Макросы.

Лекция 14.

Специальности : 230105, 010501

Назначение.

<u>Определение.</u> Макросы есть средство динамического программирования, позволяющие автоматически получать программный код или вычисляемые структуры.

Идея автоматического динамического программирования состоит в формировании вычисляемых выражений с помощью программы.

Макросы позволяют :

- **создавать расширения синтаксиса и семантики Лиспа;**
- **создавать задачно-ориентированные языки программирования.**

Использование макросов.

1) **Для расширения языка.**
2) **Для описания операторов, структур и циклов других языков программирования.**

Отличительные особенности.

Макросы определяются и используются как функции. В Коммон Лиспе (и muLISP'e) синтаксис определения макроса следующий :

(defmacro <имя> (<список формальных параметров>) <тело>)

Вызов : (<имя> <список фактических параметров>)

Макрос вычисляется в два этапа :

1) **Построение лисповской формы, задаваемой определением макроса (этап трансляции).**
2) **Вычисление значения построенной формы (этап реализации).**

Отличие от функций – в способе вычисления. В результате вычисления функции не происходит вычисления формы, которая возвращается в качестве результата. Пример (muLISP) :

```
; Макрос действует подобно SETQ, но
; блокирует вычисление второго аргумента
(defmacro setq1 (x y)
 (list 'setq x (list 'quote y)))
Вызов (setq1 lst1 (a b c)) дает результат :
(a b c)
```

```
(defun setq2 (x y)
(list 'setq x (list 'quote y)))
Вызов (setq2 lst2 '(a b c)) дает результат :
(SETQ LST2 (QUOTE (a b c)))
```

Этапы разработки макроса.

1) Точно определить назначение и функциональные особенности проектируемого макроса.
2) Определить количество, вид и тип аргументов макроса.
3) Выяснить последовательность выполняемых макросом действий.
4) Разработка модели макроса.
5) Существующими средствами функционального языка описываем вычисляемую макросом форму.
6) Запись программы (defmacro …), которая в результате выполнения дает вычисляемую макросом форму.

Пример 1: описание оператора цикла DO в muLISP'e.

Предложение DO имеет следующую форму :

```
(do ((var1 val1 step1) … (varN valN stepN))
    (условие_завершения форма11 … форма1L)
    форма21 … форма2M)
```

Первый аргумент предложения описывает параметры var1, … ,varN цикла, их начальные значения val1, … , valN, а также лисповские формы step1, … , stepN изменения значений параметров цикла. Тело (обозн. body) цикла содержит :

- **Условие окончания. Как только его значение становится истинным, последовательно вычисляются форма1i, и значение последней из них возвращается в качестве значения всего предложения DO.**

- **Описания лисповских форм форма2i, которые последовательно вычисляются в очередном проходе цикла, При этом каждой из vari присваивается значения форм stepi, вычисляемых в текущем контексте. Если для переменной vari не задана форма, по которой она обновляется, то значение переменной не меняется.**

Работа цикла DO.

В качестве примера рассмотрим использование предложения DO для определения функции вычисления n-й степени числа (n – целое, положительное).

```
(defun expt2 (x n)
 (do
    ((result x (* result x))
     (count n (- count 1)))
    ((= count 1) result)))
```

Вызов (expt2 2 3) дает 8.

Поскольку начальные значения присваиваются параметрам цикла в начале работы, а каждый новый проход сопровождается изменением начальных значений в ходе вычисления форм, то данный цикл может быть описан с помощью λ-определения :

```
((lambda (var1 … varN)
          (loop
                (body (psetq var1 step1 … varN stepN))))
  val1 … valN)
```

Здесь body есть тело цикла.

Стоит задача : написать макрос, который на этапе трансляции получает указанную форму.

Получение формы λ-вызова. Постановка задачи.

Обозначим список :

((var1 val1 step1) … (varN valN stepN)) как letlist.

Тогда целевое представление цикла будет иметь вид :

(do letlist body)

Для построения формы λ-вызова на этапе трансляции необходимо получить :

- **список формальных параметров;**
- **список фактических параметров;**
- **тело λ-выражения, в котором происходит изменение параметров цикла.**

Этап 1 : построение списка формальных параметров.

Исходные данные : letlist.

