Списъци

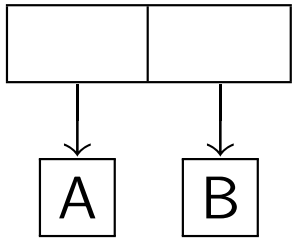
Трифон Трифонов

Функционално програмиране, 2017/18 г.

2 ноември 2017 г.

Наредени двойки

```
(A . B)
```



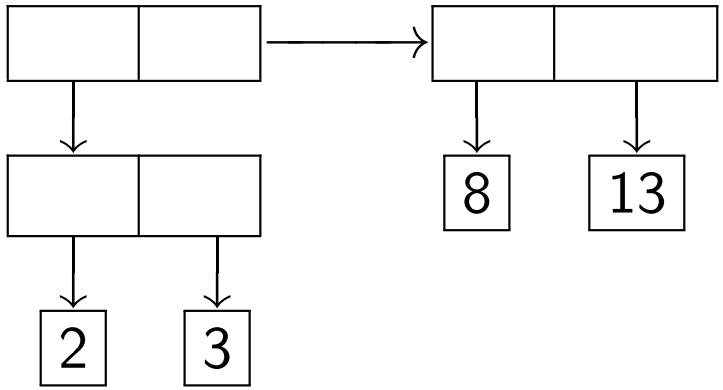
- `(cons` <израз$_1$> <израз$_2$>`)`
- Наредена двойка от оценките на <израз$_1$> и <израз$_2$>
- `(car` <израз>`)`
- **Първият** компонент на двойката, която е оценката на <израз>
- `(cdr` <израз>`)`
- **Вторият** компонент на двойката, която е оценката на <израз>
- `(pair?` <израз>`)`
- Проверява дали оценката на <израз> е наредена двойка

Примери

```
(cons (cons 2 3) (cons 8 13))
```

↓

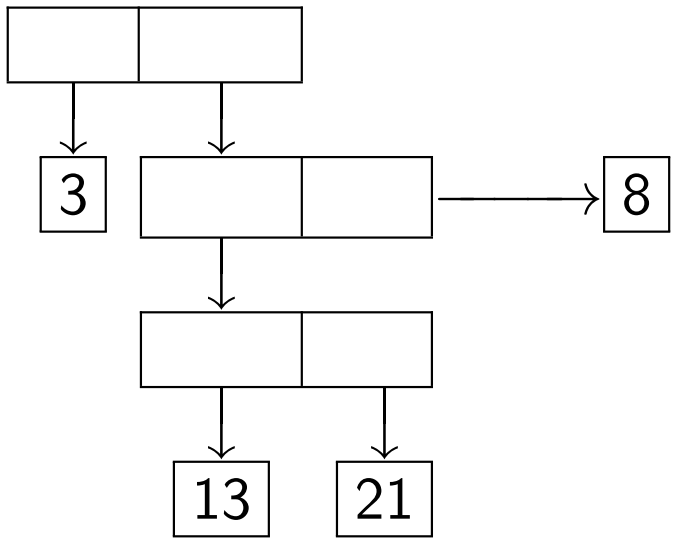
```
((2 . 3) . (8 . 13))
```




```
(cons 3 (cons (cons 13 21) 8))
```

↓

```
(3 . ((13 . 21) . 8))
```



All you need is λ — наредени двойки

Можем да симулираме `cons`, `car` и `cdr` чрез `lambda`!

Вариант №1:

```
(define (lcons x y) (lambda (p) (if p x y)))
(define (lcar z) (z #t))
(define (lcdr z) (z #f))
```

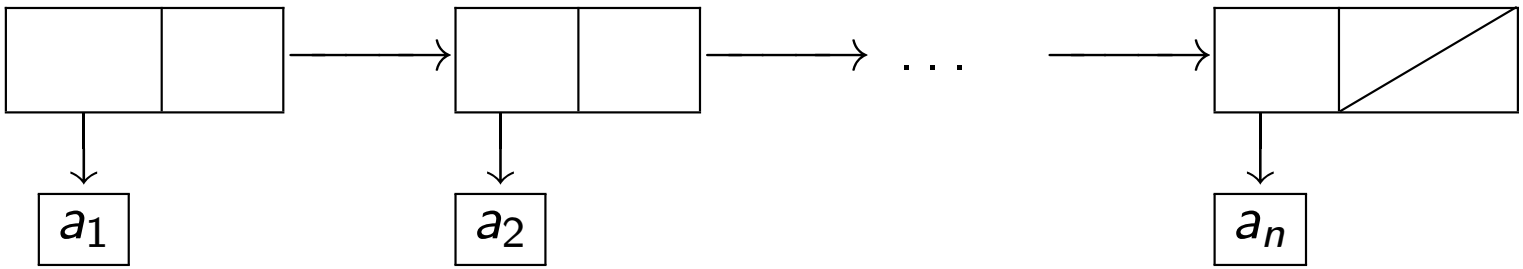
Вариант №2:

```
(define (lcons x y) (lambda (p) (p x y)))
(define (lcar z) (z (lambda (x y) x)))
(define (lcdr z) (z (lambda (x y) y)))
```

