```
function [route,numExpanded] = AStarGrid (input_map, start_coords, dest_coords)

% Run A* algorithm on a grid.

% Inputs :

%   input_map : a logical array where the freespace cells are false or 0 and

%   the obstacles are true or 1

%   start_coords and dest_coords : Coordinates of the start and end cell

%   respectively, the first entry is the row and the second the column.

% Output :

%    route : An array containing the linear indices of the cells along the

%    shortest route from start to dest or an empty array if there is no

%    route. This is a single dimensional vector

%    numExpanded: Remember to also return the total number of nodes

%    expanded during your search. Do not count the goal node as an expanded
node.


% set up color map for display

% 1 - white - clear cell

% 2 - black - obstacle

% 3 - red = visited

% 4 - blue  - on list

% 5 - green - start

% 6 - yellow - destination
```

```
cmap = [1 1 1; ...
    0 0 0; ...
    1 0 0; ...
    0 0 1; ...
    0 1 0; ...
    1 1 0; ...
    0.5 0.5 0.5];

colormap(cmap);

% variable to control if the map is being visualized on every
% iteration
drawMapEveryTime = true;

[nrows, ncols] = size(input_map);

% map - a table that keeps track of the state of each grid cell
map = zeros(nrows,ncols);
map(~input_map) = 1;   % Mark free cells - assign 1 to the elements that are "true"
map(input_map)  = 2;   % Mark obstacle cells - assign 2 to the elements that are
"true"

% Generate linear indices of start and dest nodes
```

```
% IND = sub2ind(SIZ,I,J) returns the linear index equivalent to the
%    row and column subscripts in the arrays I and J for a matrix of
%    size SIZ.
start_node = sub2ind(size(map), start_coords(1), start_coords(2));
dest_node  = sub2ind(size(map), dest_coords(1),  dest_coords(2));

map(start_node) = 5;    % can also run map(start_coords(1), start_coords(2)) = 5
map(dest_node)  = 6;

% meshgrid will `replicate grid vectors' nrows and ncols to produce
% a full grid
% type `help meshgrid' in the Matlab command prompt for more information
parent = zeros(nrows,ncols);

%
[X, Y] = meshgrid (1:ncols, 1:nrows);     % X is the value on X-axis and Y is the value on Y-axis for every cell

xd = dest_coords(1);
yd = dest_coords(2);

% Evaluate Heuristic function, H, for each grid cell
% Manhattan distance
```

```
H = abs(X - xd) + abs(Y - yd);          % calculate the distance from every cell to the destination all at once

H = H';

% Initialize cost arrays

f = Inf(nrows,ncols);                   % every cell has a f-value and a g-value

g = Inf(nrows,ncols);


g(start_node) = 0;

f(start_node) = H(start_node);


% keep track of the number of nodes that are expanded

numExpanded = 0;


% Main Loop


while true


    % Draw current map

    map(start_node) = 5;

    map(dest_node) = 6;


    % make drawMapEveryTime = true if you want to see how the

    % nodes are expanded on the grid.
```

```
    if (drawMapEveryTime)
        image(1.5, 1.5, map); % image (x, y, C): When C is a 2-dimensional m-by-n matrix, the elements of C are used as
                              % indices into the current COLORMAP to
                              % determine the color. x and y specify the
                              % locations of the corners corresponding to
                              % C(1, 1) if (x, y) is scalar and C(m, n) if
                              % (x, y) is two-element vectors.
        grid on;
        axis image;
        drawnow;
    end

    % Find the node with the minimum f value
    [min_f, current] = min(f(:));      % note that min(f) has different outputs from min(f(:)).
                                       % The latter outputs the minimum value => min_f across the whole
                                       % grid and its "linear" position in the grid => current.

    if ((current == dest_node) || isinf(min_f))
        break;
    end;
```

```
% Update input_map
map(current) = 3;
f(current) = Inf; % remove this node from further consideration
numExpanded = numExpanded + 1;
% Compute row, column coordinates of current node
% [I,J] = ind2sub(SIZ,IND) returns the arrays I and J containing the
% equivalent row and column subscripts corresponding to the index
% matrix IND for a matrix of size SIZ.
[i, j] = ind2sub(size(f), current);


%
*******************************************************************
***
% ALL YOUR CODE BETWEEN THESE LINES OF STARS
% Visit all of the neighbors around the current node and update the
% entries in the map, f, g and parent arrays
%


% Find the neighbors of the current cell (i, j) - only clear cells are
% considered
n = AStarNeighbors (map, i, j);
% numExpanded = numExpanded + length(n);
```

```
    k = g(n) > (g(current) + 1);    % edge cost from current cell to each of its
neighbors is 1

    indx = find (k ==1);            % the index of the neighbors in list n that have
higher g-value than current g-value

    g(n(indx)) = g(current) + 1;    % update g-value for current's neighbors

    f(n(indx)) = g(n(indx)) + H(n(indx));   % update f-value for the current's
neighbors and add them to f-list

    parent(n(indx)) = current;

    map(n(indx)) = 4;               % blue (4) = on list

%*******************************************************************
****

end

%% Construct route from start to dest by following the parent links

if (isinf(f(dest_node)))

    route = [];

else

    route = [dest_node];

    while (parent(route(1)) ~= 0)

        route = [parent(route(1)), route];

    end
```

```
    % Snippet of code used to visualize the map and the path
    for k = 2:length(route) - 1
        map(route(k)) = 7;
        pause(0.1);
        image(1.5, 1.5, map);
        grid on;
        axis image;
    end
end

end

function n = AStarNeighbors (G, I, J)
% find the neighbors of cell (I, J) on a 2D grid
% Input: G - the 2D grid presented as a matrix
%       (I, J) - the coordinate of the cell on the grid, measured from the top-left
%                corner as (1, 1), I = #row, J = #column
% Output: a list of linear positions of the neighbors on the grid
nrows = size(G, 1);
ncols = size(G, 2);
i = 1;
n = [];
```

```
if I - 1 > 0 && (G(I-1, J) == 1 || G(I-1, J) == 6)    % if the upper cell is still inside of the grid and it is a clear cell or destination

    n(i) = sub2ind(size(G), I-1, J);

    i = i+1;

end

if I + 1 <= nrows && (G(I+1, J) == 1 || G(I+1, J) == 6)    % if the lower cell is within the grid and it is a clear cell or destination

    n(i) = sub2ind(size(G), I+1, J);

    i = i+1;

end

if J - 1 > 0 && (G(I, J-1) == 1 || G(I, J-1) == 6)

    n(i) = sub2ind(size(G), I, J-1);

    i = i+1;

end

if J + 1 <= ncols && (G(I, J+1) == 1 || G(I, J+1) == 6)

    n(i) = sub2ind(size(G), I, J+1);

    i = i+1;

end

end
```