```c
//
// Created by hfwei on 2023/12/20.
//

#include <stdlib.h>
#include <stdio.h>
#include "ll.h"

void Init(LinkedList *list) {
  list->head = NULL;
  list->tail = NULL;
}

bool IsEmpty(const LinkedList *list) {
  return list->head == NULL;
}

bool IsSingleton(const LinkedList *list) {
  // return list->head == list->tail;
  return !IsEmpty(list) && (list->head == list->tail);
}

int GetHeadVal(const LinkedList *list) {
  if (IsEmpty(list)) {
    return -1;
  }

  return list->head->val;
}

void Append(LinkedList *list, int val) {
  Node *node = malloc(sizeof *node);
  if (node == NULL) {
    printf("malloc failed\n");
    return;
  }
  node->val = val;

  if (IsEmpty(list)) {
    list->head = node;
  } else {
    list->tail->next = node;
  }

```

```c
  list->tail = node;
  list->tail->next = list->head;
}

void Delete(LinkedList *list, Node *prev) {
  if (IsEmpty(list)) {
    return;
  }

  if (IsSingleton(list)) {
    free(list->head);
    Init(list);
    return;
  }

  Node *cur = prev->next;
  Node *next = cur->next;

  prev->next = next;

  // cur != list->head || cur != list->tail
  if (cur == list->head) {
    list->head = next;
  }

  if (cur == list->tail) {
    list->tail = prev;
  }

  free(cur);
}

void Print(const LinkedList *list) {
  if (IsEmpty(list)) {
    printf("empty list\n");
    return;
  }
  Node *node = list->head;

  do {
    printf("%d ", node->val);
    node = node->next;
  } while (node != list->head);

```

```c
  // wrong version
  // while (node != list->head) {
  //   printf("%d ", node->val);
  //   node = node->next;
  // }
}

void Free(LinkedList *list) {
  while (!IsEmpty(list)) {
    Delete(list, list->head);
  }
}
```

```c
//
// Created by hfwei on 2023/12/20.
//

#ifndef INC_2023_CPL_CODING_0_13_LINKED_LIST_LL_LL_H_
#define INC_2023_CPL_CODING_0_13_LINKED_LIST_LL_LL_H_

// adding code below

#include <stdbool.h>
typedef struct node {
  int val;
  struct node *next;
} Node;

typedef struct ll {
  Node *head;
  Node *tail;
  // int size;
} LinkedList;

void Init(LinkedList *list);

bool IsEmpty(const LinkedList *list);
bool IsSingleton(const LinkedList *list);

/**
 * @brief Get the value of the head node.
 * @param list
 * @return -1 if list is empty, otherwise the value of the
 head node.
 */
int GetHeadVal(const LinkedList *list);
Node *Search(const LinkedList *list, int val);

void Print(const LinkedList *list);

void Append(LinkedList *list, int val);
void Insert(LinkedList *list, Node *prev, int val);

/**
 * @brief Delete the node after prev from list.
 * @param list
 * @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_
```

```c
//
// Created by hfwei on 2023/12/20.
//

#include <stdio.h>
#include <assert.h>
#include "ll/ll.h"

#define NUM 10

void SitAroundCircle(LinkedList *list, int num);
void KillUntilOne(LinkedList *list);
int GetSurvivor(const LinkedList *list);

int main(void) {
  printf("I hate the Josephus game!\n");

  LinkedList list;
  Init(&list);

  SitAroundCircle(&list, NUM);
  // Print(&list);

  KillUntilOne(&list);
  int survivor = GetSurvivor(&list);
  printf("%d : %d\n", NUM, survivor);

  Free(&list);

  return 0;
}

void SitAroundCircle(LinkedList *list, int num) {
  for (int i = 1; i <= num; i++) {
    Append(list, i);
  }
}

void KillUntilOne(LinkedList *list) {
  Node *node = list->head;

  while (!IsSingleton(list)) {
    // use node to delete node->next
    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  }
```

```
1 add_executable(josephus josephus.c ll/ll.c)
```