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 № 7

По курсу: "Функциональное и логическое программирование"

Студент:

Турсунов Жасурбек Рустамович

Группа: ИУ7-66Б

Преподователи:

Толпинская Наталья Борисовна

Строганов Юрий Владимирович

Москва, 2021 г.

Содержание

Введение **2**

Задание 1 **3**

Задание 2 **4**

Задание 3 **6**

Задание 4 **7**

Задание 5 **8**

Ответы на вопросы **11**

Цель работы: приобрести навыки работы с управляющими структурами Lisp.

Задачи работы: изучить работу функций с произвольным количеством аргументов, функций разрушающих и неразрушающих структуру исходных аргументов.

Введение

Многие стандартные функции Lisp являются формами и реализуют особый способ работы со своими аргументами. К таким функциям относятся функции, позволяющие работать с произвольным количеством аргументов: and, or, append, или особым образом обрабатывающее свои аргументы: функции cond, if, append, remove, reverse, substitute.

Если на вход функции подается структура данных(списков), то возникает вопрос: сохранится ли возможность в дальнейшем работать с исходными струтурами, или они изменятся в процессе реализации функции. В Lisp существуют функции, использующие списки в качестве аргументов и разрушающие структуру исходных аргументов при этом часть из них позволяет использовать произвольное количество аргументов, а часть нет.

Задание 1

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

```
(defun check_pal(lst)
  (cond
    (
      (null lst)
    )
    (
      (and
        (equal
          (car lst)
          (mapcon #'(lambda (el)
            (cond
              (
                (null (cdr el))
                (car el)
              )
            )
          ) lst
         )
        )
        (check_pal
          (mapcon #'(lambda (el)
            (cond
              (
                (cdr el)
                (cons (car el) nil)
              )
            )
           ) (cdr lst)
          )
        )
      )
    )
  )
)
```

Условием выхода из рекурсии является достижение середины списка(аргумент - nil) - возвращается t.

Осуществляется проверка на равенство первого и последнего элементов списка, если они равны, то осуществляется рекурсивный вызов текущей функции для списка-аргумента без первого и последнего аргументов. Иначе возвращается nil.

Задание 2

Написать предикат set-equal, который возвращает t, если два его множества-аргумента содержат одни и те же элементы, порядок которых не имеет значения.

```
(defun check-sets-equal (set1 set2)
    (and
        (is-subset set1 set2)
        (is-subset set2 set1)
    )
)
```

Листинг 1: Функция проверки эквивалентности двух множеств

set1 и **set2** - списки-множества.

```
(defun is-subset (set subset)
    (cond
        (
            (null subset)
        )
        (
            (and
                (contains set (car subset))
                (is-subset set (cdr subset))
            )
        )
    )
)
```

Листинг 2: Функция проверки вхождения подмножества во множество

set и **subset** - первое множество и второе множество, для которого проверяется является ли оно подмножеством первого множества.

```
(defun contains (lst element)
    (cond
        (
            (null lst)
            nil
        )
        (
            (equal (car lst) element)
        )
        (
            (contains (cdr lst) element)
        )
    )
)
```

Листинг 3: Функция проверки вхождения элемента в список

lst - список, **element** - элемент, для которого проверяется, входит ли он во список-аргумент.

Рекурсивная реализация является более эффективной по памяти, так как в процессе вычислений не выделяет никаких списочных ячеек. Также она является более эффективной по времени, так как прерывает вычисления, как только обнаруживается, что элемент одного множества не входит в другое.

Примеры работы:

```
    > (check-sets-equal '(1 3 2) '(3 2 1))
    T
    > (check-sets-equal '(1 2) '(3 2 1))
    NIL
    > (check-sets-equal '(1) '(1))
    T
    > (check-sets-equal '(E H H H) '(H E))
    T
```

Задание 3

Напишите необходимые функции, которые обрабатывают таблицу из точечных пар: (страна . столица), и возвращают по стране - столицу, а по столице - страну.

```
(defun find_in_pairs (lst name)
    (cond
        (
            (null lst)
            nil
        )
        (
            (equal (caar lst) name)
            (cdar lst)
        )
        (
            (equal (cdar lst) name)
            (caar lst)
        )
        (
            (find_in_pairs (cdr lst) name)
        )
    )
)
```

Листинг 4: Функция поиска в списке точечных пар

lst - список точечных пар, **element** - элемент, для которого проверяется, входит ли он в одну из точечных пар списка-аргумента. Рекурсивная реализация является более эффективной по времени, так как если искомый элемент найден, то он сразу возвращается.

