COMP 281 Report.

Problem 1081.

This Problem required me to write a program in C that would simulate the "Game of Life" by John Conway. The game works by simulating a grid of cells. Each cell has either two states, Dead ('.') or alive ('X'). The game logic runs on a step by step basis. The state of each cell can change once during each time step depending on the state of its neighbouring cells. For example, a dead cell can only change to an alive cell if it has exactly 3 alive neighbouring cells. An alive cell can only change to a dead cell if it has either less than two alive neighbours or more then 3 alive neighbours. Any dead or alive cell that doesn't fall into any of those criteria stays the same into the next time step. Once the game has ran for a specific amount of time steps the game ends and the resulting grid is printed out to the console.

The first thing I did to solve this problem was get the starting input e.g read in the number of rows and columns of the grid and the number of steps to be simulated. This was done in a function I called 'getStartingInput'.

The next thing I did was make the 2-dimensional arrays for the grids and allocate memory to them. For the game I realised it would be best if I had two arrays. One which would hold the 'old' array (e.g. the one before the changes that will be performed within the step) and another that holds the 'new' array (e.g. the one that holds the array with the changes from that step).

The memory allocation for the arrays is done in a function which I called 'allocateArrayMemory'. This essentially allocates enough memory for each cell of each grid to hold one char variable.

The next thing I did was get the starting input for the old grid array (The one that we start with). I did this in a function I called 'getGridStartingPoint'. This function basically has a for loop that iterates over each row of the grid. Inside this for loop I then declared a char array that had a length of the number of columns. I then had another four loop that iterates once for each column and asks the user to enter a char. The char entered is then stored in the char array in its corresponding position. Once that loop has finished I have another one that also iterates for each column and sets the corresponding position in the grid to the char stored in the char array.

The next thing I did was equalise the arrays (e.g. make the new grid array equal to the old grid array). This is done in a function I called 'equaliseArrays'.

The next function I created is called 'start'. This function basically starts the game and contains the games main loop. Inside this function I have a for loop that runs for the number of steps. Inside that for loop I then have another for loop that runs for the number of rows. Inside that for loop I then have another for loop that runs for the number of columns.

Inside here I then create an integer variable called alive. This is set equal to the return value of a function called 'aliveNeighbours' which is passed the indexes of the two above for loops (row and col essentially). This function basically returns the number of alive cells surrounding the cell it is passed. This function is described in more detail below and I am going to carry on describing the start function.

The next thing that happens in the start function is to check if the current cell being checked (uses the indexes of the for loops) is alive or not. If it is then it checks if it has less then two alive neighbours or if it has more then 3 alive neighbours. If either of these are true it sets that cell in the NEW grid array to be dead ('.'). If the cell is not alive then it must be dead and so we check to see if the cell has exactly 3 alive neighbours. If it does, then then that cell in the NEW grid array is set to alive ('X').

The last thing to happen in the start function is to set the old grid array equal to the new grid array. This is done in the same way as the equaliseArrays function but with the arrays switched.

The 'aliveNeighbours' function returns an integer value relating to how many alive cells surround a certain cell. To do this the function must be passed the position on the grid (row and column) of the cell we are checking. The way in which the function keeps track of the number of alive cells found is by using an integer variable called alive that is declared and set to 0 at the start of the function. Next we get into the for loops and if statements.

The first for loop is set so that it will start from i = row – 1 and carry on till i < row +2. This is done to check the 3 rows above, on and below the cell that we are checking. The next thing that is done is an if statement that makes sure we don't get any out of bounds error (e.g. if i = -1 or number of rows). The next for loop runs from j = col -1 and carries on till j < col + 2. This essentially checks each cell on the left, right and current column. Another if statement is then used to make sure we don't get another out of bounds errors (e.g. if j = -1 or number of columns). The last thing that is done is the checking of the current cell we are looking at (e.g. oldGridArray[i][j]). If that cell is alive ('X') then we increase alive by 1. Once the for loops have finished we return alive.

Problem 1081.

This program required me to write a program in C that would simulate vehicles on a multilane highway. This highway is made up of a grid of a number of columns and rows. As input the program takes 3 integers, the number of rows, the number of columns and the number of steps to simulate.

The main simulation runs on a step by step basis. During each step two main things happen. The first thing that takes place is called the movement phase. During this phase all columns on the grid are moved one column to the right. The next phase is the arrival phase. This where the vehicles that are set to arrive on that step are placed on the left most column of the grid on their respective row.

Each vehicle that will enter the simulation has an arrival time and a row index. To read in and store the information on the vehicles that would be added to the grid I first created a struct called vehicle. This struct would hold two integers called ‘arrivalTime’ and ‘rowIndex’.

The next thing I did was create my arrays and allocate memory to them. The first array I created was a 2-dimentional char array which I called ‘mainGrid’. The other array I called ‘vehicles’ as it would be used to hold an array of structs of the type Vehicle.

I allocated memory to these arrays in a separate function which I called ‘allocateArrayMemory’. This function essentially allocates enough memory to both arrays. I allocated enough memory to the mainGrid array so that each cell can hold 1 character. I allocated enough memory to the vehicles array so that it can hold a certain number of vehicles. This number I declared at the top of the program as a constant. I set it to a default of 30. This function also sets each cell of the mainGrid to a dot (‘.’);

The next thing I did was create a function that took the input for the vehicles. I did this in a function called ‘getVehiclePositions’. In this function I first declared three integer variables called arrivalTime, rowIndex and i. I then created a while loop. Inside the condition statement of this while loop I have a scanf statement that asks for two integers separated by a space. As along as the value returned from the scanf is not EOF then the while loop continues. I also do some value checking on the numbers entered. The numbers entered into the scanf are scanned into arrivalTime and Index. I then set vehicles[i]’s arrivalTime and rowIndex. I then increment i.

The next function I created was the ‘start’ function. Inside this function is where my main simulation for loop runs. This for loop runs for the number of steps. The next thing that happens in this for loop is another function is ran called ‘rightShiftGrid’. This function essentially shifts every column in the grid one cell to the right. This is what makes up the ‘movement’ phase.

The next thing that happens inside the main for loop is another for loop that runs from 0 up to the constant for vehicles that I declared above. Essentially this loop runs though every vehicle struct held in the vehicles array. It checks every vehicles arrival time. If the current vehicles arrival time is equal to the current step then it sets the first column of the vehicles row index in the main grid to ‘1’. It then sets that current vehicles arrival time to -1 so that it will not be added again during the next step and instead it will be skipped over.

Once that for loop has finished I then have another for loop that runs once for each row. It checks to see if the first column of the current row is equal to a ‘1’ or not. If it isn’t then it sets that position in the main grid to ‘.’.

Once the main loop finishes the grid is printed to the console using printf In a function I called ‘printGrid’.