Goal:  Learn some scheme and play with passing functions as parameters.

Problem Statement:  Write a program to search for things!

Analysis:  The search function takes a data structure, an element to search for, and four functions: current-item, done, found, and next. Function current-item returns the current item at the front, head, top, whatever of the data structure. When passed an element and the data structure, done returns true if the search should stop with failure. When passed an element and the current item of the data structure, found returns true if the current item of the data structure is the element being searched for. Finally, passed an element and the data structure, next returns the part of the data structure to be searched next. The search function should return the element from the data structure if found (not the element searched for), or either nil of false (#f) if not found.

Example.  To search an unordered list, we might use the following:
(define unsorted-list '(6 7 4 45 7 76 3 67 7 63 19))
(define (second_empty first second) (equal? second '()))
(define (cdr_second a b) (cdr b))
(search unsorted-list '67 car second_empty equal? cdr_second)

After writing your search function, you should write the auxiliary functions and the calls to search a sorted list and a binary tree:

(define sorted-list '(2 6 10 14 55 65 78 99 102))
(define binary-search-tree
 '(10 (8 (4 () (6 () ())) (9 () ())) (15 () (18 () ()))))

Design:  The only design requirement is the use of higher-order functions in Scheme.

Test Plan:  The above examples provide proof of concept. They are not a test plan. Please provide one.

What to Hand In
(1)  Email your scheme code attached as <your name>--search.scm.
(2)  An edited and annotated script[man script(1)] of your program searching the list, the ordered list, and the binary tree attached as <your name>--output. Be sure to include searches that are successful and searched that are unsuccessful.

Extra Credit (3 super-sized points)
In addition to the value found, return a list that contains a $ for every time next is called. For example,
(search unsorted-list '67 car second_empty equal? cdr_second)
produces
($ $ $ $ $ $ $ 67).