"Scheming Sets"

Programming Lab 4

Programming Language Fundamentals (CS 381)

Scheme

For this lab, you will use Scheme. Turnin will use Racket (http://racket-lang.org/).

> Protip: You will find more examples if you search general Scheme questions rather than Racket specific questions.

Warmup

Begin by entering this function in `lab4.rkt`. Semicolons denote comments.

```
(define (f lst)
  ; (a) ;
  (if (null? lst)
      ; (b) ;
      '()
      ; (c) ;
      (cons (+ 1 (car lst))  (f (cdr lst)))))
```

Self-Check Questions

Question 1: Run this function as follows. What output do you get?

```
(f '(3 1 4 1 5 9))
```

Question 2: What does this function `f` do?

Question 3: Give a comment that explains the line following `(a)`, `(b)`, and `(c)`.

Question 4: Trace the call given in Question 1, showing each recursive call to the function. Expand the term `lst` to specific lists or atoms at each depth of the recursion.

Member? function

Write a function `member?` that determines if an element `e` is part of list `lst`. This function will return `#t` if `e` is a member of the list `lst` and `#f` otherwise. You may use common Scheme functions (see list below). You may **not** use the built-in member function in your answer (see below). Comment your function.

```
(define (member?  e lst)

  ; complete this function definition

)
```

PROTIP: Your other function will rely on your `member?` function, so test it well.

Set? function

Write a function `set?` that checks whether its argument `lst` is a well formed set, that is a set with no duplicates, and returns true `#t` or false `#f` accordingly. You may find it useful to make use of your `member?` function in your function.

For example:

```
(set? '(x y z)) => #t
(set? '(a 1 b 2 c 3)) => #t
(set? ()) => #t  ; empty set is a good set
(set? '(6 2 2)) => #f   ; duplicate, bad set
(set? '(x y z x)) => #f   ; duplicate, bad set

(define (set?  lst)

  ; complete this function definition

)
```

PROTIP: You **match** the order of the output given in the test .out files. This is often merely requires a re-ordering of your expressions.

Union function

Write a function `union` that takes the set union of list `lst1` and list `lst2` and returns a list representing the mathematical union of the two lists. Your function does not need to work on embedded lists. You may use the functions you defined previously (`set?` and `member?`), if useful, in addition to the common Scheme functions. Comment your function.

```
(define (union lst1 lst2)

   ; complete this function definition

)
```

Intersect function

Write a function `intersect` that takes the set intersection of list `lst1` and list `lst2` and returns a list representing the mathematical intersection of the two lists. Your function does not need to work on embedded lists. Comment your function.

```
(define (intersect lst1 lst2)

   ; complete this function definition

)
```

Difference function

Write a function `difference` that takes the set difference of list `lst1` minus list `lst2` and returns a list representing the mathematical difference of the two lists. You do not need to implement a symmetric difference, only $A - B$. Your function does not need to work on embedded lists. Comment your function.

```
(define (difference lst1 lst2)

   ; complete this function definition

)
```

PROTIP: For `union`, `intersect`, and `difference`, you can assume `lst1` and `lst2` are each themselves proper sets (i.e. no duplicates).

Troubleshooting

This lab requires an independent study of the Scheme (Racket) language. You are encouraged to use any web tutorials and resources to learn Scheme. Allow yourself plenty of time, and use patience, perseverance, and the Internet to debug your code. I will gladly answer clarifying questions about the goals and instructions of the Lab assignment, but I will not be able to debug your code for you.

Testing

The test files are located on Canvas. Keep the test files in a subdirectory `./tests/` in order to use the run all test instructions below.
Copy the following lines of code and paste them at the end of your program. This code reads lines from standard input and evaluates them. It terminates on end-of-input.

```
(define-namespace-anchor anc)
(define ns (namespace-anchor->namespace anc))
(let loop ()
    (define line (read-line (current-input-port) 'any))
    (if (eof-object? line)
       (display "")
       (begin  (print (eval (read (open-input-string line)) ns)) (newline) (loop))))
```

You can now use the tests via standard input.

PROTIP: You do not need to run every test every time. Manually run the test for the task you are working on.

Run Individual Tests

```
$ racket Lab4.rkt < ./tests/t01.in > ./myout/t01.myout
$ diff ./tests/t01.out ./myout/t01.myout
```

PROTIP: The order of the tests follow your progress through this lab. For example `t11` tests `set?`, `t21` tests `union`, and so on.

Run all Tests

You can also run all the tests at once. For this there is a Python3 script named `lab4-tester.py`. This is a Python script that executes `racket lab4.rkt` for each test and performs a `diff` between the expected output and your output. It also saves a `vimdiff` image in the `./diff` directory.

```
$ python3 lab4-tester.py
```

This script assumes:

- your racket file is named `lab4.rkt` and in the same directory
- the subdirectory `./tests` is in the same directory
- it has permissions to make a subdirectory `./myouts`
- that your OS has the `diff` tool[1]

Submission

Each student will complete and submit this assignment individually. Do not consult with others. However, you are encouraged to use the Internet to learn Scheme **but not to find solutions to the specific tasks of this assignment**.

Submit your program `Lab4.rkt` on Canvas.

This project is worth 100 points. Comment your program heavily. Intelligent comments and a clean, readable formatting of your code account for 20% of your grade.

PROTIP: If you do not finish in time, turn in whatever work you have. If you turn nothing, you get a zero. If you turn in something, you receive partial credit.

[1] Windows users: http://gnuwin32.sourceforge.net/packages/diffutils.htm

Function Restrictions

Allowed Functions

```
define
#t, #f
car, cdr, cadr, ... (etc)
cons, append
eq?, eqv?, equal?
pair?, list?
null?, null
if, cond, else
```

Prohibited Functions

```
member, memv, memq
set-equal?, set-eqv?, set-eq?, set?
set, seteqv, seteq
set-member?, set-add, set-remove, set-empty?
set-count, set-first, set-rest
set-union, set-intersect
set-subtract, set-symmetric-difference
set=?, subset?, proper-subset?
for/set
```

Some functions are prohibited above because they accomplish your tasks for you. You must write your functions from scratch using only those in the allowed functions list (as well as lots and lots of parentheses).

Solutions using a prohibited function will not receive any credit. You may use any of your functions as helper functions in any of your other functions. You may write additional helper functions (but you should not need to).

PROTIP: If you find yourself needing one of the following, you should rethink your solution. You do not need to use any of the following.

```
let, length, remove, sort, and, or, map, display, print, eval, begin
```

Extra Credit (10 pts)

Write a function `flatten` that takes two lists `lst1` and `lst2` and returns a new list that contains all elements in both lists, but eliminates any embedded lists. You may retain duplicates (or choose eliminate them). You may freely use of any of the functions you wrote for this lab.

```
(define (flatten lst1 lst2)

  ; complete this function definition

)
```