Assignment DS4

Due Date: October 27

Purpose

The main purpose of this assignment is to help you slide into the wonderful world of recursion. The recursive solutions run the gamut from easy (input and output) to more challenging (finding all paths; see below). You should also practice using good OOP techniques, meaning that you should design a class containing good private/public members and reuse code as much as possible.

Problem

The United States Geological Survey (USGS) is the mapping agency for the country. One of the most challenging areas to map are underground cavern systems. The agency has borrowed technology from other government agencies in the form of small, autonomous vehicles. These vehicles can roam through caves and help map the area.

The problem with such a vehicle is that software is needed to guide it. You have been enlisted to write a search algorithm in C++ that will guide the vehicle through any cave system automatically. Finding every path is inherently difficult; recursion is used routinely in such applications. You will write a recursive search algorithm that will map out all of the paths in a given cave.

Input

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

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. Lastly, only 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!*

Note that the vehicle is not smart, and will simply follow **all** possible paths. Most of the paths will result in dead ends. Some will emerge out of the cave (which means that the vehicle reached an edge of the grid). Thus, the vehicle will backtrack much of the time. This is where the recursion is so useful; let the computer keep track of the vehicle's position!

Specifics

- You must create a class called "cave." All of the methods and data must belong to this class.
- The constructor should read the file and initialize the cave. You may wish to call an additional method from the constructor.
- 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

  Any additional methods should be private.
- Here's the big one: **NO LOOPS OF ANY KIND are allowed in this program!** *All* of your repetition must be done with recursion.
- 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; it should be very short and call the above methods, similar to the following:

```
int main () {
   cave theCave;
   cout << "Initial grid: " << endl;
   theCave.displayCave();
   cout << "Searching for path(s) through cave..." << endl;
   theCave.searchPath();
   cout << "Path(s) found: " << endl;
   theCave.displayPaths();
   cout << "Direct Path found: " << endl;
   theCave.displayDirect();
   return 0;
}
```

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

Notes

As usual, write one method at a time. Begin by reading the initial cave and then displaying it. Then add other methods one at a time. Note that the most difficult method may be displaying the direct path. Leave this for last.

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

The solution to this problem is not very long, due to the use of recursion. However, it may not be particularly easy, due to the use of recursion!

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 October 28^{th}.

At the recurrent end of the unending...
– T.S. Eliot