Problem 1:

In this solution the main function dynamically builds a singly linked list. The code prompts the user to enter integers continuously and uses a loop that terminates when an EOF is encountered. Each value is stored in a freshly allocated node, which is attached to the end of the list using a head–tail pointer strategy. Once all the data has been entered, the program traverses the list and prints the integers in the order they were added.

A separate function, called revList, is then invoked to reverse the list. This function iterates through the current list, carefully saving the next node before altering the current node's pointer to link it to a new, initially empty, list. The process repeats until the complete list is reversed, and the head pointer is updated accordingly. Finally, the program prints the reversed list and frees the allocated memory.

Problem 2:

This solution also starts in the main function by reading integers from the user and dynamically creating a singly linked list. The list is built by appending new nodes, one for each value entered, until the end-of-file signal is received.

The core sorting work is done in a function that implements a bubble sort algorithm. Here, instead of swapping the contents of nodes, the sorting algorithm reorders the nodes themselves. A pointer-to-pointer approach is used to walk through the list, comparing adjacent nodes. When a node with a higher value is found before one with a lower value, their pointers are swapped. The outer loop of the function gradually reduces the unsorted portion of the list, while the inner loop handles the comparison and potential swap of nodes. After the sort is complete, the main function iterates over the now-ordered list and prints each value on its own line before cleaning up the allocated memory.