Практические задания

1. Написать функцию, которая по своему списку-аргументу lst определяет, является ли он палиндромом (то есть равны ли lst и (reverse lst)). Списки одноуровневые.

1. Проверка на равенство исходного списка и инвертированного исходного списка.

```
(defun is_palindrome (lst)
  (equalp lst (reverse lst))
)
```

2. Проверка на равенство первой половины исходного списка и инвертированной второй половины исходного списка (если список нечетной длины, то центральный элемент не попадает ни в первый, ни во второй список) (этот вариант и было предложено реализовать).

(nthcdr n lst) выполняет для списка lst операцию cdr n раз, и возвращает результат. (floor n) усекает значения по нижней границе. (ceiling n) усекает значения по верхней границе.

```
(defun first_n (n lst)
  (cond
    ((or (null lst) (= n 0)) Nil)
    (t (cons
          (car lst)
          (first_n (- n 1) (cdr lst))
       )
    )
  )
)

(defun is_palindrome (lst)
  (let ((half_len (/ (length lst) 2)))
    (equalp
      (first_n (floor half_len) lst)
      (reverse (nthcdr (ceiling half_len) lst))
    )
  )
)
```

3. Рекурсивно: сравнить первый и последний элемент исходного списка, первый и последний элемент исходного списка без первого и последнего элемента, и так далее (если длина списка нечетная, то центральный элемент ни с чем не сравнивается).

```
(defun list_without_last (lst)
  (cond
    ((null (cdr lst)) Nil)
    (t (cons
        (car lst)
        (list_without_last (cdr lst))
      )
    )
  )
)

(defun is_palindrome (lst)
  (cond
    ((null (cdr lst)) t)
    ((eql (car lst) (car (last lst))) ;т. к. (last '(1 2))=>(2)
      (is_palindrome (list_without_last (cdr lst))))
  )
)
```

Все варианты функций проверялись на следующих тестах:

```
(is_palindrome Nil) => T
(is_palindrome '(1)) => T
(is_palindrome '(1 2 3)) => NIL
(is_palindrome '(1 2 1)) => T
(is_palindrome '(1 2 3 1)) => NIL
(is_palindrome '(1 2 2 1)) => T
```

2. Написать предикат set-equal, который возвращает t, если два его множества-аргумента содержат одни и те же элементы, порядок которых не имеет значения

Все элементы первого множества последовательно удаляются из обоих множеств. Если исходные множества эквиваленты, то в конце получим два пустых множества.

```
(defun set-equal (set1 set2)
  (cond
    ((null set1) (null set2)) ;3 тест
    ((null set2) Nil) ;4 тест
    (t (set-equal (cdr set1) (remove (car set1) set2)))
  )
)

(set-equal '(1 2 3) '(1 2 3)) => T
(set-equal '() '()) => T
(set-equal '(1 2) '(1 2 3)) => NIL
(set-equal '(1 2) '(1)) => NIL
```

3. Напишите свои необходимые функции, которые обрабатывают таблицу из 4-х точечных пар: (страна . столица), и возвращают по стране - столицу, а по столице — страну.

1. Используя информацию о том, что в таблице ровно 4 точечные пары.

```
(defun find_capital_by_country (table country)
  (cond
    ((eql (caar table) country) (cdar table))
    ((eql (caadr table) country) (cdadr table))
    ((eql (caaddr table) country) (cdaddr table))
    ((eql (caaddr (cdr table)) country) (cdaddr (cdr table)))
  )
)

(defun find_country_by_capital (table capital)
  (cond
    ((eql (cdar table) capital) (caar table))
    ((eql (cdadr table) capital) (caadr table))
    ((eql (cdaddr table) capital) (caaddr table))
    ((eql (cdaddr (cdr table)) capital) (caaddr (cdr table)))
  )
)
```

2. Используя функционал some.

(some #'test lst1 ... lstn) выполняет действия предиката test над car-элементами списков lst1,...,lstn, затем - над cadr-обьектами каждого списка и т.д. до тех пор, пока тест не вернет значение, отличное от Nil, или не встретится конец списка. Если тест возвращает значение, отличное от Nil, функция some возвращает это значение, если же конец списка достигнут, функция some возвращает Nil.

```
(defun find_capital_by_country (table country)
  (some
    #'(lambda (row) (cond ((eql (car row) country) (cdr row))))
    table
  )
)

(defun find_country_by_capital (table capital)
  (some
    #'(lambda (row) (cond ((eql (cdr row) capital) (car row))))
    table
  )
)
```

3. Используя функции assoc/rassoc.

assoc (rassoc) выбирает из ассоциативного списка, заданного вторым аргументом, первую точечную пару, в которой первый (второй) элемент совпадает со значением первого аргумента.

```
(defun find_capital_by_country (table country)
  (cdr (assoc country table))
)

(defun find_country_by_capital (table capital)
  (car (rassoc capital table))
)
```

Все варианты функций проверялись на следующих тестах:

```
(defvar table)
(setq table '((Russia . Moscow) (GreatBritain . London)
                    (France . Paris) (Italy . Roma)))

(find_capital_by_country table 'Russia) => MOSCOW
(find_capital_by_country table 'Italy) => ROMA
(find_capital_by_country table 'USA) => NIL

(find_country_by_capital table 'Moscow) => Russia
(find_country_by_capital table 'Paris) => FRANCE
(find_country_by_capital table 'Washington) => NIL
```

5. Напишите функцию swap-two-element, которая переставляет в списке-аргументе два указанных своими порядковыми номерами элемента в этом списке.

