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 2007

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*

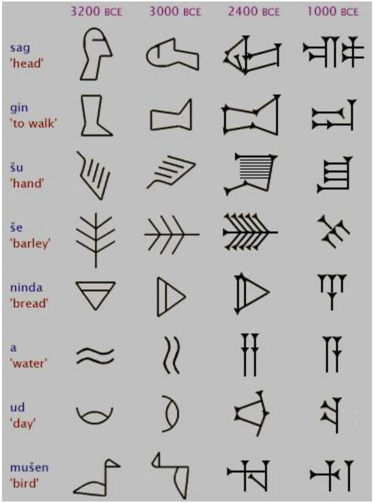
- *Japanese* vs. *English*:

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?

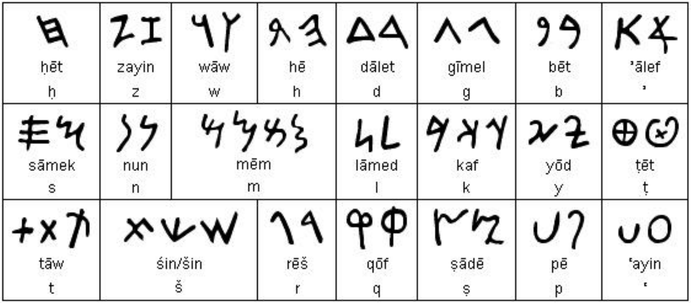
Development of Human Languages

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



Development of Human Languages ..

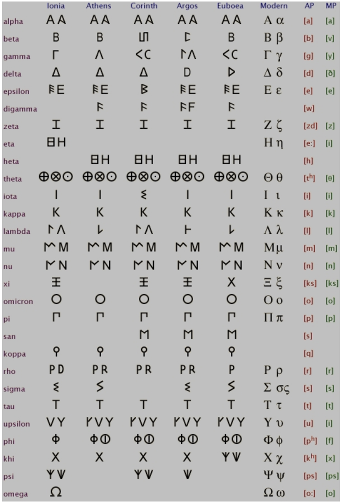
1st alphabet: Phoenician, 1100 B.C.; only consonants.



ḥēt ḥ	zayin z	wāw w	hē h	dālet d	gīmel g	bēt b	ʾālef ʾ
sāmek s	nun n	mēm m		lāmed l	kaf k	yōd y	ṭēt ṭ
tāw t	śin/šin š		rēš r	qōf q	ṣādē ṣ	pē p	ʿayin ʿ

Development of Human Languages ...

1st complete alphabet: Greek, 800 B.C.; consonants + vowels.



	Ionia	Athens	Corinth	Argos	Euboea	Modern	AP	MP
alpha	A A	A A	A A	A A	A A	Α α	[a]	[a]
beta	B	B	Ш	ᑕ	B	Β β	[b]	[v]
gamma	Γ	Λ	<C	↾Λ	<C	Γ γ	[g]	[γ]
delta	Δ	Δ	Δ	D	▷	Δ δ	[d]	[ð]
epsilon	𐌄E	𐌄E	Ᏸ	𐌄E	𐌄E	Ε ε	[e]	[e]
digamma		Ϝ	Ϝ	ϜF	Ϝ		[w]	
zeta	Ⲓ	Ⲓ	Ⲓ	Ⲓ	Ⲓ	Ζ ζ	[zd]	[z]
eta	⊟H					Η η	[e:]	[i]
heta		⊟H	⊟H	⊟H	⊟H		[h]	
theta	⊕⊗⊙	⊕⊗⊙	⊕⊗⊙	⊕⊗⊙	⊕⊗⊙	Θ θ	$[t^h]$	[θ]
iota	I	I	Ƹ	I	I	Ι ι	[i]	[i]
kappa	K	K	K	K	K	Κ κ	[k]	[k]
lambda	↾Λ	ᒪ	↾Λ	⊦	ᒪ	Λ λ	[l]	[l]
mu	ᛗM	ᛗM	ᛗM	ᛗM	ᛗM	Μμ	[m]	[m]
nu	ͶN	ͶN	ͶN	ͶN	ͶN	Ν ν	[n]	[n]
xi	⌶		⌶	⌶	X	Ξ ξ	[ks]	[ks]
omicron	O	O	O	O	O	Ο ο	[o]	[o]
pi	Ր	Ր	Ր	Ր	Ր	Π π	[p]	[p]
san			M	M	M		[s]	
koppa	Ϙ	Ϙ	Ϙ	Ϙ	Ϙ		[q]	
rho	PD	PR	PR	PR	P	Ρ ρ	[r]	[r]
sigma	Ƹ	ϟ		Ƹ	ϟ	Σ σς	[s]	[s]
tau	T	T	T	T	T	Τ τ	[t]	[t]
upsilon	VY	ᚴVY	ᚴVY	ᚴVY	ᚴVY	Υ υ	[u]	[i]
phi	Φ	Φ⦶	Φ⦶	Φ⦶	Φ⦶	Φ ϕ	$[p^h]$	[f]
khi	X	X	X	X	ΨѰ	Χ χ	$[k^h]$	[x]
psi	ΨѰ		ΨѰ	Ѱ		Ψ ψ	[ps]	[ps]
omega	Ω					Ω ω	[o:]	[o]

