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"

Тема Использование управляющих структур, работа со списками

Студент Цветков И.А.

Группа ИУ7-63Б

Оценка (баллы)

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

Москва — 2022 г.

1 Практические задания

1.1 Задание 1

Условие: написать функцию, которая по своему списку-аргументу `lst` определяет является ли он палиндромом (то есть равны ли `lst` и `(reverse lst)`)

```
(defun my-append (lst1 lst2)
    (cond ((null lst1) lst2)
          (T (cons (car lst1) (my-append (cdr lst1) lst2)))
    )
)

(defun my-reverse (lst)
    (cond ((null lst) Nil)
          (T (my-append (my-reverse (cdr lst)) (list (car lst))))
    )
)

(defun palindrom (lst)
    (equal lst (my-reverse lst))
)
```

1.2 Задание 2

Условие: написать предикат set-equal, который возвращает t, если два его множества-аргумента содержат одни и те же элементы, порядок которых не имеет значения.

```
(defun my-remove (elem lst)
    (cond ((null lst) Nil)
          ((equal (car lst) elem) (my-remove elem (cdr lst)))
          (T (cons (car lst) (my-remove elem (cdr lst))))
    )
)

```

```
8
9  (defun set-equal (lst1 lst2)
10     (cond ((and (null lst1) (null lst2)) T)
11           (T (set-equal (my-remove (car lst1) lst1) (my-remove (car
                 lst1) lst2)))
12     )
13 )
```

1.3 Задание 3

Условие: напишите свои необходимые функции, которые обрабатывают таблицу из 4-х точечных пар: (страна . столица), и возвращают по стране - столицу, а по столице — страну.

```
1  (defvar table
2      (list (cons "Country1" "Capital1")
3          (cons "Country2" "Capital2")
4          (cons "Country3" "Capital3")
5          (cons "Country4" "Capital4")
6      )
7  )
8
9
10 (defun get-capital (table country)
11     (cond ((null table) Nil)
12         ((equal (caar table) country) (cdar table))
13         (T (get-capital (cdr table) country))
14     )
15 )
16
17
18 (defun get-country (table capital)
19     (cond ((null table) Nil)
20         ((equal (cdar table) capital) (caar table))
21         (T (get-country (cdr table) capital))
22     )
23 )
```

1.4 Задание 4

Условие: напишите функцию `swap-first-last`, которая переставляет в списке-аргументе первый и последний элементы.

```
(defun my-append (lst1 lst2)
    (cond ((null lst1) lst2)
          (T (cons (car lst1) (my-append (cdr lst1) lst2)))
    )
)

(defun my-reverse (lst)
    (cond ((null lst) Nil)
          (T (my-append (my-reverse (cdr lst)) (list (car lst))))
    )
)

(defun remove-last (lst)
    (my-reverse (cdr (my-reverse lst)))
)

(defun swap-first-last (lst)
    (cons (car (my-reverse lst)) (my-append (cdr (remove-last lst))
       (list (car lst))))
)
```

1.5 Задание 5

Условие: напишите функцию `swap-two-ellement`, которая переставляет в списке-аргументе два указанных своими порядковыми номерами элемента в этом списке.

```
    (defun my-length-rec (lst n)
    (cond ((null lst) n)
          (T (my-length-rec (cdr lst) (+ n 1)))
    )
)


```

```
(defun my-length (lst)
    (my-length-rec lst 0)
)


(defun find-by-index-rec (lst ind cur-ind)
    (cond ((null lst) Nil)
          ((= ind cur-ind) (car lst))
          (T (find-by-index-rec (cdr lst) ind (+ cur-ind 1)))
    )
)


(defun find-by-index (lst ind)
    (find-by-index-rec lst ind 0)
)


(defun swap-two-elements-rec (lst ind1 ind2 src-list res cur-ind)
    (cond ((null lst) (reverse res))
          ((= cur-ind ind1) (swap-two-elements-rec (cdr lst) ind1
             ind2 src-list (cons (find-by-index src-list ind2) res)
             (+ cur-ind 1)))
          ((= cur-ind ind2) (swap-two-elements-rec (cdr lst) ind1
             ind2 src-list (cons (find-by-index src-list ind1) res)
             (+ cur-ind 1)))
          (T (swap-two-elements-rec (cdr lst) ind1 ind2 src-list
             (cons (car lst) res) (+ cur-ind 1)))
    )
)


(defun swap-two-elements (lst ind1 ind2)
    (cond ((= ind1 ind2) lst)
          (T (swap-two-elements-rec lst ind1 ind2 lst () 0))
    )
)
```

