

CAR EQUAL
CONS
CDR ATOM

```
(diff0 (1 x) (atom 1))  
(diff0 (eq 1 x) 1) ;l=1  
0)) ;l=константа  
(diff0 (first l) '+)  
(list '+  
  (diff0 (second l) x)  
  (diff0 (third l) x)))  
(diff0 (eq (first l) '*)  
(list '+  
  (list '*  
    (diff0 (second l) x)
```

Функциональное программирование: базовый курс

Лекция 7. Макросы в языке Lisp.

CAR EQUAL
CONS
CDR ATOM

Функциональное программирование: базовый курс

Лекция 7

Макросы в языке Lisp

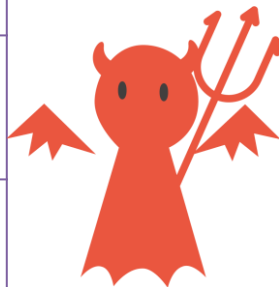
Основные сведения о макросах

CAR EQUAL
CONS
CDR ATOM

Функции и макросы



Функции	Макросы
выполняются на этапе выполнения программы (runtime)	выполняются на этапе компиляции программы
аргументы вычисляются	аргументы не вычисляются
результат работы – одно или несколько значений	результат работы – программный код



CAR EQUAL
CONS
CDR ATOM


```
(defun my-if (condition true-form false-form)
  (cond
    (condition true-form)
    (t false-form)))
```

```
[1]> (my-if (> 20 10) (print "TRUE") (print "FALSE"))
"TRUE"
"FALSE"
"TRUE" ; возвращаемое значение
```

CAR EQUAL
CONS
CDR ATOM

```
(defmacro my-if (condition true-form false-form)
  `(cond
    (,condition ,true-form)
    (t ,false-form)))
```

```
[1]> (my-if (> 20 10) (print "TRUE") (print "FALSE"))
"TRUE"
"TRUE" ; возвращаемое значение
```



теперь эта форма
не вычисляется

```
[2]> (if (> 20 10) (print "TRUE") (print "FALSE"))
"TRUE"
"TRUE" ; возвращаемое значение
```

CAR EQUAL
CONS
CDR ATOM

```
(defmacro имя-макроса (список-аргументов)  
  "Строка документации с описанием макроса"  
  
  ; тело макроса  
)
```

CAR EQUAL
CONS
CDR ATOM

```
[1]> (defparameter a 10)
```

A

```
[2]> (defparameter b 20)
```

B

```
[3]> (swapf a b)
```

NIL

```
[4]> (list a b)
```

(20 10)

```
[5]> (rotatef a b)
```

NIL

```
[6]> (list a b)
```

(10 20)

CAR EQUAL
CONS
CDR ATOM

```
(defun swapf (x y)
  (let ((temp x))
    (setf x y) (setf y temp)))
```

[1]> (list a b)

(10 20)

[2]> (swapf a b)

10

[3]> (list a b)

(10 20)



ничего не изменилось, так как местами
поменялись только локальные переменные
внутри функции swapf

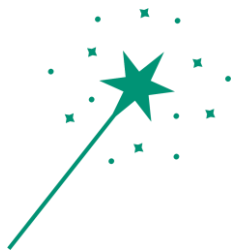
CAR EQUAL
CONS
CDR ATOM

```
[1]> (list a b)  
(10 20)
```

```
[2]> (let ((temp a)) (setf a b) (setf b temp)))  
10
```

вызов макроса (swapf a b) должен
"превратиться" в такую конструкцию

```
[3]> (list a b)  
(20 10)
```



CAR EQUAL
CONS
CDR ATOM

```
(defmacro swapf (x y)
  (list 'let (list (list 'temp x))
        (list 'setf x y) (list 'setf y 'temp))))
```

```
[1]> (list a b)
(10 20)
```

```
[2]> (swapf a b)
20
```

```
[3]> (list a b)
(20 10)
```

CAR EQUAL
CONS
CDR ATOM

```
[1]> (list a b)  
(10 20)
```

```
[2]> (macroexpand '(swapf a b))  
(LET ((TEMP A))  
  (SETF A B)  
  (SETF B TEMP))
```

