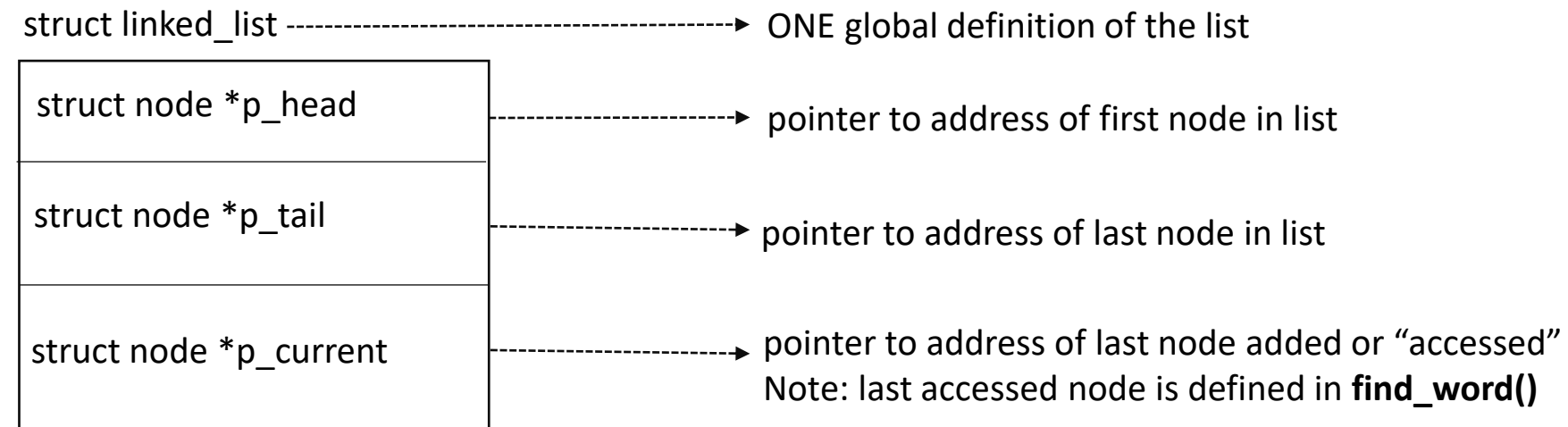
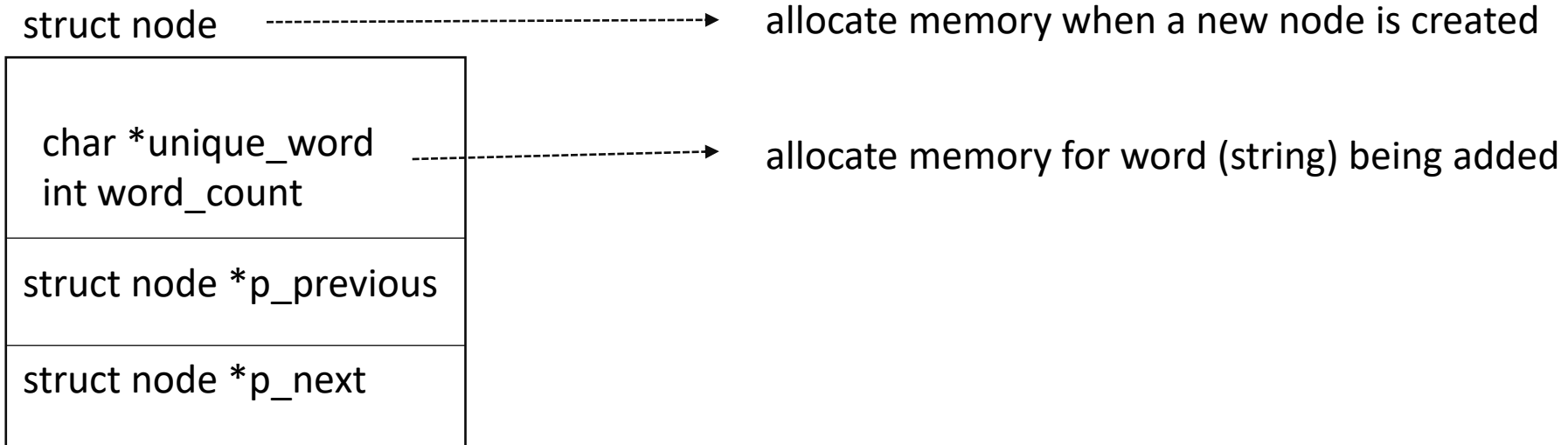


