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»

Дисциплина: «Функциональное и логическое программирование»

Лабораторная работа №9

Студент: Левушкин И. К.
Группа: ИУ7-62Б
Преподаватели: Толпинская Н. Б.,
Строганов Ю. В.

Москва, 2020 г.

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

Примечание: под множеством я понимаю список НЕПОВТОРЯЮЩИХСЯ элементов, порядок которых не имеет значения

Поскольку в скинутой Натальей Борисовной лекции есть пример преобразования списка в множество, где элементы в результате не повторяются:

Несколько примеров на функционалы

Превращение списка в множество. Обратите внимание, что порядок следования элементов в результате не очевидный

```
(defun consist_of (lst)
        (if member (car lst) (cdr lst) 1 0)
)
(defun all_last_element (lst)
        (if (eql (consist_of lst) 0)
                (list (car lst))
        )
)
(defun collection_to_set (lst)
        (mapcon #'all_last_element lst)
)
(collection_to_set '(i t i g t k s i f k)) -> (g t s i f k)
```

Для вводимой коллекции: consist_of даёт (1 1 1 0 0 1 0 0 0 0), далее все 0 добавляются в список.

Рис. 1: Пример множества из лекции по рекурсии

Реализация задания

```
;##################
;функционалы
;##################

(defun reduce_and (lst); применяем and к списку
    (reduce #'(lambda (prev lst) (and prev lst)) lst)
)

(defun set-equal (col1 col2)
    (cond ((and (null col1) (null col2)) t)
          ((or (null col1) (null col2)) nil)
          ((equal (length col1) (length col2)) (
              reduce_and (mapcar (lambda (el) (if (member el col2) t nil)) col1))
              )
          (t nil)
    )
)
```

Рис. 2: Реализация функции set-equal с использованием функционалов

```
;##################
;хвостовая рекурсия
;##################

(defun set-equal (col1 col2)
    (cond ((and (null col1) (null col2)) t)
          ((or (null col1) (null col2)) nil)
          (t  (if (member (car col1) col2)
          (set-equal (cdr col1) (set-difference col2 (cons (car col1) nil))) nil
              )
          )
    )
)
```

Рис. 3: Рекурсивная реализация фунции set-equal

Назначение параметров функций

- Функция reduce_and применяет and к списку
- Функция member проверяет, есть ли элемент el в списке col2
- Функция set_difference получает на выходе разницу между списком col2 и головой списка col1

Результаты работы

Функции, реализованные с помощью функционалов и с помощью хвостовой рекурсии выдают одинаковые результаты на одних и тех же параметрах:

Выражение	**Результат**
'(1 2 3 4 5) '(4 3 2 5 1)	T
'(1 2 3 4 5 5 4 3 2 1) '(4 3 2 5 1)	NIL
'(1 2 3) '(1 2)	NIL
'(1) '(1)	T
'() '()	T

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

Реализация задания

```
;##################
;функционалы
;##################

;создание списка из точечных пар
(defun two_dot_pair_list (capital country) (
    mapcar #'cons capital country
))

;нахождение страны по столице в списке точечных пар
(defun capital_for_two_dot_pair_list (lst capital) (
    reduce (lambda (answer lst) (
        if (equal answer Nil) (
            if (equal (car lst) capital) (cdr lst) Nil
            ) answer
    )) lst :initial-value Nil
))

;нахождение столицы по стране в списке точечных пар
(defun country_for_two_dot_pair_list (lst country) (
    reduce (lambda (answer lst) (
        if (equal answer Nil) (
            if (equal (cdr lst) country) (car lst) Nil
            ) answer
    )) lst :initial-value Nil
))
```

Рис. 4: Реализация функций с использованием функционалов

```
;##################
;хвостовая рекурсия
;##################

;создание списка из точечных пар
(defun create_list_recurs (capital country result)
    (cond ((or (null country) (null capital)) result)
          (t (create_list_recurs (cdr capital) (cdr country)
          (nconc result (list (cons (car country) (car capital))))))
    )
)

(defun two-dot-pair-list (capital country)
    (create_list_recurs capital country nil)
)
```

Рис. 5: Рекурсивная реализация функций создания списка точечных пар

```
;нахождение страны по столице в списке точечных пар

(defun check_capital (lst capital)
    (if (equal (cdar lst) capital) t nil)
)

(defun capital_for_two_dot_pair_list (lst capital)
    (cond ((null lst) nil)
        ((check_capital lst capital) (caar lst))
        (t (capital_for_two_dot_pair_list (cdr lst) capital))
    )
)
```

