;This program is a maze solving program for a 8 by 8 grid maze.

;I am using the A\* algorithm to find the shortest path.

(setf maze (make-array '(8 8)

 :initial-contents

 '((x 0 0 0 w 0 0 w)

 (0 0 0 0 w 0 0 w)

 (0 0 w 0 0 0 0 0)

 (0 0 w 0 0 w 0 0)

 (w 0 0 0 0 w w 0)

 (w 0 w w 0 0 w 0)

 (w w 0 0 0 w 0 0)

 (w 0 0 w w w 0 d))))

(setf g (make-array '(8 8)))

(setf h (make-array '(8 8)))

(setf f (make-array '(8 8)))

(setq count\_x 0)

(setq count\_y 0)

;figure out where you start in the maze

(defun **search\_for\_start** (count\_x count\_y)

 (cond ((eq (setf value (aref maze count\_x count\_y)) 'x)

 (setq start\_x count\_x)

 (setq start\_y count\_y))

 ((and (>= count\_x 0) (<= count\_x 6))

 (setq count\_x (+ 1 count\_x))

 (search\_for\_start count\_x count\_y))

 ((= count\_x 7)

 (setq count\_y (+ 1 count\_y))

 (setq count\_x 0)

 (search\_for\_start count\_x count\_y))

 )

)

;figure out where the door is

(defun **search\_for\_door** (count\_x count\_y)

 (cond ((eq (setf value (aref maze count\_x count\_y)) 'd)

 (setq door\_x count\_x)

 (setq door\_y count\_y))

 ((and (>= count\_x 0) (<= count\_x 6))

 (setq count\_x (+ 1 count\_x))

 (search\_for\_door count\_x count\_y))

 ((= count\_x 7)

 (setq count\_y (+ 1 count\_y))

 (setq count\_x 0)

 (search\_for\_door count\_x count\_y))

 )

)

;calculate g value

(defun **calculate\_g** (current\_square\_x current\_square\_y)

 (cond ((eq (setf value (aref maze (abs(- 1 current\_square\_x))

 (abs(- 1 current\_square\_y)))) 'w)

 (setf (aref g (abs(- 1 current\_square\_x))

 (abs(- 1 current\_square\_y))) 'w))

 ((OR (< (abs(- 1 current\_square\_x)) 0)

 (< (abs(- 1 current\_square\_y)) 0) nil))

 (T (setf (aref g (abs(- 1 current\_square\_x))

 (abs(- 1 current\_square\_y))) 14))

 )

 (cond ((eq (setf value (aref maze (+ 1 current\_square\_x)

 (+ 1 current\_square\_y))) 'w)

 (setf (aref g (+ 1 current\_square\_x)

 (+ 1 current\_square\_y)) 'w))

 ((OR (> (+ 1 current\_square\_x) 8)

 (> (+ 1 current\_square\_y) 8) nil))

 (T (setf (aref g (+ 1 current\_square\_x)

 (+ 1 current\_square\_y)) 14))

 )

 (cond ((eq (setf value (aref maze (+ 1 current\_square\_x) current\_square\_y)) 'w)

 (setf (aref g (+ 1 current\_square\_x) current\_square\_y) 'w))

 ((> (+ 1 current\_square\_x) 8) nil)

 (t (setf (aref g (+ 1 current\_square\_x) current\_square\_y) 10))

 )

 (cond ((eq (setf value (aref maze (abs(- 1 current\_square\_x)) current\_square\_y)) 'w)

 (setf (aref g (abs(- 1 current\_square\_x)) current\_square\_y) 'w))

 ((< (abs(- 1 current\_square\_x)) 0) nil)

 (t (setf (aref g (abs(- 1 current\_square\_x)) current\_square\_y) 10))

 )

 (cond ((eq (setf value (aref maze current\_square\_x (+ 1 current\_square\_y))) 'w)

 (setf (aref g current\_square\_x (+ 1 current\_square\_y)) 'w))

 ((> (+ 1 current\_square\_y) 8) nil)

