CSC 533: Organization of Programming Languages

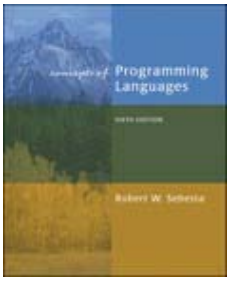
Spring 2005

See online syllabus at:

http://www.creighton.edu/~davereed/csc533

Course goals:

- understand issues in designing, implementing, and evaluating programming languages
- appreciate strengths and tradeoffs of different programming paradigms
- working knowledge of C++, Java & Scheme



Why are there different programming languages?

in theory, all programming languages are equivalent

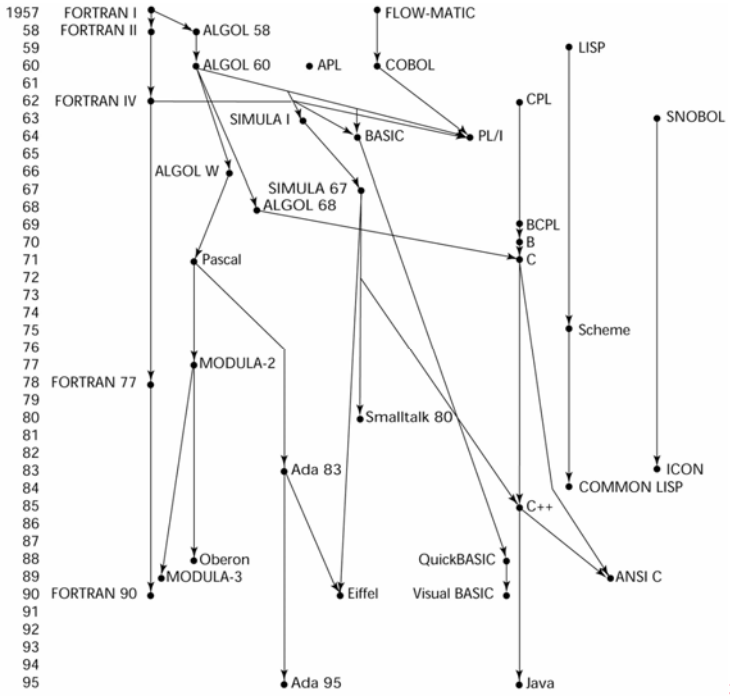
- compiled/interpreted into basic machine operations
- Church-Turing thesis applies

Why are there different natural languages?

in practice, different languages provide distinct voices

- different cultures (application domains)
- different primitive concepts (operations)
- different ways of thinking about the world (perspectives)

Family tree of high-level languages



Programming paradigms

similarly, different problem-solving approaches (paradigms) exist and are better suited to different types of tasks

imperative approaches: programs specify sequences of state changes

- *block-structured*: subroutines & nested scopes (Pascal, C)
- *object-based*: interacting objects (Ada, Modula)
- *object-oriented*: objects + inheritance (C++, Java, Smalltalk)
- *distributed*: concurrency (Occam, Linda)

declarative approaches: programs specify what is to be computed

- *functional*: functions in lambda calculus (LISP/Scheme, ML)
- *logic*: relations in predicate calculus (Prolog)
- *database*: info records and relations (SQL)

Example: HW1

HW1 involves calculating the molecular weight of a formula

- can be simple element
  `H` → `1.008` `O` → `15.994`
- can be sequence of elements
  `HHO` → `18.0154` `HHSOOOO` → `98.0736`
- can be sequence with multiple elements
  H_2O → `18.0154` H_2SO_4 → `98.0736`
- can be formula with multiple sub-formulas
  $(CH_3)_2CO$ → `58.0804`

this is not a particularly hard task in C++/Java, but does require design choices

