

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

```
54
55     if (list->tail == NULL) {
56         list->head = node;
57     } else {
58         list->tail->next = node;
59     }
60
61     list->tail = node;
62     node->next = list->head;
63 }
64
65 void Insert(LinkedList *list, Node *prev, int val) {
66     Node *node = malloc(sizeof *node);
67     if (node == NULL) {
68         printf("Error: malloc failed in Insert()\n");
69         return;
70     }
71     node->val = val;
72
73     node->next = prev->next;
74     prev->next = node;
75
76     if (prev == list->tail) {
77         list->tail = node;
78     }
79 }
80
81 void Delete(LinkedList *list, Node *prev) {
82     Node *cur = prev->next;
83     Node *next = cur->next;
84     prev->next = next;
85
86     if (cur == prev) {
87         list->head = NULL;
88         list->tail = NULL;
89         free(cur);
90         return;
91     }
92
93     if (cur == list->head) {
94         list->head = next;
95     }
96
97     if (cur == list->tail) {
98         list->tail = prev;
99     }
100
101     free(cur);
102 }
103
104 void Print(const LinkedList *list) {
105     if (IsEmpty(list)) {
106         return;
```

```
107     }
108
109     Node *iter = list->head;
110     do {
111         printf("%d\t", iter->val);
112         iter = iter->next;
113     } while (iter != list->head);
114 }
115
116 void Free(LinkedList *list) {
117     while (!IsEmpty(list)) {
118         Delete(list, list->head);
119     }
120 }
```

```
1 /**
2  * file: ll.h
3  *
4  * Created by hengxin on 12/19/21.
5  */
6
7 #include <stdbool.h>
8
9 #ifndef C_PL_CODING_0_11_LINKEDLIST_LL_LL_H_
10 #define C_PL_CODING_0_11_LINKEDLIST_LL_LL_H_
11
12 typedef struct node {
13     int val;
14     struct node *next;
15 } Node;
16
17 typedef struct ll {
18     Node *head;
19     Node *tail;
20 } LinkedList;
21
22 void Init(LinkedList *list);
23
24 bool IsEmpty(const LinkedList *list);
25 bool IsSingleton(const LinkedList *list);
26
27 int Head(const LinkedList *list);
28 Node *Search(const LinkedList *list, int val);
29
30 void Append(LinkedList *list, int val);
31 /**
32  * The caller is responsible for ensuring that
33  * *prev is a pointer to a node in the list.
34  *
35  * @param list
36  * @param prev
37  * @param val
38  */
39 void Insert(LinkedList *list, Node *prev, int val);
40 void Delete(LinkedList *list, Node *prev);
41
42 void Free(LinkedList *list);
43 void Print(const LinkedList *list);
44
45 #endif //C_PL_CODING_0_11_LINKEDLIST_LL_LL_H_
```

```
1  /**
2   * file: llist.c
3   *
4   * Created by hengxin on 12/19/21.
5   */
6
7  #include <stdlib.h>
8  #include <stdio.h>
9  #include "l1st.h"
10
11 Node *head = NULL;
12 Node *tail = NULL;
13
14 void Init() {
15     head = tail = NULL;
16 }
17
18 int IsEmpty() {
19     return (head == NULL);
20 }
21
22 int IsSingleton() {
23     return (head == tail) && (head != NULL);
24 }
25
26 int Head() {
27     return head->val;
28 }
29
30 void Push(int val) {
31     Node *node = malloc(sizeof(*node));
32     if (node == NULL) {
33         printf("Error: malloc failed in Push()\n");
34         exit(EXIT_FAILURE);
35     }
36     node->val = val;
37
38     node->next = head;
39     head = node;
40
41     if (tail == NULL) {
42         tail = head;
43     }
44     tail->next = head;
45 }
46
47 void Delete(Node *prev) {
48     Node *cur = prev->next;
49     Node *next = cur->next;
50     prev->next = next;
51
52     if (cur == prev) {
53         head = tail = NULL;
```

```
54     free(cur);
55     return;
56 }
57
58 if (cur == head) {
59     head = next;
60 }
61
62 if (cur == tail) {
63     tail = prev;
64 }
65
66 free(cur);
67 }
68
69 void Print() {
70     if (IsEmpty()) {
71         return;
72     }
73
74     Node *iter = head;
75     do {
76         printf("%d\t", iter->val);
77         iter = iter->next;
78     } while (iter != head);
79 }
80
81 void Free() {
82     // TODO:
83 }
```

```
1 /**
2  * file: llist.h
3  *
4  * Created by hengxin on 12/19/21.
