

**1. В структурированном (сложном списке, в котором другие списки) один заданный элемент заменить на другой. Например 1 на 6.**

```
(setf lst1 (2 (10 2 3) 4 5 1 () (2 3 2) 2))
```

```
(defun func_cov (lst a b) (func lst a b lst))
```

```
(defun func (lst a b lst_help)
  (
    cond ((null lst) lst_help)
          ((listp (car lst)) (and (func (car lst) a b lst_help) (func (cdr lst) a b lst_help)))
          ((= (car lst) a) (func (cdr (rplaca lst b)) a b lst_help))
          (t (func (cdr lst) a b lst_help)))
  )
)

(defun ffunc (lst a b)
  (mapcar #'(lambda (x) (if (listp x) (ffunc x a b) (if (= x a) b x))) lst)
)
```

**2. посчитать среднее значение чисел в списке, которые на нечетных позициях. Когда используешь функционалы, рекомендуют делать с редусом и скакойто его настройкой инит-валю-чего-то-там, чтоб было эффективно.**

```
(defun avg_cov (lst) (avg lst 0 0 0))
```

```
(defun avg (lst pos count sum)
  (
    cond ((null lst) (/ sum count))
          ((oddp pos) (avg (cdr lst) (+ pos 1) (+ count 1) (+ sum (car lst))))
          (t (avg (cdr lst) (+ pos 1) count sum)))
  )
)
```

## Билет 2

**Найти A/B где A и B числовые множества полученные из 2х одноуровневых списков(хз зачем это)**

```
(defun f (lst1 lst2)
  (remove-if (lambda (x) (member x lst2 :test #'equalp)) lst1)
)
```

```
(defun f (lst1 lst2 res)
  (let ((a (car lst1)))
    (if (cdr lst1)
      (if
        (member a lst2 :test #'equalp)
        (f (cdr lst1) lst2 res)
        (f (cdr lst1) lst2 (cons a res)))
      res)
  )
)
```

```
)
```

## 2. Удаление элементов в структурированном списке.

```
(defun myrem (lst el)
  (let((ls (delete el lst :test #'equalp)))
    (cond
      ((null ls) nil)
      ((listp(car ls))(cons(myrem (car ls) el)(myrem (cdr ls) el)))
      (t(cons (car ls)(myrem (cdr ls) el)))
    )
  )
)
```

```
(defun myfunk(X El)
  (if (equalp x el)
      nil
      (if(listp x)
          (list (myrem x el))
          (list x)
        )
    )
)
```

```
(defun myrem (lst el)
  (mapcan #'(lambda(X)(myfunk X El)) lst
  )
)
```

## Билет 8

### 1. Реализовать добавление в ассоциативный список нескольких точечных пар, заданных списком ключей и списком значений. (своя функция)

С рекурсией:

```
(setq lst_help '((1 . 1val)(2 . 2val)(3 . 3val)(4 . 4val)))
```

```
(defun add_to_lst (lst couple)
  (nconc lst (list couple))
  (print lst)
)
```

```
(defun add_item (lst a b)
  (add_to_lst lst (cons a b))
)
```

```
(defun add_items (lst key_list val_list)
  (if (car key_list)
      (and (add_item lst (car key_list) (car val_list))
           (add_items lst (cdr key_list) (cdr val_list))
      )
      (print lst)
  )
)
```

```
)
С функционалом:
(defun add_items2 (lst key_list val_list)
  (nconc lst (mapcar #'(lambda (x y) (cons x y)) key_list val_list))
)
```

