NAME:	Jinyi Xia
STUDENT ID:	2021212057
CLASS NUMBER:	2021211802
CONTAINER NUMBER:	df492137f3eca5844e1f8c7dce8b55741165f498417aafe88d30c2be6816f3d1

REPORT ON LAB 5

1 Read the code

In `symtab_ll.c`, `struct entry` is used to describe entries in the symbol table. It has two fields. `key` stores symbol's name, and `value` stores symbol's value.

```
typedef struct entry {
    char key[KEY_LEN+1];
    VAL_T value;
} entry;
```

`struct symtab` implements the symbol table with a linked list.

```
struct symtab {
    entry entry;
    struct symtab *next;
};
```

`symtab_init`, `symtab_insert`, `symtab_lookup`, and `symtab_remove` implement these operations on symbol tables with corresponding implementations on linked lists.

```
symtab *symtab_init(){
    symtab *self = malloc(sizeof(symtab));
    memset(self, '\0', sizeof(symtab));
    self->next = NULL;
    return self;
}
```

```
int symtab_insert(symtab *self, char *key, VAL_T value){
    symtab *ptr = self;
    while(ptr->next != NULL){
        if(strcmp(ptr->entry.key, key) == 0)
            return 0;
        ptr = ptr->next;
    }
    symtab *node = malloc(sizeof(symtab));
    memset(node, '\0', sizeof(symtab));
    entry_init(&node->entry, key, value);
    node->next = NULL;
    ptr->next = node;
    return 1;
}
```

```c
VAL_T symtab_lookup(symtab *self, char *key){
    symtab *ptr = self;
    while(ptr != NULL){
        if(strcmp(ptr->entry.key, key) == 0)
            return ptr->entry.value;
        ptr = ptr->next;
    }
    return -1;
}
```

```c
int symtab_remove(symtab *self, char *key){
    symtab *ptr = self, *tmp;
    while(ptr->next != NULL) {
        if(strcmp(ptr->next->entry.key, key) == 0){
            tmp = ptr->next;
            ptr->next = ptr->next->next;
            free(tmp);
            return 1;
        }
        ptr = ptr->next;
    }
    return 0;
}
```

2 Make the target

The result is shown in fig. 1.

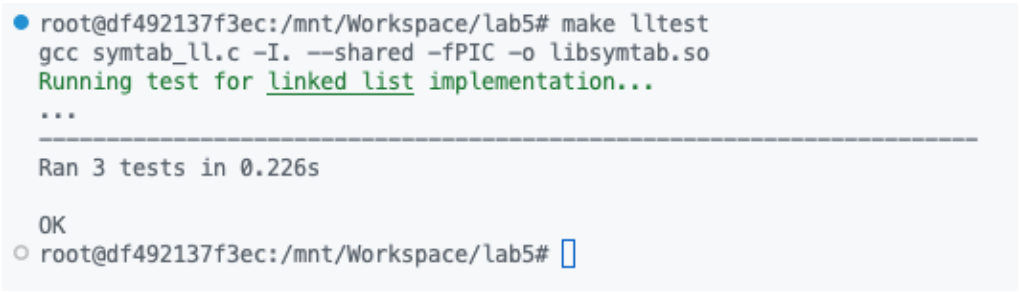

Figure 1: Test result