1.6 Задание 6

Условие: напишите две функции, `swap-to-left` и `swap-to-right`, которые производят одну круговую перестановку в списке-аргументе влево и вправо, соответственно.

```
(defun my-append (lst1 lst2)
    (cond ((null lst1) lst2)
          (T (cons (car lst1) (my-append (cdr lst1) lst2)))
    )
)

(defun my-reverse (lst)
    (cond ((null lst) Nil)
          (T (my-append (my-reverse (cdr lst)) (list (car lst))))
    )
)

(defun remove-last (lst)
    (my-reverse (cdr (my-reverse lst)))
)

(defun swap-to-left (lst)
    (my-append (cdr lst) (list (car lst)))
)

(defun swap-to-right (lst)
    (my-append (list (car (my-reverse lst))) (remove-last lst))
)
```

1.7 Задание 7

Условие: напишите функцию, которая добавляет к множеству двухэлементных списков новый двухэлементный список, если его там нет.

```
(defun my-append (lst1 lst2)
    (cond ((null lst1) lst2)
          (T (cons (car lst1) (my-append (cdr lst1) lst2)))
    )
```

```
)

(defun is-exist (lst elem)
    (cond ((null lst) Nil)
          ((equal (car lst) elem) T)
          (T (is-exist (cdr lst) elem))
    )
)

(defun add-elem-if-exist (lst elem)
    (if (is-exist lst elem) lst (my-append lst (list elem)))
)
```

1.8 Задание 8

Условие: напишите функцию, которая умножает на заданное число-аргумент первый числовой элемент списка из заданного 3-х элементного списка-аргумента, когда все элементы списка — числа; элементы списка – любые объекты.

```
; a)
(defun mul-first (num lst)
    (cond ((null lst) nil)
          ((numberp (car lst)) (setf (car lst) (* (car lst) num)))
          (T (mul-first num (cdr lst)))
    )
    lst
)

; б)
(defun mul-first (num lst)
    (cond ((null lst) Nil)
          ((symbolp (car lst)) (cons (car lst) (f (cdr lst) num)))
          ((listp (car lst)) (cons (f (car lst) num) (f (cdr lst)
             num)))
          (T (cons (* num (car lst)) (f (cdr lst) num)))
    )
)
```

1.9 Задание 9

Условие: напишите функцию, `select-between`, которая из списка-аргумента из 5 чисел выбирает только те, которые расположены между двумя указанными границами-аргументами и возвращает их в виде списка (упорядоченного по возрастанию списка чисел (+ 2 балла)).

```
1  (defun check-border (num a b)
2      (or (> a num b) (< a num b))
3  )
4
5
6  (defun select-between-rec (lst border_1 border_2 res_lst)
7      (let ((elem (car lst)))
8          (cond ((null lst) res_lst)
9                ((check-border elem border_1 border_2)
10                (select-between-rec (cdr lst) border_1 border_2
11                  (append res_lst (list elem))))
12               (T (select-between-rec (cdr lst) border_1 border_2
                     res_lst))
13         )
14     )
15 )
16
17 (defun select-between (lst border_1 border_2)
18     (let ((res_lst ()))
19         (select-between-rec lst border_1 border_2 res_lst)
20     )
21 )
```

2 Ответы на вопросы к лабораторной работе

2.1 Структуроразрушающие и не разрушающие структуру списка функции

Не разрушающие структуру списка функции – функции, которые не меняют объект-аргумент, а создают копию.

- `length` – возврат длины списка по верхнему уровню;
- `remove` – удаляет элемент по значению (использует `eql`, но можно передать другую функцию через специальный параметр: `:test 'equal`);
- `append` – объединяет списки. Это форма, можно передать больше двух аргументов. Создает копию для всех аргументов, кроме последнего;
- `reverse` – возвращает копию исходного списка, элементы которого переставленыв обратном порядке (только верхний уровень);
- `last` – проход по верхнему уровню и возврат последней списковой ячейки;
- `nth` – возврат указателя от n-ой списковой ячейки, нумерация с нуля.;
- `nthcdr` – возврат n-го хвоста;
- `member` – присутсвует ли эелмент в списке. Возвращает остаток списка, начиная с `car` указателя списковой ячейки;
- `subst` – заменяет все элементы списка, которые равны второму переданному элементу-аргументу на другой первый элемент-аргумент (по умолчанию для сравнения используется функция `eql`).

Структуроразрушающие функции – функции изменяют сам объект, при этом возврат к исходному невозможен.

- `nconc` – аналогично `append`, но не делает коипю, а разрушает структуру;
- `nsubst` – аналогично `subst`, но не создает копию;
- `nreverse` – аналогично `reverse`, но не создает копию.

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

`cons` – имеет фиксированное количество аргументов (два). В случае, когда аргументами являются атомы создает точечную пару. В случает, когда первый аргумент атом а второй список, атом становится головой, а второй аргумент (список) становится хвостом.

`list` – не имеет фиксированное количество аргументов. Создает список, у которого голова - это первый аргумент, хвост - все остальные аргументы.

`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.

`nconc` – аналогично `append`, но не создает копию (разрушает структуру списка).