T

Конструирование списков в макросах

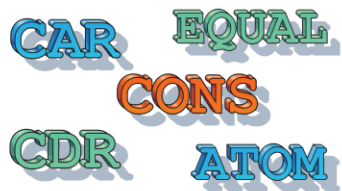
CAR EQUAL
CONS
CDR ATOM

```
[1]> (list 'sum 'of a 'and b 'is (+ a b))  
(SUM OF 10 AND 20 IS 30)
```

```
[2]> '(sum of a and b is (+ a b))  
(SUM OF A AND B IS (+ A B))
```

```
[3]> `(sum of ,a and ,b is ,(+ a b))  
(SUM OF 20 AND 10 IS 30)
```

обратная кавычка (backquote)
или
машинописный обратный апостроф

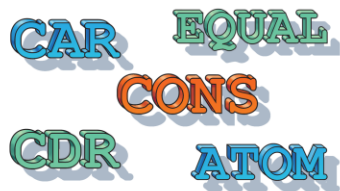


- старый вариант:

```
(defmacro swapf-old (x y)
  (list 'let (list (list 'temp x))
        (list 'setf x y) (list 'setf y 'temp))))
```

- новый вариант:

```
(defmacro swapf (x y)
  `(let ((temp ,x))
      (setf ,x ,y) (setf ,y temp))))
```



```
[1]> (defparameter lst '(11 22 33))
```

```
LST
```

```
[2]> (swapf (first lst) (third lst))
```

```
11
```

```
[3]> lst
```

```
(33 22 11)
```

```
[4]> (macroexpand '(swapf (first lst) (third lst)))
```

```
(LET ((TEMP (FIRST LST)))
```

```
  (SETF (FIRST LST) (THIRD LST))
```

```
  (SETF (THIRD LST) TEMP))
```

```
T
```

CAR EQUAL
CONS
CDR ATOM

```
[1]> (truncate 3.5)
```

```
3 ;
```

```
0.5
```

```
[2]> (multiple-value-bind (int frac) (truncate 3.5)
```

```
      (* frac frac))
```

```
0.25
```

```
[3]> (mvb (int frac) (truncate 3.5)
```

```
      (* frac frac))
```

```
0.25
```

CAR EQUAL
CONS
CDR ATOM

```
(defmacro mvb (vars-form func-form &body body)
  `(multiple-value-bind ,vars-form ,func-form ,body))
```

```
[1]> (mvb (int frac) (truncate 3.5) (* frac frac))
```

debugger invoked on a SB-INT:COMPILED-PROGRAM-ERROR in thread

#<THREAD "main thread" RUNNING {10039D4543}>:

Execution of a form compiled with errors.

Form:

```
((* FRAC FRAC))
```

Compile-time error:

illegal function call

CAR EQUAL
CONS
CDR ATOM

```
(defmacro mvb (vars-form func-form &body body)
  `(multiple-value-bind ,vars-form ,func-form ,body))
```

```
[1]> (macroexpand '(mvb (int frac) (truncate 3.5) (* frac frac)))
(MULTIPLE-VALUE-CALL
 #'(LAMBDA (&OPTIONAL (INT) (FRAC) &REST #:G704)
   (DECLARE (IGNORE #:G704))
   ((* FRAC FRAC)))
 (TRUNCATE 3.5))
```

T

```
[2]> (macroexpand-1 '(mvb (int frac) (truncate 3.5) (* frac frac)))
(MULTIPLE-VALUE-BIND (INT FRAC) (TRUNCATE 3.5) ((* FRAC FRAC)))
```

T

CAR EQUAL
CONS
CDR ATOM

```
(defmacro mvb (vars-form func-form &body body)
  `(multiple-value-bind ,vars-form ,func-form ,@body))
```

```
[1]> (macroexpand-1 '(mvb (int frac) (truncate 3.5) (* frac frac)))
(MULTIPLE-VALUE-BIND (INT FRAC) (TRUNCATE 3.5) (* FRAC FRAC))
```

T

,@ вставляет элементы списка

```
[2]> (mvb (int frac) (truncate 3.5) (* frac frac))
```

