Question 1

It is required to implement a small module which provides functions and data types for an ascendingly sorted linked list of strings. (All the functions are expected to return '0' in case of success and '-1' in case of error)

1. Create a structure data type "tstr_node" to hold the linked list node related data

2. Provide an implementation for the list insert function "sorted_list_insert" that adds a node to the list.

```
int sorted_list_insert(tstr_node **head , tstr_node *pstr_node)
{
        …
}
```

3. Provide an implementation for the list remove function "sorted_list_remove" that removes a node from the list.

```
int sorted_list_remove(tstr_node **ppstr_head , tstr_node *pstr_node)
{
        …
}
```

Kindly check the attached "C_implementation" folder, used IDE: Eclipse

Question 2

Write down all the different test scenarios required to fully test the implementation in "Question 1"

Test scenarios:

1. Insert function test scenarios:
   - Failure:
     a. Uninitialized node (Test string), which should be added, return -1.
   - Success, return 0:
     b. Use a node (test string) with a value smaller than the head node, our test string should be the head in this case.
     c. Use a node which holds an intermediate value, should be added in an intermediate position in the list is this case.
     d. Use a node which holds a value higher than all the strings in the list, should be placed in the last node in this case.
     e. Add the same node or pointer twice
2. Remove function test scenarios:
   - Failure, return -1:
     a. Use a test node, which is uninitialized.
     b. Use a test node, which is not in the list.
   - Success, return 0:
     a. Use a test node, which is the head of the list, should be freed and the head pointes to the next node.
     b. Use a node which has an intermediate value.
     c. Use a node which exists in the end of the list.

Question 3

Describe how you would implement the test scenarios defined in “Question 2”

By defining a test case functions, check their return values, and observe the change in the displayed list.

Implementation → main.c file

Question 4

Write down all the different test scenarios required to fully test the an implementation of the C standard library function:

```
void * memcpy ( void * destination, const void * source, size_t num );
```

Copies the values of *num* bytes from the location pointed to by *source* directly to the memory block pointed to by *destination*.

Function parameters:

- dest – This is pointer to the destination array where the content is to be copied, type-casted to a pointer of type void*.
- src – This is pointer to the source of data to be copied, type-casted to a pointer of type void*.
- n – This is the number of bytes to be copied.

Test scenarios:

- Failure cases, which should be handled:

1) Test with null pointer to src.
2) Test with null pointer to destination.
3) Test with a size larger than the destination.
4) Test if both src and destination are overlapping (i.e. function argument is misused).
5) Test for invalid data types.

- Success cases:

6) After changing the value of the destination, compare it’s value with the expected value by using a function like strcmp if the data type is char in this case.
7) Compare the function return value (i.e. pointer) to an expected value and observe the result.

Question 5

Answer “Question 1” again, but now using an object-oriented language (preferred Python or Java). Consider writing an efficient implementation for the language you have chosen. Briefly explain why this would be an efficient implementation.

1) Python performs memory management compared to C.
2) Python is more readable using PEP8 style, and the OOP concept simplifies the problem.
3) Python does exception handling, for example if the input data type was not correct.
4) The implementation itself counts for different corner cases.
5) The concept of passing by value or by reference is simpler and less error prone.
6) The list will always be sorted ascendingly, as for the “redundant” case, I assumed the list would have a redundant nodes (i.e. have the same string), as they might be used in different places, so they should not be removed, or misused.
7) Code is more compact (e.g. swapping done in one line), the unittest is more compact and efficient.