

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
 Department of Electrical Engineering and Computer Science
 6.001—Structure and Interpretation of Computer Programs
 Spring Semester, 2004

Quiz I

Closed Book – one sheet of notes

Separately, we have distributed an answer sheet. You may use the space on the exam booklet for whatever temporary work you find useful, but you **MUST** enter your answers into the answer sheet. **Only the answer sheet will be graded.** Each problem that requires an answer has been numbered. Place your answer at the corresponding number in the answer sheet.

Note that any procedures or code fragments that you write will be judged not only on correct function, but also on clarity and good programming practice.

The answer sheet asks for your section number and tutor. For your reference, here is the table of section numbers.

Section	Time	Room	Instructor	Tutor
R01	WF 9 AM	36-155	Berthold Horn	Joshua Tauber
R02	WF 10 AM	34-301	Randy Davis	David Ziegler
R03	WF 10 AM	36-155	Berthold Horn	Sam Daitch
R04	WF 10 AM	34-304	Ben Vandiver	Iuliu Vasilescu
R05	WF 11 AM	34-301	Randy Davis	Murali Vajapeyam
R06	WF 2 PM	26-322	Ben Vandiver	Iuliu Vasilescu/Murali Vajapeyam
R07	WF 12 PM	34-301	Daniela Rus	Bryan Russell/Joshua Tauber
R08	WF 1 PM	34-303	Barbara Cutler	Manish Jethwa
R09	WF 1 PM	34-301	Daniela Rus	Bryan Russell
R10	WF 2 PM	34-303	Barbara Cutler	Manish Jethwa/Sam Daitch
R11	WF 11 AM	26-314	Michael Collins	Kristina Foster/David Ziegler
R12	WF 12 PM	26-314	Michael Collins	Kristina Foster

Part 1: (12 points)

For each of the following expressions you are to (1) draw the box and pointer diagram corresponding to the list or pair structure created, and (2) write the printed result of evaluating the last expression in the sequence.

```
(define a (list 1 (cons 2 nil) 3))  
a
```

Question 1. Draw box and pointer for a

Question 2. Printed result

```
(define b (list 4 5))  
(define c (cons (cons b nil) (list 6 (list 7))))  
c
```

Question 3. Draw box and pointer for c

Question 4. Printed result

```
(define e (cons 8 9))  
(define d (cdr (list 7 e 10)))  
d
```

Question 5. Draw box and pointer for d

Question 6. Printed result

```
(define x (list 1 2))  
(define y (list 3 (list 4)))  
(define z (append x y))  
z
```

Question 7. Draw box and pointer for z¹

Question 8. Printed result

¹You can find the code for `append` on the last page of the exam booklet.

Part 2: (22 points)

Acme Industries has just won a UN contract to support a world-wide census of adult populations across nations, states within nations, counties within states, and other more finely resolved units as desired. Acme assigns Ben Bitdiddle to the task, who decides to use a tree structure to represent census data. In addition to the basic Scheme language, Ben also has available to him `map`, `filter`, and `fold-right` (a copy of these are available to you at the end of this exam booklet). Note that `fold-right` is sometimes referred to as `accumulate`.

Ben's data structure is shown below:

```
(define (make-unit name adults subunits)
  (list name adults subunits))
(define (unit-name unit) (car unit))
(define (unit-adults unit) (cadr unit))
(define (subunits unit) (caddr unit))
```

An example census, which is both incomplete and inconsistent, is shown below as `census1`.

```
(define census1
  (make-unit "usa" 0
    (list (make-unit "alaska" 1 nil)
          (make-unit "california" 2 nil)
          (make-unit "massachusetts" 0
            (list (make-unit "middlesex" 10 nil)
                  (make-unit "suffolk" 14 nil))))
          (make-unit "new york" 4 nil))))
```

Ben begins to implement some utility functions to go with his data representation. He decides to write `empty-unit` which is specified to create a copy of the full census tree structure, but having zero for the number of adults in that unit (and zero also for the number of adults in all smaller units within that unit). Complete the code for Ben:

```
(define (empty-unit unit)
  (make-unit (unit-name unit)
            0
            Q9-ANSWER))
```

Question 9. What code should be inserted in place of Q9-ANSWER?

Ben notices that the data he is given often appears inconsistent: the population of a particular unit does not always equal the sum of the population of the subunits within that unit. He checks with the UN, who tell him they didn't add things up; only the data for the "bottom-most" units in any branch of the tree are correct. For instance, the stated population for Middlesex County is assumed correct, and the population for Alaska is correct as well (since they have no subunits). However, the numbers for Massachusetts or for the USA are not correct as supplied by the UN.

