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 №7 по курсу "Функциональное и логическое программирование"

Тема Использование управляющих структур, модификация списков

Студент Пересторонин П.Г.

Группа ИУ7-63Б

Оценка _____

Преподаватель Толпинская Н. Б.

Москва — 2021 г.

Оглавление

1 Задания

1.1 Написать функцию, которая по своему аргументу-списку `lst` определяет, является ли он полиндромом (то есть равны ли `lst` и `(reverse lst)`)

```
(defun polyndromp (lst)
  (equal lst (reverse lst)))
```

1.2 Написать предикат `set-equal`, который возвращает `t`, если два его множества-аргумента содержат одни и те же элементы, порядок которых не имеет значения

```
(defun set-equal (lst1 lst2)
  (and (subsetp lst2 lst1) (subsetp lst1 lst2)))
```

1.3 Напишите необходимые функции, которые обрабатывают таблицу из точечных пар: `(страна . столица)`, и возвращают по стране столицу, а по столице — страну

```
(defun get-cptl (cntry cntry-cptl)
  (let ((pair (assoc cntry cntry-cptl)))
```

```
    (and pair (cdr pair))))

(defun get-cntry (cptl cntry-cptl)
  (let ((pair (rassoc cptl cntry-cptl)))
    (and pair (car pair))))
```

1.4 Напишите функцию `swap-first-last`, которая переставляет в списке аргументе первый и последний элементы

1.4.1 Разрушающая структуру

```
(defun nswap-first-last (lst)
  (let ((el1 (car lst))
        (last-el (last lst)))
    (setf (car lst) (car last-el))
    (setf (car last-el) el1)
    lst))
```

1.4.2 Не разрушающая структуру

```
(defun swap-first-last (lst)
  (let ((el1 (car lst))
        (last-el (car (last lst))))
    (reverse (cons el1
          (cdr
            (reverse
              (cons last-el (cdr lst))))))))
```

1.5 Напишите функцию `swap-two-ellement`, которая переставляет в списке-аргументе два указанных своими порядковыми номерами элемента в этом списке

1.6 Разрушающая структуру

```
(defun nswap-two-ellement (n1 n2 lst)
  (let ((len (length lst)))
    (and (< n1 len) (< n2 len)
         (let ((el1 (nth n1 lst))
               (el2 (nth n2 lst)))
           (setf (nth n1 lst) el2)
           (setf (nth n2 lst) el1)
           lst))))
```

1.7 Не разрушающая структуру

```
(defun swap-two-ellement (n1 n2 lst)
  (let ((len (length lst))
        (lst-copy (copy-list lst)))
    (and (< n1 len) (< n2 len)
         (let ((el1 (nth n1 lst))
               (el2 (nth n2 lst)))
           (setf (nth n1 lst-copy) el2)
           (setf (nth n2 lst-copy) el1)
           lst-copy))))
```

1.8 Напишите две функции, `swap-to-left` и `swap-to-right`, которые производят круговую перестановку в списке-аргументе влево и вправо, соответственно

```
(defun swap-to-left (lst)
  (and lst
       (let ((tail (cdr lst))
             (head (car lst)))
         (reverse (cons head (reverse tail))))))

(defun swap-to-right (lst)
  (and lst
       (let ((last-el (car (last lst))))
         (reverse (cdr (reverse (cons last-el lst)))))))
```

2 Ответы на вопросы к лабораторной работе

2.1 Способы определения функций

2.1.1 Через `defun`

Синтаксис:

```
(defun func-name (list-of-argument) function-body)
```

Пример определения:

```
(defun sqr(x) (* x x))
```

Пример вызова:

```
(sqr 2)
```

Результат: 4

2.1.2 Через `lambda`

Синтаксис:

```
(lambda (list-of-arguments) function-body)
```

Пример использования:

```
((lambda (x) (* x x)) 2)
```

Результат: 4

2.2 Варианты и методы модификации списков

2.2.1 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 `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-ому переданному элементу-аргументу на 1-ый элемент-аргумент. *По умолчанию для сравнения используется функция* `eql`.

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

2.2.2 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.

2.3 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)`.