# CSCI 334:
# Principles of Programming Languages

## Lecture 23: Why PL Matters

Instructor: Dan Barowy

Williams

## Topics

Wildly different models of programming.

Which language do I use?

## Your to-dos

1. **Mostly working** project, **due Sunday 12/10**
2. Want to talk about your project?
   Last chance.
   Office hours tomorrow 10-11am, 1:30-2:30pm.

## Final project timeline

1. ~~Project proposal (Lab 8), **due Sun 11/12**~~
2. ~~Minimally working version (Lab 9), **due Sun 11/19**~~
3. ~~Language specification doc (Lab 10), **due Sun 12/3**~~
4. Mostly working version (Lab 11), **due Sun 12/10**
5. Project + video presentation (Lab 12), **due Sun 12/17**

# Announcements

**CS Holiday Party**

**Friday, Dec 8 @ 2:35pm**
**CS Common Room**

Join the CS faculty and your peers for an end-of-semester celebration. We will have hot cocoa and treats for you to enjoy. Last gathering of the year!

# 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.

# Java, Python, etc. pass the rectangle test

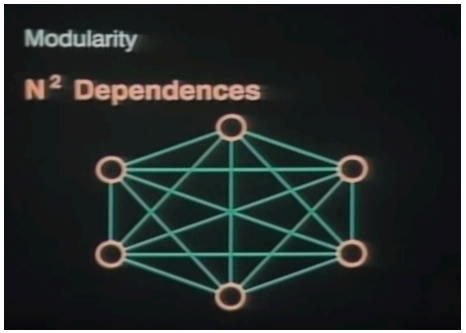
# The right choice depends on the problem

- 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?

# Prolog

# Prolog

- The goal of AI is to enable a computer to answer declarative queries.
- I.e., it already knows how to answer you.
- Prolog was an attempt to solve this problem.
- Since this was early work, the input language was somewhat primitive: predicate logic.
- As you will see, formulating queries in pure logic is not the easiest thing to do.
- However, for certain classes of logic, there are known efficient, deterministic algorithms for solving every possible query.

# Logic Programming

- Logic programming began as a collaboration between AI researchers (e.g., John McCarthy) and logicians (e.g., John Alan Robinson) to solve problems in artificial intelligence.
- Cordell Green built the first "question and answer" system using Robinson's "unification algorithm," demonstrating that it was practical to prove theorems automatically.

# Prolog

- Alain Colmerauer and Phillippe Roussel at Aix-Marseille University invented Prolog in 1972.
- They were inspired by a particular formulation of logic, called "Horn clauses," popularized by the logician Robert Kowalski.
- Horn clauses have a "procedural interpretation," meaning that they suggest a simple procedure for solving them, called "resolution."
- John Alan Robinson's unification algorithm is an efficient algorithm for doing resolution, and this is essentially the algorithm used by Prolog.

# Horn Clause

- Horn clauses are composed of two simple pieces:
  - facts
  - rules (clauses)
- Rules are composed of facts
- Complex facts may also be composed using conjunction.
- We will explore these concepts using Prolog syntax.
- Note that Horn clauses can be "satisfied" in polynomial time.
- In fact, Horn logic is the most expressive form of logic known to be satisfiable in polynomial time.

# Facts (Prolog syntax)

- Here are some facts:

```
raining.
cloudy.
thursday.
```

- Facts are assumed to be true.
- Facts of this form are sometimes called "atoms", since they are indivisible.
- The meaning of these facts is up to the programmer.
- Facts can also be compound:

```
raining,cloudy.
cloudy,thursday.
```

- "`,`" denotes "logical and".
- Note that, in Prolog, facts are always lowercase and must begin with a letter.

# Rules (Prolog syntax)

- Here are some rules:

```
sleep_deprived :- thursday.
unhappy :- raining,cloudy.
```

- The interpretation of a rule `X :- Y` is:
  if `Y` is true, then `X` is true
- In other words, `Y` is the antecedent and `X` is the consequent.
- So, we might interpret the above as:
  "students are sleep deprived if it is Thursday"
  "I am unhappy if it is raining and cloudy."

# Variables (Prolog syntax)

- Note that I just used a generalization of rules without definition:

```
X :- Y
```

- Prolog explicitly allows generalizations of facts like this.
- We call these generalizations "variables", because their precise values (i.e., facts) may not be known to us.
- In the "execution" of a Prolog program, we seek to "instantiate" variables with facts.
- In Prolog, variables are always written starting with an uppercase letter.
- We will come back to variables shortly.

