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»

Отчет по лабораторной работе №6 по курсу "Функциональное и логическое программирование"

Тема Рекурсивные функции

Студент Гурова Н.А.

Группа ИУ7-64Б

Оценка (баллы)

Преподаватель Толпинская Н.Б., Строганов Ю.В.

Москва — 2023 г.

1 Теоретические вопросы

Рекурсия — это ссылка на определяемый объект во время его определения. Т.к. в Lisp используются рекурсивно определенные структуры, то рекурсия — это естественный принцип обработки таких структур. Существуют типы рекурсивных функций: хвостовая, дополняемая, множественная, взаимная рекурсия и рекурсия более высокого порядка.

1.1 Хвостовая рекурсия

Частный случай рекурсии, при котором любой рекурсивный вызов является последней операцией перед возвратом из функции.

```
1    ; Пример хвостовой рекурсии для нахождения
2    ; суммы элементов конечного списка
3
4    (defun sum_tail (lst acc)
5      (if (null lst)
6          acc
7          (sum-tail (cdr lst) (+ acc (car lst)))))
8
9    (print (sum_tail '(1 2 3 4 5) 0))                ; 15
```

1.2 Множественная рекурсия

Рекурсия, содержащая несколько самоссылок.

1.3 Рекурсия более высокого порядка

Рекурсия более высокого порядка — это когда функция вызывает саму себя, но передает в качестве аргумента другую функцию.

1.4 Взаимная рекурсия

Вид рекурсии, когда два математических или программных объекта, таких как функции или типы данных, определяются в терминах друг друга.

```
1     ; Пример взаимной рекурсии для определения четности числа
2     (
3         defun is-even (n)
4             (if
5                 (= n 0)
6                 t
7                 (if
8                     (= n 1)
9                     nil
10                    (is-odd (- n 1))
11                )
12            )
13    )
14
15    (
16        defun is-odd (n)
17            (if
18                (= n 0)
19                nil
20                (if
21                    (= n 1)
22                    t
23                    (is-even (- n 1))
24                )
25            )
26    )
27
28    (print (is-odd 1))      ; T
29    (print (is-odd 2))      ; NIL
30    (print (is-even 2))     ; T
```

1.5 Дополняемая рекурсия

Форма рекурсии, при которой вызов рекурсивной функции происходит после выполнения последней операции в теле функции. Обычно это означает, что результат предыдущего вызова рекурсивной функции используется для вычисления результата текущего вызова.

```
1    ; Пример дополняемое рекурсии для определения факториала числа
2    (
3        defun factorial (n &optional (acc 1))
4            (if
5                (<= n 1)
6                acc
7                (factorial (- n 1) (* n acc))
8            )
9    )
10
11   (print (factorial 5))           ; 120
```

2 Практические задания

2.1 Написать хвостовую рекурсивную функцию my-reverse, которая развернет верхний уровень своего списка-аргумента lst

```
1   ; С помощью append
2   (
3       defun my_reverse1 (lst)
4           (if
5               (null lst)
6               nil
7               (append (my_reverse1 (cdr lst)) (list (car lst)))
8           )
9   )
10
11  (print (my_reverse1 '()))               ; NIL
12  (print (my_reverse1 '(1 2 3 4 5)))      ; (5 4 3 2 1)
13  (print (my_reverse1 '(1 (2 3) 4)))      ; (4 (2 3) 1)
14
15
16  ; Без append
17  (
18      defun my_reverse2 (lst &optional (result ()))
19          (if
20              (null lst)
21              result
22              (my_reverse2 (cdr lst) (cons (car lst) result))
23          )
24  )
25
26  (print (my_reverse2 '()))               ; NIL
27  (print (my_reverse2 '(1 2 3 4 5)))      ; (5 4 3 2 1)
28  (print (my_reverse2 '(1 (2 3) 4)))      ; (4 (2 3) 1)
```

2.2 Написать функцию, которая возвращает первый элемент списка - аргумента, который сам является непустым списком

```
1   (
2       defun return_first_list (lst)
3           (cond
4               (
5                   (null lst)
6                   nil
7               )
8               (
9                   (and (not (atom (car lst))) (not (null (car lst))))
10                  (car lst)
11              )
12              (
13                  t
14                  (return_first_list (cdr lst))
15              )
16          )
17  )
18
19  (print (return_first_list '(1 ())))                ; nil
20  (print (return_first_list '(1 2 3 4 5)))           ; nil
21  (print (return_first_list '((1 2) 3 4)))           ; (1 2)
22  (print (return_first_list '(1 2 (3 4))))           ; (3 4)
23  (print (return_first_list '(1 (2) (3 4))))         ; (2)
24  (print (return_first_list '(1 () (3 4))))          ; (3 4)
```