 (t (setf (aref g current\_square\_x (+ 1 current\_square\_y)) 10))

 )

 (cond ((eq (setf value (aref maze current\_square\_x (abs(- 1 current\_square\_y)))) 'w)

 (setf (aref g current\_square\_x (abs(- 1 current\_square\_y))) 'w))

 ((< (abs(- 1 current\_square\_y)) 0) nil)

 (t (setf (aref g current\_square\_x (abs(- 1 current\_square\_y))) 10))

 )

 (cond ((eq (setf value (aref maze (abs(- 1 current\_square\_x)) (+ 1 current\_square\_y))) 'w)

 (setf (aref g (abs(- 1 current\_square\_x)) (+ 1 current\_square\_y)) 'w))

 ((OR (< (abs(- 1 current\_square\_x)) 0) (> (+ 1 current\_square\_y) 8) nil))

 (T (setf (aref g (abs(- 1 current\_square\_x)) (+ 1 current\_square\_y)) 14))

 )

 (cond ((eq (setf value (aref maze (+ 1 current\_square\_x) (abs(- 1 current\_square\_y)))) 'w)

 (setf (aref g (+ 1 current\_square\_x) (abs(- 1 current\_square\_y))) 'w))

 ((OR (> (+ 1 current\_square\_x) 8) (< (abs(- 1 current\_square\_y)) 0) nil))

 (T (setf (aref g (+ 1 current\_square\_x) (abs(- 1 current\_square\_y))) 14))

 )

)

;calculate h values

(defun **calculate\_h** (current\_square\_x current\_square\_y)

 (cond ((eq (setf value (aref maze (abs(- 1 current\_square\_x)) (abs(- 1 current\_square\_y)))) 'w)

 (setf (aref h (abs(- 1 current\_square\_x)) (abs(- 1 current\_square\_y))) 'w))

 ((OR (< (abs(- 1 current\_square\_x)) 0) (< (abs(- 1 current\_square\_y)) 0) nil))

 (T (setf (aref h (abs(- 1 current\_square\_x)) (abs(- 1 current\_square\_y)))

 (\* (+ (abs(- door\_x (abs(- 1 current\_square\_x)))) (abs(- door\_y (- 1 current\_square\_y)))) 10)))

 )

 (cond ((eq (setf value (aref maze (+ 1 current\_square\_x) (+ 1 current\_square\_y))) 'w)

 (setf (aref h (+ 1 current\_square\_x) (+ 1 current\_square\_y)) 'w))

 ((OR (> (+ 1 current\_square\_x) 8) (> (+ 1 current\_square\_y) 8) nil))

 (T (setf (aref h (+ 1 current\_square\_x) (+ 1 current\_square\_y))

 (\* (+ (abs(- door\_x (+ 1 current\_square\_x))) (abs(- door\_y (+ 1 current\_square\_y)))) 10)))

 )

 (cond ((eq (setf value (aref maze (+ 1 current\_square\_x) current\_square\_y)) 'w)

 (setf (aref h (+ 1 current\_square\_x) current\_square\_y) 'w))

 ((> (+ 1 current\_square\_x) 8) nil)

 (t (setf (aref h (+ 1 current\_square\_x) current\_square\_y)

 (\* (+ (abs(- door\_x (+ 1 current\_square\_x))) (abs(- door\_y current\_square\_y))) 10)))

 )

 (cond ((eq (setf value (aref maze (abs(- 1 current\_square\_x)) current\_square\_y)) 'w)

 (setf (aref h (abs(- 1 current\_square\_x)) current\_square\_y) 'w))

 ((< (abs(- 1 current\_square\_x)) 0) nil)

 (t (setf (aref h (abs(- 1 current\_square\_x)) current\_square\_y)

 (\* (+ (abs(- door\_x (- 1 current\_square\_x))) (abs(- door\_y current\_square\_y))) 10)))

 )

 (cond ((eq (setf value (aref maze current\_square\_x (+ 1 current\_square\_y))) 'w)

 (setf (aref h current\_square\_x (+ 1 current\_square\_y)) 'w))