Примеры работы:

```
    > (find_in_pairs '((moscow . russia) (london . england) (washington . usa)) 
    RUSSIA
    > (find_in_pairs '((moscow . russia) (london . england) (washington . usa)) 
    MOSCOW
    > (find_in_pairs '((moscow . russia) (london . england) (washington . usa)) 
    NIL
```

Задание 4

Напишите функцию swap-first-last, которая переставляет в списке аргументе первый и последний элементы.

```
(defun swap-first-last(lst)
    (cond
        (
            (null (cdr lst))
            lst
        )
        (
            (nconc
                (
                    mapcon #'(lambda (el)
                    (cond
                        (
                            (null (cdr el))
                            el
                        )
                    )
                   ) lst
                )
                (
                    mapcon #'(lambda (el)
                    (cond
                        (
                            (cdr el)
                            (cons (car el) nil)
                        )
                    )
                   ) (cdr lst)
                )
                (cons (car lst) nil)
            )
        )
    )
)
```

Листинг 5: Функция перестановки первого и последнего элементов

lst - входной список.

С помощью первого mapcon получается список из последнего элемента списка аргумента, с помощью второй mapcon получается список-аргумент без первого и последнего элементов, и создается список из первого элемента списка-аргумента. Затем эти списки объединяются с помощью nconc.

Примеры работы:

```
> (swap-first-last '(1 2 3 4))
(4 2 3 1)
> (swap-first-last nil)
NIL
> (swap-first-last '(1))
(1)
> (swap-first-last '((1 2) 3 4 (5)))
((5) 3 4 (1 2))
```

Задание 5

Напишите две функции, swap-to-left и swap-to-right, которые производят круговую перестановку в списке-аргументе влево и вправо, соответственно на k позиций.

```
(defun swap-to-left (lst k)
    (cond
        (
            (<= k 0)
            lst
        )
        (
            (swap-to-left
                (nconc
                    (cdr lst)
                    (cons (car lst) nil)
                )
                (- k 1)
            )
```

```
        )
    )
)
```

Листинг 6: Функция круговой перестановки элементов списка влево

lst - входной список, **k** - количество позиций, на которое необходимо выполнить перестановку.

Условием выхода из рекурсии является окончание перестановки элементов (второй аргумент - 0) - возвращается первый аргумент.

С помощью nconc объединяется хвост списка-аргумента со списком, указатель на голову которого указывает на голову списка-аргумента, а указатель на хвост - на nil. Затем осуществляется рекурсивный вызов текущей функции для созданного списка и второго аргумента, уменьшенного на 1.

```
(defun swap-to-right (lst k)
    (cond
        (
            (<= k 0)
            lst
        )
        (
            (swap-to-right
            (nconc
                (mapcon #'(lambda (el)
                    (cond
                        (
                            (null (cdr el))
                            el
                        )
                    )
                  ) lst
                )
                (mapcon #'(lambda (el)
                    (cond
                        (
                            (cdr el)
                            (cons (car el) nil)
                        )
```

```
                    )
                   ) lst
                )
            )
            (- k 1)
        )
      )
    )
)
```

Листинг 7: Функция круговой перестановки элементов списка вправо

lst - входной список, **k** - количество позиций, на которое необходимо выполнить перестановку.

Условием выхода из рекурсии является окончание перестановки элементов (второй аргумент - 0) - возвращается первый аргумент.

С помощью первого mapcon осуществляется получение списка, указатель на голову которого указывает на последний элемент списка-аргумента, а указатель на хвост - на nil. С помощью второго mapcon осуществляется получение списка-аргумента без последнего элемента. Затем с помощью nconc происходит объединение этих двух списков. Далее осуществляется рекурсивный вызов текущей функции для созданного списка и второго аргумента, уменьшенного на 1.

Примеры работы:

```
> (swap-to-right nil 3)
NIL
> (swap-to-right '(1) 3)
(1)
> (swap-to-right '(1 2 3 4 5) 1)
(5 1 2 3 4)
> (swap-to-right '(1 2 3 4 5) 3)
(3 4 5 1 2)
> (swap-to-right '(1 2 3 4 5) 5)
(1 2 3 4 5)
> (swap-to-right '((1) 2 3 (4 5)) 1)
((4 5) (1) 2 3)
> (swap-to-left '(1 2 3 4 5) 1)
```

```
    (2 3 4 5 1)
    > (swap-to-left '(1 2 3 4 5) 3)
    (4 5 1 2 3)
    > (swap-to-left '(1 2 3 4 5) 5)
    (1 2 3 4 5)
    > (swap-to-left '((1) 2 3 (4 5)) 1)
    (2 3 (4 5) (1))
```

Ответы на вопросы:

1) Способы определения функций

Собственную функцию можно определить через спец функцию DEFUN, которая принимает три аргумента(первый должен быть атомом(название), второй аргумент список атомов(аргументы), третий аргумент произвольной формы(тело функции)) Но функция не обязательно должна иметь имя, для того, чтобы определить функцию, не имеющую имени, необходимо воспользоваться лямбда-выражением. Лямбда-выражение – это список, содержащий символ lambda и следующие за ним список аргументов и тело, состоящее из нуля или более выражений.

2) Варианты и методы модификации элементов списка:

Структурно разрушающие функции - это те функции, которых после обработки теряются данные. Эти функции при работе работают с самимми данными, то есть не создают копии чтобы с ними работать

Структурно не разрущающие - это функции в которых после обработки остаются данные. Эти функции при работе при работе создают копию и работают с ней. Например функцци с приставкой n - разрушающие функции, а без - не разрушающие.