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

О т ч е т

по лабораторной работе № 9

Дисциплина: «Функциональное и логическое программирование»

Выполнила: Овчинникова А.П.

Группа: ИУ7-65Б

Преподаватель: Толпинская Н.Б.

Строганов Ю.В.

Москва, 2020

Теоретическая часть.

Вопрос 1.

Повторные вычисления в Лисп могут быть организованы с помощью рекурсии или с помощью функционалов.

Функционалы бывают:

- применяющие:
  - *(apply #' fun args)* – применяет функцию fun к аргументам args, которых должно быть не менее одного. Последний аргумент args должен быть списком. Функция fun может быть также символом, и в этом случае используется связанная с ним глобальная функция.
  - *(funcall #'fun args)* – применяет функцию fun к аргументам args. Функция может быть задана как символ, и в этом случае используется определение глобальной функции, связанной с этим символом.
- отображающие:
  - *(mapar #'fun lst)* – ко всем элементам списка *lst* применяется функция *fun*. Из результатов применения этой функции к элементам списка формируется результирующий список. Функция *fun* должна быть одноаргументной.
  - *(mapcar #'fun lst1 … lstN)* – применяет функцию *fun* сначала ко всем первым элементам списков *lst1 … lstN*, затем ко всем последовательным элементов каждого списка *lst1 … lstN*. Прекращает работу, когда заканчиваются элементы самого короткого из списков *lst1 … lstN*. В результате получается список списков результатов каждого вызова функции *fun*. Функция *fun* должна иметь n аргументов.
  - *(maplist #'fun lst)* – вызывает функцию *fun* n раз (n – длина списка *lst*) для *lst* целиком, затем для всех последовательных cdr списка *lst*, заканчивая (n-1)-м. Возвращает список

значений, полученных функцией *fun*. Функция *fun* должна быть одноаргументной.

- *(maplist #'fun lst1 ... lstN)* – вызывает функцию *fun* n раз (n – длина кратчайшего из списков *lst1 ... lstN*) для каждого списка *lst1 ... lstN* целиком, затем для всех последовательных cdr каждого *lst1 ... lstN*, заканчивая (n-1)-м. Возвращает список значений, полученных функцией *fun*. Функция *fun* должна иметь n аргументов.

Во всех случаях функция *fun* может быть задана именем функции или лямбда-определением. Здесь лямбда-определение будет более эффективным, так как нет необходимости искать функцию по имени среди атомов.

Вопрос 2.

Использование функционалов mapcar и maplist описано в вопросе 1. Кроме этого, существуют также следующие функционалы:

1. *(mapcan #'fun prolist)* – эквивалент применения nconc к результату вызова mapcar с теми же аргументами.
2. *(mapcon #'fun prelist)* – эквивалент применения nconc к результату вызова maplist с теми же аргументами.
3. *(find-if #'predicate proseq)* – возвращает первый соответствующий предикату элемент proseq.
4. *(remove-if #'predicate proseq)* – возвращает последовательность, похожую на proseq, но без всех элементов, для которых справедлив предикат predicate.
5. *(reduce #fun lst)* – применяет fun к элементам lst каскадным образом. Если lst пуст, то fun вызывается без аргументов.
6. *(some #'predicate proseq)* – возвращает истину, если предикат predicate, который должен быть функцией стольких аргументов, сколько задано последовательностей proseq, возвращает истину,

будучи примененным к первым элементам всех последовательностей, или ко вторым, или к n-м, где n – длина кратчайшей последовательности proseq. Проход по последовательностям останавливается тогда, когда предикат вернет истину.

7. *(every #'predicate preseq)* – возвращает истину, если предикат predicate, который должен быть функцией стольких аргументов, сколько последовательностей передано, истинен для всех первых элементов последовательностей proseq, затем для всех вторых элементов, и так до n-го элемента, где n – длина кратчайшей последовательности proseq. Проход по последовательностям завершается, когда предикат возвратит nil. В этом случае в результате всего вызова возвращается nil.

