# Rekursiivne vs. iteratiivne käitumine

- Faktoriaali rekursiivne definitsioon

```
(define fact
  (lambda (n)
    (if (zero? n) 1 (* n (fact (- n 1))))))
```

- `fact` iga järgmine rekursiivne väljakutse toimub üha suuremas kontekstis

```
(fact 4)  ->  (* 4 (fact 3))
          ->  (* 4 (* 3 (fact 2)))
          ->  (* 4 (* 3 (* 2 (fact 1))))
          ->  (* 4 (* 3 (* 2 (* 1 (fact 0)))))
          ->  (* 4 (* 3 (* 2 (* 1 1))))
          ->  24
```

# Rekursiivne vs. iteratiivne käitumine

- Faktoriaali saba-rekursiivne definitsioon

```
(define fact-iter (lambda (n) (fact-iter-acc n 1)))

(define fact-iter-acc (lambda (n a)
    (if (zero? n) a (fact-iter-acc (- n 1) (* n a)))))
```

- `fact-iter` kõik rekursiivsed väljakutsed toimuvad samas kontekstis

```
(fact-iter 4)  ->  (fact-iter-acc 4 1)
               ->  (fact-iter-acc 3 4)
               ->  (fact-iter-acc 2 12)
               ->  (fact-iter-acc 1 24)
               ->  (fact-iter-acc 0 24)
               ->  24
```

# Rekursiivne vs. iteratiivne käitumine

- Protseduur on iteratiivse käitumisega kui ta kasutab O(1) mälu
- Protseduur on iteratiivse kontroll-käitumisega kui ta kasutab O(1) mälu konteksti säilitamiseks

```
(define revappend
  (lambda (ls1 ls2)
    (if (null? ls1)
        ls2
        (revappend (cdr ls1) (cons (car ls1) ls2)))))
```

- Protseduur on rekursiivse (kontroll-)käitumisega, kui ta ei ole iteratiivse (kontroll-)käitumisega

# Jätkud

- Vaatleme `fact` arvutamise sammu

```
(* 4 (* 3 (* 2 (fact 1))))
```

- `(fact 1)` arvutatakse kontekstis

```
(* 4 (* 3 (* 2 □)))
```

- Me saame konteksti (“ühe auguga” avaldist) esitada ühe argumendilise lambda-avaldisena, mida kutsutakse jätkuks (ingl. *continuation*)

```
(lambda (v) (* 4 (* 3 (* 2 v))))
```

- Tähistagu `k` seda lambda-avaldist, siis kehtib

```
(k (fact 1))  ->  (* 4 (* 3 (* 2 (fact 1))))
```

# Jätkud

- Analoogselt on kõik `fact` arvutamise sammud esitatavad kujul `(k (fact n))`
- CPS = *Continuation Passing Style*
- CPS-kujul faktoriaal

```
(define fact-cps
  (lambda (n k)
    (if (zero? n)
        (k 1)
        (fact-cps (- n 1)
          (lambda (v) (k (* n v)))))))
```

# Jätkud

- Iga `k` ja `n` korral kehtib

```
(k (fact n))  =>  (fact-cps n k)
```

- Tõestus induktsiooniga
  - kui `n = 0`

```
(k (fact 0))  =>  (k 1)  => (fact-cps 0 k)
```

  - kui `n > 0`

```
   (k (fact n))
=> (k (* n (fact (- n 1))))
=> ((lambda (v) (k (* n v))) (fact (- n 1)))
=> (fact-cps (- n 1) (lambda (v) (k (* n v))))
=> (fact-cps n k)
```

# Jätkud

`fact-cps` on iteratiivse kontroll-käitumisega

```
   (fact-cps 4 k)
-> (fact-cps 3 (lambda (v) (k (* 4 v))))
-> (fact-cps 2 (lambda (v) (k (* 4 (* 3 v)))))
-> (fact-cps 1 (lambda (v) (k (* 4 (* 3 (* 2 v))))))
-> (fact-cps 0 (lambda (v) (k (* 4 (* 3 (* 2 (* 1 v)))))))
-> ((lambda (v) (k (* 4 (* 3 (* 2 (* 1 v)))))) 1)
-> (k (* 4 (* 3 (* 2 (* 1 1))))   -> (k 24)
```

"Kontekstist sõltumatu" faktoriaali saame, kui anname algjätkuks identsusprotseduuri

```
(define fact
  (lambda (n) (fact-cps n (lambda (v) v)))
```