Рис. 6: Рекурсивная реализация функций нахождения страны по столице в списке точечных пар

```
;нахождение столицы по стране в списке точечных пар

(defun check_country (lst country)
    (if (equal (caar lst) country) t nil)
)

(defun country_for_two_dot_pair_list (lst country)
    (cond ((null lst) nil)
        ((check_country lst country) (cdar lst))
        (t (country_for_two_dot_pair_list (cdr lst) country))
    )
)
```

Рис. 7: Рекурсивная реализация функций нахождения столицы по стране в списке точечных пар

Назначение параметров функций

- Функция two_dot_pair_list создает список точечных пар
- Функция capital_for_two_dot_pair_list ищет страну (второй элемент точечной пары) по столице (первый элемент точечной пары)
- Функция country_for_two_dot_pair_list ищет столицу (первый элемент точечной пары) по стране (второй элемент точечной пары)

Описание функционалов: В предыдущих двух функциях используется функция reduce с начальным значением Nil, чтобы иметь возможность накапливать полученный результат в лямбда-функции в первом параметре.

Описание хвостовой рекурсии

- Функция create_list_recurs является рекурсивной реализацией поставленной задачи, накапливая результат в result (two-dot-pair-list - оберточная функция)
- Функция check_capital сравнивает столицу capital со столицей в точечной паре головы списка lst

Результаты работы

Функции, реализованные с помощью функционалов и с помощью хвостовой рекурсии выдают одинаковые результаты на одних и тех же параметрах:

Выражение	Результат two_dot_pair_list
'(1 2 3 4 5 6) '(a b c d e f)	((1 . A) (2 . B) (3 . C) (4 . D) (5 . E) (6 . F))
'(1 2 3 4 5 6) '(a b c d e)	((1 . A) (2 . B) (3 . C) (4 . D) (5 . E))
'(1 2 3 4 5 6) '(a b c d e f g)	((1 . A) (2 . B) (3 . C) (4 . D) (5 . E) (6 . F))
'() '()	NIL

Выражение	Результат capital_for_two_dot_pair_list
'((1 . A) (2 . B) (3 . C) (4 . D) (5 . E) (6 . F)) 1	A
'((1 . A) (2 . B) (3 . C) (4 . D) (5 . E) (6 . F)) 5	E
'((1 . A) (2 . B) (3 . C) (4 . D) (5 . E) (6 . F)) 7	NIL
'((1 . A) (2 . B) (3 . C) (4 . D) (5 . E) (6 . F)) 'A	NIL

Выражение	Результат country_for_two_dot_pair_list
'((1 . A) (2 . B) (3 . C) (4 . D) (5 . E) (6 . F)) 'A	1
'((1 . A) (2 . B) (3 . C) (4 . D) (5 . E) (6 . F)) 'E	5
'((1 . A) (2 . B) (3 . C) (4 . D) (5 . E) (6 . F)) 'G	NIL
'((1 . A) (2 . B) (3 . C) (4 . D) (5 . E) (6 . F)) 1	NIL

5.7. 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.

Реализация задания

```
;###################
;функционалы
;###################

(defun multiply_numbers (lst number)
    (mapcar (lambda (x) (* x number)) lst)
)
```

Рис. 8: Реализация функции multiply_numbers с использованием функционалов

```
;##################
;хвостовая рекурсия
;##################


(defun recurs_multiply (lst number result)
    (cond ((null lst) result)
          (t (recurs_multiply (cdr lst) number (cons (* (car lst) number) result)))
    )
)

(defun multiply_numbers (lst number)
    (reverse (recurs_multiply lst number nil))
)
```

Рис. 9: Рекурсивная реализация функции multiply_numbers

```
;##################
;функционалы+рекурсия
;##################
(defun multiply (lst number)
    (mapcar (lambda (obj)
                (cond ((numberp obj) (* obj number))
                      ((listp obj) (multiply obj number))
                      (t obj)
                )
            )
    lst)
)
```

Рис. 10: Реализация функции multiply с использованием функционалов и рекурсии

Назначение параметров функций

- Функции multiply_numbers умножают на заданное число-аргумент все числа из заданного списка-аргумента, когда все элементы — числа
- Функция recurs_multiply является рекурсивной реализацией поставленной задачи, накапливая результат в result (multiply_numbers - оберточная функция)
- Функция multiply умножаетт на заданное число-аргумент все числа из заданного списка-аргумента, когда все элементы — любые объекты, причем

использует она как функционалы, так и хвостовую рекурсию.