Ben realizes this is a mess, and decides to write a procedure `update-unit` that will create a new version of a given census structure, with the correct addition performed on units.

```
(define (update-unit unit)
  (if (null? (subunits unit))
      unit
      (let ((new-subunits (map update-unit (subunits unit))))
        (make-unit (unit-name unit)
                   (fold-right Q10-ANSWER
                               Q11-ANSWER
                               Q12-ANSWER)
                   new-subunits))))

; Example
(define census2 (update-unit census1))
census2
=> ("usa" 31 (("alaska" 1 nil)
              ("california" 2 nil)
              ("massachusetts" 24 (("middlesex" 10 nil) ("suffolk" 14 nil)))
      ("new york" 4 nil)))
```

Question 10. What code should be inserted in place of Q10-ANSWER?

Question 11. What code should be inserted in place of Q11-ANSWER?

Question 12. What code should be inserted in place of Q12-ANSWER?

The UN wants to focus attention on “big” units, where a big unit is one that has a population bigger than some cutoff population. They ask Ben for a procedure that will create a census data structure that only includes those units and subunits that meet this criteria.

```
(define (big-units unit cutoff)
  (let ((big-subunits (filter Q13-ANSWER
                              Q14-ANSWER)))
    (if (or (> (unit-adults unit) cutoff)
          (not (null? big-subunits)))
        (make-unit (unit-name unit)
                   (unit-adults unit)
                   big-subunits)
        nil)))

; Example
(big-units census2 5)
;Value: ("usa" 31 ("massachusetts" 24 (("middlesex" 10 nil) ("suffolk" 14 nil))))
```

Question 13. What code should be inserted in place of Q13-ANSWER?

Question 14. What code should be inserted in place of Q14-ANSWER?

Part 3: (16 points)

For each of the following expressions or sequences of expressions, state the value returned as the result of evaluating the final expression in each set, or indicate that the evaluation results in an error. If the expression does not result in an error, also state the “type” of the returned value, using the notation from lecture. If the result is an error, state in general terms what kind of error (e.g. you might write “error: wrong type of argument to procedure”). If the evaluation returns a built-in procedure, write **primitive procedure**, and its **type**. If the evaluation returns a user-created procedure, write **compound procedure**, and also indicate its **type** using the notation we introduced in class. You may assume that evaluation of each sequence takes place in a newly initialized Scheme system.

For example, for the expression 88 your answer would be

Value:	88
Type:	number

Question 15.

```
((lambda (a b) (a b))
 (lambda (c) (* 2 c))
 10)
```

Question 16.

```
(lambda (m n)
 (if m (* 2 n) (/ 2 n)))
```

Question 17.

```
((lambda (x y z) (z y x)) < 10 20)
```

Question 18.

```
(let ((x 10))
 (lambda (y) x))
```

Question 19.

```
(define three
 (lambda (a b c) (* a b c)))
(three (three 1) (three 2) (three 3))
```

Question 20.

```
(define (double x) (* 2 x))
(define (check y)
 (cond ((< (double y) 10) "yip")
 (> (double y) 6) "yay")
 (else "yuck")))
check
```

Part 4: (30 points)

Ben Bitdiddle, having struck it rich with his census software, was travelling the world. Due to lack of foresight, his money ran out as he was passing through India. In order to make enough money to get home, he hires himself out as a programmer. Being the new guy, they assign him the task of reversing lists. Luckily, Ben thought to pack his usual library of list procedures (see back of exam booklet) including `map`, `filter`, `fold-right`, `append`, and `length` for use in this part.

```
(define (reverse lst)
  Q21-ANSWER)
```

Question 21. What code should be inserted in place of Q21-ANSWER so that the result gives rise to a recursive process?

In a fit of creativity, Ben decides he could reverse a list by moving the first element to the front, then the second element to the front, then the third element to the front, and so on.

```
In order to reverse    (1 2 3 4 5)
move 1 to the front... (1 2 3 4 5)
move 2 to the front... (2 1 3 4 5)
move 3 to the front... (3 2 1 4 5)
move 4 to the front... (4 3 2 1 5)
move 5 to the front... (5 4 3 2 1)
                      reversed!
```

In order to implement this plan, Ben first writes two helper procedures:

1. `swap-first-and-second` swaps the first two elements of a list that contains at least two elements.

```
(define (swap-first-and-second lst)
  Q22-ANSWER)
```

```
(swap-first-and-second (list 1 2 3))
;Value: (2 1 3)
```

Question 22. What code should be inserted in place of Q22-ANSWER?

