**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 по курсу «Функциональное и логическое программирование»

**Тема** Использование управляющих структур, работа со списками

**Студент** Богаченко А. Е.

**Группа** ИУ7-65Б

**Оценка (баллы)** _____

**Преподаватели** Строганов Ю. В., Толпинская Н. Б.

Москва — 2022 г.

# Задание 1

Каковы результаты следующих выражений?

Листинг 1 – Задание 1

```
(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)
```

# Задание 2

Каковы результаты следующих выражений?

Листинг 2 – Задание 2

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

# Задание 3

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

Листинг 3 – Задание 3

```
(defun last-recursive (lst)
  (if (cdr lst)
      (last-recursive (cdr lst))
      (car lst)))

(defun int-last-recursive-2 (lst el)
  (if (eql nil lst)
     el
     (last-recursive-2 (cdr lst) (car lst))))
```

```
(defun my-last-recursive-2 (lst)
  (int-last-recursive-2 lst nil))

(defun last-reduce (lst)
  (reduce #'(lambda (acc e) e) lst))
```

# Задание 4

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

Листинг 4 – Задание 4

```
(defun no-last-int (lst acc)
  (if (cdr lst)
      (no-last-int (cdr lst) (cons (car lst) acc))
      (nreverse acc)))

(defun no-last (lst)
  (and lst (no-last-int lst nil)))

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

# Задание 5

в котором 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. Результат игры и значения выпавших костей выводить на экран с помощью функции `print`.

Листинг 5 – Задание 5

```lisp
(defmacro ->> (initial-form &rest forms)
  (reduce #'(lambda (acc next)
              (if (listp next)
                  (append next (list acc))
                  (list next acc)))
          forms
          :initial-value initial-form))

(defconstant +auto-win-scores+ '(7 11))
(defconstant +rethrow-combinations+ '((1 1) (6 6)))

(defun throw-dices ()
  (let ((first-throw (1+ (random 6)))
        (second-throw (1+ (random 6))))
    (list first-throw second-throw)))

(defun score-results (player-id &optional (score 0))
  (let* ((dices (throw-dices))
         (dices-sum (apply #'+ dices))
         (result-score (+ score dices-sum)))
    (format T "Player ~a throws ~a: ~a -> ~a ~%" player-id dices score result-score)
    (cond ((member dices +rethrow-combinations+ :test #'equal)
           (score-results player-id result-score))
          ((member dices-sum +auto-win-scores+)
           (cons player-id -1))
          (T (cons player-id result-score)))))

(defun pick-a-winner (a b)
  (let ((score-a (cdr a))
        (score-b (cdr b)))
    (cond ((= score-a -1) a)
          ((= score-b -1) b)
          ((>= score-a score-b) a)
          (T b))))

(defun play (players)
  (let ((init-player-id 0))
    (->> (loop repeat players collect (incf init-player-id))
         (mapcar #'(lambda (player-id)
                     (score-results player-id)))
         (reduce #'pick-a-winner)
         car
         (format T "Winner: ~a")
         not)))
```

# Контрольные вопросы

# 1. Структуроразрушающие и не разрушающие структуру списка функции

## Не разрушающие структуру списка функции

Данные функции не меняют сам объект-аргумент, а создают копию.

**Функция** `append`

Объединяет списки. Это форма, можно передать больше 2 аргументов. Создает копию для всех аргументов, кроме последнего.

Пример: `(append '(1 2) '(3 4))` — `(1 2 3 4)`.

**Функция** `reverse`

Возвращает копию исходного списка, элементы которого переставлены в обратном порядке. **В целях эффективности работает только на верхнем уровне**.

Пример: `(reverse '(1 2 3 4))` — `(4 3 2 1)`.

**Функция** `last`

Проход по верхнему уровню и возврат последней списковой ячейки.

Пример: `(last '(1 2 3 4))` — `(4)`.

**Функция** `nth`

Возврат указателя от n-ной списковой ячейки, нумерация с нуля.

Пример: `(nth 1 '(1 2 3 4))` — 2.

**Функция** `nthcdr`

Возврат n-ого хвоста.

Пример: `(nthcdr 1 '(1 2 3 4))` — `(2 3 4)`.

**Функция** `length`

Возврат длины списка (**только по верхнему уровню**).
Пример: `(length '(1 2 (3 4)))` — `3`.

**Функция** `remove`

Модифицирует, но работает с копией, поэтому не разрушает. Данная функция удаляет элемент по значению (Часто разрушающая аналогичная функция называется `delete`). По умолчанию используется `eql` для сравнения на равенство, но можно передать другую функцию через ключевой параметр `:test`.

Примеры:

1. `(remove 3 '(1 2 3))` — `(1 2)`;
2. `(remove '(1 2) '((1 2) (3 4)))` — `((1 2) (3 4))`;
3. `(remove '(1 2) '((1 2) (3 4)) :test #'equal)` — `((3 4))`;

**Функция** `rplaca`

Переставляет `car`-указатель на 2 элемент-аргумент ($S$-выражение).
Пример: `(rplaca '(1 2 3) 3)` — `(3 2 3)`.

**Функция** `rplacd`

Переставляет `cdr`-указатель на 2 элемент-аргумент ($S$-выражение).
Пример: `(rplacd '(1 2 3) '(4 5))` — `(1 4 5)`.

**Функция** `subst`

Заменяет все элементы списка, которые равны 2 переданному элементу-аргументу на другой 1 элемент-аргумент. *По умолчанию для сравнения используется функция* `eql`.

Пример: `(subst 2 1 '(1 2 1 3))` — `(2 2 2 3)`.

### Структуроразрушающие функции

Данные функции меняют сам объект-аргумент, невозможно вернуться к исходному списку. Чаще всего такие функции начинаются с префикса `n-`.

#### Функция `nconc`

Работает аналогично `append`, только не копирует свои аргументы, а разрушает структуру.

#### Функция `nreverse`

Работает аналогично `reverse`, но не создает копии.

#### Функция `nsubst`

Работае аналогично функции `nsubst`, но не создает копии.

## 2. Отличие в работе функций `cons`, `list`, `append` и в их результате

Функция `cons` — чисто математическая, конструирует списковую ячейку, которая может вовсе и не быть списком (будет списком только в том случае, если 2 аргументом передан список).

Примеры:

1. `(cons 2 '(1 2))` — `(2 1 2)` — список;
2. `(cons 2 3)` — `(2 . 3)` — не список.

Функция `list` — форма, принимает произвольное количество аргументов и конструирует из них список. Результат — всегда список. При нуле аргументов возвращает пустой список.

Примеры:

1. `(list 1 2 3)` — `(1 2 3)`;

2. `(list 2 '(1 2))` — `(2 (1 2))`;

3. `(list '(1 2) '(3 4))` — `((1 2) (3 4))`;

Функция `append` — форма, принимает на вход произвольное количество аргументов и для всех аргументов, кроме последнего, создает копию, ссылая при этом последний элемент каждого списка-аргумента на первый элемент следующего по порядку списка-аргумента (так как модифицируются все списки-аргументы, кроме последнего, копирование для последнего не делается в целях эффективности).

Примеры:

1. `(append '(1 2) '(3 4))` — `(1 2 3 4)`;

2. `(append '((1 2) (3 4)) '(5 6))` — `((1 2) (3 4) 5 6)`.