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 №6 по курсу "Функциональное и логическое программирование"

Тема Использование управляющих структур, работа со списками

Студент Перестороним П.Г.

Группа ИУ7-63Б

Оценка _____

Преподаватель Толпинская Н. Б.

Москва — 2021 г.

Оглавление

1 Задания **2**

1.1 Чем принципиально отличаются функции `cons`, `list`, `append`? Пусть... 2

1.2 Каковы результаты следующих выражений? 2

1.3 Написать, по крайней мере, два варианта функции, которая возвращает последний элемент своего списка-аргумента . . . 3

1.3.1 Рекурсия 1 3

1.3.2 Рекурсия 2 3

1.3.3 С помощью `reduce` 3

1.4 Написать, по крайней мере, два варианта функции, которая возвращает свой список-аргумент без последнего элемента . 4

1.4.1 Рекурсия 4

1.4.2 Функции ядра 4

1.5 Написать простой вариант игры в кости, в котором... 4

2 Ответы на вопросы к лабораторной работе **6**

2.1 Структуроразрушающие и не разрушающие структуру списка функции 6

2.1.1 Не разрушающие структуру списка функции 6

2.1.2 Структуроразрушающие функции 8

2.2 Отличие в работе функций `cons`, `list`, `append` и в их результате 8

1 Задания

1.1 Чем принципиально отличаются функции cons, list, append? Пусть...

Продолжение задания:

Пусть

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

Каковы результаты следующих выражений?

```
(cons lst1 lst2)
```

Результат: `((A B) C D)`

```
(list lst1 lst2)
```

Результат: `((A B) (C D))`

```
(append lst1 lst2)
```

Результат: `(A B C D)`

1.2 Каковы результаты следующих выражений?

```
(reverse ())
```

Результат: `Nil`

```
(last ())
```

Результат: `Nil`

```
(reverse '(a))
```

Результат: `(a)`

```
(last '(a))
```

Результат: `(a)`

```
(reverse '((a b c)))
```

Результат: `((a b c))`

```
(last '((a b c)))
```

Результат: `((a b c))`

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

1.3.1 Рекурсия 1

```
(defun my-last-recursive-internal (lst)
  (if (cdr lst)
   (my-last-recursive-internal (cdr lst))
   (car lst)))
(defun my-last-recursive (lst)
  (and lst (my-last-recursive-internal lst)))
```

1.3.2 Рекурсия 2

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

1.3.3 С помощью `reduce`

```
(defun my-last-reduce (lst)
  (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

```
(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

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

1.5 Написать простой вариант игры в кости, в котором...

Продолжение задания:

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 1.1: Игра в кости

```
(defconstant +dices-amount+ 2)
(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 6) 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+ 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 Ответы на вопросы к лабораторной работе

2.1 Структуроразрушающие и не разрушающие структуру списка функции

2.1.1 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)`.

2.1.2 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.

2.2 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)`.