Structures used to maintain doubly-linked list



The following (not complete) test program is used for the linked list illustration that follows. Note we have added a utility function `print_list()` to output the word contents of the linked list to help with testing.

```
print_list(struct linked_list *p_list) {
    struct node *current = p_list->p_head;
    while( current != NULL){
        printf("%s  ", current->unique_word);
        current = current->p_next;
    }
    printf("\n");
    return;
}

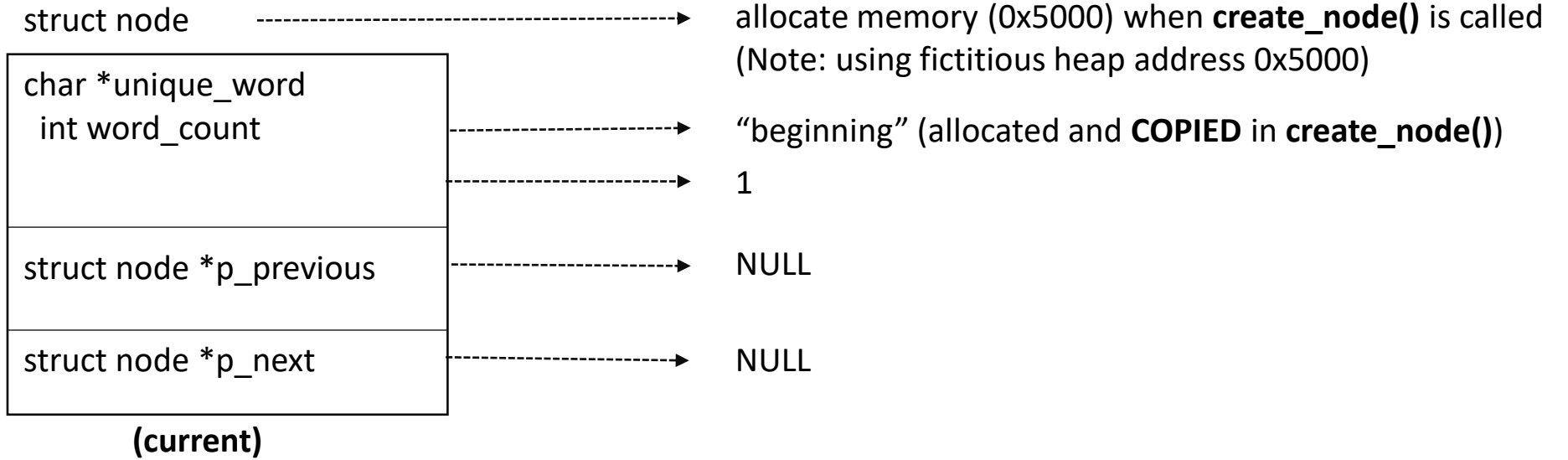
main() {
    struct linked_list myList ;
    memset( &myList, 0, sizeof( myList ) ) ;
    add_node_at_head( &myList, "beginning" ) ;
    print_list(&myList);
    add_node_after_current( &myList, "middle");
    print_list(&myList);
    add_node_after_current( &myList, "whatever");
    print_list(&myList);
    find_word(&myList, "sponge");
    add_node_after_current( &myList, "sponge");
    print_list(&myList);
}
```

