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»

ОТЧЕТ

по Лабораторной работе №4

по курсу «Функциональное и логическое программирование»

на тему: «Использование управляющих структур, работа со списками»

Студент ИУ7-63Б _____ Миронов Г. А.
(Группа) (Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Толпинская Н. Б.
(Подпись, дата) (И. О. Фамилия)

2022 г.

1 Практическая часть

1.1 Чем принципиально отличаются функции cons, list, append?

Пусть:

Листинг 1.1 – объявление функций из условия

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

В Таблице 1.1 приведены результаты вычисления выражений.

Таблица 1.1 – Результаты вычисления выражений

Выражение	**Результат**
`(cons lst1 lst2)`	`((A B) C D)`
`(list lst1 lst2)`	`((A B) (C D))`
`(append lst1 lst2)`	`(A B C D)`

1.2 Каковы результаты вычисления следующих выражений?

В Таблице 1.2 приведены результаты вычисления выражений.

Таблица 1.2 – Результаты вычисления 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))`

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

1.3.1 Рекурсия 1

Листинг 1.2 – объявление функций из условия

```
1 (defun my-last-recursive-internal (lst)
2   (if (cdr lst)
3    (my-last-recursive-internal (cdr lst))
4    (car lst)))
5 (defun my-last-recursive (lst)
6   (and lst (my-last-recursive-internal lst)))
```

1.3.2 Рекурсия 2

Листинг 1.3 – объявление функций из условия

```
1 (defun my-last-recursive-internal-2 (lst last-el)
2   (if (eql nil lst)
3     last-el
4     (my-last-recursive-internal-2 (cdr lst) (car lst))))
5 (defun my-last-recursive-2 (lst)
6   (my-last-recursive-internal-2 lst nil))
```

1.3.3 С помощью `reduce`

Листинг 1.4 – объявление функций из условия

```
1 (defun my-last-reduce (lst)
2   (reduce #'(lambda (acc el) el) lst))
```

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

1.4.1 Рекурсия

Листинг 1.5 – объявление функций из условия

```
(defun no-last-internal (lst acc)
  (if (cdr lst)
   (no-last-internal (cdr lst) (cons (car lst) acc))
   (nreverse acc)))
(defun no-last (lst)
  (and lst (no-last-internal lst nil)))
```

1.4.2 Функции ядра

Листинг 1.6 – объявление функций из условия

```
(defun no-last-kern (lst)
  (and lst (nreverse (cdr (reverse lst)))))
```

1.5 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 имеет 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 1.7 – Игра в кости

```
(defconstant +dices-amount+ 2)
(defconstant +max-score+ 6)
(defconstant +magic-scores+ '(7 11))
(defconstant +rethrow-combinations+ '((1 1) (6 6)))

(defun iter-times-internal (times fn acc)
  (if (<= times 0)
    acc
    (iter-times-internal (- times 1) fn (cons (funcall fn times)
       acc))))
(defun iter-times (times fn)
  (iter-times-internal times fn nil))

(defun throw-dices (times)
  (iter-times times #'(lambda (_x) (+ (random +max-score+) 1))))

(defun score-with-rules (i dices)
  (let ((sum (reduce #'+ dices)))
    (format T "Player␣~a␣has␣~a!~%" i dices)
    (cond ((member dices +rethrow-combinations+ :test #'equal)
           (score-with-rules i (throw-dices +dices-amount+)))
          ((member sum +magic-scores+) (cons i (+ (* +
             dices-amount+ +max-score+) 1)))
          (T (cons i sum)))))

(defun collect-throws (amount)
  (mapcar #'(lambda (x) (score-with-rules (car x) (cdr x)))
          (iter-times amount #'(lambda (i) (cons i (throw-dices +
             dices-amount+))))))

(defun is-next-better (prev next)
  (or (< (cdr prev) (cdr next))))

(defun play (players_amount)
  (let ((winner (reduce
                  #'(lambda (cur next) (if (is-next-better cur
                     next) next cur))
                  (collect-throws players_amount))))
    (format T "Winner:␣~a" (car winner))))
```

2 Контрольный вопросы

2.1 Базис языка

Базис состоит из:

1. структуры, атомы;
2. встроенные (примитивные) функции (`atom`, `eq`, `cons`, `car`, `cdr`);
3. специальные функции и функционалы, управляющие обработкой структур, представляющих вычислимые выражения (`quote`, `cond`, `lambda`, `label`, `eval`).