2.3 Напишите рекурсивную функцию, которая умножает на заданное число-аргумент все числа из заданного списка-аргумента, когда все элементы списка — числа, элементы списка – любые объекты

```
1   ; a
2   (
```

```
3         defun a_multiply_all (lst x)
4             (if
5                 (null lst)
6                 nil
7                 (cons (* x (car lst)) (a_multiply_all (cdr lst) x))
8             )
9     )
10
11    (print (a_multiply_all '() 2))                    ; nil
12    (print (a_multiply_all '(1 2 3 4 5) 2))           ; (2 4 6 8 10)
13
14
15    ; b
16    (
17        defun b_multiply_all (lst x)
18            (cond
19                ((null lst) nil)
20                ((not (numberp (car lst))) (cons (car lst)
                     (b_multiply_all (cdr lst) x)) )
21                (t (cons (* x (car lst)) (b_multiply_all (cdr lst) x)))
22            )
23    )
24
25    (print (b_multiply_all '() 2))                    ; nil
26    (print (b_multiply_all '(1 2 3 4 5) 2))           ; (2 4 6 8 10)
27    (print (b_multiply_all '(1 (2) 3 4 5) 2))         ; (2 (2) 6 8 10)
28    (print (b_multiply_all '(1 (2) 3 "1" 5) 2))       ; (2 (2) 6 "1"
         10)
```

2.4 Напишите функцию, select-between, которая из списка-аргумента, содержащего только числа, выбирает только те, которые расположены между двумя указанными границами-аргументами и возвращает их в виде списка

```
1     (
2         defun get_numbers (lst start end result flag)
3             (cond
```

```
4                   ((null lst) nil)
5                   ((= start (car lst)) (cons (car lst) (get_numbers (cdr
                        lst) start end result 1)))
6                   ((= end (car lst)) (cons (car lst) (get_numbers (cdr
                        lst) start end result 0)))
7                   ((= 1 flag) (cons (car lst) (get_numbers (cdr lst)
                        start end result 1)))
8                   ((= 0 flag) (get_numbers (cdr lst) start end result 0))
9               )
10      )
11
12      (
13          defun select_between (lst start end)
14              (get_numbers lst start end () 0)
15      )
16
17      ; Сортировка
18      (
19          defun put_elem_in_result (x result)
20              (cond
21                  ((null result) (cons x result))
22                  ((<= x (car result)) (cons x result))
23                  (t (cons (car result) (put_elem_in_result x (cdr
                        result))))
24              )
25      )
26
27      (
28          defun my_sort (lst &optional (result ()))
29              (if
30                  (null lst)
31                  result
32                  (my_sort (cdr lst) (put_elem_in_result (car lst)
                        result))
33              )
34      )
35
36      ; (print (my_sort '(5 4 3 2 1)))       ; (1 2 3 4 5)
37      ; (print (my_sort '(5 14 3 12 1)))     ; (1 3 5 12 14)
```

```
38
39      (select_between '(1 2 3 4 5 6) 1 6)                ; (1 2 3 4 5 6)
40      (select_between '(1 2 3 4 5 6) 2 4)                ; (2 3 4)
41      (select_between '(10 2 5 4 1 9) 2 1)               ; (2 5 4 1)
42
43      (my_sort (select_between '(10 2 5 4 1 9) 2 1))     ; (2 5 4 1)
44      (my_sort (select_between '(10 23 -5 4 1 9) -5 9)) ; (-5 1 4 9)
```

2.5 Написать рекурсивную версию (с именем rec-add) вычисления суммы чисел заданного списка одноуровнего смешанного, структурированного

```
1       (
2           defun a_rec_add (lst &optional (result 0))
3               (cond
4                   ((null lst) result)
5                   ((numberp (car lst)) (a_rec_add (cdr lst) (+ result
                        (car lst))))
6                   (t (a_rec_add (cdr lst) result))
7               )
8       )
9
10      (print (a_rec_add '()))                 ; 0
11      (print (a_rec_add '(1 2 3 4 5)))        ; 15
12      (print (a_rec_add '(1 (2) 3 4 5)))      ; 13
13
14
15      (
16          defun b_rec_add (lst &optional (result 0))
17              (cond
18                  ((null lst) result)
19                  ((numberp (car lst)) (b_rec_add (cdr lst) (+ result
                        (car lst))))
20                  ((listp (car lst)) (+ (b_rec_add (car lst) 0)
                        (b_rec_add (cdr lst) result)))
21                  (t (a_rec_add (cdr lst) result))
22              )
```

```
23      )
24
25      (print (b_rec_add '()))                    ; 0
26      (print (b_rec_add '(1 2 3 4 5)))           ; 15
27      (print (b_rec_add '(1 (2) 3 4 5)))         ; 16
28      (print (b_rec_add '(1 (4) 3 4 5)))         ; 18
29      (print (b_rec_add '(1 (4) 3 "4" 5)))       ; 14
30      (print (b_rec_add '(1 (4 3) 3)))           ; 11
31      (print (b_rec_add '(1 (4 (3 5)) 3)))       ; 16
```