0.25

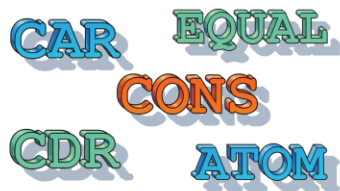
CAR EQUAL
CONS
CDR ATOM

Функциональное программирование: базовый курс

Лекция 7

Макросы в языке Lisp

Особенности разработки макросов: совпадение имен



```
(defmacro swapf (x y)
  `(let ((temp ,x))
      (setf ,x ,y) (setf ,y temp)))
```

```
[1]> (defparameter temp '(11 22 33))
```

TEMP

```
[2]> (swapf (first temp) (third temp))
```

*** - THIRD: 11 is not a list

The following restarts are available:

ABORT :R1 Abort main loop

CAR EQUAL
CONS
CDR ATOM

```
(defmacro swapf (x y)
  `(let ((temp ,x))
      (setf ,x ,y) (setf ,y temp)))
```

```
[1]> (macroexpand-1 '(swapf (first temp) (third temp)))
(LET ((TEMP (FIRST TEMP)))
  (SETF (FIRST TEMP) (THIRD TEMP))
  (SETF (THIRD TEMP) TEMP))
```

T

CAR EQUAL
CONS
CDR ATOM

Неправильное решение проблемы с совпадением имен

```
(defmacro swapf (x y)
  `(let ((long-and-unusual-name ,x))
      (setf ,x ,y) (setf ,y long-and-unusual-name)))
```

для переменной необходимо выбрать имя,
которое не совпадет с именами переменных
в вызывающем коде

```
(defmacro swapf (x y)
  `(let ((!LKNe fg$$jogdsdklj%etgui+df ,x))
      (setf ,x ,y) (setf ,y !LKNe fg$$jogdsdklj%etgui+df)))
```

CAR EQUAL
CONS
CDR ATOM

```
[1]> (gensym)
```

```
#:G813
```

```
[2]> (gensym)
```

```
#:G814
```

```
[3]> (gensym "MY-PREFIX")
```

```
#:MY-PREFIX815
```

```
[4]> (eq '#:G813 '#:G813)
```

```
NIL
```

CAR EQUAL
CONS
CDR ATOM

Правильное решение проблемы с совпадением имен

```
(defmacro swapf (x y)
```

```
  (let ((temp (gensym)))
```

```
    (let ((,temp ,x))
```

```
      (setf ,x ,y) (setf ,y ,temp))))
```

форма let вычисляется во время
компиляции (когда макрос раскрывается)
и не остается в результирующем коде

```
[1]> (macroexpand-1 '(swapf (first temp) (third temp)))
```

```
(LET ((#:G820 (FIRST TEMP)))
```

```
  (SETF (FIRST TEMP) (THIRD TEMP))
```

```
  (SETF (THIRD TEMP) #:G820))
```

T

уникальное имя, полученное с помощью gensym

CAR EQUAL
CONS
CDR ATOM

```
(defmacro swapf (x y)
  (let ((temp (gensym)))
    `(let ((,temp ,x))
      (setf ,x ,y) (setf ,y ,temp))))
```

```
[1]> (defparameter temp '(11 22 33))
TEMP
```

```
[2]> (swapf (first temp) (third temp))
11
```

```
[3]> temp
(33 22 11)
```

CAR EQUAL
CONS
CDR ATOM

Анафорические макросы

```
(format t "~a! = ~a~%"
```

```
5
```

```
(funcall #'(lambda (n)
```

```
(if (> n 1)
```

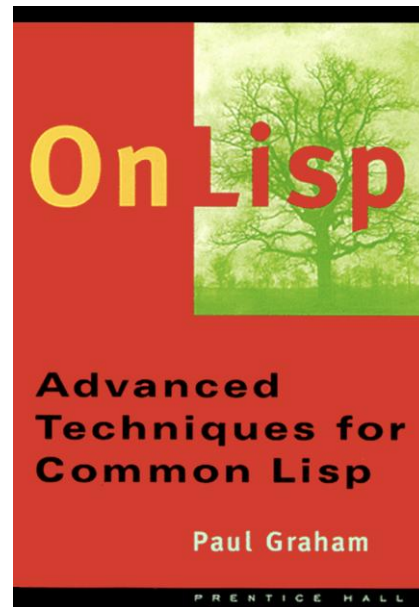
```
(* n (??? (- n 1)))
```

```
1))
```

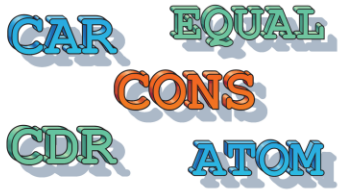
```
5))
```

↑
что нужно написать тут,
чтобы вызвать эту анонимную
функцию?

