Project pp11

Programming project due the week of 12/2

Due dates:
Section 01 (Block H): 11:00 pm on Wednesday 12/4
Section 02 (Block E): 11:00 pm on Thursday 12/5

Project pp11: 10 points. Find your way through a maze using recursion.

You will be given a maze problem that your program must solve, and must do so using recursion. You may use loops to read in data, print it out, initialize arrays, and such. But finding the solution must use recursion (actually, you would not want it any other way!). The maze will be a rectangular grid, which we will call a region. The maximum size for a region is 15 rows by 15 columns. Somewhere in the region is one occurrence of the char ’O’ (the capital letter ”O”), which stands for the goal that you are trying to reach. Every other character in the region is either a ’+’ or a space. A ’+’ is a wall – you cannot go through it. A space is a place you can go. You will be given a starting point in the region that is a space. Your job is to find a path from the starting point to the ’O’, if there is one.

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 & 4 \\
0 & + & + & & \\
1 & + & + & & \\
2 & + & 0 & & \\
3 & + & + & + & \\
4 & + & + & & \\
\end{array}
\quad
\begin{array}{cccc}
0 & 1 & 2 & 3 & 4 \\
0 & X & + & & \\
1 & X & + & & \\
2 & X & X & & \\
3 & X & X & + & \\
4 & X & X & + & + \\
\end{array}
\]

(a) Sample maze  
(b) Solution

In (a) above is a sample 5 by 5 maze. Let the starting point be location (0,0) – that’s row zero, column zero. One solution is shown as a trail of ’X’s shown in (b). In general there can be many solutions – you need find only one. Note that no diagonal moves are allowed – each move must go up, down, left or right.

Your program should do the following.

- Read the size of the region, which is the number of rows, followed by the number of columns. The region can vary in size from 2 to 15 rows and from 2 to 15 columns.

- Read a starting point, which is a row number, followed by a column number that will be within the region.

- Read in the characters in the region. You will be given the region on a character by character basis. The format has each row written on a separate line. For example, a 5x5 region will have 5 characters on the first line representing the first row (i.e., row 0), followed by 5 characters on the second line representing the second row (row 1), etc. Thus, the input to your program for the above example would be as follows.
Notice that we use the character ‘-’ instead of each space. For your convenience, a file containing the test data shown can be found in:

/comp/11/projects/maze.sample

- Find a solution if one exists. Print out "#Yes" if a solution exists. Otherwise, print out "#No".
- If a solution exists, print out the maze that shows your solution path using cout. When you print the maze, you should print one row to a line, with a ‘$’ just before each row and another ‘$’ just after it. For the maze above, your program should print as follows. Again, print the character ‘-‘ instead of each space.

```
#Yes
*X-++*
*X-+-*
*X+XXO*
*X+X++*
*XXX++*
```

The grader will only look for either "#Yes" or "#No", and if it is "#Yes", it will read the resulting maze that you printed. The only characters it is concerned with are the ’X’s and the ’O’ inbetween the ‘$’s. If you have other characters in the maze, that’s OK – the grader will ignore them.

I suggest that you avoid using strings when representing the region. Instead, use a 2-dimensional array of characters. Note that this 2-D array will not contain strings – just a collection of individual characters (there is a difference). The array should be 15x15. If the given maze is smaller, you will use only a portion of the array.

You must use recursion to find the solution. I suggest that you write a recursive function as follows.

```cpp
bool follow (char A[15][15], int nrows, int ncols, int row, int col);
```

“follow” takes the region (the array plus the size of the region, namely, nrows and ncols) plus a point to continue from, i.e., (row, col). It returns true if it finds a path to the ’O’ starting from (row, col), and leaves the array A marked with ’X’s to show the path. If it fails to find a path to the ’O’, it returns false and removes any ’X’s it may have added.

The strategy is as follows. As you search each (row, col) in the array A, you mark it with an ’X’. Then you perform recursive calls to go up, then down, then left, and then right, in an attempt to extend the path by trying each possible direction. If one of the recursive calls succeeds, then you will have found a path to the goal! Leave the mark there – you’ll need it when you later print out the array. But if all the recursive calls fail, then you must get rid of the ’X’ at the current (row, col) since this current step in the path will not work.
Of course, there are several base cases to get right. For example, what do you do when you go out of the region? Or when you hit a wall or the 'O'?

If your program seems to be in an infinite loop, you are probably not marking the array correctly. Let’s say that you don’t mark it with 'X' as you go. Look at the example from above. We start at (0,0), don’t place the ‘X’, and go down. From (1,0) we go, say, up. Hmmm ... we’ve already been here before. OK, from (0,0) we go down (again). And from (1,0), we go up (again). I trust that you realize that this can go on forever. However, if you mark the array with ‘X’s as you go, when you come back to an 'X’, then you ... (you figure it out).

Advice: The first thing to do is write functions to read the maze and to print it using cout. Make sure that these functions are working perfectly before writing the recursive routine. When testing your "follow" function, first test it on a very small maze (e.g., 2x2 or 3x3) and trace every move. In other words, every time follow is called, print out the value of its arguments and then print the maze (if you do this for large mazes, you’ll get a lot of output). Capture the output in a file using Unix re-direction so you can see it with Emacs (the output will almost definitely be more than one screenful). Then you’ll see your code working (or not working!). To run any program in Unix that reads its input from the file inputfile and puts its output on the file outfile, type:

```
%a.out <inputfile >outputfile
```

Keep working with small mazes (2x2, 3x3, 4x4) until you are sure that your program works. Then "comment out" the routines that trace the follow function. (By "commenting out" I mean put a ‘//’ at the start of each line. This turns it into a comment and gets ignored. But if you need it again, you can later un-comment it.) If you are still having trouble, put a cout along with each base case that prints out which base case has been discovered. If no base case prints out, then you know it went into the recursive step. Good luck!

You must use functions and recursion for this project. If you fail to use functions and recursion, you will receive a zero for the entire project! However, you may use iteration to read, print and initialize the maze.

How to submit your program

Assume that your program is in a file named maze.cpp. To submit it, type:

```
provide comp11 pp11 maze.cpp
```

from an Andante computer. You can submit up to 3 times. Your grade will be the highest grade you receive out of all your submissions.