CSE 3341/5341 Introduction

M. Scott, Chapter 1

Objectives
• 3341: Principles of Programming Languages
• Master important concepts for PLs
• Master several different language paradigms
  – Imperative, object-oriented, functional
• Master some implementation issues
  – You will have some idea how to implement compilers and interpreters for PLs
• Other related courses
  – 6341: Foundations of Programming Languages
  – 5343: Compiler Design and Implementation

Programming in Machine Code
• Too labor-intensive and error-prone
• Euclid’s GCD algorithm in MIPS machine code

Evolution of Programming Languages
• Hardware
• Machine code
• Assembly language
• Macro assembly language
• FORTAN, 1954: first machine-independent, high-level programming language
  – The IBM Mathematical FORmula TRANsling System
• LISP, 1958 (LIST Processing)
• ALGOL, 1958 (ALGOirthmic Language)
• Many hundreds of languages since then

Incomplete History

Why So Many Programming Languages?
• Evolution of language features and user needs
  – Control flow: goto vs. if-then, switch-case, while-do
  – Procedures (Fortran, C) vs. classes/objects (C++, Java)
  – Weak types (C) vs. strong types (Java)
  – Memory management: programmer (C, C++) vs. language (Java through garbage collection)
  – Error conditions: error codes (C) vs. exceptions and exception handling (C++, Java)
Why So Many Programming Languages?

- Different application domains require different specialized languages
  - Scientific computing (Fortran, C, Matlab)
  - Business applications (Cobol)
  - Artificial intelligence (Lisp)
  - Systems programming (C, C++)
  - Enterprise computing (Java, C#)
  - Web programming (PHP, JavaScript)
  - String processing (AWK, Perl)

Programming Languages Spectrum

- Imperative languages
  - What are the steps the computer should follow in order to achieve the programmer’s goals?
  - "Prescriptive" attitude
  - Traditional (non-object-oriented) imperative; object-oriented
- Declarative languages
  - What are the properties of the desired?
  - "Descriptive" attitude – higher level of abstraction
  - Often, lower performance than imperative languages
  - Functional; logic
- The lines are blurred – e.g., F#

Why Study Programming Languages?

- Choose the right language for the job
  - They all have strengths and weaknesses
- Learn new languages faster
  - This is a course on common principles of PL
- Understand your tools better
  - Compilers, interpreters, virtual machines, debuggers, assemblers, linkers
- Write your own languages
  - Happens more often than you’d think!
- To fix bugs & make programs fast, often you need to understand what’s happening “under the hood”

Example: Euclid’s GCD Algorithm

```c
int gcd(int a, int b) {
    while (a != b) {
        if (a > b) a = a - b;
        else b = b - a;
    }
    return a;
} /* C procedure */
```

```scheme
(define gcd (λ(a b) (cond ((= a b) a) ((> a b) (gcd (- a b) b)) ((else (gcd (- b a) a))))) ; Scheme function
```

Programming Languages Paradigms

- **(Non-OO) Imperative** (Fortran, C, Pascal)
  - Underlying model: von Neumann machine
  - Primary abstraction: procedure
- **Object-oriented** (Smalltalk, C++, Java, C#)
  - Underlying model: object calculus
  - Primary abstraction: class or object
- **Functional** (Lisp, Scheme, ML, Haskell)
  - Underlying model: lambda calculus
  - Primary abstraction: mathematical function
- **Logic** (Prolog)
  - Underlying model: first-order logic

Implementation Methods

- **Compilation** (C, C++, ML)
- **Interpretation** (Lisp)
- **Hybrid systems** (Java)
Intermediate Languages for Portability

- Java: the translator produces **Java bytecode**
  - Executed on the Java Virtual Machine (JVM)

  ![Diagram of Java bytecode execution process]

  - Inside the JVM, there is a **bytecode interpreter** and a **just-in-time (JIT) compiler** (triggered for “hot” code)
  - Android: Java bytecode → Dalvik bytecode, for execution on the Dalvik Virtual Machine

- C can be used as an intermediate language: a C compiler is available on pretty much any machine