Our common framework

Goal: eliminate superficial differences

• Makes comparisons easy
• Differences that remain must be important!

No new language ideas.

Imperative programming with an IMPerative CORE:

• Has features found in most languages (loops and assignment)
• Trivial syntax (from LISP)
Idea of LISP syntax

Parenthesized prefix syntax:
• Names and numerals are basic atoms
• Other constructs bracketed with (…) or [...]
  (Possible keyword after opening bracket)

Examples:

(+ 2 2)
(if (isbound? x rho) (lookup rho x) (error 99))

(For now, we use just the round brackets)
Impcore structure

Two syntactic categories: expressions, definitions

No statements!—expression-oriented (compositional)

(if e1 e2 e3)
(while e1 e2)
(set x e)
(begin e1 ... en)
(f e1 ... en)

Evaluating $e$ has value, may have side effects

Functions $f$ named (e.g., $+$ $-$ $*$ $/$ $=$ $<$ $>$ print)

The only type of data is “machine integer”
(deliberate oversimplification)
Syntactic structure of Impcore

An Impcore program is a sequence of definitions

(define mod (m n) (¬ m (* n (/ m n))))

Compare

int mod (int m, int n) {
    return m - n * (m / n);
}
Impcore variable definition

Example

(val n 99)

Compare

int n = 99;
Concrete syntax for Impcore

Definitions and expressions:

\[
\text{def ::= (define f (x1 \ldots xn) exp) ;; "true" defs}
\]
\[
| \quad \text{(val x exp)}
\]
\[
| \quad \text{exp}
\]
\[
| \quad \text{(use filename)} \quad ;; "extended" defns
\]
\[
| \quad \text{(check-expect exp1 exp2)}
\]
\[
| \quad \text{(check-assert exp)}
\]
\[
| \quad \text{(check-error exp)}
\]

\[
\text{exp ::= integer-literal}
\]
\[
| \quad \text{variable-name}
\]
\[
| \quad \text{(set x exp)}
\]
\[
| \quad \text{(if exp1 exp2 exp3)}
\]
\[
| \quad \text{(while exp1 exp2)}
\]
\[
| \quad \text{(begin exp1 \ldots expn)}
\]
\[
| \quad \text{(function-name exp1 \ldots expn)}
\]
Recursive-function problem

Exercise: all-fours?

Write a function that takes a natural number \( n \) and returns true (1) if and only if all the digits in \( n \)'s numeral are 4's.

Begin with unit tests (which also document):

(\text{check-assert} \quad (\text{all-fours?} \quad 4))

(\text{check-assert} \quad (\text{not} \quad (\text{all-fours?} \quad 5)))

(\text{check-assert} \quad (\text{all-fours?} \quad 44))

(\text{check-assert} \quad (\text{not} \quad (\text{all-fours?} \quad 14)))

Choose inductive structure for natural numbers:

• Which case analysis do we want?
Solution to “all-fours?”

(check-assert (all-fours? 4))
(check-assert (not (all-fours? 5)))
(check-assert (all-fours? 44))
(check-assert (not (all-fours? 14)))

(define all-fours? (n)
  (if (< n 10)
      (= n 4)
      (and (= 4 (mod n 10))
           (all-fours? (/ n 10)))))

;; D2 recursion: n is d, where 0 < d < 10, or
;; n is 10 * m + d, where m > 0
(Now we can talk a bit about the course.)