Списъци
част 2: основни операции

доц. Атанас Семерджиев

1

Примерно обхождане на списък

Примерно итеративно обхождане:

1. Проверяваме дали сме достигнали неговия край чрез `null?`.
2. Взимаме всеки пореден елемент чрез `car`.
3. Продължаваме рекурсивно върху останалите елементи с помощта на `cdr`.

Полезно правило: когато обхождаме списък `cdr` винаги връща списък, докато `car` връща елемент.

2

2

```
; Извежда елементите на списъка L
(define (print L)
   (if (null? L)
       (display "End")
       (begin
           (display (car L))
           (display #\space)
           (print (cdr L))
       )
))
```

3

3

```
; Извежда елементите на списъка L
(define (print L)

  (define (loop L)
    (cond ((pair? L)
           (display ", ")
           (display (car L))
           (loop (cdr L)) )))

  (cond ((null? L)
         (display "{empty}"))
        (else
         (display #\{)
         (display (car L))
         (loop (cdr L))
         (display #\}))))
```

```
> (print '(1 2 3 4))
{1, 2, 3, 4}

> (print '())
{empty}
```

4

4

```
; Намира дължината на L рекурсивно
(define (length-rec L)
   (if (null? L) 0
       (+ 1 (length (cdr L)))
   )
)
```

```
; Итеративно решение
(define (length-iter L)
  (define (loop L result)
    (if (null? L) result
        (loop (cdr L) (+ 1 result))))

  (loop L 0)
)
```

5

5

Динамично генериране на списък

Списъците могат да бъдат генерирани динамично. Това става чрез последователно влагане на наредени двойки с помощта на `cons`.

Важно: Не забравяйте да добавите **`()`** в края.

6

6

```
; Генерира списък от всички цели числа в интервала [a, b]
(define (interval a b)
   (if (> a b) '()
       (cons a
             (interval (+ a 1) b)))
)
```

7

7

```
(define (accumulate op term init a next b)
   (define (loop i)
      (if (<= i b)
          (op (term i) (loop (next i)))
          init))

   (loop a)
)
```


```
(define (interval a b)
  (accumulate cons id '() a 1+ b)
)
```

8

8

```
; Генерира списък от всички четни числа в интервала [a, b]
(define (collect-even a b)
   (cond ((> a b) '())

         ((even? a) (cons a
                          (collect-even (+ a 1) b) ))

         (else (collect-even (+ a 1) b))
))
```

9

9

```
; Генерира списък от всички четни числа в интервала [a, b]
(define (collect-even a b)

  (define (op i result)
    (if (even? i)
        (cons i result)
        result))

  (accumulate op id '() a 1+ b)
)
```

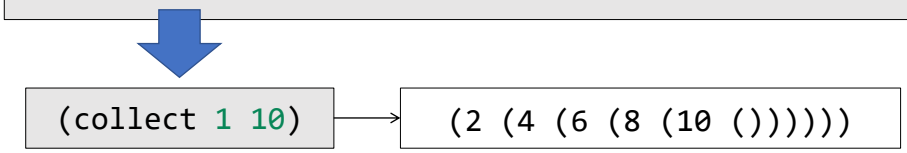
10

10

(list) vs (cons)

Забележете, че НЕ можем да използваме `list` вместо `cons`:

```
(define (collect a b)
   (cond ((> a b) '())
         ((even? a) (list a
                          (collect (+ a 1) b)))
         (else (collect (+ a 1) b)) ))
```



11

11

Конкатенация на два списъка

`append` е вградена функция.

Възможна имплементация:

```
(define (append L1 L2)
   (if (null? L1) L2
       (cons (car L1)
             (append (cdr L1) L2))
))
```

12

12

```
; Конкатенира списъците L1 и L2 итеративно.
(define (append L1 L2)
   (define (loop L result)
      (if (null? L) result
          (loop (cdr L) (cons (car L) result)) ))

   (loop (reverse L1) L2)
)
```

13

13

Обръщане на списък

`reverse` е вградена функция.

Възможна имплементация:

```
; Обръща наопаки списъка L
(define (reverse L)
  (define (loop L result)
    (if (null? L) result
        (loop (cdr L) (cons (car L) result)) ))

  (loop L '())
)
```

14

14

Обхождане на няколко нива

В редица задачи ни се налага да работим със списъци, които съдържат други списъци. Например:

```
( (1 2) (3 4) (((5))) (6 (7 8)) )
```

Нещо повече, един списък може да съдържа едновременно както атоми, така и наредени двойки и други списъци:

```
( 1 2 "Hello world!" (1 2) ((("Hello"))) )
```

15

15

Обхождане на няколко нива

Пример за такава задача е т.нар. изглаждане.

При нея имаме произволен списък и искаме да извлечем от него всички атомарни стойности, независимо на какво ниво се намират.

Например:

```
(flatten '((1 2 (3)) ((("abc")))))  →  (1 2 3 "abc")
```

16

16

Обхождане на няколко нива

При този тип задачи списъкът се обхожда както обикновено по дължина, но за всеки негов елемент:

- Ако е атом, го добавяме към резултата;
- Ако е вложен списък, прилагаме операцията и върху него, в дълбочина.

17

17

```
(define (flatten L)
  (cond
    ((null? L) '())

    ((pair? (car L))
      (append (flatten (car L))
              (flatten (cdr L)) ))
    (else
      (cons (car L)
            (flatten (cdr L)) ))
))
```

18

18

19