Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 10

Дисциплина	Функциональное и логическое программирование
Студент	Сиденко А.Г.
Группа	ИУ7-63Б
Преподаватель	Толпинская Н.Б., Строганов Ю.В.

Москва, 2020 г.

1. **Пусть list-of-list список, состоящий из списков. Написать функцию, которая вычисляет сумму длин всех элементов list-of-list.**

   На вход список.

   Пока список ненулевой прибавляем к длине головы функцию от хвоста (вернет длину хвоста). Когда дойдет до конца функция вернет 0.

```
1 (defun len_lists(lst)
2   (if lst
3     (+ (length (car lst)) (len_lists (cdr lst)))
4     0
5   )
6 )
```

   Примеры:

```
1 (len_lists '((4 44 3 2 ) (2 3 4) (6)))
```

   Результат: 8

2. **Написать рекурсивную версию (с именем rec-add) вычисления суммы чисел заданного списка.**

   На вход список.

   Идея та же, что и в предыдущем номере, только прибавляем не длину, а само значение.

```
1 (defun rec-add(lst)
2   (if lst
3     (+ (car lst) (rec-add (cdr lst)))
4     0
5   )
6 )
```

   Примеры:

```
1 (rec-add '(1 2 3 4 5))
```

   Результат: 15

3. **Написать рекурсивную версию с именем recnth функции nth.**

   На вход список и число(позиция).

   Функция recnth, вызывает рекурсивную функцию recnth_next с параметрами список и число (длина списка, который должен остаться), которая как раз в свою очередь и находит необходимую нам позицию в списке.

```
(defun recnth_next(lst k)
  (cond
    ((= k (length lst)) (car lst))
    (t (recnth_next (cdr lst) k))
  )
)

(defun recnth(lst k)
  (cond
    ((> k (length lst)) Nil)
    ((< k 1) Nil)
    (t (recnth_next lst (+ 1 (- (length lst) k))))
  )
)
```

Примеры:

```
(recnth '(1 2 3 4 5) 6)
```

Результат: Nil

```
(recnth '(1 (2 1) (3 4 5) 4 5) 3)
```

Результат: (3 4 5)

4. **Написать рекурсивную функцию alloddr, которая возвращает t когда все элементы списка нечетные.**

   На вход список.

   Пока список ненулевой применяем and к проверки на четность головы и функции от хвоста (вернет t или Nil хвоста). Когда дойдет до конца функция вернет t.

```
(defun alloddr(lst)
  (if lst
    (and (oddp (car lst)) (alloddr (cdr lst)))
    t
  )
)
```

Примеры:

```
(alloddr '(1 3 5 7 9))
```

Результат: T

```
(alloddr '(1 2 3 4 5))
```

Результат: Nil

5. **Написать рекурсивную функцию, относящуюся к хвостовой рекурсии с одним тестом завершения, которая возвращает последний элемент списка-аргумента.**

   Пока второй элемент не Nil идем дальше, иначе возвращаем голову

```
(defun last_elem (lst)
  (if (NULL (cadr lst)) (car lst)
    (last_elem (cdr lst))
  )
)
```

   Примеры:

```
(last_elem '((1) (2 4 5) (3 4 5 (6 7)) (4 (-3) 8)))
```

   Результат: (4 (-3) 8)

6. **Написать рекурсивную функцию, относящуюся к дополняемой рекурсии с одним тестом завершения, которая вычисляет сумму всех чисел от 0 до n-ого аргумента функции. Вариант: 1) от n-аргумента функции до последнего; 2) от n-аргумента функции до m-аргумента с шагом d.**

   Дополняемая рекурсия, считающая сумму всех чисел от 0 до n-ого аргумента. Используем дополнительную функцию, так как необходимо преобразовать индекс к длине хвоста. Далее просто прибавляются все числа, пока длина хвоста длиннее аргумента.