Результаты работы

Функции, реализованные с помощью функционалов и с помощью хвостовой рекурсии выдают одинаковые результаты на одних и тех же параметрах:

Выражение	**Результат multiply_numbers**	**Результат multiply**
'(1 2 3 4 5 6) 3	(3 6 9 12 15 18)	(3 6 9 12 15 18)
'(1 2 3 4 5 6) 0	(0 0 0 0 0 0)	(0 0 0 0 0 0)
'(1 2 3 4 5 6) 1	(1 2 3 4 5 6)	(1 2 3 4 5 6)
'() 5	NIL	NIL
'(1 2 3 (1 2) 4 5) 4	—	(4 8 12 (4 8) 16 20)
'(1 2 3 (1 2 (a 3)) 4 5) 6	—	(6 12 18 (6 12 (A 18)) 24 30)

6.2. Напишите функцию, которая уменьшает на 10 все числа из списка аргумента этой функции.

Реализация задания

Так как в условии задачи не сказано, является ли список-аргумент списком чисел, будем считать, что элементы списка - любые объекты:

```
;##################
;функционалы+рекурсия
;##################

(defun decrease_ten (lst)
    (mapcar (lambda (obj)
                (cond ((numberp obj) (- obj 10))
                      ((listp obj) (decrease_ten obj 10))
                      (t obj)
                )
            )
    lst)
)
```

Рис. 11: Реализация функции decrease_ten с использованием функционалов и рекурсии

Назначение параметров функций

- Функция decrease_ten по своей логике аналогична предыдущей функции multiply. Она также использует функционалы и хвстовую рекурсию.

Результаты работы

Выражение	**Результат**
'(1 2 3 4 5 6)	(-9 -8 -7 -6 -5 -4)
'()	NIL
'(1 2 3 (1 2) 4 5)	(-9 -8 -7 (-9 -8) -6 -5)
'(1 2 3 (1 2 (a 3)) 4 5)	(-9 -8 -7 (-9 -8 (A -7)) -6 -5)

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

Реализация задания

```
;##################
;функционалы
;##################

(defun reduce_or (lst); применяем or к списку
    (reduce #'(lambda (prev lst) (or prev lst)) lst)
)

(defun get_first_list (lst)
    (reduce_or (mapcar (lambda (x) (if (listp x) x nil)) lst))
)
```

Рис. 12: Реализация функции, возвращающей первый непустой список-аргумент списка-аргумента, с использованием функционалов

```
;##################
;хвостовая рекурсия
;##################

(defun get_first_list (lst)
    (cond ((null lst) nil)
          ((null (car lst)) (get_first_list (cdr lst)))
          ((listp (car lst)) (car lst))
          (t (get_first_list (cdr lst)))
    )
)
```

Рис. 13: Рекурсивная реализация функции, возвращающей первый непустой список-аргумент списка-аргумента

Назначение параметров функций

- Функция reduce_or применяет or на список lst
- Функции get_first_list возвращает первый список-аргумент списка-аргумента
- Для проверки элемента, является ли он списком или нет, используется функция listp
- В рекурсивной реализации, чтобы возвращался не пустой список, присутствует дополнительная проверка на nil (второе условие в cond)

Результаты работы

Функции, реализованные с помощью функционалов и с помощью хвостовой рекурсии выдают одинаковые результаты на одних и тех же параметрах:

Выражение	**Результат**
'(1 2 3 4 5 6)	NIL
'(1 2 3 4 5 (1 2) (3 4) 6)	(1 2)
'(1 2 3 4 NIL (1 2) 5 6)	(1 2)
'()	NIL

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

Реализация задания

Так как в условии задачи не сказано, является ли список-аргумент списком чисел, будем считать, что элементы списка - любые объекты. Ниже представлен вариант для заданных двух границ:

```
;##################
;функционалы+рекурсия
;##################

(defun get_all_numbers_between (lst less more)
    (mapcan (lambda (x)
                (cond ((null x) nil)
                      ((listp x) (get_all_numbers_between x less more))
                      ((numberp x) (if (and (> x less) (< x more)) (cons x nil) nil))
                      (t nil)
                )
            ) lst
    )
)

(defun less_and_more_numbers (lst a b)
    (let ((less (min a b))
          (more (max a b))
    )
        (get_all_numbers_between lst less more)
    )
)
```

Рис. 14: Реализация функции, выбирающей из списка числа между двумя заданными границами, с использованием функционалов и рекурсии

Назначение параметров функций

- Для реализации поставленной задачи используются функционалы и хвостовая рекурсия вместе, чтобы решить задачу наиболее оптимальным способом
- Функция less_and_more_numbers - главная функция, запускающая рекурсивную функцию less_and_more_numbers с переданным ей списком lst и переменными less, more

- Переменные less, more - минимальный и максимальные элементы из входных параметров a, b соответственно
- В функции get_all_numbers_between используется функционал mapcan, чтобы проходясь по всему списку lst, объединять полученный результат в результирующий список
- Функции listp null и numberp - функции-проверки на nil, список и число соответственно

