Principles of Programming Languages
COMP3031: Introduction

Prof. Dekai Wu

Human Language Technology Center
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

Fall 2012
Why Study PLs?

- Hundreds of different PLs have been designed and implemented.
- They may be grouped into different families of PLs.
- We are not surveying PLs, but studying the programming concepts and constructs behind the different designs.

**Goal:**

- Improve your understanding of the language you are using.
- Systematically learn the various programming concepts and constructs.
- Help you learn a new language.
- Make it easier to design a new language.
- Allow a better choice of programming language.
How About Human Languages?

- **Chinese vs. English:**
  - pictorial (WYSIWYG) vs. phonetic
  - hieroglyphic vs. alphabetical

- **Japanese vs. English:**
  - wa-ta-shi-wa ni-hon-go wa-ka-ri ma-sen.
  - I Japanese understand don’t.

An intriguing question: Do the differences in human language designs reflect how differently people think?
Development of Human Languages

1st written language: Sumerian, 3500 B.C. (c.f. Chinese, Shang Dynasty, 2000 B.C.)

![SumerianLogos](image.png)
1st alphabet: Phoenician, 1100 B.C.; only consonants.

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>日</td>
<td>王</td>
<td>大</td>
<td>甲</td>
<td>万</td>
<td>人</td>
<td>六</td>
<td>大</td>
</tr>
<tr>
<td>hēt</td>
<td>zayin</td>
<td>wāw</td>
<td>hē</td>
<td>dālet</td>
<td>gīmel</td>
<td>bēt</td>
<td>'ālef</td>
</tr>
<tr>
<td>sāmek</td>
<td>nun</td>
<td>mēm</td>
<td>lāmed</td>
<td>kaf</td>
<td>yōd</td>
<td>tēt</td>
<td></td>
</tr>
<tr>
<td>tāw</td>
<td>šin/šin</td>
<td>rēš</td>
<td>qōf</td>
<td>šādē</td>
<td>pē</td>
<td>'ayin</td>
<td></td>
</tr>
</tbody>
</table>
1st complete alphabet: Greek, 800 B.C.; consonants + vowels.

