Задание №12
Сложные списочные структуры

1. Общая постановка задачи

Опишите функцию, выполняющую обработку, описанную в задании с номером вашего варианта.

Приведите набор тестовых вызовов описанной функции, демонстрирующих все варианты ее работы.

Опишите программу в текстовом файле с именем `task12-NN.lsp`, где `NN` — номер вашего варианта. Полученный файл загрузите на портал в качестве выполненного задания.

2. Предварительные замечания

Если в задании используется понятие «номер уровня», то следует считать, что уровни нумеруются с единицы (элементы заданного списка — элементы первого уровня).

Следует считать, что элементы списка нумеруются с единицы.

Понятие «подсписок» означает элемент списка, являющийся списком. Понятия «часть списка» и «фрагмент списка» означают последовательность подряд идущих элементов списка или подсписка на любом из уровней списка.

Не следует делать предположений на счет задания, не сформулированных явно в условии. Если возникают сомнения — задайте вопрос на форуме «Язык Lisp».

3. Пример выполнения задания

ЗАДАНИЕ: Описать функцию `number-depth` аргумента `l`, где аргумент — списочная структура с произвольными элементами. Вызов `(number-depth l)` должен возвращать максимальный номер уровня списка, на котором в списке встречается число. В случае отсутствия чисел в списке, результат функции — 0. Например, вызов

```
(number-depth '(5 (7 8 a) ((6 (b () 9) 7) s) b))
```

должен возвратить число 4.

РЕШЕНИЕ:
Содержимое файла `task12-NN.lsp`:

```
;; Стратегия решения следующая. Будем просматривать элементы списка
;; по уровням: сначала все эелементы первого уровня, потом все элементы
;; второго и т. д. Элементы следующего уровня можем получить
;; объединив все элементы-списки текущего уровня.
;; Решение - функция от одного аргумента, но чтобы не определять
;; вспомогательной функции, определим дополнительные
;; необязательные параметры
;; l - обязательный параметр - заданный список
;; остальные параметры необязательные
;; new-l - список элементов следующего уровня
;; cur-depth-num - номер текущего уровня; первоначально равен 1
;; cnt - количество элементов текущего уровня, являющихся списками
;; res - номер последнего уровня, на котором обнаружено число
(defun number-depth (l &optional new-l (cur-depth-num 1) (cnt 0) (res 0))
  (if (null l)           ; если список l стал пустым
      (if (= cnt 0) res  ; и на текущем уровне не было подсписков,
                         ; значит вернуть результат
          ; если же подсписки были, то есть еще один уровень, элементы которого
          ; составляют список new-l, от которого рекурсивно запускаем функцию
          (number-depth new-l () (+ cur-depth-num 1) 0 res))
      ; если список l пока не пустой, то в нем есть первый элемент,
      ; обозначим его el
      (let ((el (car l)))
        (cond
          ((numberp el) ; если el - число, то номер текущего уровня передаем
                        ; в качестве параметра res при рекурсивном вызове
             (number-depth (cdr l)
                           new-l
                           cur-depth-num
                           cnt
                           cur-depth-num))
          ((listp el) ; если el - список, то его элементы добавляем
                      ; к элементам следующего уровня
             (number-depth (cdr l)
                           (append el new-l)
                           cur-depth-num
                           (+ 1 cnt)
                           res))
          ; в противном случае просто игнорируем элемент el
          (T (number-depth (cdr l) new-l cur-depth-num cnt res))))))

(print (number-depth '(5 (7 8 a) ((6 (b () 9) 7) s) b)))
(print (number-depth '(a b (c ((d e) f 5 (g) h) i (j (k () l)))
                       (m (n ((o p (9))) q ((r) (s t))) (u ((v) w)))
                       ((x) ()) ((((10 (() y)))))))))
(print (number-depth '(() 6)))
(print (number-depth '()))
```

Файлы с примерами можно загрузить с портала.