Вопрос 3.

Рекурсия — это ссылка на определяемый объект во время его определения. Т. к. в Lisp используются рекурсивно определенные структуры (списки), то рекурсия — это естественный принцип обработки таких структур.

Классификация рекурсивных функций.

1. **Хвостовая рекурсия**. В целях повышения эффективности рекурсивных функций рекомендуется формировать результат не на выходе из рекурсии, а на входе в рекурсию, все действия выполняя до ухода на следующий шаг рекурсии. Это и есть хвостовая рекурсия.
2. **Рекурсия по нескольким параметрам.**
3. **Дополняемая рекурсия** — при обращении к рекурсивной функции используется дополнительная функция не в аргументе вызова, а вне его.

4. **Функции множественной рекурсии**. На одной ветке происходит сразу несколько рекурсивных вызовов. Количество условий выхода также может зависеть от задачи.

Вопрос 4.

В целях повышения эффективности рекурсивных функций рекомендуется формировать результат не на выходе из рекурсии, а на входе в рекурсию, все действия выполняя до ухода на следующий шаг рекурсии, то есть использовать хвостовую рекурсию.

Функция *(set-difference prolist1 prolist2)* возвращает список элементов *prolist1*, не имеющихся в *prolist2*.

Функция *(union prolist1 prolist2)* возвращает список элементов, имеющихся в *prolist1* и *prolist2*. Сохранение порядка следования элементов в возвращаемом списке не гарантируется. Если либо *prolist1*, либо *prolist2* имеет повторяющиеся элементы, то эти элементы будут дублироваться в возвращаемом списке.

Практическая часть.

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.

Первый вариант, который работает, даже если множества-аргументы содержат повторяющиеся элементы. Функция *my_member* принимает два параметра: el – элемент, который мы ищем в списке lst. В случае, если список lst пуст, возвращается nil. Функция сравнивает каждый элемент списка lst с элементом el. Если ни один элемент lst не равен el (то есть мы дошли до конца списка и не нашли ни одного совпадения), возвращается nil. Если в lst нашелся элемент, равный el, происходит выход из рекурсии и возвращается t.

Функция *check-set*, принимающая два параметра (множества set1 и set2), проверяет, все ли элементы множества set1 содержатся в множестве set2. Если

в set1 обнаруживается элемент, которого нет в set2, происходит выход из рекурсии и возвращается nil. Если же мы дошли до конца set1 (т.е. в результате рекурсивных вызовов set1 стало пустым), возвращается t.

Функция set-equal, принимающая два параметра-множества set1 и set2, проверяет, что все элементы set1 содержатся в set2 и что все элементы set2 содержатся в set1. Если это так, возвращается t.

```
(defun my_member (el lst)
        (cond
           ( (null lst) nil )
                ( (equal el (car lst)) t )
                ( t (my_member el (cdr lst)) )
        ))
(defun set-equal (set1 set2)
   (cond ( (and (check-set set1 set2) (check-set set2 set1) ) t)
   )
)
(defun check-set (set1 set2)
   (cond
      ( (null set1) t)
      ( (my_member (car set1) set2) (check-set (cdr set1 ) set2) )
   )
)
```

Второй вариант:

```
(defun set-equal2 (set1 set2)
   (and
      (every #'(lambda (elem)
                 (my_member elem set2)
               )
             set1
      )
```

```
        (every #'(lambda (elem)
                    (my_member elem set1)
                 )
                set2
        )
    )
)
```

Функция set-equal2 работает по тому же принципу, что и set-equal, но использует для этого функционал every. set-equal2 также проверяет, что каждый элемент set1 содержится в set2 и что каждый элемент set2 содержится в set1. Функция работает, если множества содержат повторяющиеся элементы.

Задание 3.

