Programming Languages

Introduction
(Based on [Sethi 1996])

Yih-Kuen Tsay

Department of Information Management
National Taiwan University

What They Are

- *Programming languages* are notations for specifying, organizing, and reasoning about computations.
- According to Stroustrup, a programming language is
  - a tool for instructing machines,
  - a means for communicating between programmers,
  - a vehicle for expressing high-level designs,
  - a notation for algorithms,
  - a way of expressing relationships between concepts,
  - a tool for experimentation, and
  - a means for controlling computerized devices.

Machines, Machine Language, and Assembly Language

- Programming languages were invented to make machines easier to use.
- Machine computations are low level, more about the inner workings of the machine rather than what the computation is for.
- *Machine language* is the native language to which a computer responds directly.
- However, programs in machine language (consisting only of 0's and 1's) is unintelligible to a human.
- *Assembly language* is a variant of machine language in which names and symbols take the place of the actual codes for machine operations, values, and storage locations.

Assembly Code

Program

```
 1 :  M[0] := 0
 2 :  read(M[1])
 3 :  if M[1] ≥ 0 then goto 5
 4 :  goto 7
 5 :  M[3] := M[0] − M[1]
 6 :  if M[3] ≥ 0 then goto 16
 7 :  write(M[1])
 8 :  read(M[2])
 9 :  M[3] := M[2] − M[1]
10 :  if M[3] ≥ 0 then goto 12
11 :  goto 14
12 :  M[3] := M[1] − M[2]
13 :  if M[3] ≥ 0 then goto 8
14 :  M[1] := M[2] + M[0]
15 :  goto 3
16 :  halt
```

Assembly Code (cont.)

If we are allowed the following conditionals, the code can become more readable.

- **if** $M[j] = 0$ **then goto** i
- **if** $M[j] = M[k]$ **then goto** i

Program

```
1 :  M[0] := 0
2 :  read(M[1])
3 :  if M[1] = 0 then goto 9
4 :  write(M[1])
5 :  read(M[2])
6 :  if M[2] = M[1] then goto 5
7 :  M[1] := M[2] + M[0]
8 :  goto 3
9 :  halt
```

Toward Higher-Level Languages

- Language designers seek a balance between two goals:
  - making computing convenient for people
  - making efficient use of computing machines
- Convenience comes first. Without it, efficiency is irrelevant.
- Programming languages were invented to make machines easier to use. They thrive because they make problems easier to solve.
- Programming languages are designed to be both higher level and general purpose.
  - A language is *higher level* if it is independent of the underlying machine.
  - A language is *general purpose* if it can be applied to a wide range of problems.

Benefits of Higher-Level Languages

Higher-level languages have replaced machine language and assembly language in virtually all areas of programming, because they provide benefits like the following:

- Readable, familiar notations
- Machine independence (portability)
- Availability of program libraries
- Consistency checks during implementation that can detect errors

Problems of Scale

- The problems of programming are ones of scale.
- Any one change to a program is easy to make.
- But, the effect of a change can ripple through the program, perhaps introducing errors or bugs into some forgotten corner.
- Programming languages can help in two ways:
  - Their readable and compact notations reduce the likelihood of errors.
  - They provide ways of organizing computations so that they can be understood one piece at a time.

Problems of Scale (cont.)

- *Code inspection* and *program testing* are two common techniques for detecting program errors.
- But as Dijkstra said, *program testing can be used to show the presence of bugs, but never to show their absence*.
- We must organize the computations in such a way that our limited powers are sufficient to guarantee that the computation will establish the desired effect.

Programming Paradigms

- *Imperative Programming*
  Imperative languages are action oriented; that is, a computation is viewed as a sequence of actions. They include Fortran, Algol, Pascal, C, etc.
- *Functional Programming*
  Simply put, functional programming is programming without assignments. Functional programming languages include Lisp, Scheme, ML, etc.
- *Object-Oriented Programming*
  Central to object-oriented programming is the concept of objects and their classification into classes and subclasses.
  Object-oriented programming languages include Smalltalk, C++, Java, etc.
- *Concurrent Programming*
- *Logic Programming*

Language Implementation

There are two basic approaches to implementing a program in a higher-level language:

- Compilation
  The language is brought down to the level of the machine, using a translator called a *compiler*.
- Interpretation
  The machine is brought up to the level of the language, by building a higher-level machine called an *interpreter*.

Compilation vs. Interpretation

- Compilation is biased toward static properties, while interpretation can deal with dynamic properties. They can be compared as follows.
- *Compilation can be more efficient than interpretation*.
  - Unlike a compiler, which translates the source program once and for all, an interpreter examines the program repeatedly.
- *Interpretation can be more flexible than compilation*.
  - An interpreter allows programs to be changed "on the fly" to add features or correct errors.
  - It can also pinpoint an error in the source text and report it accurately.