6.001 SICP Spring 2004 3/15/2004

Tutorial 6

Topics

Tagged Data and Tables

Upcoming Deadlines

- Problem set 5 due Tuesday, 3/16 at midnight
- Lecture 12 due Wednesday, 3/17 at 9am
- Lecture 13 LIVE on Thursday, 3/18 at 10am
- Project 3 due Friday, 3/19 at 6pm
- Reminder: my office hours are Wednesdays from 3-5pm
- Next week is spring break (no office hours)

Tables - Binary Tree Implementation

Some constructors, accessors, and predicates:

```
(define (make-empty-node) nil)
(define empty-node? null?)
(define node-tag 'node)
(define (make-node key val)
  (list node-tag key val (make-empty-node) (make-empty-node)))
(define (node? n)
  (and (pair? n) (eq? (car n) node-tag)))
(define get-key cadr)
(define get-val caddr)
(define get-left cadddr)
(define get-right caddddr)
(define (put! node val)
  (set-car! (cdr node) val) node)
(define (put-left! node left-node)
  (set-car! (cdddr node) left-node) node)
(define (put-right! node right-node)
  (set-car! (cddddr node) right-node) node)
```

```
(define table-tag 'table)
; comp-proc is a procedure that compares two values x and y.  If
; (= x y) is true, comp-proc returns 0.  If (< x y) is true, then
; comp-proc returns -1.  Else, it returns 1.
(define (make-table comp-proc)
  (list table-tag comp-proc (make-empty-node)))
```

Exercise 1

Write the procedure `table-get` that gets the value associated with a `key`.

```
(define (table-get tbl key)
  EXERCISE-1)
```

Exercise 2

Fill in the procedure `table-put!` that inserts a `key, value` pair into the table.

```
(define (table-put! tbl key val)
  (define (insert-help node key val comp-proc)
    EXERCISE-2)
  (set-car! (cddr tbl) (insert-help (caddr tbl) key val (cadr tbl))))
```

Exercise 3

What is the time complexity for insertion? How about retrieval?

Tables - Memoization

Exercises 4-6

We wish to store the result of computation on problems of smaller size for use in future computations (assume that we use a hash table and that we've defined `size` and `hashfunc` elsewhere). Use this procedure to memoize `fib`. What is the time and space complexity of `memo-fib`?

```
(define (memoize f)
  (let ((table (make-table size hashfunc)))
    (lambda (x)
      (let ((previously-computed-result EXERCISE-4))
        (or previously-computed-result
            (let ((result EXERCISE-5))
              EXERCISE-6
              result))))))

(define memo-fib
  EXERCISE-7)
```