classmate
Date 7.11.2019
Page 1

UNDERSTANDING OBJECTS

CONTINUATION PASSING STYLE

```
> (define g
    (λ (x)
      (+ x 2)))

> (g 3)
=> 5

> (+ 7 (g 3))
=> 12
```

with — continuation

```
> (define g/k
    (λ (x k)
      (k (+ x 2))))
```

reciever

```
> (g/k 3 (λ (v) (+ 7 v)))
=> 12

> (+ 3 (g/k 3 (λ (v) (+ 7 v))))
=> 15
```

take queue add the paramotor

```
.... (let ([v (lookup-env e id)])
       ⋮
     )
```

```
(define (lookup-env e x)
  (... (if (...)
           (error 'lookup-env "")))

(define (lookup-env/k e x succ fail)
  (... (succ v)
       (fail x)))

(lookup-env/k e x (λ (v) (...))
              (λ (x) (error 'unbound identifier" x)))

(λ (e) (lookup-env/k e2 x succ failure))
```

~~examples~~

EXAMPLES

```
(define (! n)
  (if (= n 0)
      1
      (* n (! (sub1 n)))))

(! 2)
= (* 2 (! 1))
= (* 2 (* 1 (! 0)))
= (* 2 (* 1 1)) = 2
```

CPS

continuation

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

dabt.

continuation receiver

call

$k_0 = (\lambda\ (\square)\ (k\ (*\ n\ \square)))$

CPS – tail recursive

```
> (!/k 2 (λ (m) m))
(!/k 1 k1)
(!/k 0 k2)
(k2 1)
(k1 (* 1 1)) = (k1 1)
(k0 (* 2 1)) = (k0 2) = 2
```

$k_1 = (\lambda (v)\ (k_0\ (*\ 2\ v)))$

$k_2 = (\lambda (0)\ (k_1\ (*\ 1\ 0)))$

```
> (!/k 2 (λ(m) m)) =
          \_____/
             k0

= (!/k 1 k1)
= (!/k 0 k2)
= (k2 0)
= (k1 (* 1 1))
= (k1 1)
= (k0 (* 2 1)
= (k0 2)
= 2
```

with accumulator

```
> (define (! /ac n)
      (helper n 1))
> (define (helper n a)
    (if (= n 0)
        a
        (helper (sub1 n) (* a n))))
```

CPS version.

```
(define (!/acc-k n k)
    (helperk n 1 k))
(define (helperk n a k)
    (if (= n 0)
        (k a)
        (helperk (sub1 n) (* a n) k)))
                          argument
```

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

the goal of CPS is to remove stacks.
entire control of -