LAB

HOMEWORK

HW 01

HW 01

```
(make-change (- total biggest) biggest) ; ((x ...) ...)
(make-change total (- biggest 1))       ; ((x ...) ...)
```

HW 01

```
(map (lambda (x) (cons biggest x))
     (make-change (- total biggest) biggest)) ; ((x ...) ...)
(make-change total (- biggest 1))          ; ((x ...) ...)
```

HW 01

```
(append
 (map (lambda (x) (cons biggest x))
      (make-change (- total biggest) biggest))
 (make-change total (- biggest 1)))
```

HW 01

```scheme
(define (make-change total biggest)
  (cond ((= total 0) '(()))
        ((or (< total 0) (= biggest 0)) nil)
        (else (append
                (map (lambda (lst) (cons biggest lst))
                     (make-change (- total biggest) biggest))
                (make-change total (- biggest 1))))))
```

HW 02

Direct Recursion:

```scheme
(define (find n lst)
  (define (find-tail i n curr)
    (if (or (null? curr) (= n (car curr)))
        i
        (find-tail (+ i 1) n (cdr curr))))
  (find-tail 0 n lst))
```

HW 03

```
(cons a b)
```

1. search a
2. if failed, search b

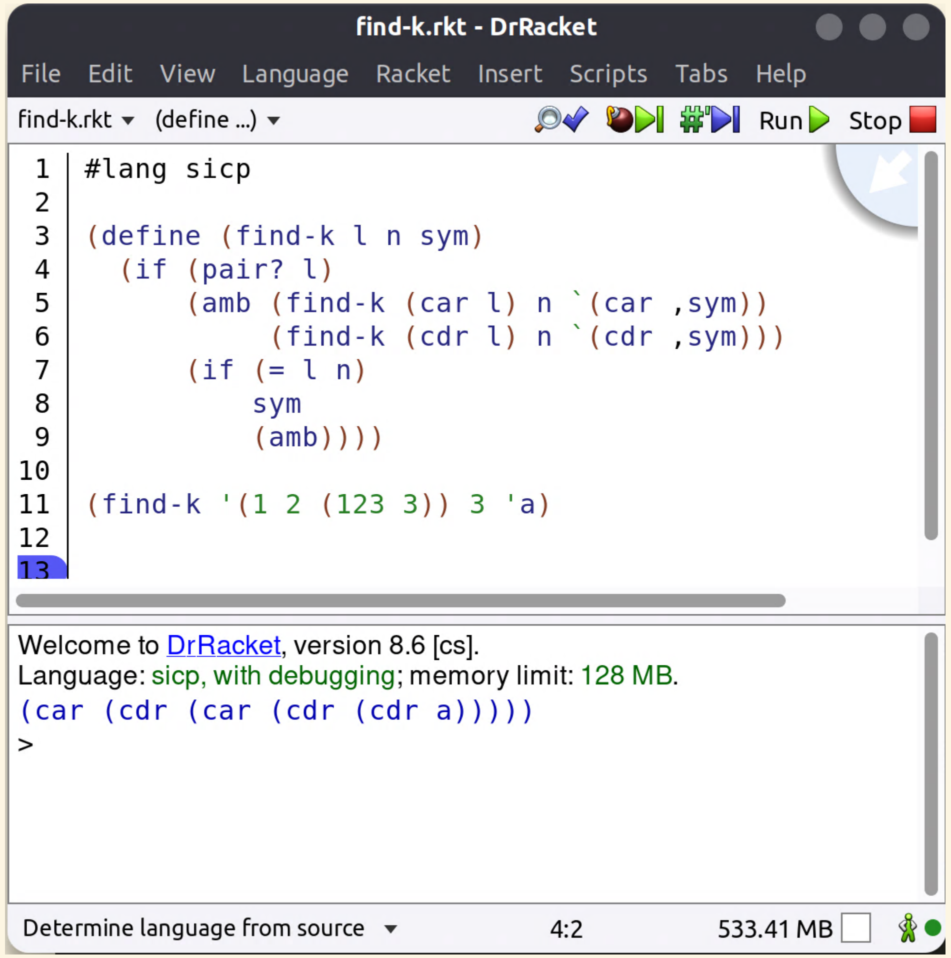
HW 03

HW 03

```
(define (find-nest n sym)
  (define (helper lst expr)
    (if (pair? lst)
      (let ((r1 (helper (car lst) (list 'car expr))))
            (if (null? r1)
                (helper (cdr lst) (list 'cdr expr))
                r1))
      (if (and (number? lst) (= n lst)) expr nil)))
  (helper (eval sym) sym))
```

