Замыкания в Common Lisp

А. Г. Фенстер, http://info.fenster.name

6 февраля 2009 г.

Определение замыкания

Замыканием (closure) называется сохранение состояния контекста определения функции в момент её определения. Рассмотрим следующее определение двух функций:

```
(let ( (x 0) ) ; x -- переменная, действующая в блоке let
    (defun set-x-1 (value) (setq x value))
    (defun get-x-1 () x)
)
```

и сравним его с аналогичным определением, но без блока `let`:

```
(defun set-x-2 (value) (setq x value))
(defun get-x-2 () x)
```

В первом случае перед нами замыкание: функции `set-x-1` и `get-x-1` при определении сохраняют контекст, в котором они были определены. Переменная `x` из первого определения теперь доступна только из функций `set-x-1` и `get-x-1`:

```
> (set-x-1 7)
7
> (get-x-1)
7
> (set-x-1 66)
66
```

```
> (get-x-1)
66
> x

*** - EVAL: переменной X не присвоено значение
```

Во втором случае замыкание не было создано: функции `set-x-2` и `get-x-2` всегда обращаются к глобальной переменной `x`, значение которой доступно не только этим функциям:

```
> (setq x 12)
12
> (get-x-2)
12
> (get-x-1) ; берётся другой x!
66
```

Значение «скрытой» переменной `x` можно узнать при помощи специальной функции `(function-lambda-expression 'get-x-1)`.

Таким образом, замыкания позволяют создавать функции, сохраняющие состояние в переменных из контекста определения функции, недоступных извне.

Генератор замыканий

Напишем функцию, которая создаёт замыкание. Результатом её будет анонимная функция, которая перебирает натуральные числа от 0 и далее по порядку, запоминая последнее выданное число.

```
> (defun make-counter ()
    (let ( (x -1) )
      (lambda () (setq x (+ x 1)))
    )
  )
MAKE-COUNTER
> (setq c1 (make-counter))
#<FUNCTION :LAMBDA NIL (SETQ X (+ X 1))>
> (funcall c1)
```

```
0
> (funcall c1)
1
> (apply c1 nil)
2
> (setq c2 (make-counter))
#<FUNCTION :LAMBDA NIL (SETQ X (+ X 1))>
> (funcall c2)
0
> (funcall c2)
1
> (funcall c1)
4
```