

What is a PL?

= comm. human \rightarrow machine
human \rightarrow human
m/c \rightarrow m/c.

- hard wired plug boards?
- low level (asm)
- mid level (C) = portable asm
- high level (Prolog, ML) - details by compiler.
 - hides machine
 - readable
 - libs vs lang.
 - consist checks
 - portable.

von Neumann m/c - memory/store/CPU.

Computation?

~~no~~ Turing m/c. = sequence code.
 λ -calculus = eval fns.

lang: - special purpose "little" lang
- general purpose

Readability: - machine = efficiency
- efficient translation

- " execution.

- interpretation vs compilation
- compilable: unambiguous

- Human Rd. & writability

- abstraction

- modularity

- polymorphism

- size > any other engr discipline

10^3 - student asgt

10^6 - O.S or compile

10^9 - major system

Syntax
; $\{ \}$ nl

Zipf's law
Whorf's law

Software engineering

- multiple pgms
- multiple modules
- tools. (make, cvs, ...)

Louden
1 Intro(2)

1.2 ABSTRACTIONS

data } { basic
vs } { structured
control } { unit(module)

Data abstr: • basic = int | float, etc.
- locations in memory

• structured = arrays, lists
(strings: basic or struct?)
why type char

• unit = ADT | module ← class confuses these two.
- encapsulation
- info hiding
- reusable
- components
- containers (parameterized)
- libraries
- interoper.

CONTROL abstr

- Basic: assign, operators, ctrl transfer (goto)
- Structured: if/while/~~switch~~/for
- procedure/subroutine/function.
| \ declaration | invocation
| \ params: formals
| \ actuals (args)
runtime environment: calling seq.
call stack

break
continue
throw
return.

procedure: args \rightarrow results
or
none.

Louden
1 Intro (3)

sometimes multiple

ex: let $(x, y) = f(a, b)$
tuple.

- Unit - collection of procs sharing a unit
ex: Java methods of a class
entries in a module in C.
 - threads vs processes
 \uparrow shares heap; private stack.

1.3 Paradigms.

- Imperative (structured)

\uparrow \uparrow ~~if, while, recur~~
cond, loop, [recur] complete
call / procs.

EVIL: goto ~~pointers~~
pointers
global vars.

ex: C.

- von Neumann bottleneck
 - seq of simple insns
 - one datum @ a time
 - no concurrency.

- Object Oriented

- currently THE big one.
- imperative with notational wrapper
delayed binding

— 00 —

classes - instances

- methods
- fields

▷ dynamic dispatch & inheritance

$a \rightarrow f(x, y) \Rightarrow (a \rightarrow vft \rightarrow f)(a, x, y)$
(c++) (c) ↑ dispatch table.

duck
typing.

priv
prot
public friend
pkg

- ctor / dtor (or gc col)
- single, mixin, multiple
 ↑ J'interface

• Functional

- eval of fns
- λ calculus
- generally no side effects (asgts)
- fns as 1st class objects.

ex: let add x y = x + y ;
 let incr = add 1 ;
 ⇒ makes incr same as (1+)

~~let~~ ~~add x y~~ →
(add 3) 4 → 7
add 3 → fn: int → int

curry

($\lambda x. x + 1$)

- higher order fns
- parametric polymorphism
- type inference

ex: let fac n = match n with
 | 0 → 1
 | n when > 0 → n * f(n-1)
 | invalid_arg n ;

Louden

1 Intro (5).

- applicative langs.
 - order of arg eval not relevant
 - explicitly seq when needed.

$y = f(g(x), h(z));$

↑ ↑ no side effects.

- ~~strict (eager)~~ vs nonstrict (lazy)
 - ↑ call by value ↑ call by need.
 - Scheme Haskell.
 - O'Carroll ?:

• Logic - Declarative (Prolog)

- expert systems.
- unification of Horn clauses.

clause: $a :- b, c, d.$

• Scripting, Tools, "Little Languages"

- shell scripts (Bourne) bash.

~~Perl~~

- pipes: ex: `gpick | gtbl | gean | gtriff`
- rapid prototyping / slow performance.

Perl - swiss army chainsaw

- glue of the internet

- whip of titude & manipulexity

Python
Ruby

ruby
python
javascript

- Tools: ex: `gmake`, `flex`, `bison`.

