Specifying Data via Interfaces

Exercise 2.1

```scheme
(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))))

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

(define successor
  (lambda (n)
    (if (is-zero? n)
        (one)
        (bignum-s (increment-lsb n)))))

(define bignum-s
  (lambda (n)
    (cond ((if-zero? n) (zero))
          ((overflow? (lsb n) base)
           (cons (remainder (lsb n) base)
                 (bignum-s (successor (rest-bigits n)))))
          (else n))))

(define overflow? >=)

(define increment-lsb
  (lambda (n)
    (cons (+ (lsb n) 1)
          (rest-bigits n))))

(define predecessor
  (lambda (n)
    (if (is-zero? n)
        (zero)
        (bignum-p (decrement-lsb n)))))

(define bignum-p
  (lambda (n)
    (cond ((if-zero? n) (zero))
          ((underflow? (lsb n) base)
           (a

(define lsb car)
(define rest-bigits cdr)
```