4. Варианты заданий

1. Описать функцию `find-list-part` двух аргументов `l1` и `l2`, где оба аргумента — списочные структуры с произвольными элементами. Вызов `(find-list-part l1 l2)` должен возвращать список, в котором каждый элемент — путь к вхождению `l1` в качестве фрагмента в `l2`: список позиций на уровнях, приводящий к вхождению фрагмента. Например, вызов

```
(find-list-part '(a a (b) a)
                '(c a a (b) a a (b) a (a d a a (b a a (b a a (b) a z) a h a a (b) a) a)))
```

должен возвратить

```
((2) (5) (9 5 4 2) (9 5 7))
```

2. Описать функцию `del-sublist` двух аргументов `l` и `n`, где первый аргумент список, а второй — число. Вызов `(del-sublist l n)` должен возвращать список, полученный из `l` удалением на уровне `n` всех элементов, являющихся списками. Например, вызов

```
(del-sublist '(c a (a ((b) a a (b)) a) (a d (a a) (b a a (b a a (b) a z) a h a a (b) a) a)) 2)
```

должен возвратить

```
(c a (a a) (a d a))
```

3. Описать функцию `insert-elem` четырех аргументов `l`, `n`, `m` и , `a`, где первый аргумент список, `n` — номер уровня, `m` — номер позиции. Вызов `(insert-elem l n m a)` должен возвращать список, полученный из `l` вставкой элемента `a` в позиции `m` в каждом подсписке на уровне `n`. Если в каком то подсписке заданного уровня отсутствовало заданное количество позиций, то заданный элемент должен быть поставлен на последнюю позицию. Например, вызов

```
(insert-elem '(c (a ((b) a (b)) a) (a d (a) (b (b a (b) a z) a h a a (b) a) a)) 3 3 'new)
```

должен возвратить

```
(c (a ((b) a new (b)) a) (a d (a new) (b (b a (b) a z) new a h a a (b) a) a))
```

4. Описать функцию `del-elem` трех аргументов `l`, `n`, `m`, где первый аргумент список, `n` — номер уровня, `m` — номер позиции. Вызов `(del-elem l n m)` должен возвращать список, полученный из `l` удалением элемента в позиции `m` в каждом подсписке на уровне `n`. Например, вызов

```
(del-elem '(c (a ((b) a (b)) a) (a d (a) (b (b a (b) a z) a h a a (b) a) a)) 3 3)
```

должен возвратить

```
(c (a ((b) a) a) (a d (a) (b (b a (b) a z) h a a (b) a) a))
```

5. Описать функцию `list-statistics` одного списочного аргумента `l`. Функция должна привести статистику — сколько и каких элементов на каждом уровне вложенности заданного списка. Элементы подсчитываются по позициям: атомы, списки, числа, идентификаторы. Результат должен выдаваться в виде
`((ATOM` l_1`) (LIST` l_2`) (NUMBER` l_3`) (SYMBOL` l_4`))`.
Здесь l_i — списки пар `(`N `.` K`)`, где N — номер уровня, K — количество соответствующих элементов на данном уровне. Например, вызов

```
(list-statistics '(b 12 (c (((+ 34 63) (cons 25))) ((cons) 8 (list (2 () 7) 9)))))
```

должен возвратить

```
((ATOM ((1 . 2) (2 . 1) (3 . 1) (4 . 3) (5 . 8) (9 . 0)))
 (LIST ((1 . 1) (2 . 2) (3 . 3) (4 . 3) (5 . 1) (9 . 0)))
 (NUMBER ((1 . 1) (2 . 0) (3 . 1) (4 . 1) (5 . 5) (9 . 0)))
 (SYMBOL ((1 . 1) (2 . 1) (3 . 0) (4 . 2) (5 . 3) (9 . 0))))
```

6 (бонус 20%)**.** Описать функцию `evaluate-tree` с одним параметром `l`. Аргумент `l` — список произвольной вложенности. Вызов `(evaluate-tree l)` должен в списке `l` заменить все подсписки, которые являются корректно построенными выражениями языка Lisp и которые можно вычислить, на результат их вычисления. Например, вызов

```
(evaluate-tree '(b 12 (c ((+ 34 63) (cons 25)) (cons 8 (list 2 7 9)))))
```

должен возвратить

```
(b 12 (c (97 (cons 25)) (8 2 7 9)))
```

Для выполнения задания потребуется использование вызовов

```
(handler-case expr (T () 'no-answer))
```

где `expr` — выражение Lisp, которое необходимо вычислить. Результат вызова — результат `expr`, если он завершается без ошибки и атом `no-answer`, если вычисление `expr` завершается ошибкой.

