Task 1

In this task, I implemented a generic linked list class, which included the creation of a Node struct and a LinkedList struct. Several operations were developed to manipulate the linked list effectively. The tests were expanded to include a second data type, demonstrating the versatility of the linked list. The outcomes of these tests, reflecting both the original and the secondary data types, were documented through images included in the final report.

Original:

```
After initialization
value: 18
value: 16
value: 14
value: 12
value: 10
value: 8
value: 6
value: 4
value: 2
value: 0

After squaring
value: 324
value: 256
value: 196
value: 144
value: 100
value: 64
value: 36
value: 16
value: 4
value: 0

removed: 324

Found: 256

removed: 256

After removals
value: 196
value: 144
value: 100
value: 64
value: 36
value: 16
value: 4
value: 0
```

```
After append
value: 196
value: 144
value: 100
value: 64
value: 36
value: 16
value: 4
value: 0
value: 11

After clear

After appending
value: 0
value: 1
value: 2
value: 3
value: 4

popped: 0
popped: 1

After popping
value: 2
value: 3
value: 4

List size: 3
```

Second:

```
After initialization
value: O
value: N
value: M
value: L
value: K
value: J
value: I
value: H
value: G
value: F

After moving char by 1 (adding 1)
value: P
value: O
value: N
value: M
value: L
value: K
value: J
value: I
value: H
value: G

removed: O

removed: P

After removals
value: N
value: M
value: L
value: K
```

```
value: J
value: I
value: H
value: G

After append
value: N
value: M
value: L
value: K
value: J
value: I
value: H
value: G
value: O

After clear

After appending
value: x
value: y
value: z
value: {
value: |

popped: x
popped: y

After popping
value: z
value: {
value: |

List size: 3

data from deleted node: {

After deleting index 1
value: z
value: |
```

The freefunc parameter in the ll_clear function properly manages memory by allowing the user to specify how the data linked to each node should be freed. Without this function, the allocated memory for node data cannot be properly released, leading to memory leaks.

For example, a llinked list where each node includes a pointer to a custom object created dynamically. If the freefunc is either not provided or not used by ll_clear, the memory allocated for these objects remains in use, despite the nodes being deleted from the list. This results in memory leaks. Therefore, a correctly implemented freefunc ensures that each node's associated memory is released, preventing any memory from being wasted.