Why Are We Interested in Programming Languages?

What's the alternative?
  Programming in machine code is way too tedious/error-prone

There are a lot of languages out there
  Learn to learn new languages quickly

Introduction

Textbook, Chapter 1
Euclid’s GCD Algorithm

- In MIPS R4000 machine code:

- In Assembly:

Evolution of Programming Languages

- Hardware
- Machine code
- Assembly
- Macro Assembly
- FORTRAN 1954
- etc.
Why So Many Languages?

- Evolution
  - From goto to loops, case statements
- Personal Preference
  - Syntax
  - Loops vs. recursion
  - Pointers vs. recursive data types
- Special Purposes

Application Domains

- Scientific applications (Fortran, TCE)
- Business applications (Cobol)
- Artificial intelligence (Lisp)
- Systems programming (C, C++)
- Web service programming (Java, C#)
- Scripting languages (perl, Python, etc)
- Special purpose languages (make, sh)

What Makes a Language Succeed?

- Expressive Power
- Ease of Use for Novice
- Ease of Implementation
- Open Source
- Availability of Compilers, Libraries
- Economics, Patronage, Inertia
- Syntax that looks like C
Language Design Issues

- Readability (*p++*)
- Abstractions (functions, classes)
- Orthogonality (no special cases)
- Reliability (type checking)
- Cost (training programmers)

Programming Paradigms

- Imperative (C, Pascal, etc.)
- Functional (Lisp, ML, Haskell)
- Logic (Prolog)
- Object-Oriented (C++, Java, CLOS)

Why Do We Study Programming Languages?

- Understand obscure language features
- 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, ML)**
  - Source program
  - Compiler
  - Target program
  - Output

- **Interpretation (early Lisp)**
  - Source program
  - Interpreter
  - Output

- **Hybrid Systems (early Java)**
  - Source program
  - Translator
  - Interpreter
  - Output

Overview of Compilation

Source Code for GCD

```java
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
...
```
```
; Assembly language code

section .data
    message db 'Hello, world!', 0

section .text
    global _start

_start:
    mov eax, 4
    lea eax, [message]
    mov ebx, 1
    int 0x80
    mov eax, 1
    xor ebx, ebx
    int 0x80
```