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 № 10

Дисциплина Функциональное и логическое программирование

Тема Рекурсивные функции

Студент Ильясов И. М.

Группа ИУ7-63Б

Оценка (баллы) _____

Преподаватель Толпинская Н. Б.

Москва, 2020 г.

Задание 1. Пусть list-of-list список, состоящий из списков. Написать функцию, которая вычисляет сумму длин всех элементов list-of-list, т.е., например, для аргумента ((1 2) (3 4)) -> 4.

Функция получает на вход список. Используется функционал mapcar и каскадным образом суммируются длины подсписков списка lst.

```
(defun length-list (lst)
	(apply #'+ (mapcar #'length lst))
)
```

Задание 2. Написать рекурсивную версию (с именем reg-add) вычисления суммы чисел заданного списка. Например: (reg-add (2 4 6)) -> 12

Функция на вход получает список. При каждом вызове функции происходит проверка того, что текущий car-элемент – число, и, если это так, производится суммирование значения текущего car-элемента и следующего. Функция рекурсивно вызывается для оставшегося хвоста списка пока список не равен Nil.

```
(defun reg-add (lst)
	(if lst (+ (if (numberp (car lst)) (car lst) 0) (reg-add (cdr lst))) 0)
)
```

Задание 3. Написать рекурсивную версию с именем recnth функции nth.

Функция на вход получает номер индекса и список. При каждом вызове функции происходит проверка того, что список не пустой, что значение N > 0 и что N = 0. Далее производится рекурсивный вызов для оставшегося хвоста списка с вычитанием из N единицы.

```
(defun recnth (N lst)
	(cond
		((null lst) nil)
		((minusp N) 'error)
		((zerop N) (car lst))
		(T (recnth (1- N) (cdr lst))))
)
```

Задание 4. Написать рекурсивную функцию alloddr, которая возвращает t когда все элементы списка нечетные.

Функция на вход получает список. При каждом вызове функции происходит проверка того, что текущий car-элемент – нечетное число, и, если это так, производится проверка, что оставшийся хвост списка не Nil и рекурсивный вызов функции для оставшегося хвоста списка.

```
(defun alloddr (lst)
	(if lst (if (oddp (car lst)) (if (cdr lst) (alloddr (cdr lst)) T)))
)
```

Задание 5. Написать рекурсивную функцию, относящуюся к хвостовой рекурсии с одним тестом завершения, которая возвращает последний элемент списка – аргументы.

Функция на вход получает список. При каждом вызове функции происходит проверка того, что хвост списка не равен Nil, и, если это так, производится рекурсивный вызов функции для оставшегося хвоста списка. Таким образом возвращается последний элемент списка.

```
(defun lastelem (lst)
	(if (null (cdr lst)) (car lst) (lastelem (cdr lst)))
)
```

Задание 6. Написать рекурсивную функцию, относящуюся к дополняемой рекурсии с одним тестом завершения, которая вычисляет сумму всех чисел от 0 до n-ого аргумента функции.

```
(defun sum (lst N)
	(cond
		((null lst) 0)
		((> N 0) (+ (car lst) (sum (cdr lst) (- N 1))))
		(T (sum (cdr lst) (- N 1)))
	)
)
```

Задание 7. Написать рекурсивную функцию, которая возвращает последнее нечетное число из числового списка, возможно создавая некоторые вспомогательные функции.

Функция на вход получает список. Используется лямбда-функция, которая проверяет, что элемент является нечетным. Функция рекурсивно применяется к оставшемуся хвосту списка, пока список не станет Nil.

```
(defun last-odd (lst)
	(if lst
		(
			(lambda (elem result)
				(cond (result)
					((oddp elem) elem)
				)
			) (car lst) (last-odd (cdr lst))
		)
	)
)
```

Задание 8. Используя cons-дополняемую рекурсию с одним тестом завершения, написать функцию которая получает как аргумент список чисел, а возвращает список квадратов этих чисел в том же порядке.

Функция на вход получает список. Производится проверка, что список не равен Nil и создается при помощи cons список квадратов элементов исходного списка. Функция рекурсивно применяется к оставшемуся хвосту списка, пока список не станет Nil.

```
(defun cons-dop (lst)
	(if lst (cons (* (car lst) (car lst)) (cons-dop (cdr lst))))
)
```

Задание 9. Написать функцию с именем select-odd, которая из заданного списка выбирает все нечетные числа. (Вариант 1: select-even, вариант 2: вычисляет сумму всех нечетных чисел(sum-all-odd) или сумму всех четных чисел (sum-all-even) из заданного списка).

