

## CS 3721: Programming Languages Lab

Lab #02 Solutions: Recursion over Lists in Scheme

**Due date: February 5, 3:30pm.** At the beginning of the next recitation.

Goals of this lab:

- How to conditionally evaluate expressions using `cond` ; What `lambda` is ; How to create and use an anonymous function with `lambda` ; How to use `define` and `lambda` to create a named function ; How to build a simple function that recurses over a list

1. Conditionals in Scheme. We will talk about two control-flow operators, *if*, and *cond*.

**cond syntax:**

`(cond < clause1 > < clause2 > ...)`

Each `< clause >` should have the form:

`(< test > < expression >)`

and the last `< clause >` may be an “else clause” which has the form:

`(else < expression >)`

*Semantics:* A **cond** expression is evaluated by evaluating the `< test >` expressions of successive `< clause >`s in order until one of them evaluates to a true value. When a `< test >` evaluates to a true value, then the corresponding `< expression >` in its `< clause >` is evaluated, and the result of that `< expression >` is returned as the result of the entire **cond** expression.

What does the following *cond* expression evaluate to? Fill in the blank.

```
(cond
  ((< 3 1) 3)
  ((> 4 2) (> 2 4))           => (evaluates to) ___false___
  ((= 5 5) '(= 5 5))         (Since the first condition that holds is
  (else 'else)                (> 4 2), the corresponding expression,
)                               (> 2 4), is evaluated.)
```

Write your own example expression using the *cond* operator, and then write down what that expression evaluates to.

```
(cond
  ((> 5 5) 'greater)
  ((< 5 5) 'less)
  (else 'equal)
)
```

=> 'equal ; Since the tests in the first 2 clauses don't hold,  
; the expression in the else clause is evaluated.

**if syntax:**

`(if < test > < consequent > < alternate >)` , or

`(if < test > < consequent >)`

*Semantics:* An **if** expression is evaluated as follows: first, `< test >` is evaluated. If it yields a true value, then `< consequent >` is evaluated and its value is returned. Otherwise `< alternate >` is evaluated and its value is returned. Write an example expression using the *if* operator, and then write down what that expression evaluates to.

```
(if (> 3 2) (- 3 2) (+ 3 2))
```



4. Recursion is the primary programming technique in functional languages such as Lisp or Scheme. A recursive program is composed of three parts: bind a name to a function using `define`, the base case implementation of the recursive algorithm, and the recursive case implementation of the algorithm (by calling itself).

In general, keep in mind the followings while processing lists recursively:

★ `(null? myList)` tests if there are no more list elements. Base Case

★ `(car myList)` accesses the current element of the list (first element of the remaining list), and is usually used to process each element.

★ `(cdr myList)` is usually passed as a parameter to the recursive call.

★ the recursive function should take the list (or the part of the list that still needs to be processed) as a parameter.

Define a function `member?`, which scans through a list (passed as its second argument) for an element matching its first argument. `member?` evaluates to `#t` if there is a match, otherwise it evaluates to `#f`. (*Hint: use `eqv?`*)

**Solution:**

```
(define member? (lambda (x myList)
  (cond
    ((null? myList) #f)
    (else (or (eqv? x (car myList))
              (member? x (cdr myList))))))
```

```
>(member? 4 '(2 3 5 6 7))
```

```
false
```

```
>(member? 'a '(3 4 2 a 2))
```

```
true
```