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 № 7

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

Москва, 2020 г.

1. **Написать предикат, который принимает два числа-аргумента и возвращает T, если первое число не меньше второго.**

```
(defun eqb (a b)
  (if (> b a) Nil T)
)
```

2. **Какой из следующих двух вариантов предиката ошибочен и почему?**

```
(defun pred1 (x)
  (and (numberp x) (plusp x))
)

(defun pred2 (x)
  (and (plusp x) (numberp x))
)
```

Ошибочен второй, так как перед выполнением операции, не делается проверка на число.

3. **Переписать функцию how-alike, приведенную в лекции и не пользующую COND, используя конструкция IF, AND/OR.**

```
(defun how_alike (x y)
  (cond
    ((or (= x y) (equal x y)) 'the_same)
    ((and (oddp x) (oddp y)) 'both_odd)
    ((and (evenp x) (evenp y)) 'both_even)
    (T 'difference)
  )
)

(defun how_alike_if (x y)
  (if (or (= x y) (equal x y)) 'the_same
    (if (and (oddp x) (oddp y)) 'both_odd
      (if (and (evenp x) (evenp y)) 'both_even
        'difference
      )
    )
  )
)
```

Результат:

(how_alike 11 30) → DIFFERENCE

(how_alike_if 11 30) → DIFFERENCE

(how_alike 3 3) → THE_SAME

(how_alike_if 3 3) → THE_SAME

(how_alike 1 3) → BOTH_ODD

(how_alike_if 1 3) → BOTH_ODD

(how_alike -2 30) → BOTH_EVEN

(how_alike_if -2 30) → BOTH_EVEN

4. **Чем принципиально отличаются функции cons, list, append?**

   (cons lst1 lst2) → ((1 2 3) 4 5)

   (list lst1 lst2) → ((1 2 3) (4 5))

   (append lst1 lst2) → (1 2 3 4 5)

   CONS – позволяет создавать списки (возвращает бинарную ячейку (точечная пара, список), расставляя указатели, обязательно 2 аргумента).

   APPEND – функция двух аргументов x и y, сцепляющая два списка в один.

   LIST – создает столько списковых ячеек, сколько аргументов (всегда возвращает список).

5. **Каковы результаты вычисления следующих выражений?**

   (reverse ()) → Nil

   (last ()) → Nil

   (reverse '(a)) → (a)

   (last '(a)) → (a)

   (reverse '((a b c))) → ((a b c))

   (last '((a b c))) → ((a b c))

6. **Написать, по крайней мере, два варианта функции, которая возвращает последний элемент своего списка-аргумента.**

   Рекурсивно – пока второй элемент не Nil идем дальше, иначе возвращаем голову:

```
1 (defun last_elem (lst)
2   (if (NULL (cadr lst)) (car lst)
3     (last_elem (cdr lst))
4   )
5 )
```

С использованием функционала – с использованием функционала reduce, возвращая последний полученный результат, а возвращаем каждый раз второй аргумент:

```
(defun last_elem (lst)
  (reduce #'(lambda (a x) x) lst)
)
```

7. **Написать, по крайней мере, два варианта функции, которая возвращает свой список-аргумент без последнего элемента.**

   С использованием функционала – пользуясь тем, что mapcar будет работать до хвоста в самом коротком списке (cdr lst всегда на один короче lst), возвращаем значение из lst, таким образом получится список без последнего элемента:

```
(defun centr (lst)
  (mapcar (lambda (x y) y) (cdr lst) lst)
)
```

   Рекурсивно – пока хвост не Nil, соединяем с помощью cons голову и возвращаем значение функции от хвоста:

```
(defun centr (lst)
  (if (NULL (cadr lst)) ()
    (cons (car lst) (centr (cdr lst)))
  )
)
```

8. **Написать простой вариант игры в кости, в котором бросаются две правильные кости. Если сумма выпавших очков равна 7 или 11 – выигрыш, если выпало (1,1) или (6,6) — игрок право снова бросить кости, во всех остальных случаях ход переходит ко второму игроку, но запоминается сумма выпавших очков. Если второй игрок не выигрывает абсолютно, то выигрывает тот игрок, у которого больше очков. Результат игры и значения выпавших костей выводить на экран с помощью функции print.**

   Функция game запускает игру, в которой анализируются 2 рандомных числа (от 1 до 6) для 2 игроков. Далее сравниваются их суммы, в зависимости от этого возвращается результат.

   Функция analyze принимает на вход 2 числа, и в зависимости от них, либо перекидывает кости (функция вызывается еще раз с новыми параметрами) либо возвращает число (сумму очков).

   Функция printS выводит очки и их сумму на экран.

```
1  (defun printS (x y s)
2    (print (list x '+ y '= s))
3  )
4
5  (defun analyze (x y)
6    (cond
7      ((and (= 1 x) (= 1 y)) (printS x y (+ x y))
8          (analyze (+ 1 (random 6)) (+ 1 (random 6))))
9      ((and (= 6 x) (= 6 y)) (printS x y (+ x y))
10         (analyze (+ 1 (random 6)) (+ 1 (random 6))))
11     (T (printS x y (+ x y)) (+ x y))
12   )
13 )
14
15 (defun game ()
16   (let ((s1 (analyze (+ 1 (random 6)) (+ 1 (random 6))))
17         (s2 (analyze (+ 1 (random 6)) (+ 1 (random 6)))))
18     (cond
19       ((or (= s1 7) (= s1 11) (> s1 s2)) 'you-win)
20       ((or (= s1 7) (= s2 11) (< s1 s2)) 'he-wins)
21       (T 'draw)
22     )
23   )
24 )
```

Примеры работы:

```
(1 + 5 = 6)
(6 + 6 = 12)
(4 + 6 = 10) HE-WINS%
```

```
(3 + 2 = 5)
(3 + 2 = 5) DRAW%
```

```
(2 + 2 = 4)
(5 + 6 = 11) HE-WINS%
```

```
(3 + 3 = 6)
(5 + 4 = 9) HE-WINS%
```

```
(4 + 4 = 8)
(5 + 1 = 6) YOU-WIN%
```