Why are there so many programming languages?

- Evolution -- we've learned better ways of doing things over time
- Socio-economic factors:
  - Proprietary interests,
  - Commercial advantage orientation toward special purposes
  - Orientation toward special hardware
  - Diverse ideas about what is pleasant to use

What makes a language successful?

- **Ease of use:**
  - Easy to learn (BASIC, Pascal, LOGO, Scheme)
  - Easy to express things
  - Easy to use once fluent -- "powerful" (C++, Common Lisp, APL, Algol-68, perl)
  - Easy to implement (BASIC, Forth) possible to compile to very good (fast/small) code (Fortran)
- **Cost factors:**
  - Backing of a powerful sponsor (COBOL, PL/1, Ada, Visual Basic)
  - Wide dissemination at minimal cost (Pascal, Turing, Java)
Why do we have programming languages?

- Programmer’s perspective:
  - way of thinking
  - way of expressing algorithms
  - languages from the user’s point of view
- Abstraction of virtual machine
  - way of specifying what you want the hardware to do without getting down into the bits
  - languages from the implementor’s point of view

Course Objective: balance coverage of two angles:
- Commonalities and differences among languages
- Implementations of languages

History of Programming Languages

http://www.webopedia.com/TERM/P/programming_language.html

Major Influences to Programming Languages

- Computer Architecture
- Von Neumann Architecture
- Programming methodologies
- Programming paradigms
Influence of Computer Architecture

<table>
<thead>
<tr>
<th>Computer Architecture</th>
<th>Programming Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory cells</td>
<td>Variables</td>
</tr>
<tr>
<td>Pipelined execution of instructions</td>
<td>Computation is viewed as a sequence of actions</td>
</tr>
</tbody>
</table>

Programming Paradigms

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imperative</td>
<td></td>
</tr>
<tr>
<td>Procedural (von Neumann)</td>
<td>Fortran, C, Basic, Pascal, Algol60</td>
</tr>
<tr>
<td>Object-Oriented</td>
<td>Simula67, C++, Smalltalk, Java, C#</td>
</tr>
<tr>
<td>Declarative</td>
<td></td>
</tr>
<tr>
<td>Functional</td>
<td>LISP, Scheme, ML, Haskell</td>
</tr>
<tr>
<td>Logical</td>
<td>Prolog, VisiCalc, RPG, spreadsheets</td>
</tr>
</tbody>
</table>
Desirable Qualities of Software

1. Software must be reliable
   - Program performs to its specification under all conditions

2. Software must be maintainable
   - It is no longer feasible to always build software from scratch
   - Must be able to easily modify existing software

3. Software must execute efficiently
   - Although hardware getting cheaper and has better performance, need for efficient execution remains due to increasingly demanding applications

Why study programming languages?

- Help you choose a language.
  - C vs Modula-3 vs C++ for systems programming
  - Fortran vs APL vs Ada for numerical computations
  - C vs Ada vs Modula-2 for embedded systems
  - Common Lisp vs Scheme vs ML for symbolic data manipulation
  - Java vs C/CORBA for networked PC programs

- Make it easier to learn new languages (by analogy)
  - some languages are similar; easy to walk down family tree (e.g., Fortran 77, Fortran90)
  - Identify common concepts:
    - E.g.: iteration, recursion, abstraction

- Think of an analogy to human languages:
  - good grasp of grammar makes it easier to pick up new languages (at least romance languages).
Make better use of language

Understand obscure features:

- In C, help you understand unions, arrays & pointers, separate compilation, varargs.
- In Common Lisp, help you understand first-class functions/closures, streams, catch and throw, symbol internals.

Make better use of language

Understand implementation costs: choose between alternative ways of doing things, based on knowledge of what will be done underneath:

- Use simple arithmetic equalities (use \(x^2\) instead of \(x**2\)).
- Use C pointers or Pascal "with" statement to factor address calculations
  - avoid call by value with large data items in Pascal
  - avoid the use of call by name in Algol 60
  - choose between computation and table lookup.