# Complex facts (Prolog syntax)

- Prolog allows one additional form:

```
musician(mia).
musician(john).
friends_with(mia,john).
```

- Statements of this form are called "complex facts."
- Again, the interpretation is up to you.
- E.g.,

  "Mia is a musician."

  "John is a musician."

  "Mia is friends with John."
- Note that we do not automatically assume that "John is friends with Mia"!

# Queries

- Taken together, facts and rules form a "knowledge base."

```
raining.
cloudy.
thursday.
sleep_deprived :- thursday.
unhappy :- raining,cloudy.
```

- A query asks the knowledge base a question. E.g.,

```
?- sleep_deprived.
true
?- unhappy.
true
```

# Resolution

- "Resolution" is the name of the procedure that Prolog uses to "satisfy" a query.

```
raining.
cloudy.
thursday.
sleep_deprived :- thursday.
unhappy :- raining,cloudy.
```

- Essentially, we seek to reduce a query expression to the expression true by substitution.
- Remember that facts are assumed to be true.

# Resolution

```
1. raining.
2. cloudy.
3. thursday.
4. sleep_deprived :- thursday.
5. unhappy :- raining,cloudy.
```

```
?- sleep_deprived.
```

- For a given query, we first seek either a fact that immediately makes the query true, or we seek a rule whose consequent is the query.
- When a rule is reduced to the form `X :- true`, then X is true.

```
6. sleep_deprived :- thursday (by KB4)
7. sleep_deprived :- true (by KB3)
8. true (by deduction)
```

# Resolution

- Given the following knowledge base,

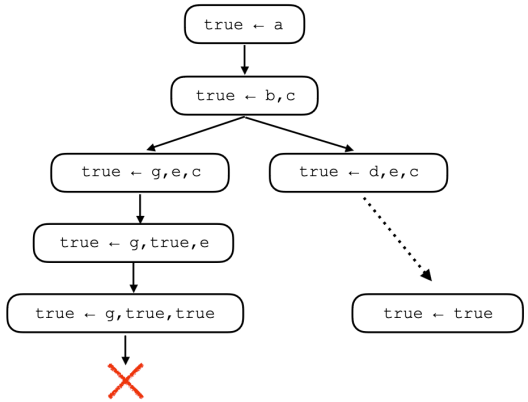
```
1. a :- b,c.
2. b :- d,e.
3. b :- g,e.
4. c :- e.
5. d.
6. e.
7. f :- a,g.
```

- let's try to satisfy the following query using resolution:

```
?- a.
```

# Proof Search

- Nonetheless, Prolog is not generally sensitive to the order of the facts in a database. How does this work?
- The answer is that resolution is actually a form of backtracking search.




# Resolution with Variables

- Resolution with variables can be very computationally expensive.
- Unification allows resolution with variables to be completed in polynomial time.
- The basic insight is to "instantiate" variables "on demand" instead of enumerating all possible variable instantiations into facts.
- Hindley-Milner is essentially just unification.

```
1. musician(mia).
2. musician(john).
3. friends_with(X,Y) :- musician(X),musician(Y).
```

- Let's resolve the following query:

```
?- friends_with(mia,john).
```

# Resolution with Variables

- When asking a query that utilizes variables, Prolog will both search for a satisfying assignment and it will return that assignment.
- There may be more than one possible assignment.
- If so, use the ";" command to ask for another solution.
- Let's resolve the following query:

```
?- friends_with(mia,Who).
```

- We may even ask:

```
?- friends_with(Who1,Who2).
```

# Exercise

- Construct the a knowledge base containing the following facts:
  - "Giants eat people."
  - "Giants eat bunnies."
  - "Bunnies eat grass."
  - "People eat bunnies."
  - "People eat people."
  - "Those who are eaten by others hate those others."
  - "Monsters love those who hate themselves."
- Then supply a query that can answer:
  - "Who do monsters love?"

