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 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.

1.3 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

Пусть list-of-list список, состоящий из списков. Написать функцию, которая вычисляет сумму длин всех элементов list-of-list. Например для аргумента ((1 2) (3 4)) -> 4.

```
1 (defun len_lists (lst)
2     (cond
3         ((null lst) 0)
4         (t (+ (length (car lst))
5               (len_lists (cdr lst))))
6     )
7 )
```

2.2 Задание №2

Написать рекурсивную версию (с именем reg-add) вычисления суммы чисел заданного списка. Например: (reg-add (2 4 6)) -> 12

```
1 (defun reg-add (lst)
2     (cond
3         ((null lst) 0)
4         (t (+ (car lst) (reg-add (cdr lst))))
5     )
6 )
```

2.3 Задание №3

Написать рекурсивную версию с именем recnth функции nth.

```
1 (defun recnth (n lst)
2     (cond
3         ((or (= 0 n)
4              (null lst)) (car lst))
5         (t (recnth (- n 1) (cdr lst)))
6     )
7 )
```

2.4 Задание №4

Написать рекурсивную функцию alloddr, которая возвращает t когда все элементы списка нечетные.

```
1 (defun alloddr (lst)
2     (cond
3         ((null lst) t)
4         ((eql (mod (car lst) 2) 0) nil)
5         (t (alloddr (cdr lst)))
6     )
7 )
```

2.5 Задание №5

Написать рекурсивную функцию, относящуюся к хвостовой рекурсии с одним тестом завершения, которая возвращает последний элемент списка - аргументы.

```
1 (defun latest (lst)
2     (cond
3         ((null lst) nil)
4         ((eql (length lst) 1) (car lst))
5         (t (latest (cdr lst)))
6     )
7 )
```

2.6 Задание №6

Написать рекурсивную функцию, относящуюся к дополняемой рекурсии с одним тестом завершения, которая вычисляет сумму всех чисел от 0 до n-ого аргумента функции. Вариант:

а) от n-аргумента функции до последнего >= 0,

б) от n-аргумента функции до m-аргумента с шагом d.

Дополняющая рекурсия, считающая сумму всех чисел от 0 до n-го аргумента.

```
1 (defun sum_first(lst n)
2   (cond
```

```
3       ((= n (length lst)) (car lst))
4       (t (+ (car lst) (sum_first (cdr lst) n)))
5     )
6   )
7
8   (defun sum_first_n(lst n)
9     (sum_first lst (+ 1 (- (length lst) n)))
10  )
```

Дополняющая рекурсия, считающая сумму всех чисел от n-го аргумента до конца.

```
1   (defun sum_last(lst n)
2     (cond
3       ((= 1 (length lst)) (car lst))
4       ((>= n (length lst)) (+ (car lst)
5           (sum_last (cdr lst) n))
6       )
7       (t (sum_last (cdr lst) n))
8     )
9   )
10
11  (defun sum_last_n(lst n)
12    (sum_last lst (+ 1 (- (length lst) n)))
13  )
```

Дополняющая рекурсия, считающая сумму всех чисел от n-го аргумента до m-го с шагом d.

```
1   (defun sum_range(lst n m d)
2     (let ((len (length lst)))
3       (cond
4         ((= m len)
5           (if (= 0 (rem (- n m) 3))
6             (car lst)
7             0
8           )
9         )
10        ((and (>= n len) (= 0 (rem (- n len) 3)))
11          (+ (car lst) (sum_range (cdr lst) n m d))
12        )
13        (t
14          (sum_range (cdr lst) n m d)
15        )
16      )
17    )
```

```
18 )
19
20 (defun sum_range_nmd(lst n m d)
21   (let ((len (length lst)))
22     (sum_range lst (+ 1 (- len n)) (+ 1 (- len m)) d)
23   )
24 )
```

2.7 Задание №7

Написать рекурсивную функцию, которая возвращает последнее нечетное число из числового списка, возможно создавая некоторые вспомогательные функции.

```
1  (defun last_odd_helper (lst lodd)
2    (cond ((null lst) lodd)
3          (t (last_odd_helper (cdr lst)
4                              (cond ((oddp (car lst)) (car lst))
5                                    (t lodd))))
6    )
7  )
8
9  (defun last_odd (lst)
10     (last_odd_helper lst nil)
11 )
```

2.8 Задание №8

Используя cons-дополняемую рекурсию с одним тестом завершения, написать функцию которая получает как аргумент список чисел, а возвращает список квадратов этих чисел в том же порядке.

```
1  (defun square_helper (lst)
2      (cons (* (car lst) (car lst))
3            (if (> (length (cdr lst)) 0)
4                (square_helper (cdr lst))
5                nil
6            )
7      )
8  )
9
10 (defun square (lst)
```

```
11      (if (null lst)
12          nil
13          (square_helper lst)
14      )
15  )
```

2.9 Задание №9

Написать функцию с именем select-odd, которая из заданного списка выбирает все нечетные числа.

а) Вариант 1: select-even,

б) Вариант 2: вычисляет сумму всех нечетных чисел(sum-all-odd) или сумму всех четных чисел (sum-all-even) из заданного списка.)

```
1  (defun filter (predicate lst)
2    (cond ((null lst) nil)
3          (t (cond ((funcall predicate (car lst))
4                    (cons (car lst) (filter predicate (cdr lst))))
5                   (t (filter predicate (cdr lst)))
6             )
7          )
8    )
9  )
10
11 (defun select_odd (nums)
12   (filter #'oddp nums)
13 )
14
15 (defun select_even (nums)
16   (filter #'evenp nums)
17 )
18
19 (defun sum_all_odd (nums)
20   (reduce #'+ (select_odd nums))
21 )
22
23 (defun sum_all_even (nums)
24   (reduce #'+ (select_even nums))
25 )
```