CSE 341: Programming Languages

- The Team:
  - Alan Borning, instructor
  - Ken Yasuhara, teaching assistant
  - Harr Chen, teaching assistant
- Non-majors who want to take the class:
  - Please talk with one of the staff advisors in the main Computer Science & Engineering office!!
- "It's on the Web"
  - www.cs.washington.edu/341
- Add yourself to the class listserv
  - Directions are on the class web page

Course topics

- Four languages:
  - Miranda (a pure functional programming language)
  - Java
  - Smalltalk (a pure object-oriented language)
  - Scheme (like Lisp … lots-o-parentheses)
- Maybe:
  - perl
  - CLP(R) (constraint logic programming)
- General programming language concepts

Required work

- Warmup program and moderate-sized in each language
- Larger project of your own choosing, in either Java or Smalltalk (can be done in groups)
- Midterm, final
- Some written homework
- Homework normally due on a Monday (so that we can get it back to you in section on Thurs)

Texts

- These are all recommended, not required:
  - Simon Thompson, *Miranda: The Craft of Functional Programming* [low priority to buy]
  - Timothy Budd, *Understanding Object-Oriented Programming with Java* [medium priority]
  - Mark Guzdial, *Squeak: Object Oriented Multimedia Applications* [high priority]
  - Hal Abelson and Gerald Sussman, *Structure and Interpretation of Computer Programs* [medium priority]
- The web page also has more thoughts about which books to buy, if you only want to buy some of them
- On 4 hour reserve in the Engineering Library (along with other useful references – complete list is on the web)
- Will try to put some in the ACM library in Sieg

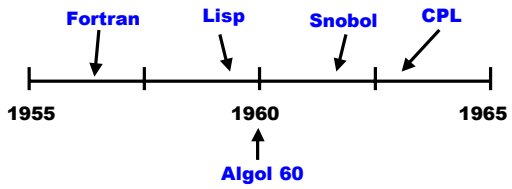
Grading Policy

- Grading scale:
  - homework (40%)
  - project (20%)
  - midterm (15%)
  - final (25%)
- Late policy:
  - Each student is granted two late days to use at his/her discretion during the quarter (see the web page for detailed rule)
  - No other late days or extensions except under very unusual circumstances

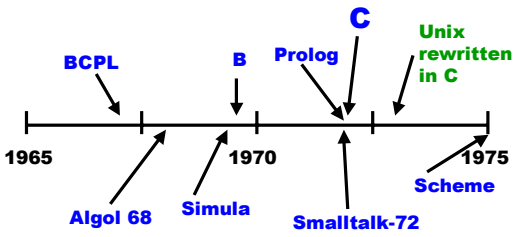
Collaboration Policy

- Collaboration policy: "Gilligan's Island Rule" (see the web page)
  - OK (and encouraged) to talk with other students in the class about assignments
  - Don't take away any written material from the discussion
  - Do something mindless for 0.5 hours
  - Then do your assignment
- Freedom of Information Rule
  - Write the names of your collaborators on any assignment
- Cases of academic misconduct will be turned over to the Cheating Committee

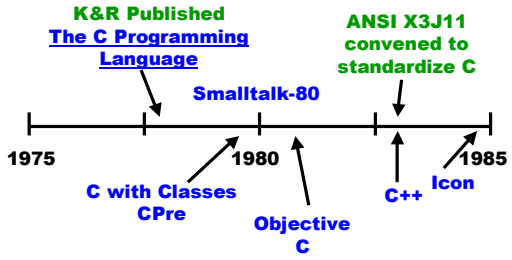
History of Programming Languages



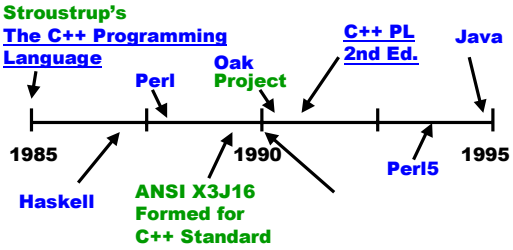
History of Programming Languages



History of Programming Languages



History of Programming Languages



What is a programming language for?

- Instructing machines?
- Communicating among programmers?
- Expressing high level designs?
- Notation for algorithms?
- Tool for experimentation?

Languages are for both humans and computers!

Effective Use of Programming Languages

"Learning the fundamentals of a programming language is one thing: learning how to design and write effective programs in that language is something else entirely."

—Scott Meyers

Why do we care?

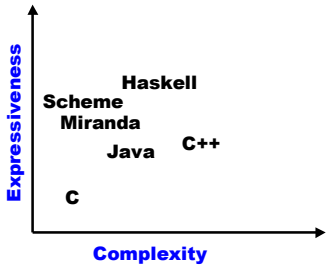
- Whorf-Sapir hypothesis for natural languages
- Tradeoffs among languages
  - reusability, maintainability
  - performance, robustness
  - flexibility, dynamicism
  - libraries
  - aesthetics (i.e., "fun-ness")

Language classification

- Imperative (Fortran, Algol, C)
- Object-oriented (Smalltalk, Java, C++)
- Functional ("Pure" Scheme/Lisp, Miranda)
- Logic/Constraint (Prolog, CLP(R))

æ Languages may encourage a certain style even if they do not force it on you!

Complexity vs. Expressiveness



What's wrong with imperative?

```
int i = 7;

   ⋮

printf("%d\n",i*2);
```

- What gets printed?

Assignments make reasoning difficult!

```
int i = 7;

   ⋮

i = 3;

   ⋮

printf("%d\n",i*2);
```


Imperative programming

- Nice for execution, translation… BUT:
- Harder for humans to understand and reason about
- Harder for sophisticated software tools
  - Proving correctness is harder
  - Restricts code motion, limits optimizer (especially important for parallel machines)

Object-Oriented programming

- A kind of imperative programming language
- Metaphor: objects that communicate with each other by sending and receiving messages
- Each object is an instance of a class
- Classes come in hierarchies
- Big benefits of OO programming:
  - Natural way of decomposing many problems
  - Modular
  - Good for supporting software reuse (frameworks)

The Functional Approach

- Eliminates assignments (side effects), focus on expressions
- Tell **what** to compute, not **how** (leave order of computation unspecified)
- Higher level programming model— leave more details to machine

Miranda (and Haskell)

- Pure functional languages
- Statically-typed
- "Lazy" evaluation

Sample Miranda function definition:

```
factorial n = product [1..n]
```


Scheme

- Very simple syntactically
- Still an imperative language, though
- But encourages a functional style
- Can write in a purely functional subset
  - we will do this in the beginning
  - still has assignment statement
- Dynamically typed

Constraint Logic Programming

- Metaphor: theorem proving and equation solving
- Again, no side effects
- Variables are like those in mathematics

Sample CLP(R) rule:

```
centigrade_fahrenheit(C,F) :- 1.8*C=F-32.
```

Use:

```
?- centigrade_fahrenheit(X,212).
```