# 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
- Maybe Prolog?
- C?

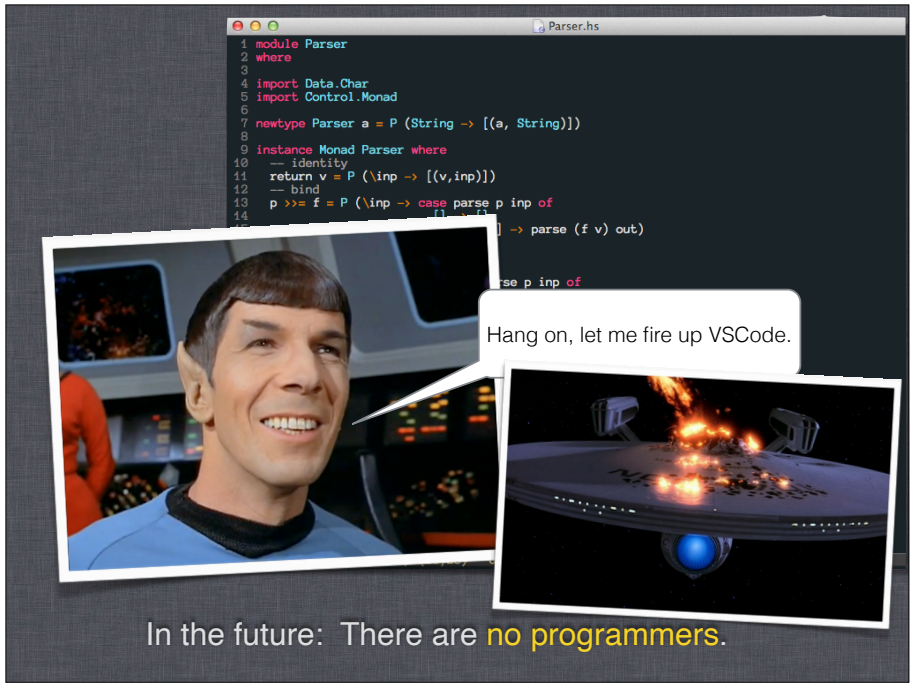
# FlashMeta

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

Daniel Barowy

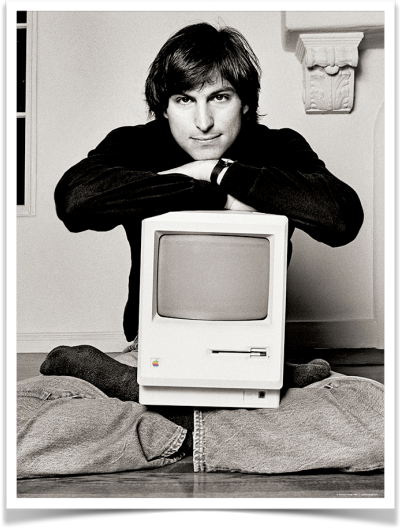
UMass Amherst

Williams College, January 9, 2017




In the future: There are no programmers.

## A Bicycle for the Mind



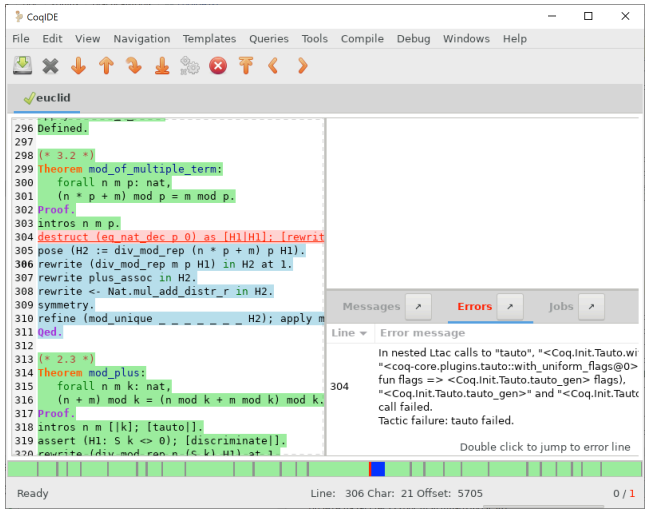
This work is not done yet.