5! = 120



В книге Пола Грэма "О Лиспе"
подробно рассматриваются
различные способы
применения макросов



В Perl анонимная функция может сослаться на себя с помощью специального идентификатора `__SUB__`:

```
use 5.016;
use strict;
use warnings;

printf "%d! = %d\n", 5,
    sub {
        my $n = shift;
        return $n > 1 ?
            __SUB__->($n - 1) * $n : 1 }->(5);
```



вызов текущей функции

5! = 120

CAR EQUAL
CONS
CDR ATOM

```
(defun Y (fn arg)
  (funcall fn fn arg))
```



функция fn вызывается, и ей
передается ссылка
на саму эту функцию

```
[1]> (Y #'(lambda (fn n)
           (if (> n 1)
               (* n (funcall fn fn (- n 1)))
               1))
      5)
```

CAR EQUAL
CONS
CDR ATOM

```
(defmacro alambda (params &body body)
  `(labels ((self ,params ,@body))
    #'self))
```

```
(format t "~a! = ~a~%"
  5
  (funcall (alambda (n)
    (if (> n 1)
      (* n (self (- n 1)))
      1))
    5))
```

5! = 120

CAR EQUAL
CONS
CDR ATOM

Функциональное программирование: базовый курс

Лекция 7

Макросы в языке Lisp

Макрос LOOP

CAR EQUAL
CONS
CDR ATOM

```
[1]> (dotimes (i 3) (print i))
```

0

1

2

NIL

```
[2]> (loop for i below 3 do (print i))
```

0

1

2

NIL

CAR EQUAL
CONS
CDR ATOM

```
[1]> (loop for i from 1 below 3 do (print i))
```

1

2

NIL

```
[2]> (loop for i from 1 to 3 do (print i))
```

1

2

3

NIL

CAR EQUAL
CONS
CDR ATOM

```
[1]> (loop repeat 3 do (print "*"))
```

```
"*"
```

```
"*"
```

```
"*"
```

```
NIL
```

CAR EQUAL
CONS
CDR ATOM

```
[1]> (loop for x from 3.5 to 6.5 do (print x))
```

3.5

4.5

5.5

6.5

NIL

```
[2]> (loop for x from 3.5 to 6.5 by 1.33 do (print x))
```

3.5

4.83

6.16

NIL

CAR EQUAL
CONS
CDR ATOM

Несколько предложений for

```
[1]> (loop for x from 1 to 3  
        for y from 2 to 6  
        do (format t "x: ~a, y: ~a~%" x y))
```

x: 1, y: 2

x: 2, y: 3

x: 3, y: 4

NIL

CAR EQUAL
CONS
CDR ATOM

```
[1]> (loop for x from 1 to 3 do  
      (loop for y from 2 to 4 do  
        (format t "x: ~a, y: ~a~%" x y)))
```

x: 1, y: 2

x: 1, y: 3

x: 1, y: 4

x: 2, y: 2

x: 2, y: 3

x: 2, y: 4

x: 3, y: 2

x: 3, y: 3

x: 3, y: 4

NIL

CAR EQUAL
CONS
CDR ATOM

```
[1]> (defparameter *lst* '(("a" 10) ("b" 20) ("c" 30) ("d" 40))  
LST
```

```
[2]> (dolist (i *lst*) (print i))  
("a" 10)  
("b" 20)  
("c" 30)  
("d" 40)  
NIL
```

CAR EQUAL
CONS
CDR ATOM

```
[1]> (defparameter *lst* '("a" 10) ("b" 20) ("c" 30) ("d" 40))  
LST
```

```
[2]> (loop for x in *lst* do (print x))  
("a" 10)  
("b" 20)  
("c" 30)  
("d" 40)  
NIL
```

