Московский государственный технический университет
имени Н. Э. Баумана

Факультет: Информатика и системы управления
Кафедра: Программное обеспечение ЭВМ и информационные технологии
Дисциплина: Функциональное и логическое программирование

Лабораторная работа №4

Выполнили: Никичкин А. С., Фокеев А. И.
Группа: ИУ7–61

Москва, 2015 г.

1 Чем принципиально отличаются функции CONS, LIST, APPEND

Пусть

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

Тогда при использовании CONS

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

Таким образом CONS создаёт *списочную ячейку*, у которой *car-указатель* указывает на первый аргумент, а *cdr-указатель* — на второй. При использовании LIST

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

Получаем, что LIST гарантирует возвращение нового *списка* из элементов, переданных функции в качестве аргументов. При использовании APPEND

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

APPEND возвращает новый список, который формируется из объединения списков переданных функции в качестве аргументов. При этом последний аргумент может являться не списком. CDR указатель нового списка будет указывать как раз на последний аргумент.

2 Вычислить результат выражения

Задание 2.1

```
(reverse ()) => NIL
```

Задание 2.2

```
(last ()) => NIL
```

Задание 2.3

```
(reverse '(a)) => (A)
```

Задание 2.4

```
(last '(a)) =>
```

Задание 2.5

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

Задание 2.6

```
(last '((A B C))) => ((A B C))
```

3 Написать функцию

Задание 3.1 Функции, которые возвращают последний элемент своего списка-аргумента

```
1  (defun problem-2-1 (arg)
2      (first (last arg)))
3
4  (defun problem-2-1 (arg)
5      (car (last arg)))
6
7  (defun problem-2-2 (arg)
8      (first (reverse arg)))
9
10 (defun problem-2-3 (arg)
11     (car (reverse arg)))
```

Задание 3.2 Функции, которые возвращают свой список-аргумент без последнего элемента

```
1  (defun problem-3-2-1 (arg)
2      (reverse (cdr (reverse arg))))
3
4  (defun problem-3-2-2 (arg)
5      (reverse (rest (reverse arg))))
6
7  (defun problem-3-2-3 (arg)
8      (butlast arg))
```

4 Написать программу

Задание 4.1 Простой вариант игры в кости, в которой 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.

```
(defun throw-die (&optional (n-sides 6))
  "Throw die with n sides"
  (+ 1 (random n-sides)))

(defun throw-dice2 (&optional (n-sides 6))
  "Throw two dice with n sides"
  (cons (throw-die n-sides)
        (throw-die n-sides)))

(defun sum-dice2 (dice)
  "Sum of two dice"
  (+ (first dice)
     (rest dice)))

(defun is-lucky (dice)
  "Lucky combination"
  (or (equal dice '(1 . 1))
      (equal dice '(6 . 6))))

(defun is-winner (dice)
  "Winner combination"
  (or (eql 7 (sum-dice2 dice))
      (eql 11 (sum-dice2 dice))))

(defun play (name &optional all-turns (score 0))
  (let ((turn (throw-dice2)))
  (format T "~:s: ~a~%" name turn)
  (cond ((is-lucky turn)
         (play name
               `(,all-turns ,turn)
               (+ score (sum-dice2 turn))))
        ((is-winner turn)
         (format T "~:s is winner!~%" name) T)
        (T (+ (sum-dice2 turn) score)))))

(defun game (&optional (player-name1 'Player1)
                       (player-name2 'Player2))
  (let ((score1 (play player-name1)))
    (if (numberp score1)
      (let ((score2 (play player-name2)))
        (if (numberp score2)
          (cond ((> score1 score2)
                 (format T "~:s is winner!~%" player-name1))
                ((> score2 score1)
                 (format T "~:s is winner!~%" player-name2))
                (T
                 (format T "Drawn Game~%"))))))
    "GG WP"))
```