CSE 341: Programming Languages

- The Team:
  - Alan Borning, instructor
  - Andrei Alexandrescu, teaching assistant
  - Eric Bessette, teaching assistant
- "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

- Three languages:
  - Java
  - Scheme (like Lisp … lots-o-parentheses)
  - Haskell (a pure functional language with an interesting type system)
- General programming language concepts
- Maybe:
  - perl
  - squeak
  - CLP(R) (constraint logic programming)

Required work

- Warmup and moderate-sized program in each language
- Course project of your own choosing
  - Probably in Java, but we're willing to discuss doing projects in another language
  - Can be done in groups
  - Eclipse and cvs recommended for Java group projects
- Midterm, final
- Some written homework

Books

- Required text:
  - Allen Tucker and Robert Noonan, *Programming Languages: Principles and Paradigms*, McGraw-Hill, 2002
- Additional reference books for the different languages are on 4 hour reserve in the Engineering Library (along with other useful references – complete list is on the web)
  - List of reserve books is on the class web page
  - ACM library in Sieg may also have some of these books

Grading Policy

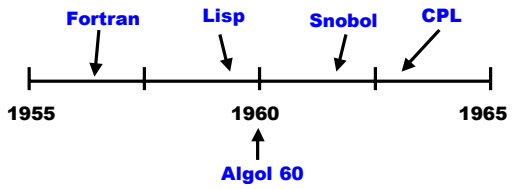
- Grading scale:
  - homework (45%)
  - project (15%)
  - midterm (15%)
  - final (25%)
- Late policy:
  - Each student is granted two late days to use at his/her discretion during the quarter (see the web page for detailed rule)
  - No other late days or extensions except under very unusual circumstances

Collaboration Policy

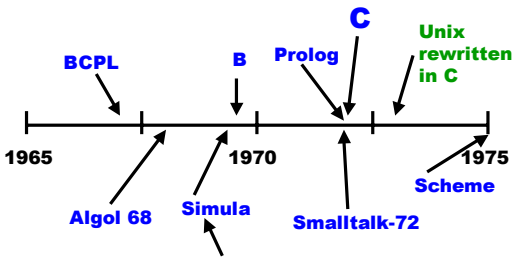
- Collaboration policy: "Gilligan's Island Rule" (see the web page)
  - OK (and encouraged) to talk with other students in the class about assignments
  - Don't take away any written material from the discussion
  - Do something mindless for 0.5 hours
  - Then do your assignment
- Freedom of Information 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

Pre-Fortran: machine code or assembly language

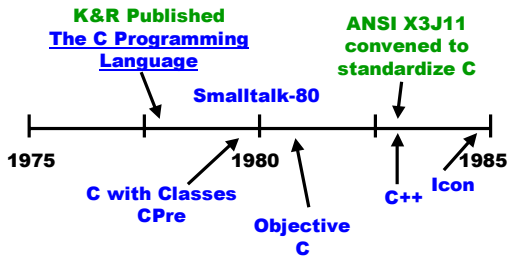


History of Programming Languages

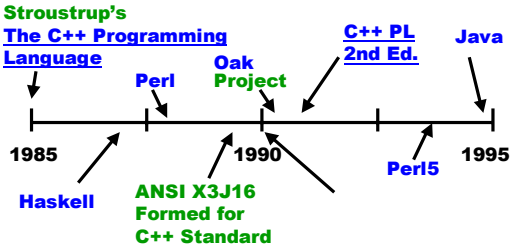


The first object-oriented language!

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, Haskell)
- Logic/Constraint (Prolog, CLP(R))

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

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)

Examples of object-oriented languages

- Java
- C++
- Squeak (a Smalltalk dialect)
  - Interesting features:
    - a pure object-oriented language
    - control structures are handled just by sending messages (no special syntax)

The Functional Approach

- In a pure functional language, there are no side effects (for example, no assignment statements)
- Like functions in mathematics
- Pure model, easy to reason about
- (Arguably) not a good fit for modeling objects that change over time

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

Haskell

- A pure functional language
- Statically-typed
- "Lazy" evaluation

Sample Haskell 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).
```