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 по лабораторной работе №6 по курсу «Функциональное и логическое программирование»

Тема: Использование управляющих структур, работа со списками

Студент: Сорокин А. П. ИУ7-66Б

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

2021 г.

1. Результаты выражений с cons, list, append.

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

Выражение	Результат
`(cons lst1 lst2)`	`((a b) c d)`
`(list lst1 lst2)`	`((a b) (c d))`
`(append lst1 lst2)`	`(a b c d)`

2. Результаты выражений.

Выражение	Результат
`(reverse ())`	`NIL`
`(last ())`	`NIL`
`(reverse '(a))`	`(A)`
`(last '(a))`	`(A)`
`(reverse '((a b c)))`	`((A B C))`
`(last '((a b c)))`	`((A B C))`

3. Функция, которая возвращает последний элемент своего списка-аргумента.

```
(defun my-last (lst)
    (if (null (cdr lst))
        (car lst)
        (my-last (cdr lst))
    )
)

(defun my-last-2 (lst)
    (car (reverse lst))
)
```

4. Функция, которая возвращает свой список-аргумент без последнего элемента.

```
(defun no-last (lst)
    (if (null (cdr lst))
        Nil
        (cons (car lst) (no-last (cdr lst)))
    )
)

(defun no-last-2 (lst)
    (reverse (cdr (reverse lst)))
)
```

5. Простой вариант игры в кости

```
; Функция, "бросающая" случайный образом кости
(defun roll-dice ()
    (print (cons (+ (random 6) 1) (+ (random 6) 1)))
)
; Функция, складывающая числа на костях
(defun sum-dices (dice) (+ (car dice) (cdr dice)))

; Функция, определяющая абсолютную победу по выпавшим костям
(defun is-win (dice)
    (or
        (= 7 (setq dsum (sum-dices dice)))
        (= 11 dsum)
    )
)
; Функция, определяющая, может ли игрок кинуть заново
(defun is-retry (dice)
    (or (equal '(1.1) dice)
        (equal '(6.6) dice)
    )
)
; Функция хода
(defun play-turn (no)
    (setq dice (roll-dice))
    (if (is-win dice)
        (format NIL "P~A won!" no)
        (if (is-retry dice)
            (play-turn no)
            (sum-dices dice)
        )
    )
)
; Функция игры
(defun play()
    (if (numberp (setq p1 (play-turn 1)))
        (if (numberp (setq p2 (play-turn 2)))
            (cond
                ((> p1 p2) "P1 won!")
                ((> p2 p1) "P2 won!")
                (T "Tie!")
            )
            p2
        )
        p1
    )
)
```

Ответы на вопросы

1. Разрушающие и неразрушающие структуру списка функции.

Разрушающие структуру функции не сохраняют возможность работа со старыми структурами, т. к. эти функции изменяют их.

Неразрушающие структуру списков функции сохраняют такую возможность. Если требуется вернуть модифицированный вариант списка-аргумента, то возвращается его изменённая копия.

Неразрушающие	Разрушающие
append	nconc
reverse	nreverse
last	rplaca
nth	rplacd
nthcdr	
length	
remove	delete
subst	nsubst

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

Функция cons – базисная, чистая математическая функция, принимающая ровно два аргумента. Создаёт бинарный узел, расставляя его указатели на два аргумента.

Функция list является формой. Создаёт ровно столько списковых ячеек, сколько передано аргументов. каждый car-указатель ссылается на соответственный аргумент, а cdr-указатель одной ячейки указывает на следующую ячейку.

Функция append является формой, неразрушающей структуру списка: она создаёт новый список, в котором создаются новые списковые ячейки для каждого списка-аргумента, кроме последнего. Cdr-указатель последней списковой ячейки копии одного списка-аргумента указывает на первую списковую ячейку копии следующего списка-аргумента. Cdr-указатель предпоследнего списка будет указывать на первую ячейку самого последнего списка-аргумента, а не его копии.

```
(cons ‘(a b) ‘(c d)) -> ((a b) c d)

(list ‘(a b) ‘(c d)) -> ((a b) (c d))

(append ‘(a b) ‘(c d)) -> (a b c d)
```