 ((> (+ 1 current\_square\_y) 8) nil)

 (t (setf (aref h current\_square\_x (+ 1 current\_square\_y))

 (\* (+ (abs(- door\_x current\_square\_x)) (abs(- door\_y (+ 1 current\_square\_y)))) 10)))

 )

 (cond ((eq (setf value (aref maze current\_square\_x (abs(- 1 current\_square\_y)))) 'w)

 (setf (aref h current\_square\_x (abs(- 1 current\_square\_y))) 'w))

 ((< (abs(- 1 current\_square\_y)) 0) nil)

 (t (setf (aref h current\_square\_x (abs(- 1 current\_square\_y)))

 (\* (+ (abs(- door\_x current\_square\_x)) (abs(- door\_y (abs(- 1 current\_square\_y))))) 10)))

 )

 (cond ((eq (setf value (aref maze (abs(- 1 current\_square\_x)) (+ 1 current\_square\_y))) 'w)

 (setf (aref h (abs(- 1 current\_square\_x)) (+ 1 current\_square\_y)) 'w))

 ((OR (< (abs(- 1 current\_square\_x)) 0) (> (+ 1 current\_square\_y) 8) nil))

 (T (setf (aref h (abs(- 1 current\_square\_x)) (+ 1 current\_square\_y))

 (\* (+ (abs(- door\_x (abs(- 1 current\_square\_x)))) (abs(- door\_y (+ 1 current\_square\_y)))) 10)))

 )

 (cond ((eq (setf value (aref maze (+ 1 current\_square\_x) (abs(- 1 current\_square\_y)))) 'w)

 (setf (aref h (+ 1 current\_square\_x) (abs(- 1 current\_square\_y))) 'w))

 ((OR (> (+ 1 current\_square\_x) 8) (< (abs(- 1 current\_square\_y)) 0) nil))

 (T (setf (aref h (+ 1 current\_square\_x) (abs(- 1 current\_square\_y)))

 (\* (+ (abs(- door\_x (+ 1 current\_square\_x))) (abs(- door\_y (abs(- 1 current\_square\_y))))) 10)))

 )

)

;calculate f values

(defun **calculate\_f** (current\_square\_x current\_square\_y)

 (cond ((eq (setf value (aref maze (abs(- 1 current\_square\_x)) (abs(- 1 current\_square\_y)))) 'w)

 (setf (aref f (abs(- 1 current\_square\_x)) (abs(- 1 current\_square\_y))) 'w))

 ((OR (< (abs(- 1 current\_square\_x)) 0) (< (abs(- 1 current\_square\_y)) 0) nil))

 (T (setf (aref f (abs(- 1 current\_square\_x)) (abs(- 1 current\_square\_y)))

 (+ (setf value (aref g (abs(- 1 current\_square\_x)) (abs(- 1 current\_square\_y)))) (setf value (aref h (abs(- 1 current\_square\_x)) (abs(- 1 current\_square\_y)))))))

 )

 (cond ((eq (setf value (aref maze (+ 1 current\_square\_x) (+ 1 current\_square\_y))) 'w)

 (setf (aref f (+ 1 current\_square\_x) (+ 1 current\_square\_y)) 'w))

 ((OR (> (+ 1 current\_square\_x) 8) (> (+ 1 current\_square\_y) 8) nil))

 (T (setf (aref f (+ 1 current\_square\_x) (+ 1 current\_square\_y))

 (+ (setf value (aref g (+ 1 current\_square\_x) (+ 1 current\_square\_y))) (setf value (aref h (+ 1 current\_square\_x) (+ 1 current\_square\_y))))))

 )

 (cond ((eq (setf value (aref maze (+ 1 current\_square\_x) current\_square\_y)) 'w)

 (setf (aref f (+ 1 current\_square\_x) current\_square\_y) 'w))

 ((> (+ 1 current\_square\_x) 8) nil)

 (t (setf (aref f (+ 1 current\_square\_x) current\_square\_y)

 (+ (setf value (aref g (+ 1 current\_square\_x) current\_square\_y))

