Louden
1 Intro (1)

What is a PL?

= comm. human → machine
human → human
m/c → m/c.

- hard wired plugboards?
- 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/ldstore/CPU.

Computation?

~~to~~ 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

syntax
is 2nd

- Human Rd. & writeability
  - abstraction
  - modularity
  - polymorphism
  - size > any other engr discipline
    10^3 - student asgt
    10^6 - O.S or compile
    10^9 - major system

Zipf's law
Whorf's law

Louden
1 Intro (2)

Software engineering
- multiple pgmrs
- multiple modules
- tools. (make, cvs, ...)

1.2 ABSTRACTIONS

data vs control } { basic, structured, 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
- reusible
- components
- containers (parameterized)
- libraries
- interoper.

CONTROL abstr
• Basic: assign, operators, ctl transfer (goto)
• Structured: if / while / ~~sw~~ switch / for

break
continue
throw
return.

- procedure / subroutine / function.
\ declaration | invocation
\ params: formals
actuals (args)

runtime environment: calling seq.
call stack

Louden
1 Intro (3)

procedure : args → results
or
none.

sometimes multiple
ex: let $\underbrace{(x, y)}_{\text{tuple}}$ = f (a, b)

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

1.3 Paradigms.

- Imperative (structured)
  ↑ cond, loop, [recur], call /procs.
  ↖ ~~if {while, recur}~~ ~~complete~~

  EVIL: goto
  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

Louden
1 Intro (4)

— OO —
classes - instances
- methods
- fields
▷ dynamic dispatch & inheritance
$a \to f(x,y) \Rightarrow (a \to vft \to f)(a, x, y)$
(C++) (C) ↑ dispatch table.

priv friend
prot
public pkg

duck typing.

- ctor / dtor (or gcol)
- 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

(λx. 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
  - explitly seq when needed.

    $y = f(g(x), h(z));$ — no side effects.

~~strict (ea~~
- strict (eager) vs nonstrict (lazy)
  - strict: call by value — Scheme, O'Caml
  - nonstrict: call by need. — Haskell.

||
&&
?:

- <u>Logic</u> – Declarative (Prolog)
  - expert systems.
  - unification of Horn clauses.

    clause: a :- b, c, d.

- Scripting, Tools, "Little Languages"
  - shell scripts (Bourne) bash.
  - ~~Perl~~.
  - pipes: ex: gpic|gtbl|geqn|gtroff
  - rapid prototyping / slow performance.

    Perl – swiss army chainsaw
    – glue of the internet
    Python
    Roby – whipuptitude & manipulexity

ruby
python
jscript

- Tools: ex: gmake, flex, bison.

HTML <u>NOT</u> a PL; its a markup lang

1.4 Language Defn

Louden
1 Intro (6)

- ref manual
  - C – ca 1970 = 31 pages.
  - Algol 60 – 44 pages
  - R^5 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

Loudon
Intro (7)

translator
interpreter
compiler

shell

$$\begin{tikzcd}
 & \text{source} \arrow[d] & \\
\text{inp} \arrow[r] & \text{interp} \arrow[r] & \text{out}.
\end{tikzcd}$$

Perl.

$$\begin{tikzcd}
 & \text{source} \arrow[d] & \\
 & \text{compile to bytecodes} \arrow[d] & \\
\text{inp} \arrow[r] & \text{interp} \arrow[r] & \text{out}.
\end{tikzcd}$$

Java

$$\begin{tikzcd}
 & \text{source} \arrow[d] & & \\
 & \text{compile to bytecode} \arrow[d] & & \\
 & \text{bytecodes} \arrow[d] & & \text{JIT?} \arrow[dll, dotted, dash] \\
\text{inp} \arrow[r] & \text{interp} \arrow[r] & \text{out} &
\end{tikzcd}$$

C

$$\begin{tikzcd}
 & \text{source} \arrow[d] & \\
 & \text{compile} \arrow[d] & \\
 & \text{m/code} \arrow[d] & \\
\text{inp} \arrow[r] & \text{exec} \arrow[r] & \text{out}.
\end{tikzcd}$$

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 $\longrightarrow$ Perl.
write once, run anywhere $\longrightarrow$ Java (?!).

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