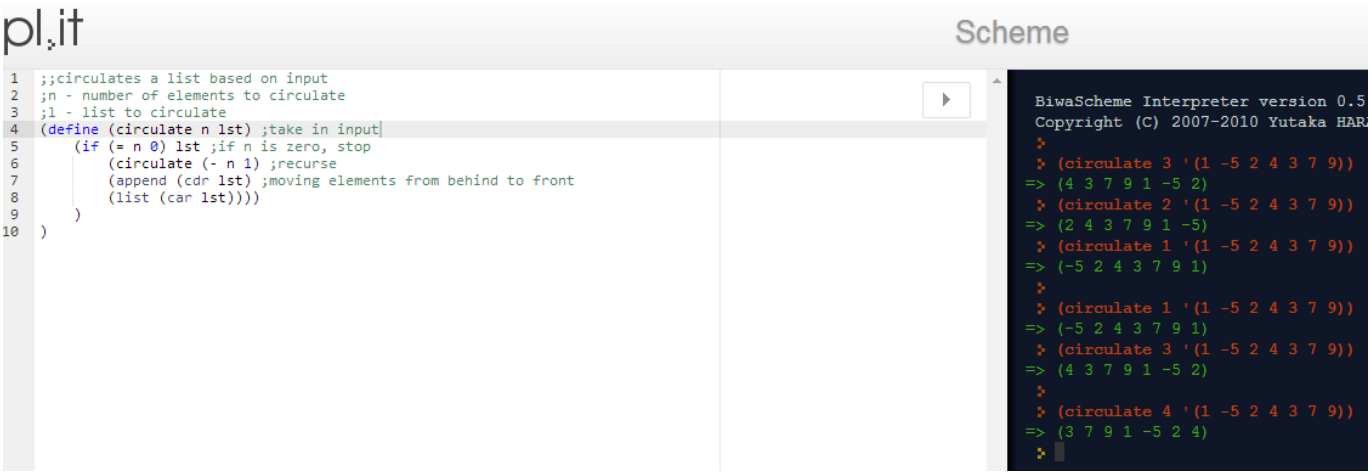
CS 396: Assignment 2
Scheme
Shariq Jamil
April 18th, 2014

Program your solution in Scheme

Problem 1 [2 points]
Write a function to circulate a list. The function takes two parameters, the first defines how many elements of the list to circulate, and the second is the list. The output should be the circulated list.

```
;;circulates a list based on input
;n - number of elements to circulate
;l - list to circulate
(define (circulate n lst) ;take in input
  (if (= n 0) lst ;if n is zero, stop
      (circulate (- n 1) ;recurse
      (append (cdr lst) ;moving elements from behind to front
      (list (car lst))))
  )
)
```



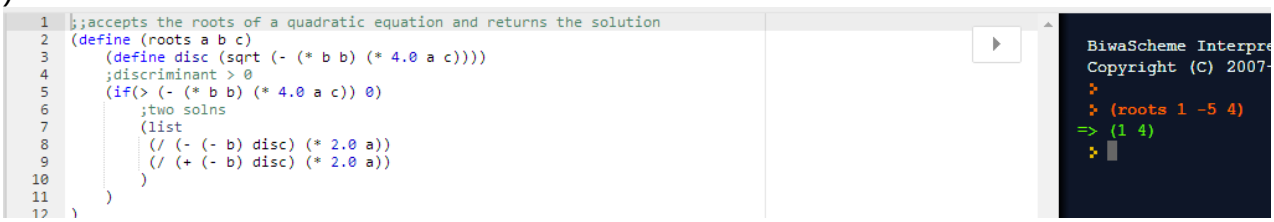
The code shifts one more character over than the prompt demands. This is because I am thinking the opposite way and moving elements from back to front as opposed to front to back.

Problem 2 [2 points]
Write a function to obtain the solution of equations of the form ax2 + bx + c = 0. The equation is represented in a list as (a b c). The output should be a list of the two solutions for the equations.