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.

Can You Understand This?

00001001001011100110011001101001011011000110010100001001001000100110110001100101011000110111101000111
01010111001001100101001100010010111001100011001000100000101001100111011000110110001100110010010111111
01100011011011110110110101110000011010010110110001100101011001000010111000111010000010100010111001111
00110110010101100011011110100011010010110111101101110000010010010001000101110011101000110010101111000
01110100001000100000101000000100100101110011000010110110001101001011001110110111000100000001101000000
10100000100100101110011001110110110001101111011000100110000101101100001000000110110101100000101101001
01101110000010100000100100101110011101000111100101110000011001010000100100100000011011010110000010110
10010110111000101100001000110110011001110101011011100110001101110100011010010110111101101110000010100
00001001001011100111000001110010011011110110001100001001001100000011010000000101001101101011000001011
01001011011100011101000000101000000100100100001001000110101000000101001001001111010011000100111101000111
01010101010001010010001100100000000110000000001010000001001011100110110000101110110011001010010000000100
01010111001101110000000101100001011010011000100110010001110000001011000010010101110011011100000000101
00000101000000100100100001001000110101000001010010010011110100110001001111010001110101010101010001010010
00110010000000110001000010100000010010110110101101111011101100010000000110001001011000010010101101111
00110000000001010000001001011100110111010000100000000100101011011110011000000010110001011011001001010110
01100111000000101101001100100011000000101110100001010000001001011011010110111101110110001000000001100101
00101100001001010110111100110000000010100000010010111001101110100001000000001001010110111100110000000101
11000101101100100101011001100111000000101101001100100011010001011101000010100000010010110110001100100
00100000001011011001001010110011001110000000101101001100100011000000101110100101100001001010110111100111
00000000101000000100101101100011001000010000000101101100100101011001100111000000101101001100100011010010
01011101001011000010010101101111001100010000101000000100101100001011001000110010000100000001001010110
11110011000000010110000100101011011110011000100101100001001010110111100110000000010100000010010111001111
01110100001000000010010101101111001100000001011000101101100100101011001100111000000101101001100100011
10000101110100001010000001001011011010110111101110110001000000001100000010110000100101011010010011000000
00001010000001001011000100010000000010111001001100010011000011000100001010000001001011011100110111101111
00000000101000101110010011000100110000110001001110100000101000000100101110010011001010111010000001010
00001001011100100110010101110011011110100011011110111001001100101000010100010111001001100010011000110
01100110010100110001001110100000101000000100100101110011100110110100101111010011001010000100100100000
01101101011000010110100101101110001011000010111001001100010011000110011001100101001100010010110101100
11010110000101101001011011100000101000000100100101110011010010110010001100101011011100111010000001001
00100010010001110100001101000011001110100010000000101000010001110100111001010101001010010010010000000111
0010001011100011100000101110001100010010001000001010

How About This?

```
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
```

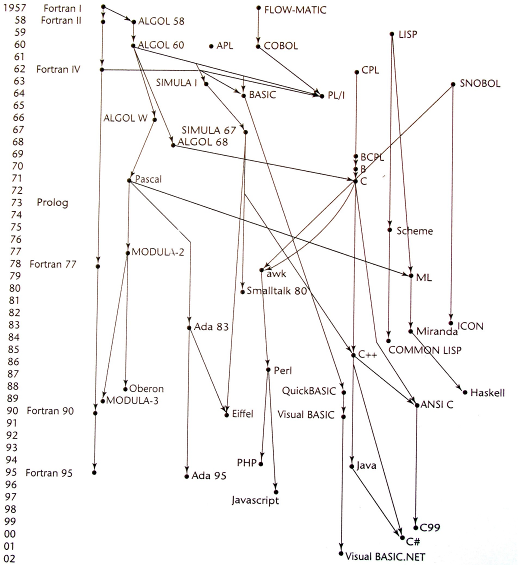
Is This Better Now?

```
#include <stdio.h>

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

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

    return 0;
}
```

