Assignment DS5

Due Date: November 10

Purpose

You will use the stack abstract data type (ADT) to solve a problem you're quite familiar with.

Problem

In DS4, you solved the problem of finding paths through a cave. While the USGS is happy with your solution, it turns out that the autonomous vehicles can be programmed only with a low-level assembly language. Such a language has no recursive capabilities. Thus, you now have to rewrite your previous solution in a non-recursive fashion. This means that you will now need to keep track of the position of the vehicle using a stack rather than letting recursion store that information. Once you write this solution in C++, the agency will translate your code to assembly that can then run on the vehicle.

Input

The input is exactly as it was for the previous problem. Your program should prompt the user for a file name. The program should then read the plain text file, the first line of which contains two integers representing R and C, where the maximum number of rows R or columns C is 50. Note that it is possible $R \neq C$. What follows is a grid of size $R \times C$ containing characters. A '_' at a grid location means there is an open path; an 'X' at a grid location means that there is a blockage. The edges surrounding the grid are assumed to be blocked, as well. The path will always be at most one space wide. Paths that diverge will never come back together. The starting point will always be at location (0, 0) in the upper left corner.
For example, a 10×10 grid might look like the "Initial Grid" shown below left:

Initial Grid	All Paths	Direct Path
_XXXXXXXXX	0	0
_____XXX_X	00000 0	00000
X_XX_XXX_X	0 0 0	0
X_XX_XXX_X	0 0 0	0
__XX_____X	00 00000	00
_XXXX_XXXX	0 0	0
_XXXX_XXXX	0 0	0
XXXXX__XXX	00	00
XXXXXX____	0000	0000
XXXXXXXXX_	0	0

A sample input file is available on the course web page.

Output

As before, the vehicle should search the entire grid from the upper left corner to the lower right corner by following all open pathways. The output should be the initial grid, followed by the grid showing the entire search path(s) the vehicle took to go from the upper left corner to the lower right. Next, the direct path from the start to the end should be displayed. For the latter two displays, only the path should be shown (not the entire grid). For example (using the input above), the figure above shows the complete search path ("All Paths") and the path that goes directly

from the start to the finish ("Direct Path"). These become the maps the USGS will use for further processing. **Note:** *The grid/paths should not be displayed side-by-side; the output shown above is for space efficiency only!* Finally, a new added feature is that the number of dead ends will be displayed. In the above example, the number of dead ends would be two.

Note that the vehicle **must** follow a path! There can be places in the cave where there are paths that can not be reached from the starting location! You can not just search for blank locations; you must actually follow a path.

Specifics

- You must create a class called "cave." All of the methods and data relating to your application program must belong to this class (*not the stack implementation! – see below*).
- The constructor should read the file and initialize the cave.
- Your class should contain the following public methods:

displayCave()	–	display the original cave
searchPath()	–	simulate the vehicle searching through the cave
displayPaths()	–	display all the paths through the cave
displayDirect()	–	display the direct path (only!) through the cave
deadEnds()	–	return the number of dead end paths in the cave

  Any additional methods should be private.
- There should be **no** use of recursion in this project. This is either good news or bad news, depending on your view of the situation...
- You will need a stack to keep track of where the vehicle has been. It should be implemented as a template class so that it can store any type of data. The implementation should be accomplished with an array; the maximum size you can infer from the maximum size of the legal grids that will be input. Furthermore, it should be written in its own `stack.h` and `stack.cpp` files separate from your search implementation. The ADT functions the stack should support should be:
  - the stack class should be called `stack`
  - a constructor that initializes an empty stack
  - `push(x)`, where `x` is any type of data that is pushed onto the top of the stack
  - `top()`, which returns the top item of the stack
  - `pop()`, which removes the top item of the stack
  - `isEmpty()`, which returns true if the stack is empty
  - `isFull()`, which returns true if the stack is full
- Your program should promote code reuse. That is, identical or very similar code should not be repeated from one function/method to another. Take advantage of OOP's capabilities and features.

- Your `main()` should be a test program almost identical to the previous project:

```
int main () {
   cave theCave;
   cout << "Initial grid: " << endl;
   theCave.displayCave();
   theCave.searchPath();
   cout << "Path(s) found: " << endl;
   theCave.displayPaths();
   cout << "Direct Path found: " << endl;
   theCave.displayDirect();
   cout << "Number of dead ends: " << theCave.deadEnds() << endl;
   return 0;
}
```

  Note that **none of the public methods may have any parameters**.

Notes

You do not need dynamic allocation of any arrays. Just set a global **constant** to the maximums (see above) and create that size array.

Submit your source code the usual way, using the usual naming conventions. Turn in hard copy of your code at the beginning of class on November 11^{th}.

The answer is either m or something else.
– Eva Ma, one of my grad school professors