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 №5

по курсу «Функциональное и логическое программирование»

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

Студент ИУ7-63Б (Группа) _____ (Подпись, дата) Недолужко Д. В. (И. О. Фамилия)

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

2022 г.

1 Практическая часть

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

Листинг 1.1 – Функция, проверяющая, является ли список палиндромом

```
1 (defun polyndromp (lst)
2   (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

Листинг 1.2 – Реализация set-equal с использованием subsetp

```
1 (defun set-equal (lst1 lst2)
2     (cond ((subsetp lst2 lst1) (subsetp lst1 lst2))))
```

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

Листинг 1.3 – Реализация указанных функций с использованием assoc/rassoc

```
1 (defun get-by-key (cntry tbl)
2   (let ((pair (assoc cntry tbl :test #'equal)))
3        (cond (pair (cdr pair)))))
4
5 (defun get-by-value (cptl tbl)
6   (let ((pair (rassoc cptl tbl :test #'equal)))
7        (cond (pair (car pair)))))
```

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

Листинг 1.4 – Функция, переставляющая местами первый и последний элемент списка

```
(defun swap-first-last (lst)
  (let ((el1 (car lst))
        (last-el (car (last lst))))
       (nreverse (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

Листинг 1.5 – Функция, переставляющая местами два элемента списка

```
(defun replace-nth (lst i newelem)
  (cond ((null lst) nil)
        ((zerop i) (cons newelem (cdr lst)))
        ((cons (car lst) (replace-nth (cdr lst) (- i 1) newelem))
          )))

(defun swap-two-ellement (i1 i2 lst)
  (cond ((= i1 i2) lst)
        ((replace-nth (replace-nth lst i2 (nth i1 lst)) i1 (nth
          i2 lst)))))
```

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

Листинг 1.6 – Реализация функции циклического сдвига влево

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

Листинг 1.7 – Реализация функции циклического сдвига вправо

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

1.7 Напишите функцию, которая добавляет к множеству двухэлементных списков новый двухэлементный список, если его там нет

Листинг 1.8 – Функция, добавляющая элемент в список при его отсутствии

```
(defun insert (lst ins)
  (cond ((member ins lst :test #'equal) lst)
        (T(cons ins lst))))
```

1.8 Напишите функцию, которая умножает на заданное число-аргумент все числа из заданного списка-аргумента, когда...

1.8.1 а) все элементы списка – числа

Листинг 1.9 – Функция, умножающая каждый элемент списка на число

```
(defun mult-all-numbers (mult lst)
  (mapcar #'(lambda (el) (* el mult)) lst))
```

1.8.2 б) элементы списка – любые объекты

Листинг 1.10 – Функция, умножающая каждое число из списка на число

```
(defun compl-mult-all-numbers (mult lst)
  (mapcar #'(lambda (el)
              (cond ((listp el) (compl-mult-all-numbers mult el))
                    ((* el mult))))
          lst))
```

1.9 Напишите функцию, select-between, которая из списка-аргумента, содержащего только числа, выбирает только те, которые расположены между двумя указанными границами-аргументами и возвращает их в виде списка (упорядоченного по возрастанию списка чисел)

Листинг 1.11 – Реализация select-between

```
(defun get-n (n lst acc)
  (cond ((or (null lst) (<= n 0)) acc)
  ((get-n (1- n) (cdr lst) (cons (car lst) acc)))))

(defun select-between (from to lst)
  (sort (get-n (1+ (- to from)) (nthcdr from lst) Nil) #'<))
```

2 Контрольный вопросы

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 `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)`.

2.1.2 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`

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

2.2 Отличие в работе функций cons, list, append, nconc 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)`.

Функция `nconc` работает аналогично `append`, но не копирует свои аргументы, а разрушает структуру.