CAR EQUAL
CONS
CDR ATOM

```
[1]> (defparameter *lst* '(("a" 10) ("b" 20) ("c" 30) ("d" 40)))
```

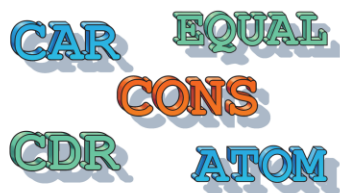
```
LST
```

```
[2]> (loop for x in *lst* by #'cddr do (print x))
```

```
("a" 10)
```

```
("c" 30)
```

```
NIL
```



Деконструкция списков в предложении for

```
[1]> (defparameter *lst* '(("a" 10) ("b" 20) ("c" 30) ("d" 40)))  
LST
```

```
[2]> (loop for (x y) in *lst* do  
      (format t "~a : ~a~%" x y))
```

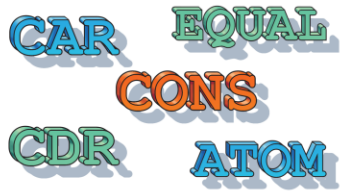
```
a : 10
```

```
b : 20
```

```
c : 30
```

```
d : 40
```

```
NIL
```

```
[1]> (defparameter *arr* #(10 20 30 40))
```

```
LST
```

```
[2]> (loop for x across *arr* do (print x))
```

```
10
```

```
20
```

```
30
```

```
NIL
```

```
[3]> (loop for ch across "test" do (print ch))
```

```
#\t
```

```
#\e
```

```
#\s
```

```
#\t
```

```
NIL
```

CAR EQUAL
CONS
CDR ATOM

Задание значения переменной цикла с помощью =

```
[1]> (loop for x = (random 100) do  
      (if (zerop x) (return) (print x)))
```

12

16

85

19

83

25

19

NIL

CAR EQUAL
CONS
CDR ATOM

Предложение then

```
[1]> (loop for x = 33 then (random 100) do  
      (if (zerop x) (return) (print x)))
```

33

79

12

73

1

56

NIL

CAR EQUAL
CONS
CDR ATOM

Предложения sum, count, minimize и maximize

```
[1]> (loop repeat 5 for x = (random 100)
      sum x into x-sum
      count x into x-count
      minimize x into x-min
      maximize x into x-max
      do (print x)
      finally (format t "~%~a ~a ~a ~a~%" x-sum x-count x-min x-max))
```

```
67
94
90
28
64
343 5 28 94
NIL
```

CAR EQUAL
CONS
CDR ATOM

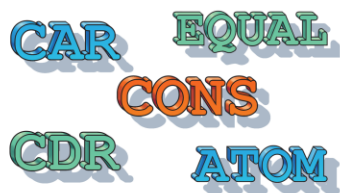
```
[1]> (loop repeat 50 for x = (random 100)
      count (oddp x) into x-odds
      finally (return x-odds))
```

28

CAR EQUAL
CONS
CDR ATOM

```
[1]> (loop repeat 5 for x = (random 1000) collect x)  
(374 549 968 283 105)
```

```
[2]> (loop for x across "test" collect x)  
(#\t #\e #\s #\t)
```



Последовательное присваивание значений переменным цикла

```
[1]> (loop repeat 10
      for x = 1 then y
      for y = 1 then (+ x y)
      collect x)
(1 1 2 4 8 16 32 64 128 256)
```

```
[2]> (loop repeat 10
      for x = 1 then y
      and y = 1 then (+ x y)
      collect x)
(1 1 2 3 5 8 13 21 34 55)
```

CAR EQUAL
CONS
CDR ATOM

Предложения collect и nconc

```
[1]> (loop for g = (gensym) for n in '(a b c) collect `(,n ,g))  
( (A #:G3068) (B #:G3069) (C #:G3070) )
```

```
[2]> (loop for g = (gensym) for n in '(a b c) nconc `(,n ,g))  
(A #:G3084 B #:G3085 C #:G3086)
```


CAR EQUAL
CONS
CDR ATOM

```
[1]> (loop repeat 5 for x = (random 100)
      when (oddp x) do (print x))
```

11

47

NIL

```
[2]> (loop repeat 5 for x = (random 100)
      unless (oddp x) do (print x))
```

