CMPSC 461

Programming Language Concepts

Gang Tan
Computer Science and Engineering
Penn State University

THE PL COURSE

Course Staff

- My name: Gang (Gary) Tan, Westgate W358, 814-8657364, gtan@cse.psu.edu
  - Office hours: Wed 2-4pm or by appointment
  - Research interests: software security; programming languages
- Teaching Assistants
  - Mengran Fan (mxf97@psu.edu)
  - Ashley Huhman (abhuhman@gmail.com)
  - Office hours to be announced
- We also have two graders

Course Goals

- Non-goal: not necessarily making you an expert of any particular language
- Appreciate history and diversity of ideas in programming
  - We will study different ways of programming
    - functional programming (Scheme)
  - Why do we study "non-mainstream" languages?
    - 1970s dominant language Fortran has no recursive functions; now recursion is in every language
    - Garbage collection: introduced by LISP; popularized by Java
  - Moral
    - Futuristic ideas may be part of languages you use tomorrow, may even be useful problem-solving methods now

Course Goals

- Be able to pick up the best language for your app
  - Understand the languages you use, by comparison
    - Comparisons between functional, imperative, object-oriented programming
  - Develop a way of critical thinking
    - Properties of language, not syntax or sales pitch (Javascript)
- The ultimate goal is to give you the ability to learn a new programming language independently

Languages we will discuss

- Imperative programming: C
- OO programming: Java, Python
- Functional programming: Scheme (a variant of LISP)

- A few programming projects
- Some homework assignments

Lecture and Exams

- Lecture format
  - A combination of blackboard and slides
  - Note: slides won't include everything we will discuss in class
- Exams
  - 2 midterm exams
  - One final exam
  - No practice exams; questions will be similar to those in homework and discussed in class
- Some unannounced in-class quizzes
  - Quizzes are ungraded
  - Based on clickers

Clickers

- i-Clicker remotes required for the class
  - Start after the regular add/drop deadline
- Remember to register your clicker at clickers.psu.edu
  - You have to do this once a year
- Used for random in-class quizzes
  - For each question, you get 1 point as long as you participate through your clicker (regardless of correctness)

Participation Score Calculation

- Scoring (a total of 5 points)
  - >80% of total points, 5
  - [75%,80%), 4
  - [70%,75%), 3
  - [65%,70%), 2
  - [60%,65%), 1
  - <60%, 0
- You get the full participation score as long as you get 80% of total points
  - If you forget to bring your clicker or miss a class because of an interview (or any other legit reason), we **will not** reward back your missed clicker points because of the 20% buffer

Course websites

- Canvas course site (canvas.psu.edu)
  - Slides
  - All homework assignments should be submitted there
- A course public website
  - Schedule and extra links posted there
- Discussion forum
  - In Piazza; will send out an announcement email about where to sign up

Prerequisites

- CMPSC 221 (OO programming) and CMPSC 360 (Discreet math)
- You must take these courses before this one
- CSE department will remove anyone who doesn't satisfy prerequisites this week

Academic Integrity

- Programming projects
  - you cannot borrow code from any other source, including the internet or other students
  - We run automatic plagiarism detection tools

CH1 INTRODUCTION

What is a Programming Language?

- An informal def: A PL tells a computer what to do
- However, gap between computers and PLs
  - Computers are physical devices: CPU, memory, display, keyboards, hard drive, …
  - Machine language/assembly language: what the computer understands
    - Also called native languages
    - Concepts closely related to machine resources
  - High-level programming languages
    - Abstract, machine-independent concepts that aid programming
    - If you think in Java, you think about classes, objects, fields, methods, types
- How do we close the gap?

A PL is a conceptual universe

- A programming language is a "conceptual universe" (Alan Perlis)
  - Framework for problem-solving
  - Useful concepts and programming methods
  - Each PL provides its own abstractions

Why so many languages?

- People have different philosophies about how a program should be written
  - Endless debate about which language is the best
  - Cultures matter; e.g., which PL does Apple prefer?
- Different app domains need different languages
  - Business domain: Cobol
  - Scientific computing: Fortran, C
  - Systems programming: C/C++
  - Education: BASIC, Pascal, Scheme, Java, Python
  - Web: JavaScript; PhP; …
  - Future domains: Cars, robots, …
- General-purpose vs domain-specific languages
  - General-purpose languages: Java, C, Scheme,
  - Domain-specific languages: SQL, Verilog (for hardware design)