   На вход список и число, порядок аргумента.

```
(defun sum_next(lst n)
  (cond
    ((= n (length lst)) (car lst))
    (t (+ (car lst) (sum_next (cdr lst) n)))
  )
)

(defun sumN(lst n)
  (sum_next lst (+ 1 (- (length lst) n)))
)
```

   Примеры:

```
(sumN '(1 2 3 4 5 6 7) 5)
```

Результат: 15

Дополняемая рекурсия, считающая сумму всех чисел от n-ого аргумента до конца. Используем дополнительную функцию, так как необходимо преобразовать индекс к длине хвоста. Далее просто прибавляются все числа, пропускаются те элементы, где длина хвоста длиннее, прибавляем голову, где длина хвоста равна аргументу и далее до конца (длина равна 1).

На вход список и число, порядок аргумента.

```
(defun sum_nextlast(lst n)
  (cond
    ((= 1 (length lst)) (car lst))
    ((>= n (length lst)) (+ (car lst)
        (sum_nextlast (cdr lst) n))
    )
    (t (sum_nextlast (cdr lst) n))
  )
)

(defun sumlastN(lst n)
  (sum_nextlast lst (+ 1 (- (length lst) n)))
)
```

Примеры:

```
(sumlastN '(1 2 3 4 5 6 7) 5)
```

Результат: 18

Дополняемая рекурсия, считающая сумму всех чисел от n-ого аргумента до m-го с шагом d. Используем дополнительную функцию, так как необходимо преобразовать индекс к длине хвоста. Далее просто прибавляются все числа, пропускаются те элементы, где длина хвоста длиннее n, прибавляем голову, где длина хвоста равна n и далее до m, причем прибавляются только числа позиция которых кратна шагу.

На вход список и 3 числа, порядок аргументов и шаг.

```
(defun sum_nextNMD(lst n m d)
  (let ((len (length lst)))
    (cond
      ((= m len)
        (if (= 0 (rem (- n m) 3))
          (car lst)
```

```
7             0
8           )
9         )
10        ((and (>= n len) (= 0 (rem (- n len) 3)))
11          (+ (car lst) (sum_nextNMD (cdr lst) n m d))
12        )
13        (t
14          (sum_nextNMD (cdr lst) n m d)
15        )
16      )
17    )
18  )
19
20  (defun sumNMD(lst n m d)
21    (let ((len (length lst)))
22      (sum_nextNMD lst (+ 1 (- len n)) (+ 1 (- len m)) d)
23    )
24  )
```

Примеры:

```
1  (sumNMD '(1 2 3 4 5 6 7 8 9 10) 2 9 3)
```

Результат: 15

7. **Написать рекурсивную функцию, которая возвращает последнее нечетное число из числового списка, возможно создавая некоторые вспомогательные функции.**

   Рекурсивная функция: список пустой, возвращает Nil; если число четное – возвращается значение рекурсии, если нечетное – проверяется, что вернула рекурсия, если вернула не Nil(было найдено нечетное число), то возвращается ранее найденное число(оно будет в списке позже); если вернула Nil то возвращается само нечетное число.

   На вход список.

```
1  (defun last_odd (lst)
2    (cond ((null lst) Nil)
3          ((oddp (car lst))
4            (let ((rec (last_odd (cdr lst))))
5              (if (null rec) (car lst) rec)
6            )
7          )
8          (t (last_odd (cdr lst)))
9    )
```

```
10 )
```

Примеры:

```
1 (last_odd '(1 2 3 4 5 6 7))
```

Результат: 7

8. **Используя cons-дополняемую рекурсию с одним тестом завершения, написать функцию которая получает как аргумент список чисел, а возвращает список квадратов этих чисел в том же порядке.**

   Пока список не Nil, объединяем все числа в списке, возведенные в квадрат.