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

Функции create-list и create-list2 принимают в качестве аргументов два списка – список стран и список столиц, а возвращают таблицу точечных пар (СТРАНА . СТОЛИЦА). Функция create-list делает это рекурсивно (рекурсивные вызовы происходят до тех пор, пока один из двух списков-параметров не окажется пустым): используя cons-дополняемую рекурсию.

Функция *create-list2* создает таблицу точечных пар, используя mapcar.

```
(defun create-pair (lst1 lst2)
   (cond ( (or (null lst1) (null lst2)) Nil)
         ( t (cons (cons (car lst1) (car lst2)) (create-pair (cdr lst1) (cdr lst2) )) )
   )
)
 (defun create-list2 (lst1 lst2)
   (mapcar #'cons lst1 lst2)
)
 (create-list '(ФИНЛЯНДИЯ ГЕРМАНИЯ) '(ХЕЛЬСИНКИ БЕРЛИН))
```

(create-list2 '(ФИНЛЯНДИЯ ГЕРМАНИЯ) '(ХЕЛЬСИНКИ БЕРЛИН))

Функции find_by_capital и find_by_capital2 производят поиск в таблице точечных пар по столице (из расчета, что элементы таблицы имеют вид (СТРАНА . СТОЛИЦА)). Функция find_by_capital рекурсивная (исп. хвостовую рекурсию), выход из рекурсии происходит, если таблица в результате рекурсивных вызовов стала пустая (т.е. страна с данной столицей не найдена) или если найдено совпадение (в этом случае возвращается страна).

Функция find_by_capital2 использует функционал mapcar, поэтому в случае, когда в таблице находится несколько стран с данной столицей, возвращается список с названиями всех стран с данной столицей.

```
(defun find_by_capital (table capital)
   (cond ( (null table) 'unknown)
         ( (eql (cdr(car table)) capital) (car (car table)) )
         ( t (find_by_capital (cdr table) capital) )
   )
)
(defun find_by_capital2 (table capital)
   (mapcan #'(lambda (table_elem)
               (cond
                  ((eql (cdr table_elem) capital) (list (car table_elem)))
               )
          )
          table
   )
)
```

(find_by_capital '((ФИНЛЯНДИЯ . ХЕЛЬСИНКИ) (ГЕРМАНИЯ . БЕРЛИН) (РОССИЯ . МОСКВА)) 'ХЕЛЬСИНКИ)

(print (find_by_capital2 '((ФИНЛЯНДИЯ . ХЕЛЬСИНКИ) (ГЕРМАНИЯ . БЕРЛИН) (РОССИЯ . МОСКВА)) 'ХЕЛЬСИНКИ))

Функции find_by_country и find_by_country2 работают аналогично функциям find_by_capital и find_by_capital2, только поиск осуществляется по стране.

```
(defun find_by_country (table country)
   (cond ( (null table) 'unknown)
        ( (eql (car(car table)) country) (cdr (car table)) )
        ( t (find_by_country (cdr table) country) )
   )
)
(defun find_by_country2 (table country)
   (mapcan #'(lambda (table_elem)
              (cond
                 ((eql (car table_elem) country) (list (cdr table_elem)))
              )
         )
         table
   )
)
(find_by_country '((ФИНЛЯНДИЯ . ХЕЛЬСИНКИ) (ГЕРМАНИЯ . БЕРЛИН) (РОССИЯ . МОСКВА)) 'ФИНЛЯНДИЯ)
(find_by_country2 '((ФИНЛЯНДИЯ . ХЕЛЬСИНКИ) (ГЕРМАНИЯ . БЕРЛИН) (РОССИЯ . МОСКВА)) 'ФИНЛЯНДИЯ)
```

Задание 7.

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

- все элементы списка – числа;
- элементы списка – любые объекты.

