Метапрограммирование

Препроцессор C (CPP)

```
#define SWAP(a, b, type) { type __tmp_c; c = b; b = a; a = c; }

int main()
{
   int a = 3;
   int b = 5;
   printf("a is %d and b is %d\n", a, b);
   SWAP(a, b, int);
   printf("a is now %d and b is now %d\n", a, b);
   return 0;
}
```

Препроцессор C (CPP)

SWAP(a, b, int)

{int __tmp_c; __tmp_c = b; b = a; a = __tmp_c;}.

Макрос, возвращающий минимум из двух значений

```
#define MIN(x, y) ((x) > (y) ? (y) : (x))

MIN(27, b=32)

27 > b = 32 ? b = 32 : 27
```

Макрос, возвращающий минимум из двух значений

MIN(do_long_calc(), do_long_calc2())

((do_long_calc()) > (do_long_calc2()) ? (do_long_calc2()) : (do_long_calc()))

Макрос обмена значений в Scheme

```
;;Определить SWAP как макрос
(define-syntax SWAP
  ;;Мы используем метод syntax-rules для создания макроса
  (syntax-rules ()
    ;;Rule Group
    (
      ;;Это шаблон, соответствие которому мы проверяем
      (SWAP a b)
      ;;Во что мы его преобразовываем
      (let (
            (c b))
        (set! b a)
        (set! a c)))))
(define first 2)
(define second 9)
(SWAP first second)
(display "first is: ")
(display first)
(newline)
(display "second is: ")
(display second)
(newline)
```

Возможное преобразование макроса перестановки значений

```
(define first 2)
(define second 9)
(let
   (
     (__generated_symbol_1 second))
   (set! second first)
   (set! first __generated_symbol_1))
(display "first is: ")
(display first)
(newline)
(display "second is: ")
(display second)
(newline)
```

```
;;Определить макрос
(define-syntax at-compile-time
  ;;x - это синтаксический объект для преобразования
  (lambda (x)
    (syntax-case x ()
      (
        ;;Шаблон, аналогичный шаблону syntax-rules
        (at-compile-time expression)
        ;;with-syntax позволяет нам создавать синтаксические объекты
        ;;динамически
        (with-syntax
          (
            ;это - создаваемый нами синтаксический объект
            (expression-value
              ;после вычисления выражения преобразуем его в синтаксический объект
              (datum->syntax
                ;домен syntax
                (syntax at-compile-time)
                ;отметить значение кавычками, поскольку оно является литеральным значением
                (list 'quote
                ;вычислить значение преобразования
                  (eval
                    ;;преобразовать выражение из синтаксического представления
                    ;;в список
                    (syntax-object->datum (syntax expression))
                     )))))
          ;;Просто возвратить сгенерированное значение как результат
          (syntax expression-value))))))

(define a
  ;;преобразовать в 5 во время компиляции
  (at-compile-time (+ 2 3)))
```

Создание таблицы квадратных корней в Scheme

```
(define sqrt-table
  (at-compile-time
    (list->vector
      (let build
        (
          (val 0))
        (if (> val 20)
          '()
          (cons (sqrt val) (build (+ val 1))))))))

(display (vector-ref sqrt-table 5))
(newline)
```

Макрос для создания таблиц преобразования во время компиляции

```
(define-syntax build-compiled-table
  (syntax-rules ()
    (
      (build-compiled-table name start end default func)
      (define name
        (at-compile-time
          (list->vector
            (let build
              (
                (val 0))
              (if (> val end)
                '()
                (if (< val start)
                  (cons default (build (+ val 1)))
                  (cons (func val) (build (+ val 1))))))))))))

(build-compiled-table sqrt-table 5 20 0.0 sqrt)
(display (vector-ref sqrt-table 5))
(newline)
```

Написание макросов syntax-case в Scheme

```
(define-syntax macro-name
  (lambda (x)
    (syntax-case x (другие ключевые слова, если имеются)
      (
        ;;Первый шаблон
        (macro-name macro-arg1 macro-arg2)
        ;;Расширение макроса (одна или несколько форм)
        ;;(syntax - это зарезервированное слово)
        (syntax (расширение макроса находится здесь))
      )
      (
        ;;Второй шаблон - версия с одним аргументом
        (macro-name macro-arg1)
        ;;Расширение макроса
        (syntax (расширение макроса находится здесь))
      )
 )))
```

Макрос для определения расширенной версии оператора if

```
;;определить my-if как макрос
(define-syntax my-if
  (lambda (x)
    ;;установить, что "then" и "else" - это ключевые слова
    (syntax-case x (then else)
      (
        ;;шаблон для соответствия
        (my-if condition then yes-result else no-result)
        ;;преобразователь
        (syntax (if condition yes-result no-result))
       )
)))


(my-if  (> a b)  then     a      else    b)
   /       /      /       /       /      /
   /       /      /       /       /      /
   v       v      v       v       v      v
(my-if condition then yes-result else no-result)
```

Определение вашего собственного макроса swap!

```
;;Определить новый макрос
(define-syntax swap!
  (lambda (x)
    ;;здесь мы не используем ключевых слов
     (syntax-case x ()
       (
         (swap! a b)
         (syntax
           (let ((c a))
             (set! a b)
             (set! b c)))
       )
)))

(define a 1)
(define b 2)
(swap! a b)
(display "a is now ")(display a)(newline)
(display "b is now ")(display b)(newline)
```

Не работающий макрос определения математических констант

```
(define-syntax with-math-defines
  (lambda (x)
    (syntax-rules x ()
      (
        (with-math-defines expression)
        (syntax
          (let ( (pi 3.14) (e 2.71828) )
              expression))
      )
)))
```

Работающий макрос определения математических констант

```
(define-syntax with-math-defines
  (lambda (x)
    (syntax-case x ()
      (
       ;;Шаблон
       (with-math-defines expression)

       ;;with-syntax определяет новые переменные шаблона
       (with-syntax
         (
          (expr ;;новая переменная шаблона
           ;;преобразовать выражение в синтаксический объект
           (datum->syntax
            ;;syntax - местная магия
            (syntax k)
            ;;выражение для преобразования
            `(let ( (pi 3.14) (e 2.72))
                ;;Вставить код для переменной шаблона "expression"
                ;;сюда.
                ,(syntax->datum (syntax expression))))))
         ;;Использовать новую созданную переменную шаблона "expr"
         ;;как конечное выражение
         (syntax expr))
       )
  )))
(with-math-defines (* pi e))
```