# Jätkud

- Rekursiivne `length`

```
(define length
  (lambda (ls)
    (if (null? ls)
        0
        (+ (length (cdr ls)) 1))))
```

- CPS-kujul `length`

```
(define length-cps
  (lambda (ls k)
    (if (null? ls)
        (k 0)
        (length-cps (cdr ls)
          (lambda (v) (k (+ v 1)))))))
```

# Saba vormid

- Millal on protseduur iteratiivse kontroll-käitumisega?
- Üldjuhul lahendamatu ülesanne

```
(lambda (n)
   (if (strange-predicate? n)
       (fact n)
       (fact-iter n)))
```

- Avaldis on nn. saba vormis (ingl. *tail form*), kui protseduuri väljakutsed on alati derivatsioonis kõige välimised
  - see tingimus on staatiliselt kontrollitav

# Saba vormid

- Alamavaldis on saba-positsioonis (*tail position*) teda sisaldavas avaldises, kui tema väärtustamise korral on ta väärtus “koheselt” kogu avaldise väärtuseks

  ```
  (if <test-expr> <then-expr> <else-expr>)
  ```

- Saba-positsioonis oleva alamavaldise väärtustamisel pole vaja kontrollinformatsiooni salvestada
- Alamavaldis $E'$ on saba-positsioonis avaldises $E$, kui iga $E$ alamavaldis mis sisaldab $E'$-i on saba-positsioonis teda vahetult ümbritsevas alamavaldises

  ```
  (if (zero? x) (f (g x)) (if (positive? x) 1 (f x)))


  (f (g x)),  1,  (f x),  (if (positive? x) 1 (f x))
  ```

# Saba vormid

- Alamavaldis mida võib väärtustada esimesena on pea-positsioonis (*head position*)

```
( prim-op H ... H)
(H H ... H)
(if H T T)
(let ((b H) ... (b H)) T)
(lambda (v ... v) E)
```

- Alamavaldis $E'$ on avaldises $E$ saadaval (*available*), kui ta ei sisaldu üheski $E$ lambda-alamavaldises

```
(if (f a) (lambda () (g 2)) (h x y))
```

- Avaldis on lihtne, kui ta ei sisalda saadavaid (mitte-primitiivseid) aplikatsioone

# Saba vormid

```
(define-record prim-app (primitive rands))


(define simple?
  (lambda (exp)
    (and (not (app? exp))
         (andmap simple? (head-exps exp))
         (andmap simple? (tail-exps exp)))))
```

# Saba vormid

```
(define head-exps
  (lambda (exp)
    (variant-case exp
      (lit (datum) '())
      (varref (var) '())
      (if (test-exp then-exp else-exp)
        (list test-exp))
      (proc (formals body) '())
      (let (decls body) (map decl->exp decls))
      (letrec (decls body) (map decl->exp decls))
      (prim-app (primitive rands) rands)
      (app (rator rands) (cons rator rands)))))
```

# Saba vormid

```
(define tail-exps
  (lambda (exp)
    (variant-case exp
      (lit (datum) '())
      (varref (var) '())
      (if (test-exp then-exp else-exp)
        (list then-exp else-exp))
      (proc (formals body) '())
      (let (decls body) (list body))
      (letrec (decls body) (list body))
      (prim-app (primitive rands) '())
      (app (rator rands) '()))))
```

# Saba vormid

- Avaldis on saba vormis, kui iga tema mitte-saba-positsioonis olev alamavaldis on lihtne

| | | |
|---|---|---|
| `(car x)` | lihtne | saba vormis |
| `(if p x (car (cdr x)))` | lihtne | saba vormis |
| `(f (car x))` | ei ole lihtne | saba vormis |
| `(car (f x))` | ei ole lihtne | ei ole saba vormis |
| `(if p x (f (cdr x)))` | ei ole lihtne | saba vormis |
| `(if (f x) x (f (cdr x)))` | ei ole lihtne | ei ole saba vormis |
| `(lambda (x) (f x))` | lihtne | saba vormis |
| `(lambda (x) (car (f x)))` | lihtne | ei ole saba vormis |

# Saba vormid

- Avaldis on iteratiivse kontroll-käitumisega, kui avaldis on saba vormis ja kõik muutujatega seotud protseduurid, mis on saadavad, on saba vormis
- Avaldis on esimest-järku kujul, kui ta ei sisalda lambda-alamavaldisi ja kõik operaatoravaldised on muutujad mis on seotud välimisel tasemel

# Järgmiseks korraks

- Lugeda läbi EOPL ptk. 8.1 – 8.3