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

Лабораторная работа №9

По предмету: «Функциональное и логическое программирование»

Студент:
Коротков Андрей Владимирович
Группа: ИУ7-65Б

Преподаватели:
Толпинская Наталья Борисовна
Строганов Юрий Владимирович

Москва, 2020 г.

Задание №1. 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 (x y)
	(and (subsetp x y)  (subsetp y x)))


; рекурсия
(defun decart_inner (elem lst result)
	(cond ((null lst) result)
		(T (decart_inner elem (cdr lst) (cons (list elem (car lst)) result)))
	)
	)

(defun is_some_eql (lst)
	(let ((x (car lst)))
	(cond ((null lst) nil)
		((equal (car x) (cadr x)) T)
		(T (is_some_eql (cdr lst)))
		)))


(defun set-equal (x y num)
	(cond ((and (= num 0) (null x) (not (null y))) nil)
		((null x) T)
		((is_some_eql (reverse (decart_inner (car x) y '()))) (set-equal (cdr x) y
1))
		(T nil)
	))

(defun start_set_equal (x y)
	(and (set-equal x y 0) (set-equal y x 0))
	)
```

Задание №2. Напишите необходимые функции, которые обрабатывают таблицу из точечных пар.

```
; рекурсия
(defun find_capital_list_point(table country)
	(cond ((null table) nil)
		((equal (caar table) country) (cdr (car table)))
		(T (find_country_list_point (cdr table) country))
	))

(defun find_country_list_point(table capital)
	(cond ((null table) nil)
		((equal (cdr (car table)) country) (caar table)
		(T (find_country_list_point (cdr table) country))
	))

; функционалы
(defun find_country_list_point (table capital)
	(let ((result (mapcan #'(lambda (x) (if (not (null x)) (list x)))
	(mapcar #'(lambda (pair) (cond
		((equal (car pair) capital) (cdr pair))
		(T nil)
	)) table))))
	(if (null result)
		nil
		(car result))))

(defun find_capital_list_point (table country)
	(let ((result (mapcan #'(lambda (x) (if (not (null x)) (list x)))
	(mapcar #'(lambda (pair) (cond
		((equal (cdr pair) country) (car pair))
		(T nil)
	)) table))))
	(if (null result)
		nil
		(car result))))
```

Задание №3. Напишите функцию, которая умножает на заданное число-аргумент все числа из заданного списка-аргумента, когда
а) все элементы списка — числа,
б) элементы списка — любые объекты.

```
а) (defun mult (lst num)
	(mapcar #'(lambda (x) (* x num)) lst))
```

```
; рекурсивно
(defun mult (lst num)
	(reverse (mult_inner lst num '())))

(defun mult_inner (lst num rst)
  (cond ((null lst) rst)
    ((numberp (car lst)) (mult_inner (cdr lst) num (cons (* num (car lst)) rst)))
	))
```

б)
```
(defun mult (lst num)
	(mapcar #'(lambda (x)
				(cond ((numberp x) (* x num))
					((atom x) x)
					(T (mult x num)))) lst))
; рекурсивно
(defun mult (lst num)
	(reverse (mult_inner lst num '())))

(defun mult_inner (lst num rst)
	(cond ((null lst) rst)
		((numberp (car lst)) (mult_inner (cdr lst) num (cons (* num (car lst)) rst)))
		((listp (car lst)) (mult_inner (cdr lst) num (cons (mult_inner (car lst) num) rst)))
	))
```

Задание №4. Напишите функцию, которая уменьшает на 10 все числа из списка аргумента этой функции.

```
(defun minus10 (lst)
	(mapcar #'(lambda (x)
				(cond ((numberp x) (- x 10))
					((atom x) x)
					(T (minus10 x)))) lst))

;рекурсия
(defun minus10 (lst)
	(reverse (minus10_inner lst '())))

(defun minus10_inner (lst rst)
  (cond ((null lst) rst)
    ((numberp (car lst)) (minus10_inner (cdr lst) (cons (- (car lst) 10) rst)))
	((listp (car lst)) (minus10_inner (cdr lst) (cons (minus10_inner (car lst) '()) rst)))
	))
```

Задание №5. Написать функцию, которая возвращает первый аргумент списка-аргумента, который сам является непустым списком.

```
(defun get_first_element (lst)
	(if	(and	(listp (car lst))
				(not (null (car lst)))
		)
		(get_first_element (cdr lst))
		(car lst)
	)
)
```

Задание №6. Написать функцию, которая выбирает из заданного списка только те числа, которые больше 1 и меньше 10. (Вариант: между заданными границами.)

```
(defun select_between_inner (lst left right result)
	(mapcar	#'(lambda (x)
				(cond ((listp x) (select_between_inner x left right result))
					((and (numberp x) (> x left) (< x right))
						(nconc result (cons x nil))
					)))
			lst)
	(cdr result))

(defun select_between (lst left right)
	(select_between_inner lst left right (cons nil nil)))

; рекурсия
(defun select_between_inner (lst left right result)
		(let ((x (car lst)))
			(cond ((null lst) result)
				((listp x) (select_between_inner (cdr lst) left right (cons (select_between_inner x left right '()) result)))
				((and (numberp x) (> x left) (< x right)) (select_between_inner (cdr lst) left right (cons x result)))
				(T (select_between_inner (cdr lst) left right result)))))

(defun select_between (lst left right)
	(reverse (select_between_inner lst left right '())))
```

Задание №7. Написать функцию, вычисляющую декартово произведение двух своих списков-аргументов.

```
(defun decart (X Y)
	(mapcan #'
		(lambda (x)
			(mapcar #'
				(lambda (y) (list x y))
				Y
			)
		)
		X
	)
)


;рекурсия
(defun decart_inner (elem lst result)
	(cond ((null lst) result)
			(T (decart_inner elem (cdr lst) (cons (list elem (car lst)) result)))
		)
	)

(defun decart (lst1 lst2 result)
	(cond ((null lst1) result)
			(T (decart (cdr lst1) lst2 (nconc result (reverse (decart_inner (car
lst1) lst2 '()))))))
	))

(defun start_decart (lst1 lst2)
	(decart lst1 lst2 '())
```

Задание №8. Почему так реализован reduce?
(reduce #'+ ()) -> 0
(reduce #'* ()) -> 1

Если подпоследовательность пуста, а начальное значение не задано, то функция вызывается с нулевыми аргументами. Это единственный случай, когда функция вызывается не с двумя аргументами. Также reduce использует аргумент :initial-value. Этот аргумент определяет значение, к которому будет применена функция при обработке первого элемента списка-аргумента. Если список-аргумент пуст, то будет возвращено значение initial-value.

Ответы на теоретические вопросы

- Способы организации повторных вычислений в Lisp?
- Есть 2 способа организации повторных вычислений:
1) Рекурсия
2) Функционал

- Различные способы использования функционалов?
- Существует две группы функционалов: применяющие и отображающие.

- Что такое рекурсия? Способы организации рекурсивных функций?
- Рекурсия — это ссылка на определяемый объект во время его определения. Т.к. в Lisp используются рекурсивно определенные структуры, то рекурсия — это естественный принцип обработки таких структур. Существуют типы рекурсивных функций: хвостовая, дополняемая, множественная, взаимная рекурсия и рекурсия более высокого порядка.

- Способы повышения эффективности реализации рекурсии.
- В целях повышения эффективности рекурсивных функций рекомендуется формировать результат не на выходе из рекурсии, а на входе в рекурсию, все действия выполняя до ухода на следующий шаг рекурсии. Это и есть хвостовая рекурсия.