Problem 4

Every classic RPG had a dungeon. It's a tradition as old as pencils and Cheeto-stained graph paper. For this problem, you'll be working with a 2D maze on a **22x22 square grid**. Every grid square is either completely filled or completely empty, and is represented by a 2D **row-major** array of integers telling you whether the cell is **walkable (1)** or **blocked (0)**. You may assume that the maze is sealed; that is, all the cells along the four edges (those with an X or Y coordinate of 0 or 21) will be set to zero. You can rely on this, and you do not need to check for it in your code. You can move in the four cardinal directions (up, down, left, and right) between any pair of adjacent walkable cells, but you cannot move diagonally or through blocked cells. The maze is not necessarily connected; there may be walkable cells that can never be reached from certain other walkable cells. A smaller 9x7 maze would be declared like this in C#:

```
var mazeDef = new int[7,9] { { 0,0,0,0,0,0,0,0,0 },
                             { 0,1,1,1,1,1,1,0,0 },
                             { 0,0,0,0,0,0,1,1,0 },
                             { 0,1,1,1,1,1,1,0,0 },
                             { 0,1,0,1,0,0,1,0,0 },
                             { 0,1,0,1,1,1,1,0,0 },
                             { 0,0,0,0,0,0,0,0,0 } };
```

Your task is to write a function that will take as parameters a 22x22 2D array and (X,Y) coordinates for a start and end cell. You must return a single integer indicating the **number of cells in the longest possible non-overlapping path** through the two points, or else **-1 if no such path is possible**. In C# your function will look something like this:

```
int longestPath(int[,] maze, int startX, int startY,
                             int endX, int endY);
```

If you were working with the example maze above instead of a full 22x22 maze, some sample inputs and their expected outputs would be:

longestPath(mazeDef, 1,1, 6,1) == 6 *(straight across the top corridor)*
longestPath(mazeDef, 1,1, 1,2) == -1 *(can't reach a blocked cell)*
longestPath(mazeDef, 3,3, 4,3) == 10 *(go around the loop through 6,5 in the bottom right)*
longestPath(mazeDef, 3,3, 3,3) == 1 *(can't go around the loop because we'd overlap our path at the end)*

You may write any subfunctions, declare any data structures, and allocate any memory that you like. You may use system library functions freely, including the C++ STL, C#'s System.Collections.Generic and LINQ, or any other container and/or algorithm classes that come with the latest version of your language. You may destructively modify the passed-in maze data if you want.