**Write a C program to solve a rat breadcrumb maze**

/ Here is an example of a solved maze:  
//  
// ###########################  
// #S....#.oo..#.oooOoooooo..#  
// #####.###.#.#####o#####.#.#  
// # #.#...#.....#.#.....#.#  
// # # #.#.#######.###.#####.#  
// # # #...# #.#...#.....#  
// # ####### # ###.#.###.#####  
// # # ...# ....E#  
// ###########################

**Algorithm logic:**

runner.c file:

#include <stdio.h>  
#include <stdbool.h>  
// Including a header file lets your file refer to the functions \*declared\* in  
// that header file. The functions are usually \*defined\* elsewhere though, in  
// this case, mazelib.c  
#include "mazelib.h"

#define NORTH 0  
#define EAST 1  
#define SOUTH 2  
#define WEST 3

void runner\_solve(void) {  
// Use a few variables to keep track of the current row and column position  
// of the "runner". The runner should always start at the 'S' symbol in the  
// maze. Luckily, the 'S' symbol is always in the same place

// You should also keep track of which way the runner is facing. You can use  
// the preprocessor defines at the top of this file to make your life easier  
// instead of just having to remember "0 is North, 1 is East, etc."  
// Basically, any time you use NORTH in your code, a zero will be substituted  
// in for it at compile time.

// Ok, algorithm time:  
//  
// 1. As long as you are not standing on the 'E', do the following:  
// 2. Turn to your left  
// 3. Can you go that way? I.E., is there not a wall in front of you?  
// 4. If your path is blocked, turn to your right and go back to (3)  
// 5. If your path is open, throw down the appropriate breadcrumb (see note)  
// 6. Move forward and go back to (1)  
// 7. You are standing on the 'E', STOP, you're done!

// NOTE: By "appropriate breadcrumb" I mean this:  
// If you are in an empty square, the breadcrumb should be '.'  
// If you are in a square with '.', the breadcrumb should be 'o'  
// If you are in a square with 'o', the breadcrumb should be 'O'  
// If you are in a square with 'O', the breadcrumb should be '@'  
//  
// Here is an example of a solved maze:  
//  
// ###########################  
// #S....#.oo..#.oooOoooooo..#  
// #####.###.#.#####o#####.#.#  
// # #.#...#.....#.#.....#.#  
// # # #.#.#######.###.#####.#  
// # # #...# #.#...#.....#  
// # ####### # ###.#.###.#####  
// # # ...# ....E#  
// ###########################  
}

**An example code follows the runner.c algorithm logic:**

#include <stdio.h>  
#include <stdbool.h>  
// Including a header file lets your file refer to the functions \*declared\* in  
// that header file.  The functions are usually \*defined\* elsewhere though, in  
// this case, mazelib.c  
#include "mazelib.h"  
#include "runner.h"

#define NORTH 0  
#define EAST 1  
#define SOUTH 2  
#define WEST 3

/\*  
  \* Assignment: needs to find a way through a maze. Write the code  
 \* to send it from the START position to the END. Input two initial   
 \* arguments: an integer representing the maze width or columns [9,79]   
 \* and an integer representing the maze height or rows [9,25].   
 \* SAMPLE DATA: Two possible integers to start with are 35 and 15.  
\*/

/\* BEGIN INITIALIZE VARIABLES \*/  
int ratRow, ratCol; // Int values of Row, Column of a space in the maze, also of the Rat  
int row = 1;        // Row value of the 'S'  
int col = 1;        // Column value of the 'S'  
int ratCoords[2] = {1, 1};  // Array to hold current Row,Col values of the Rat position

char nextFourSquares[4] = {'!', '!', '!', '!'}; // CHAR array to hold 4 directional views with dummy character  
int nextFourRows[4] = {1,1,1,1};  // INT array to hold Row coordinates of each View N,E,S,W.   
int nextFourCols[4] = {1,1,1,1}; // INT array to hold Col coordinates of each View N,E,S,W.