32

64

12

0

NIL

CAR EQUAL
CONS
CDR ATOM

```
[1]> (defparameter *h* (hash-table-from-list  
      '("a" 42 "b" 21 "c" 77) :test #'equal))
```

```
*H*
```

```
[2]> (loop for n in '("a" "f" "t" "c")  
      when (gethash n *h*) collect it)
```

```
(42 77)
```

CAR EQUAL
CONS
CDR ATOM

```
[1]> (defparameter *h* (hash-table-from-list  
      '("a" 42 "b" 21 "c" 77) :test #'equal))
```

H

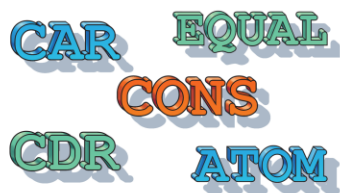
```
[2]> (loop for k being the hash-keys in *h*  
      using (hash-value v) do  
      (format t "~a: ~a~%" k v))
```

c: 77

b: 21

a: 42

NIL



```
[1]> (loop for x in '(1 3 4 5 7) always (oddp x))  
NIL
```

```
[2]> (loop for x in '(1 3 5 7) always (oddp x))  
T
```

```
[3]> (loop for x in '(1 3 5 7) never (evenp x))  
T
```

```
[4]> (loop for x in '(1 3 4 5 7) thereis (evenp x))  
T
```

CAR EQUAL
CONS
CDR ATOM

Функциональное программирование: базовый курс

Лекция 7

Макросы в языке Lisp

Особенности разработки макросов: повторные вычисления аргументов

CAR EQUAL
CONS
CDR ATOM

```
(defmacro mid-1 (x y z)
  `(cond
    ((or (<= ,y ,x ,z) (<= ,z ,x ,y)) ,x)
    ((or (<= ,x ,y ,z) (<= ,z ,y ,x)) ,y)
    (t ,z)))
```

```
[1]> (mid-1 10 30 20)
```

20

```
[2]> (mid-1 (random 100) (random 100) (random 100))
```

16

CAR EQUAL
CONS
CDR ATOM

```
[1]> (mid-1 (random 100) (random 100) (random 100))  
33
```

```
[2]> (macroexpand '(mid-1 (random 100) (random 100) (random 100)))  
(IF (OR (<= (RANDOM 100) (RANDOM 100) (RANDOM 100))  
        (<= (RANDOM 100) (RANDOM 100) (RANDOM 100)))  
    (PROGN (RANDOM 100))  
    (COND  
      ((OR (<= (RANDOM 100) (RANDOM 100) (RANDOM 100))  
           (<= (RANDOM 100) (RANDOM 100) (RANDOM 100)))  
       (RANDOM 100))  
      (T (RANDOM 100))))
```

T

CAR EQUAL
CONS
CDR ATOM

```
(defmacro mid-1.5 (x y z)
  `(let (($z ,z) ($y ,y) ($x ,x))
    (cond
      ((or (<= $y $x $z) (<= $z $x $y)) $x)
      ((or (<= $x $y $z) (<= $z $y $x)) $y)
      (t $z))))
```

```
[1]> (mid-1.5 10 30 20)
```

```
20
```


CAR EQUAL
CONS
CDR ATOM

```
(defmacro mid-2 (x y z)
  (let (($x (gensym)) ($y (gensym)) ($z (gensym)))
    `(let ((,$x ,x) (,$y ,y) (,$z ,z))
      (cond
        ((or (<= , $y , $x , $z) (<= , $z , $x , $y)) , $x)
        ((or (<= , $x , $y , $z) (<= , $z , $y , $x)) , $y)
        (t , $z))))))
```

```
[1]> (mid-2 10 30 20)
```

```
20
```

CAR EQUAL
CONS
CDR ATOM

Макрос with-gensyms-

```
(defmacro with-gensyms- (names &body body)
  `(let , (loop for n in names collect `(,n (gensym)))
    ,@body))
```

```
[1]> (macroexpand '(with-gensyms- ($x $y $z) (print $x)))
(LET (($X (GENSYM)) ($Y (GENSYM)) ($Z (GENSYM)))
  (PRINT $X))
```