2.2 Классификация функций

Функции в `Lisp` классифицируют следующим образом:

- чистые математические функции;
- рекурсивные функции;
- специальные функции — формы (сегодня 2 аргумента, завтра - 5);
- псевдофункции (создают эффект на внешнем устройстве);
- функции с вариативными значениями, из которых выбирается 1;
- функции высших порядков — функционал: используется для синтаксического управления программ (абстракция языка).

По назначению функции разделяются следующим образом:

- конструкторы — создают значение (`cons`, например);
- селекторы — получают доступ по адресу (`car`, `cdr`);
- предикаты — возвращают `Nil`, `T`.
- функции сравнения – такие как: `eq`, `eql`, `equal`, `equalp`.

2.3 Способы создания функций

Функции в Lisp можно задавать следующими способами:

Lambda-выражение

Синтаксис:

(lambda <λ-список> форма)

Пример:

Листинг 2.1 – Функция определенная Lambda-выражением

```
1 (lambda (a b) (sqrt (+ (* a a) (* b b))))
```

Именованая функция

Синтаксис:

(defun <имя функции> <λ-выражение>)

Пример:

Листинг 2.2 – Функция определенная Lambda-выражением

```
1 (defun hyp (a b) (sqrt (+ (* a a) (* b b))))
```

2.4 Работа функций and, or, if, cond

2.4.1 Функция and

Синтаксис:

Листинг 2.3 – функция and

```
1 (and expression-1 expression-2 ... expression-n)
```

Функция возвращает первое expression, результат вычисления которого = Nil. Если все не Nil, то возвращается результат вычисления последнего выражения.

Примеры:

Листинг 2.4 – пример использования `and`

```
(and 1 Nil 2)
```

Результат: `Nil`

Листинг 2.5 – пример использования `and`

```
(and 1 2 3)
```

Результат: 3

2.4.2 Функция `or`

Синтаксис:

Листинг 2.6 – функция `or`

```
(or expression-1 expression-2 ... expression-n)
```

Функция возвращает первое expression, результат вычисления которого не `Nil`. Если все `Nil`, то возвращается `Nil`.

Примеры:

Листинг 2.7 – пример использования `or`

```
(or Nil Nil 2)
```

Результат: 2

Листинг 2.8 – пример использования `or`

```
(or 1 2 3)
```

Результат: 1

2.4.3 Функция `if`

Синтаксис:

Листинг 2.9 – функция `if`

```
(if condition t-expression f-expression)
```

Если вычисленный предикат не `Nil`, то выполняется `t-expression`, иначе - `f-expression`.

Примеры:

Листинг 2.10 – пример использования `if`

```
(if Nil 2 3)
```

Результат: 3

Листинг 2.11 – пример использования `if`

```
(if 0 2 3)
```

Результат: 2

2.4.4 Функция `cond`

Синтаксис:

Листинг 2.12 – Функция `cond`

```
(cond
  (condition-1 expression-1)
  (condition-2 expression-2)
  ...
  (condition-n expression-n))
```

По порядку вычисляются и проверяются на равенство с `Nil` предикаты. Для первого предиката, который не равен `Nil`, вычисляется находящееся с ним в списке выражение и возвращается его значение. Если все предкаты вернут `Nil`, то и `cond` вернет `Nil`.

Примеры:

Листинг 2.13 – Пример использования `cond`

```
(cond (Nil 1) (2 3))
```

Результат: 3

Листинг 2.14 – Пример использования `cond`

```
(cond (Nil 1) (Nil 2))
```

Результат: `Nil`