

```
1 //
2 // Created by hfwei on 2023/12/20.
3 //
4
5 #ifndef INC_2023_CPL_CODING_0_13_LINKED_LIST_LL_LL_H_
6 #define INC_2023_CPL_CODING_0_13_LINKED_LIST_LL_LL_H_
7
8 // adding code below
9
10 #include <stdbool.h>
11 typedef struct node {
12     int val;
13     struct node *next;
14 } Node;
15
16 typedef struct ll {
17     Node *head;
18     Node *tail;
19     int size;
20 } LinkedList;
21
22 void Init(LinkedList *list);
23
24 bool IsEmpty(const LinkedList *list);
25 bool IsSingleton(const LinkedList *list);
26
27 /**
28  * @brief Get the value of the head node.
29  * @param list
30  * @return -1 if list is empty, otherwise the value of the head node.
31  */
32 int GetHeadVal(const LinkedList *list);
33 Node *Search(const LinkedList *list, int val);
34
35 void Print(const LinkedList *list);
36
37 void Append(LinkedList *list, int val);
38 void Insert(LinkedList *list, Node *prev, int val);
39
40 /**
41  * @brief Delete the node after prev from list.
42  * @param list
43  * @param prev
44  */
45 void Delete(LinkedList *list, Node *prev);
46
47 void Free(LinkedList *list);
48
49 #endif //INC_2023_CPL_CODING_0_13_LINKED_LIST_LL_LL_H_
```

```
1 //
2 // Created by hfwei on 2023/12/20.
3 //
4
5 #include <stdlib.h>
6 #include <stdio.h>
7 #include "ll.h"
8
9 void Init(LinkedList *list) {
10     list->head = NULL;
11     list->tail = NULL;
12 }
13
14 bool IsEmpty(const LinkedList *list) {
15     return list->head == NULL;
16 }
17
18 bool IsSingleton(const LinkedList *list) {
19     // return list->head == list->tail;
20     return !IsEmpty(list) && (list->head == list->tail);
21 }
22
23 int GetHeadVal(const LinkedList *list) {
24     if (IsEmpty(list)) {
25         return -1;
26     }
27
28     return list->head->val;
29 }
30
31 void Append(LinkedList *list, int val) {
32     Node *node = malloc(sizeof *node);
33     if (node == NULL) {
34         printf("malloc failed\n");
35         return;
36     }
37     node->val = val;
38
39     if (IsEmpty(list)) {
40         list->head = node;
41     } else {
42         list->tail->next = node;
43     }
44
45     list->tail = node;
46     list->tail->next = list->head;
47 }
48
49 void Delete(LinkedList *list, Node *prev) {
50     if (IsEmpty(list)) {
51         return;
52     }
53 }
```

```
54  if (IsSingleton(list)) {
55      free(list->head);
56      Init(list);
57      return;
58  }
59
60  Node *cur = prev->next;
61  Node *next = cur->next;
62
63  prev->next = next;
64
65  // cur != list->head || cur != list->tail
66  if (cur == list->head) {
67      list->head = next;
68  }
69
70  if (cur == list->tail) {
71      list->tail = prev;
72  }
73
74  free(cur);
75 }
76
77 void Print(const LinkedList *list) {
78     if (IsEmpty(list)) {
79         printf("empty list\n");
80         return;
81     }
82     Node *node = list->head;
83
84     do {
85         printf("%d ", node->val);
86         node = node->next;
87     } while (node != list->head);
88
89     // wrong version
90     // while (node != list->head) {
91     //     printf("%d ", node->val);
92     //     node = node->next;
93     // }
94 }
95
96 void Free(LinkedList *list) {
97     while (!IsEmpty(list)) {
98         Delete(list, list->head);
99     }
100 }
```

```
1  //
2  // Created by hfwei on 2023/12/20.
3  //
4
5  #include <stdio.h>
6  #include <assert.h>
7  #include "ll/ll.h"
8
9  #define NUM 10
10
11 void SitAroundCircle(LinkedList *list, int num);
12 void KillUntilOne(LinkedList *list);
13 int GetSurvivor(const LinkedList *list);
14
15 int main(void) {
16     printf("I hate the Josephus game!\n");
17
18     LinkedList list;
19     Init(&list);
20
21     SitAroundCircle(&list, NUM);
22     // Print(&list);
23
24     KillUntilOne(&list);
25     int survivor = GetSurvivor(&list);
26     printf("%d : %d\n", NUM, survivor);
27
28     Free(&list);
29
30     return 0;
31 }
32
33 void SitAroundCircle(LinkedList *list, int num) {
34     for (int i = 1; i <= num; i++) {
35         Append(list, i);
36     }
37 }
38
39 void KillUntilOne(LinkedList *list) {
40     Node *node = list->head;
41
42     while (!IsSingleton(list)) {
43         // use node to delete node->next
44         Delete(list, node);
45         node = node->next;
46     }
47 }
48
49 int GetSurvivor(const LinkedList *list) {
50     assert(IsSingleton(list));
51
52     return GetHeadVal(list);
53 }
```