char markersArray[8] = {'E', 'S', '#', ' ', '.', 'o', 'O', '@' };  // Array to hold characters used in logic IF-THEN statements

char crumbNew;            // Variable to hold the value of a replacement char on the maze  
bool moveUnmade = true;   // Boolean value used as a switch when evaluating the character IF-THEN statements  
bool trophyFound = false; // Boolean value used as a switch in WHILE loop until the END square is found  
/\* END INITIALIZE VARIABLES \*/

/\* BEGIN FUNCTION PROTOTYPES \*/  
char choose\_breadcrumb (char currentMarker);  // Will choose new char to place on available square  
void find\_next\_step();  // The 'brain' of this program: lets the Rat look at available squares and find next move  
void get\_north\_view (int row, int col); // Function to see what char value sits in the square to Rat's NORTH  
void get\_east\_view (int row, int col);  // Function to see what char value sits in the square to Rat's EAST  
void get\_south\_view (int row, int col); // Function to see what char value sits in the square to Rat's SOUTH  
void get\_west\_view (int row, int col);  // Function to see what char value sits in the square to Rat's WEST  
void build\_four\_views\_array();  // Once the four N,E,S,W characters are seen by Rat, save those 4 choices to an Array  
bool move\_rat();  // Function that returns a switch value of 'false' to indicate Rat HAS moved.  
/\* END FUNCTION PROTOTYPES \*/

/\* FUNCTION TO CHOOSE APPROPRIATE BREADCRUMB: either a . OR o OR O OR @   
\*/  
char choose\_breadcrumb (char currentMarker) {  
  if (currentMarker == ' ') {  
    crumbNew = '.';  
  }  
  else if (currentMarker == '.') {  
    crumbNew = 'o';  
  }  
  else if (currentMarker == 'o') {  
    crumbNew = 'O';  
  }  
  else if (currentMarker == 'O') {  
    crumbNew = '@';  
  }  
  else {  
    // This ELSE choice was used in testing.  
    printf("If this message fires, an unexpected result has happened. \n");

  }

  return crumbNew;  
}

/\* FUNCTION TO BUILD AN ARRAY OF 4 ELEMENTS. EACH ELEMENT HOLDS THE CHAR OF THE 4 SURROUNDING SQUARES.  
  THIS FUNCTION WILL CALL 4 SEPARATE FUNCTIONS THAT CALCULATE THE N,E,W, and S row,columns.  
\*/  
void build\_four\_views\_array() {  
  get\_north\_view(ratCoords[0], ratCoords[1]);  
  get\_east\_view(ratCoords[0], ratCoords[1]);  
  get\_south\_view(ratCoords[0], ratCoords[1]);  
  get\_west\_view(ratCoords[0], ratCoords[1]);

}

/\* FOUR FUNCTIONS TO LOOK AT THE N, E, W, and S SQUARES THAT SURROUND THE RAT. \*/  
void get\_north\_view (int row, int col) {  
  row = row-1;  
  char c = maze\_get\_char(row, col); //Show me the Char in the North position  
  nextFourSquares[NORTH] = c;  
  nextFourRows[NORTH] = row;  
  nextFourCols[NORTH] = col;  
}

void get\_east\_view (int row, int col) {  
  col = col+1;  
  char c = maze\_get\_char(row, col); //Show me the Char in the East position  
  nextFourSquares[EAST] = c;  
  nextFourRows[EAST] = row;  
  nextFourCols[EAST] = col;  
}

void get\_south\_view (int row, int col) {  
  row = row+1;  
  char c = maze\_get\_char(row, col); //Show me the Char in the South position  
  nextFourSquares[SOUTH] = c;  
  nextFourRows[SOUTH] = row;  
  nextFourCols[SOUTH] = col;

}

void get\_west\_view (int row, int col) {  
  col = col-1;  
  char c = maze\_get\_char(row, col); //Show me the Char in the West position  
  nextFourSquares[WEST] = c;  
  nextFourRows[WEST] = row;  
  nextFourCols[WEST] = col;