HW 03

HW 03


find-k.rkt - DrRacket
File Edit View Language Racket Insert Scripts Tabs Help
find-k.rkt
(define ...)
Run
Stop

```
#lang sicp

(define (find-k l n sym)
  (if (pair? l)
      (amb (find-k (car l) n `(car ,sym))
           (find-k (cdr l) n `(cdr ,sym)))
      (if (= l n)
          sym
          (amb))))

(find-k '(1 2 (123 3)) 3 'a)
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: sicp, with debugging; memory limit: 128 MB.
(car (cdr (car (cdr (cdr a)))))
>
```

Determine language from source
4:2
533.41 MB

HW 03

See Structure and Interpretation of Computer Programs, chap 4.3

HW 04

```
(or (+ 1 2) #f 3)

(let ((t (+ 1 2)))
  (if t
      t
      (let ((t #f)))
        (if t
            t
            (let ((t 3))
              (if t
                  t
                  #f))))))
```

Why let ?

==> Only eval once

HW 04

HW 04

```
scm> (define (f x) (print x) (+ x 1))
f
scm> (my/or ((f 10) (f 10)))
10
11
```

HW 04

```
(define-macro (my/or operands)
  (cond
    ((null? operands) #f)
    ((null? (cdr operands)) (car operands))
    (else
      `(let ((t ,(car operands)))
         (if t
             t
             (my/or ,(cdr operands)))))))
```

HW 05

Solution1: Write 2 functions

HW 05

Solution1: Write 2 functions

```scheme
(define (remove lst indices curr)
  ; '(a b c d) -> '(a c)
  (cond ((or (null? lst) (null? indices)) lst)
        ((= curr (car indices))
         (remove (cdr lst) (cdr indices) (+ 1 curr)))
        (else (cons (car lst)
                    (remove (cdr lst) indices (+ 1 curr))))))

(define (replace lst vals indices curr)
  ; '(a b c d) -> '(a 2 c 4)
  (cond ((or (null? lst) (null? indices)) lst)
        ((= curr (car indices))
          (cons (car vals)
                (replace (cdr lst) (cdr vals) (cdr indices) (+ 1 c
        (else (cons (car lst)
                    (replace (cdr lst) vals indices (+ 1 curr)))))
```

HW 05

Solution1: Write 2 functions

```
(define-macro (k-curry fn args vals indices)
  `(lambda
    ,(remove args indices 0)
    ,(cons fn (replace args vals indices 0))))
```

HW 05

HW 05

```
scm> (append-vals '(a b c d) '(1 3) '(2 4) 0)

'(a
  (b . 2)
  c
  (d . 4))
```

HW 05

```
(define-macro (k-curry fn args vals indices)
  (let ((arg/vals (append-vals args indices vals 0)))
    (let ((args (filter symbol? arg/vals))
          (vals (map to-val arg/vals)))
      `(lambda ,args ,(cons fn vals)))))

(define (to-val x)
  (if (pair? x) (cdr x) x))
```

HW 05

```
(define (append-vals lst indices vals curr)
  (cond
    ((or (null? lst) (null? indices)) lst)
    ((= curr (car indices))
     (cons (cons (car lst) (car vals))
           (append-vals (cdr lst) (cdr indices) (cdr vals) (+ cur
    (else (cons (car lst)
                (append-vals (cdr lst) indices vals (+ curr 1))))))
```

HW 06

```
(let* ((a 1)
       (b a))
  b)

(let ((a 1))
  (let (b a))
    b))
```

HW 06

```
(let* ((a 1)
       (b a))
  b)

(let ((a 1))
  (let (b a))
    b))
```

```
(define-macro (let* bindings expr)
  (if (null? bindings)
      `(let ,bindings ,expr)
      `(let (,(car bindings))
        (let* ,(cdr bindings) ,expr))))
```

