Name: Pratyay Kumar
Date: 11/16/2022
Lab: 9

Program #9: Lisp Programming

Problem Description:

Given a CD, you are to write the following routines:

1. Write a function that counts the number of times a logical operator is used in a CD; for example (count-operator 'OR '((OR 1 (AND 1 A1))) would return 1.
2. Write a function that uniquely lists all the input VARIABLES.
3. Write a function that, given a CD, reduces the CD to a simpler form by using tautologies. Example (reduce '(OR 0 (AND A1 A2))) would yield (AND A1 A2) -- this can be tricky. We can reduce with the following identities: (AND 1 S) == S, (OR 0 S) == S, (AND 0 S) == 0, (OR 1 S) == 1, (NOT O) == 1, (NOT 1) == 0.

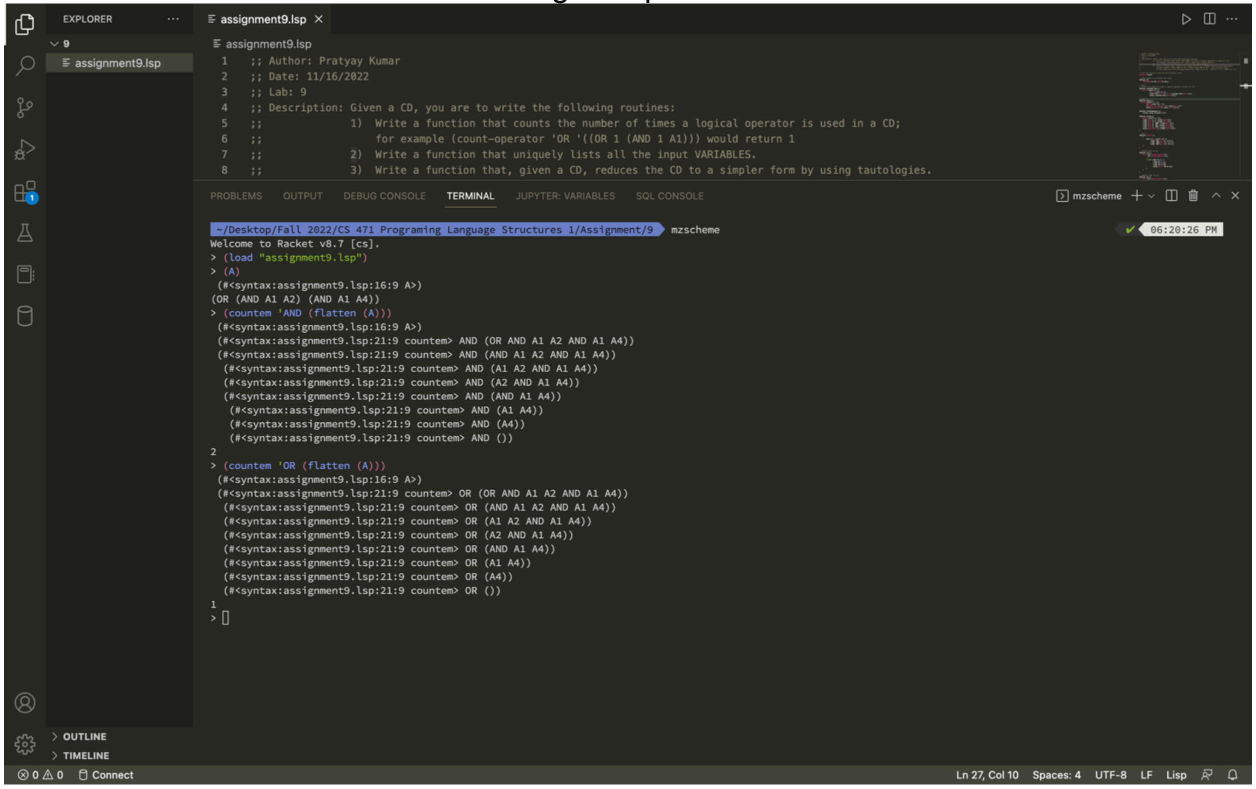
Code:

```
;; Author: Pratyay Kumar
;; Date: 11/16/2022
;; Lab: 9
;; Description: Given a CD, you are to write the following routines:
;;          1)  Write a function that counts the number of times a logical operator is used in a CD;
;;              for example (count-operator 'OR '((OR 1 (AND 1 A1))) would return 1
;;          2)  Write a function that uniquely lists all the input VARIABLES.