Результат : список (var1 … varN)

Вариант 1 – использование (mapcar ‘car letlist)

С учетом произвольности формы списка letlist наиболее предпочтительным является вариант использования для вычисления элементов искомого списка следующего λ-определения :

(lambda (var) (if (atom var) var (car var)))

Используя letlist в качестве фактического параметра получаем :

(mapcar '(lambda (var)(if (atom var) var (car var))) letlist)

Этап 2 : список начальных значений цикла как фактические параметры λ-вызова.

Исходные данные : letlist.

Результат : список (val1 … valN).

Искомый список получается применением функционального аргумента :

(lambda (var)(if (atom var) nil (cadr var)))

к элементам списка letlist с помощью mapcar :

(mapcar '(lambda (var)(if (atom var) nil (cadr var))) letlist)

Этап 3 : построение формы для изменения параметров цикла.

Основная смысловая нагрузка данного этапа – получение списка : (var1 step1 … varN stepN) на основе исходного letlist.

Искомый список получается путем применения функционального аргумента :

(lambda (var)((atom var) nil)((caddr var)(list (car var)(caddr var))))

к элементам списка letlist с последующим раскрытием скобок в списке-результате :

```
(mapcan '(lambda (var)
                 ((atom var) nil)
                 ((caddr var)(list (car var)(caddr var))))
 letlist)
```

Само тело λ-выражения получается следующим образом :

(cons ‘loop (cons body (list (cons ‘psetq (<вызов mapcan>)))))

DO : описание макроса.

```
(defmacro do (letlist body)
 (cons (list 'lambda (mapcar '(lambda (var)(if (atom var) var (car var))) letlist)
          (cons 'loop (cons body
                       (list (cons 'psetq (mapcan '(lambda (var)
                                                    ((atom var) nil)
                                                    ((caddr var)(list (car var)(caddr var))))
                                                  letlist)
                             )
                       )
                )
          )
       )
       (mapcar '(lambda (var)(if (atom var) nil (cadr var))) letlist)
 )
)
```

Пример 2 : преобразование LET в LAMBDA.

Задача : преобразовать локальное определение LET в локальное определение LAMBDA средствами muLISP'a.

```
; Преобразование локального определения LET
; в локальное определение LAMBDA

(defmacro let (letlist body)
  (cons (list 'lambda (mapcar 'car letlist) body)
        (mapcar 'cadr letlist)))

; Тестирование локального определения LET

(let
; Список формальных и фактических параметров
    ((x '(1 2))
     (y '(3 4)))
; Тело функции
    (((atom x) y)
     ((atom y) x)
     (cons x y))
)
```

Описание макросов в newLISP-tk.

Основное отличие - при вызове макроса возвращается лисповская форма, получаемая на этапе трансляции.

Пример – описание применяющего функционала *funcall* на основе *apply* :

```
(define-macro (funcall fun)
  (list 'apply fun (list 'quote (args))))
```

Здесь посредством встоенной функции args формируется список аргументов произвольной длины.

Вызов

```
(funcall '+ 1 2 3)
```

дает (apply '+ (quote (1 2 3))). Для вычисления значения полученной формы воспользуемся функцией-интерпретатором *eval* :

```
(eval (funcall '+ 1 2 3))
```

Тестирование макросов.

Некорректное определение макроса может привести на этапе трансляции макроса к неожиданным выражениям. Основная трудность поиска ошибок здесь заключается в недоступности построенных выражений. В этих целях для тестирования макросов в Коммон Лиспе и muLISP'е существует специальная функция, которая осуществляет лишь трансляцию макровызова без вычисления построенной формы :

(macroexpand <макровызов>)

Пример тестирования для макроса do.

```
; Здесь для тестирования макроса рассмотрен
; пример его использования для определения
; функции expt2 вычисления степени числа
(macroexpand '(do
              ((result x (* result x))
               (count n (- count 1)))
              ((= count 1) result))
)
```

Вызов macroexpand для данного макроса возвратит задаваемую определением макроса лисповскую форму :

```
((LAMBDA (RESULT COUNT)
   (LOOP
      ((= COUNT 1) RESULT)
      (PSETQ RESULT (* RESULT X)
                  COUNT (- COUNT 1))))
X N)
```