Sample run

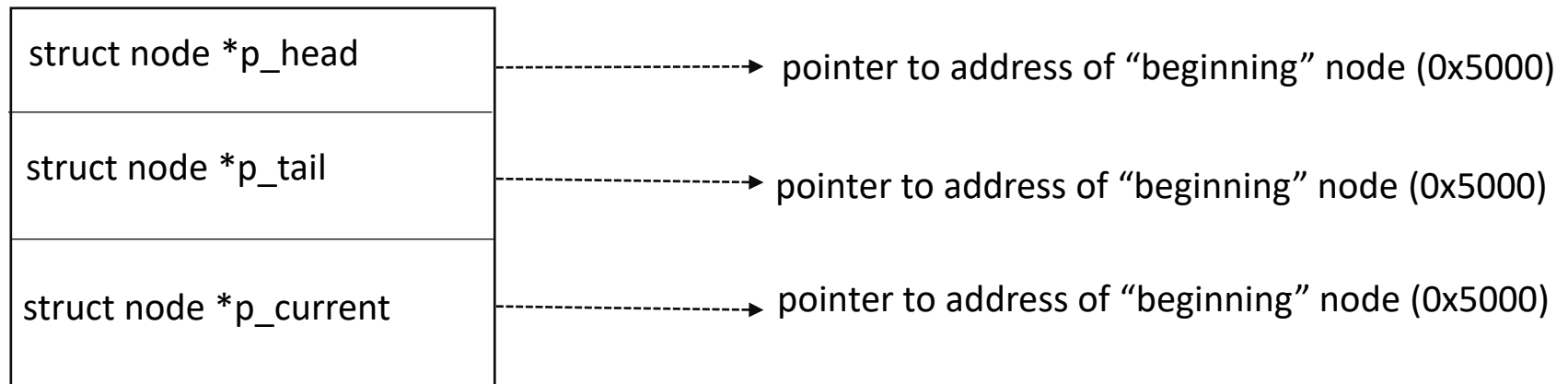
```
bash-4.4$ ./test
beginning
beginning  middle
beginning  middle  whatever
beginning  middle  sponge  whatever
bash-4.4$
```

```
struct linked_list myList;  
add_node_at_head( &myList, "beginning")
```

```
// all fields initialized to NULL via memset()  
// add_node_at_head() calls create_node()
```

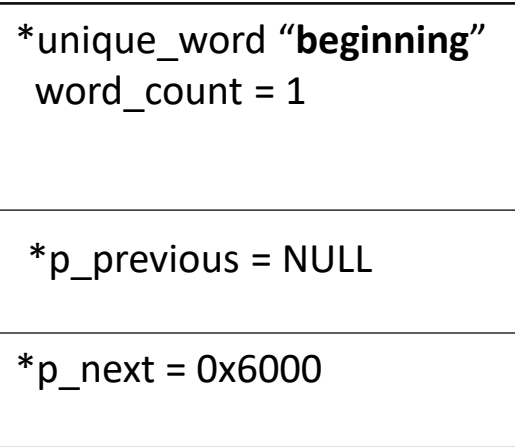


```
struct linked_list myList
```

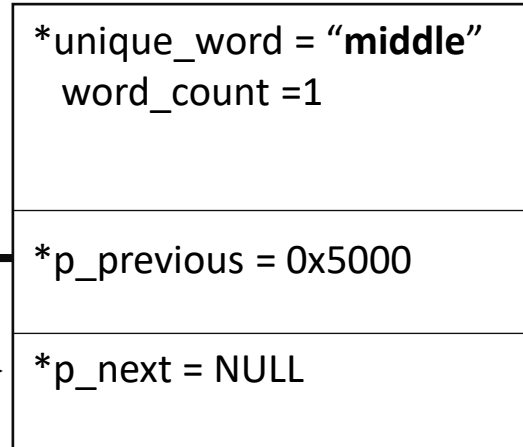


```
add_node_after_current( &myList, "middle") // add_node_after_current() calls create_node()
```

struct node (0x5000)

