Introduction

4190.310
Programming Languages
Spring 2014

Lecture 01

Course Introduction

Instructor

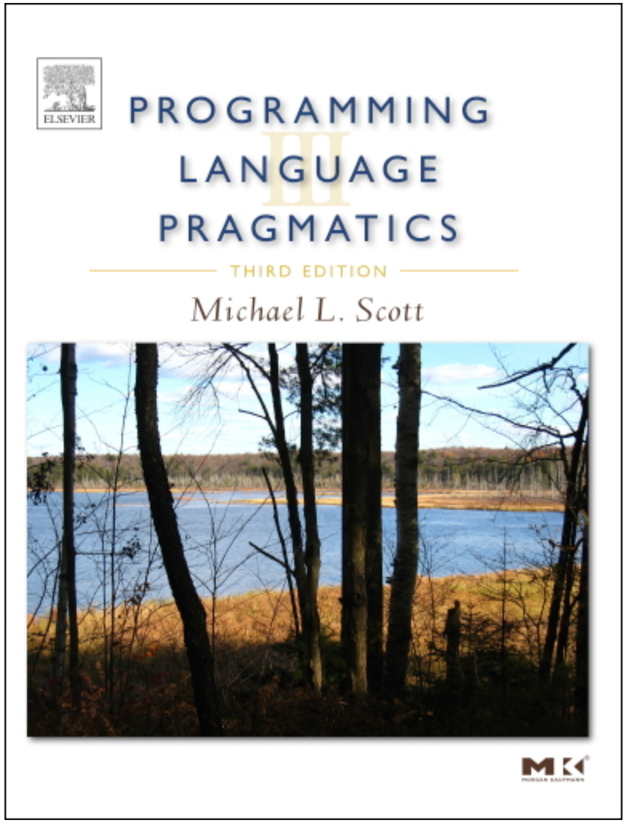
- Prof. Jaejin Lee
  - Email: jaejin@snu.ac.kr
  - URL: http://aces.snu.ac.kr/~jlee
  - Office: Room 505, Building 301
  - Office hours: Mon. and Wed. 13:30PM – 14:30PM
  - Phone: 880-1863

Teaching Assistant

- Wookeun Jung
  - Email: wookeun@aces.snu.ac.kr
  - Office: Room 520, Building 301
  - Office hours: TBD
- Yongjun Lee
  - Email: yongjun@aces.snu.ac.kr
  - Office: Room 520, Building 301
  - Office hours: TBD

Textbook

- Lecture slides
- Programming Language Pragmatics (third edition), Michael L. Scott, Elsevier



Attendance and Course URL

- Attendance
  - Students are required to attend class regularly
  - Attendance will be recorded and will affect your grades directly or indirectly
  - The decision is up to the instructor
- Course URL
  - http://etl.snu.ac.kr

Assignments (tentative)

- There will be approximately 7 homework and programming assignments
- At the beginning of the semester, each student has a total of 3 grace days that can be used as extension days for any assignments ***other than the last assignment***
- You can use all 3 days on one assignment or split them up across two or three assignments
- After you use all your 3 grace days, the late submission will not be accepted, and you will get a 0 on the assignment

Exams and Academic Integrity

- There will be an in-class midterm exam and a final exam
  - The exam time and locations will be announced later
- All assignments must be done from scratch
- The solutions of the problems must be your own work
- Any sort of cheating is not allowed
- We expect all students to adhere to Seoul National University's school regulations on integrity of scholarship and grades

Grading (tentative)

- Final grades will be based on the following:
    - 10% Class attendance and participation
    - 30% Homework and programming assignments
    - 30% Midterm Exam
    - 30% Final Exam

Other Policies

- The course website will reflect all modifications
  - The instructor and TAs will use the website to notify you of important changes
  - You are responsible for checking your email and the website regularly
- Failure to take an examination at the scheduled time will result in a 0 for the examination except in the cases of documented emergency
  - You should discuss with the instructor any extenuating circumstances that impact on your participation in the course as soon as those circumstances are known

Note

- Lecture notes are based on the author's (Michael scott) lecture slides
  - Copyrighted by Elsevier
- Most of the figures in the lecture notes are also copyrighted by Elsevier

Introduction

Reading Assignments:

Chapter 1. Introduction

Sections 1.1, 1.2, 1.3, 1.4, and 1.5

Why So Many Programming Languages?

- Evolution
  - Learned better ways of doing things over time
- Socio-economic factors
  - Proprietary interests
  - Commercial advantages
- Orientation toward special purposes
- Orientation toward special hardware
- Diverse ideas about what is pleasant to use

Successful Programming Languages?

- Easy to learn
  - BASIC, Pascal, LOGO, Scheme, Python
- Easy to express things, easy to use once fluent, and powerful
  - C, Common Lisp, APL, Algol-68, Perl
- Easy to implement
  - BASIC, Forth
- Possible to compile to very good (fast/small) code
  - Fortran
- 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?

- Way of thinking
  - Way of expressing algorithms
- Abstraction of a virtual machine
  - Way of specifying what you want
- Need languages from the user's point of view

Why do We Study Programming Languages?

