectangle test

Which language do I use?

Turing Tarpit

A **Turing tarpit** is a programming language flexible enough to do anything (i.e., it is **Turing equivalent**) while also being **difficult to learn and use** for everyday tasks.

“Beware of the Turing tar-pit in which everything is possible but nothing of interest is easy.” —Alan Perlis

Examples:

- Turing machines
- The Lambda Calculus
- Breph
- C?

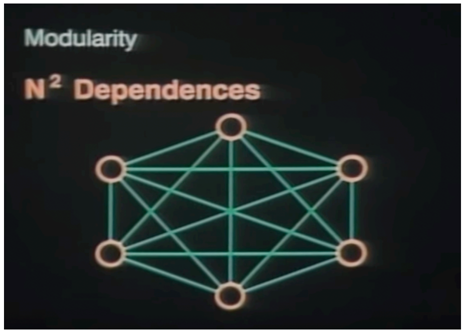
The right choice depends on the problem

- OO offers a **different kind of extensibility** than functional (or function-oriented) languages.
- Suppose you’re **modeling a hospital**.

Operation	Doctor	Nurse	Orderly
`Print`	`Print Doctor`	`Print Nurse`	`Print Orderly`
`Pay`	`Pay Doctor`	`Pay Nurse`	`Pay Orderly`

- FP makes it **easy to add operations** (**rows** above).
- OOP makes it **easy to add data** (**columns** above).

Why I like OO



OO is fundamentally based on **the idea that people matter** in the design of a programming language.

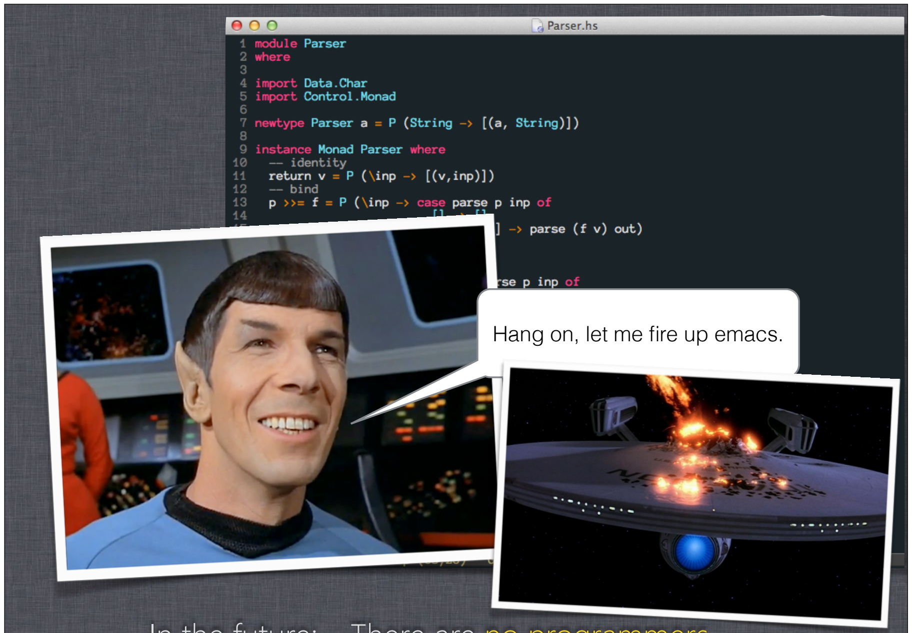
How do we **minimize human effort** while designing large pieces of software?