}

/\* FUNCTION TO MOVE RAT TO NEXT CHOSEN SQUARE AND DROP NEW BREADCRUMB.  
  RETURNS A BOOLEAN VALUE 'FALSE' TO CALLING FUNCTION TO EXIT A LOOP AND   
  STOP EVALUATING THE FOUR CHOICES OF MOVES ON THIS ONE STEP.  
\*/  
bool move\_rat(int i) {

  ratCoords[0] = nextFourRows[i];   // Update Rat's current maze position with chosen next square.  
  ratCoords[1] = nextFourCols[i];  
  crumbNew = choose\_breadcrumb(nextFourSquares[i]); // Choose next breadcrumb char.  
  maze\_set\_char(ratCoords[0], ratCoords[1], crumbNew); // Set the new char in the chosen space.  
    
  moveUnmade = false;   //Rat move is now 'made' so return a value of false of an "unmade" status.  
  return moveUnmade;

}

/\* FUNCTION TO DETERMINE RAT'S NEXT STEP. THIS IS THE 'BRAIN' OF THE PROGRAM.  
  THIS CALLS FUNCTIONS THAT LOOK AT THE FOUR N,E,W,S POSITIONS AND DECIDE   
  TO WHICH NEXT SQUARE THAT RAT SHOULD MOVE.  
 \*/  
void find\_next\_step() {  
    
  build\_four\_views\_array();   // Go look at the four possible squares of a next move.

  bool moveUnmade = true;     // Current status of Rat's next move is "unamde" at beginning of this loop.

  /\* HERE BEGINS A WHILE OUTER LOOP. THIS WILL ITERATE THROUGH OUR ARRAY SET OF   
    CHARACTERS THAT REPRESENT ALL POSSIBLE SQUARES ONTO WHICH RAT CAN MOVE BEGINNING WITH ARRAY ELEMENT 03.   
    THIS LOOP STOPS WHEN RAT HAS MOVED AND SENDS ITS BOOLEAN 'FALSE' SWITCH.  
 \*/  
  while (moveUnmade == true) {

    for (int marker=3; marker<8; marker++) {

      if (moveUnmade == false) {  
        marker=7;   // Once the Unmade switch is received, force this outer loop to end.  
      }

      for (int i=0; i<4; i++) {   // Here begins an inner loop to compare 4 possible next squares against our character set of available values.   
        if (nextFourSquares[i] == markersArray[marker]) {  
          moveUnmade = move\_rat(i);   // If Rat sees a good next square, go make that move.  
          i=3;  
        }  
        else if (nextFourSquares[i] == markersArray[0]) {  
          trophyFound = true;   // When Rat sees the END character, set all switches and loop iterations to stop. Game ends.  
          moveUnmade = false;  
          i=3;  
        }  
      }  
    }  
  }

// The outer and inner loops end here when the four view choices have been evaluated.  
// Return control to runner\_solve function.  
   
}  
    
/\* THIS IS THE PRE-SET FUNCTION IN THIS CLASS ASSIGNMENT.  
  THIS BEGINS A WHILE LOOP THAT EVALUATES FALSE UNTIL THE 'TROPHY' (THE END SQUARE)  
  IS SEEN BY THE RAT. IT WILL CALL THE FIND\_NEXT\_STEP FUNCTION FOR EVERY MOVE   
  WITHIN THE MAZE, ONE MOVE AT A TIME, UNTIL THE END IS REACHED.  
\*/  
void runner\_solve(void) {

  while (trophyFound == false) {

    row = ratCoords[0]; // Varibales hold whatever is the current Row, Col of the Rat.  
    col = ratCoords[1];  
    find\_next\_step(row, col); // Send those two Row,Col values to the 'brain' function of the game.  
  }

  if (trophyFound == true) {  // Print final message when the maze is completed.  
    printf(" \n" );  
    printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n" );  
    printf("\* CONGRATULATIONS! YOU REACHED THE END! \* \n" );  
    printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n\n" );  
  }  
     
}