# ЛЕКЦИЯ 9
# Макросы

# Пример макроса. Swap

- чтобы поменять местами значения переменных a и b нужно:

```
(let ((c b))
   (set! b a)
   (set! a c))
```

- чтобы не писать подобный фрагмент каждый раз, определим макрос swap:

```
(define-syntax swap
   (syntax-rules ()
         ((swap a b)
          (let ((c b))
             (set! b a)
             (set! a c))
         )))
```

# Пример макроса. Swap

- как работает swap?

> (define first 5)

> (define second 'a)

> (swap first second)

> first  ==> 'a

> second ==> 5

> c ==> ошибка!

- при подстановке c заменяется на временное имя:

> (define c 10)

> (swap first c)

> first ==> 10

> c ==> 'a

- работает!

# Пример макроса. Swap

- возможный результат подстановки (swap first c):

```
(let ((__generated_symbol_1 c))
  (set! c first)
  (set! first __generated_symbol_1))
```

- описывая макрос, мы указали имя:

```
(define-syntax swap
```

- указали тип применяемого преобразования:

```
(syntax-rules ()
```

- указали правило, содержащее образец:

```
((swap a b)
```

- и шаблон:

```
(let ((c b)) (set! b a) (set! a c))
        )))
```

# Swap c define

- перепишем swap без let, чтобы убедиться, что гигиену обеспечивает syntax-rules, а не let:

```
(define-syntax swap2
    (syntax-rules ()
          ((swap2 a b)
           (begin (define c b)
              (set! b a)
              (set! a c))
          )))
> (define c 5)
> (define x 1)
> (define y 2)
> (swap2 x y)
> (swap2 x c)
> (display (list x y c)) ==> (5 1 2)
```

# Итог примера

- Макрос это:
  - специальным образом описанная трансформация кода
  - трансформация, осуществляемая без вычисления кода
  - трансформация, не использующая данные времени выполнения
- Другой пример cond-set!

```
(cond-set! (> test 4) var 15)
```

- при подстановке даёт:

```
(cond ((> test 4) (set! var 15)))
```

- не вычисляет свои аргументы!

# Макроподстановка

- Приблизительно подстановка происходит так
  1) сопоставитель находит вхождение имени макроса в тексте
  2) сопоставитель перебирает образцы, описанные в макросе
  3) если удаётся сопоставить с образцом, в шаблон подставляются параметры макровызова
  4) конфликтующие имена из шаблона заменяются на сгенерированные символы
  5) получившийся код вставляется на место макровызова

# Когда уместно использовать макрос?

- Используйте макросы для изменения порядка вычислений (определения собственных спецформ). Например:
  1) условные вычисления (cond, case, and, or)
  2) циклы (do)
  3) связывания (let, let*)
  4) не вычисляемые имена (=>)

# Когда неуместно использовать макрос?

- Всегда, когда без него можно обойтись!
  1) Макросы в отличие от функций не являются объектами первого класса.
  2) Макросы затрудняют отладку.

# cond-set!

- Вернёмся к примеру:

```
(cond-set! (> test 4) var 15)
```

- Можно ли описать cond-set! функцией?

```
(define (cond-set! test variable value)
        (cond (test (set! variable value))))
```

- не работает!

```
> (define a 5)
> (define b 10)
> (cond-set! (< a b) a b)
> a ==> 5
```

- меняется значение локальной переменной функции!
- Макрос уместен.

# Специальные формы и макросы

- Примитивных (настоящих) спецформ мало:
  - lambda
  - if
  - quote
  - set!

- Все остальные можно описать макросами.
- Хотя в трансляторе они могут быть реализованы напрямую.

# Специальные формы и макросы

- and как макрос:

```
(define-syntax and
  (syntax-rules ()
        ((and) #t)
        ((and test) test)
        ((and test1 test2 ...)
              (if test1 (and test2 ...) #f))))
```

# Системы макросов в Scheme

- Разные реализации Scheme поддерживают разные системы макросов
  - defmacro (унаследована от Лиспа)
  - syntax-rules (стандартная R5RS, «гигиеничная»)
  - syntax-case
  - syntax-table
  - …
- Рассмотрим syntax-rules (она есть в Racket!)