;;          3)  Write a function that, given a CD, reduces the CD to a simpler form by using tautologies.
;;              Example (reduce '(OR 0 (AND A1 A2))) would yield (AND A1 A2) -- this can be tricky.
;;              We can reduce with the following identities: (AND 1 S) == S,  (OR 0 S ) == S, (AND 0 S) == 0, ( OR 1 S)
== 1, (NOT O) == 1, (NOT 1) == 0.

;; Turn on tracing to see how the execution works.
(require trace)

;; Make things in a flatten list uniq
(define (A)
  '(OR (AND A1 A2) (AND A1 A4)))

;; Part 1
;; Counts the number of times a logical operator is used in a CD
(define (countem term L)
  (cond ((null? L) 0)
        ((not (list? L)) 0)
        ((eq? term (car L)) (+ 1 (countem term (cdr L))))
```

```
            (else (countem term (cdr L)))))

;; Part 2: Uniquely list all the input variables.
(define (uniq L)
  (cond ((null? L) '())
        ((not (list? L)) '())
        ((member (car L) (cdr L)) (uniq (cdr L)))
        (else (cons (car L) (uniq (cdr L))))))

(define (findinputvars L)
    (clean (uniq (flatten L))))

;; Removes all the stuff which is not required.
(define (clean L)
    (cond ((null? L) '())
    ((eq? (car L) 1) (clean (cdr L)))
    ((eq? (car L) 0) (clean (cdr L)))
    ((eq? (car L) 'AND) (clean (cdr L)))
    ((eq? (car L) 'OR) (clean (cdr L)))
    ((eq? (car L) 'NOT) (clean (cdr L)))
    (else (cons (car L) (clean (cdr L)))))
)

;; Part 3
(define (reduce L)
    (
        cond ((not (list? L)) L)
             ((eq? 'OR (car L)) (ro L))
             ((eq? 'AND (car L)) (ra L))
             ((eq? 'NOT (car L)) (rn L))
    )
)

;; Reduce an OR circuit