T

CAR EQUAL
CONS
CDR ATOM

Макрос mid с использованием with-gensyms-

```
(defmacro mid-3 (x y z)
  (with-gensyms- ($x $y $z)
    `(let ((, $x ,x) (, $y ,y) (, $z ,z))
      (cond
        ((or (<= , $y , $x , $z) (<= , $z , $x , $y)) , $x)
        ((or (<= , $x , $y , $z) (<= , $z , $y , $x)) , $y)
        (t , $z))))))
```

CAR EQUAL
CONS
CDR ATOM

Макрос mid с использованием with-gensyms-init

```
(defmacro mid-4 (x y z)
  (with-gensyms-init ($x $y $z) (x y z)
    `(cond
      ((or (<= , $y , $x , $z) (<= , $z , $x , $y)) , $x)
      ((or (<= , $x , $y , $z) (<= , $z , $y , $x)) , $y)
      (t , $z))))
```

CAR EQUAL
CONS
CDR ATOM

Макрос mid с использованием with-gensyms-init

```
(defmacro mid-4 (x y z)
  (with-gensyms-init ($x $y $z) (x y z)
    `(cond
      ((or (<= , $y , $x , $z) (<= , $z , $x , $y)) , $x)
      ((or (<= , $x , $y , $z) (<= , $z , $y , $x)) , $y)
      (t , $z))))
```

```
(with-gensyms- ($x $y $z)
  `(let ((, $x , x) (, $y , y) (, $z , z))
```

CAR EQUAL
CONS
CDR ATOM

```
(defmacro with-gensyms-init (names init-vals &body body)
  `(with-gensyms- (,@names)
    `(let ( список_инициализации_временных_переменных )
      ,,@body)))
```

CAR EQUAL
CONS
CDR ATOM

Двойные обратные кавычки

```
[1]> (defparameter G250 42)  
G250
```

```
[2]> (defparameter n 'G250)  
N
```

```
[3]> `(,n)  
(G250)
```

```
[4]> ``(,,n)  
`(,G250)
```

```
[5]> (eval ``(,,n))  
(42)
```

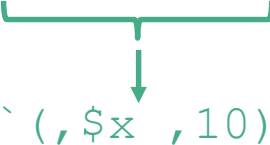
``(,,n) => `(,G250) => (42)

первое раскрытие макроса второе раскрытие макроса

CAR EQUAL
CONS
CDR ATOM

Макрос with-gensyms-init

```
(defmacro with-gensyms-init (names init-vals &body body)
  `(with-gensyms- (,@names)
    `(let (,,@(loop for v in init-vals for n in names
                    collect `(,n ,v)))
      ,,@body)))
```



```
  `(, $x , 10)
```

```
[1]> (macroexpand '(with-gensyms-init ($x $y $z) (10 30 20) (print $x)))
```

```
(LET (($X (GENSYM)) ($Y (GENSYM)) ($Z (GENSYM)))
  `(LET (, `(, $X , 10) , `(, $Y , 30) , `(, $Z , 20))
    , (PRINT $X)))
```


CAR EQUAL
CONS
CDR ATOM

Макрос mid с использованием with-gensyms-init

```
(defmacro mid-4 (x y z)
  (with-gensyms-init ($x $y $z) (x y z)
    `(cond
      ((or (<= , $y , $x , $z) (<= , $z , $x , $y)) , $x)
      ((or (<= , $x , $y , $z) (<= , $z , $y , $x)) , $y)
      (t , $z))))
```

```
[1]> (mid-4 (+ 4 2) 8 2)
6
```

```
[2]> (macroexpand '(mid-4 (+ 4 2) 8 2))
(LET ((#:G671 (+ 4 2)) (:G672 8) (:G673 2))
  (COND ((OR (<= #:G672 #:G671 #:G673) (<= #:G673 #:G671 #:G672)) #:G671)
        ((OR (<= #:G671 #:G672 #:G673) (<= #:G673 #:G672 #:G671)) #:G672)
        (T #:G673)))
```

T

CAR EQUAL
CONS
CDR ATOM

Макрос mid с использованием once-only

```
(defmacro mid-5 (x y z)
  (once-only (x y z)
    `(cond
      ((or (<= ,y ,x ,z) (<= ,z ,x ,y)) ,x)
      ((or (<= ,x ,y ,z) (<= ,z ,y ,x)) ,y)
      (t ,z))))
```