HW 07

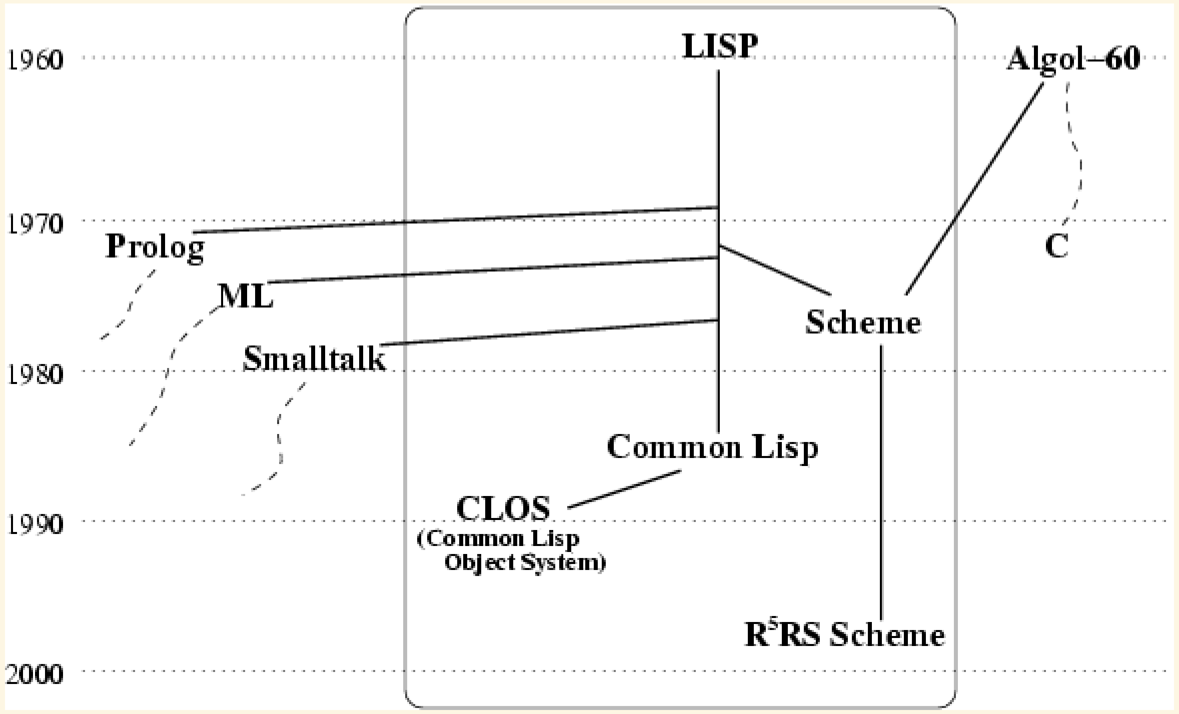
```
    '(a + 1 ...)
=>  `(+ a ,(infix-cal '(1 ...))

    '(a * 1 ...)
=>  (infix-cal '((* a 1) ...)
```

HW 07

```
(define (infix-cal expr)
  (if (pair? expr)
    (if (null? (cdr expr)) (infix-cal (car expr))
      (let ((left (infix-cal (car expr)))
            (operator (cadr expr))
            (right (cddr expr)))
        (if (eq? operator '+)
          (+ left (infix-cal right))
          (infix-cal
           (cons (* left (infix-cal (car right)))
                 (cdr right))))))
    (eval expr)))
```

HW 08


1960
1970
1980
1990
2000
LISP
Algol–60
Prolog
ML
Smalltalk
C
Scheme
Common Lisp
CLOS
(Common Lisp Object System)
R5RS Scheme

SCHEME - INVENTIONS

- proper tail call
- lexical scope with closure
- first class continuations
- **hygienic macro**(from R5RS)