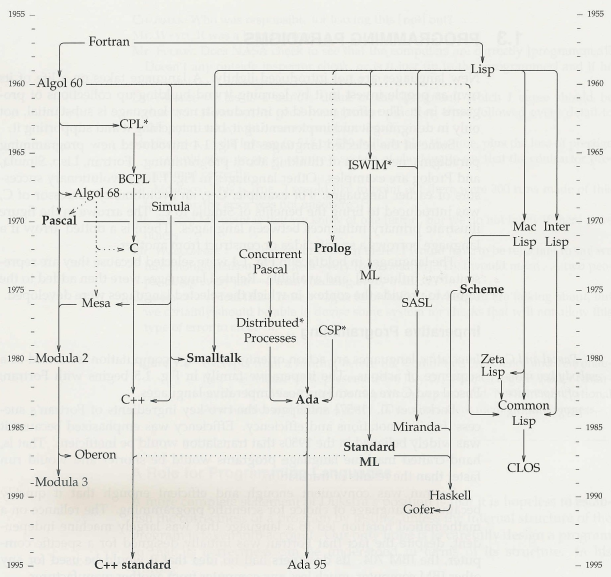
Levels of PLs

- 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



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!

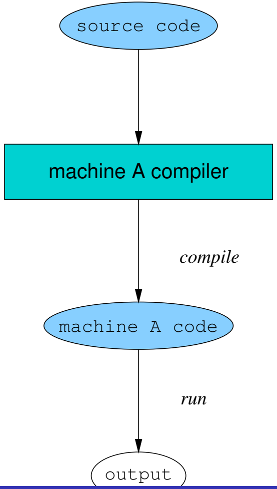
IP/PP, OOP, FP, LP

IP/PP	OOP	FP	LP
FORTRAN Pascal C	Smalltalk C++ Java	LISP Scheme SML	Prolog
action-oriented	object-oriented	function-oriented	logic-oriented
procedural assignments	classes inheritance	functions mapping: $x \longrightarrow f(x)$	logic reasoning “Are you sick?”
algorithm design	system building reusable software	formal specification program correctness	expert system database queries
compile	compile	interpret	interpret

Compilation: From Source to Runnable Program

A **compiler** translates source programs into machine codes that run directly on the target computer. e.g. a.c ⟶ a.out.

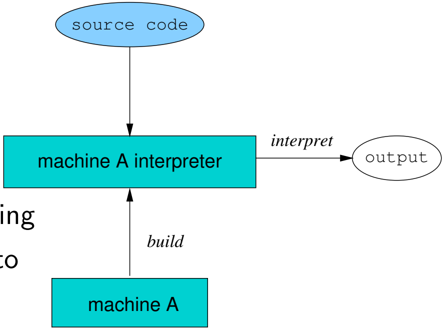
- static codes
- compile once, run many
- optimized codes
  ⇒ more efficient
- examples: FORTRAN, Pascal, C++



Interpretation: From Source to Program Output

An **interpreter** is a virtual machine implemented on a target computer which runs a source program directly.

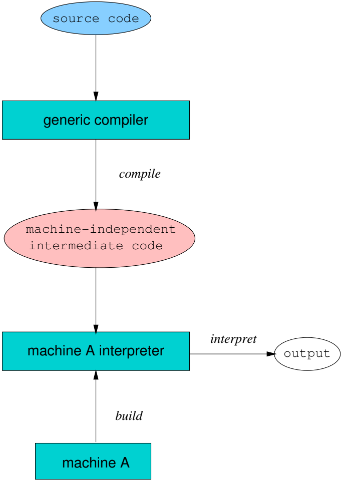
- slower
- interpret many, run many
- interactive mode: easy debugging
- more flexible: allow programs to be changed "on the fly"
- examples: many script languages (sh, csh, tcl, awk), ML, PROLOG



Hybrid Implementation System

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



Recapitulate

- √ 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.