**2 лаба**

**1.** Добавляющую заданное параметром x число в упорядоченный по неубыванию числовой список таким образом, чтобы сохранилась упорядоченность. Сортировку не использовать! Например, x=7, L=(0 3 3 6 9) –> (0 3 3 6 7 9).

(defun func (n lst)

(cond

((null lst) (list n))

((>= (car lst) n) (cons n lst))

(t (cons (car lst) (func n (cdr lst))))

)

)

Вызов:

(func 5 '(3 4 6))

Этот код представляет собой определение функции Lisp с именем `func`, которая принимает два аргумента: `n` и `lst`. Эта функция выполняет следующие действия:

1. Внутри функции определено условное выражение `cond`, которое оценивает несколько вариантов в зависимости от условий.

2. Первое условие: `((null lst) (list n))` проверяет, является ли `lst` пустым списком (пустой список обозначается как `nil` или `()` в Lisp). Если `lst` пуст, то функция возвращает список, содержащий только элемент `n`.

3. Второе условие: `((>= (car lst) n) (cons n lst))` проверяет, больше или равно ли первый элемент `(car lst)`значению `n`. Если это верно, то функция возвращает новый список, в котором `n` добавлен в начало `lst` с помощью функции `cons`.

4. Последнее условие `(t (cons (car lst) (func n (cdr lst))))` выполняется, если ни одно из предыдущих условий не истинно. В этом случае функция берет первый элемент `lst` и добавляет его в начало результата вызова функции `func` с `n` и "хвостом" списка `lst`, который получается удалением первого элемента с помощью функции `(cdr lst)`.

Таким образом, эта функция рекурсивно обрабатывает список `lst`, добавляя элемент `n` в правильное место в упорядоченном порядке, сохраняя при этом порядок элементов в `lst`.

**2.** Удаляющую элементы с четными номерами из списка (нумерация элементов должна начинаться с 1). Для проверки на четность можно воспользоваться предикатом EVENP или функцией нахождения остатка от деления REM. Например, (-2 6 s -1 4 f 0 z x r) –> (-2 s 4 0 x).

(defun func4 (n)

(cond

((null n) n)

(t (cons (car n) (func4 (cddr n))))

)

)

Вызов:  
(func4 '(1 2 3 4 5 6))

Функция `func4` принимает один аргумент `n` и выполняет следующее действие:

1. Внутри функции определено условное выражение `cond`, которое оценивает два варианта в зависимости от условий.

2. Первое условие `((null n) n)` проверяет, является ли список `n` пустым (пустой список обозначается как `nil` или `()` в Lisp). Если `n` пуст, то функция возвращает сам `n`.

3. Второе условие `(t (cons (car n) (func4 (cddr n))))` выполняется, если `n` не пусто. В этом случае функция создает новый список, в котором первый элемент `n` (`(car n)`) добавлен в начало, а затем вызывает саму себя для обработки "хвоста" `n`, который получается удалением первых двух элементов с помощью функции `(cddr n)`.

Таким образом, функция `func4` извлекает из списка `n` элементы с четными индексами и возвращает новый список, содержащий только эти элементы.

**3.** Вычисляющую глубину списка. Например, ((((1))) 2 (3 4)) –> 4

(defun func (n)

(cond

((null n) 0)

((atom n) 0) %??????????????????????????

(t (1+ (max (func (car n)) (- (func (cdr n)) 1))))

)

)

Вызов:

(func '((((1))) 2 (3 4)))

Данный код представляет собой функцию **func**, которая принимает аргумент **n**, предполагая, что **n** - это некоторая структура данных, возможно, список. Эта функция вычисляет "глубину" структуры **n**.

В данной функции используется рекурсивный подход для определения глубины структуры **n**. Она работает следующим образом:

1. Если **n** является пустым списком (null n), то функция возвращает 0. Это базовый случай рекурсии, когда достигнут конец структуры.
2. Если **n** является атомом (atom n), то функция возвращает 1. Это случай, когда **n** не является списком, и его глубина считается равной 1.
3. В противном случае (если **n** не пусто и не атом), функция рекурсивно вызывает себя для головы списка **(car n)** и хвоста списка **(cdr n)**. Затем она вычисляет максимум между результатом вызова **func** для головы и разностью результата вызова **func** для хвоста и 1, и увеличивает этот результат на 1. Это позволяет вычислить максимальную глубину структуры **n**, учитывая глубины её подструктур.

Таким образом, функция **func** рекурсивно проходит по структуре **n** и возвращает её максимальную глубину.

Начало формы