Асоциативни списъци

доц. Атанас Семерджиев

1

Функция (member)

Общ вид:

```
(member elem L)
```

Ако `elem` се съдържа в `L`, връща подсписъка с начало първото срещане на `elem`.

В противен случай връща `#f`.

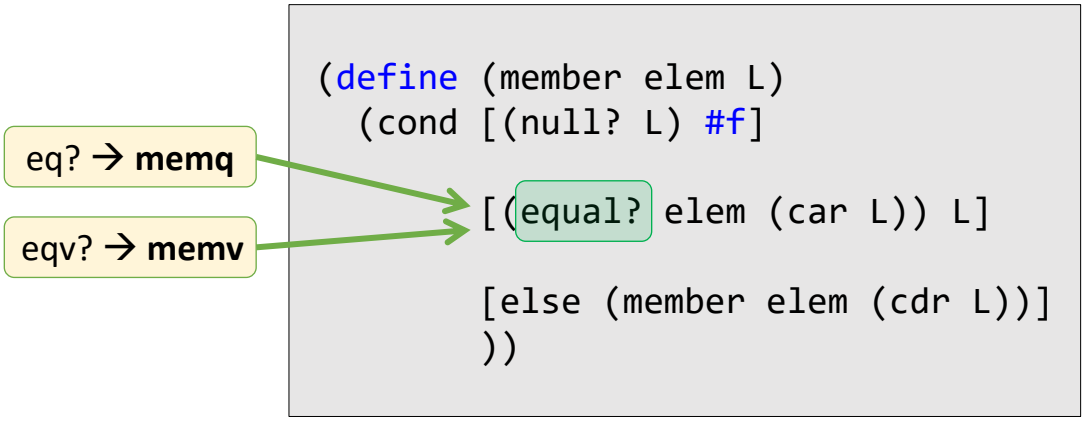
```
(member 'a (list 1 2 'a 5 6))
```
→ `(a 5 6)`

```
(member 'b (list 1 2 'a 5 6))
```
→ `#f`

2

2

Функция (member)



3

3

Дефиниране чрез генерираща функция

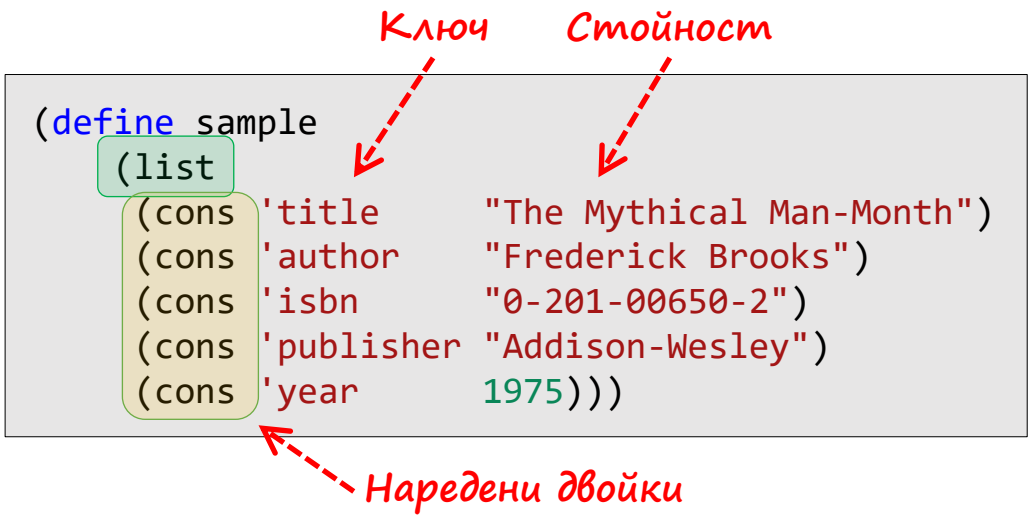
```
(define (generate-member comp?)
    (letrec ([f (lambda (elem L)
                  (cond [(null? L) #f]
                        [(comp? elem (car L)) L]
                        [else (f elem (cdr L))]
                        ))])
      f))

(define member (generate-member equal?))
(define memv   (generate-member eqv?))
(define memq   (generate-member eq?))
```

