

	<p>Министерство образования и науки 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)</p>
-----------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе № 8

По курсу: "Функциональное и логическое
программирование"

Студент:

Турсунов Жасурбек Рустамович

Группа: ИУ7-66Б

Преподаватели:

Толпинская Наталья Борисовна

Строганов Юрий Владимирович

Москва, 2021 г.

Содержание

Введение	2
Задание 1	3
Задание 2	3
Задание 3	4
Задание 4	4
Ответы на вопросы	5

Цель работы: приобрести навыки использования рекурсии и функционалов.

Задачи работы: изучить способы применения функционалов и рекурсии при обработке одноуровневых и структурированных списков.

Введение

Для организации многократных вычислений в Lisp могут быть использованы функционалы - функции, которые особым образом обрабатывают свои аргументы. Функционалы - это функции более высокого порядка, так как они в качестве своего первого аргумента принимают функциональный объект - функцию, имеющую имя(глобально определенную функцию), или функцию, не имеющую имени(локально определенную функцию). При использовании рекурсии - необходимо обеспечить эффективность работы, путем использования хвостовой рекурсии. В рекурсивных функциях могут быть использованы дополнительные функции - не в аргументах вызова, а вне них. Рекурсивные вызовы могут быть организованы как в одной ветке, так и в нескольких ветках программы. Рекурсивные функции могут вызывать друг друга.

Задание 1

Написать функцию, вычисляющую декартово произведение двух списков-аргументов.

```
1 (defun decart (set1 set2)
2   (mapcan #'(lambda (x)
3     (mapcar #'(lambda (y)
4       (cons x (cons y nil))
5     ) set2
6   )
7   ) set1
8 )
9 )
```

Листинг 1: Функция вычисления декартова произведения

set1 и **set2** - списки-множества.

С помощью `mapcar` получается декартово произведение одного элемента первого множества на второе множество, с помощью `mapcan` получается декартово произведение всех элементов первого множества на второе.

Примеры работы:

```
1 > (decart '(1 2) '(a b c))
2 ((1 A) (1 B) (1 C) (2 A) (2 B) (2 C))
3 > (decart '(1) '(A))
4 ((1 A))
5 > (decart '(1) nil)
6 NIL
```

Задание 2

Написать функцию, которая выбирает из заданного списка только те числа, которые больше 1 и меньше 10.

```
1 (defun selector_func (lst a b)
2   (remove nil
3     (mapcar (lambda (el)
```

```

4         (and
5           (numberp el)
6           (> el a)
7           (< el b) el)
8       ) lst
9   )
10 )
11 )

```

mapcar возвращает исходный список, в котором все непрошедшие проверку элементы заменены на nil. Эти элементы удаляются с помощью remove

Задание 3

Почему так реализовано reduce, в чем причина?

```

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

```

Причина в том, что reduce должен корректно обрабатывать поданный ему список любого размера, поэтому когда список пуст и аргумент :initial-value не задан, возвращается значение функции-аргумента, которую вызвали без аргументов, а если аргумент :initial-value задан, то возвращается его значение.

Задание 4

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

Хвостовая рекурсия:

```

1 (defun count-length (lst)
2   (count-length-helper lst 0)
3 )

```

Листинг 2: Функция-обертка для вычисления суммарной длины списков

lst - входной список.

```

1 (defun count-length-helper (lst length)
2   (cond
3     (
4       (null lst)
5       length
6     )
7     (
8       (listp (car lst))
9       (count-length-helper
10        (cdr lst)
11        (count-length-helper (car lst) length))
12     )
13     (
14       (count-length-helper (cdr lst) (+ length 1))
15     )
16   )
17 )

```

Листинг 3: Функция для вычисления суммарной длины списков

lst - входной список, **length** - длина списка.

Условием выхода из рекурсии является достижение конца списка (первый аргумент - `nil`) - возвращается второй аргумент, в котором накапливается суммарная длина списка. Если голова первого аргумента список, то осуществляется рекурсивный вызов текущей функции для хвоста первого аргумента и рекурсивного вызова, который осуществляется для головы первого аргумента и второго аргумента. Иначе осуществляется рекурсивный вызов текущей функции для хвоста списка и второго аргумента, увеличенного на 1.

Примеры работы:

```

1 > (count-length nil)
2 0
3 > (count-length '(1))
4 1
5 > (count-length '(1 2 3))
6 3
7 > (count-length '((1 2) (3 4)))
8 4

```

```
9      > (count-length '((1 2 (3) 4) 5 (6)))
10      6
```

Ответы на вопросы:

1) Классификация рекурсивных функций

Рекурсия — это ссылка на определяемый объект во время его определения. Способы организации рекурсивных функций:

- Хвостовая рекурсия

Результат формируется не на выходе из рекурсии, а на входе в рекурсию, все действия выполняются до ухода на следующий шаг рекурсии.

```
1      (defun fun (x)
2          (cond
3              (end_test1 end_value1)
4              ...
5              (end_testN end_valueN)
6              ( (fun reduced_x) )
7          )
8      )
9
```

- Рекурсия по нескольким параметрам

```
1      (defun fun (n x)
2          (cond
3              (end_test end_value)
4              ( t (fun (reduced_n) (reduced_x)))
5          )
6      )
7
```

- Дополняемая рекурсия

При обращении к рекурсивной функции используется дополнительная функция не в аргументе вызова, а вне его.

```
1      (defun fun (x)
```

```

2      (cond
3        (test end_value)
4        (t (add_fun add_value (fun reduced_x)) )
5      )
6    )
7

```

- Множественная рекурсия

На одной ветке происходит сразу несколько рекурсивных вызовов. Количество условий выхода также может зависеть от задачи.

```

1  (defun fun (x)
2    (cond
3      (test end_val)
4      ( t (combine (fun changed1_x)
5                  (fun changed2_x))
6      )
7    )
8  )
9

```