- Help you choose a language
- Make it easier to learn new languages
- Help you make better use of whatever language you use

Help You Choose a Language

- C vs. Modula-3 vs. C++ for systems programming
- Fortran vs. APL vs. Ada for numerical computations
- 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

- Some languages are similar
- Concepts have even more similarity
  - Iteration, recursion, and abstractions found in different languages
- It easier to assimilate the syntax and semantic details of a new language than if you try to pick it up in a vacuum

Help You Make Better Use of Whatever Language You Use

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

Help You Make Better Use of Whatever Language You Use (contd.)

- Understand implementation costs
  - Choose between alternative ways of doing things, based on knowledge of what will be done underneath:
    - Use simple arithmetic equal (use x*x instead of x**2)
    - Use C pointers 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

Help You Make Better Use of Whatever Language You Use (contd.)

- Figure out how to do things in languages that do not support them explicitly
  - Lack of suitable control structures in Fortran
    - Use comments and programmer discipline for control structures
  - Lack of recursion in Fortran, CSP, etc.
    - Write a recursive algorithm then use mechanical recursion elimination (even for things that are not 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
    - Fake them with functions

Imperative vs. Declarative

- Declarative
  - Focus on what the computer is to do
- Imperative
  - Focus on how the computer should do

Imperative vs. Declarative (contd.)

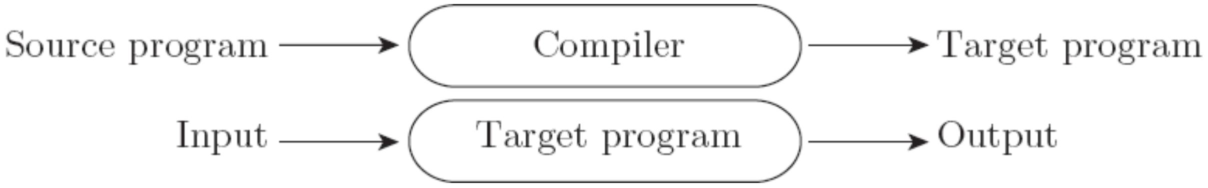
- Imperative
  - von Neumann
    - Fortran, Pascal, Basic, C, Ada, …
  - object-oriented
    - Smalltalk, Eiffel, C++, Java, …
  - scripting languages
    - Perl, Python, JavaScript, PHP, …
- Declarative
  - functional
    - Scheme, ML, Lisp, Haskell, FP, …
  - logic, constraint-based
    - Prolog, VisiCalc, RPG, …

Imperative vs. Declarative (contd.)

- Imperative languages, particularly the von Neumann languages, predominate
  - They will occupy the bulk of our attention
- We also plan to spend some time on functional, logic languages

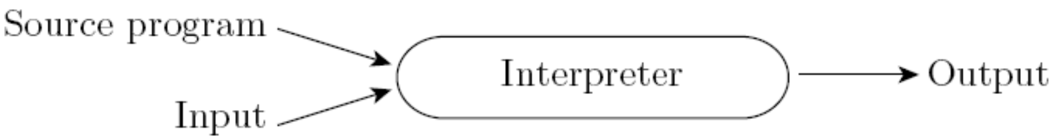
Compilation vs. Interpretation

- Not opposites
- Not a clear-cut distinction
- Pure compilation
  - The compiler translates the high-level source program into an equivalent target program (typically in machine language), and then goes away



Compilation vs. Interpretation (contd.)

- Pure interpretation
  - Interpreter stays around for the execution of the program
  - Interpreter is the locus of control during execution

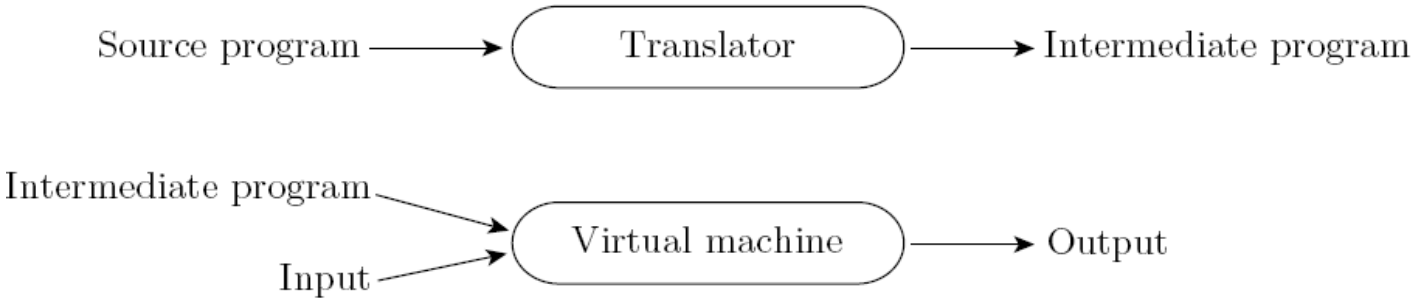


Compilation vs. Interpretation (contd.)

- Interpretation
  - Greater flexibility
  - Better diagnostics (error messages)
- Compilation
  - Better performance

Compilation vs. Interpretation (contd.)

- Common case is compilation or simple pre-processing, followed by interpretation
- Most language implementations include a mixture of both compilation and interpretation



Compilation vs. Interpretation (contd.)

- Note that compilation does not have to produce machine language for some sort of hardware

- Compilation is translation from one language into another, with full analysis of the meaning of the input

- Compilation entails semantic understanding of what is being processed
  - Pre-processing does not; a pre-processor will often let errors through