(define (ro L)
    (let ([S1 (reduce (cadr L))]
          [S2 (reduce (caddr L))]
         )
        ( cond ((eq? S1 1) 1)
               ((eq? S2 1) 1)
               ((eq? S1 0) S2)
               ((eq? S2 0) S1)
               (else (list 'OR S1 S2))
        )
    )
)

;; Reduce a AND circuit
```

```
(define (ra L)
   (let ([S1 (reduce (cadr L))]
        [S2 (reduce (caddr L))]
        )
        ( cond ((eq? S1 0) 0)
               ((eq? S2 0) 0)
               ((eq? S1 1) S2)
               ((eq? S2 1) S1)
               (else (list 'AND S1 S2))
       )
   )
)

;; Reduce a NOT circuit
(define (rn L)
   (let ([S1 (reduce (cadr L))]
        )
       ( cond ((eq? S1 1) 0)
              ((eq? S1 0) 1)
              (else (list 'NOT S1))
       )
   )
)
```

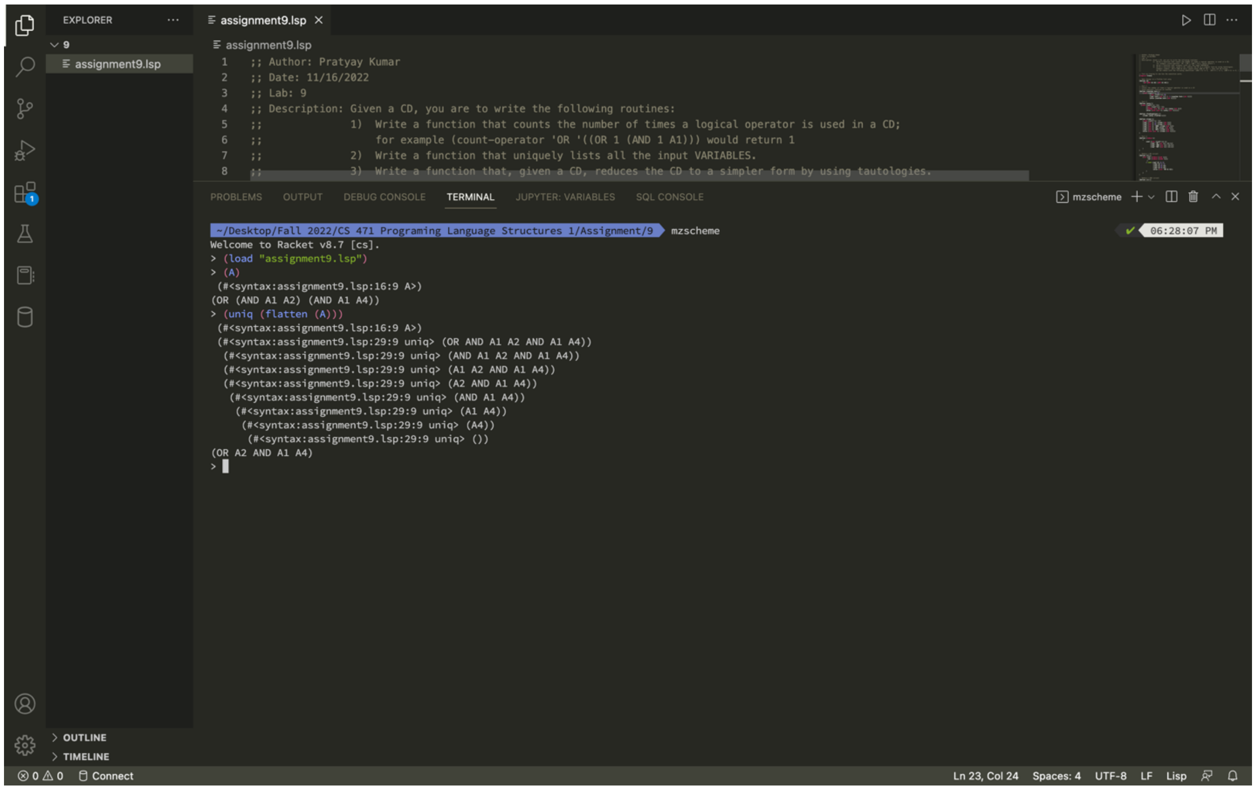
Outputs:
Task1: Count the number of times a logical operator is used in a CD



```
;; Author: Pratyay Kumar
;; Date: 11/16/2022
;; Lab: 9
;; Description: Given a CD, you are to write the following routines:
;;              1)  Write a function that counts the number of times a logical operator is used in a CD;
;;                  for example (count-operator 'OR '((OR 1 (AND 1 A1))) would return 1
;;              2)  Write a function that uniquely lists all the input VARIABLES.
;;              3)  Write a function that, given a CD, reduces the CD to a simpler form by using tautologies.
```

```
~/Desktop/Fall 2022/CS 471 Programing Language Structures 1/Assignment/9  mzscheme
Welcome to Racket v8.7 [cs].
> (load "assignment9.lsp")
> (A)
 (#<syntax:assignment9.lsp:16:9 A>)
(OR (AND A1 A2) (AND A1 A4))
