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»

Отчёт по лабораторной работе №5 по дисциплине «Функциональное и логическое программирование»

Тема: Использование функционалов
Студент: Княжев А. В.
Группа: ИУ7-62Б
Оценка (баллы): _____
Преподаватели: Толпинская Н. Б., Строганов Ю. В.

Москва — 2023 г.

Оглавление

1. Практическая часть

1.1. Задание 1

Задание

Напишите функцию, которая уменьшает на 10 все числа из списка-аргумента этой функции, проходя по верхнему уровню списковых ячеек. (* Список смешанный структурированный)

Решение

```
(defun f1 (x)
        (mapcar #'(lambda (el) (if (numberp el) (- el 10) el)) x)
)
```

1.2. Задание 2

Задание

Написать функцию которая получает как аргумент список чисел, а возвращает список квадратов этих чисел в том же порядке.

Решение

```
(defun f2 (x)
        (mapcar #'(lambda (el) (* el el)) x)
)
```

1.3. Задание 3

Задание

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

1. все элементы списка — числа;
2. элементы списка — любые объекты.

Решение

Приведен вариант 2. Для первого варианта убрать проверку на число.

```
(defun f3 (x n)
        (mapcar #'(lambda (el) (if (numberp el) (* el n) el)) x)
)
```

1.4. Задание 4

Задание

Написать функцию, которая по своему списку-аргументу lst определяет является ли он палиндромом (то есть равны ли lst и (reverse lst)), для одноуровнего смешанного списка.

Решение

```
(defun my_and (x y) (and x y))
(defun f4 (lst)
        (reduce #'my_and (mapcar #'equalp lst (reverse lst)))
)
```

1.5. Задание 5

Задание

Используя функционалы, написать предикат set-equal, который возвращает t, если два его множества-аргумента (одноуровневые списки) содержат одни и те же элементы, порядок которых не имеет значения.

Решение

```
(defun in_element (e x)
        (reduce
                #'(lambda (res elem) (or res (equal elem e)))
                x
                :initial-value NIL
        )
)

(defun in (x y)
        (reduce
                #'my_and
                (mapcar #'(lambda (elem) (in_element elem y)) x)
                :initial-value T
        )
)

(defun set-equal (x y)
        (and (in x y) (in y x))
)
```

1.6. Задание 6

Задание

Напишите функцию, select-between, которая из списка-аргумента, содержащего только числа, выбирает только те, которые расположены между двумя указанными числами -

границами-аргументами и возвращает их в виде списка (упорядоченного по возрастанию (+ 2 балла)).

Решение

```
(defun f5 (x a b)
        (reduce
                #'(lambda (res elem)
                        (if (< a elem b) (append res (cons elem nil)) res)
                )
                x
                :initial-value ()
        )
)
```

1.7. Задание 7

Задание

Написать функцию, вычисляющую декартово произведение двух своих списков- аргументов. (Напомним, что A x B это множество всевозможных пар (a b), где a принадлежит A, b принадлежит B).

Решение

```
(defun f6 (x y)
    (reduce
        #'append
        (mapcar
            #'(lambda (elem1)
                (mapcar #'(lambda (elem2) (cons elem1 elem2)) y)
            )
            x
        )
        :initial-value ()
    )
)
```

1.8. Задание 8

Задание

Почему так реализовано reduce, в чем причина?

```
(reduce #'+ ()) -> 0
```

```
(reduce #'* ()) -> 1
```

Решение

В данном случае, `reduce` применяет функцию-аргумент к нулевому количеству аргументов. Так как при отсутствии аргументов, функции `+` и `*` возвращают нейтральные элементы, то `reduce` возвращает соответствующее значение.

1.9. Задание 9

Задание

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

Решение

```
(defun f (l)
        (reduce
                (lambda (res elem) (+ res (length elem)))
                l
                :initial-value 0
        )
)
```