Make better use of given language

Figure out how to do things in languages that don't support them explicitly:

- Lack of suitable control structures in Fortran IV
  - use comments and programmer discipline for control structures
- Lack of recursion in Fortran 77, CSP, etc.
  - write a recursive algorithm then use mechanical recursion
  - elimination (even for things that aren't quite tail recursive)
  - lack of named constants and enumerations in Fortran
  - use variables that are initialized once, then never changed
  - lack of modules in C and Pascal
  - use comments and programmer discipline
  - lack of iterators in just about everything
  - use them with (member?) functions
Language Design Criteria

- **Simplicity**
  - Simpler language is easier to master
  - Achieved by having a small number of features
  - Feature multiplicity:
    - More than one way to accomplish the same task
    - E.g.: `count+=1; count = count + 1; count += 1`
  - Operator overloading:
    - Same operator has more than one meaning
    - Overload `+` operator for:
      - Summing integers in two arrays
      - Concatenating two arrays

Language Design Criteria

- **Orthogonality**
  - Every possible combination of primitive constructs of the language is legal and meaningful
  - Lack of orthogonality means lots of exceptions in the language rules
    - Assembly language for IBM mainframe:
      - `A Reg1, memory cell` and `AR Reg1, Reg2` registers
      - Two different rules for addition
      - Cannot use A for adding 2
  - Too much orthogonality makes language becomes overly complex (Algol 68)

Language Design Criteria-Syntax

- **Identifier forms**
  - Fortran77 restricts identifiers to ≤6 characters
- **Special words**
  - Pascal requires `begin-end` pairs to form groups for all control constructs
  - Ada uses “end if” or “end loop” to distinguish closing syntax for each type of control statement
  - Fortran 90 allows special words such as DO and END to be legal variable names
Language Design Criteria

- Safety:
  - Language should not provide features that make it possible to write harmful programs
  - E.g., goto statements and pointer variables
- Type checking:
  - Testing for type errors in a given program, during compilation or program execution
- Aliasing:
  - Having two or more distinct referencing methods or names for the same memory cell (e.g., union members and pointers in C)
- Robustness:
  - Provides the ability to deal with undesired events (arithmetic overflows, invalid input, etc)
  - E.g., Exception handling facility in Java and C++

Organization of Programming Languages-Cheng (Fall 2004)

Design Trade-offs

- Reliability vs Efficiency:
  - Ada demands all references to array to be checked to ensure that indices are within their legal ranges
  - C does not require index range checking, and thus executes faster

Buffer overflow problem:

```c
void function (char *str) {
    char buffer[16];
    strcpy(buffer, str);
}
```

```c
int main () {
    char *str = "I am greater than 16 bytes";
    function (str);
}
```

Organization of Programming Languages-Cheng (Fall 2004)
Design Trade-offs

- Flexibility vs Safety
  - Pascal variant records allow a memory cell to contain either a pointer or an integer
  - This allows a program to do arithmetic on pointers, which is sometimes convenient, but is a dangerous practice.

Course Organization

- Lectures material supplement textbook material
  - (i.e., come to class)
- Homework assignments reinforce key concepts
- Programming assignments give hands-on experience
- Exams provide a way for you to demonstrate what you’ve learned
  - (and I need something to use for calculating grades)

Course Organization

<table>
<thead>
<tr>
<th></th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homework</td>
<td>15%</td>
</tr>
<tr>
<td>Programming Assignments</td>
<td>30%</td>
</tr>
<tr>
<td>Exams (2)</td>
<td></td>
</tr>
<tr>
<td>Exam 1: 10/14/04</td>
<td>50%</td>
</tr>
<tr>
<td>Exam 2: 12/17/04</td>
<td></td>
</tr>
<tr>
<td>In-class participation</td>
<td>5%</td>
</tr>
<tr>
<td>Course Textbook</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td></td>
</tr>
<tr>
<td>✓ Best programming language textbook (based on student and faculty reviews)</td>
<td></td>
</tr>
<tr>
<td>✓ 2nd edition due out in 2005</td>
<td></td>
</tr>
<tr>
<td>✓ MSU participate in evaluation of draft of 2nd edition (Programming Language Pragmatics, M. Scott)</td>
<td></td>
</tr>
<tr>
<td>✓ Cheap textbook (Purchase course pack at Anthony Hall copy center)</td>
<td></td>
</tr>
<tr>
<td>✓ Perform evaluations of each chapter</td>
<td></td>
</tr>
<tr>
<td>✓ (part of in-class participation grade)</td>
<td></td>
</tr>
</tbody>
</table>