ВВЕДЕНИЕ

Целью работы является приобретение навыков использования списков и стандартных функций Lisp.

Задачи работы: изучить способ использования списков для фиксации информации, внутреннее представление одноуровневых и структурированных списков, методы их обработки с использованием базовых функций Lisp.

1 Теоретические сведения

1.1 Способы организации повторных вычислений в Lisp

— использование функционалов

— использование рекурсии

1.2 Различные способы использования функционалов

`mapcar` – функция func применяется ко всем элементам списка, начиная с первого.

`maplist` – функция func применяется ко всем элементам списка, начиная с последнего.

`mapcan, mapcon` – аналогичны mapcar и maplist, используется память исходных данных, не работают с копиями.

`reduce` – функция func применяется каскадным образом (сначала для первого и второго элемента, потом для результата и следующего и т.д.).

1.3 Что такое рекурсия? Способы организации рекурсивных функций

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.

Способы организации рекурсивных функций

— 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.

— Возможна рекурсия по нескольким параметрам

— Дополняемая рекурсия – при обращении к рекурсивной функции используется дополнительная функция не в аргументе вызова , а вне его

— Выделяют группу функций множественной рекурсии. На одной ветке происходит сразу несколько рекурсивных вызовов. Количество условий выхода также может зависеть от задачи.

1.4 Способы повышения эффективности реализации рекурсии

— Использование хвостовой рекурсии. Если условий выхода несколько, то надо думать о порядке их следования.

— Превращение не хвостовой рекурсии в хвостовую. 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 создат фунецию – оболочку для реализации очевидного обращения к функции.

2 Практическая часть

2.1 Задание №1

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

```
(defun set-equal (lst1 lst2)
    (and (subsetp lst1 lst2) (subsetp lst2 lst1))
```

2.2 Задание №2

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

Создаём точеченые пары:

```
(setq cities '(Berlin Paris Moscow London))
(setq countries '(Germany France Russia England))

(defun set_points (a b)
  (cons a b)
)

(setq points (mapcar #'set_points countries cities))
```

Рекурсивный поиск:

```
(defun found_country (city lst)
  (cond ((null lst) nil)
        ((eql city (cdr (car lst))) (caar lst))
        (t (found_country city (cdr lst)))
  )
)

(defun found_city (country lst)
  (cond ((null lst) nil)
        ((eql country (car (car lst))) (cdr (car lst)))
        (t (found_city country (cdr lst)))
  )
)
```

Поиск с использование функционалов:

```
(defun found_country_func (city lst)
  (defun found (lst1 lst2)
    (if (consp lst1)
      (or (if (eql city (cdr lst1)) (car lst1) Nil)
          (if (eql city (cdr lst2)) (car lst2) Nil) )
      (or lst1 (if (eql city (cdr lst2)) (car lst2) Nil) )
    )
  )
  (reduce #'found newPoints)
)

(defun found_city_func (country lst)
  (defun found (lst1 lst2)
    (if (consp lst1)
      (or (if (eql country (car lst1)) (cdr lst1) Nil)
          (if (eql country (car lst2)) (cdr lst2) Nil) )
      (or lst1 (if (eql country (car lst2))(cdr lst2) Nil))
    )
  )
  (reduce #'found newPoints)
)
```

2.3 Задание №3

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

а) все элементы списка — числа,

б) элементы списка – любые объекты.

С использованием рекурсии:

```
(defun mul_num_rec (lst k)
  (if lst
    (cons (* (car lst) k) (mul_num_rec (cdr lst) k))
  )
)

(defun mul_all_rec (lst k)
  (if lst
    (cons
      (if (numberp (car lst)) (* (car lst) k) (car lst))
      (mul_all_rec (cdr lst) k)
```

```
12      )
13    )
14  )
```

С использованием функционалов:

```
1  (defun mul_num (lst k)
2    (mapcar #'(lambda (x) (* x k)) lst)
3  )
4
5  (defun mul_all (lst k)
6    (mapcar #'(lambda (x) (if (numberp x) (* x k) x)) lst)
7  )
```

2.4 Задание №4

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

С использованием рекурсии:

```
1  (defun minus_ten_rec (lst)
2      (cond
3          ((null lst) Nil)
4          ('T (cons (- (car lst) 10) (minus_ten_car (cdr lst))))
5      )
6  )
```

С использованием функционалов:

```
1  (defun minus_ten_car (lst)
2      (mapcar #'(lambda (el) (- el 10)) lst)
3  )
```

2.5 Задание №5

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

С использованием рекурсии:

```
1  (defun first_list_rec (lst)
2      (cond
3          ((null lst) Nil)
```

```
4           ((not (atom (car lst))) (car lst))
5           ('T (first_list_rec (cdr lst)))
6       )
7   )
```

С использованием функционалов:

```
1   (defun first_list_map (lst)
2       (reduce #'
3           (lambda (el1 el2)
4               (or
5                   (and (not (atom el1)) el1)
6                   (and (not (atom el2)) el2)
7               )
8           ) lst
9       )
10  )
```

2.6 Задание №6

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

С использованием рекурсии:

```
1   (defun found_between_rec (a b lst)
2     (if lst
3       (append
4         (if (and (< a (car lst)) (< (car lst) b))
5           (list (car lst)) Nil
6         )
7         (found_between_rec a b (cdr lst))
8       )
9     )
10  )
```

С использованием функционалов:

```
1   (defun found_between (a b lst)
2     (defun found (lst1 lst2)
3       (if (numberp lst1)
4         (append(if(and(< a lst1)(< lst1 b))(list lst1) Nil)
5                (if(and(< a lst2)(< lst2 b))(list lst2) Nil))
6         (append lst1
```

```
7                 (if(and(< a lst2)(< lst2 b))(list lst2) Nil))
8       )
9     )
10    (reduce #'found lst)
11  )
```

2.7 Задание №7

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.)

С использованием рекурсии:

```
1  (defun decart (el lst)
2      (cond
3          ((null lst) Nil)
4          ('T (cons (cons el (car lst)) (decart_one_element el (cdr lst))))
5      )
6  )
7
8  (defun decart_rec (lst1 lst2)
9      (cond
10         ((null lst1) Nil)
11         ('T (append (decart (car lst1) lst2) (decart_rec (cdr lst1) lst2)))
12     )
13 )
```

С использованием функционалов:

```
1  (defun decart_map (lst1 lst2)
2      (mapcan #'
3          (lambda (x)
4              (mapcar #'(lambda (y) (cons x y)) lst2)
5          ) lst1
6      )
7  )
```

2.8 Задание №8

Почему так реализовано reduce, в чем причина?

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

Результатом функции будет значение по умолчанию (то есть 0), так как список, к которому применяется функция является пустым.