Principles of Programming Languages

COMP251: Introduction

Prof. Dekai Wu

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Hong Kong, China

Fall 2006
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?
1st written language: Sumerian, 3500 B.C. 
(c.f. Chinese, Shang Dynasty, 2000 B.C.)
1st alphabet: Phoenician, 1100 B.C.; only consonants.

<table>
<thead>
<tr>
<th>LETTER</th>
<th>PHONETIC</th>
<th>SOUND</th>
</tr>
</thead>
<tbody>
<tr>
<td>ℰ</td>
<td>hēt</td>
<td>h</td>
</tr>
<tr>
<td>ℱ</td>
<td>zayin</td>
<td>z</td>
</tr>
<tr>
<td>ℰ</td>
<td>wāw</td>
<td>w</td>
</tr>
<tr>
<td>ℰ</td>
<td>hē</td>
<td>h</td>
</tr>
<tr>
<td>ℰ</td>
<td>dālet</td>
<td>d</td>
</tr>
<tr>
<td>ℰ</td>
<td>gīmel</td>
<td>g</td>
</tr>
<tr>
<td>ℰ</td>
<td>bēt</td>
<td>b</td>
</tr>
<tr>
<td>ℰ</td>
<td>’ālef</td>
<td>’</td>
</tr>
<tr>
<td>ℰ</td>
<td>sāmek</td>
<td>s</td>
</tr>
<tr>
<td>ℰ</td>
<td>nun</td>
<td>n</td>
</tr>
<tr>
<td>ℰ</td>
<td>mēm</td>
<td>m</td>
</tr>
<tr>
<td>ℰ</td>
<td>lāmed</td>
<td>l</td>
</tr>
<tr>
<td>ℰ</td>
<td>kaf</td>
<td>k</td>
</tr>
<tr>
<td>ℰ</td>
<td>yōd</td>
<td>y</td>
</tr>
<tr>
<td>ℰ</td>
<td>tēt</td>
<td>t</td>
</tr>
<tr>
<td>ℰ</td>
<td>tāw</td>
<td>t</td>
</tr>
<tr>
<td>ℰ</td>
<td>šin/šin</td>
<td>š</td>
</tr>
<tr>
<td>ℰ</td>
<td>rēš</td>
<td>r</td>
</tr>
<tr>
<td>ℰ</td>
<td>qōf</td>
<td>q</td>
</tr>
<tr>
<td>ℰ</td>
<td>šādē</td>
<td>š</td>
</tr>
<tr>
<td>ℰ</td>
<td>pē</td>
<td>p</td>
</tr>
<tr>
<td>ℰ</td>
<td>’ayin</td>
<td>’</td>
</tr>
</tbody>
</table>
1st complete alphabet: Greek, 800 B.C.; consonants + vowels.
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 (Sethi 1996)
4 Paradigms of PL Design

- Imperative Programming (IP) or Procedural Programming (PP)
  - 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</td>
<td>classes</td>
<td>functions</td>
<td>logic reasoning</td>
</tr>
<tr>
<td>assignments</td>
<td>inheritance</td>
<td>mapping: $x \rightarrow f(x)$</td>
<td>“Are you sick?”</td>
</tr>
<tr>
<td>algorithm design</td>
<td>system building</td>
<td>formal specification</td>
<td>expert system</td>
</tr>
<tr>
<td></td>
<td>reusable software</td>
<td>program correctness</td>
<td>database queries</td>
</tr>
<tr>
<td>compile</td>
<td>compile</td>
<td>interpret</td>
<td>interpret</td>
</tr>
</tbody>
</table>
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

Prof. Dekai Wu, HKUST (dekai@cs.ust.hk)
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.