```racket
#lang racket

(struct Zero () #:transparent)
(struct Succ (n) #:transparent)

(define (make-number prev)
  (match prev
    ((Succ prev) (Succ (Succ prev)))
    ((Zero) (Succ (Zero)))
    (_ (printf "Error: not a valid number!"))))

(define one (make-number (Zero)))
one

(define two (make-number one))

two

(define (addition n1 n2)
  (match n1
    ((Zero) n2)
    ((Succ i)(addition i (Succ n2)))))

(addition two two)

(struct Pred (n) #:transparent)

(Pred (Zero))

(define (make-number-2 prev)
  (match prev
    ((Succ prev) (Succ (Succ prev)))
    ((Zero) (Succ (Zero)))
    ((Pred n) n)
    (_ (printf "Error: not a valid number!\n"))))

(make-number-2 (Zero))

```