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»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

Название: Использование управляющих структур, работа со списками

Дисциплина: Функциональное и логическое программирование

Студент	ИУ7-66Б		Т. А. Казаева
	Группа	Подпись, дата	И. О. Фамилия
Преподаватель			Н. Б. Толпинская
		Подпись, дата	И. О. Фамилия

Москва, 2022 г.

1. Теоретические вопросы

1. *Синтаксическая форма и хранение программы в памяти.*
Единообразие представления данных в памяти дает возможность воспринимать одну и ту же информацию как данные и как программу одновременно (благодаря определенным манипуляциям). По-умолчанию считается, что любое S-выражение, попадающее на вход интерпретатора, является программой. Обработка выражения в lisp – это получение доступа к какой-либо информации по указателю.

2. *Трактовка элементов списка.*
Первый элемент в списке трактуется как функция, остальные – как ее аргументы.

3. *Порядок реализации программы.*
При вызове функции указываются лексические переменные – символьные атомы. Для них выделяется память (5 указателей) и в процессе работы тела функции указатели могут переставляться или может происходить смена значений по указателю. На каждом очередном шаге использования лексического параметра система заново вычисляет значение по этого символьного атома. При завершении работы функции система считает, что место освобождено и значения теряются в памяти.

4. *Способы определения функции.*
Обычно функции определяются при помощи макроса DEFUN. В качестве имени может использоваться любой символ. Как правило, имена функций содержат только буквы, цифры и знак минус. Список параметров функции определяет переменные, которые будут использоваться для хранения аргументов, переданных при вызове функции. Тело DEFUN состоит из любого числа выражений Lisp.

2. Практические задания

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)

(defun test(lst) (loop for a from 0 to (- (length lst) 3) do (cdr lst))) Решения:

1) **cons** объединяет значения своих аргументов в точечную пару. Если вторым аргументом будет передан список, то в результате получится список, в котором второй аргумент будет добавлен в начало: ((A B)C D)

2) **list** составляет из своих аргументов список: ((A B)(C D))

3) **append** создает копию всех аргументов, кроме последнего, т. е списковые ячейки. Связываются последними указателями. Результирующее значение: (A B C D)

2. *Каковы результаты вычисления следующих выражений, и почему?*

1) (**reverse** ())

2) (**last** ())

3) (**reverse** '(a))

4) (**last** '(a))

5) (**reverse** '((a b c)))

6) (**last** '((a b c)))

Решения:

1) NIL

2) NIL

3) (A)

4) (A)

5) ((a b c))

6) ((a b c))

3. *Написать, по крайней мере, два варианта функции, которая возвращает последний элемент своего списка-аргумента.* Повторные вычисления:

```
(defun last-repeat(lst)
  (loop for x in lst
    finally (return x)))
```

Рекурсия:

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

Reverse:

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

4. *Написать, по крайней мере, два варианта функции, которая возвращает свой список-аргумент без последнего элемента.* Повторные вычисления:

```
(defun but-last-repeat(lst)
  (loop for x from 0 to (- (length lst) 2)
    collect (nth x lst) into buf
    finally (return buf)))
```

Рекурсия:

```
(defun but-last-recurs (lst)
  (if (null (cdr lst))
    nil
    (cons (car lst) (but-last-recurs (cdr lst)))))
```

Reverse:

```
(defun but-last-reverse (lst)
   (reverse(cdr (reverse lst))))
```

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.*

```
(setf *random-state* (make-random-state t))

(defun dices-roll (edgen)
  (let* (
    (fdice (+ (random edgen) 1))
    (sdice (+ (random edgen) 1))
    (sum (+ fdice sdice)))
  (if (or (and (eq fdice 1) (eq sdice 1)) (and (eq fdice 6) (eq sdice 6)))
      (progn (format t "~%~a ~a - reroll chance! player rolls again... " fdice
          sdice) (setq sum (dices-roll edgen)))
      (progn (format t "~a ~a~%" fdice sdice) sum))))

(defun early-winp(score)
    (or (eq score 7) (eq score 11)))

(defun final-score (fp sp)
  (cond ((early-winp sp) (format t "player 2 scored ~a and won!~%" sp))
    ((> fp sp) (format t "player 1 won!~%"))
    ((> sp fp) (format t "player 2 won!~%"))
    (t (write-line "it's a draw!"))))

(defun play()
  (let (
    (pl (progn (format t "player 1 rolls dices...") (dices-roll 6))))
  (if (early-winp pl)
    (format t "player 1 scored ~a and won!~%" pl)
    (final-score pl (progn (format t "player 2 rolls dices... ") (dices-roll
        6))))))

(play) ;; driver code
```