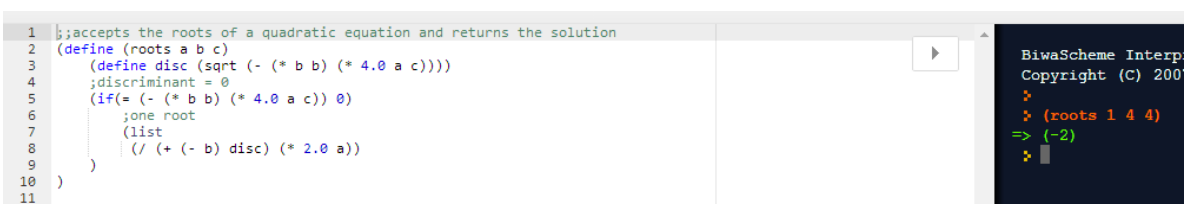
I could not consolidate all the if statements into one program and created three different programs to show results for the different scenarios.

;;accepts the roots of a quadratic equation and returns the solution

```
(define (roots a b c)
    (define disc (sqrt (- (* b b) (* 4.0 a c))))
    ;discriminant > 0
    (if(> (- (* b b) (* 4.0 a c)) 0)
        ;two solns
        (list
         (/ (- (- b) disc) (* 2.0 a))
         (/ (+ (- b) disc) (* 2.0 a))
        )
    )
)
```




```
;;accepts the roots of a quadratic equation and returns the solution
(define (roots a b c)
    (define disc (sqrt (- (* b b) (* 4.0 a c))))
    ;discriminant = 0
    (if(= (- (* b b) (* 4.0 a c)) 0)
        ;one root
        (list
         (/ (+ (- b) disc) (* 2.0 a))
    )
)
```




```
;;accepts the roots of a quadratic equation and returns the solution
(define (roots a b c)
    (define disc (sqrt (- (* b b) (* 4.0 a c))))
    ;discriminant < 0
    (if(< (- (* b b) (* 4.0 a c)) 0)
        ;no soln
        ()
    )
)
```

```
;;accepts the roots of a quadratic equation and returns the solution
(define (roots a b c)
    (define disc (sqrt (- (* b b) (* 4.0 a c))))
    ;discriminant < 0
    (if(< (- (* b b) (* 4.0 a c)) 0)
        ;no soln
        ()
    )
)
```

```
BiwaScheme Interpre
Copyright (C) 2007-
> (roots 1 -1 4)
=> ()
>
```

The program worked fairly well without the if-statements but they are necessary for the proper output formatting.

```
;;accepts the roots of a quadratic equation and returns the solution
(define (roots a b c)
    (define disc (sqrt (- (* b b) (* 4.0 a c))))
    (- (* b b) (* 4.0 a c))
        (list
         (/ (- (- b) disc) (* 2.0 a))
         (/ (+ (- b) disc) (* 2.0 a))
        )
)
```

```
BiwaScheme Interpret
Copyright (C) 2007-2
>
> (roots 1 -5 4)
=> (1 4)
>
> (roots 1 -5 4)
=> (1 4)
> (roots 1 4  4)
=> (-2 -2)
> (roots 1 -1  4)
=> (+nan.0 +nan.0)
>
```

Problem 3 [4 points]
Write a scheme program that allows two players to play a game of Tic Tac Toe.