 (setf value (aref h (+ 1 current\_square\_x) current\_square\_y)))))

 )

 (cond ((eq (setf value (aref maze (abs(- 1 current\_square\_x)) current\_square\_y)) 'w)

 (setf (aref f (abs(- 1 current\_square\_x)) current\_square\_y) 'w))

 ((< (abs(- 1 current\_square\_x)) 0) nil)

 (t (setf (aref f (abs(- 1 current\_square\_x)) current\_square\_y)

 (+ (setf value (aref g (abs(- 1 current\_square\_x)) current\_square\_y))

 (setf value (aref h (abs(- 1 current\_square\_x)) current\_square\_y)))))

 )

 (cond ((eq (setf value (aref maze current\_square\_x (+ 1 current\_square\_y))) 'w)

 (setf (aref f current\_square\_x (+ 1 current\_square\_y)) 'w))

 ((> (+ 1 current\_square\_y) 8) nil)

 (t (setf (aref f current\_square\_x (+ 1 current\_square\_y))

 (+ (setf value (aref g current\_square\_x (+ 1 current\_square\_y)))

 (setf value (aref h current\_square\_x (+ 1 current\_square\_y))))))

 )

 (cond ((eq (setf value (aref maze current\_square\_x (abs(- 1 current\_square\_y)))) 'w)

 (setf (aref f current\_square\_x (abs(- 1 current\_square\_y))) 'w))

 ((< (abs(- 1 current\_square\_y)) 0) nil)

 (t (setf (aref f current\_square\_x (abs(- 1 current\_square\_y)))

 (+ (setf value (aref g current\_square\_x (abs(- 1 current\_square\_y))))

 (setf value (aref h current\_square\_x (abs(- 1 current\_square\_y)))))))

 )

 (cond ((eq (setf value (aref maze (abs(- 1 current\_square\_x)) (+ 1 current\_square\_y))) 'w)

 (setf (aref f (abs(- 1 current\_square\_x)) (+ 1 current\_square\_y)) 'w))

 ((OR (< (abs(- 1 current\_square\_x)) 0) (> (+ 1 current\_square\_y) 8) nil))

 (T (setf (aref f (abs(- 1 current\_square\_x)) (+ 1 current\_square\_y))

 (+ (setf value (aref g (abs(- 1 current\_square\_x)) (+ 1 current\_square\_y)))

 (setf value (aref h (abs(- 1 current\_square\_x)) (+ 1 current\_square\_y))))))

 )

 (cond ((eq (setf value (aref maze (+ 1 current\_square\_x) (abs(- 1 current\_square\_y)))) 'w)

 (setf (aref f (+ 1 current\_square\_x) (abs(- 1 current\_square\_y))) 'w))

 ((OR (> (+ 1 current\_square\_x) 8) (< (abs(- 1 current\_square\_y)) 0) nil))

 (T (setf (aref f (+ 1 current\_square\_x) (abs(- 1 current\_square\_y)))

 (+ (setf value (aref g (+ 1 current\_square\_x) (abs(- 1 current\_square\_y))))

 (setf value (aref h (+ 1 current\_square\_x) (abs(- 1 current\_square\_y)))))))

 )

)

;move to new square

(defun **new\_move** (current\_square\_x current\_square\_y)

 (setq new\_square\_f 10000)

 (cond ((and (numberp (setf value (aref f (abs(- 1 current\_square\_x)) (abs(- 1 current\_square\_y)))))

 (numberp (setf value (aref f (abs(+ 1 current\_square\_x)) (abs current\_square\_y))))

 (< (setf value (aref f (abs(- 1 current\_square\_x)) (abs(- 1 current\_square\_y))))

 (setf value (aref f (+ 1 current\_square\_x) current\_square\_y))))

 (set 'new\_square\_x (abs(- 1 current\_square\_x)))

 (set 'new\_square\_y (abs(- 1 current\_square\_y)))

 (set 'new\_square\_f (setf value (aref f (abs(- 1 current\_square\_x)) (abs(- 1 current\_square\_y))))))

 (t nil)