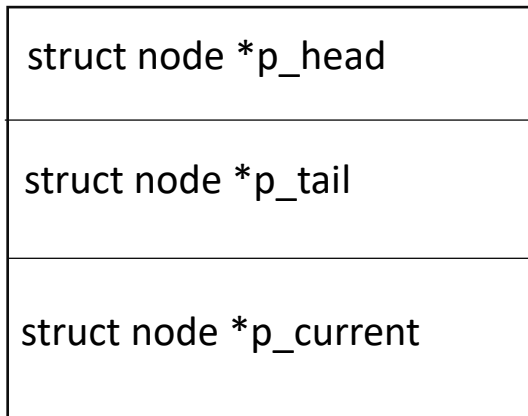


struct node (0x6000)



(current)

struct linked_list myList



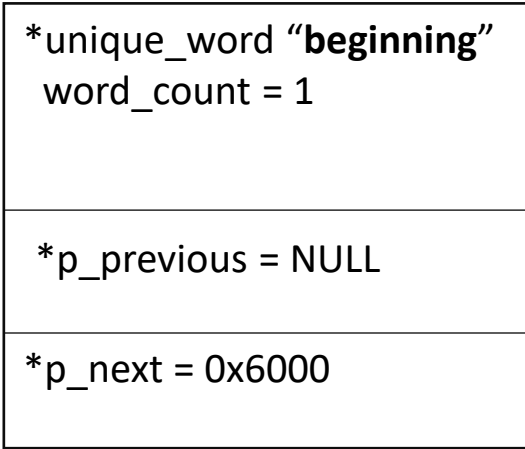
-----> pointer to address of "beginning" node (0x5000)

-----> pointer to address of "middle" node (0x6000)

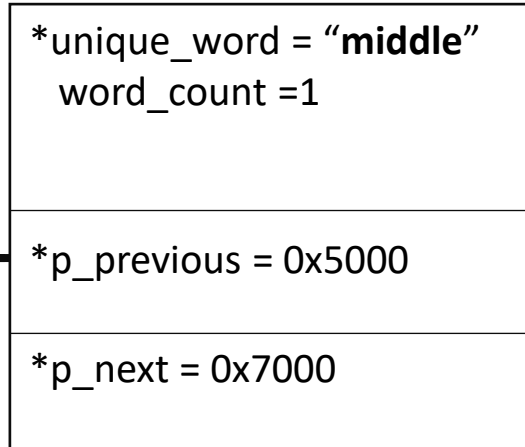
-----> pointer to address of "middle" node (0x6000)

```
add_node_after_current( &myList, "whatever") // add_node_after_current() calls create_node()
```

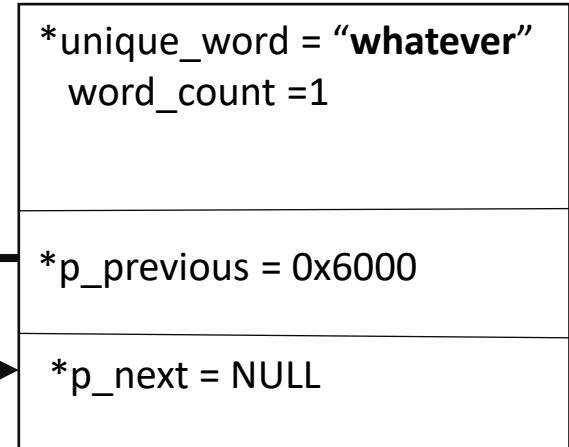
struct node (0x5000)



struct node (0x6000)

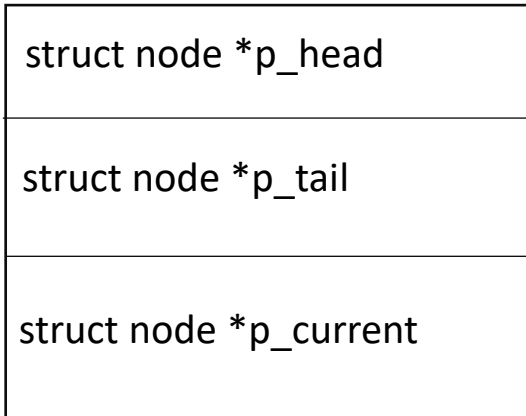


struct node (0x7000)



