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»

Лабораторная работа №7

Студент: Луговой Д.М.
Группа: ИУ7-61Б
Преподаватель: Толпинская Н.Б.

Москва, 2020 г.

Цель работы: приобрести навыки работы в управляющими структурами Lisp.

Задачи работы: изучить работу функций с произвольным количеством аргументов, функций, разрушающих и не разрушающих структуру структуру исходных аргументов.

Задание №1

Чем принципиально отличаются функции CONS, LIST, APPEND?

CONS является базовой функцией Lisp, LIST и APPEND реализованы через CONS. CONS является чистой математической функцией, принимающей 2 параметра. Она создает списочную ячейку, CAR-указатель которой указывает на 1-й аргумент, а CDR-указатель - на 2-й аргумент.

LIST является формой, так как принимает произвольное число аргументов, возвращая список, состоящий из переданных ему аргументов.

APPEND является формой, так как принимает произвольное число аргументов, которые должны быть списками, кроме последнего. Она возвращает точечную пару, CAR-указатель которой указывает на конкатенацию всех переданных ей аргументов, кроме последнего, а CDR-указатель - на последний аргумент. При этом структура исходных аргументов не разрушается, для этого происходит копирование всех аргументов, кроме последнего.

Пусть

(setf lst1 '(a b)) (setf lst2 '(c d))

Каковы результаты вычисления следующих выражений?

- (cons lst1 lst2)
  Результат: ((A B) C D)
- (list lst1 lst2)
  Результат: ((A B) (C D))
- (append lst1 lst2)
  Результат: (A B C D)

Задание №2

Каковы результаты вычисления следующих выражений?

- (reverse ())
  Результат: NIL
- (last ())
  Результат: NIL
- (reverse '(a))
  Результат: (A)
- (last '(a))
  Результат: (A)
- (reverse '((a b c)))
  Результат: ((A B C))
- (last '((a b c)))
  Результат: ((A B C))

Задание №3

Написать, по крайней мере, 2 варианта функции, которая возвращает последний элемент своего списка-аргумента.

Листинг 1: 1-й вариант функции получения последнего элемента списка

```
(defun get_last (lst) (car (last lst)))
```

Листинг 2: 2-й вариант функции получения последнего элемента списка

```
(defun get_last (lst) (car (reverse lst)))
```

Листинг 3: 3-й вариант функции получения последнего элемента списка

```
(defun get_last (lst) (cond
  (
    (cdr lst)
    (get_last (cdr lst))
  )
  (
    (car lst)
  )
))
```

Листинг 4: 4-й вариант функции получения последнего элемента списка

```
(defun get_last (lst) (mapcon
  #'(lambda (lst) (cond
    (
      (cdr lst)
      nil
    )
    (
      (car lst)
    )
  ))
  lst
))
```

Пример:

```
> (get_last '(1 2 3 4))
4
> (get_last '((1 2) 3 4 (5 (6))))
(5 (6))
> (get_last ())
NIL
```

Задание №4

Написать, по крайней мере, 2 варианта функции, которая возвращает свой список-аргумент без последнего элемента.

Листинг 5: 1-й вариант функции получения списка без последнего элемента

```
(defun remove_last (lst) (reverse (cdr (reverse lst))))
```

Листинг 6: 2-й вариант функции получения списка без последнего элемента

```
(defun remove_last (lst) (cond
  (
    (cdr lst)
    (cons
      (car lst)
      (remove_last (cdr lst))
    )
  )
))
```

Листинг 7: 3-й вариант функции получения списка без последнего элемента

```
(defun remove_last (lst) (mapcon
 #'(lambda (lst) (cond
   (
     (cdr lst)
     (cons (car lst) nil)
   )
 ))
 lst
))
```

Пример:

```
> (remove_last '(1 2 3 4))
(1 2 3)
> (remove_last '((1 2) 3 4 (5 (6))))
((1 2) 3 4)
> (remove_last ())
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 игры и значения выпавших костей выводить на экран с помощью функции print.

Листинг 8: Функция подбрасывания костей

```
(defun roll_dice () (cons
  (+ (random 6) 1)
  (+ (random 6) 1)
))
```

Листинг 9: Функция проверки переподбрасывания костей

```
(defun check_reroll (pair) (or
  (= (car pair) (cdr pair) 1)
  (= (car pair) (cdr pair) 6)
))
```

Листинг 10: Функция суммирования чисел в точечной паре

```
(defun sum_pair (pair) (+ (car pair) (cdr pair)))
```

Листинг 11: Функция проверки моментального выигрыша

```
(defun check_win (pair) (or
  (= (sum_pair pair) 7)
  (= (sum_pair pair) 11)
))
```

Листинг 12: Функция получения конечного результата подбрасывания

```
(defun get_player_result () (let ((pair (toss_dice)))
  (cond
    (
      (check_reroll pair)
      (format T "~%Got ~A, rerolling: " pair) (get_player_result)
    )
    (pair)
  )
))
```

Листинг 13: Функция игры в кости

```
(defun play_dice () (princ "Player 1: ")
  (let ((player1 (get_player_result)))
    (princ player1)
    (terpri)
    (cond
      (
        (check_win player1)
        (princ "Player 1 wins")
      )
      (
        (princ "Player 2: ")
        (let ((player2 (get_player_result)))
          (princ player2)
          (terpri)
          (cond
            (
              (check_win player2)
              (princ "Player 2 wins")

            )
            (
              (cond
                (
                  (= (sum_pair player1) (sum_pair player2))
                  (princ "Draw")
                )
                (
                  (> (sum_pair player1) (sum_pair player2))
                  (princ "Player 1 wins")
                )
                (
                  (princ "Player 2 wins")
                )
              )
            )
          )
        )
      )
    )
  )
)
```

Пример:

```
> (play_dice)
Player 1: (5 . 1)
Player 2: (5 . 3)
Player 2 wins
> (play_dice)
Player 1: (6 . 4)
Player 2: (2 . 5)
Player 2 wins
> (play_dice)
Player 1: (4 . 1)
Player 2: (3 . 2)
Draw
> (play_dice)
Player 1: (3 . 4)
Player 1 wins
> (play_dice)
Player 1:
Got (1 . 1), rerolling: (4 . 6)
Player 2: (2 . 3)
Player 1 wins
```