Процедури от по-висок ред
lambda и let изрази

Въведение

- Процедура от по-висок ред е такава процедура, която:
  - Има за параметър една или повече други процедури, И/ИЛИ
  - Връща процедура като резултат
- В Scheme – процедури = данни
  - Казваме, че процедурите са „first class objects“ на езика
- В Haskell – също
- В много други (императивни) езици вече се поддържат такива конструкции

Подаване на процедура като аргумент

- Нищо по-различно от това, което сте виждали досега
  - Защото процедури = данни

```
(define (apply op a b)
    (op a b))
```

???

```
(define (sum-integers a b)
  (if (> a b)
      0
      (+ a (sum-integers (+ a 1) b))))
```

```
(define (sum-cubes a b)
  (if (> a b)
      0
      (+ (cube a) (sum-cubes (+ a 1) b))))
```

```
(define (sum-smth a b)
  (if (> a b)
      0
      (+ (/ 1 (* a 3)) (sum-smth (+ a 2) b)))))
```

Абстрактна процедура за сума

```
(define (sum term a next b)
    (if (> a b)
        0
        (+ (term a) (sum term (next a) b))))
```

Абстракцията получаваме, като параметризираме общи елементи на процедури

Got more abstractions?

- Да, и те си имат име – accumulate
- Остана да добавим операцията, която извършваме, и стойността, която се връща при преминаване на интервала

```
(define (accumulate term op null-value a next b)
      (if (> a b)
          null-value
          (op (term a) (accumulate term op null-value (next a) next b))))
```

Неудобство

Нека имаме следните дефиниции

```
(define (cube x) (* x x x))

(define (inc x) (+ x 1))

(define (cube-sum a b)
      (accumulate cube + 0 a inc b))
```

За да се възползваме от accumulate се налага да дефинираме процедури, които (най-вероятно) ще използваме само веднъж – cube, inc.

lambda изрази

- Специална форма в Scheme
- Чрез нея създаваме анонимни процедури
- Дефинираме процедурите там, където се използват и никъде другаде
- Не се свързват с име
- Няма възможност такава процедура да бъде извикана от друго място в кода

Решение на проблем от сл.7

Синтаксис на lambda изразите:

- (lambda (<параметри>) (<тяло>))

- (lambda (x) (* x x))

- (lambda (x y z) (+ y (* z x)))

Ако искаме да напишпем cube-sum, използвайки lambda изрази вместо предварително дефинирани такива, бихме получили следния код:

```
(define (cube-sum a b)
      (accumulate (lambda (x) (* x x x)) + 0 a (lambda (x) (+ x 1)) b))
```

Последен вариант на cube-sum, обещавам:

```
(define cube-sum (lambda (a b)
    (accumulate
        (lambda (x) (* x x x))
        +
        0
        a
        (lambda (x) (+ x 1))
        b)))
```