Programming of the People, by the People, and for the People

Daniel Barowy

UMass Amherst

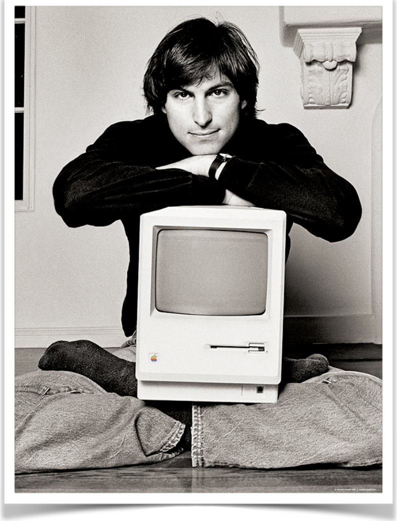
Williams College, January 9, 2017



```
module Parser
where

import Data.Char
import Control.Monad

newtype Parser a = P (String -> [(a, String)])

instance Monad Parser where
  -- identity
  return v = P (\inp -> [(v,inp)])
  -- bind
  p >>= f = P (\inp -> case parse p inp of

                                  ] -> parse (f v) out)



                                se p inp of
```

In the future: There are no programmers.

A Bicycle for the Mind



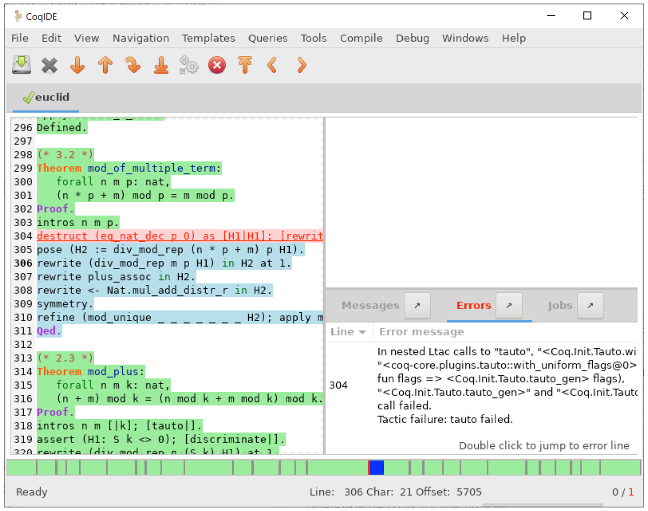
This work is not done yet.

Some things to think about

Would it be better to

- Have a programming language where **bugs are impossible**, but **programming is difficult**, or
- have a programming language where **bugs are possible** but their **consequences are minimized**?

Coq

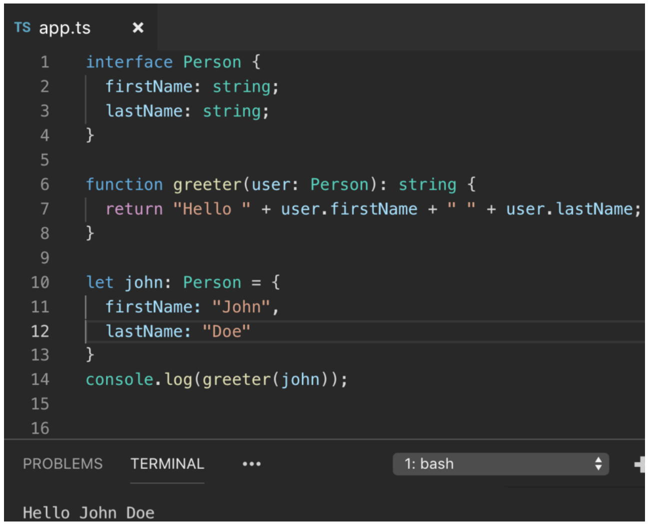


```
296 Defined.
297
298 (* 3.2 *)
299 Theorem mod_of_multiple_term:
300    forall n m p: nat,
301    (n * p + m) mod p = m mod p.
302 Proof.
303 intros n m p.
304 destruct (eq_nat_dec p 0) as [H1|H1]; [rewrit
305 pose (H2 := div_mod_rep (n * p + m) p H1).
306 rewrite (div_mod_rep m p H1) in H2 at 1.
307 rewrite plus_assoc in H2.
308 rewrite <- Nat.mul_add_distr_r in H2.
309 symmetry.
310 refine (mod_unique _ _ _ _ _ _ _ _ H2); apply m
311 Qed.
312
313 (* 2.3 *)
314 Theorem mod_plus:
315    forall n m k: nat,
316    (n + m) mod k = (n mod k + m mod k) mod k.
317 Proof.
318 intros n m [|k]; [tauto|].
319 assert (H1: S k <> 0); [discriminate|].
320 rewrite (div_mod_rep n (S k) H1) at 1.
```

```
304   In nested Ltac calls to "tauto", "<Coq.Init.Tauto.wi
      "<coq-core.plugins.tauto::with_uniform_flags@0>
      fun flags => <Coq.Init.Tauto.tauto_gen> flags),
      "<Coq.Init.Tauto.tauto_gen>" and "<Coq.Init.Tautc
      call failed.
      Tactic failure: tauto failed.
```

