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

О т ч е т

по лабораторной работе № 7

Дисциплина: «Функциональное и логическое программирование»

Выполнила: Овчинникова А.П.

Группа: ИУ7-65Б

Преподаватель: Толпинская Н.Б.

Москва, 2020

Теоретическая часть.

Специальный оператор *(setq {symbol value}*)* присваивает каждой переменной *symbol* значение соответствующего выражения *value*. Если одно из выражения ссылается на ранее определенную переменную, оно будет использовать новое значение. Возвращает значение последнего выражения *value*.

Макрос *(setf {place value}*)* является обобщением *setq*. *setf* помещает значение выражения *value* по заданному месту. Если выражение *value* ссылается на одно из предыдущих мест *places*, *setf* будет использовать новое значение этого места. Возвращает значение последнего выражения *value*.

Корректным выражением для места *place* может быть: переменная; вызов любой «устанавливаемой» функции при условии, что соответствующий аргумент является корректным выражением для *place*; вызов *apply* с первым аргументом из числа следующих: *#'aref, #'bit* или *#'sbit*; вызов функции доступа к полям структуры; выражение *the* или *values*, аргумент(ы) которого являются корректными местами *places*; вызов оператора, для которого задано *setf*-раскрытие; или макрос, раскрывающийся в что-либо из вышеперечисленного.

Функция *(append lst1 ... lstN)* выполняет объединение своих списков-аргументов. Не разрушает структуру (создает копии всех своих аргументов кроме последнего и устанавливает указатели в копиях).

Функция *(reverse proseq)* изменяет порядок элементов в аргументе. Возвращает последовательность того же типа, что и *proseq*, содержащую те же элементы в обратном порядке. Последовательность, возвращаемая *reverse*, всегда является копией.

Функция *(last list)* возвращает последнюю ячейку в *list*.

Макрос (*loop форма1 форма2 ...)* реализует бесконечный цикл, в котором формы вычисляются до тех пор, пока не встретится явный оператор завершения *return*.

Практическая часть.

Задание 1.

Чем принципиально отличаются функции cons, list и append?

Функция *(cons object1 object2)* возвращает новую списковую ячейку, car которой – *object1*, а cdr – *object2*.

Форма *(list objects)* возвращает новый список, состоящий из объектов *objects*.

Функция *(append lst1 .. lstN)* выполняет объединение своих списков-аргументов. Не разрушает структуру (создает копии всех своих аргументов кроме последнего и устанавливает указатели в копиях).

Таким образом, *cons* создает одну списковую ячейку и ставит в ней указатель на свой второй аргумент. *list* создает столько списковых ячеек, сколько ему было передано аргументов, и расставляет в них указатели. *append* создает копии всех своих аргументов, кроме последнего, после чего расставляет указатели между ними в порядке следования в списке аргументов.

(setf lst1 '(a b))

(setf lst2 '(c d))

(cons lst1 lst2) = ((A B) C D)

(list lst1 lst2) = ((A B) (C D))

(append lst1 lst2) = (A B C D)

Задание 2.

(reverse ()) = Nil

(last ()) = Nil

(reverse '(a)) = (A)

(last '(a)) = (A)

(reverse '((a b c))) = ((A B C))

(last '((a b c))) = ((A B C))

Задание 3.

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

```
(defun my_last (l)
   (car (reverse l))
)
(defun my_last2 (l)
   (nth (if (< (list-length l) 0) 0 (- (list-length l) 1)) l)
)

(my_last '(1 2 3))
(my_last '(1))
(my_last Nil)
```

Задание 4.

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

```
(defun my_tail (l)
   (reverse (cdr (reverse l)))
)
(defun my_tail2 (l)
   (nthcdr 1 (reverse l))
)
(my_tail '(1 2 3))
(my_tail '(1))
(my_tail Nil)
```

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 получает 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 и значения выпавших костей выводить на экран с помощью функции print.

```
(defun second_player (t1)
  (print "Ходит второй игрок)"
  (loop
    (setf c1 (+ (random 6) 1))
    (setf c2 (+ (random 6) 1))
    (print `(Очки ,c1 ,c2))
    (if (or (= (+ c1 c2) 7) (= (+ c1 c2) 11))
      (return `(Второй игрок выиграл со счетом ,(+ c1 c2) ))
      (if (or (and (= c1 1) (= c2 1) ) (and (= c1 6) (= c2 6)))
        (print "Второй игрок перебрасывает")
        (return
          (and (cond ((> t1 (+ c1 c2)) `(Первый игрок выиграл со счетом ,t1))
                     ((< t1 (+ c1 c2)) `(Второй игрок выиграл со счетом ,(+ c1 c2)))
                     ((= t1 (+ c1 c2)) `(Ничья со счетом `t1))
               )
          )
        )
      )
    )
  )

(defun my_game ()
  (print "Ходит первый игрок")
  (loop
    (setf c1 (+ (random 6) 1))
```

```
        (setf c2 (+ (random 6) 1))
        (print `(Очки ,c1 ,c2))
        (if (or (= (+ c1 c2) 7) (= (+ c1 c2) 11))
            (and (print `(Первый игрок выиграл со счетом ,(+ c1 c2) )) (return))
            (if (or (and (= c1 1) (= c2 1) ) (and (= c1 6) (= c2 6)))
                (print "Первый игрок перебрасывает")
                (and (print (second_player (+ c1 c2))) (return)
                )
            )
        )
    )
)
```

Второй вариант:

```
(defun random_number ()
    (list (+ (random 6) 1) (+ (random 6) 1))
)
(defun print_info (n player)
    (terpri)
    (princ  "Игрок №")
    (princ n)
    (print player)
    (terpri)
)
(defun out (lst)
    (cond ((equal lst '(6 6)) T)
          ((equal lst '(1 1)) T)
    )
)
```

```
(defun play (n)
   (setf player (random_number))
   (print_info n player)
   (do
      ()
      ((not (out player)))
      (setf player (random_number))
      (print_info n player)
   )
   player
)
(defun abs_winner (player)
   (cond ((= (+ (car player) (car (last player))) 11) T)
         ((= (+ (car player) (car (last player))) 7) T)
   )
)
(defun total_score (player)
   (+ (car player) (car (last player)))
)
(defun my_game2 ()
   (print '(Ходит первый игрок))
   (setf player1 (play 1))

   (cond
      ( (abs_winner player1) '(Первый игрок выиграл абсолютно) )
      ( (print '(Ходит второй игрок))
        (setf player2 (play 2))
        (cond ( (abs_winner player2) '(Второй игрок выиграл абсолютно))
              ( ( cond
                    ( (= (total_score player1) (total_score player2)) '(Ничья))
```

```
                    ( (> (total_score player1) (total_score player2)) '(Выиграл первый игрок))
                    ( (< (total_score player1) (total_score player2)) '(Выиграл второй игрок))
                ) )
            )
          )
        )
      )

      (print (my_game2))
```