

*О.Н. Половикова***Анализ возможностей системы макроопределений языка Common Lisp для создания новых управляющих конструкций***O.N. Polovikova***Analysis of Possibilities of Language Common Lisp Macros to Create New Control Structures**

Рассмотрены возможности системы макроопределений функционального языка Common Lisp, которая представляет интерес как для разработчиков программных приложений, так и для аналитиков абстрактного программирования. Представлены примеры макросов для создания оператора цикла и условного оператора.

Ключевые слова: Common Lisp, макроопределения, оператор цикла, условный оператор, лексические переменные.

В данной статье проанализированы возможности системы макроопределений функционального языка Common Lisp, которая представляет интерес как для разработчиков программных приложений, так и для аналитиков абстрактного программирования. Язык Common Lisp, несмотря на свои принципиальные отличия от всех остальных языков программирования, развивался и развивается согласно общим закономерностям, по которым проходят становления и другие популярные языки. Первоначальные версии языка существенно отличались от современной его реализации. Новые актуальные парадигмы и направления в технике вычислений существенно повлияли на Common Lisp. Современный язык поддерживает объектно-ориентированное программирование, в том числе множественное наследование и мультиметоды, аспектное программирование (пакет AspectL), метапрограммирование, обобщенное программирование и другие востребованные программистами парадигмы [1, 2].

Но, несмотря на множество пересечений Common Lisp с другими языками программирования, возможности создавать свои собственные макроопределения выделяют его перед остальными языками. В то время как многие из идей, появившиеся в Лиспе, от условных выражений до сборки мусора, были добавлены в другие языки, есть одна особенность языка, которая продолжает делать Common Lisp стоящим особняком от всех, это его система макросов [2].

Макроопределения можно использовать как инструмент для расширения синтаксиса самого языка

This article describes the features of the system macro functional language Common Lisp, which is interesting both for developers of software applications, and for analysts of abstract programming. The examples of macros to create the loop operator and conditional operator are presented.

Key words: Common Lisp, macros, for loop, conditional statement, lexical variables.

или для создания своего нового языка программирования. Другими словами, можно обогатить существующий язык новыми управляющими конструкциями и операциями или создать свой набор функциональных абстракций, который определит синтаксис нового языка. Несмотря на явные преимущества, которые может предоставить система макроопределений, при ее целесообразной реализации в конкретном проекте, макросы Lisp-программистами используются крайне редко. Но по некоторой причине большое количество программистов, которые фактически не используют макросы Лиспа и не задумываются о создании новых функциональных абстракций или определений новых иерархий классов для решения своих задач, боятся самой мысли о том, что они будут иметь возможность описания новых синтаксических абстракций [2]. Поэтому анализ возможностей применения макроопределений для решения практических задач является актуальным и востребованным. Представленные в статье примеры указывают на принципиальные ограничения, которые возникают при создании новых управляющих конструкций (новых операций языка), если при этом не использовать макроопределения.

Язык Common Lisp (как и некоторые другие языки программирования) поддерживает два пути, которыми создаются расширения в языках. Первый способ заключается в создании своей библиотеки функций дополнительно к стандартному ядру языка. Второй способ основывается на языках, которые

поддерживают в том числе и объектно-ориентированную парадигму, когда функциональность расширяется за счет создания своего нового класса (или шаблона), который обладает требуемыми возможностями. Оба подхода регламентируют создание новых функций, с той лишь разницей, что второй способ реализуется на основе обобщающих функций и их методов. Поэтому в качестве примера можно рассматривать ограничения от использования функции, которая программирует управляющую конструкцию или операцию.

Следующий код определяет новую функцию `if_` с возможностями управляющей конструкции «if-then-else». Реализуем простейшую функцию с тремя формальными параметрами: условие, первое выражение, которое следует выполнить, если данное условие ложно, второе выражение, которое выполняется, в противном случае.

Указание интерпретатору:

```
(defun if_ (pred expif expelse) (cond
  ((eval pred) expif) (t expelse)))
```

создание новой функции с тремя параметрами: условие, выражение в случае истинности условия, выражение в случае, если условие ложное.

Результат:

IF_

Указание интерпретатору:

```
(progn (setq x (read)) (setq y
  (read)) (if_ '(> x y) (print x)
  (print y)))
```

7

8

Результат:

7

8

8

Несмотря на то, что функциями обеспечиваются основные возможности данного языка, этот пример демонстрирует невозможность их использования для организации работы условного оператора. Ограничения на область применения функций возникают из-за отсутствия способа запретить вычисление аргументов при ее вызове. Вычисление аргументов происходит слева направо, поэтому интерпретатором будут последовательно вычислены выражения: `(print x)`, `(print y)`, а затем результаты вычислений будут переданы в качестве фактических параметров в функцию `if_`. Оба выражения интерпретируются еще до анализа условия их выполнения.

