Programming Languages: Introduction and Overview

Alan Borning
borning@cs.washington.edu
University of Washington, Seattle
CSE 341, Spring 2000

Course topics

- Scheme
- Perl
- Java
- Miranda (a pure functional language)
- CLP(R) (constraint logic programming)
- General programming language concepts

Required work

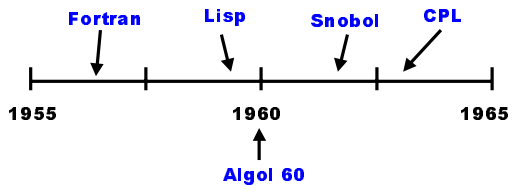
- Moderate sized programs in Scheme, Perl, Miranda, CLP(R)
- Java project
- Course project (implementation and paper) on a language of your choosing (other than Java)
- Midterm, final
- Some written homework

Organizational

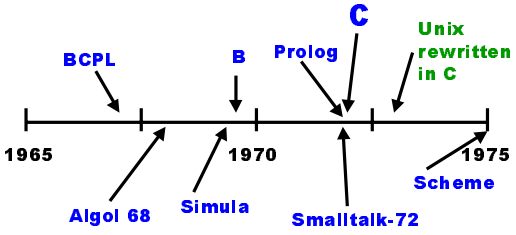
- Instructor: Alan Borning
- TA: Keunwoo Lee
- when in doubt, check the class web page

- please sign up for the cse341 mailing list: mail to majordomo@cs.washington.edu in body of message: subscribe cse341

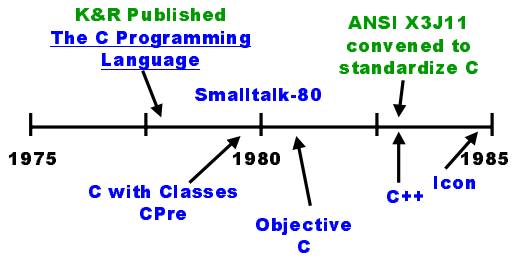
History of Programming Languages



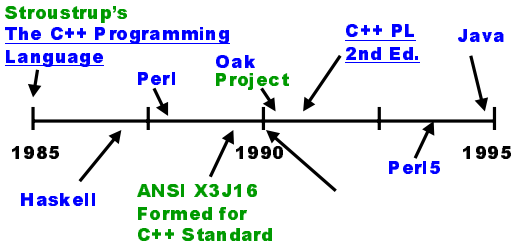
History of Programming Languages



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?

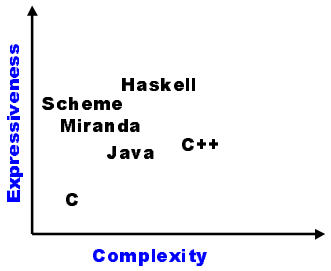
- Whorf-Sapir hypothesis for natural languages
- Tradeoffs among languages
  - reusability, maintainability
  - performance, robustness
  - flexibility, dynamicism
  - libraries
  - aesthetics (i.e., "fun-ness")

Language classification

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

⇨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)

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

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 assignments
- Dynamically typed

Miranda (and Haskell)

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

Sample Miranda function definition:

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


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).
```