Mr. Bellavia

LESSON 04 – 2D ARRAY PROBLEM SOLVING

In this lesson we will look at an example of how we can solve basic computer problems using 2d-arrays whereby we use an input file that is read into our program, which will then produce output representing a solution.

Sections:

I. 2D ARRAY PROBLEM SOLVING EXAMPLE:

There will be certain problems that require some type of data grid to be processed to solve a problem. This usually requires reading in grid data into a 2d-array and then traversing that array to solve a problem. Let's look at a concrete example:

Problem: Star-Cluster

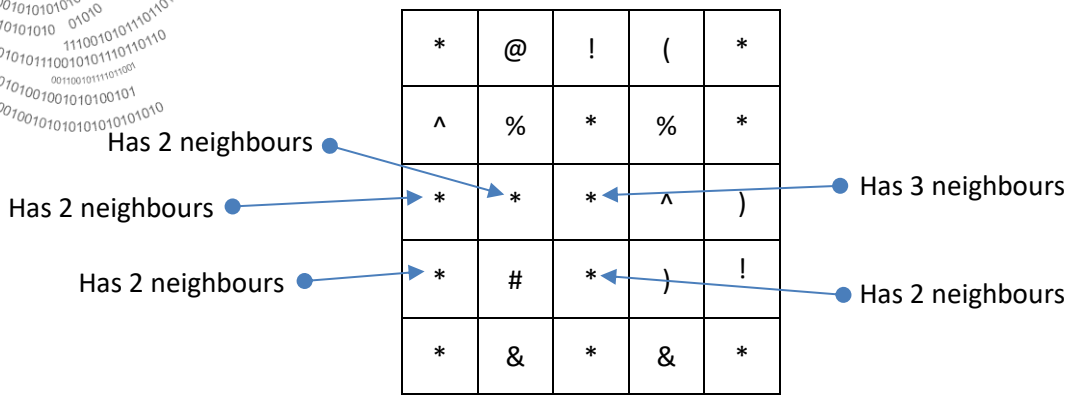
Problem Description:

Consider a grid of characters:

*	@	!	(	*
^	%	*	%	*
*	*	*	^	)
*	#	*	)	!
*	&	*	&	*

Assume that a **star-cluster** is made up of an **asterisk (*)** with at least **2 other neighboring asterisks** that appear to the left, right, above, or below**.** With the above grid we can see that there are **5 star-clusters:**

Mr. Bellavia



Write a program that finds all the star clusters that exist in a grid.

Input Specification:

The first two lines are two integers M (M > 0 and M <= 1500) and N (N > 0 and N <= 1500) representing the number of grid columns and rows respectively. The following N lines are the rows of the grid each containing M character values separated by a space.

Output Specification:

The output will be one integer representing the number of star-clusters that exist in the grid.

Sample Input:

```
5
5
* @ ! ( *
^ % * % *
* * * ^ )
* # * ) !
* & * & *
```

Sample Output:

```
5
```

Explanation:

There are 5 star-clusters in the grid above (i.e., there are 5 asterisks (*) that have at least 2 neighbours).

To solve this problem, we need to read the grid into a 2d-array, then traverse this array and count how many asterisks have at least two neighboring asterisks. Here is a solution for this question:

```
// open input file 'input.txt'
StreamReader sr = new StreamReader("input.txt");

// get grid size
int M = Convert.ToInt32(sr.ReadLine());
int N = Convert.ToInt32(sr.ReadLine());

// declare grid
char[,] grid = new char[N, M];
string[] row = new string[M];

// read grid into 2d array
string? line = "";
for(int x = 0; x < N; x++)
{
    line = sr.ReadLine();
    row = line.Split(' ');
    for(int y = 0; y < M; y++)
        grid[x, y] = char.Parse(row[y]);    // convert string to char
}

// traverse grid and determine number of star-clusters
int numStarClusters = 0;
for(int x = 0; x < N; x++)
{
    for(int y = 0; y < M; y++)
    {
        int numAsterisks = 0;

        // if the current character is an astrisk then we can check the neighbours
        //   and increase 'numAsterisks' appropriately
        if (grid[x,y] == '*')
        {
            numAsterisks++;

            // check above neighbor if in bounds
            if(x > 0 && grid[x-1,y] == '*')
                numAsterisks++;

            // check below neighbor if in bounds
            if(x < N - 1 && grid[x+1,y] == '*')
                numAsterisks++;

            // check left neighbor if in bounds
            if(y > 0 && grid[x,y-1] == '*')
                numAsterisks++;
```

```
47
48              // check right neighbor if in bounds
49              if(y < M -1 && grid[x,y+1] == '*')
50                  numAsterisks++;
51
52              // if there are at least 3 asterisks then increment 'numStarClusters' by 1
53              if(numAsterisks >= 3)
54                  numStarClusters++;
55          }
56      }
57  }
58
59  // close input file
60  sr.Close();
61
62  // output number of star-clusters
63  Console.WriteLine(numStarClusters);
```

First, we open the input file **(line 2)**, then we read in the number of columns (M) and rows (N) on **lines 5 & 6.**

Line 9 we declare a grid of chars (2d-array) of size N by M.

Line 10 we create a string array of size M that will hold the current row being read in from the file.

Lines 14 to 15 will read in the data from the input file into the 2d-array of chars. Notice **line 19:**

```
grid[x, y] = char.Parse(row[y]);
```

char.Parse() is a built-in char method that will convert a given string to a char. When we used the .Split() method to split the current row of values read in from the input file on **line 17**, each value of that split was saved into a string array. Therefore, the value of row[y] is a string and needs to be converted to a char so that it can be stored in our 2d-array of chars grid[x, y].

Lines 24 to 57 is the bulk of our solution. We traverse the grid using a nested for-loop. For each iteration of the inner-loop we set 'numAsterisks' to 0, then we check if the current grid value, grid[x, y], is an asterisk. If it is, then we can increment 'numAsterisks' by 1 and check all the neighbours. If the current grid value at grid[x, y] is not an asterisk then there is no need to check the neighbours.

Lines 36 to 54 will check the above, below, left and right neighbours of our current position grid[x,y]. We can access the appropriate neighbour by adding or subtracting a 1 to x or y. For example, to access the neighbour above, we can say grid[**x – 1**, y]. One thing to be careful however is to not go **out of bounds** in our array. This could happen at the edges of our grid. For example, if our current

position is grid[0, 2] then if we subtract a 1 from the 'x' index (0) to access the neighbour above, we will end up with grid[**-1**, 1] which will give you a **run-time error** saying that you went **out of bounds.** Therefore, we must first check that our current index 'x' is not at the edge before we check the neighbour:

```
if(x > 0 && grid[x-1,y] == '*')
    numAsterisks++;
```

Notice that we first make sure that **'x' is at least 1.** Then we use a **logical &&** to test if the neighbour is an asterisk. In C#, using a logical && has an advantage where if the left condition is not true, then the right condition will not execute. Therefore, in our case, if x is not greater than 0 then the grid[x-1, y] will not be tested, and hence we will not get an out of bounds error.

Here is a breakdown of how we test each edge:

- x > 0 → Used for the top edge of grid
- x < N – 1 → Used for the bottom edge of grid (N is the max number of rows which equals 5, therefore our largest row index for grid[x, y] will be 5 -1 = 4
- y > 0 → Used for the left edge of grid
- y < M – 1 → Used for the right edge of grid (M is the max number of columns which equals 5, therefore our largest column index for grid[x, y] will be 5 -1 = 4