 )

 (cond ((and (numberp (setf value (aref f (abs(- 1 current\_square\_x)) (abs(- 1 current\_square\_y)))))

 (numberp (setf value (aref f (abs(+ 1 current\_square\_x)) (abs current\_square\_y))))

 (> (setf value (aref f (abs(- 1 current\_square\_x)) (abs(- 1 current\_square\_y))))

 (setf value (aref f (+ 1 current\_square\_x) current\_square\_y))))

 (set 'new\_square\_x (+ 1 current\_square\_x))

 (set 'new\_square\_y current\_square\_y)

 (set 'new\_square\_f (setf value (aref f (+ 1 current\_square\_x) current\_square\_y))))

 (t nil)

 )

 (cond ((and (numberp (setf value (aref f (+ 1 current\_square\_x) (+ 1 current\_square\_y))))

 (< (setf value (aref f (+ 1 current\_square\_x) (+ 1 current\_square\_y))) new\_square\_f))

 (set 'new\_square\_x (+ 1 current\_square\_x))

 (set 'new\_square\_y (+ 1 current\_square\_y))

 (set 'new\_square\_f (setf value (aref f (+ 1 current\_square\_x) (+ 1 current\_square\_y))))

 )

 (t nil)

 )

 (cond ((and (numberp (setf value (aref f (+ 1 current\_square\_x) (abs(- 1 current\_square\_y)))))

 (< (setf value (aref f (+ 1 current\_square\_x) (abs(- 1 current\_square\_y)))) new\_square\_f))

 (set 'new\_square\_x (+ 1 current\_square\_x))

 (set 'new\_square\_y (abs(- 1 current\_square\_y)))

 (set 'new\_square\_f (setf value (aref f (+ 1 current\_square\_x) (abs(- 1 current\_square\_y)))))

 )

 (t nil)

 )

 (cond ((and (numberp (setf value (aref f (abs(- 1 current\_square\_x)) (+ 1 current\_square\_y))))

 (< (setf value (aref f (abs(- 1 current\_square\_x)) (+ 1 current\_square\_y))) new\_square\_f))

 (set 'new\_square\_x (abs(- 1 current\_square\_x)))

 (set 'new\_square\_y (+ 1 current\_square\_y))

 (set 'new\_square\_f (setf value (aref f (abs(- 1 current\_square\_x)) (+ 1 current\_square\_y))))

 )

 (t nil)

 )

 (cond ((and (numberp (setf value (aref f (+ 1 current\_square\_x) current\_square\_y)))

 (< (setf value (aref f (+ 1 current\_square\_x) current\_square\_y)) new\_square\_f))

 (set 'new\_square\_x (+ 1 current\_square\_x))

 (set 'new\_square\_y current\_square\_y)

 (set 'new\_square\_f (setf value (aref f (+ 1 current\_square\_x) current\_square\_y)))

 )

 (t nil)

 )

 (cond ((and (numberp (setf value (aref f current\_square\_x (abs(- 1 current\_square\_y)))))

 (< (setf value (aref f current\_square\_x (abs(- 1 current\_square\_y)))) new\_square\_f))

 (set 'new\_square\_x current\_square\_x)

 (set 'new\_square\_y (abs(- 1 current\_square\_y)))

 (set 'new\_square\_f (setf value (aref f current\_square\_x (abs(- 1 current\_square\_y)))))

 )

 (t nil)

 )

 (cond ((and (numberp (setf value (aref f current\_square\_x (+ 1 current\_square\_y))))

 (< (setf value (aref f current\_square\_x (+ 1 current\_square\_y))) new\_square\_f))

 (set 'new\_square\_x current\_square\_x)

 (set 'new\_square\_y (+ 1 current\_square\_y))

 (set 'new\_square\_f (setf value (aref f current\_square\_x (+ 1 current\_square\_y))))

 )

 (t nil)

 )

 (cond ((and (numberp (setf value (aref f (abs(- 1 current\_square\_x)) current\_square\_y)))

 (< (setf value (aref f (abs(- 1 current\_square\_x)) current\_square\_y)) new\_square\_f))

