Why Study Programming Languages?
Objectives

- Master important concepts for PLs
- Master different language paradigms
- Master (some) implementation issues

At the end of the course:
- You will understand important PL concepts
- You will understand the differences between imperative, functional, and object-oriented
- You will have some idea how to implement compilers and interpreters for PLs
Too labor-intensive and error-prone

**GCD algorithm in MIPS machine code:**

```assembly
27bdff0d afbf0014 0c1002a8 00000000 0c1002a8 afaf001c 8fa4001c
00401825 10820008 0064082a 10200030 00000000 00000020 00832023
00641823 1483fff9 a064082a 0000001c 8fbf00e4 27bd0020
03e00008 00001025
```

**In Assembly**

```
addiu sp,sp,-32
sw ra,20(sp)
jal getint
nop
jal getint
sw v0,28(sp)
lw a0,28(sp)
move v1,v0
beq a0,v0,D
slt at,v1,a0
A: beq at,zero,B
nop
```

```
addiu
```
Evolution of Programming Languages

- Hardware
- Machine code
- Assembly
- Macro Assembly
- FORTRAN 1954
- ...

CSE 655
Incomplete History
Why So Many Languages?

- Evolution of language features
  - e.g. goto vs. if-then, switch-case, while-do
  - weak types (C) vs. strong types (Java)
  - error conditions: error codes (C) vs. exceptions and exception handling (C++, Java)
  - memory management: programmer (C, C++) vs. language (Java through garbage collection)

- Personal preferences
  - Syntax; procedural vs. functional vs. ...

- Special-purpose languages
Application Domains

- Scientific applications (Fortran, C, Matlab)
- Business applications (Cobol)
- Artificial intelligence (Lisp)
- Systems programming (C, C++)
- Enterprise computing (Java, C#)
- Very high-level languages (perl)
- Special purpose languages (make, sh)
What Makes a Language Successful?

- Expressive power
- Ease of use for novices
- Ease of implementation
- Open source
- Availability of compilers and libraries
- Economics, promotion, inertia
- Syntax that looks like C
Programming Language Paradigms

- Imperative (C, Pascal, etc.)
  - Underlying model: von Neumann machine

- Functional (Lisp, ML, Haskell)
  - Underlying model: lambda calculus

- Logic (Prolog)
  - Underlying model: first-order logic

- Object-oriented (C++, Java, C#, CLOS)
  - Underlying model: object calculus
Why Study Programming Languages?

- Understand language features
  - Both popular and obscure
- Choose among ways to express ideas
- Make good use of debuggers, other tools
- Simulate nice features in other languages
- Choose appropriate language for problem
- Learn new languages faster
- Design simple languages
Implementation Methods

- **Compilation (C, C++, ML)**

- **Interpretation (Lisp)**

- **Hybrid systems (Java)**
Overview of Compilation

Character stream
    Token stream
        Parse tree
            Abstract syntax tree or other intermediate form
                Modified intermediate form
                    Assembly or machine language, or other target language
                        Modified target language
                            Scanner (lexical analysis)
                                Parser (syntax analysis)
                                    Semantic analysis and intermediate code generation
                                        Machine-independent code improvement (optional)
                                            Target code generation
                                                Machine-specific code improvement (optional)
                                                    Symbol table
Source Code for a GCD Algorithm

program gcd(input, output);
var i, j: integer;
begin
  read(i, j);
  while i <> j do
    if i > j then i := i – j
    else j := j – i
  writeln(j);
end.
Tokens (After Lexical Analysis)

PROGRAM, (IDENT, "gcd"), LPAREN,
(IDENT, "input"), COMMA,
(IDENT, "output"), SEM,
VAR, (IDENT, "i"), COMMA,
(IDENT, "j"), COLON, INTEGER, SEM,
BEGIN
...

CSE 655
Abstract Syntax Tree and Symbol Table
Assembly (Target Language)

```
addiu sp,sp,-32      # reserve room for local variables
lw ra,20(sp)        # save return address
jal getint          # read
nop
sw v0,28(sp)        # store i
jal getint          # read
nop
sw v0,24(sp)        # store j
lw t6,28(sp)        # load i
lw t7,24(sp)        # load j
nop
bne t6,t7,D         # branch if i = j
nop
A:  lw t8,28(sp)     # load i
lw t9,24(sp)        # load j
nop
slt at,t9,t8        # determine whether j < i
bne at,zero,B       # branch if not
lw t0,28(sp)        # load i
lw t1,24(sp)        # load j
nop
subu t2,t0,t1       # t2 := i - j
sw t2,28(sp)        # store i
b C
nop
B:  lw t3,24(sp)     # load j
lw t4,28(sp)        # load i
nop
subu t5,t3,t4       # t5 := j - i
sw t5,24(sp)        # store j
C:  lw t6,28(sp)     # load i
lw t7,24(sp)        # load j
nop
bne t6,t7,A         # branch if i <> j
nop
D:  lw a0,28(sp)     # load i
jal putint          # writeln
nop
move v0,zero        # exit status for program
b E                 # branch to E
nop
b E                 # branch to E
nop
E:  lw ra,20(sp)     # retrieve return address
addiu sp,sp,-32     # deallocate space for local variables
jr ra                # return to operating system
nop
```
Intermediate Languages for Portability

- **Java**: the translator produces bytecode

- **Intermediate language I for portability**
  - Compiler for a language L on machine M
    - Use a compiler \([L \rightarrow I]\), and then use a compiler \([I \rightarrow \text{machine code for } M]\)
    - Adding a new L or a new M is simpler
  - Common intermediate language: C
    - Hardware vendors often supply C compiler