4

4

Асоциативни списъци



5

5

Извличане на стойност

```
(assoc 'isbn sample)  → (isbn . "0-201-00650-2")
(assoc 'year sample)  → (year . 1975)
```

assoc НЕ връща отделна стойност, а целия запис, който съответства на ключа!

6

6

Функция (assoc)

```
(define (assoc key L)
  (cond [(null? L) #f]
        [(equal? key (caar L)) (car L)]
        [else (assoc key (cdr L))]))
```

eq? → **assq**

eqv? → **assv**

Може да се генерира също както направихме това за (member), (memq), (memv)

7

7

Няколко важни съображения

1. Ключът в един асоциативен списък може да бъде произволен!

```
(define sample
  (list
    (cons 'a 10)
    (cons #t 11)
    (cons "Hello" "world!")
    (cons [cons 1 1] "Hello world")
    (cons [list 5 6 (list 8)] 11)))
```

8

8

Няколко важни съображения

2. Трите функции `assq`, `assv` и `assv` могат да се държат различно върху един и същ списък:

```
(assoc [list 5 6 (list 8)] sample)   → ((5 6 (8)) . 11)
(assq  [list 5 6 (list 8)] sample)   → #f
```

9

9

Няколко важни съображения

3. Един ключ може да бъде свързан с произволен брой стойности.

```
(define sample
  (list
    (list 'a 10 11 12 13)
    (list 'b 14 15)
    (list 'c)))
```

(cons 'b (list 14 15))
е същото като:
(list 'b 14 15)

```
(assq 'a sample)   → (a 10 11 12 13)
(assq 'c sample)   → (c)
```

10

10

Няколко важни съображения

4. По дефиниция, асоциативният списък е списък от наредени двойки! Следователно, и двете дадени по-долу представяния са валидни

```
(define sample
  (list
    (cons 'a 10)
    (cons 'b 11)
 ))
```

```
(define sample
  (list
    (list 'a 10)
    (list 'b 11)
 ))
```

11

11

Няколко важни съображения

5. Когато използваме списъци, можем да имаме ключ, който не е свързан с нито една стойност.

```
(define sample
  (list
    (cons 'a 10)
    (cons 'b) ✗
 ))
```

```
(define sample
  (list
    (list 'a 10)
    (list 'b) ✔
 ))
```

12

12

Няколко важни съображения

6. Ако асоциативният списък съдържа два записа за един и същ ключ, функциите връщат този, който се намира по-напред в списъка!

```
(define sample
  (list
    (list 'a 10)
    (list 'b 11)
    (list 'b 12)
 ))
```

```
(assq 'b sample)
```

↓

```
(b 11)
```

13

13

Няколко важни съображения

Следствие: Добавянето на нов запис с ключ X към асоциативен списък скрива предишните записи с ключ X.

1. `(define L1 (list '(a 10) '(b 11)))`
2. `(assq 'a L1)` → `(a 10)`
3. `(define L2 (cons '(a 20) L1))`
4. `(assq 'a L2)` → `(a 20)`

14

14

Добавяне на запис

Вариант 1: С маскиране на предишните стойности.

```
(define (update-entry key value L)
  (cons (cons key value) L))
```

15

15

Добавяне на запис

- **Вариант 2:** С премахване на предишна стойност (ако има такава).

```
(define (update-entry key value L)
  (cond
    [(null? L) (list (cons key value))]

    [(equal? key (caar L)) (cons (cons key value)
                                 (cdr L))]

    [else (cons (car L)
                (update-entry key value (cdr L)))]))
```

16

16

17