Функции mult и mult2 принимают два аргумента: список lst и число num. Функции mult и mult2 используют функционал mapcar. В функции mult2

делается проверка, является ли текущий элемент списка lst списком, числом, или не списком и не числом. Если это число, то оно умножается на num. Если это не число и не список, то никаких действий не производится. Если это список, то он обрабатывается так же, как и исходный список.

Функции mult3 и mult4 тоже принимают два аргумента: список lst и число num. Они являются рекурсивными. В функции mult4 выполняется проверка природы элементов списка lst, аналогично той проверке, что проводится в функции mult2.

Ни в одной из четырех функций не проводится проверка природы аргумента num. Если num не является числом, произойдет ошибка.

```
(defun mult (lst num)
   (defun m (n)
      (* n num)
   )
   (mapcar #'m lst)
)
(defun mult2 (lst num)
   (defun m (n)
      (cond
         ( (listp n) (mult2 n num) )
         ( (not (numberp n)) n )
         ( (numberp n) (* n num) )
      )
   )
   (mapcar #'m lst)
)
(defun mult3 (lst num)
   (cond
      ( (null lst) Nil)
      ( (cons (* (car lst) num) (mult3 (cdr lst) num)) )
```

```
    )
)
(defun mult4 (lst num)
    (cond
        ( (null lst) Nil)
        ( (listp (car lst)) (cons (mult4 (car lst) num) (mult4 (cdr lst) num) ) )
        ( (not (numberp (car lst))) (cons (car lst) (mult4 (cdr lst) num) ))
        ( (numberp (car lst)) (cons (* (car lst) num) (mult4 (cdr lst) num)) )
    )
)
```

Задание 2.

Функции decrease и decrease2 принимают два аргумента – список и число, на которое необходимо уменьшить все числа списка-аргумента. Функция decrease использует функционал mapcar. mapcar применяет внутреннюю функцию inner ко всем элементам исходного списка lst. Если текущий элемент не является числом, то он просто возвращается. Если текущий элемент является числом, то он уменьшается на значение value и возвращается. Если текущий элемент является списком, то он обрабатывается точно так же, как исходный список.

Функция decrease является рекурсивной и также осуществляет проверку своих аргументов – числа уменьшаются на value, списки-элементы обрабатываются так же, как исходный список, а с элементами, не являющимися числами или списками, никаких действий не производится.

```
(defun decrease2 (lst value)
    (defun inner (elem)
        (cond
            ( (listp elem) (mapcar #'inner elem) )
            ( (not (numberp elem)) elem)
            ( (numberp elem) (- elem value))
        )
```

```
    )
        (mapcar #'inner lst)
)
(defun decrease (lst value)
    (cond
        ( (null lst) Nil )
        ( (listp (car lst)) (cons (decrease (car lst) value) (decrease (cdr lst) value)) )
        ( (numberp (car lst)) (cons (- (car lst) value) (decrease (cdr lst) value)) )
        ( (not (numberp (car lst))) (cons (car lst) (decrease (cdr lst) value)) )
    )
)
```

Задание 3.

Написать функцию, которая возвращает первый аргумент списка-аргумента, который сам является непустым списком.

Функция first_not_empty_list рекурсивная. Принимает один параметр – список. Выход из рекурсии происходит, если список-аргумент в результате рекурсивных вызовов оказался пустым (или если он изначально был пустым) или если найден элемент, являющийся непустым списком. В первом случае возвращается nil, а во втором – найденный элемент.

Функция first_not_empty_list2 также принимает один аргумент-список и для поиска использует функционал some. Поиск прекращается, когда найден первый непустой список-элемент (это принцип работы функционала some). Если не найдено ничего, возвращается nil.

```
(defun first_not_empty_list (lst)
    (cond
        ( (null lst) 'not-found)
        ( (and (listp (car lst)) ( not (null (car lst) ) ) ) (car lst))
        ( t (first_not_empty_list (cdr lst)) )
    )
```

```
)
(defun first_not_empty_list2 (lst)
    (some #'(lambda (elem)
                (cond
                    ( (and (listp elem) (not (null elem))) elem)
                )
            )
        lst
    )
)
```

Задание 4.

