

NAME: Jinyi XIA
STUDENT ID: 2021212057
CLASS NUMBER: 2021211802
CONTAINER NUMBER: a4d4f6a5498a4e912a4d1a76a7fa36cf23af64b416688d222a8968a38b29a711

REPORT ON LAB 1

1 Environment

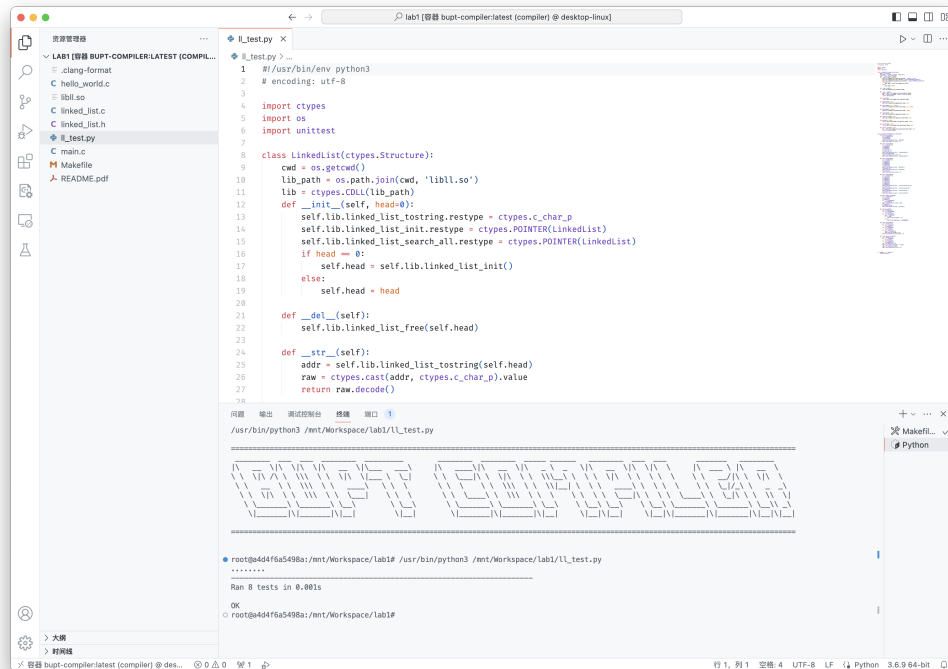
All operations are conducted in the given Docker image.

2 Procedure

1. Implement the linked list interfaces given.
2. Build the dynamic link library via `make libll` command.
3. Test code with the Python script `ll_test.py`.

3 Result

It turns out that the linked list implementation is correct as is shown in fig. 1.



```
ll_test.py > ...
1 #!/usr/bin/env python3
2 # encoding: utf-8
3
4 import ctypes
5 import os
6 import unittest
7
8 class LinkedList(ctypes.Structure):
9     cwd = os.getcwd()
10     lib_path = os.path.join(cwd, 'libll.so')
11     lib = ctypes.CDLL(lib_path)
12     def __init__(self, head=0):
13         self.lib.linked_list_tostring.restype = ctypes.c_char_p
14         self.lib.linked_list_init.restype = ctypes.POINTER(LinkedList)
15         self.lib.linked_list_search_all.restype = ctypes.POINTER(LinkedList)
16         if head == 0:
17             self.head = self.lib.linked_list_init()
18         else:
19             self.head = head
20
21     def __del__(self):
22         self.lib.linked_list_free(self.head)
23
24     def __str__(self):
25         addr = self.lib.linked_list_tostring(self.head)
26         raw = ctypes.cast(addr, ctypes.c_char_p).value
27         return raw.decode()
28
29
30 if __name__ == '__main__':
31     unittest.main()
```

```
root@4d4f6a5498a:/mnt/Workspace/lab1# /usr/bin/python3 /mnt/Workspace/lab1/ll_test.py
.....
Ran 8 tests in 0.001s
OK
root@4d4f6a5498a:/mnt/Workspace/lab1#
```

Figure 1: Test result

4 Code

```
1 void linked_list_insert(node *head, int val, int index) {
2     if (index < 0 || index > head->count) {
3         return;
4     }
5     node *new_node = (node *)malloc(sizeof(node)), *pos = head, *next;
6     new_node->value = val;
7     for (int i = 0; i < index; i++) {
8         pos = pos->next;
9     }
10    next = pos->next;
11    pos->next = new_node;
12    new_node->next = next;
13    head->count++;
14 }
```

Listing 1: Insert value at the position of index

```
1 void linked_list_delete(node *head, int index) {
2     node *pos = head, *next;
3     if (index < 0 || index >= head->count) {
4         return;
5     }
6     for (int i = 0; i < index; i++) {
7         pos = pos->next;
8     }
9     next = pos->next->next;
10    free(pos->next);
11    pos->next = next;
12    head->count--;
13 }
```

Listing 2: Delete node at the position of index

```
1 void linked_list_remove(node *head, int val) {
2     node *pos = head, *next;
3     while (pos->next) {
4         if (pos->next->value == val) {
5             next = pos->next->next;
6             free(pos->next);
7             pos->next = next;
8             head->count--;
9             return;
10        }
11        pos = pos->next;
12    }
13 }
```

Listing 3: Remove the first node with given value

```

1 void linked_list_remove_all(node *head, int val) {
2     node *pos = head, *next;
3     while (pos->next) {
4         if (pos->next->value == val) {
5             next = pos->next->next;
6             free(pos->next);
7             pos->next = next;
8             head->count--;
9         } else {
10            pos = pos->next;
11        }
12    }
13 }

```

Listing 4: Remove all nodes with given value

```

1 int linked_list_get(node *head, int index) {
2     if (index < 0 || index >= head->count) {
3         return -0x80000000;
4     }
5     for (int i = 0; i < index; i++) {
6         head = head->next;
7     }
8     return head->next->value;
9 }

```

Listing 5: Get value at the position of index

```

1 int linked_list_search(node *head, int val) {
2     int index = 0;
3     while (head->next) {
4         if (head->next->value == val) {
5             return index;
6         }
7         head = head->next;
8         index++;
9     }
10    return -1;
11 }

```

Listing 6: Get the index of the first node with given value

```

1 node *linked_list_search_all(node *head, int val) {
2     node *new_head = linked_list_init();
3     node *cur = new_head;
4     int index = 0;
5     while (head->next) {
6         if (head->next->value == val) {
7             cur->next = (node *)malloc(sizeof(node));
8             cur = cur->next;
9             cur->value = index;
10            cur->next = NULL;

```

```
11     }
12     head = head->next;
13     index++;
14 }
15 return new_head;
16 }
```

Listing 7: Get the list of all nodes with given value