<table>
<thead>
<tr>
<th>Character</th>
<th>Ionia</th>
<th>Athens</th>
<th>Corinth</th>
<th>Argos</th>
<th>Euboea</th>
<th>Modern</th>
<th>AP</th>
<th>MP</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>ΑΑ</td>
<td>ΑΑ</td>
<td>ΑΑ</td>
<td>ΑΑ</td>
<td>ΑΑ</td>
<td>Αα</td>
<td>[a]</td>
<td>[a]</td>
</tr>
<tr>
<td>beta</td>
<td>ΒΒ</td>
<td>ΒΒ</td>
<td>ΒΒ</td>
<td>ΒΒ</td>
<td>Ββ</td>
<td>[b]</td>
<td>[b]</td>
<td></td>
</tr>
<tr>
<td>gamma</td>
<td>ΓΓ</td>
<td>ΛΛ</td>
<td>ΚΚ</td>
<td>ΝΝ</td>
<td>ΡΡ</td>
<td>Γγ</td>
<td>[γ]</td>
<td>[γ]</td>
</tr>
<tr>
<td>delta</td>
<td>ΔΔ</td>
<td>ΔΔ</td>
<td>ΔΔ</td>
<td>ΔΔ</td>
<td>Δδ</td>
<td>Δδ</td>
<td>[δ]</td>
<td>[δ]</td>
</tr>
<tr>
<td>epsilon</td>
<td>ΕΕ</td>
<td>ΕΕ</td>
<td>ΕΕ</td>
<td>ΕΕ</td>
<td>Εε</td>
<td>Εε</td>
<td>[ε]</td>
<td>[ε]</td>
</tr>
<tr>
<td>digamma</td>
<td>ΙΙ</td>
<td>ΙΙ</td>
<td>ΙΙ</td>
<td>ΙΙ</td>
<td>Ιζ</td>
<td>Ζζ</td>
<td>[ζ]</td>
<td>[ζ]</td>
</tr>
<tr>
<td>zeta</td>
<td>ΞΞ</td>
<td>ΞΞ</td>
<td>ΞΞ</td>
<td>ΞΞ</td>
<td>Ξξ</td>
<td>Ξξ</td>
<td>[ξ]</td>
<td>[ξ]</td>
</tr>
<tr>
<td>etta</td>
<td>ΗΗ</td>
<td>ΗΗ</td>
<td>ΗΗ</td>
<td>ΗΗ</td>
<td>Ηη</td>
<td>Ηη</td>
<td>[η]</td>
<td>[η]</td>
</tr>
<tr>
<td>kappa</td>
<td>ΚΚ</td>
<td>ΚΚ</td>
<td>ΚΚ</td>
<td>ΚΚ</td>
<td>Κκ</td>
<td>[κ]</td>
<td>[κ]</td>
<td></td>
</tr>
<tr>
<td>lambda</td>
<td>ΛΛ</td>
<td>ΛΛ</td>
<td>ΛΛ</td>
<td>ΛΛ</td>
<td>Λλ</td>
<td>Λλ</td>
<td>[λ]</td>
<td>[λ]</td>
</tr>
<tr>
<td>mu</td>
<td>ΜΜ</td>
<td>ΜΜ</td>
<td>ΜΜ</td>
<td>ΜΜ</td>
<td>Μμ</td>
<td>Μμ</td>
<td>[μ]</td>
<td>[μ]</td>
</tr>
<tr>
<td>nu</td>
<td>ΝΝ</td>
<td>ΝΝ</td>
<td>ΝΝ</td>
<td>ΝΝ</td>
<td>Νν</td>
<td>Νν</td>
<td>[ν]</td>
<td>[ν]</td>
</tr>
<tr>
<td>xi</td>
<td>ΞΞ</td>
<td>ΞΞ</td>
<td>ΞΞ</td>
<td>ΞΞ</td>
<td>Ξξ</td>
<td>Ξξ</td>
<td>[ξ]</td>
<td>[ξ]</td>
</tr>
<tr>
<td>omicron</td>
<td>ΟΟ</td>
<td>ΟΟ</td>
<td>ΟΟ</td>
<td>ΟΟ</td>
<td>Οο</td>
<td>Οο</td>
<td>[ο]</td>
<td>[ο]</td>
</tr>
<tr>
<td>pi</td>
<td>ΠΠ</td>
<td>ΠΠ</td>
<td>ΠΠ</td>
<td>ΠΠ</td>
<td>Ππ</td>
<td>Ππ</td>
<td>[π]</td>
<td>[π]</td>
</tr>
<tr>
<td>san</td>
<td>ΜΜ</td>
<td>ΜΜ</td>
<td>ΜΜ</td>
<td>ΜΜ</td>
<td>Μσ</td>
<td>Μσ</td>
<td>[σ]</td>
<td>[σ]</td>
</tr>
<tr>
<td>kappa</td>
<td>ΦΦ</td>
<td>ΦΦ</td>
<td>ΦΦ</td>
<td>ΦΦ</td>
<td>Φφ</td>
<td>Φφ</td>
<td>[φ]</td>
<td>[φ]</td>
</tr>
<tr>
<td>rho</td>
<td>ΡΡ</td>
<td>ΡΡ</td>
<td>ΡΡ</td>
<td>ΡΡ</td>
<td>Ρρ</td>
<td>Ρρ</td>
<td>[ρ]</td>
<td>[ρ]</td>
</tr>
<tr>
<td>sigma</td>
<td>ΣΣ</td>
<td>ΣΣ</td>
<td>ΣΣ</td>
<td>ΣΣ</td>
<td>Σσ</td>
<td>Σσ</td>
<td>[σ]</td>
<td>[σ]</td>
</tr>
<tr>
<td>tau</td>
<td>ΤΤ</td>
<td>ΤΤ</td>
<td>ΤΤ</td>
<td>ΤΤ</td>
<td>Ττ</td>
<td>Ττ</td>
<td>[τ]</td>
<td>[τ]</td>
</tr>
<tr>
<td>upsilon</td>
<td>ΨΨ</td>
<td>ΨΨ</td>
<td>ΨΨ</td>
<td>ΨΨ</td>
<td>Ψψ</td>
<td>Ψψ</td>
<td>[ψ]</td>
<td>[ψ]</td>
</tr>
<tr>
<td>phi</td>
<td>ΦΦ</td>
<td>ΦΦ</td>
<td>ΦΦ</td>
<td>ΦΦ</td>
<td>Φφ</td>
<td>Φφ</td>
<td>[φ]</td>
<td>[φ]</td>
</tr>
<tr>
<td>khi</td>
<td>ΧΧ</td>
<td>ΧΧ</td>
<td>ΧΧ</td>
<td>ΧΧ</td>
<td>Χχ</td>
<td>Χχ</td>
<td>[χ]</td>
<td>[χ]</td>
</tr>
<tr>
<td>psi</td>
<td>ΨΨ</td>
<td>ΨΨ</td>
<td>ΨΨ</td>
<td>ΨΨ</td>
<td>Ψψ</td>
<td>Ψψ</td>
<td>[ψ]</td>
<td>[ψ]</td>
</tr>
<tr>
<td>omega</td>
<td>ΩΩ</td>
<td>ΩΩ</td>
<td>ΩΩ</td>
<td>ΩΩ</td>
<td>Ωω</td>
<td>Ωω</td>
<td>[ω]</td>
<td>[ω]</td>
</tr>
</tbody>
</table>
What’s a PL for?

