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 №4

Название: Использование управляющих структур, работа со списками.

Дисциплина: Функциональное и логическое программирование

Студент	ИУ7-64Б		Л.Е.Тартыков
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			Н.Б.Толпинская
		(Подпись, дата)	(И.О. Фамилия)
Преподаватель			Ю.В.Строганов
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2022

1 Теоретические вопросы

1.1 Синтаксическая форма и хранение программы в памяти

В Lisp программы и данные синтаксически выглядят в виде *S-выражения*. S-выражение может быть как атомом, который в памяти представляется в виде 5 указателей (name, value, function, properties, package), так и точечной парой – 2 указателя (бинарный узел).

1.2 Трактовка элементов списка

Элементы списка представляются следующим образом: первый - имя функции, остальные – её аргументы. Формат списка представлен в виде формулы (1.1).

$$(\text{функция аргумент\_1} \ldots \text{аргумент\_n}), n \geq 0 \tag{1.1}$$

1.3 Порядок реализации программы

Набранные S-выражения выполняются при помощи интерпретатора – функцией eval, после выполнения которой возвращается полученный результат.

1.4 Способы определения функции

1. lambda-выражение. Данный способ представлен в виде формулы (1.2).

$$(\textit{lambda}\ \lambda\text{-список форма}), \tag{1.2}$$

где λ-список – список формальных параметров, форма – тело функции.

lambda-выражение не хранится в памяти и не имеет имени. Вычисляется сразу же. Используется для повторных вычислений.

Вызов lambda-функции выполняется по формуле (1.3).

$$(\lambda\text{-выражение последовательность форм}) \tag{1.3}$$

2. С помощью *defun* по формуле (1.4).

$$(defun\ f\ \lambda\text{-выражение}) \tag{1.4}$$

Система по имени символьного атома находит его определение.

1.5 Принципиальное отличие функций cons, list, append

cons – имеет два аргумента и возвращает бинарный узел. Если вторым аргументом является атом, то возвращается точечная пара; если список – список.

list – имеет произвольное число аргументов и возвращает список.

append – имеет произвольное число аргументов; важным свойством является то, что создается копия всех аргументов, кроме последнего. В результате функции возвращается список.

Листинг 1.1 – Использование cons.

```
1  (cons 'A 'B)            ;; (A.B)
2  (cons 'A '(B C D))      ;; (A B C D)
3  (cons '(A B) '(C D))    ;; ((A B) C D)
```

Пример использования list представлен на листинге 1.2.

Листинг 1.2 – Использование list.

```
1  (list 'A 'B)            ;; (A B)
2  (list 'A '(B C) 'D)     ;; (A (B C) D)
```

Функция list может быть представлена с помощью cons (листинг 1.3).

Листинг 1.3 – Представление list с помощью cons.

```
(defun list2 (ar1 ar2) (cons ar1 (cons ar2 ())))
(defun list3 (ar1 ar2 ar3) (cons ar1 (cons ar2 (cons ar3 ()))))
...
```

Отличие в использовании cons, list, append представлен на листинге 1.4.

Листинг 1.4 – Использование cons, list, append.

```
(list '(a b) '(c d))     ;((a b) (c d))
(cons '(a b) '(c d))     ;((a b) c d)
(append '(a b) '(c d))   ;(a b c d)
```

2 Практические задания

2.1 Каковы результаты вычисления следующих выражений?

```
(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)
```

2.2 Каковы результаты вычисления следующих выражений?

```
(reverse ())           ; nil
(last ())              ; nil
(reverse '(a))         ; (a)
(last '(a))            ; (a)
(reverse '((a b c)))   ; ((a b c))
(last '((a b c)))      ; ((a b c))
```

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

```
(defun last_elem_1 (arg) (car (last arg)))

(defun last_elem_2 (arg) (first (nreverse arg)))
```

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

```
(defun without_last_1 (arg) (nreverse (cdr (nreverse arg))))

(defun without_last_2 (arg)
    (reverse (last (nreverse arg) (- (length arg) 1))))
```

2.5 Игра в кости

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.

Листинг 2.1 – Программный код «Игра в кости»

```
(defvar first_gamer nil)
(defvar second_gamer nil)

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

(defun is_winner (first_gamer_numbers)
    (cond ((= (+ (car first_gamer_numbers) (cadr first_gamer_numbers)) 7) T)
          ((= (+ (car first_gamer_numbers) (cadr first_gamer_numbers)) 11) T)
    )
)

(defun print_gamer(gamer value_gamer)
    (print gamer)
    (print (car value_gamer))
)
```

Листинг 2.2 – Программный код «Игра в кости» (продолжение)

```
(defun is_new_move (first_gamer_numbers)
   (cond (
          (and (= (car first_gamer_numbers) 1) (= (cadr first_gamer_numbers) 1))
           'one
         )
         (
          (and (= (car first_gamer_numbers) 6) (= (cadr first_gamer_numbers) 6))
              'two
         )
    )
)

(defun choose_winner()
    (or
        (if (> (+ (caar first_gamer) (cadar first_gamer))
               (+ (caar second_gamer) (cadar second_gamer)))
            (print_winner "Первый игрок победил" 1) (print_winner "Второй игрок
                победил" 1)
        )
        (print_winner "Ничья")
    )
)

(defun print_winner(str_win type_win)
    (print "Игра_завершена")
    (print_gamer "Первый игрок" first_gamer)
    (if (null second_gamer) (print "Второй игрок не сделал ход.")
        (print_gamer "Второй игрок" second_gamer))
    (if (= 1 type_win) (print str_win) (choose_winner))
)

(defun user_round (gamer)
    (setq gamer (list (roll_dice) (roll_dice)))
    (cond ((is_winner gamer) (list gamer 1))
          ((is_new_move gamer) (user_round gamer))
          (T (list gamer 0))
    )
)
```

Листинг 2.3 – Программный код «Игра в кости»(продолжение)

```
(defun play_game()
    (setq first_gamer (user_round first_gamer))
    (or (cond ((= 1 (cadr first_gamer))
               (print_winner "Абсолютная победа первого игрока" 1)))
        (and (setq second_gamer (user_round second_gamer))
             (cond ((= 1 (cadr second_gamer))
                    (print_winner "Абсолютная победа второго игрока" 1)))
        )
        (choose_winner)
    )
)

(play_game)
```