Рекурсивные функции.

Лекция 4.

Специальности : 230105, 010501

Понятие рекурсии.

Определение. **Функция называется рекурсивной, если в ее теле содержится вызов самой этой функции.**

Принято различать рекурсию по значению и рекурсию по аргументам.

В случае *рекурсии по значению* рекурсивный вызов определяет результат функции. Пример – принадлежность объекта списку.

В случае *рекурсии по аргументам* в качестве результата функции возвращается значение некоторой другой функции и рекурсивный вызов участвует в вычислении аргументов этой функции. Пример – нахождение суммы элементов списка.

Рекурсия называется *простой*, если вызов функции встречается в некоторой ветви лишь один раз. Простой рекурсии в процедурном программировании соответствует обыкновенный цикл.

Основные правила построения рекурсивных функций.

- Определить количество и вид аргументов.
- Определить характер результата.
- Задать как минимум одно условие окончания рекурсии.
- Определить формирование результата функции.
- Описать формирование новых значений аргументов для рекурсивного вызова.

Пример 1 : суммирование элементов списка (muLISP).

Количество аргументов : 1 – список.

Результат : число.

Условие окончания рекурсии – пустой список :

((null list) 0)

Формирование результата функции :

(+ (car list) сумма хвоста)

Сумма хвоста есть результат рекурсивного вызова проектируемой функции с хвостом списка в качестве аргумента.

Описание функции :

```
(defun sum (list)
   ((null list) 0)
   (+ (car list)(sum (cdr list))))
```

Результат (sum ‘(9 7 5 6 4)) есть 31.

Реализация функции суммирования элементов списка в newLISP-tk.

```
(define (sum_new lst)
 (cond
   ((null? lst) 0)
   (true (+ (first lst) (sum_new (rest lst))
         )
   )
 )
)
```

Пример 2 : принадлежность списку (muLISP).

Количество аргументов : 2 – произвольное s-выражение (obj) и список (lst).

Результат : значение T (истина), либо NIL (ложь).

Условий окончания рекурсии два :

((null lst) nil) – пустой список, искомый объект не найден,

((equal obj (car lst)) T) – объект найден.

В случае несовпадения искомого объекта и головы списка результирующее значение вычисляется путем вызова проектируемой функции с хвостом списка в качестве второго аргумента. Полное описание функции :

```
(defun member (obj lst)
((null lst) nil)
((equal obj (car lst)) T)
(member obj (cdr lst)))
```

Результат (member 7 ‘(1 2 3 4))

есть nil.

Результат (member 3 ‘(1 2 3 4))
есть T.

Реализация функции принадлежности элемента списку в newLISP-tk.

```
(define (my_member obj lst)
 (cond
   ((null? lst) nil)
   ((= obj (first lst)) true)
   (true (my_member obj (rest lst)) )
 )
)
```

Пример 3 : объединение списков (muLISP).

Вариант 1.

Количество аргументов : 2 – списки lst1 и lst2.

Результат : список.

Условий окончания рекурсии три :

((and (null lst1)(null lst2)) nil) - оба списка пустые,

((null lst1) lst2) – первый список пустой,

((null lst2) lst1) – второй список пустой.

Результат функции строится путем присоединения головы первого списка в качестве головы к результату вызова проектируемой функции с хвостом первого списка в качестве первого аргумента и вторым списком в качестве второго аргумента.

Описание первого варианта функции объединения списков.

(defun append (lst1 lst2)

((and (null lst1)(null lst2)) nil)

((null lst1) lst2)

((null lst2) lst1)

(cons (car lst1)(append (cdr lst1) lst2)))

Результат (append ‘(1 2 3 4) ‘(5 6 7 8)) есть

(1 2 3 4 5 6 7 8)

Рассмотрим варианты более компактной записи условия окончания рекурсии.

Объединение списков : второй вариант.

Количество аргументов : 2 – списки lst1 и lst2.

Результат : список.

Условий окончания рекурсии два :

((null lst1) lst2) – первый список пустой,

((null lst2) lst1) – второй список пустой.

Результат функции формируется аналогично варианту 1.

Полное описание функции :

```
(defun append (lst1 lst2)
 ((null lst1) lst2)
 ((null lst2) lst1)
 (cons (car lst1)(append (cdr lst1) lst2)))
```

Объединение списков : третий вариант.

Количество аргументов : 2 – списки lst1 и lst2.

Результат : список.

Условие окончания рекурсии : ((null lst1) lst2).

Результат функции формируется аналогично варианту 1.

Полное описание функции :

```
(defun append (lst1 lst2)
 ((null lst1) lst2)
 (cons (car lst1)(append (cdr lst1) lst2)))
```

Реализация функции объединения списков (третий вариант) в newLISP-tk.

```
(define (my_append lst1 lst2)
 (cond
   ((null? lst1) lst2)
   (true (cons (first lst1) (my_append (rest lst1) lst2) )
   )
 )
)
```