All inputs were taken using scanf and all outputs were given using printf.

Game Of Life

To read the first line of input, for example (6 6 20), scanf is used. An array to hold the grid is created with size ROWS and COLUMNS. It is then iterated through and the rest of the input is read into the grid, creating the starting generation. A space before the %c in scanf is needed to ignore the white space. The function simulation is then called to begin the game of life. A new 2D grid, the same size as the initial grid, is created to store the next generation. The number of rows and number of columns of the initial grid are then iterated through to test each and every cell. For each cell, if it is alive, initialise aliveNeighbours to -1, as it will count itself as a neighbour later, else initialise aliveNeighbours to 0. For each cell, check the vertical, horizontal, and diagonal neighbours of distance 1. If the neighbour is alive, increment aliveNeighbours by 1. Otherwise, continue. Once all neighbours have been checked, the rules of game of life are used.

- any live cell with fewer than 2 live neighbours dies
- any live cell with 2 or 3 neighbours lives
- any live cell with more than 3 live neighbours dies
- any dead cell with exactly 3 live neighbours becomes a live cell

The change in the cell is set to the new grid. Once the new grid has been completed, the old grid is set to the new grid to hold the latest generation. The code is iterated again, where the next generation is once again initialised to an empty 2D array. Once the number of steps has been reached, the function printGen is called. In this function, the rows and columns are iterated through and the grid is printed in the same format the input was taken in.
To copy the new generation to the old generation grid, memcpy is used which requires #include <string.h>.

ASCII

Based upon the initial character input by the user, different parts of the program will be called. If the user inputs C, then the compress function is called. In this function, each line is read as a character, if the character is EOF then the program ends. Otherwise, the character is checked to see if it is the same as the previous, if it is, then count is incremented. Otherwise, if the count is greater than 1, the char is output twice, followed by count followed by *. Otherwise, the char is printed just once. If the user inputs E rather than C, then the function expand is called. The same method as the compress function is used, where the program runs while the char input is not EOF. if the previous char is equal to the next char, is followed by 1 or more integers, and is followed by a *, then the char is printed for the integer value of times. If there are consecutive chars followed by at least 1 integer but no *, then the characters and integers are printed as normal. The string that holds the integer values, holdCount, is reset using memset. It must be reset or else the string will hold the count from the previous character. If none of these conditions are met, then the character is simply output by itself.
To reset the count held in holdCount, memset is used and requires #include <string.h>.
To check if the current char is a digit or not, the library ctype.h is used, and to convert from a string integer to an int integer the library stdlib.h is required.

The following websites were used to help with the extended libraries I used for this program.
https://www.geeksforgeeks.org/memcpy-in-cc/
https://www.tutorialspoint.com/c_standard_library/c_function_memset.htm
https://www.programiz.com/c-programming/library-function/ctype.h/isdigit
https://www.tutorialspoint.com/c_standard_library/c_function_atoi.htm