Double click to jump to error line

TypeScript



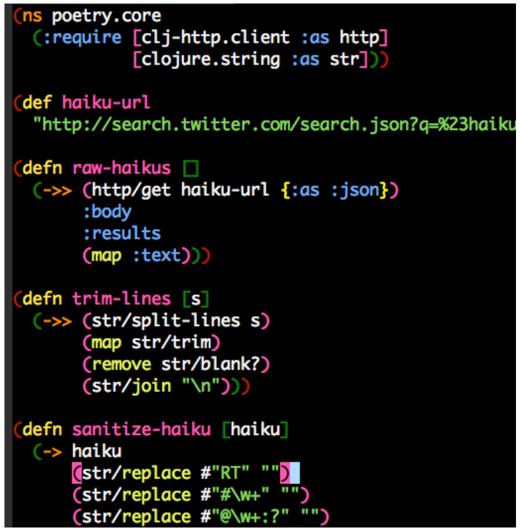
```typescript
interface Person {
  firstName: string;
  lastName: string;
}

function greeter(user: Person): string {
  return "Hello " + user.firstName + " " + user.lastName;
}

let john: Person = {
  firstName: "John",
  lastName: "Doe"
}
console.log(greeter(john));


```

```
Hello John Doe
```

Would it be better to

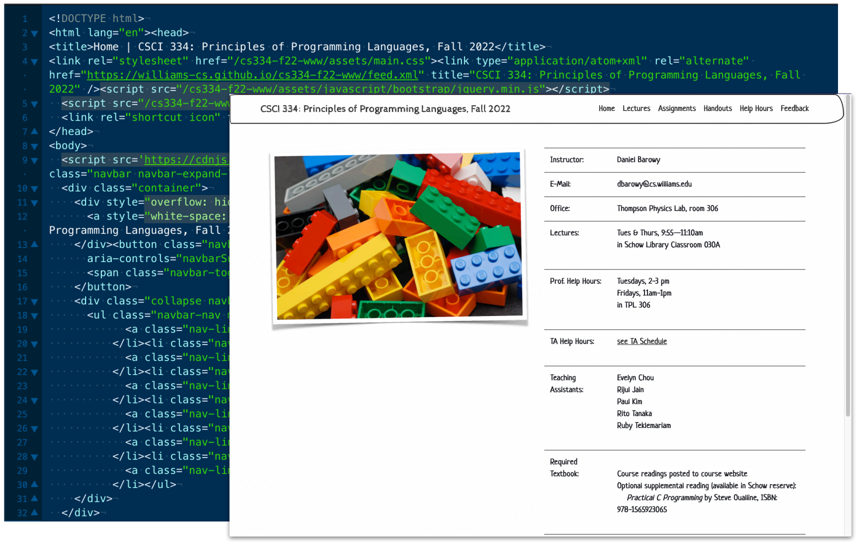
- Have **one language to rule them all**, or
- have many **different, small special-purpose languages**?