**2. Дан смешанный структурированный список. Получить отсортированный по возрастанию список из числовых элементов исходного списка, входящих в заданное в виде одноуровневого смешанного списка множество.**

```
(setq task_help '((3 2 1) f h (t 6) ((u) 7 0)))
(defun into_one (lst rst)
  (cond ((null lst) rst)
        ((atom lst) (cons lst rst))
        (t (into_one (car lst) (into_one (cdr lst) rst)) )
  )
)

(defun into_one_level (lst)
  (into_one lst ())
)

(defun get_num_list (lst)
  (remove-if-not #'numberp (into_one_level lst))
)

(defun insert_help (x lst)
  (cond ((null lst) (list x))
        ((<= x (car lst)) (cons x lst))
        (t (cons (car lst) (insert_help x (cdr lst)))))
  )
)

(defun sort_help (lst1 lst2)
  (cond ((null lst1) lst2)
        (t (sort_help (cdr lst1) (insert_help (car lst1) lst2))))
  )
)

(defun my_sort (lst)
  (sort_help lst ())
)

(defun task2(lst)
  (my_sort (get_num_list lst))
)
)
```

**Примечание:** тут есть и рекурсии, и функционалы. Признаться честно, я не в курсе, как разворачивать вложенные списки функционалом в обход рекурсии - в голове все равно получается мысль о рекурсии функционалов. Так что здесь имеет место сделать ссылку на то, что в задании не сказано двумя способами делать. Если будут идеи, готов помочь реализовать.

**Билет 1**

- 1. Все числовые элементы исходного смешанного одноуровневого списка удвоить, если сумма его первых двух числовых элементов больше 10 и уменьшить на 10 в противном случае (Использовать функционалы, Использовать рекурсию)**

Рекурсия:

```
(defun check-help (lst d)
  (cond ((endp lst) 0)
        ((and (numberp (car lst)) (eql d 1)) (car lst))
        ((and (numberp (car lst)) (eql d 0)) (+ (car lst) (check-help (cdr lst) 1)))
        (t (check-help (cdr lst) d))))

(defun sum-check (lst)
  (check-help lst 0))

(defun func (cnd x)
  (if (> cnd 10)
      (* x 2)
      (- x 10)))

(defun calc-help (lst sc)
  (cond ((null lst) nil)
        ((numberp (car lst)) (cons (func sc (car lst)) (calc-help (cdr lst) sc)))
        (t (cons (car lst) (calc-help (cdr lst) sc)))))

(defun calc (lst)
  (calc-help lst (sum-check lst)))
```

Функционалы:

```
(defun sum-check (lst)
  (let ((nlist (remove-if-not #'numberp lst)))
    (reduce #'+ (list (car nlist) (cadr nlist)))))

(defun calc (lst)
  (let ((sc (sum-check lst)))
    (mapcar #'(lambda (x) (if (numberp x) (if (> sc 10) (* x 2) (- x 10)) x)) lst)))
```

## Билет 9

- 1. Все числовые элементы исходного смешанного одноуровневого списка удвоить, если сумма его первых двух числовых элементов больше 10 и уменьшить на 10 в противном случае (Использовать функционалы, Использовать рекурсию)**

```
(defun all_minus_10 (lst)
  (mapcar #'(lambda (x)
    (cond ((numberp x) (- x 10))
          ((listp x) (all_minus_10 x))
          (T x))
```

```

    )
  )
  lst
)
)

(defun all_mult_2 (lst)
  (mapcar #'(lambda (x)
    (cond ((numberp x) (* x 2))
          ((listp x) (all_mult_2 x))
          (T x))
    )
  )
  )
  lst
)

)

(defun two_number_10 (lst sum)
  (cond
    (( and (numberp (car lst)) (> sum 0) ) (+ sum (car lst)))
    ((numberp (car lst)) (two_number_10 (cdr lst) (+ sum (car lst))) )
    (T (two_number_10 (cdr lst) sum))
  )
)

)

(defun task1 (lst)
  (cond ( (equal lst NIL) NIL )
        ( (> (two_number_10 lst 0) 10) (all_mult_2 lst) )
        ( (<= (two_number_10 lst 0) 10) (all_minus_10 lst) )
  )
)

)

