CSC112 Spring 2011
Fundamentals of Computer Science
Lab 12

In this week's lab, we will gain experience with recursion. The problem is use recursion to search for a path through a maze. The maze is defined by a 2-D character array with m rows and n columns, together with a start position and an desired ending position. At each moment in time, your recursive search will keep track of a current position i, j. The rules of the game:

1. On each step, the virtual "lab rat" in our maze can move one position up, down, left or right, with some restrictions.
2. The virtual rat may not move outside the boundaries of the maze.
3. The 2-D character array contains characters significant to our virtual rat.
   (a) A '|' character blocks the path. Our virtual rat can not enter a square occupied by an '|'. The rat may not alter a position in the maze containing an '|'.
   (b) A blank ' ' character indicates a free spot into which the virtual rat may move.
   (c) Our rat may leave a trail of virtual "bread crumbs" to prevent getting stuck going around in circles. A blank ' ' character may be changed to a '.' character, indicating a "bread crumb".
   (d) When our rat backs out of an unsuccessful path the rat can "pick up the bread crumb". I.e., change the '.' character back to a blank ' ' character.
   (e) For purposes of debugging and tracing the progress of the rat, you can change positions containing '.' or ' ' to '*', indicating the position of the virtual rat. Make sure you change it back when you move the rat. If your code is bug-free you will not need mark the current position of the virtual rat, since the search algorithm keeps track of the current position as a pair (row,column) of integers.

The maze may be downloaded from:

`http://menehune.opt.wfu.edu/CSC112/Lab12/maze.h`

Use the `#include` directive to include file `maze.h`

Positions in the maze (i, j) represent row i and column j. When you find a path, your program should output the path as a sequence of legal moves leading to the destination. For example:

```
(0,0) to (1,0)
(1,0) to (2,0)
(2,0) to (2,1)
  .
  .
  .
```

In addition, your program should print out the maze as a table, with each position along the solution path marked by a '.'. I.e., the "bread crumbs" show the path our rat followed. For example:

```
.|||||     |
....|| ||| |
 ||.||...|
 ||.||.|.||
 ||....|.||
 || ||||.|
 || ||||.  |
    ||||.| |
|   |....|||
  |||.| ||
 ||||.| |  |
   | .||
```

Pseudo-code for Searching with Backtracking In this pseudo-code, `move_stack` is a stack that keeps track of all essential moves leading up to the current position. As you make each move, you record it on the stack. As you undo each unsuccessful move, you remove it from the stack. The move stack must be passed by reference, so that there is essentially one stack for the entire computation.

```
search( maze, current_position, move_stack )
{
    if ( current_position == destination ) {
        push the current (final) position onto the move stack ;
        mark the current (final) position with a bread crumb ;
        output the path contained in move_stack ;
        return "success" ;
    }
    else {
       push the current position onto the move stack ;
       mark the current position with a bread crumb ;
       for every move m, up down, left, and right {
          compute a new_position (apply move to current_position);
          if ( (new_position is valid) and
               (there is no bread crumb in the new_position)) {
                   search( maze, new_position, move_stack ) ;
                   if search was successful, return "success" ;
          }
       }
       pick up the bread crumb from the current position ;
       pop the current position off the move stack ;
       return "unsuccessful" ;
    }
}
```

Turn in: Change to the directory containing the sub-directory "Lab12". Create a file named "lab12.tar" using the command: `tar cf lab12.tar Lab12`. Upload the file "lab12.tar" to your account on telesto.