- input file? I/O interface? data structure? …

Example: HW1 in LISP/Scheme

for comparison purposes, HW1 is ideal for Scheme

- Scheme is symbolic, has primitives for manipulating lists and structures, …

```
(define PERIODIC-TABLE
  '((H    1.0080) (He   4.0026) (Li   6.9410) (Be   9.0122) (B   10.8100) (C   12.0110)
    (N   14.0067) (O   15.9994) (F   18.9984) (Ne  20.1790) (Na  22.9898) (Mg  24.3050)
    (Al  26.9815) (Si  28.0860) (P   30.9738) (S   32.0600) (Cl  35.4530) (Ar  39.9480)
    (K   39.1020) (Ca  40.0800) (Sc  44.9559) (Ti  47.9000) (V   50.9414) (Cr  51.9960)
    (Mn  54.9380) (Fe  55.8470) (Co  58.9332) (Ni  58.7100) (Cu  63.5460) (Zn  65.3700)
    (Ga  69.7200) (Ge  72.5900) (As  74.9216) (Se  78.9600) (Br  79.9040) (Kr  83.8000)
    (Rb  85.4678) (Sr  87.6200) (Y   88.9059) (Zr  91.2200) (Nb  92.9064) (Mo  95.9400)
    (Tc  98.9062) (Ru 101.0700) (Rh 102.9055) (Pd 106.4000) (Ag 107.8680) (Cd 112.4000)
    (In 114.8200) (Sn 118.6900) (Sb 121.7500) (Te 127.6000) (I  126.9045) (Xe 131.3000)
    (Cs 132.9055) (Ba 137.3400) (La 138.9055) (Ce 140.1200) (Pr 140.9077) (Nd 144.2400)
    (Pm 145.0000) (Sm 150.4000) (Eu 151.9600) (Gd 157.2500) (Tb 158.9254) (Dy 162.5000)
    (Ho 164.9303) (Er 167.2600) (Tm 168.9342) (Yb 173.0400) (Lu 174.9700) (Hf 178.4900)
    (Ta 180.9479) (W  183.8500) (Re 186.2000) (Os 190.2000) (Ir 192.2200) (Pt 195.0900)
    (Au 196.9665) (Hg 200.5900) (Tl 204.3700) (Pb 207.2000) (Bi 208.9806) (Po 210.0000)
    (At 210.0000) (Rn 222.0000) (Fr 223.0000) (Ra 226.0254) (Ac 227.0000) (Th 232.0381)
    (Pa 231.0359) (U 238.02900) (Np 237.0482) (Pu 242.0000) (Am 243.0000) (Cm 247.0000)
    (Bk 249.0000) (Cf 251.0000) (Es 254.0000) (Fm 253.0000) (Md 256.0000) (No 256.0000)
    (Lr 257.0000)))

(define (molecular-weight formula)
    (cond ((symbol? formula) (cadr (assoc element PERIODIC-TABLE)))
          ((null? formula) 0.0)
          ((and (> (length formula) 1) (number? (cadr formula)))
           (+ (* (molecular-weight (car formula)) (cadr formula))
              (molecular-weight (cddr formula))))
          (else (+ (molecular-weight (car formula)) (molecular-weight (cdr formula))))))
```

Why study programming languages?

increased capacity to express ideas
- broader perspective on programming and problem solving, new paradigms

improved background for choosing appropriate languages
- know tradeoffs between different languages
- simplify programming task by choosing best-suited language

increased ability to learn new languages
- as with natural languages, 2nd languages is hardest to learn
- languages come and go, must be able to adapt/adjust

better understanding of the significance of implementation
- can use language more intelligently if understand implementation
- also aids in identifying bugs

increased ability to design/implement new languages

How do we judge a programming language?

readability

in software engineering, maintenance cost/time far exceeds development cost/time
→ want code to be easy to understand, modify, and maintain

- *simplicity:* language should be as small as possible, avoid redundant features
  - C++ is pretty complex, e.g., `x++;` `++x;` `x +=1;` `x = x + 1;`
  - Java slightly better (some features removed); Scheme is very simple
- *orthogonality:* small set of primitive constructs, can be combined independently and uniformly (i.e., very few special cases)
  - C++ is OK but many exceptions, e.g., functions can return structs, not arrays
  - Java slightly better; Scheme is highly orthogonal
- *natural control and data structures:* provide useful, high-level abstractions
  - C++ is good but does include some tricky ones, e.g., `:?, goto`
  - Java comparable (but no `goto`); Scheme is limited (e.g., recursion for looping)
- *simple and unambiguous syntax:* intended form of code is clear to reader
  - C++ not so good, e.g., overall complexity, dangling else, overused `static`
  - Java slightly better; Scheme syntax is simple and clear

How do we judge a programming language (cont.)?

writability

want to make programming as simple as possible for the programmer

- *simplicity*
- *orthogonality:*
- *natural control and data structures*
- *simple and unambiguous syntax*

- *support for abstraction:* need to be able to define and utilize abstractions
  - C++ is good, e.g., support for functions, libraries, classes
  - Java is comparable; Scheme is OK, but more tedious
- *expressivity:* language provides convenient ways of specifying computations
  - C++ is good, e.g., if & switch, while & do-while & for, bitwise operators, …
  - Java is slightly less (removes low-level); Scheme is not very expressive (few control structures)

note: readability & writability are often at odds

e.g., more control structures can simplify programmer's task, but make code harder to read and maintain (more to know, more redundancy)

Common LISP vs. Scheme

How do we judge a programming language (cont.)?

reliability

want to ensure that a program performs to its specifications under all conditions

→want to build in strong error checking and recovery capabilities

→also want to help the programmer to avoid errors

- *readability*
- *writability*

- *type checking:* identify errors either at compile-time or during execution
  - C++ is pretty good, e.g., most types checked at compile time, some loopholes
  - Java is slightly better; Scheme is dynamic, must do checking during execution
- *exception handling:* ability to intercept and recover from errors in code
  - C++ is OK (try/catch, but not always used)
  - Java is slightly better (libraries require exception handling), Scheme is more awkward

- *memory management:* control memory accessing, allocation/deallocation, aliasing
  - C++ is *dangerous*, e.g., can access specific addresses, must deallocate memory
  - Java is better (memory mgmt is automatic); Scheme handles all transparently