1.3 Определение функций пользователем

1.3.1 Лямбда-функции

Основа определения и вычисления функций – лямбда-исчисление Черча.

Лямбда-выражение:
(**LAMBDA** (x_1 x_2 … x_n) S_1 S_2 … S_k)
Пример: $f(x,y) = x^2 + y^2$
(lambda (x y) (+(* x x)(* y y)))

Лямбда-вызов:
(лямбда-выражение a_1 a_2 … a_n)
$f(2,3) - ?$
Пример: ((lambda (x y) (+(* x x)(* y y))) 2 3)
Передача параметров осуществляется по значению!

1.3.2 Определение функций с именем

(**DEFUN** имя-функции лямбда-список S_1 S_2 … S_k)
Возвращает имя функции

Побочный эффект: связывание имени функции с лямбда-выражением
(LAMBDA лямбда-список S_1 S_2 … S_k)

1.3.3 Ключевые слова параметров в лямбда-списке

Ключевое слово	Значение ключевого слова
&optional	необязательные параметры

Параметры до первого & обязательны при вызове.
Для необязательных параметров можно указать значение при его отсутствии (по умолчанию nil).
Пример: Можно обращаться с 1, 2, 3 аргументами

```
(defun f(x &optional (y '(1 2 3))z)
  (append x y z)
)
```

1.4 Предикаты

(**ATOM** s-выражение)

(**LISTP** s-выражение)

(**SYMBOLP** s-выражение)

(**NUMBERP** s-выражение)

(**NULL** s-выражение)

Предикаты для работы с числами:

(= n_1 …n_m), где n_i – число или связанная с числом переменная

(< n_1 …n_m), где n_i – число или связанная с числом переменная

Аналогично определяются предикаты: > ; <= ; >=; /=

Предикат для сравнения s-выражений

(**EQUAL** s_1 s_2), где s_i– s-выражение

1.5 Псевдофункция SETQ

(**SETQ** p_1 s_1 … p_n s_n),
где p_i-символ, s_i-s-выражение
Пример:
(setq a 3 b 4 c a)$\rightarrow$3 ; a $\rightarrow$3,b $\rightarrow$4,c $\rightarrow$3

1.6 Разветвление вычислений

(**COND**

(P_1 V_1)

(P_2 V_2)

……

(P_n V_n)

),

где P_i – предикат, V_i – вычислимое выражение

(i = 1,…n)

Допустимо использование (P_i) или (P_i V_{i1}… V_{ik})

В предикатах можно использовать логические функции: AND, OR, NOT

1.7 Рекурсия

Ошибки при написании рекурсивных функций:

•ошибочное условие, которое приводит к бесконечной рекурсии;

- неверный порядок условий;
- отсутствие проверки какого-нибудь случая.

1.7.1 Трассировка функций

Включение трассировки:

(**TRACE** <имя функции>)

Если трассируется несколько функций, то их имена – аргументы **TRACE**

Выключение трассировки:

(**UNTRACE**)

Если отключается трассировка некоторых функций, то их имена - аргументы **UNTRACE**

1.7.2 Простая рекурсия

- рекурсия по значению (рекурсивный вызов определяет результат функции);
- рекурсия по аргументу (результат функции – значение другой функции, аргументом которой является рекурсивный вызов исходной функции).

Пример 1: Определим функцию **ST**, возводящую число в целую неотрицательную степень.

$$x^n = \begin{cases} 1, n = 0 \\ x \cdot x^{n-1}, n > 0 \end{cases}$$

```
(defun st(x n)
 (cond
   ((= n 0) 1)
   (t (* x (st x (- n 1))))
 ))
```

1.7.2 Простая рекурсия

Пример 2: Определим функцию **COPY**, копирующую список на верхнем уровне (без учета вложенностей).

```
(defun copy(l)
  (cond
    ((null l)l)
    (t (cons (car l)(copy (cdr l))))
))
```

Пример 3: Определим функцию **MEMBER_S**, проверяющую принадлежность s-выражения списку на верхнем уровне. В случае, если s-выражение принадлежит списку, функция возвращает часть списка, начинающуюся с первого вхождения s-выражения в список.

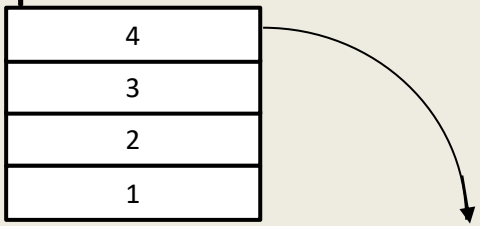
```
(defun member_s(s l)
 (cond
  ((null l) l)
  ((equal s (car l))l)
  (t (member_s s (cdr l)))
))
```

Пример 4: Определим функцию **REMOVE_S**, удаляющую все вхождения заданного s-выражения в список на верхнем уровне.

```
(defun remove_s(s l)
 (cond
  ((null l) l)
  ((equal s (car l))(remove_s s(cdr l)))
  (t (cons (car l)(remove_s s(cdr l))))
))
```

1.7.3 Использование накапливающих параметров

Пример 1: Определим **REVERSE1**, обращающую список на верхнем уровне, с дополнительным параметром для накапливания результата обращения списка.




```
(defun reverse1(l1 &optional l2)
 (cond
  ((null l1) l2)
   (t (reverse1 (cdr l1)(cons (car l1) l2)))
))
```

1.7.3 Использование накапливающих параметров

Пример 2: Определим функцию **POS**, определяющую позицию первого вхождения s-выражения в список (на верхнем уровне).

```
(defun pos(s l &optional (n 1))
 (cond
  ((null l) l)
  ((equal s (car l)) n)
  (t (pos s (cdr l)(+ n 1)))
))
```

1.7.4 Параллельная рекурсия

Пример 1: Определим функцию **COPY_ALL**, копирующую список на всех уровнях.

```
(defun copy_all  (l)
   (cond
       ((null l)  nil )
       ((atom l) l)
       (t  (cons (copy_all (car l)) (copy_all (cdr l))))
 ))
```

1.7.4 Параллельная рекурсия

Пример 2: Определим функцию **MAX_IN_LIST**, находящую максимальный элемент в числовом списке, содержащем подсписки.

```
(defun max_in_list(l)
 (cond
  ((atom l) l)
  ((null (cdr l))(max_in_list(car l)))
  (t (max (max_in_list (car l))(max_in_list (cdr l))))
))
```

1.8 Интерпретатор языка Лисп EVAL

(**EVAL** s-выражение)

Пример:

(setq a 'b b 'c c 1) →1 ; a→b, b →c, c →1

a →b

(eval a) →c

(eval (eval a)) →1