 (set 'new\_square\_x (abs(- 1 current\_square\_x)))

 (set 'new\_square\_y current\_square\_y)

 (set 'new\_square\_f (setf value (aref f (abs(- 1 current\_square\_x)) current\_square\_y)))

 )

 (t nil)

 )

 (cond ( (eq (setf value (aref maze (+ 1 current\_square\_x) (abs(- 1 current\_square\_y)))) 'd)

 (set 'new\_square\_x (+ 1 current\_square\_x))

 (set 'new\_square\_y (abs(- 1 current\_square\_y)))

 (set 'new\_square\_f 'd)

 )

 (t nil)

 )

 (cond ( (eq (setf value (aref maze (+ 1 current\_square\_x) current\_square\_y)) 'd)

 (set 'new\_square\_x (+ 1 current\_square\_x))

 (set 'new\_square\_y current\_square\_y)

 (set 'new\_square\_f 'd)

 )

 (t nil)

 )

 (cond ( (eq (setf value (aref maze (+ 1 current\_square\_x) (+ 1 current\_square\_y))) 'd)

 (set 'new\_square\_x (+ 1 current\_square\_x))

 (set 'new\_square\_y (+ 1 current\_square\_y))

 (set 'new\_square\_f 'd)

 )

 (t nil)

 )

 (cond ( (eq (setf value (aref maze current\_square\_x (+ 1 current\_square\_y))) 'd)

 (set 'new\_square\_x current\_square\_x)

 (set 'new\_square\_y (+ 1 current\_square\_y))

 (set 'new\_square\_f 'd)

 )

 (t nil)

 )

 (cond ( (eq (setf value (aref maze (abs(- 1 current\_square\_x)) (+ 1 current\_square\_y))) 'd)

 (set 'new\_square\_x (abs(- 1 current\_square\_x)))

 (set 'new\_square\_y (+ 1 current\_square\_y))

 (set 'new\_square\_f 'd)

 )

 (t nil)

 )

 (cond ( (eq (setf value (aref maze (abs(- 1 current\_square\_x)) current\_square\_y)) 'd)

 (set 'new\_square\_x (abs(- 1 current\_square\_x)))

 (set 'new\_square\_y current\_square\_y)

 (set 'new\_square\_f 'd)

 )

 (t nil)

 )

 (cond ( (eq (setf value (aref maze (abs(- 1 current\_square\_x)) (abs(- 1 current\_square\_y)))) 'd)

 (set 'new\_square\_x (abs(- 1 current\_square\_x)))

 (set 'new\_square\_y (abs(- 1 current\_square\_y)))

 (set 'new\_square\_f 'd)

 )

 (t nil)

 )

 (cond ( (eq (setf value (aref maze current\_square\_x (abs(- 1 current\_square\_y)))) 'd)

 (set 'new\_square\_x current\_square\_x)

 (set 'new\_square\_y (abs(- 1 current\_square\_y)))

 (set 'new\_square\_f 'd)

 )

 (t nil)

 )

)

(defun solve\_maze (current\_square\_x current\_square\_y)

 (calculate\_g current\_square\_x current\_square\_y)

 (calculate\_h current\_square\_x current\_square\_y)

 (calculate\_f current\_square\_x current\_square\_y)

 (new\_move current\_square\_x current\_square\_y)

 (setf (aref maze current\_square\_x current\_square\_y) 0)

 (setq current\_square\_x new\_square\_x)

 (setq current\_square\_y new\_square\_y)

 (setf (aref maze current\_square\_x current\_square\_y) 'x)

 (print '------------------------- )

 (print maze)

 (cond ((eq new\_square\_f 'd)

 (print "Door exitted"))

 (t (solve\_maze current\_square\_x current\_square\_y))

 )

)

(search\_for\_start count\_x count\_y)

(setq current\_square\_x start\_x)

(setq current\_square\_y start\_y)

(search\_for\_door count\_x count\_y)

(print maze)

(solve\_maze current\_square\_x current\_square\_y)