2. `nth-to-front` moves the `n`th element of the list to the front. `nth-to-front` is zero-based, meaning that `(nth-to-front lst 1)` returns a new list with the *second* element moved to the front. It is a good idea to use `swap-first-and-second` in your implementation of `nth-to-front`. You may assume that `n` is less than the length of `lst`.

```
(define (nth-to-front lst n)
  Q23-ANSWER)

(nth-to-front (list 1 2 3 4) 0)
;Value: (1 2 3 4)
(nth-to-front (list 1 2 3 4) 1)
;Value: (2 1 3 4)
(nth-to-front (list 1 2 3 4) 2)
;Value: (3 1 2 4)
```

Question 23. What code should be inserted in place of Q23-ANSWER?

Finally, Ben uses `nth-to-front` to implement `reverse`.

```
(define (reverse lst)
  (define (helper lst n)
    Q24-ANSWER)
  (helper lst 0))
```

Question 24. What code should be inserted in place of Q24-ANSWER?

Part 5: (20 Points)

In this part, you will analyze the order of growth in time and space of several procedures (on the next page).

As our usual measure for *space* required, in the following procedures we are interested in the maximum number of deferred operations waiting to be completed at any point during the computation.

As our usual measure for *time* required, in the following procedures we are interested in the total number of applications of some set of primitive operations. In this case (for the procedures to be analyzed in this part), for *time* we wish to count the number of explicit applications of the `cons` procedure.

Denote n as the size (length) of `lst`, or the combined length of `lst1` and `lst2` for `join`. Specify one of the letters to indicate the order of growth.

- **A** - $\Theta(1)$
- **B** - $\Theta(\log n)$
- **C** - $\Theta(n)$
- **D** - $\Theta(n \log n)$
- **E** - $\Theta(n^2)$
- **F** - $\Theta(n^3)$
- **G** - $\Theta(2^n)$
- **H** - other

```
(define (split lst left right)
  ; a procedure that splits lst into two sublists of approximately equal size.
  (if (null? lst)
      (cons left right)
      (split (cdr lst) (cons (car lst) right) left)))
```

```
; Example
(split (list 1 2 3 4) nil nil)          ;Value: ((4 2) 3 1)
```

```
(define (join lst1 lst2)
  ; takes in two sorted lists and joins them into a single sorted list
  (cond ((null? lst1) lst2)
        ((null? lst2) lst1)
        ((< (car lst1) (car lst2))
         (cons (car lst1) (join (cdr lst1) lst2)))
        (else
         (cons (car lst2) (join lst1 (cdr lst2))))))
```

```
; Example
(join (list 1 5 10 20) (list 2 7))      ;Value: (1 2 5 7 10 20)
```

```
(define (sort lst)
  ; sorts by recursively breaking the list in half,
  ; sorting each half, then joining the two results
  (if (or (null? lst) (null? (cdr lst)))
      lst
      (let ((split-list (split lst nil nil)))
        (join (sort (car split-list))
              (sort (cdr split-list))))))
```

```
; Example
(sort (list 7 12 2 8))                  ;Value: (2 7 8 12)
```

```
(define (swap-sort lst)
  ; sorts by swapping adjacent elements until none are out of order
  (define (swap-helper lst n)
    (if (= n 0)
        lst
        (swap-helper
         (fold-right (lambda (x rest)
                      (if (and (not (null? rest)) (> x (car rest)))
                          (swap-first-and-second (cons x rest))
                          (cons x rest)))
                    nil
                    lst)
         (- n 1))))
  (swap-helper lst (length lst)))
```

```
; Example
(swap-sort (list 7 12 2 8))              ;Value: (2 7 8 12)
```

Question 25. What is the order of growth in *time* of **split**?

Question 26. What is the order of growth in *space* of **split**?

Question 27. What is the order of growth in *time* of **join**?

Question 28. What is the order of growth in *space* of **join**?

Question 29. What is the order of growth in *time* of **sort**?

Question 30. What is the order of growth in *space* of **sort**?

Question 31. What is the order of growth in *time* of **swap-sort**?

Question 32. What is the order of growth in *space* of **swap-sort**?

BACKGROUND INFORMATION—THIS PAGE CONTAINS NO QUESTIONS**List Procedures**

```
(define (map op lst)
  (if (null? lst)
      nil
      (cons (op (car lst))
            (map op (cdr lst)))))

(define (filter pred lst)
  (cond ((null? lst) nil)
        ((pred (car lst))
         (cons (car lst) (filter pred (cdr lst))))
        (else (filter pred (cdr lst)))))

(define (fold-right op init lst)
  (if (null? lst)
      init
      (op (car lst)
          (fold-right op init (cdr lst)))))

(define (append lst1 lst2)
  (if (null? lst1)
      lst2
      (cons (car lst1) (append (cdr lst1) lst2))))

(define (length lst)
  (if (null? lst)
      0
      (+ 1 (length (cdr lst)))))
```

BACKGROUND INFORMATION—THIS PAGE CONTAINS NO QUESTIONS