# Характеристики системы syntax-rules

- Гигиена. Макрос не портит окружения, в которых происходит подстановка.
- Прозрачность ссылок. Окружение, в котором происходит подстановка не портит макрос.
- Язык образцов используется для описания структуры макрокоманд.
- Закрытость. Макросистема отделена от Scheme. Во время подстановки нельзя запустить какой-либо код.

# Гигиеничные макросы

- Все локальные переменные макроса автоматически переименовываются перед подстановкой, для того чтобы предотвратить конфликт имен.
- Пример: swap. Локальное имя c не конфликтует с c из окружения, в котором происходит подстановка.
- Гигиеничность означает, что вызов макроса не может неожиданно повлиять на связывания. Ожидаемые изменения возможны:

```
(define-syntax shadow
   (syntax-rules () ((shadow a b)
                     (begin (define a 5) b))))
> (define test 7)
> (shadow test test) ==> (begin (define test 5) test) ==> 5
> test ==> 5      ожидаемое изменение с таким вызовом
```

# Прозрачность ссылок

- Все свободные переменные из шаблона макроса имеют связывания из окружения, в котором описан макрос.
- Пример:

```
(define-syntax dec
   (syntax-rules () ((dec arg)
                     (set! arg (- arg 1)))))
> (define a 1)
> (define b 1)
> (let ((- +)) (dec a) (set! b (- b 1)))
> a ==> 0  минус «старый»!
> b ==> 2  минус «новый»!
```

# Язык образцов

- Язык образцов в syntax-rule прост:
  - образец – это комбинация (список)
  - первый элемент – имя макроса
  - литералы, а также списки, векторы представляют сами себя
  - бывают спецсимволы и многоточия
  - остальное – переменные образца.
- Образец сопоставится:
  - если литералы, списки, векторы совпадают точно
  - если каждая переменная образца сопоставима одному подвыражению макрокоманды.

# Язык образцов. Пример

- Дан образец:

```
(let1 (name value) body)
```

- Макрокоманда:

```
(let1    (x (read))
         (cond ((not x) (display "you said no"))))
```

- Образец сопоставится:
  - name = x
  - value = (read)
  - body = (cond ((not x) (display "you said no")))

# Язык образцов. Ещё пример

- Дан образец:

```
(contrived #((first . rest) #(3 any)))
```

- Макрокоманда:

```
(contrived #((1 2 3 4 5) #(3 ’(foo))))
```

- Образец сопоставится:
  - first = 1
  - rest = (2 3 4 5)
  - any = ’(foo)

# Язык шаблонов

- Шаблон – это код на Scheme, определяющий вместе с образцом, что будет подставлено.
  - литералы, а также списки, векторы представляют сами себя
  - символы, не встречающиеся в образце, представляют сами себя
  - остальное – переменные образца.
- При подстановке происходит замена внутри шаблона переменных образца, на то, с чем они сопоставились.

# Язык шаблонов. Пример

- Образец:

```
(let1 (name value) body)
```

- Шаблон:

```
(let ((name value)) body)
```

- Макрокоманда:

```
(let1   (x (read))
        (cond ((not x) (display "you said no"))))
```

- Подстановка:

```
(let    ((x (read)))
        (cond ((not x) (display "you said no"))))
```

# Многоточие

- Если за переменной образца следует … она сопоставляется с несколькими подвыражениями макрокоманды.
- Пример образца:

```
(dotimes count body ...)
```

- Пример макрокоманды:

```
(dotimes 5 (set! x (+ x 1)) (display x))
```

- Сопоставление:

```
count = 5
body ... = (set! x (+ x 1)) (display x)
```

# Многоточие. Пример

- В шаблоне вхождение переменной образца с многоточием заменяется на все сопоставленные ей подвыражения
- Пример:

```
(define-syntax dotimes
  (syntax-rules ()
    ((dotimes count body ...)
     (let loop ((counter count))
       (cond ((> counter 0) (begin body ...
         (loop (- counter 1)))))))))
```

```
> (define x 5)
> (dotimes 5 (set! x (+ x 1)) (display x))
```

- в результате код, печатающий 678910

# Многоточие. Пример

- В шаблоне вхождение переменной образца с многоточием заменяется на все сопоставленные ей подвыражения
- в результате код, печатающий 678910

```
(let loop ((counter 5))
  (cond ((> counter 0)
    (begin (set! x (+ x 1)) (display x)
      (loop (- counter 1))))))
```