Some things to think about

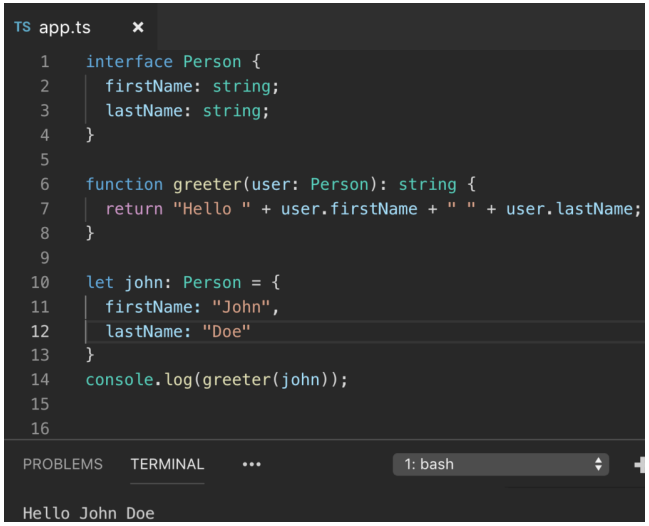
## Would it be better to

- 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




## TypeScript



## Would it be better to

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

## 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

```
<!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
    </div><button class="navb
      aria-controls="navbarS
      <span class="navbar-to
    </button>
    <div class="collapse navb
      <ul class="navbar-nav
          <a class="nav-li
        </li><li class="na
          <a class="nav-li
        </li><li class="na
          <a class="nav-li
        </li><li class="na
          <a class="nav-li
        </li><li class="na
          <a class="nav-li
        </li><li class="na
          <a class="nav-li
        </li></ul>
    </div>
  </div>
```

CSCI 334: Principles of Programming Languages, Fall 2022

Home Lectures Assignments Handouts Help Hours Feedback

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

```
494  // Look for a matching path substitution and return the path this command should use
495  string Command::substitutePath(string p) noexcept {
496    auto iter = _current_run._substitutions.find(p);
497    if (iter == _current_run._substitutions.end()) return p;
498
499    LOG(exec) << this << ": Replacing path " << p << " with " << iter->second;
500
501    return iter->second;
502  }
503
504  // Inform this command that it used a temporary file
505  void Command::addTempfile(shared_ptr<Artifact> tempfile) noexcept {
506    // Add the tempfile and mark it as not accessed (the value in the map)
507    _current_run._tempfiles.emplace(tempfile, false);
508  }
509
510  // Get a reference from this command's reference table
511  const shared_ptr<Ref>& Command::getRef(Ref::ID id) noexcept {
512    ASSERT(id >= 0 && id < _current_run._refs.size())
513        << "Invalid reference ID " << id << " in " << this;
514    ASSERT(_current_run._refs[id]) << "Access to null reference ID " << id << " in " << this;
515    return _current_run._refs[id];
516  }
517
518  // Store a reference at a known index of this command's local reference table
519  void Command::setRef(Ref::ID id, shared_ptr<Ref> ref) noexcept {
520    ASSERT(ref) << "Attempted to store null ref at ID " << id << " in " << this;
521
522    // Are we adding this ref onto the end of the refs list? If so, grow as needed
523    if (id >= _current_run._refs.size()) _current_run._refs.resize(id + 1);
```

# Spreadsheets

New CS Colloquium Attendance

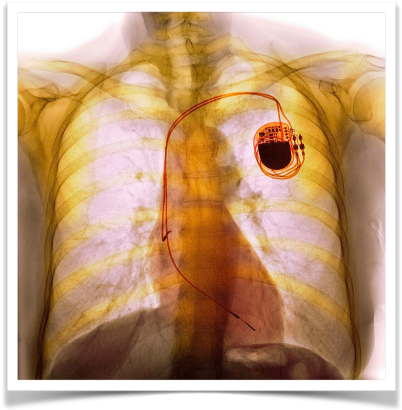
File Edit View Insert Format Data Tools Extensions Help Last edit was 2 days ago

100% $ % .0 .00 123 Default (Ari... 10

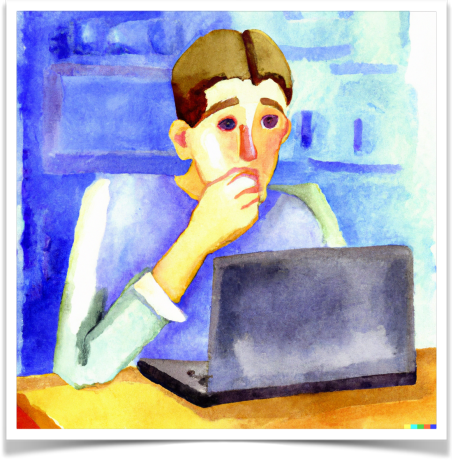
C3 =IF(NOT(ISBLANK($A3)),COUNTIF(Swipes!$D$2:$D,"="&$A3),"")

| | 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)**

# Which language do I use?

# 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**.