To solve the problem, I split it into separate problems.

**Main**

The first thing I did to make it work within terminal is to check the amount arguments, as there should only be one argument given. An output will be displayed of the amount of arguments given and the usage of the programme if there are more than 2.

If the argument of one text file is given, with an error presented if the file could not open, the programme starts and displays the developer signature. The main part of the programme goes through every character in the text file and stores them into a 2D array of fixed size, each symbol being counted in an additional easy IF statement with the exception of newline characters that are written as \n. The IF statement checks which symbol it is. If the symbol is neither for sea nor what is currently defined as land, the land character counter will increment and store but with no knowledge of what the symbol is.

The next part of the programme outputs each of the counted variables, each variable having already been counted apart from landmasses, which calls a function called islandCount.

There is an additional commented function that prints the contents of the 2D array, which of course displays the same as the text file.

**islandCount**

the second thing I did was to make the function that would be counting the landmasses present within the 2D array. I decided that instead of attempting to count landmasses by either sinking them (aka turning all land symbols to water) or by counting the land mass size and hoping that no two landmasses had the same area, I would instead make a second 2D array to mimic the first, that would be a binary version of the map. This 2D array starts out empty, every space being false, or 0, then as it discovers land makes that space true, or 1.

The count for landmasses, that will be returned as land masses, starts at zero, and will increment each time a character of land that has not been discovered already appears. The function checks each character on whether it’s not a sea character and whether it’s been discovered, skipping if either is true. If neither is true it calls the last function, called areaCheck, that discovers an entire landmass on its own, returns, and increments the count of landmass.

Once this loop ends the number value of landmass is returned and displayed in the main function, ending the program.

**areaCheck**

The last step I made was to create the code to discover a whole landmass. Originally, I had planned to put this in islandCounts function, however since an entire landmass could have any size this function needed to be recursive, so was easier made in a separate function rather than in the counting function.

First two arrays are defined, rowChecker and colChecker, which are basically combined to made rough co-ordinates of the surrounding characters, then the current character, the one that started the function call, is set as discovered.

A FOR loop is then set up that checks every co-ordinate within the 8 surrounding characters, the if statement inside checking if each is within the correct map range and if the character is land. If a surrounding character is land then the function calls itself, sets this new character as discovered and searches around that character, which can get deeper into recursion the more surrounding land there is, searching around each new land character until the entire landmass is discovered, upon which it exits its loop, ending the function.

In order to complete this challenge I decided to do some research on the challenge itself, finding a simpler version of the challenge as a basis, along with some research to refresh my memory on C++ as well as how to take in arguments from the command line.

I enjoyed solving the problem and may have spent more than 3 hours on it, but a large part of that was taken up with downloading/installing and setting up visual studio.