# Многоточие. Пример

- Многоточие в шаблоне после подвыражения с переменной образца с многоточием вызывает многократную подстановку подвыражения с каждым сопоставлением переменной.
- Пример:

```
(define-syntax thunkify
  (syntax-rules ()
        ((thunkify body ...)
         (list (lambda () body) ...))))
```

- Макрокоманда:

```
(thunkify 5 (* x x))
```

- Постановка

```
(list (lambda () 5) (lambda () (* x x)))
```

# Многоточие. Ещё пример

```
(define-syntax update-if-true!
   (syntax-rules ()
         ((update-if-true! (condition variable) ...)
          (begin (let ((test condition))
                   (cond (test (set! variable test)))) ...))))
```

- Макрокоманда:

```
(update-if-true! ((> x 5) x-is-big) ((zero? y) y-is-zero))
```

- Постановка

```
(begin
    (let ((test (> x 5)))
     (cond (test (set! x-is-big test))))
    (let ((test (zero? y)))
     (cond (test (set! y-is-zero test)))))
```

# Многоточия. Пример

```
(define-syntax quoted-append
   (syntax-rules ()
         ((quoted-append (arg ...) ...)
          (quote (arg ... ...)))))
```

- Макрокоманда:

```
(quoted-append (1 2 3) (a b c) (+ x y))
```

- Постановка

```
’(1 2 3 a b c + x y)
```

# Группировка пар образец-шаблон

- Одно макроопределение может содержать несколько описаний пар образец-шаблон:

```
(define-syntax and
  (syntax-rules ()
        ((and) #t)
        ((and test) test)
        ((and test1 test2 ...)
              (if test1 (and test2 ...) #f))))
```

пары просматриваются сверху вниз!

# Спецсимволы в образцах

- Пусть мы ходим, чтобы макрокоманда:

```
(implications (a => b) (c =>d) (e =>f))
```

- давала подстановку:

```
(begin (cond (a b)) (cond (c d)) (cond (e f)))
```

- проблема в том, как указать, что => -- не переменная!

```
(syntax-rules ()
((implications (condition => consequent) ...)
(begin (cond (condition consequent)) ...)))
```

- приводит к тому, что при макровызове

```
(implications (test =. (set! testp #t)))
```

- получается сопоставление

```
condition = test        => = =.
consequent =  (set! testp #t)
```

# Спецсимволы в образцах

- Спецсимволы следует указывать в списке после syntax-rules:

```
(syntax-rules (=>)
((implications (condition => consequent) ...)
(begin (cond (condition consequent)) ...)))
```

- теперь при макровызове

```
(implications (test =. (set! testp #t)))
```

- не будет сопоставления, так как спецсимвол сопоставляется только сам с собой
- другой случай, когда сопоставление невозможно:

```
(let ((=> 5)) (implications (foo => bar)))
```

=> разные!

# Итоги по спецсимволам в образцах

- Спецсимволы из макроопределения относятся к окружению макроопределения.
- Символы из макрокоманды относятся к окружению, в котором встретилась команда.
- Спецсимвол сопоставится, только если он не перекрыт в окружении макрокоманды.

# Итоги по syntax-rules

- Сочетание гигиеничности с прозрачностью ссылок делает макросы системы syntax-rules лексически безопасными.
  - макрос не портит неожиданно окружение, в котором происходит подстановка
  - окружение, в котором происходит подстановка, не портит макрос
  - действует «правило наименьшего удивления»

# Вспомним dotimes

```
(define-syntax dotimes
  (syntax-rules ()
    ((dotimes count body ...)
     (let loop ((counter count))
       (cond ((> counter 0) (begin body ...
         (loop (- counter 1))))))))))
```

- что если?

```
> (define counter 5)
> (dotimes 5 (set! counter (+ counter 1)) (display counter))
```

- в результате код, печатающий 678910
- макрос безопасный – counter’ы разные.

# Рекомендации по стилю макроопределений

- сортируйте пары образец-шаблон (сначала короткие, как в and)
- имя макроса в образце можно не писать, а ставить прочерк (так проще переименовывать макросы)

```
(define-syntax dec
   (syntax-rules () (( _ arg)
                     (set! arg (- arg 1)))))
```

# Итоги лекции 9

- Мы рассмотрели не всё. Бывают:
  - cps-style макросы
  - обход гигиеничности (Petrofsky’s find-identifier)
  - синтезирование символов
- Того, что рассмотрено, достаточно.