```
; tests first
(require (planet schematics/schemeunit:3))
(require (planet schematics/schemeunit:3/text-ui))

(load "formula.scm")

(define formula->string-tests
  (test-suite
   "Tests for formula->string"
   (test-case "true and false"
              (check-equal? (formula->string #t) "1" "true")
              (check-equal? (formula->string #f) "0" "false"))
   (test-case "basic variables"
              (check-equal? (formula->string (make-variable 1)) "X_1"
                                                                "variable: X1")
              (check-equal? (formula->string (make-variable 99)) "X_99"
                                                                 "variable: X99"))
   (test-case "negation of variable"
              (check-equal? (formula->string (make-negation (make-variable 1)))
                            "¬X_1" "negation of variable X1"))
   (test-case "combination of two variables using conjunction"
              (check-equal? (formula->string (make-conjunction (make-variable 1)
                                    (make-variable 2))) "(X_1 ∧ X_2)" "X1 and X2"))
   (test-case "combination of two variables using disjunction"
              (check-equal? (formula->string (make-disjunction (make-variable 1)
(make-variable 2))) "(X_1 ∨ X_2)" "X1 or X2"))
   (test-case "combination of two variables using conditional"
              (check-equal? (formula->string (make-conditional (make-variable 1)
                             (make-variable 2))) "(X_1 → X_2)" "X1 implies X2"))
   (test-case "combination of two variables using biconditional"
              (check-equal? (formula->string (make-biconditional (make-variable 1)
(make-variable 2))) "(X_1 ↔ X_2)" "X1 implies X2 and  X2 implies X1"))
   (test-case "combination of a conjunction and conditional using conditional"
              (check-equal? (formula->string (make-conditional (make-conjunction
(make-variable 1) #f) (make-conditional (make-variable 3) (make-variable 4))))
"((X_1 ∧ 0) → (X_3 → X_4))" "(X1 and false) imply (X3 implies X4)"))
  )
)

(run-tests formula->string-tests)
```

```
; implementation next
(define (formula->string formula)
  (cond ((eq? formula #t) "1")
        ((eq? formula #f) "0")
        ((variable? formula)
         (string-append "X_" (number->string (variable-index formula))))
        ((negation? formula)
         (string-append "¬" (formula->string (negation-negated formula))))
        ((conjunction? formula)
         (string-append "("
                        (formula->string (conjunction-left formula))
                        " ∧ "
                        (formula->string (conjunction-right formula))
                        ")"))
        ((disjunction? formula)
         (string-append "("
                        (formula->string (disjunction-left formula))
                        " ∨ "
                        (formula->string (disjunction-right formula))
                        ")"))
        ((conditional? formula)
         (string-append "("
                        (formula->string (conditional-left formula))
                        " → "
                        (formula->string (conditional-right formula))
                        ")"))
        ((biconditional? formula)
         (string-append "("
                        (formula->string (biconditional-left formula))
                        " ↔ "
                        (formula->string (biconditional-right formula))
                        ")"))
  ))
```