This part of the project was very difficult for me and I spent a lot of time researching online. An implementation I found (http://n4sw.wordpress.com/2008/01/10/a-scheme-implementation-of-tic-tac-toe/) was very useful and I spent a lot of time deciphering, commenting and refactoring the code to understand it. This was a great exercise in understanding code written by another programmer in a disorganized fashion, formatting it properly and altering functions to match requirements.

```
;reference: http://n4sw.wordpress.com/2008/01/10/a-scheme-implementation-of-tic-tac-toe/
;refactored to fit CS 396 Tic-Tac-Toe assignment.

;user input
(define n 0) ;start from beginning

(define (slst? l)
    ;is a symbolic list
    (if (null? l) #t
        ;is not a symbolic list
        (if (number? (car l)) #f
            (slst? (cdr l))
        )
    )
)

;display grid
(define (view l)
```

```
    (begin (display "The grid: \n")
        (display ;first line
            (cons (car l) (cons (cadr l) (list (caddr l))))
        )
        (newline)
        (display ;second line
            (cons (cadddr l)
                (cons (cadddr (cdr l))(list(cadddr (cddr l)))
                )
            )
        )
        (newline)
        (display ;third line
            (cdddr(cdddr l))
        )
        (newline) ;for prompt
    )
)

;check for free spots
(define (free? l e) ;spots not taken
    (if (null? l) ;if empty
        (if (not (equal? (car l) e)) ;stays the same
            (free? (cdr l) e) ;fill in
        )
    )
)

;insert value in proper place
(define (tr l e s)
    (if (equal? (car l) e) ;find chosen spot
        ;replace with player symbol
        (cons s (cdr l))
        (cons (car l) (tr (cdr l) e s))
    )
)

;winning?
(define (win l wpos)
    (cond ((null? wpos) #f)
        ((eqc? (car wpos) l) #t)
    (else (win l (cdr wpos)))
    )
)

;list of winning scenarios
(define (win? l)
    (win l '((1 2 3) (4 5 6) (7 8 9) (1 4 7) (2 5 8) (3 6 9) (1 5 9) (7 5 3)))
)

;check to see if all elements are in the list
```

```
(define (eqc? l1 l2)
    (if (null? l1) #t
        ;they are in the list
        (if (cc l2 (car l1))
            ;not in list
            (eqc? (cdr l1) l2) #f
        )
    )
)

;does it contain the element?
(define (cc l e)
    (if (null? l) #f
    ;does not contain
        (if (equal? (car l) e) #t
            ;it does
            (cc (cdr l) e)
        )
    )
)

;the game
(define (wplay genlst plslst mnslst turn)
    ;winner prompt
    (let ((wis '(display "The winner is: ")))
        (view genlst)
        ;winning conditions
        ;x won
        (cond ((win? plslst) (eval wis) (display "X"))
            ;0 won
            ((win? mnslst) (eval wis) (display "0"))
            ;no one won
            ((slst? genlst) (display "no winner"))
            ;keep playing
            (else
                (begin
                    (display "Enter a number between 1 and 9, Turn: ")
                    (display turn)
                    (newline)
                    ;read input and display
                    (set! n (read)) (display "you insert: ") (display n)
                    (newline)
                    ;handle input
                    (if (number? n)
                        ;cant be larger than 9
                        (if (< n 10)
                            ;has to be empty
                            (if (free? genlst n)
                                ;insert move
                                ;X's move
                                [if (equal? turn 'X)
```

```
                        ;place X
                        (wplay (tr genlst n 'X)
                            (append plslst
                                (list n)
                            ) mnslst '0)
                        ;else place 0
                        (wplay (tr genlst n '0) plslst
                            (append mnslst
                                (list n)) 'X
                        )
                    ]
                    ;already taken
                    (begin
                        (display "this is a busy place\n")
                        (wplay genlst plslst mnslst turn)
                    )
                )
                ;cant place here
                (begin
                    (display "not valid place\n")
                    (wplay genlst plslst mnslst turn)
                )
            )
            ;out of range
            (begin
                (display "Enter a number between 1 and 9\n")
                (wplay genlst plslst mnslst turn)
            )
        )
    )
  )
 )
)

;start game
(define play
    ;first player
    (begin (display "X begins the game!\n")
    ;tic tac toe board
    (wplay (list 1 2 3 4 5 6 7 8 9)
    (list 'X) (list '0) 'X)
    )
)
```

```
                                ;already taken
                                (begin
                                    (display "this is a busy place\n")
                                    (wplay genlst plslst mnslst turn)
                                )
                            )
                            ;cant place here
                            (begin
                                (display "not valid place\n")
                                (wplay genlst plslst mnslst turn)
                            )
                        )
                        ;out of range
                        (begin
                            (display "Enter a number between 1 and 9\n")
                            (wplay genlst plslst mnslst turn)
                        )
                    )
                )
            )
        )
    )
)

;start game
(define play
    ;first player
    (begin (display "X begins the game!\n")
    ;tic tac toe board
    (wplay (list 1 2 3 4 5 6 7 8 9)
    (list 'X) (list '0) 'X)
    )
)
```

```
Enter a number between 1 and 9, Turn: 0
 7
you insert: 7
The grid:
(X 2 3)
(4 5 6)
(0 8 9)
Enter a number between 1 and 9, Turn: X
 2
you insert: 2
The grid:
(X X 3)
(4 5 6)
(0 8 9)
Enter a number between 1 and 9, Turn: 0
 8
you insert: 8
The grid:
(X X 3)
(4 5 6)
(0 0 9)
Enter a number between 1 and 9, Turn: X
 3
you insert: 3
The grid:
(X X X)
(4 5 6)
(0 0 9)
The winner is: X>
```