7 (бонус 30%)**.** Описать функцию `find-pattern` двух аргументов `l` и `p`, где оба аргумента — списочные структуры. Элементы списка `l` — произвольные. Список `p` представляет собой шаблон искомой части списка `p`. `p` может содержать подсписки любой вложенности и, в качестве атомарных элементов, имена функций `atom`, `listp`, `numberp`, `consp`, `symbolp`. Вызов `(find-pattern l p)` должен возвращать непрерывную часть списка `l`, удовлетворяющую шаблону `p`: списочные структуры должны совпадать, а на позициях имен функций в `p` должны содержаться элементы, для которых данные функции выдают `T`. Например, вызов

```
(find-pattern '(a () (b 45 (((a 15 ((c))) b) 33 15) 18) d)
              '(numberp ((listp symbolp) atom)))
```

должен возвратить

```
(45 (((a 15 ((c))) b) 33))
```

8 (бонус 30%)**.** Описать функцию `list-corrections` двух аргументов `l` и `cmd`. Аргумент `l` — список произвольной вложенности. Аргумент `cmd` — команды корректуры: список из подсписков, в котором каждый подсписок представляется в виде
$(level\ \ predicate\ \ command)$.
Здесь $level$ — номер уровня вложенности в список `l`; $predicate$ — имя предиката для определения корректируемого элемента: имя любого из предикатов `atom`, `listp`, `numberp`, `consp`, `symbolp`; $command$ — любая из команд: `duplicate`, `delete`, `wrap`.

Команда `duplicate` означает, что заданный элемент нужно продублировать, команда `delete` — удалить, `wrap` — обернуть в список. Функция должна последовательно выполнить команды корректуры в порядке обхода списочной структуры в глубину. Например, вызов

```
(list-corrections '(b 12 (c (((+ 34 63) (cons 25))) ((cons) 8 (list (2 () 7) 9))))
                  '((1 atom delete) (2 atom duplicate) (5 numberp wrap)
                    (3 listp delete) (4 numberp wrap)))
```

должен возвратить

```
(12 (c c (((+ (34) 63) (cons 25))) (8 (list (2 () 7) (9)))))
```

9 (бонус 30%)**.** Описать функцию `arg-bounding` двух аргументов `args-formal` и `args-fact` сопоставляющую каждому элементу списка формальных параметров `args-formal` значение. Аргумент `args-formal` — список произвольной вложенности моделирует описание списка параметров макроса на Lisp. Символьные атомы (элементы типа SYMBOL) в списке `args-formal` — имена параметров. Имена параметров могут содержаться на любом уровне вложенности списков. Если в списке параметров встречается атом `opt-args`, то это значит, что последующие параметры являются необязательными. Для необязательных параметров может быть указано значение по умолчанию — атом или список (без апострофов), объединенный с именем параметра в единый список (в котором сначала идет имя параметра, а затем значение по умолчанию). Если в списке параметров встречается атом `rest-arg`, то это значит, что далее может следовать только один параметр, принимающий и объединяющий в едином списке все фактические параметры, переданные сверх количества обязательных и необязательных параметров на данном уровне вложенности списка параметров. Вложенные подсписки параметров могут встречаться только среди обязательных параметров данного уровня и не могут следовать среди или после необязательных параметров. Список `args-fact` должен представлять список передаваемых фактических параметров, организованный в той же структуре, что и список формальных параметров.

Функция `arg-bounding` должна ставить в соответствие каждому формальному параметру фактический параметр. В случае, если список `args-formal` не удовлетворяет правилам оформления списка параметров или если невозможно сопоставить фактические параметры формальным, результатом должен быть пустой список `NIL`. Иначе, результатом должен быть список пар, сопоставляющих формальные и фактические параметры. В случае отсутствия значения для необязательного параметра, ему должно быть назначено значение по умолчанию. Если оно не указано, то его значением должен быть пустой список. Например, вызов

```
(arg-bounding '(a (b (c d opt-args e (f (25 13)) rest-arg g) h rest-arg i) opt-args (j 7) rest-arg k)
              '((3 5) (47 ((a 15) 19 16) 27 38 10 11) 27 3 5))
```

должен возвратить

```
((a (3 5)) (b 47) (c (a 15)) (d 19) (e 16) (f (25 13)) (g NIL) (h 27) (i (38 10 11)) (j 27) (k (3 5)))
```