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)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе № 9

По курсу: "Функциональное и логическое программирование"

Студент:

Турсунов Жасурбек Рустамович

Группа: ИУ7-66Б

Преподователи:

Толпинская Наталья Борисовна

Строганов Юрий Владимирович

Москва, 2021 г.

Содержание

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 имеющею 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 своих списков-аргументов.

```
(defun decart (set1 set2)
    (mapcan #'(lambda (x)
        (mapcar #'(lambda (y)
            (cons x (cons y nil))
            ) set2
        )
      ) set1
    )
)
```

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:

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

Задание 2

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

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

```
            (and
                (numberp el)
                (> el a)
                (< el b) el)
                ) lst
        )
    )
)
```

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?

```
    (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:

```
(defun count-length (lst)
    (count-length-helper lst 0)
)
```

Листинг 2: Функция-обертка для вычисления суммарной длины списков

lst - входной список.

```
(defun count-length-helper (lst length)
    (cond
        (
            (null lst)
            length
        )
        (
            (listp (car lst))
            (count-length-helper
            (cdr lst)
            (count-length-helper (car lst) length))
        )
        (
            (count-length-helper (cdr lst) (+ length 1))
        )
    )
)
```

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:

```
    > (count-length nil)
    0
    > (count-length '(1))
    1
    > (count-length '(1 2 3))
    3
    > (count-length '((1 2) (3 4)))
    4
```

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

```
(defun fun (x)
    (cond
        (end_test1  end_value1)
        ...
        (end_testN  end_valueN)
        ( (fun reduced_x) )
    )
)

```

- Рекурсия по нескольким параметрам

```
(defun fun (n x)
    (cond
        (end_test   end_value)
        ( t (fun (reduced_n) (reduced_x)))
    )
)

```

- Дополняемая рекурсия

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

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

```
        (cond
            (test end_value)
            (t (add_fun add_value   (fun reduced_x)) )
        )
    )

```

- Множественная рекурсия

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

```
    (defun fun (x)
        (cond
            (test end_val)
            ( t (combine    (fun changed1_x)
                (fun changed2_x))
            )
        )
    )

```