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

ФАКУЛЬТЕТ «Информатика и системы управления»_____

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт
к лабораторной работе № 6
По курсу: «Функциональное и логическое программирование»
Тема: «Работа интерпретатора Lisp.»

Студент Прохорова Л. А.

Группа ИУ7-63Б

Оценка (баллы) _____

Преподаватели Толпинская Н. Б., Строганов Ю. В.

Москва.
2021 г.

Цель работы: приобрести навыки работы с управляющими структурами Lisp.
Задачи работы: изучить работу функций с произвольным количеством аргументов, функций разрушающих и неразрушающих структуру исходных аргументов.

Задание 1
Чем принципиально отличаются функции cons, list, append?
Пусть (setf lst1 '(a b)) (setf lst2 '(c d)). Каковы результаты вычисления следующих выражений?

1. (cons lst1 lst2)
2. (list lst1 lst2)
3. (append lst1 lst2)

Функция **cons** создает точечную пару (устанавливает указатель car на первый полученный аргумент, cdr- на второй), принимает фиксированное число аргументов, равное двум. Является базисом.

Функция **list** создает список из своих аргументов. Принимает нефиксированное число аргументов. Написана на основе cons. Входит в ядро.

Функция **append** объединяет списки. Является формой. Создаёт копию всех аргументов, кроме последнего. Не разрушает структуру. Так как последний аргумент не копируется, то при модификации результата может поменяться список который был задан последним.

(cons lst1 lst2) -> ((a b) c d)
(list lst1 lst2) -> ((a b) (c d))
(append lst1 lst2) -> (a b c d)

Задание 2
Каковы результаты вычисления следующих выражений?

(reverse ()) -> Nil
(last ())-> Nil
(reverse '(a)) -> (a)
(last '(a)) -> (a)
(reverse '((a b c))) -> ((a b c))
Так как функции применяются только к верхнему уровню списковых ячеек.
(last '((a b c))) -> ((a b c))

Задание 3
Написать, по крайней мере, два варианта функции, которая возвращает последний элемент своего списка-аргумента.

```
(defun get_last1 (lst) (car (reverse lst)))
```

```
 (defun get_last2(lst)
(cond ((equal (cdr lst) nil)(car lst))
(t (get_last2 (cdr lst)))))
```

Задание 4
Написать, по крайней мере, два варианта функции, которая возвращает свой список-аргумент без последнего элемента.

```
(defun get_without_last1 (lst) (reverse (cdr (reverse lst))))

(defun get_without_last2(lst)
(cons (car lst)
(if (null (cdr (cdr lst))) Nil (get_without_last1 (cdr lst)))))
```

Задание 5
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.

(random n) *принимает положительное число n и возвращает число такого же типа между нулём (включительно) и n (не включительно). Число n может быть целым или с плавающей точкой.*

```
; одна кость - рандомное число
(defun dice-score() (+ (random 6) 1))

; бросок кости - сумма выпавших очков
(defun roll-dices(num)
   (let* (
         (a (dice-score))
         (b (dice-score))
      )
      (car (last
         (list
            (print (format t "player ~d dice-> ~d : ~d" num a b))
            (+ a b))
         )
      )
   )
)
```

```
; (1, 1) - сумма == 2, другими способами 2 не получить
; (6, 6) - сумма == 12, другими способами 12 не получить

; если сумма выпавших очков равна 7 или 11 выигрыш,
(defun is-win(sum) (if (or (= sum 7) (= sum 11)) t Nil) )



; если выпало (1,1) или (6,6) игрок получает право снова бросить кости,
(defun is-repeat(sum) (if (or (= sum 2) (= sum 12)) t Nil))

; 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
(defun play(player-1 player-2)
   (let* (
         (player-1 (if (null player-1) (roll-dices 1) player-1))
      )
      (cond
         ((is-win player-1) (print (format t "player 1 wins: ~d" player-1)))
         ((is-repeat player-1) (play Nil player-2))
         (t
            (let* (
                  (player-2 (if (null player-2) (roll-dices 2) player-2))
               )
               (cond
                  ((is-win player-2) (print (format t "player 2 wins: ~d " player-2)))
                  ((is-repeat player-2) (play player-1 Nil))
                  (t

                     (cond ((> player-2 player-1) (print (format t "player 2 wins: ~d > ~d"
player-2 player-1)))
                           ((= player-2 player-1) (print "tie"))
                           (t (print (format t "player 1 wins: ~d > ~d" player-1 player-2)))
                     )
                  )
               )
            )
         )
      )
   )
)

(defun run-play() (play Nil Nil))
```

Теоретические вопросы:
1)Структуроразрушающие и не разрушающие структуру списка функции.

Не разрушающие структуру списка функции в lisp создают копии всех аргументов – списковых ячеек, но не значений по car-указателям – кроме, разве что, последнего (например, в append) и расставляют cdr-указатели в новых ячейках. Копия последнего аргумента не создается для оптимизации по времени. После вызова не разрушающей структуру функции, представление ее аргументов в памяти не изменяется – следовательно, с ними можно работать в исходном виде. Примеры: append, conc, reverse.
Структуроразрушающие функции не создают копий аргументов, только переставляют cdr-указатели исходных списковых ячеек. Имена структуроразрушающих функций начинаются с буквы n. Примеры: nconc, nreverse.

2)Отличие в работе функций cons, list, append и их результате.

Cons:
• принимает ровно два аргумента – s-выражений
• создает одну списковую ячейку и расставляет указатели на первый и второй аргументы соответственно
• результат: одна точечная пара

List:
• принимает переменное число аргументов – s-выражений
• создает список, т.е. столько же списковых ячеек, сколько передано аргументов, расставляет car-указатели на переданные значения аргументов
• результат: список

Append:
• принимает переменное число аргументов – только списков
• создает копии всех аргументов, кроме последнего – только списковых ячеек, не значений по car указателям – расставляет car указатели копий ячеек на соответствующие значения в памяти и устанавливает cdr-указатели последних ячеек каждого из аргументов, кроме последнего, на голову следующего аргумента.
• результат: список, являющийся результатом последовательного соединения аргументов друг с другом. При этом результирующий список составляют копии аргументов, а не исходные списки в памяти. Так как последний аргумент не копируется, то при модификации результата может поменяться список который был задан последним.