```

## **2. Даны два структурированных смешанных списка. Получить из этих списков числовые множества (одноуровневые списки) и найти пересечение этих двух множеств (Использовать функционалы, Использовать рекурсию.)**

```

(defun in-list (a lst)
  (cond
    ((null lst) NIL)
    ((eq a (car lst)) T)
    (T (in-list a (cdr lst)))
  )
)

)

(defun mult_of_lists (a b)
  (cond
    ((null a) NIL)
    ((null b) NIL)
    ((in-list (car a) b) (cons (car a) (mult_of_lists (cdr a) b)) )
    (T (mult_of_lists (cdr a) b))
  )
)

)

(defun lst_to_enum (lst enum)
  (mapcan #'(lambda (x)
    (cond ((numberp x) (cons x enum))

```

```

                (T NIL)
            )
        )
    lst
)

)

(defun task2 (lst1 lst2)
  (mult_of_lists (lst_to_enum lst1 NIL) (lst_to_enum lst2 NIL))
)

```

## Билет 11.

**Вычислить  $n!$ , где  $n$  - кол-во чисел в одноуровневом смешанном списке**

**Рекурсия:**

```

(defun d_f_r (n)
  (if
    (> n 3)
    (* n (d_f_r (- n 2)))
    n
  )
)

```

Сделал через `reduce`, предварительно создав список. Увы, но больше идей в голову не приходит, как реализовать эту программу через функционалы. Проще через рекурсию делать)

```

(defun my_cons (n)
  (if
    (> n 1) (cons n (my_cons (- n 2)))
  )
)

(defun d_f_f (n)
  (if
    (> n 3) (reduce #'* (my_cons n))
    n
  )
)

```

**2. Есть 2 списка с подсписками. В нем могут быть и числа, и буквы. Надо сделать из них множества из чисел, а потом найти их пересечение.**

**Сделать с помощью рекурсии и функционалов.**

```

(defun into_one (lst rst)
  (cond ((null lst) rst)
        ((atom lst) (cons lst rst))
        (t (into_one (car lst) (into_one (cdr lst) rst)) )
  )
)

(defun into_one_level (lst)
  (into_one lst ())
)

```

```

(defun get_num_list (lst)
  (remove-if-not #'numberp (into_one_level lst))
)

(defun consist-of (lst)
  (if (member (car lst) (cdr lst)) 1 0)
)

(defun all-last-element (lst)
  (if (eql (consist-of lst) 0)
      (list (car lst)())
  )
)

(defun collections-to-set (lst)
  (remove-if #'(lambda(x)(if (equal x nil) t nil))(mapcon #'all-last-element lst))
)

(defun intersect(a b)
  (remove-if #'(lambda(x)
    (if (equal x nil) t nil)
  )
    (mapcar #'(lambda(x)
      (if (and (member x a) (member x b)) x nil)
    )
      a))
)

(defun task2(lst1 lst2)
  (intersect (collections-to-set(get_num_list lst1))
    (collections-to-set(get_num_list lst2)))
)

```

**Примечание: здесь развертывание в одноуровневый список происходит без помощи функционалов.**

**А вот здесь с функционалами;**

```

(defun get_num_list (lst)
  (remove-if-not #'numberp (into_one_level lst))
)

(defun consist-of (lst)
  (if (member (car lst) (cdr lst)) 1 0)
)

```

)

```
(defun all-last-element (lst)
  (if (eql (consist-of lst) 0)
      (list (car lst))
      )
  )
```

)

```
(defun collections-to-set (lst)
  (remove-if #'(lambda(x)(if (equal x nil) t nil))(mapcon #'all-last-element lst))
  )
```

```
(defun intersect(a b)
  (remove-if #'(lambda(x)
    (if (equal x nil) t nil)
    )
    (mapcar #'(lambda(x)
    (if (and (member x a) (member x b)) x nil)
    )
    a))
  )
```

```
(defun task2(lst1 lst2)
  (intersect (collections-to-set(get_num_list lst1))
    (collections-to-set(get_num_list lst2)))
  )
```

```
(defun into_one2 (lst predicat)
  (reduce
    #'nconc
    (mapcar
      #'(lambda (x)
        (cond
          ((atom x)
           (cond
             ((funcall predicat x) (list x))
             (T nil)
            )
          )
      )
    )
    (T (into_one2 x predicat))
  )
```



```

        )
        lst
    )
)
(defun into_one_level2 (lst)
  (values
    (nconc (into_one2 lst #'(lambda (x) (and (numberp x))))
    (into_one2 lst #'symbolp))
  )
)

