```
;
; CS161 Fall 2011 HW6 Problem 3: Graph coloring to SAT conversion
;
; All functions you need to write are marked with 'EXERCISE' in their header comments.
; Same rules apply regarding Lisp functions you are allowed to use.
; In fact, you do not need a lot of Lisp functions to finish this assignment.
;

;;;;;;;;;;;;;;;;;;;;;
; General util.
;
(defun reload()
  (load "hw6.lsp")
  );end defun

; EXERCISE: Fill this function.
; returns the index of the variable
; that corresponds to the fact that
; "node n gets color c" (when there are k possible colors).
;
(defun node2var (n c k)
  )

; EXERCISE: Fill this function
; returns *a clause* for the constraint:
; "node n gets at least one color from the set {c,c+1,...,k}."
;
(defun at-least-one-color (n c k)
  )

; EXERCISE: Fill this function
; returns *a list of clauses* for the constraint:
; "node n gets at most one color from the set {c,c+1,...,k}."
;
(defun at-most-one-color (n c k)
  )

; EXERCISE: Fill this function
; returns *a list of clauses* to ensure that
; "node n gets exactly one color from the set {1,2,...,k}."
;
(defun generate-node-clauses (n k)
  )

; EXERCISE: Fill this function
; returns *a list of clauses* to ensure that
; "the nodes at both ends of edge e cannot have the same color from the set {1,2,...,k}."
;
(defun generate-edge-clauses (e k)
  )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Your exercises end here. Below are top-level
; and utility functions that you do not need to understand.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;
; Top-level function for converting the graph coloring problem
; of the graph defined in 'fname' using k colors into a SAT problem.
; The resulting SAT problem is written to 'out-name' in a simplified DIMACS format.
; (http://www.satcompetition.org/2004/format-solvers2004.html)
;
; This function also returns the cnf written to file.
;
; *works only for k>0*
;
(defun graph-coloring-to-sat (fname out-name k)
  (progn
    (setf in-path (make-pathname :name fname))
    (setf in (open in-path :direction :input))
    (setq info (get-number-pair-from-string (read-line in) #\ ))
    (setq cnf nil)
```

```
    (do ((node 1
               (+ node 1)
               ))
        ((> node (car info)))
      (setq cnf (append (generate-node-clauses node k) cnf))
      );end do
    (do ((line (read-line in nil 'eof)
               (read-line in nil 'eof)))
        ((eql line 'eof) (close in))
      (let ((edge (get-number-pair-from-string line #\ )))
        (setq cnf (append (generate-edge-clauses edge k) cnf))
        );end let
      );end do
    (close in)
    (write-cnf-to-file out-name (* (car info) k) cnf)
    (return-from graph-coloring-to-sat cnf)
    );end progn
  );end defun

;
; A utility function for parsing a pair of integers.
;
(defun get-number-pair-from-string (string token)
  (if (and string token)
      (do* ((delim-list (if (and token (listp token)) token (list token)))
            (char-list (coerce string 'list))
            (limit (list-length char-list))
            (char-count 0 (+ 1 char-count))
            (char (car char-list) (nth char-count char-list))
            )
           ((or (member char delim-list)
                (= char-count limit))
            (return
               (if (= char-count limit)
                   (list string nil)
                   (list (parse-integer (coerce (butlast char-list (- limit char-count))
                                'string))
                         (parse-integer (coerce (nthcdr (+ char-count 1) char-list) 'string))
                         )))))))

;
; Writes clause to file handle 'out'.
;
(defun write-clause-to-file (out clause)
  (cond ((null clause) (format out "0~%"))
        (t (progn
             (format out "~A " (car clause))
             (write-clause-to-file out (cdr clause))
             );end progn
           );end t
        );end cond
  );end defun

;
; Writes the formula cnf with vc variables to 'fname'.
;
(defun write-cnf-to-file (fname vc cnf)
  (progn
    (setf path (make-pathname :name fname))
    (setf out (open path :direction :output))
    (setq cc (length cnf))
    (format out "p cnf ~A ~A~%" vc cc)
    (dolist (clause cnf)
      (write-clause-to-file out clause)
      );end dolist
    (close out)
    );end progn
  );end defun
```