COM S 327, Spring 2016
Programming Project 1.03
Path Finding

So far, we've got this lovely dungeon. And we can. . . save and restore it. And, you know. . . look at it. That's about it. Kind of boring.

Once you have monsters (next week), they (at least the smart ones) will need to find a path to the player through the dungeon. To find that path, you'll need to implement a path-finding algorithm. We're going to have some monsters that can tunnel through walls and others that can only move through open space, so we'll actually need two slightly different pathfinding algorithms. In both cases we'll use Dijkstra's Algorithm, treating each cell in the dungeon as a cell in a graph. For the non-tunneling monsters, we'll give a weight of 1 for floor and ignore wall cells (i.e., don't try to find paths through walls). For the tunnelers, we'll have to use weights based on the hardness; cells with a hardness of 0 have a weight of 1, and cells with hardnesses in the ranges $[1, 84]$, $[85, 170]$, and $[171, 254]$ have weights of 1, 2, and 3, respectively. A hardness of 255 has infinite weight. We don't have to assign a value to this. Instead, we simply do not put it in the queue.

A naïve implementation will call pathfinding for every monster in the dungeon, but in practice, every monster is trying to get to the same place, so rather than calculating paths from the monsters to the player character (PC), we can instead calculate the distance from the PC to every point in the dungeon, and this only needs to be updated when the PC moves or the dungeon changes. Each monster will choose to move to the neighboring cell with the lowest distance to PC. This is gradient descent; the monsters move "downhill". Unless the monster is already collocated with the PC, there is always at least one cell with a shorter distance than its current cell. In the case of multiple downhill cells having the same distance, the monster may choose any one of them.

Take the following map with PC, '`@`'.

```
                                                                 ....
                                               .....             ....
      ....                                     .....             ....
      ....                                     .....             ....
      ....                                     #                 ....
      ....####### ..........               #####                 ....
      ....#     ##..........######..........                     ....
      ....               #        .......@..            #########....
     ##                  #        ..........            #        #
......                   #                 #        #####        #
......                   #                 #        #          ###
......                   #                ..........#          #
......                   #                ..........           #
......                   ....             ..........          ##
......                   ....             ..........         ##
......                                    ..........        ##
......                                    ..........#####....
                                                         ....
```

Distances from the PC are marked with a single letter in the order 0–9, a–z, A–Z.

```
                                                                 wwww
                                               9999a             vvvv
      zyyy                                     8889a             uuuu
      zyxx                                     7789a             tttt
      zyxw                                     6                 ssst
      zyxwvutsrqp nmlkjihgfe               23456                 rrst
      zyxwv     ponmlkjihgfedcba987654321112                     qrst
      zyxw               g        7654321012            iijklmnopqrst
     Az                  h        7654321112            h        r
FEDCBA                   i                 2        defgh        q
FEDCBB                   j                 3        c          ppq
FEDCCC                   k                44456789abc          o
FEDDDD                   l                55556789ab           n
FEEEEE                   mmno             66666789ab          mn
FFFFFF                   nnno             77777789ab         lm
GGGGGG                                    88888889ab        kl
HHHHHH                                    99999999abcdefghijk
                                                         hijk
```

A distance map for tunneling monsters would include the rock cells.

Dijkstra's Algorithm is described here: http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm Scroll down to find the pseudocode under "Using a priority queue". Obviously, you'll need a priority queue, one with a decrease priority operation. You may use the Fibonacci queue that I provided with my solution to 1.01. You may use the binary heap that I'm posting with this assignment. Or you may implement any other priority queue you like.

My corridor building code uses a modified Dijkstra's algorithm, so you may start with that or start from scratch.

To test your code, select a random floor point in the dungeon for your PC, which you will render with an '`@`'. Render your dungeon with the PC. Then render your non-tunneling monster distance map, marking distances as above. For distances greater than 61 ('`Z`'), render the dungeon normally. Repeat for the tunneling monster distance map. Note that your distance maps will be integers from zero to some max value. We are only *displaying them* using the values above, not storing them that way.

Your submission, when run, should generate a dungeon, calculate all both distance maps, render all three views of the dungeon, and exit.

All code is to be written in C.