CONTINUATIONS

```
(def !/k
  (λ (n k)
    (if (= n 0)
        (k 1)
        (!/k (sub1 n)
             (λ (v)
               (k (* n v)))))))

(!/k 2 top-k)
```

(!/k 1 (λ (v) (top-k (* 2 v)))) k_1

=

(!/k 0 (λ (v) (k1 (* 1 v)))) k_2

=

(k2 1)

= (k1 (* 1 1))

= (k1 1)

= (top-k (* 2 1))

= (top-k 2)

= 2

CPS programs ~~prog~~ are always tail recursive.

```
(define map
  (λ (h ls)
    (if (null? ls) '()
     (cons (h (car ls))
       (map h (cdr ls))))))
```

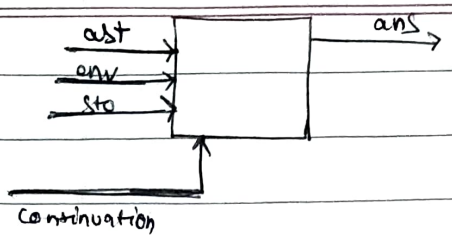
```
(define map/k
  (λ (h/k ls k)
    (if (null? ls)
      (k '())
    (h/k (car ls)
      (λ (v)
        (map/k h/k (cdr ls)
          (λ (w)
            (k (cons v w))))))))
```

take a program to cps must be tail recursive

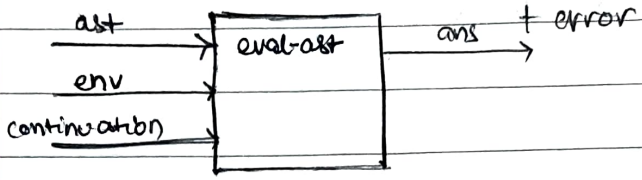
~~(map/k (λ (v) k) k~~

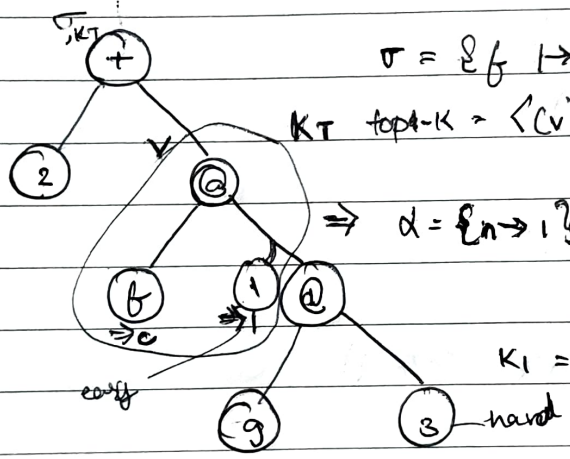
```
(map/k (λ (v k) (k (add1 v)))
       '(3 1 7)
       (λ (w)
         (cons 2 w)))
```

= '(2 4 2 8)



ans = (exp. value, store)





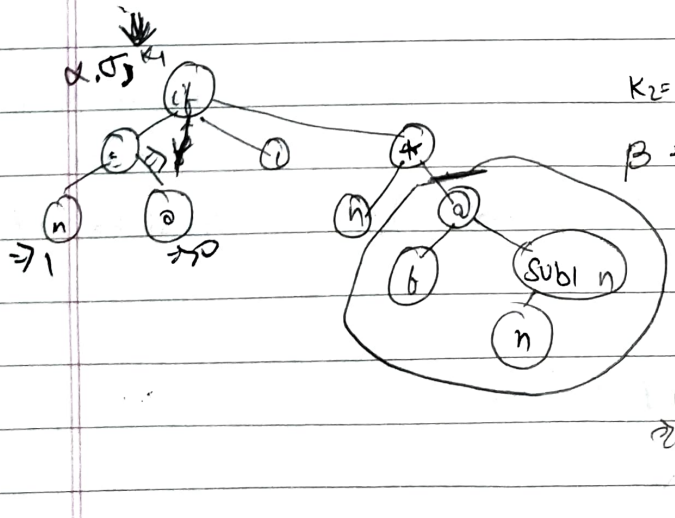
$\sigma = \{f \mapsto c\}$ ↗ : fn. $C = \langle (n), b, e \rangle$

K_T top-k = $\langle (v), v, \text{empty-env} \rangle$

$\Rightarrow \alpha = \{n \to 1\}$

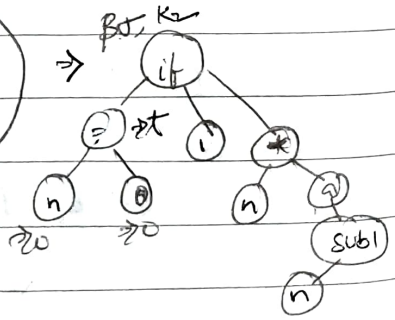
$K_1 = \langle (v), (+2\ v), \sigma, K_T \rangle$

continuation → list of closures



$K_2 = \langle (v), (*\ n\ v), \alpha\sigma, K_1 \rangle$

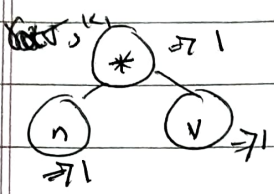
$\beta = \{n \mapsto 0\}$



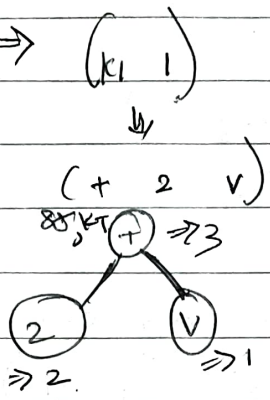
$\Rightarrow (K_2\ 1)$

(K2 1) $\gamma = \{v \mapsto 1\}$

$\delta = \{v \mapsto 1\}$



⇒



$\epsilon = \{v \mapsto 3\}$

env

evaluator

```
(define 'lookup-env/k
  (λ (e id k)
    (cases env e
      [empty-env ()
        (format "error: unbound id ~a" id)]
      [extended-env (ids vals outer-env)
        (letrec ([loop (λ (ids vals)
          (if (null? ids)
            (lookup-env/k id outer-env k)
            (if (eq? id (car ids))
              (k (car vals))
              (loop (cdr ids) (cdr vals)))))])
          (loop ids vals))])))
```