> (countem 'AND (flatten (A)))
 (#<syntax:assignment9.lsp:16:9 A>)
 (#<syntax:assignment9.lsp:21:9 countem> AND (OR AND A1 A2 AND A1 A4))
 (#<syntax:assignment9.lsp:21:9 countem> AND (AND A1 A2 AND A1 A4))
  (#<syntax:assignment9.lsp:21:9 countem> AND (A1 A2 AND A1 A4))
  (#<syntax:assignment9.lsp:21:9 countem> AND (A2 AND A1 A4))
  (#<syntax:assignment9.lsp:21:9 countem> AND (AND A1 A4))
   (#<syntax:assignment9.lsp:21:9 countem> AND (A1 A4))
   (#<syntax:assignment9.lsp:21:9 countem> AND (A4))
   (#<syntax:assignment9.lsp:21:9 countem> AND ())
2
> (countem 'OR (flatten (A)))
 (#<syntax:assignment9.lsp:16:9 A>)
 (#<syntax:assignment9.lsp:21:9 countem> OR (OR AND A1 A2 AND A1 A4))
  (#<syntax:assignment9.lsp:21:9 countem> OR (AND A1 A2 AND A1 A4))
  (#<syntax:assignment9.lsp:21:9 countem> OR (A1 A2 AND A1 A4))
  (#<syntax:assignment9.lsp:21:9 countem> OR (A2 AND A1 A4))
  (#<syntax:assignment9.lsp:21:9 countem> OR (AND A1 A4))
  (#<syntax:assignment9.lsp:21:9 countem> OR (A1 A4))
  (#<syntax:assignment9.lsp:21:9 countem> OR (A4))
  (#<syntax:assignment9.lsp:21:9 countem> OR ())
1
> 
```

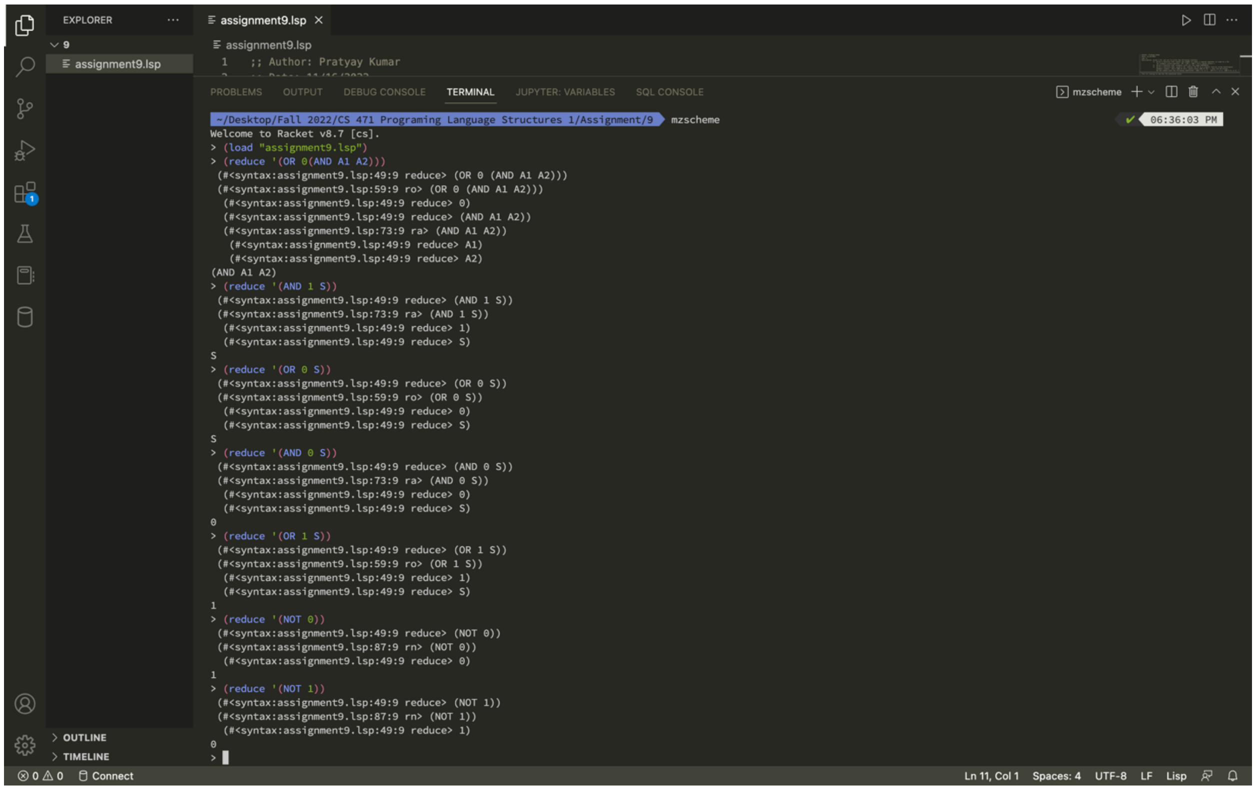
Task2: A function that uniquely lists all the input VARIABLES.



```
;; Author: Pratyay Kumar
;; Date: 11/16/2022
;; Lab: 9
;; Description: Given a CD, you are to write the following routines:
;;              1)  Write a function that counts the number of times a logical operator is used in a CD;
;;                  for example (count-operator 'OR '((OR 1 (AND 1 A1))) would return 1
;;              2)  Write a function that uniquely lists all the input VARIABLES.