Clojure



```clojure
(ns poetry.core
  (:require [clj-http.client :as http]
            [clojure.string :as str]))

(def haiku-url
  "http://search.twitter.com/search.json?q=%23haiku

(defn raw-haikus []
  (->> (http/get haiku-url {:as :json})
       :body
       :results
       (map :text)))

(defn trim-lines [s]
  (->> (str/split-lines s)
       (map str/trim)
       (remove str/blank?)
       (str/join "\n")))

(defn sanitize-haiku [haiku]
  (-> haiku
      (str/replace #"RT" "")
      (str/replace #"#\w+" "")
      (str/replace #"@\w+:?" "")
```

HTML



```html
<!DOCTYPE html>
<html lang="en"><head>
<title>Home | CSCI 334: Principles of Programming Languages, Fall 2022</title>
<link rel="stylesheet" href="/cs334-f22-www/assets/main.css"><link type="application/atom+xml" rel="alternate"
href="https://williams-cs.github.io/cs334-f22-www/feed.xml" title="CSCI 334: Principles of Programming Languages, Fall
2022" /><script src="/cs334-f22-www/assets/javascript/bootstrap/jquery.min.js"></script>
  <script src="/cs334-f22-ww
  <link rel="shortcut icon"
</head>
<body>
  <script src='https://cdnjs.
class="navbar navbar-expand-
  <div class="container">
    <div style="overflow: hi
      <a style="white-space:
Programming Languages, Fall 2
    </div><button class="navb
      aria-controls="navbarSu
      <span class="navbar-tog
    </button>
    <div class="collapse navb
      <ul class="navbar-nav m
            <a class="nav-li
          </li><li class="nav
            <a class="nav-li
          </li><li class="nav
            <a class="nav-li
          </li><li class="nav
            <a class="nav-li
          </li><li class="nav
            <a class="nav-li
          </li><li class="nav
            <a class="nav-li
          </li></ul>
    </div>
  </div>
```

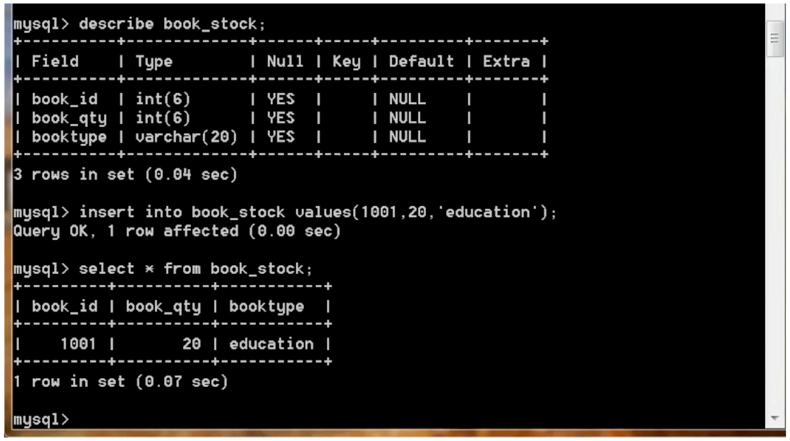
CSCI 334: Principles of Programming Languages, Fall 2022

Instructor:	Daniel Barowy
E-Mail:	dbarowy@cs.williams.edu
Office:	Thompson Physics Lab, room 306
Lectures:	Tues & Thurs, 9:55—11:10am in Schow Library Classroom 030A
Prof. Help Hours:	Tuesdays, 2-3 pm Fridays, 11am-1pm in TPL 306
TA Help Hours:	see TA Schedule
Teaching Assistants:	Evelyn Chou Rijul Jain Paul Kim Rito Tanaka Ruby Teklemariam
Required Textbook:	Course readings posted to course website Optional supplemental reading (available in Schow reserve): *Practical C Programming* by Steve Oualline, ISBN: 978-1565923065

And if “many languages” is the answer, would it be better to

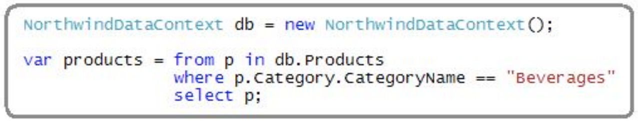
- Have many **standalone languages**, or
- have many languages that **can be embedded in a host language**?