Списъци в Scheme

Дефиниция

1. Празният списък `()` е списък
2. `(h . t)` е списък ако `t` е списък
   - `h` — глава на списъка
   - `t` — опашка на списъка



$$(a_1 \ . \ (a_2 \ . \ (\ \ldots (\ a_n \ . \ () \ ) \ ) \ ) \ ) \iff (a_1 \ a_2 \ \ldots \ a_n)$$

Вградени функции за списъци

- (null? <израз>) — дали <израз> е празният списък ()
- (list? <израз>) — дали <израз> е списък
  - (define (list? l) (or (null? l) (and (pair? l) (list? (cdr l)))))
- (list {<израз>}) — построява списък с елементи <израз>
- (list <израз$_1$> <израз$_2$> ... <израз$_n$>) $\Longleftrightarrow$
  (cons <израз$_1$> (cons <израз$_2$> ... (cons <израз$_n$> '()))))
- (cons <глава> <опашка>) — списък с <глава> и <опашка>
- (car <списък>) — главата на <списък>
- (cdr <списък>) — опашката на <списък>
- () не е наредена двойка!
- (car '()) $\longrightarrow$ Грешка!, (cdr '()) $\longrightarrow$ Грешка!

Съкратени форми на car и cdr

Нека $l = (a_1\ a_2\ a_3\ \ldots\ a_n)$.

- `(car l)` $\longrightarrow a_1$
- `(cdr l)` $\longrightarrow (a_2\ a_3\ \ldots\ a_n)$
- `(car (cdr l))` $\longrightarrow a_2 \longleftarrow$ `(cadr l)`
- `(cdr (cdr l))` $\longrightarrow (a_3\ \ldots\ a_n) \longleftarrow$ `(cddr l)`
- `(car (cdr (cdr l)))` $\longrightarrow a_3 \longleftarrow$ `(caddr l)`
- имаме съкратени форми за до 4 последователни прилагания на `car` и `cdr`

Форми на равенство в Scheme

- `(eq?` <израз$_1$> <израз$_2$>`)` — връща `#t` точно тогава, когато оценките на <израз$_1$> <израз$_2$> заемат едно и също място в паметта
- `(eqv?` <израз$_1$> <израз$_2$>`)` — връща `#t` точно тогава, когато оценките на <израз$_1$> и <израз$_2$> заемат едно и също място в паметта или са едни и същи по стойност **атоми** (дори и да заемат различно място в паметта)
  - Ако `(eq?` <израз$_1$> <израз$_2$>`)`,
    то със сигурност `(eqv?` <израз$_1$> <израз$_2$>`)`
- `(equal?` <израз$_1$> <израз$_2$>`)` — връща `#t` точно тогава, когато оценките на <израз$_1$> и <израз$_2$> са едни и същи по стойност **атоми или наредени двойки**, чиито компоненти са равни в смисъла на `equal?`
  - В частност, `equal?` проверява за равенство на списъци
  - Ако `(eqv?` <израз$_1$> <израз$_2$>`)`,
    то със сигурност `(equal?` <израз$_1$> <израз$_2$>`)`

Вградени функции за списъци

- (length <списък>) — връща дължината на <списък>
- (append {<списък>}) — конкатенира всички <списък>
- (reverse <списък>) — елементите на <списък> в обратен ред
- (list-tail <списък> n) — елементите на <списък> без първите n
- (list-ref <списък> n) — n-ти елемент на <списък> (от 0)
- (member <елемент> <списък>) — проверява дали <елемент> се среща в <списък>
  - По-точно, връща <списък> от <елемент> нататък, ако го има
  - Връща #f, ако <елемент> го няма в <списък>
  - Сравнението на елементи става с equal?
- (memqv <елемент> <списък>) — като member, но сравнява с eqv?
- (memq <елемент> <списък>) — като member, но сравнява с eq?