5  */
6
7 #ifndef C_PL_CODING_0_11_LINKEDLIST_LLIST_H_
8 #define C_PL_CODING_0_11_LINKEDLIST_LLIST_H_
9
10 #include <stddef.h>
11
12 // recursive declaration
13 typedef struct node {
14     int val;
15     struct node *next;
16 } Node;
17
18 extern Node *head;
19 extern Node *tail;
20
21 void Init(void);
22 int IsEmpty(void);
23 int IsSingleton(void);
24 int Head(void);
25 void Push(int val);
26 Node* Pop(void);
27 void Search(int val);
28 void Insert(Node *node, int val);
29
30 /**
31  * Delete the node after this NODE.
32  * The caller is responsible for ensuring that
33  * node->next is a node in the list.
34  *
35  * @param node
36  */
37 void Delete(Node *prev);
38
39 void Free(void);
40 void Print(void);
41
42 #endif //C_PL_CODING_0_11_LINKEDLIST_LLIST_H_
```

```

1  /**
2   * file: josephus-ll.c
3   *
4   * Created by hengxin on 12/19/21.
5   */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <assert.h>
10 #include "ll/ll.h"
11
12 void SitInCircle(LinkedList *list, int num);
13 void KillUntilOne(LinkedList *list);
14 int GetSurvivor(const LinkedList *list);
15
16 int main(int argc, char *argv[]) {
17     if (argc < 2) {
18         printf("Usage: josephus num\n");
19         return 0;
20     }
21     int num = strtol(argv[1], NULL, 10);
22
23     LinkedList list;
24     for (int i = 1; i <= num; i++) {
25         Init(&list);
26         SitInCircle(&list, i);
27         KillUntilOne(&list);
28         printf("%d: %d\n", i, GetSurvivor(&list));
29     }
30
31     Free(&list);
32
33     return 0;
34 }
35
36 void SitInCircle(LinkedList *list, int num) {
37     for (int i = 1; i <= num; i++) {
38         Append(list, i);
39     }
40 }
41
42 void KillUntilOne(LinkedList *list) {
43     Node *node = list->head;
44
45     while (!IsSingleton(list)) {
46         Delete(list, node);
47         node = node ->next;
48     }
49 }
50
51 int GetSurvivor(const LinkedList *list) {
52     assert(IsSingleton(list));
53

```



```
54     return Head(list);  
55 }
```

```
1  /**
2   * file: josephus.h
3   *
4   * Created by hengxin on 12/19/21.
5   */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include "llist/llist.h"
10
11 void SitInCircle(int num);
12 void KillUntilOne(void);
13
14 int main(int argc, char *argv[]) {
15     if (argc < 2) {
16         printf("Usage: josephus num\n");
17         return 0;
18     }
19
20     int num = strtol(argv[1], NULL, 10);
21     // SitInCircle(num);
22     // KillUntilOne();
23     // printf("%d: %d\n", num, Head());
24
25     for (int i = 1; i <= num; i++) {
26         Init();
27         SitInCircle(i);
28         KillUntilOne();
29         printf("%d: %d\n", i, Head());
30     }
31
32     Free();
33
34     return 0;
35 }
36
37 void SitInCircle(int num) {
38     for (int i = num; i > 0; i--) {
39         Push(i);
40     }
41 }
42
43 void KillUntilOne() {
44     Node *node = head;
45
46     while (!IsSingleton()) {
47         Delete(node);
48         node = node ->next;
49     }
50 }
```

```
1 # 11-linkedlist
2
3 ## `ll`
4 - `ll/ll.h`
5 - `ll/ll.c`
6 - `josephus-ll.c`
7
8 ## `llist`
9 - `llist/llist.h`
10 - `llist/llist.c`
11 - `josephus-llist.c`
```