```

### Билет 13

#### Номер1.

**Простите! Но тут творится какая-то хуйня, но работает.**

**Запускать программу один раз, после надо перезапускать lisp.**

```

(defun is_in(lst x)
  (
    cond ((null lst) nil)
          ((= x (car lst)) t)
          (t (is_in (cdr lst) x))
  )
)

(defun insert-asc (elem cur next)
  (cond
    ((null next) (rplacd cur (cons elem Nil)))
    ((<= elem (car next)) (rplacd cur (cons elem next)))
    (t (insert-asc elem next (cdr next)))
  )
  ;(rplacd cur (cons elem next))
)

```

```

(defun insert-sort (src result)
  (cond
    ((null src) (cdr result))
    (t
      (and
        (insert-asc (car src) result (cdr result))
        (insert-sort (cdr src) result)
      )
    )
  )
)

```

```
)  
  
)  
)  
)  
  
(defun numbers(lst1 lst2 s_p f_p pos)  
  (let ((st_ (car lst1))  
        (l_res nil)  
        )  
    (  
      cond ((null lst1) nil)  
            ((numberp st_) (cond ((oddp st_) (cond ((is_in lst2 st_)  
  
(cond  
  
              ((<= s_p pos f_p) (cons st_ (numbers (cdr lst1) lst2 s_p f_p (+ pos 1))))  
  
              (t (numbers (cdr lst1) lst2 s_p f_p (+ pos 1)))  
  
            )  
  
          ))  
          (t (numbers (cdr lst1) lst2 s_p f_p (+ pos  
1))))  
  
        )  
        )  
  
      (t (numbers (cdr lst1) lst2 s_p f_p (+ pos 1)))  
    )  
  )  
)  
  
(defun get_numbers(lst1 lst2 s_p f_p)  
  (insert-sort (numbers lst1 lst2 s_p f_p 0))  
)
```

```
(sort (remove-if #'(lambda(x)(or(or(> x1 x)(< x2 x))(not(member x
B))))(remove-if #'(lambda(x)(or(symbolp x)(evenp x))) A)) #'<=)
)
```

**Номер 2. Подсчитать в структурированном, смешанном списке количество символьных элементов, принадлежащих множеству, заданному в виде одноуровневого списка.**

Рекурсия:

```
(defun zad2rec (lst settt)
  (cond
    ((null lst) 0)
    ((listp lst)
     (+ (zad2rec (car lst) settt) (zad2rec (cdr lst) settt)))
    )
    (t
     (if (member lst settt)
         1
         0)
     )
  )
)
```

Функционалы:

```
(defun zad2func (lst settt)
  (reduce #'+ (mapcar #'(lambda (x)
    (cond
      ((listp x) (zad2func x settt))
      (t
       (if (member x settt)
           1
           0)
       )
      )
    )
    lst
  )
)
```

**Билет 14.**

**Номер 1. Реализовать выделение из ассоциативной таблицы с числовыми ключами элементов , стоящих на нечетных позициях и уменьшить все ключи результирующей таблицы на количество элементов в ней.**

Рекурсив:

```
(defun get-odd (lst)
```

```

      (cond
        ((null lst) Nil)
        ((null (cdr lst)) Nil)
        (t (cons (car lst) (get-odd (cddr lst)))))
      )
    )
  )

```

```

(defun mi10 (lst head vichitaemoe)
  (cond
    ((null lst) head)
    (t (and (rplaca lst `(- (caar lst) vichitaemoe) . ,(cdar lst))) (mi10 (cdr lst)
head vichitaemoe))))
  )
)

```

```

(defun len (table)
  (cond ((null table) 0)
    (t (+ 1 (len (cdr table)))))
  )
)