Stroustrup (C++ designer, 1994):

- tool for instructing machines?
- means for communicating between programmers?
- vehicle for expressing high level designs?
- notation for algorithms?
- way of expressing relationships between concepts?
- tool for experimentation?
- means for controlling computerized devices?
- collection of “neat” features?

His answer: All of the above except the last one.
main:

!#PROLOGUE# 0
save %sp,-128,%sp

!#PROLOGUE# 1
mov 1,%o0
st %o0,[%fp-20]
mov 2,%o0
st %o0,[%fp-24]
ld [%fp-20],%o0
ld [%fp-24],%o1
add %o0,%o1,%o0
st %o0,[%fp-28]
mov 0,%i0
nop
#include <stdio.h>

int main()
{
    int x, y, z;

    x = 1;
    y = 2;
    z = x+y;

    return 0;
}

Levels of PLs

- **machine (binary) language** is unintelligible
- **assembly language** is low level
  - mnemonic names for machine operations
  - explicit manipulation of memory addresses/contents
  - machine dependent
- **high level language**
  - readable
    - instructions are easy to remember
    - faster coding
    - less error-prone (fewer bugs?)
    - easier to maintain
  - no mention of memory locations
  - machine independent = portable
Genealogy of Common PLs

Prof. Dekai Wu, HKUST (dekai@cs.ust.hk)
4 Paradigms of PL Design

- Procedural Programming (PP) or Imperative Programming (IP)
  - See http://en.wikipedia.org/wiki/Procedural_programming
- Object-Oriented Programming (OOP)
- Declarative Programming
  - Functional Programming (FP)
  - Logic Programming (LP)

PL design is a balance among:
- efficiency
- readability
- support
- taste!
<table>
<thead>
<tr>
<th>IP/PP</th>
<th>OOP</th>
<th>FP</th>
<th>LP</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORTRAN</td>
<td>Smalltalk</td>
<td>LISP</td>
<td>Prolog</td>
</tr>
<tr>
<td>Pascal</td>
<td>C++</td>
<td>Scheme</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Java</td>
<td>SML</td>
<td></td>
</tr>
<tr>
<td>action-oriented</td>
<td>object-oriented</td>
<td>function-oriented</td>
<td>logic-oriented</td>
</tr>
<tr>
<td>procedural assignments</td>
<td>classes</td>
<td>functions</td>
<td>logic reasoning</td>
</tr>
<tr>
<td>algorithm design</td>
<td>system building</td>
<td>formal specification</td>
<td>expert system</td>
</tr>
<tr>
<td>compile</td>
<td>compile</td>
<td>interpret</td>
<td>database queries</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **FORTRAN**, **Pascal**, **C**: procedural
- **Smalltalk**, **C++**, **Java**: object-oriented
- **LISP**, **Scheme**, **SML**: function-oriented
- **Prolog**: logic-oriented

**mapping**: $x \rightarrow f(x)$

- **Are you sick?**

**Algorithm design**, **System building**, **Reusable software**: formal specification, program correctness

**Compile**, **Interpret**: compile, interpret
A **compiler** translates source programs into machine codes that run directly on the target computer. e.g. a.c $\rightarrow$ a.out.

- static codes
- compile once, run many
- optimized codes $\Rightarrow$ more efficient
- examples: FORTRAN, Pascal, C++
An **interpreter** is a virtual machine implemented on a target computer which runs a source program directly.

- Slower
- Interpret many, run many
- Interactive mode: easy debugging
- More flexible: allow programs to be changed “on the fly”
- Examples: many script languages (sh, csh, tcl, awk), ML, PROLOG
A hybrid system translates high-level source programs to an intermediate language which then allows fast and easy interpretation.

- compile once, interpret many
- Examples: UCSD Pascal, Perl, Python, Java
There are hundreds of different PLs.

It is easier to write (large) programs with a high-level PL.

Will emphasize the basic programming concepts/constructs.

Will address 4 programming paradigms: IP/PP, OOP, FP, LP.

2 approaches to types within the FP paradigm: latent typing vs. static typing with type inference.

2 ways to implement PLs: compilation vs. interpretation.