(current)

struct linked_list myList



-----> pointer to address of "beginning" node (0x5000)

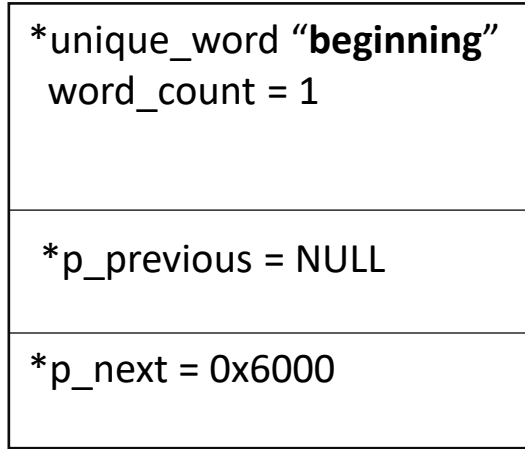
-----> pointer to address of "whatever" node (0x7000)

-----> pointer to address of "whatever" node (0x7000)

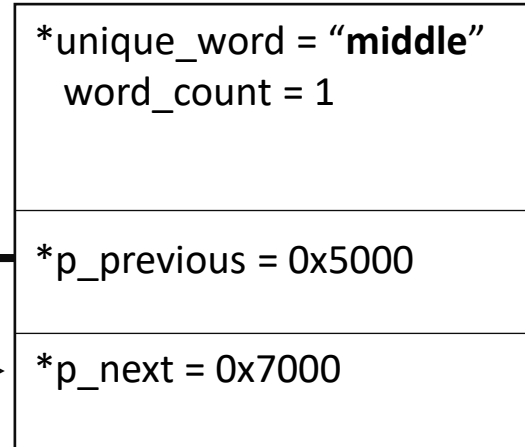
```
find_word( &myList, "sponge")
```

```
// "sponge" is not in the list, returns -1 (not success)  
// current will be set to "middle" node  
// list remains unchanged except for current pointer
```

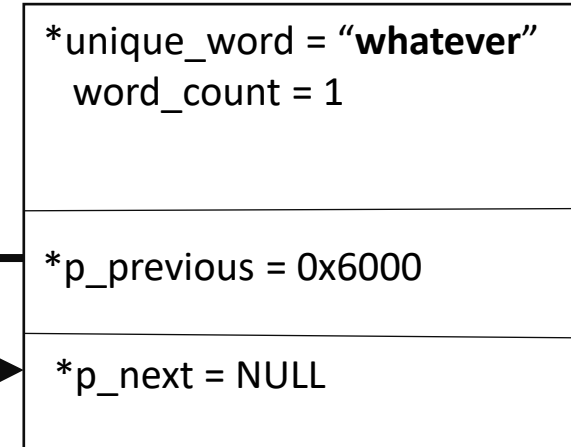
struct node (0x5000)



struct node (0x6000)



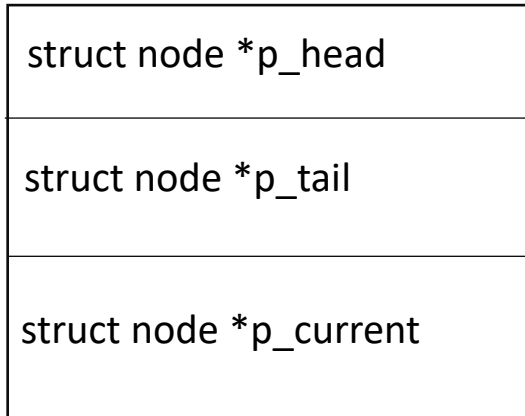
struct node (0x7000)



(current)

(alphabetically sponge
should have been here)

struct linked_list myList



pointer to address of "beginning" node (0x5000)

pointer to address of "whatever" node (0x7000)