;;              3)  Write a function that, given a CD, reduces the CD to a simpler form by using tautologies.
```

```
~/Desktop/Fall 2022/CS 471 Programing Language Structures 1/Assignment/9  mzscheme
Welcome to Racket v8.7 [cs].
> (load "assignment9.lsp")
> (A)
 (#<syntax:assignment9.lsp:16:9 A>)
(OR (AND A1 A2) (AND A1 A4))
> (uniq (flatten (A)))
 (#<syntax:assignment9.lsp:16:9 A>)
 (#<syntax:assignment9.lsp:29:9 uniq> (OR AND A1 A2 AND A1 A4))
  (#<syntax:assignment9.lsp:29:9 uniq> (AND A1 A2 AND A1 A4))
  (#<syntax:assignment9.lsp:29:9 uniq> (A1 A2 AND A1 A4))
  (#<syntax:assignment9.lsp:29:9 uniq> (A2 AND A1 A4))
   (#<syntax:assignment9.lsp:29:9 uniq> (AND A1 A4))
    (#<syntax:assignment9.lsp:29:9 uniq> (A1 A4))
     (#<syntax:assignment9.lsp:29:9 uniq> (A4))
      (#<syntax:assignment9.lsp:29:9 uniq> ())
(OR A2 AND A1 A4)
> 
```

Task3: A function that, given a CD, reduces the CD to a simpler form by using tautologies.



```
;; Author: Pratyay Kumar
```

```
~/Desktop/Fall 2022/CS 471 Programing Language Structures 1/Assignment/9  mzscheme
Welcome to Racket v8.7 [cs].
> (load "assignment9.lsp")
> (reduce '(OR 0(AND A1 A2)))
 (#<syntax:assignment9.lsp:49:9 reduce> (OR 0 (AND A1 A2)))
 (#<syntax:assignment9.lsp:59:9 ro> (OR 0 (AND A1 A2)))
  (#<syntax:assignment9.lsp:49:9 reduce> 0)
  (#<syntax:assignment9.lsp:49:9 reduce> (AND A1 A2))
  (#<syntax:assignment9.lsp:73:9 ra> (AND A1 A2))
   (#<syntax:assignment9.lsp:49:9 reduce> A1)
   (#<syntax:assignment9.lsp:49:9 reduce> A2)
(AND A1 A2)
> (reduce '(AND 1 S))
 (#<syntax:assignment9.lsp:49:9 reduce> (AND 1 S))
 (#<syntax:assignment9.lsp:73:9 ra> (AND 1 S))
  (#<syntax:assignment9.lsp:49:9 reduce> 1)
  (#<syntax:assignment9.lsp:49:9 reduce> S)
S
> (reduce '(OR 0 S))
 (#<syntax:assignment9.lsp:49:9 reduce> (OR 0 S))
 (#<syntax:assignment9.lsp:59:9 ro> (OR 0 S))
  (#<syntax:assignment9.lsp:49:9 reduce> 0)
  (#<syntax:assignment9.lsp:49:9 reduce> S)
S
> (reduce '(AND 0 S))
 (#<syntax:assignment9.lsp:49:9 reduce> (AND 0 S))
 (#<syntax:assignment9.lsp:73:9 ra> (AND 0 S))
  (#<syntax:assignment9.lsp:49:9 reduce> 0)
  (#<syntax:assignment9.lsp:49:9 reduce> S)
0
> (reduce '(OR 1 S))
 (#<syntax:assignment9.lsp:49:9 reduce> (OR 1 S))
 (#<syntax:assignment9.lsp:59:9 ro> (OR 1 S))
  (#<syntax:assignment9.lsp:49:9 reduce> 1)
  (#<syntax:assignment9.lsp:49:9 reduce> S)
1
> (reduce '(NOT 0))
 (#<syntax:assignment9.lsp:49:9 reduce> (NOT 0))
 (#<syntax:assignment9.lsp:87:9 rn> (NOT 0))
  (#<syntax:assignment9.lsp:49:9 reduce> 0)
1
> (reduce '(NOT 1))
 (#<syntax:assignment9.lsp:49:9 reduce> (NOT 1))
 (#<syntax:assignment9.lsp:87:9 rn> (NOT 1))
  (#<syntax:assignment9.lsp:49:9 reduce> 1)
0
>
```