HTML NOT a PL; it's a markup lang

1.4 Language Defn

Louden
1 Intro (6)

- ref manual

C - ca 1970 = 31 pages.

Algol 60 - 44 pages

R⁵ scheme - 50 pages

C++ - 800⁺ pages

Java - 300 pages + 1000's lib pages

- types : | tutorial
 | user guide
 | LRM.

- ISO/ANSI standards docs.

- Syntax - two levels
 lexical - regular.

BNF context-free tokens & rules.

"Backus-Naur format"

- Semantics

- informal operational

- denotational, axiomatic

- strict or loose

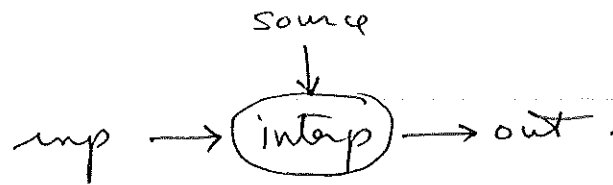
- harder to code.
 - harder to compile

1.5 Lang. Translation

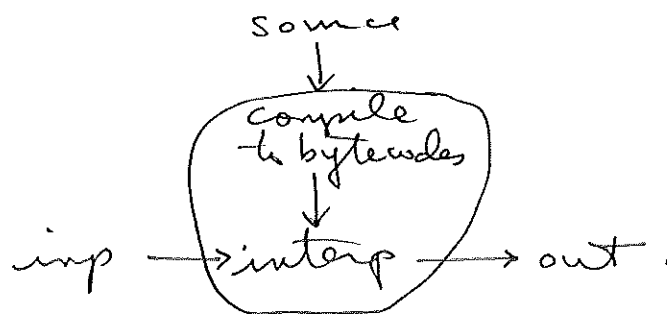
translator
interpreter
compiler

Loudon
1 intro (7)

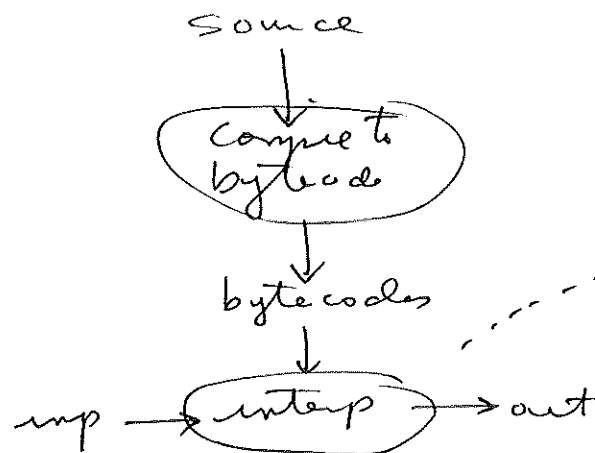
shell



Perl.

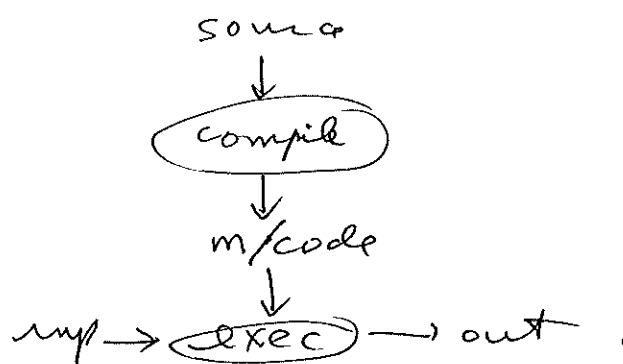


Java



JIT?

C



compile
assemble
link
exec (load)

Loudon
1 Intro (8)

static properties
dynamic properties.

~~Top loop:~~

fn lang: top level = loop { read; eval; print }
L R E P

runtime environment

- start routine
- lib.o
- dyn link lib / static link lib.

dyn/stat

- types
- untyped?
- stack / heap based.

1.6 Language Design

- readability / writeability
- security / compile time correct
- ptrs vs refs vs no null values
- complexity

choose PL? 1 hr & acceptable speed → Perl.

write once, run anywhere → Java (!?)

- ability to learn new lang quickly
- design little langs.