HYGIENE PROBLEM

HYGIENE PROBLEM

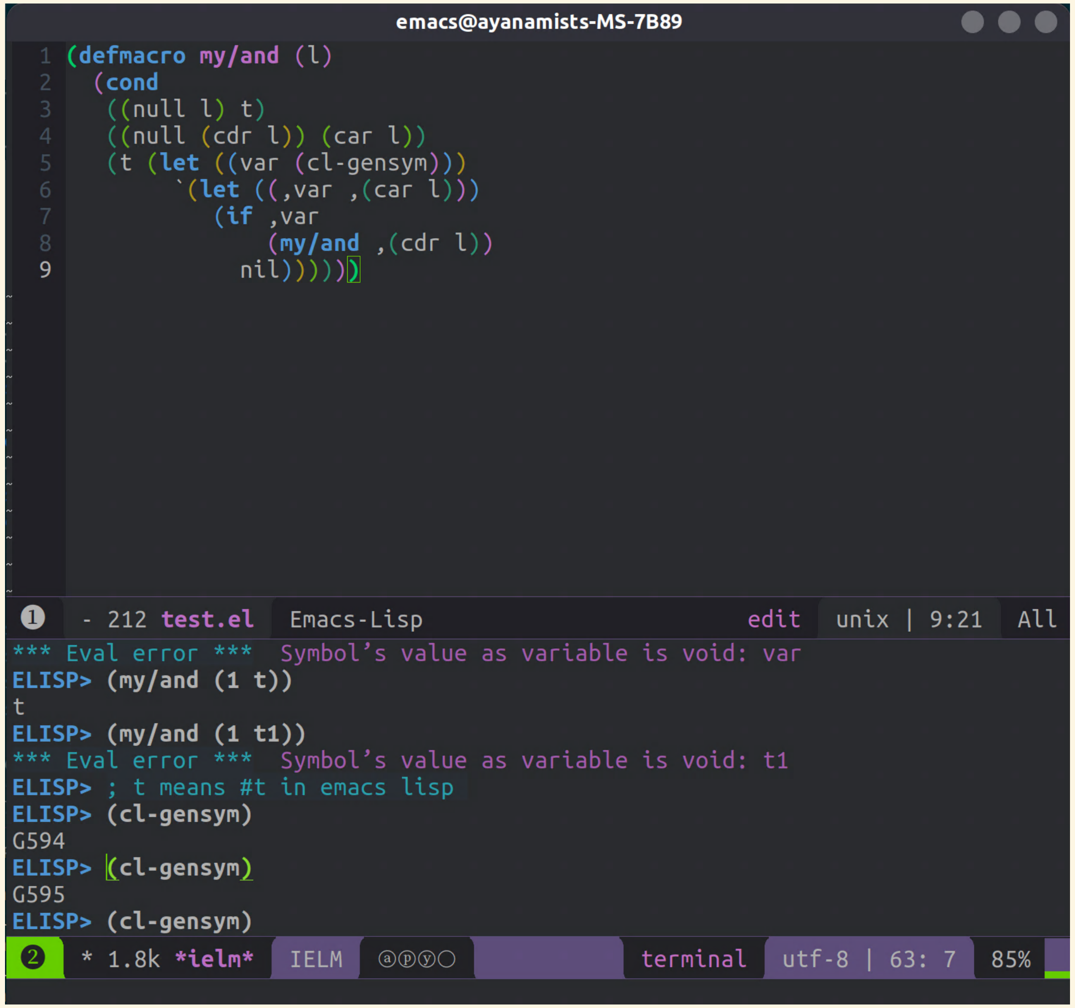
```
scm> (my/and (1 t))
1

scm> (and (1 t))
Error
```

HYGIENE PROBLEM

```
(let (t 1)
  (if t
      t  ; unexpected capture of t
      #f))
```