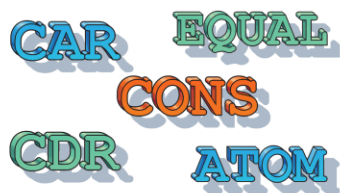
```
[1]> (mid-5 (+ 4 2) 8 2)
```

6

CAR EQUAL
CONS
CDR ATOM

Макрос once-only

```
(defmacro once-only ((&rest names) &body body)
  (let ((gensyms (loop for n in names collect (gensym))))
    `(with-gensyms-init (,@gensyms) (,@names)
      (let (,@(loop for n in names for g in gensyms
                    collect `(,n ,g)))
        ,@body))))
```



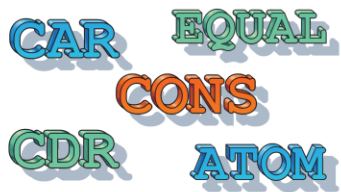
Раскрытие макроса once-only

```
[1]> (macroexpand-1 '(once-only (x y z) `(print ,x)))  
(WITH-GENSYMS-INIT (#:G728 #:G729 #:G730) (X Y Z)  
  (LET ((X #:G728) (Y #:G729) (Z #:G730))  
    `(PRINT ,X)))
```

T

```
[2]> (macroexpand-1 (macroexpand-1 '(once-only (x y z) `(print ,x))))  
  
(WITH-GENSYMS- (#:G732 #:G733 #:G734)  
  `(LET (, `(, #:G732 ,X) , `(, #:G733 ,Y) , `(, #:G734 ,Z))  
    , (LET ((X #:G732) (Y #:G733) (Z #:G734))  
      `(PRINT ,X))))
```

T



Раскрытие макроса once-only

```
[3]> (macroexpand-1 (macroexpand-1  
  (macroexpand-1 '(once-only (x y z) `(print ,x)))))
```

```
(LET ((#:G736 (GENSYM)) (#:G737 (GENSYM)) (#:G738 (GENSYM))) ; 1  
  `(LET (, `(, #:G736 ,X) , `(, #:G737 ,Y) , `(, #:G738 ,Z)) ; 2  
    , (LET ((X #:G736) (Y #:G737) (Z #:G738)) ; 3  
      `(PRINT ,X))) ; 4
```

T

```
[4]> (macroexpand '(mid 10 30 20))
```

```
(LET ((#:G744 10) (#:G745 30) (#:G746 20)) ; 1'  
  (COND ((OR (<= #:G745 #:G744 #:G746) (<= #:G746 #:G744 #:G745)) #:G744) ; 2'  
        ((OR (<= #:G744 #:G745 #:G746) (<= #:G746 #:G745 #:G744)) #:G745) ; 3'  
        (T #:G746))) ; 4'
```

T

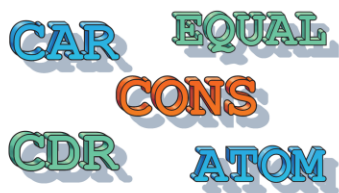
CAR EQUAL
CONS
CDR ATOM

Макрос mid с использованием once-only

```
(defmacro mid (x y z)
  (once-only (x y z)
    `(cond
      ((or (<= ,y ,x ,z) (<= ,z ,x ,y)) ,x)
      ((or (<= ,x ,y ,z) (<= ,z ,y ,x)) ,y)
      (t ,z))))
```

```
[1]> (mid (+ 4 2) 8 2)
```

6



Что мы узнали из этой лекции

- что такое макросы и метапрограммирование, чем макросы отличаются от функций
- как работает обратная кавычка и как вычислить аргумент внутри обратно заковыченного списка
- с какими основными трудностями приходится сталкиваться при разработке макросов: совпадения имен, неверный порядок вычисления аргументов, повторные вычисления аргументов
- как работает и для чего используется функция gensym
- что такое анафорический макрос
- какие возможности для организации циклов предоставляет в распоряжение программиста стандартный макрос loop
- как написать классические макросы with-gensyms и once-only