1. Собирать список из элементов исходного списка в следующем порядке:

- элементы, которые стоят до индекса min(index1 index2) (не включительно);
- элемент с индексом max(index1 index2);
- элементы, которые стоят между индексами min(index1 index2) и max(index1 index2) (не включительно);
- элемент с индексом min(index1 index2);
- элементы, которые стоят после индекса max(index1 index2) (не включительно);

```
; элемент n не входит в результат
(defun list_till_n (n lst)
  (cond
    ((or (null lst) (= n 0)) Nil)
    (t (cons (car lst) (first_n (- n 1) (cdr lst))))
  )
)

; элементы to и from не входят в результат
(defun list_slice_from_to (lst from to)
  (nthcdr (+ from 1) (list_till_n to lst))
)

; в этой функции index1 < index2
(defun swap-two-element-inner (lst index1 index2)
  (nconc
    (list_till_n index1 lst)
    (cons (nth index2 lst) (list_slice_from_to lst index1 index2))
    (cons (nth index1 lst) (nthcdr (+ index2 1) lst))
  )
)
```

```
(defun swap-two-element (lst index1 index2)
  (cond
    ((and (< index1 index2) (< -1 index1) (< index2 (length lst)))
      (swap-two-element-inner lst index1 index2)
    )
    ((and (> index1 index2) (< -1 index2) (< index1 (length lst)))
      (swap-two-element-inner lst index2 index1)
    )
    ((= index1 index2) lst)
  )
)
```

2. Рекурсивно записывать в конец результирующего списка голову исходного списка, голову хвоста исходного списка и так далее. Если достигли элемента с индексом index1 (index2), то записать вместо головы элемент с индексом index2 (index1). В конце инвертировать результирующий список

(a) Без использования nconc, рекурсивно доходя до конца исходного списка

```
; в этой функции index1 < index2
(defun swap-two-element-inner (lst index1 index2 res elem1)
  (cond
    ((null lst) (reverse res))
    ((= index1 0) (swap-two-element-inner (cdr lst) (- index1 1) (-
       index2 1) (cons (nth index2 lst) res) elem1))
    ((= index2 0) (swap-two-element-inner (cdr lst) index1 (- index2 1)
        (cons elem1 res) elem1))
    (t (swap-two-element-inner (cdr lst) (- index1 1) (- index2 1) (
       cons (car lst) res) elem1))
  )
)

(defun swap-two-element (lst index1 index2)
  (cond
    ((and (< index1 index2) (< -1 index1) (< index2 (length lst)))
      (swap-two-element-inner lst index1 index2 () (nth index1 lst))
    )
    ((and (> index1 index2) (< -1 index2) (< index1 (length lst)))
      (swap-two-element-inner lst index2 index1 () (nth index2 lst))
    )
    ((= index1 index2) lst)
  )
)
```

(б) С использованием ncon, рекурсивно доходя только до элемента с индексом index2 (swap-two-element-inner не заменяет элемент с индексом index2 на элемент с индексом index1).

```
; в этой функции index1 < index2
(defun swap-two-element-inner (lst index1 index2 res)
  (cond
    ((= index2 0) (reverse res))
    ((= index1 0) (swap-two-element-inner (cdr lst) (- index1 1) (-
       index2 1) (cons (nth index2 lst) res)))
    (t (swap-two-element-inner (cdr lst) (- index1 1) (- index2 1) (
       cons (car lst) res)))
  )
)

(defun swap-two-element (lst index1 index2)
  (cond
    ((and (< index1 index2) (< -1 index1) (< index2 (length lst)))
      (nconc (swap-two-element-inner lst index1 index2 ()) (cons (nth
         index1 lst) (nthcdr (+ index2 1) lst)))
    )
    ((and (> index1 index2) (< -1 index2) (< index1 (length lst)))
      (nconc (swap-two-element-inner lst index2 index1 ()) (cons (nth
         index2 lst) (nthcdr (+ index1 1) lst)))
    )
    ((= index1 index2) lst)
  )
)
```

Все варианты функций проверялись на следующих тестах:

```
(swap-two-element '(0 1 2 3 4 5 6 7) 2 4) => (0 1 4 3 2 5 6 7)
(swap-two-element '(0 1 2 3 4 5 6 7) 0 7) => (7 1 2 3 4 5 6 0)
(swap-two-element '(0 1 2 3 4 5 6 7) -1 2) => Nil
(swap-two-element '(0 1 2 3 4 5 6 7) 1 8) => Nil
(swap-two-element '(0 1 2 3 4 5 6 7) 5 3) => (0 1 2 5 4 3 6 7)
(swap-two-element '(0 1 2 3 4 5 6 7) 3 3) => (0 1 2 3 4 5 6 7)
```

Литература