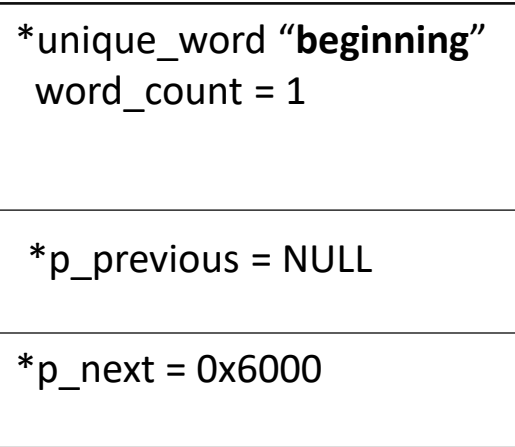
pointer to address of "middle" node (0x6000)

```
add_node_after_current( &myList, "sponge")
```

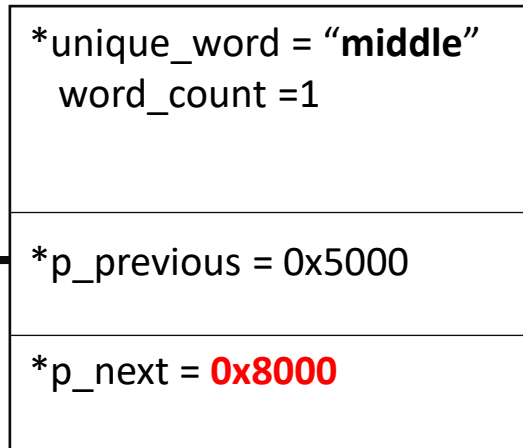
```
// relies on current being set correctly!
```

```
// note re-wiring of pointers
```

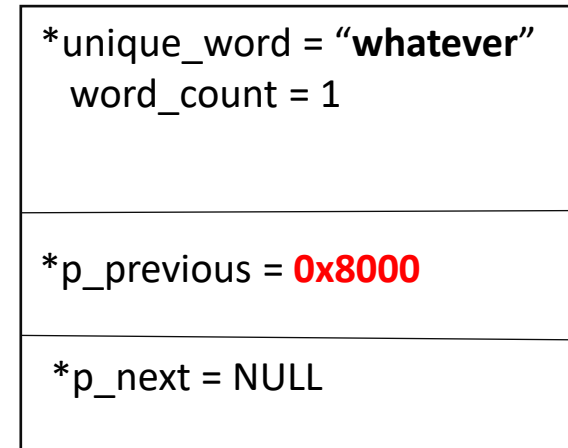
struct node (0x5000)



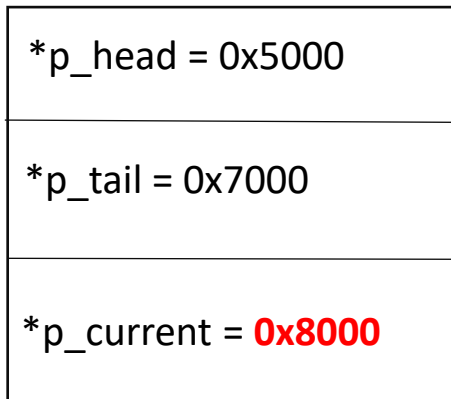
struct node (0x6000)



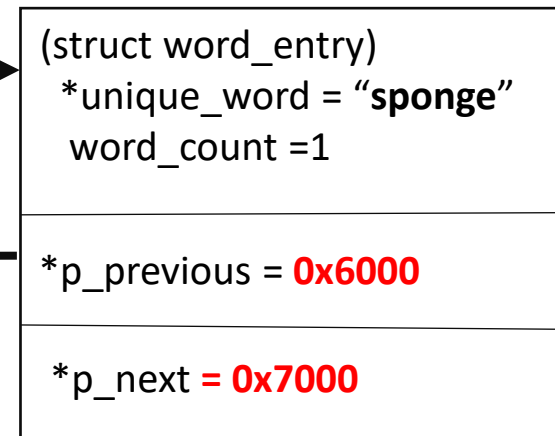
struct node (0x7000)



struct linked_list myList



struct node (0x8000)



(current)