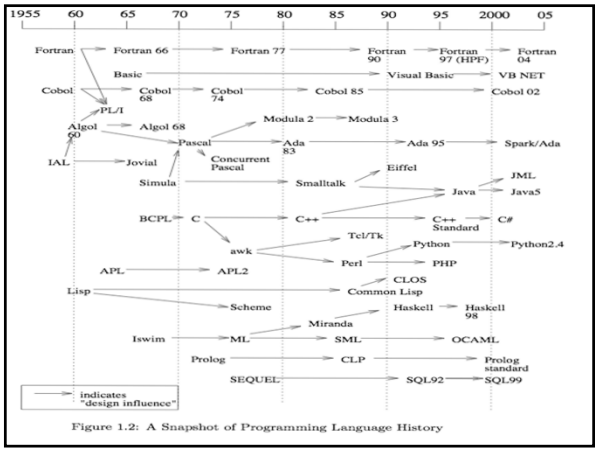
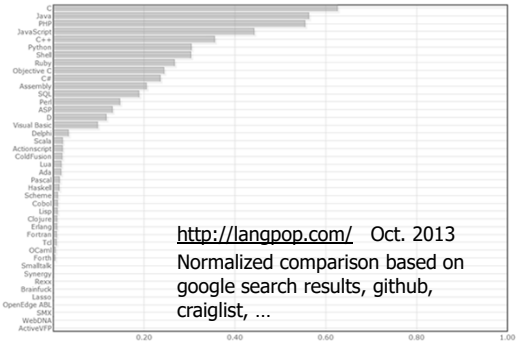


Figure 1.2: A Snapshot of Programming Language History

A Bit of History

- Early languages (1958-1960)
  - Fortran, Algol, Cobol, and LISP
- The road to C
  - From Algol 60, CPL, BCPL, B, C
- The road to Java
  - Simula, Smalltalk, C++, Oak, Java

Language Popularity Comparison



http://langpop.com/ Oct. 2013

Normalized comparison based on google search results, github, craiglist, …

Language = Syntax + Semantics + Design Philosophy

◆Syntax: specifies what valid programs are

```
class MyFirstJavaProg {
  public static void main(String args[]) {
      int x = 3 + 4;
      System.out.println("x= " + x);
  }
}
```

If we write + 3 4, then that's not a valid Java program, or not syntactically correct

Semantics

◆Semantics: dictates what a program does
- The meaning of a program
- Informal description: English description, by examples
  - E.g., the "Java language spec" book
- Formal specification
  - Denotational semantics; operational semantics; axiomatic semantics
  - Structural operational semantics: meaning of a program given by how it executes

Language Design Philosophy: Paradigms

- A **programming paradigm** is a style of programming
  - A single computing task can be accomplished in many ways
  - Different philosophies of how programs should accomplish a task leads to many programming paradigms

Major Programming Paradigms

◆Imperative programming
- Computation as a sequence of commands that change a program's state
- Example languages: C, Pascal

◆Object-oriented programming (OOP)
- Computation as objects and their interaction
- interaction: message-passing between objects for changing their states
- Example languages: Java, C++, Smalltalk

◆Functional Programming (FP)
- Computation as mathematical functions: input and output
- Pure FP: no notion of states
- Example languages: Lisp, ML, Haskell, Scheme

◆Logic Programming
- Computation using mathematical logical rules
- Rule-based programming
- Example language: Prolog

More Programming Paradigms

◆Aspect-Oriented Programming (AOP)

◆Dataflow languages

◆Scripting languages

◆…

◆A language usually uses a mix of those paradigms
- C++: mix of imperative and OO programming
- Scala: OO and functional programming

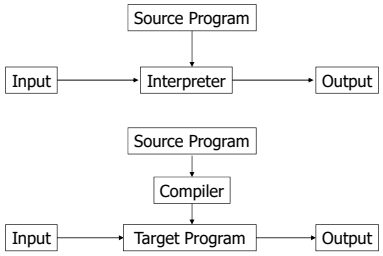
Go Through the Course Syllabus

COMPILATION AND INTERPRETATION

Programming Language Implementation

- Go from Syntax to Semantics

Interpreter vs Compiler

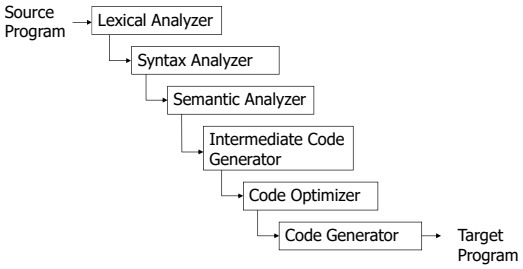


Compiler: transform program's syntax into machine instructions that can be executed to cause the correct sequence of actions to occur

Interpreter vs. Compiler

- Interpreter: easy to implement, but slow
  - Mix the translation and execution; translation performed multiple times on the same function if it is executed multiple times
  - Can be 10 times slower than the compiler
- Compiler: harder to implement, but more efficient
- Many language starts with an interpreter, then a compiler

Typical Compiler



See summary in textbook

Language virtual machines

- Java: a mixed mode
  - Java compiler produces instructions for an architecture-independent machine (Java bytecode)
  - JVM interprets these instructions to machine code
  - Additionally,
    - Most frequently used methods are compiled into native code
    - JIT compilation
- There are many other ways of mixing compilation and interpretation
  - See book Sec 1.4