Написать функцию, которая выбирает из заданного списка только те числа, которые больше 1 и меньше 10. (Вариант: между двумя заданными границами.)

Функции chose_between и chose_between2 принимают три аргумента: список lst, минимальная граница min и максимальная граница max. Функция chose_between использует фукционал mapcan. mapcan применяет функцию inner ко всем элементам списка-аргумента lst. В случае, если какой-либо элемент lst оказался списком, он обрабатывается так же, как и исходный список. Если какой-либо элемент lst не является числом, он просто не включается в результат.

Функция chose_between2 обрабатывает элементы lst по такому же принципу, однако является рекурсивной.

```
(defun chose_between (lst min max)
  (defun inner (elem)
    (cond
      ( (listp elem) (list (mapcan #' inner elem)) )
      ( (and (numberp elem) (<= elem max) (>= elem min)) (list elem))
    )
  )
```

```
    (mapcan #'inner lst)
  )
  (defun chose_between2 (lst min max)
    (defun inner ()
      (cond
        ( (null lst) Nil)
        ( (numberp (car lst))
          (cond
            ((and (<= (car lst) max) (>= (car lst) min))
              (cons (car lst) (chose_between2 (cdr lst) min max)))
            ( (cons Nil (chose_between2 (cdr lst) min max)) )
          )
        )
        ( (listp (car lst)) (cons (chose_between2 (car lst) min max) (chose_between2 (cdr lst) min max)) )
        ( (not (numberp (car lst))) (cons Nil (chose_between2 (cdr lst) min max)) )
      )
    )
    (remove Nil (inner))
  )
```

Задание 5.

Написать функцию, вычисляющую декартово произведение двух своих списков-аргументов.

Функции decart и decart2 принимают два аргумента-множества.

Функция decart (из лекции) использует функционал mapcan. Верхняя (первая) лямбда-функция получает по очереди каждый элемент множества X, а нижняя лямбда-функция, получая по очереди каждый элемент множества Y, объединяет текущий элемент x множества X с текущим элементом y

множества Y (объединяет – то есть создает список из двух списковых ячеек: (x y)).

Функция decart3 рекурсивная и использует внутри другую функцию – inner. Фукнция inner принимает два параметра-можества. Она составляет список пар, состоящих из первого элемента первого множества и каждого элемента второго множества. В функции inner используется рекурсивная функция inner2, которая выполняет всю работу.

```
(defun decart (X Y)
        (mapcan #'
                (lambda (x)
                        (mapcar #'
                                (lambda (y)
                                   (list x y)
                                )
                                Y
                        )
                )
                X
        )
)
(defun decart2 (x y)
   (mapcar #'(lambda (set1)
                (cond
                   ( (null y) Nil)
                   ( t (cons (list (car set1) (car y)) (dec set1 (cdr y)) ))
                )
           )
      x
   )
)
```

```
(defun inner (set1 set2)
   (defun inner2 (set)
      (cond
         ( (null set) Nil )
         ( t (cons (cons (car set1) (cons (car set) Nil)) (inner2 (cdr set)) ) )
      )
   )
   (inner2 set2)
)
(defun decart3 (set1 set2)
   (cond
      ( (null set1) Nil )
      ( t (append (inner set1 set2) (decart3 (cdr set1) set2)) )
   )
)
```

Задание 6.

(reduce #'+ ()) = 0

(reduce #'fun lst) применяет функцию fun к элементам lst каскадным образом. Если lst состоит из одного элемента, возвращает сам элемент. Если lst пуст, то функция вызывается без аргументов. В данном примере без аргументов вызывается функция +, которая при таком вызове возвращает 0.

(reduce #'+0) = Error. Второй аргумент reduce должен быть списком.