Результаты работы

Выражение	**Результат**
'(1 2 3 4 5 6 7 8 9) 3 7	(4 5 6)
'(1 2 3 4 5 6 7 8 9) 7 3	(4 5 6)
'(1 2 3 4 5 6 7 8 9) 3.3 4.3	(4)
'(1 2 3 4 5 6 7 8 9) 3.3 3.6	NIL
'(1 2 3 4 5 6 7 8 9) 10 13	NIL
'(1 2 3 4 5 6 7 8 9) -1 2.1	(1 2)

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

Реализация задания

```
;##################
;функционалы
;##################

(defun decart (X Y)
    (mapcan #'
        (lambda (x)
            (mapcar #'
                (lambda (y) (list x y))
                Y
            )
        )
        X
    )
)
```

Рис. 15: Реализация функции, вычисляющей декартово произведение, с использованием функционалов

```
;#################
;хвостовая рекурсия
;#################


(defun decart-y (x Y result)
    (cond ((null Y) result)
          (t (decart-y x (cdr Y) (cons (list x (car Y)) result) ))
    )
)

(defun decart-x (X Y result)
    (cond ((null X) result)
          (t (decart-x (cdr X) Y (decart-y (car X) Y result)))
    )
)

(defun decart (X Y)
    (decart-x X Y nil)
)
```

Рис. 16: Рекурсивная реализация функции, вычисляющей декартово произведение, с использованием функционалов

Назначение параметров функций

- Функция decart, написанная с помощью функционалов проходится сначала по всем элементам списка X с помощью функционала mapcan и конкатенирует полученные списки в один список
- Затем, "во внутреннем цикле она проходится по всем элементам списка Y, объединяя полученные пары в списки из двух элементов
- Рекурсивная реализация использует обертку - функцию decart, запускающую рекурсивную функцию decart-x с начальным результатом nil
- В свою очередь функция decart-x запускает рекурсивную функцию decart-y для каждого элемента из списка X

Результаты работы

Функции, реализованные с помощью функционалов и с помощью хвостовой рекурсии выдают одинаковые результаты на одних и тех же параметрах:

6.6. Почему так реализовано reduce, в чем причина?

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

Выражение	**Результат**
'(1 2 3) '(a b c)	((1 A) (1 B) (1 C) (2 A) (2 B) (2 C) (3 A) (3 B) (3 C))
'(1 2 3) '(a b)	((1 A) (1 B) (2 A) (2 B) (3 A) (3 B))
'(1 2) '(a b c)	((1 A) (1 B) (1 C) (2 A) (2 B) (2 C))
'(1 2 3) '()	NIL
'() '(a b c)	NIL

Причина: *Если подпоследовательность пуста, а начальное значение не задано, то функция вызывается с нулевыми аргументами, а функция reduce возвращает то, что делает функция. Это единственный случай, когда функция вызывается не с двумя аргументами.* [1]

Таким образом, функции + и *, вызывающиеся без аргументов, возвращают 0 и 1, сответственно.

Выводы

По итогу можно сделать вывод, что рекурсивные реализации функций будут работать чуть эффективней, чем функции, использующие функционалы, так как функционалы будут тратить время на вызов дополнительных функций.

Ответы на вопросы

Способы организации повторных вычислений в Lisp:

1. Использование функционалов.
2. Использование рекурсиии.

Различные способы использования функционалов.

Функционалы:

1. Применяющие – однократное применение функции, являющейся аргументом, к остальным аргументам. Примеры применяющих функционалов: *apply*, *funcall*.
2. Отображающие – многократное применение функции, являющейся аргументом, к остальным аргументам по верхнему уровню. Примеры отображающих функционалов: *mapcar*, *reduce*, *maplist*, *mapcan*.

Что такое рекурсия?

Рекурсия – это ссылка на определяемый объект во время его определения.

Способы организации рекурсивных функций:

1. Хвостовая рекурсия.
2. Рекурсия по нескольким параметрам.
3. Дополняемая рекурсия.
4. Множественная рекурсия.

Способы повышения эффективности реализации рекурсии.

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*.

Для превращения не хвостовой рекурсии в хвостовую и в целях формирования результата (результирующего списка) на входе в рекурсию рекомендуется использовать дополнительные (рабочие) параметры. При этом становится необходимым создать функцию-оболочку для реализации очевидного обращения к функции.

Список литературы

[1] Common Lisp HyperSpec tm [Электронный ресурс]. – Режим доступа: http://clhs.lisp.se/Body/f_reduce.htm, свободный. (Дата обращения: 28.03.2020 г.)