Specifying Data via Interfaces

Exercise 2.1

```
(define base 16)
(define zero (lambda () ’()))
(define is-zero? (lambda (n) (null? n)))

(define successor
  (lambda (n)
    (if (is-zero? n)
        (cons 1 (zero))
        (let ((b (+ 1 (lsb n))))
          (if (overflow? b base)
              (cons 0 (successor (rest-bigits n)))
              (cons b (rest-bigits n)))))))

(define predecessor
  (lambda (n)
    (if (is-zero? n)
        (zero)
        (let ((b (- (lsb n) 1)))
          (cond ((one->zero? b (rest-bigits n))
                 (zero))
                ((underflow? b)
                 (cons (- base 1) (predecessor (rest-bigits n))))
                (else (cons b (rest-bigits n))))))))

(define overflow? =)
(define underflow? (lambda (b) (< b 0)))
(define one->zero?
  (lambda (b n)
    (and (= b 0)
         (is-zero? n))))
(define lsb car)
(define rest-bigits cdr)

(define factorial
  (lambda (n)
    (if (is-zero? n)
        (successor (zero))
        (times n (factorial (predecessor n))))))

(define times
  (lambda (x y)
    (cond ((is-zero? y) (zero))
          ((is-one? y) x)
          (else (plus x (times x (predecessor y)))))))

(define plus
  (lambda (x y)
    (if (is-zero? x)
        y
        (successor (plus (predecessor x) y)))))

(define is-one?
  (lambda (n)
    (is-zero? (predecessor n))))
```

Exercise 2.2

The unary representation fails to print a numeral. The Scheme representation uses an already made representation, and implicitly allows for Scheme operations to be applied to these numbers, and also prints useful numerals. The bignum representation prints numerals which typically need to be translated to a more readable base. The representations clearly satisfy the specification, however the specifications do not make any demands about how the representation is printed, and this is where it fails for the client.