This program takes in a file of data, either strings or integers, and prints the data out, sorted. The program sorts the data by using either insertion sort or quicksort, and this is selected by the user when the program is run.

The program first checks to see if the input is entered correctly: valid flag and test file. The given file is then read by the number of bytes the file contains and a test is conducted to determine if the data consists of integers or characters using the isdigit() and isalpha() functions. Both functions take in characters and the output is a non-zero number if true and zero if false. The program then creates a linked list using the data that was read. It loops through each character in the data and stores it in a temp character array. The program keeps adding characters to the temp array until it reads a comma. Once this is done, the string of characters is added to the linked list. If the data consists of integers, then the string is converted into an integer using atoi() and then added to the linked list. If the data consists of characters, the string is directly added to the linked list. Whitespaces such as spaces, tabs, and new lines are ignored and are added to the linked list as empty tokens unless they are the last token. If the last token consists of whitespace, it is simply ignored and not added to the linked list. Once the linked list is created, it can be passed into one of two sorting functions depending on what the user specified in the input. If the user inputed '-i', the list will be sorted using insertion sort. If the user inputed '-q', the list will be sorted using quicksort.

The program consists of two comparator functions: one for integers and one for character arrays. For both functions, two nodes are passed through and an integer value is returned. For the intComparator(), the integer values of both nodes are compared with each other and returns -1 if the first value is less than the second value, 1 if the first value is greater than the second value, and 0 if both values are equal. The strComparator() does the same except each character in both nodes are compared with each other based on their ascii values. The function loops through both strings until it finds a differing character. Then, the characters are compared and the function returns -1 if the differing character in the first string is less than that of the second string, 1 if the differing character in the first string is greater than that of the second string, and 0 if the function loops through both strings and finds no differing characters (meaning that they are equal).

The program's insertionSort() function takes in two parameters, a void pointer (in which we passed the head of our linked list) and a function pointer (in which we passed the appropriate comparator function), and returns an integer value. Insertion sorts goes through each of the nodes in the linked list and compares it with the nodes before it to determine if a node is out of place. If a node is out of place, it is inserted into the correct place. Starting at the second node, the function looks at a node and compares it, using the appropriate comparator function, with the node before it. If the value of the initial node is greater than or equal to the value of the previous node, no insertion occurs, and the pointer moves on to check the next node. If the value of the initial node is less than the value of the previous node, the pointer keeps moving back to the previous nodes until it finds a node that has a value less than that of the initial node. The initial node is then inserted after the node with the value that is just less than its own value. The point of this sort is that each node is inserted between two other nodes and not swapped with other nodes.

The program's quickSort() function takes in two parameters, a void pointer (in which we passed the head of our linked list) and a function pointer (in which we passed the appropriate comparator function), and returns an integer value. Quicksort sorts the data by making the first value a pivot and moving the values less than the pivot to the left of it and the values greater than the pivot to the right of it. This is done by swapping nodes around. A swap() function in the program takes the head of the linked list and two nodes and returns the head of the linked list with the two nodes swapped. Once the values less than and greater than the pivot are moved to their respective sides, the pivot is in the correct place and the data to the left and right of it divides and the same process repeats. Quicksort is done recursively to accomplish this. At the end of the sort, all of the values will be in their correct positions.

After the linked list is sorted, each of the sorted values are printed out. After that, memory is freed, the file is closed, and the program ends.

Errors and warnings such as having the wrong arguments, inputting an empty file, malloc() returning NULL, etc. will cause error / warning statements to be printed out.