Функция, выбирающая все нечетные числа из списка. Функция на вход получает список. Производится проверка, что список не равен Nil, затем если элемент – нечетное число, то при помощи функции cons собирается новый список из нечетных чисел. Функция рекурсивно применяется к оставшемуся хвосту списка, пока список не станет Nil.

```
(defun select-odd (lst)
	(cond
		((null lst) nil)
		((oddp (car lst)) (cons (car lst) (select-odd (cdr lst))))
		(T (select-odd (cdr lst))))
)
```

Функция, выбирающая все четные числа из списка. Функция на вход получает список. Производится проверка, что список не равен Nil, затем если элемент – четное число, то при помощи функции cons собирается новый список из нечетных чисел. Функция рекурсивно применяется к оставшемуся хвосту списка, пока список не станет Nil.

```
(defun select-even (lst)
	(cond
		((null lst) nil)
		((evenp (car lst)) (cons (car lst) (select-even (cdr lst))))
		(T (select-even (cdr lst))))
)
```

Функция, складывающая все нечетные числа из списка. Функция на вход получает список. Производится проверка, что список не равен Nil, затем если элемент – нечетное число, то при помощи функции + ищется сумма нечетных чисел. Функция рекурсивно применяется к оставшемуся хвосту списка, пока список не станет Nil.

```
(defun sum-all-odd (lst)
	(cond
		((null lst) 0)
		((oddp (car lst)) (+ (car lst) (sum-all-odd (cdr lst))))
		(T (sum-all-odd (cdr lst))))
)
```

Функция, складывающая все четные числа из списка. Функция на вход получает список. Производится проверка, что список не равен Nil, затем если элемент – четное число, то при помощи функции + ищется сумма нечетных чисел. Функция рекурсивно применяется к оставшемуся хвосту списка, пока список не станет Nil.

```
(defun sum-all-even (lst)
	(cond
		((null lst) 0)
```

```
        ((evenp (car lst)) (+ (car lst) (sum-all-even (cdr lst))))
        (T (sum-all-even (cdr lst)))
    )
)
```

Ответы на вопросы

- Способы организации повторных вычислений в Lisp.

Существует 2 способа повторных вычислений:

1) Применяющие (apply, funcall)
2) Отображающие (mapcar, maplist, reduce)

- Различные способы использования функционалов.

- mapcar – функция func применяется к головам первым элементам списков, затем ко вторым и т.д., и результаты применения собираются в результирующий список.
- maplist – func применяется к “хвостам” списков, начиная с полного списка.
- mapcan, mapcon – аналогичны mapcar и maplist, используется память исходных данных, не работают с копиями.
- (reduce #’func lst). reduce – функция func применяется каскадным образом (к первым двум, затем к результату и следующему и так далее).

- Что такое рекурсия? Способы организации рекурсивных функций.

Рекурсия — это ссылка на определяемый объект во время его определения.

Рекурсивная функция вызывает саму себя. Вопрос организации рекурсии — это вопрос эффективного способа организации рекурсии. В Lisp существует классификация рекурсивных функций:

1. простая рекурсия - один рекурсивный вызов в теле
2. рекурсия первого порядка - рекурсивный вызов встречается несколько раз

взаимная рекурсия - используется несколько функций, рекурсивно вызывающих друг друга.

В силу возможной сложности и разнообразия постановок задач, возможны комбинации и усложнения приведенных групп функций. Существуют типы рекурсивных функций: хвостовая, дополняемая, множественная, взаимная рекурсия и рекурсия более высокого порядка.

- Способы повышения эффективности реализации рекурсии.

Использование хвостовой рекурсии. Если условий выхода несколько, то надо думать о порядке их следования. Некачественный выход из рекурсии может привести к переполнению памяти из-за "лишних" рекурсивных вызовов.

Преобразование не хвостовой рекурсии в хвостовую, возможно путем использования дополнительных параметров. В этом случае необходимо использовать функцию-оболочку для запуска рекурсивной функции с начальными значениями дополнительных параметров.