**CS 255 Assignment #3 Due: start of class, Wed. 4/11**

**This is an individual assignment. Do your own work. You may not discuss any aspect of the problem (design, approach, code) with anyone but the course instructor. You may use C++ language references (Visual Studio context help with MSDN & cppreference.com), but may not search for algorithms or code solutions online. You must do all the phases of software development discussed in class yourself. Use programming features discussed in class to solve the problem. You may not use ones that haven't been discussed in class before the assignment was given.**

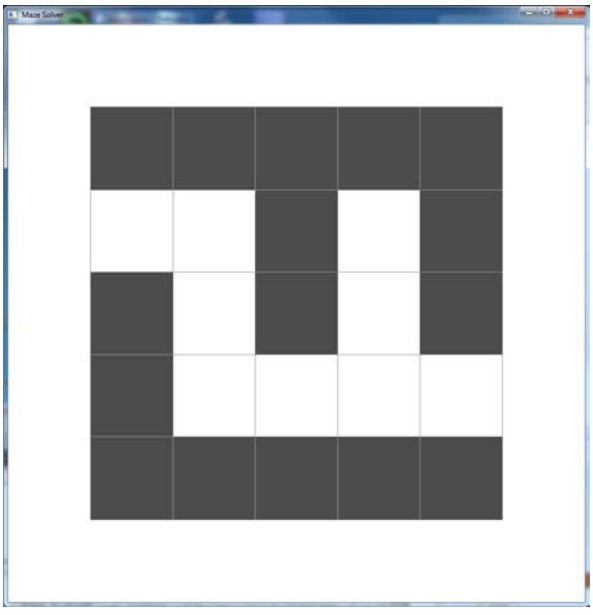
This assignment requires you to create a maze solver using an input file, several STL container classes, and the provided graphics code. The input file that represents the maze will have the following format:

```
numrows numcols // number of rows and columns of data - positive ints
d0 d1 ... dnumcols-1 // first row of 0/1 cell data
d0 d1 ... dnumcols-1 // second row of 0/1 cell data
.
.
.
d0 d1 ... dnumcols-1 // last row of 0/1 cell data (numrows rows all
together)
```

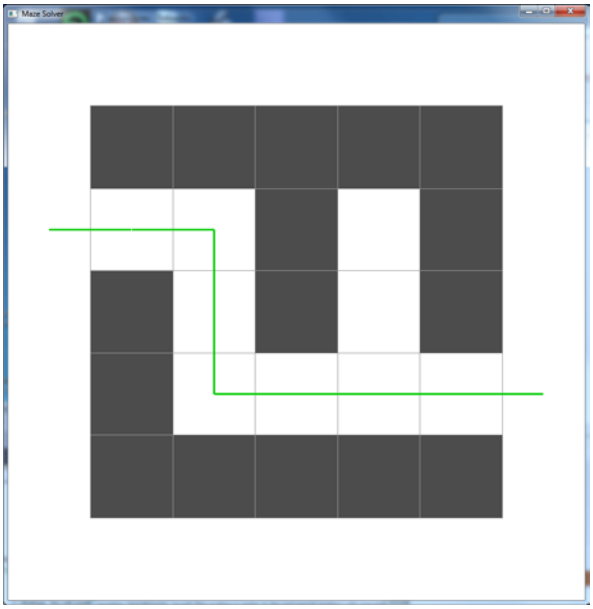
Each data item after the first row is either 0 or 1; 0 indicates open space, 1 indicates a wall. Here's an example of a very simple maze file (maze5_01.txt):

```
5 5
1 1 1 1 1
0 0 1 0 1
1 0 1 0 1
1 0 0 0 0
1 1 1 1 1
```

This represents a maze that looks like this:



After running your program, the path through the maze should look like this:



All the provided maze files have an open space start position one row below the top-left corner and an open space end position one row above the bottom-right corner as shown in the pictures above. Based on the number of rows and columns in the user-chosen maze file, that's how you choose the start position and end position you want to reach.

**This assignment will have you use STL lists, stacks and queues.**

## Specific Requirements

1. Create your own new C++ project using Visual Studio or another IDE.
2. Add a constructor to the provided position struct to let you specify what the row and col should be when you create a position. Also add a member function to let you compare two positions (i.e., determine if they are or are not the same position) to make it easier to check if you've reached your goal.
3. After asking the user for the name of the maze file and using it to initialize the global variables (rows, columns, and mazeArray), create position structs for the start and end positions based on the specified maze file.
4. Each path that you extend to try to find a path through the maze will be stored in **an STL queue** and each path on the queue will be **an STL stack** with the start position on the bottom and the last position along that path on the top. You'll start with a stack with just the start position on it and have that be the first path on your queue.
5. To search for the path through the maze, you'll get the path at the front of the queue and remove it. If that path has reached the end position (it's on the top of that stack), that will end your search loop. Otherwise, you'll extend that path using appropriate moves and put each appropriate extension at the end of the queue. Appropriate extensions push a new position onto the top of the path stack that you're extending. **Don't use diagonal moves**; only left, right, up, or down, and only if that move would take you to an open space (not a wall space) that's in the maze based on its dimensions.
6. When the path at the front of the queue has reached the end position of the maze, create **an STL list** called solution (which the graphics code uses) that is a list of positions **in order from start to end**.
7. Before the call of start_graphics_loop, use an iterator to **output all the positions on the solution path as ordered pairs from start to end**.
8. **Submit your program** into the class folder's drop box folder. Make a folder using your name in the drop box of our class folder and copy your source .cpp file into that folder. Use Windows to copy your source file into the folder you just made (should be able to do with a single copy and paste or drag and drop.) Alternatively, you may make the source file have your name in its name and upload it to the drop box using the web interface (pathfinder.bloomu.edu) from off campus. **Turn in a printout of your source code** to me.