SQL



```
mysql> describe book_stock;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| book_id   | int(6)      | YES  |     | NULL    |       |
| book_qty  | int(6)      | YES  |     | NULL    |       |
| booktype  | varchar(20) | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
3 rows in set (0.04 sec)

mysql> insert into book_stock values(1001,20,'education');
Query OK, 1 row affected (0.00 sec)

mysql> select * from book_stock;
+---------+----------+-----------+
| book_id | book_qty | booktype  |
+---------+----------+-----------+
|    1001 |       20 | education |
+---------+----------+-----------+
1 row in set (0.07 sec)

mysql>
```

LINQ

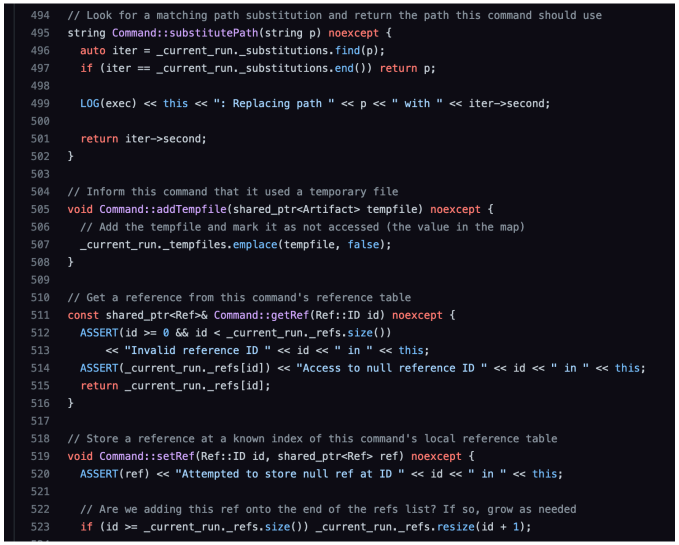


```
NorthwindDataContext db = new NorthwindDataContext();

var products = from p in db.Products
               where p.Category.CategoryName == "Beverages"
               select p;
```

Would it be better to

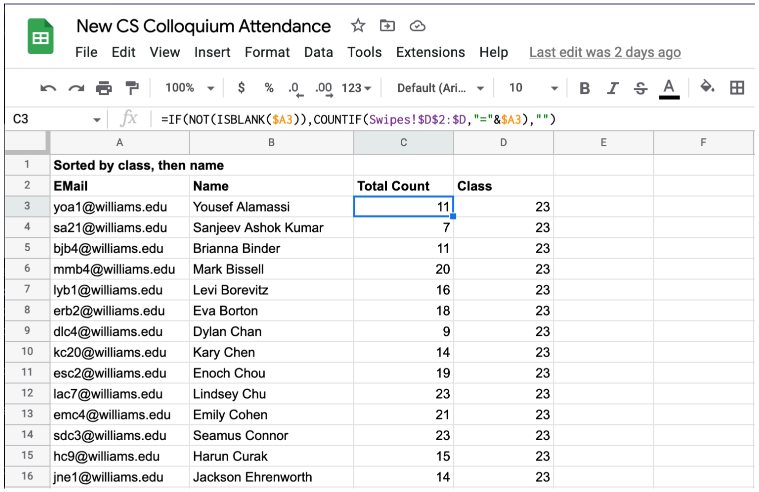
- Leave programming to the **experts**, or
- let **anybody** do it?