   На вход список.

```
1 (defun square (lst)
2   (if lst
3     (cons
4       (* (car lst) (car lst))
5       (square (cdr lst))
6     )
7   )
8 )
```

Примеры:

```
1 (square '(1 -2 3 4 5))
```

Результат: (1 4 9 16 25)

9. **Написать функцию с именем select-odd, которая из заданного списка выбирает все нечетные числа. (Вариант 1: select-even, вариант 2: вычисляет сумму всех нечетных чисел(sum-all-odd) или сумму всех четных чисел (sum-all-even) из заданного списка.)**

   На вход список.

   Пока список не Nil (каждый раз функция применяется для хвоста списка), ищутся все нечетные числа и создается список.

```
1 (defun select_odd (lst)
2   (if lst
3     (append
4       (if (oddp (car lst))
5         (list (car lst)) Nil
6       )
7       (select_odd (cdr lst))
```

```
8       )
9     )
10 )
```

Примеры:

```
1 (select_odd '(1 2 3 4 5 6 7))
```

Результат: (1 3 5 7)

Пока список не Nil (каждый раз функция применяется для хвоста списка), ищутся все четные числа и создается список.

```
1  (defun select_even (lst)
2    (if lst
3      (append
4        (if (evenp (car lst))
5          (list (car lst)) Nil
6        )
7        (select_even (cdr lst))
8      )
9    )
10 )
```

Примеры:

```
1 (select_even '(1 2 3 4 5 6 7))
```

Результат: (2 4 6)

Пока список не Nil (каждый раз функция применяется для хвоста списка), ищутся все нечетные числа и считается их сумма.

```
1  (defun sum_all_odd (lst)
2    (cond ((null lst) 0)
3          ((oddp (car lst))
4            (+ (car lst) (sum_all_odd (cdr lst)))
5          )
6          (t (sum_all_odd (cdr lst)))
7    )
8  )
```

Примеры:

```
1 (sum_all_odd '(1 2 3 4 5 6 7))
```

Результат: 16

Пока список не Nil (каждый раз функция применяется для хвоста списка), ищутся все четные числа и считается их сумма.

```
(defun sum_all_even (lst)
  (cond ((null lst) 0)
        ((evenp (car lst))
          (+ (car lst) (sum_all_even (cdr lst))))
        )
        (t (sum_all_even (cdr lst)))
  )
)
```

Примеры:

```
(sum_all_even '(1 2 3 4 5 6 7))
```

Результат: 12

Теоретические вопросы:

1. Способы организации повторных вычислений в Lisp.

   Использование функционалов или рекурсии.

2. Что такое рекурсия? Классификация рекурсивных функций в Lisp.

   Рекурсия – это ссылка на определяемый объект во время его определения.

   (a) Простая рекурсия – один рекурсивный вызов в теле.

   (b) Рекурсия первого порядка – рекурсивный вызов встречается несколько раз.

   (c) Взаимная рекурсия – используется несколько функций, рекурсивно вызывающих друг друга.

3. Различные способы организации рекурсивных функций и порядок их реализации.

   (a) Хвостовая рекурсия – результат формируется не на выходе из рекурсии, а на входе в рекурсию, все действия выполняя до ухода на следующий шаг рекурсии.

   (b) Рекурсия по нескольким параметрам

   (c) Дополняемая рекурсия – при обращении к рекурсивной функции используется дополнительная функция не в аргументе вызова, а вне его

   (d) Множественная рекурсия – на одной ветке происходит сразу несколько рекурсивных вызовов.

4. Способы повышения эффективности реализации рекурсии.

   Использование хвостовой рекурсии. Если условий выхода несколько, то надо думать о порядке их следования. Некачественный выход из рекурсии может привести к переполнению памяти из-за "лишних"рекурсивных вызовов.

   Преобразование не хвостовой рекурсии в хвостовую, возможно путем использования дополнительных параметров. В этом случае необходимо использовать функцию-оболочку для запуска рекурсивной функции с начальными значениями дополнительных параметров.