Результирующий список – список атомов без повторений, т.е. множество.

Как и в задаче выравнивания списка, `Collect` использует вспомогательную функцию с накапливающим параметром `Res`, но добавление атома в `Res` происходит только в том случае, когда этого атома там нет:

```
(defun Collect (X) (Col X nil))
(defun Col (Y Res) ;вспомогательная функция
    (cond ((null Y) Res)
          ((atom Y) (cond ((member Y Res)Res)
                          (T(cons Y Res))))
          (T (Col (car Y) (Col (cdr Y) Res)) ) ))
```

2.6. Задачи на программирование рекурсии

Простая рекурсия

1. Составить функцию `(RemoveLast L)`, удаляющую из списка последний элемент. Например:
   `(RemoveLast '(A (S D) E (Q))) => (A (S D) E)`
2. Определить функцию-предикат `(OneLevel L)`, которая проверяет, является ли список-аргумент одноуровневым списком:
   `(OneLevel '(A B C)) => T,`
   `(OneLevel '((A) B C)) => NIL`
3. Запрограммировать функцию `(Bubl N A)` с двумя вычисляемыми аргументами – числом `N` и атомом `A`. Функция строит список глубины `N`; на самом глубоком уровне элементом списка является `A`, а на любом другом уровне список состоит из одного элемента. Например: `(Bubl 3 5)=>(((5))).`
4. Определить функцию `(LastAtom L)`, выбирающую последний от начала списка (невзирая на скобки) атом списка:
   `(LastAtom '(((5)A))) => A`
5. Составить функцию `(Delete L X)`, удаляющую из списка `L` на его верхнем уровне
   а) первое вхождение значения `X`;
   б) все вхождения значения `X`.
6. Написать функцию `(Remove2 L)`, удаляющую из списка каждый второй элемент верхнего уровня:
   `(Remove2 '(A B C D E)) => (A C E).`
7. Составить функцию `(Pair L)`, которая разбивает элементы списка `L` на точечные пары, например:
   `(Pair '(A B C D E)) => ((A . B)(C . D)(E))`

8. Определить функцию (Mix1 L1 L2), которая образует новый список, чередуя элементы заданных:
   (Mix1 '(A B C) '(Z X)) => (A Z B X C)
9. Определить функцию (Mix2 L1 L2), которая образует список точечных пар элементов, взятых последовательно из заданных списков L1 и L2, например:
   (Mix2 '(A B C) '(Z X))=> ((A . Z)(B . X)(C))
10. Написать функцию (Elem N L), которая выдаёт N-тый элемент верхнего уровня списка L. Если длина списка меньше N, то функция возвращает NIL.
11. Составить функцию (Position X L), возвращающую порядковый номер значения X в списке L, либо 0, если выражение X не встречается в списке на верхнем уровне.
12. Запрограммировать функцию (RevBr L), которая переворачивает свой аргумент-список атомов и разбивает его на уровни; количество уровней равно количеству элементов исходного списка:
    (RevBr '(A B C)) => (((C) B) A)
13. Определить функцию (RightBr L), которая преобразует свой аргумент – список атомов, разбивая его на уровни. Количество уровней равно количеству атомов, на самом глубоком уровне находится последний атом исходного списка:
    (RightBr '(A B C)) => (A (B (C)))
14. Определить функцию (LeftBr L), которая делает преобразование исходного списка атомов, подобное RightBr, но на самом глубоком уровне находится первый атом исходного списка:
    (LeftBr '(A B C)) => (((A) B) C)
15. Составить функцию (RightBrOut L), являющуюся обратной к функции RigthBr: (RightOut '(A(B(C)))) => (A B C)
16. Составить функцию (LeftBrOut L), обратную к функции LeftBr:
    (LeftOut '(((A)B)C)) => (A B C)
17. Написать функцию (Fact N) с одним аргументом – натуральным числом N. Функция строит списочное выражение, являющееся записью произведения натуральных чисел от 1 до N:
    (Fact 4)=>(1 * 2 * 3 * 4) или (4 * 3 * 2 * 1).
18. Определить функцию (MakeSet L), которая преобразует свой аргумент L – одноуровневый список атомов во множество, исключая в нём повторяющиеся элементы.

19. Запрограммировать основные операции с множествами: пересечение, объединение, разность множеств. Множества представляются как списки атомов без повторений. Составить также функции-предикаты для проверки равенства множеств и вхождения одного множества в другое.

Параллельная рекурсия и рекурсия высшего порядка

1. Определить функцию `(Depth L)`, вычисляющую глубину списка `L`, т.е. максимальное количество уровней в нём. Например:
```
(Depth '(((A (5) 8) B (K))(G (C)))) => 4
```
2. Составить функцию `(Subst A L E)`, заменяющую в произвольном списочном выражении `L` на всех его уровнях все вхождения атома `A` на выражение `E`. Например:
```
(Subst 'Q '(Q (B (Q)) C ((Q) 8))) '(A Z)) =>
              ((A Z) (B ((A Z))) C (((A Z)) 8)))
```
3. Определить функцию-предикат `(OnlyZ L)`, которая вырабатывает `T` в том случае, если в списке `L` на всех его уровнях встречается только атом `Z`, иначе вырабатывает `NIL`. Например:
```
(OnlyZ '((Z (Z())Z)()Z) )=> T,
(OnlyZ '((Z (Z())8)()Z) )=> NIL.
```
4. Написать функцию `(Trans N S)`, которая упрощает структуру списочного выражения `S`, заменяя в нём все списочные элементы, находящиеся на уровне `N` (`N>=1`), на атом “#”. Например:
```
(Trans 2 '(((A(5)8)B(K))(G(C)))) => ((# B #)(G #))
```
5. Определить функцию `(Level N S)`, которая строит список из элементов списочного выражения `S`, находящихся на уровне `N` (`N>=1`), например:
```
(Level 2 '(((A (5) 8) B) 7 (G ((())))))
                        => ((A (5) 8) B G (NIL))
```
6. Составить функцию `(Deep S)`, вырабатывающую атом списочного выражения `S`, который находится на наиболее глубоком уровне (если таких атомов несколько, выбирается любой). Например:
```
(Deep '(A (B (A)) C ((D) 8))) => A
```
7. Определить функцию `(Freq S)`, которая для каждого атома, входящего в списочное выражение `S`, вычисляет частоту его вхождения в `S` и выдает список пар вида: *атом – его частота*, например:
```
(Freq '(A (B (A)) C ((A) 8)))
                => ((A 3)(B 1)(C 1)(8 1))
```