C++



```cpp
// Look for a matching path substitution and return the path this command should use
string Command::substitutePath(string p) noexcept {
  auto iter = _current_run._substitutions.find(p);
  if (iter == _current_run._substitutions.end()) return p;

  LOG(exec) << this << ": Replacing path " << p << " with " << iter->second;

  return iter->second;
}

// Inform this command that it used a temporary file
void Command::addTempfile(shared_ptr<Artifact> tempfile) noexcept {
  // Add the tempfile and mark it as not accessed (the value in the map)
  _current_run._tempfiles.emplace(tempfile, false);
}

// Get a reference from this command's reference table
const shared_ptr<Ref>& Command::getRef(Ref::ID id) noexcept {
  ASSERT(id >= 0 && id < _current_run._refs.size())
      << "Invalid reference ID " << id << " in " << this;
  ASSERT(_current_run._refs[id]) << "Access to null reference ID " << id << " in " << this;
  return _current_run._refs[id];
}

// Store a reference at a known index of this command's local reference table
void Command::setRef(Ref::ID id, shared_ptr<Ref> ref) noexcept {
  ASSERT(ref) << "Attempted to store null ref at ID " << id << " in " << this;

  // Are we adding this ref onto the end of the refs list? If so, grow as needed
  if (id >= _current_run._refs.size()) _current_run._refs.resize(id + 1);
```

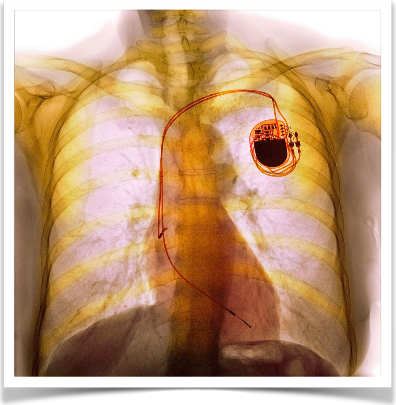
Spreadsheets



=IF(NOT(ISBLANK($A3)),COUNTIF(Swipes!$D$2:$D,"="&$A3),"")

	A	B	C	D	E	F
1	**Sorted by class, then name**					
2	**EMail**	**Name**	**Total Count**	**Class**		
3	yoa1@williams.edu	Yousef Alamassi	11	23		
4	sa21@williams.edu	Sanjeev Ashok Kumar	7	23		
5	bjb4@williams.edu	Brianna Binder	11	23		
6	mmb4@williams.edu	Mark Bissell	20	23		
7	lyb1@williams.edu	Levi Borevitz	16	23		
8	erb2@williams.edu	Eva Borton	18	23		
9	dlc4@williams.edu	Dylan Chan	9	23		
10	kc20@williams.edu	Kary Chen	14	23		
11	esc2@williams.edu	Enoch Chou	19	23		
12	lac7@williams.edu	Lindsey Chu	23	23		
13	emc4@williams.edu	Emily Cohen	21	23		
14	sdc3@williams.edu	Seamus Connor	23	23		
15	hc9@williams.edu	Harun Curak	15	23		
16	jne1@williams.edu	Jackson Ehrenworth	14	23		

PL matters because computers are everywhere



It's up to us how we want to use our machines

"A tasteful watercolor painting of a person pondering what to do with a computer."
(DALL·E, Dec 2022)

Next steps
(aka, some things to do over the summer)

- **Teach yourself** another programming language.
- Dig in to **a problem that bugs you**.
  (me: I've always wanted to write a computer algebra solver)
- **Keep playing** with your project! It's yours! (and you can **show it off** to interviewers)
- Most of all, **do something that excites you**.

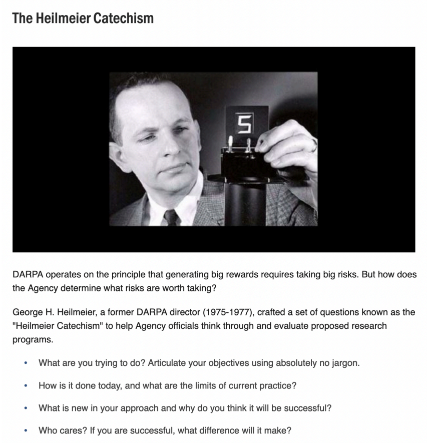
CSCI 334:
Principles of Programming Languages

Lecture 22-2:
How to give a **good** talk

Instructor: Dan Barowy

Williams

Video presentation



The Heilmeier Catechism

DARPA operates on the principle that generating big rewards requires taking big risks. But how does the Agency determine what risks are worth taking?

