The #1 programmer excuse for legitimately slacking off:

"My code's compiling."

Hey! Get back to work!

Compiling!

Oh, carry on.
Homework 1

- Springer and Friedman
  - 1.2, 1.3, 1.4, 1.5, 1.6
  - 1.10, 1.14
  - 2.1, 2.3, 2.4, 2.6, 2.7, 2.10
  - 2.12, 2.13, 2.14, 2.15, 2.16
  - 2.18

- Any answers which are not Scheme definitions should be commented out using ;;
Scheme Datatypes

expressions

pairs

lists

atoms

symbols

booleans

numbers

functions
cons Makes Pairs

(cons 1 2) →

pair

1 2

car →
cdr ←
Lists

• A list is either
  – an empty list '()
  – a pair whose cdr is a list.
Adding Something to the Front of a List

(cons 1 (list 2 3))  (cons 1 (cons 2 (cons 3 '(')))  (list 1 2 3)
Symbols and Quotation

(define fred (cons 1 2))
(define foo 'fred)
(define bar fred)
Symbols and Quotation (contd.)

(define foo (cons 1 2))
(define bar (cons 1 2))
(define bert 'foo)
(define ernie bar)
eq? versus equal?

(eq? foo bar) → #f

(equal? foo bar) → #t

(eq? bert 'foo) → #t

(eq? bert foo) → #f

(equal? bert 'foo) → #t

(equal? bert ernie) → #f

(eq? bar ernie) → #t

(eq? x y) → (equal? x y)
**cond** special-form

\[
(\text{cond} \ (\text{pred}_1 \ \text{val}_1) \ \ldots \ (\text{pred}_{N-1} \ \text{val}_{N-1}) \ \text{(else} \ \text{val}_N))
\]

- The **cond** special-form evaluates **pred**\(_1\).
- If **pred**\(_1\) is not #f it evaluates and returns **val**\(_1\).
- Otherwise **cond** evaluates **pred**\(_2\).
- If **pred**\(_2\) is not #f it evaluates and returns **val**\(_2\).
- If none of **pred**\(_1\) … **pred**\(_{N-1}\) evaluates to not #f **cond** evaluates and returns **val**\(_N\).
or special-form

(or pred₁ pred₂ ... predₙ₋₁ predₙ)

- The or special-form evaluates pred₁.
- If pred₁ is not #f or returns it.
- Otherwise or evaluates pred₂.
- If pred₂ is not #f or returns it.
- If none of pred₁ ... predₙ₋₁ evaluates to not #f or returns predₙ.
and special-form

(and pred₁ pred₂ … predₙ₋₁ predₙ)

- The and special-form evaluates pred₁.
- If pred₁ is #f and returns #f.
- Otherwise and evaluates pred₂.
- If pred₂ is #f and returns #f.
- If none of pred₁ … predₙ₋₁ evaluates to #f and returns predₙ.
Imperative Programs

- A program in an imperative language is a sequence of statements.
- Each statement transforms the state of the machine, i.e., the contents of registers and memory.
- The goal is to find a sequence of statements that will transform the input state into the desired output state.
- The sequence of statements is a description of a process.
Functional Programs

- A program in a functional language is an expression.
- Expressions are evaluated by recursively evaluating subexpressions.
- The expression is the *definition of the answer to a problem.*
A Program that Recognizes Lists

• Recall that a list is either
  - an empty list '()
  - a pair whose \textit{cdr} is a list.

• In Scheme, the program that recognizes lists is \textit{literally} the definition of a list

\[
\begin{align*}
\text{(define list? (lambda (sexpr) (or (null? sexpr) (and (pair? sexpr) (list? (cdr sexpr)))))})
\end{align*}
\]