```

```

(defun bil6zad1rec (table)
  (let
    (
      (
        (mita (get-odd table))
      )
      (mi10 mita mita (len mita))
    )
  )
)

```

**функционалы:**

```

(setf table '((1 . 4) (4 . 6) (7 . 8) (9 . 4)))

```

```

(defun len (table)
  (reduce #'(lambda (x y) (1+ x)) table :initial-value 0)
)

```

```

(defun change (table)
  (
    mapcar #'(lambda (x) (cons (- (car x) (len table)) (cdr x))) table
  )
)

```

## Номер 2. Дан смешанный структурированный список. Выделить числа и атомы из него в отдельные списки.

### Функционалы

```
(defun f14p-help (lst predicat)
  (reduce
    #'nconc
    (mapcar
      #'(lambda (x)
        (cond
          ((atom x)
            (cond
              ((funcall predicat x) (list x))
              (T nil)
            )
          )
        )
      )
      (T (f14p-help x predicat))
    )
    lst
  )
)

(defun f14p (lst x1 x2)
  (values
    (f14p-help lst #'(lambda (x) (and (numberp x) (>= x x1) (<= x x2))))
    (f14p-help lst #'symbolp)
  )
)
```

### Рекурсия

```
(defun f14f-help (lst predicat)
  (cond
    ((not (cdr lst))
      (cond
        ((listp (car lst)) (f14f-help (car lst) predicat))
        (T
          (cond
            ((funcall predicat (car lst)) `(. (car lst)))
            (T nil)
          )
        )
      )
    )
    (T
      (cond
        ((listp (car lst)) (append (f14f-help (car lst) predicat) (f14f-help (cdr lst) predicat)))
        (T
          (cond
            ((funcall predicat (car lst)) (append `(. (car lst)) (f14f-help (cdr
lst) predicat))))
            (T (f14f-help (cdr lst) predicat))
          )
        )
      )
    )
  )
)
```

```

)
(defun f14f (lst x1 x2)
  (values
    (f14f-help lst #'(lambda (x) (and (numberp x) (>= x x1) (<= x x2))))
    (f14f-help lst #'symbolp)
  )
)

```

## Билет 15.

Выделить из одноуровневого, смешанного списка числа, сформировать из них множество (в виде одноуровневого списка) и найти количество элементов в нем.

**Рекурсия:**

```

(defun len (lst)
  (cond ((null lst) 0)
        (t (+ 1 (len (cdr lst)))))
)

```

```

(defun my_union (lst count)
  (cond
    ((null lst) (AND (print count) Nil))
    ((numberp (car lst))
     (union (cons (car lst) nil) (my_union (cdr lst) )))
    (t (my_union (cdr lst) count))
  )
)

```

**Функционал:**

```

(defun len (lst)
  (reduce #'(lambda (x y) (1+ x)) lst :initial-value 0)
)

```

```

(defun my_union_func (lst)
  (remove-if #'null

    (maplist #'(lambda (x)
      (cond
        ((member (car x) (cdr x)) Nil)
        (t (car x))
      ))

    (remove-if #'(lambda (x)
      (cond
        ((symbolp x) t)
        (t Nil)
      )
    )

    lst)
  )))

```

**Без билетные:**

## Есть два смешанных множества, оставить в обоих только числа и вычислить их объединение

Функционал:

```
(defun operset(A B)
  (union (remove-if #'(lambda(x)(null x)) (mapcar #'(lambda(x)(if(numberp x)x)) A))
         (remove-if #'(lambda(x)(null x)) (mapcar #'(lambda(x)(if(numberp x)x)) B))
  )
)
```

Рекурсия:

```
(defun opersetR(A B)
  (union
    (if (not(null A))
      (if (numberp (car A))
        (cons (car A) (opersetR (cdr A) B))
        (opersetR (cdr A) B)
      )
      ()
    )
    (if (not(null B))
      (if (numberp (car B))
        (cons (car B) (opersetR A (cdr B)))
        (opersetR A (cdr B))
      )
      ()
    )
  )
)
```