Когда использовать макросы

Как мы можем отличить когда данная функция должна действительно быть функцией, а не макросом? Почти всегда существует чёткое отличие между случаями которые имеют склонность к реализации с помощью макросов и которые такой склонности не имеют. (плохо) По умолчанию мы должны использовать функции: неопрятно использовать макрос там, где место функции. Мы должны использовать макросы только там, где они дают нам специфические преимущества.

Когда же макросы дают преимущества? Это предмет обсуждения данной главы. Обычно вопрос не столько в примуществе, сколько в неоходимости. Большую часть вещей, которую мы делаем с помощью макросов не может быть осуществлена функциями. Секция 8.1 перечисляет заряктерные операции, которые могут быть реализованы только как макросы. Однако, существует небольшой (но интересный) класс пограничных классов, где оператор справедливо может быть написан и в виде функции и в виде макроса. Для таких ситуаций, в секции 8.2 приводятся аргументы за и против макросов. В итоге, ознакомившись с тем, с чем могут справиться макросы, мы переходим в секции 8.3 к связанному вопросу (related question): какого рода вещи люди с помощью них делают?

Когда больше ничего не помогает

Общий принцип хорошего дизайна таков, что если вы находите похожий код в нескольких местах программы вы должны написать подпрограмму и заменить похожие участки вызовом этой подпрограммы. Когда мы применяем этот принцип к Lisp программам, мы должны решить, должна ли быть «подпрограмма» функцией или макросом.

В некоторых случаях проще принять решение писать макрос вместо функции, так как только макрос может сделать то, что

нужно. Функция похожая на 1+ потенциально могла быть написана как функция и как макрос:

```
(defun 1+ (x) (+ 1 x))

(defmacro 1+ (x) `(+ 1 ,x))
```

Но WHILE, из секции 7.3, может быть определена только как макрос:

```
(defmacro while (test &body body)
    `(do ()
         ((not ,test))
         ,@body))
```

Не существует способа повторить поведение этого макроса при помощи функции. Определение WHILE вклеивает выражения переданные как BODY в DO, где они вычисляются только если TEST выражение возвращает NIL. Ни одна функция не может сделать этого; в вызове функции все аргменты вычисляются ещё до того как вызовется сама функция.

Когда вам потребовался макрос, что вы хотите получить от него? Макросы могу сделать две вещи, которые не могу сделать функции: они могут контролировать (или предотвратить) вычисление их аргументов, и также они разворачиваются непосредственно в вызываемый контекст. Любое приложение, нуждается в макросах, в конечном счёте испытывает потребность в одном или обоих этих свойствах.

Неформальное объяснение, что «макросы не вычисляют свои аргументы» слегка неверно. Точнее будет сказать, что макросы *контролируют* вычисление своих аргументов в вызове макроса. В зависимости от того, куда аргумент помещён в развёртке макроса, он может быть вычислен однажды, много раз или совсем никогда. Макросы производят такой контроль четыремя способами:

1) *Трансформация*

2) *Связывание*

3) gfg

4) fgf