HYGIENE PROBLEM - SOLUTION 1

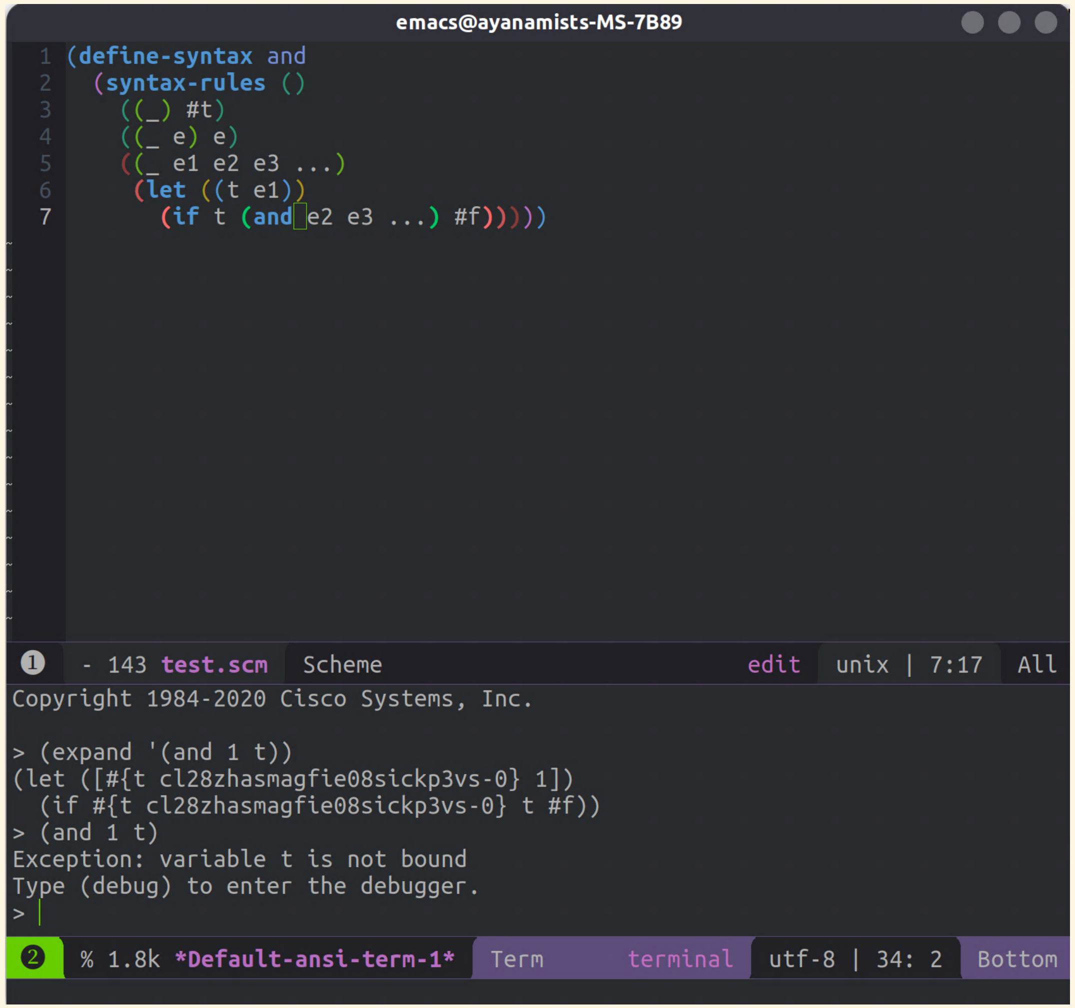


emacs@ayanamists-MS-7B89

```
(defmacro my/and (l)
  (cond
   ((null l) t)
   ((null (cdr l)) (car l))
   (t (let ((var (cl-gensym)))
        `(let ((,var ,(car l)))
           (if ,var
               (my/and ,(cdr l))
             nil))))))
```

```
1  - 212 test.el  Emacs-Lisp                              edit  unix | 9:21  All
*** Eval error ***  Symbol's value as variable is void: var
ELISP> (my/and (1 t))
t
ELISP> (my/and (1 t1))
*** Eval error ***  Symbol's value as variable is void: t1
ELISP> ; t means #t in emacs lisp
ELISP> (cl-gensym)
G594
ELISP> (cl-gensym)
G595
ELISP> (cl-gensym)
2  * 1.8k *ielm*  IELM  ⓐⓟⓨ○                terminal  utf-8 | 63: 7  85%
```

HYGIENE PROBLEM - SOLUTION 2



emacs@ayanamists-MS-7B89

```scheme
(define-syntax and
  (syntax-rules ()
    ((_) #t)
    ((_ e) e)
    ((_ e1 e2 e3 ...)
     (let ((t e1))
       (if t (and e2 e3 ...) #f)))))
```

```
❶  - 143 test.scm   Scheme                                   edit   unix | 7:17   All
```

```
Copyright 1984-2020 Cisco Systems, Inc.

> (expand '(and 1 t))
(let ([#{t cl28zhasmagfie08sickp3vs-0} 1])
  (if #{t cl28zhasmagfie08sickp3vs-0} t #f))
> (and 1 t)
Exception: variable t is not bound
Type (debug) to enter the debugger.
>
```

```
❷  % 1.8k *Default-ansi-term-1*   Term        terminal   utf-8 | 34: 2   Bottom
```