Если заблокировать вычисление выражений в вызове функций с использованием цитирования или квазичитирования, то возникнет неопределенность с лексическими переменными более внешнего вызова, по отношению к функции `if_`. Рассмотрим следующий пример, в котором все аргументы в вызове функции `if_` заблокированы.

Указание интерпретатору:

```
(defun if_ (pred expif expelse) (cond
  ((eval pred) (eval expif)) (t (eval
  expelse))) )
```

Результат:

IF_

Указание интерпретатору:

```
(defun fn (x y) (if_ '(> x y) x y));
```

создание функции, которая вызывает функцию `if_`

Результат:

fn

Указание интерпретатору:

```
(fn 3 6)
```

Результат:

Error: The variable X is unbound.

При вызове функции с параметрами возникает ошибка. Функция `if_` является внешней функцией для функции `fn`, поэтому лексические переменные (`x` и `y`) данной функции `fn` в вызываемой функции `if_` не видны. Цитирование и квазичитирование выражений, которые применяются в качестве фактических параметров в вызове функции, позволяют заблокировать их вычисления, но вызывают неопределенности с лексическими переменными. Использование функций возможно только для случая, когда либо все входящие выражения заранее вычислены, либо вычисления этих выражений не зависят от лексических переменных внешних функций (вызывающих функций).

Аналогичные ограничения возникают при попытке реализовать другие управляющие конструкции, в том числе и операторы цикла, с применением функций. Таким образом, можно сделать вывод, что управляющие конструкции невозможно реализовать на основе функций, для этих целей следует использовать другие формы s-выражений языка.

Рассмотрим пример реализации новой управляющей конструкции «if-then-else» на основе макроопределений.

Указание интерпретатору:

```
(defmacro if_ (condition bodyif bodyelse)
  `(cond (,condition ,bodyif)
  (t ,bodyelse)))
```

Результат:

IF_

Указание интерпретатору:

```
(progn (setq x (read)) (setq y
  (read)) (if_ '(> x y) (print x) (print
  y)))
```

7

8

Результат:

8

8

Данный пример кода демонстрирует корректную работу реализуемого условного оператора. Выражения, которые передаются макроопределению

в качестве фактических параметров, заранее не вычисляются, поэтому их можно не блокировать. Следующий блок кода показывает отсутствие неопределенностей с лексическими переменными при использовании макросов.

Указание интерпретатору:

```
(defun fn(x y) (if_ (> x y) x y) )
```

Результат:

FN_

Указание интерпретатору:

```
(fn_ 6 7)
```

Результат:

7

Есть еще одна возможность макроопределений, которая не добавляет функциональности макросам, но делает их использование более прозрачным. Де-структурирование параметров избавляет программистов от лишних преобразований над входящими параметрами. Следующий пример кода показывает применение данной возможности макроопределений при реализации нового оператора цикла LOOPCDR. Новая форма организует циклическую конструкцию, подобную оператору DOLIST, но, в отличие от макроопределения DOLIST, переменная цикла var должна последовательно принимать значения не элементов списка list, а результатов работы функции CDR для этого списка. То есть на

первой итерации переменная цикла должна принять значение списка list, затем значение хвоста этого списка, далее значение хвоста для хвоста списка и т.д., пока список не закончится.

```
(defmacro loopcdr ((var list res)
```

```
&rest forma)
```

```
`(do (( ,var ,list (cdr ,var) ))
```

```
((null ,var ) ,res ) (progn
```

```
,@forma )) )
```

Результат:

LOOPCDR

Проверим работу построенного оператора цикла.

Указание интерпретатору:

```
(loopcdr (a '(c cd f) t) (print a))
```

Результат:

```
(C CD F)
```

```
(CD F)
```

```
(F)
```

T

Макроопределения позволяют не только создавать свои условные операторы и операторы цикла, но также могут программировать работу любых n-арных операций, в том числе логических, побитовых логических арифметических и т.д. Макросы предоставляют инструментарий для расширения существующего языка Common Lisp, а также для создания своего интерпретированного языка программирования.

Библиографический список

1. Kent M. Pitman. Condition handling in the lisp language family. – 2001.

2. Peter Seibel. Practical Common Lisp. – 2006.