CS 257: Non-Imperative Programming: Scheme!
Homework 2 (Spring ’05)

A *natural* is an abstract data type (ADT) which supports the following primitive operations (also shown are implementations of the primitive operations in Scheme):

- *n0* - the representation of the natural, 0.

```
(define n0 '())
```

- *nzero?* - Given natural, x, returns #t if $x = 0$ and #f otherwise.

```
(define nzero? null?)
```

- *nadd1* - Given natural, x, returns natural, $x+1$.

```
(define nadd1
  (lambda (x)
    (cons x '())))
```

- *nsub1* - Given natural, x, returns natural, $x-1$.

```
(define nsub1 car)
```

Using only the above primitive operations, it is possible to define the following arithmetic operations for the natural ADT:

- *n+* - Given naturals, x and y, returns natural, $x+y$.

```
;; x+0 = x
;; x+y = [x + (y-1)] + 1
(define n+
  (lambda (x y)
    (if (nzero? y)
      x
      (nadd1 (n+ x (nsub1 y))))))
```

- *n-* - Given naturals, x and y, returns natural, $x-y$.

```
;; x-0 = x
;; x-y = [x - (y-1)] - 1
(define n-
  (lambda (x y)
    (if (nzero? y)
      x
      (nsub1 (n- x (nsub1 y))))))
```

- *n** - Given naturals, x and y, returns natural, $x \times y$.

```
;; x*0 = 0
;; x*y = [x*(y-1)] + x
(define n*
  (lambda (x y)
    (if (nzero? y)
      n0
      (n+ x (n* x (nsub1 y))))))
```

- *n*$^\wedge$ - Given naturals, x and y, returns natural, x^y.

```
;; xˆ0 = 1
;; xˆy = x * xˆ(y-1)
(define nˆ
  (lambda (x y)
    (if (nzero? y)
      n1
      (n* x (nˆ x (nsub1 y))))))
```

An *integer* is an abstract data type which supports the following primitive operations:

- *i0* - the representation of the integer, 0.
- *izero?* - Given integer, x, returns #t if $x = 0$ and #f otherwise.
- *iadd1* - Given integer, x, returns integer, $x + 1$.
- *isub1* - Given integer, x, returns integer, $x - 1$.
- *iminus* - Given integer, x, returns $-x$.
- *ipositive?* - Given integer, x, returns #t if $x > 0$ and #f otherwise.
- *inegative?* - Given integer, x, returns #t if $x < 0$ and #f otherwise.

Using only the above primitive operations, it is possible to define the following arithmetic operations for the integer ADT:

- *i+* - Given integers, x and y, returns integer, $x+y$.
- *i-* - Given integers, x and y, returns integer, $x-y$.
- *i** - Given integers, x and y, returns integer, $x \times y$.
- *i/* - Given integers, x and y, returns integer, x/y.

```
;; x/x = 1
;; x/1 = x
;; x/y = 0 if y > x
;; x/y = (x-y)/y + 1
(define i/
  (lambda (x y)
    (cond ((i= x y) i1)
          ((i= y i1) x)
          ((i= y i0) (error "Divide by zero."))
          ((inegative? x)
           (iminus (i/ (iminus x) y)))
          ((inegative? y)
           (iminus (i/ x (iminus y))))
          (else
           (if (i> y x)
               i0
               (iadd1 (i/ (i- x y) y)))))))
```

- *i%* - Given integers, x and y, returns integer $x \bmod y$.

```
;; x mod y = x - [(x/y) * y]
(define i%
  (lambda (x y)
    (i- x (i* (i/ x y) y))))
```

- *i=* - Given integers, x and y, returns #t if $x=y$ and #f otherwise.
- *i>* - Given integers, x and y, returns #t if $x>y$ and #f otherwise.

Problems

1. Give definitions for *izero?*, *iadd1*, *isub1*, *iminus, ipositive?* and *inegative?* based on the following implementation for the integer ADT:

$$
\begin{array}{ccc}
(\textit{isub1}(\textit{isub1}(\textit{isub1}\ \textit{i0}))) & \rightarrow & (((()()())()) \\
(\textit{isub1}(\textit{isub1}\ \textit{i0})) & \rightarrow & (((()())()) \\
(\textit{isub1}\ \textit{i0}) & \rightarrow & ((())()) \\
\textit{i0} & \rightarrow & (()()) \\
(\textit{iadd1}\ \textit{i0}) & \rightarrow & (()(())) \\
(\textit{iadd1}(\textit{iadd1}\ \textit{i0})) & \rightarrow & (()(()())) \\
(\textit{iadd1}(\textit{iadd1}(\textit{iadd1}\ \textit{i0}))) & \rightarrow & (()(()()()))
\end{array}
$$

2. Give definitions for the functions *number–>integer* which converts a Scheme number to the corresponding instance of the integer ADT and *integer–>number* which converts an instance of the integer ADT to the corresponding Scheme number. An example follows:

```
> (integer->number (iadd1 (iadd1 (iadd1 i0))))
3
> (number->integer 3)
(() (() () ()))
>
```

3. Using only the primitive operations for the integer ADT, *i.e.*, the primitives you have defined above, give definitions for the integer arithmetic operations, *i*+, *i*-, *i**, *i*= and *i*>.

4. Do the following exercises from Springer and Friedman: 4.1, 4.4, 4.6, 4.10, 4.11