George H. Heilmeier, a former DARPA director (1975-1977), crafted a set of questions known as the "Heilmeier Catechism" to help Agency officials think through and evaluate proposed research programs.

- What are you trying to do? Articulate your objectives using absolutely no jargon.
- How is it done today, and what are the limits of current practice?
- What is new in your approach and why do you think it will be successful?
- Who cares? If you are successful, what difference will it make?

https://www.darpa.mil/work-with-us/heilmeier-catechism

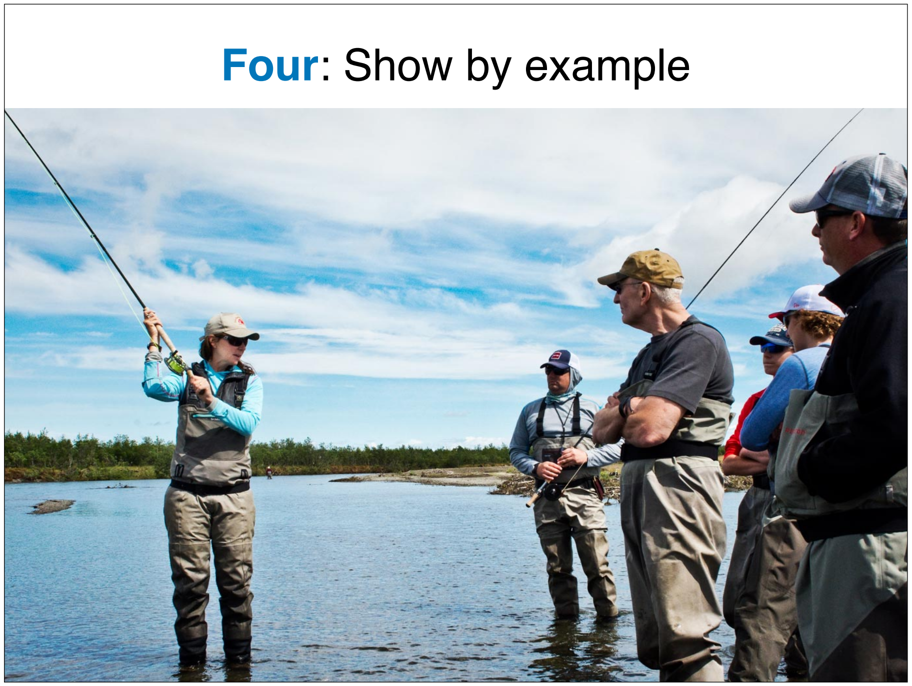
How to give a good talk

Five tips

One: Have a story

Two: Don’t “bury the lede”

Three: Don’t make your audience read



Four: Show by example

Five: Stay on script

Six (oops!): Finish on time

Recap & Next Class

This lecture:

Why PL matters

How to give a good talk

Next lecture:

No next lecture!

Evaluation Forms

(all of these are anonymous)

We **listen carefully** to what you say in these forms. Please take your time and write thoughtful responses.

Your feedback is **very valuable** to us!

Purpose of SCS Forms

"[T]he SCS provides instructors with feedback regarding their courses and teaching. The faculty legislation governing the SCS provides that SCS results are made available to the appropriate department chair, the Dean of the Faculty, and at appropriate times, to members of the Committee on Appointments and Promotions (CAP). The results are considered in matters of faculty reappointment, tenure, and promotion."

—Office of the Provost, Williams College

Purpose of "Blue Sheets"

Student comments on the blue sheets […] are solely for [instructor] benefit. They are not made available to department or program chairs, the Dean of the Faculty, or the CAP for evaluation purposes.

—Office of the Provost, Williams College

Blue sheet prompts:

* What **course topic** did you **enjoy the most**?

* What **course topic** did you **least enjoy**? Do you think that it was valuable to learn anyway?

* Are there **other aspects** of the course that you **liked** or **disliked**? (E.g., *office hours*, *TAs*, *assignments*, *course structure*, *meeting times*, etc.) Feel free to suggest alternative approaches!

* Did you **look forward to coming to class**?