```c
/**
 * file: ll.c
 *
 * Created by hengxin on 12/19/21.
 */

#include <stddef.h>
#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 (! IsEmpty(list)) &&
         (list->head == list->tail);
}

int Head(const LinkedList *list) {
  return list->head->val;
}

Node *Search(const LinkedList *list, int val) {
  if (IsEmpty(list)) {
    return NULL;
  }

  Node *iter = list->head;
  do {
    if (iter->val == val) {
      return iter;
    }
    iter = iter->next;
  } while (iter != list->head);

  return NULL;
}

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

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

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

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

  node->next = prev->next;
  prev->next = node;

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

void Delete(LinkedList *list, Node *prev) {
  Node *cur = prev->next;
  Node *next = cur->next;
  prev->next = next;

  if (cur == prev) {
    list->head = NULL;
    list->tail = NULL;
    free(cur);
    return;
  }

  if (cur == list->head) {
    list->head = next;
  }

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

  free(cur);
}

void Print(const LinkedList *list) {
  if (IsEmpty(list)) {
    return;
  }

  Node *iter = list->head;
  do {
    printf("%d\t", iter->val);
    iter = iter->next;
  } while (iter != list->head);
}

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

```c
/**
 * file: ll.h
 *
 * Created by hengxin on 12/19/21.
 */

#include <stdbool.h>

#ifndef C_PL_CODING_0_11_LINKEDLIST_LL_LL_H_
#define C_PL_CODING_0_11_LINKEDLIST_LL_LL_H_

typedef struct node {
  int val;
  struct node *next;
} Node;

typedef struct ll {
  Node *head;
  Node *tail;
} LinkedList;

void Init(LinkedList *list);

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

int Head(const LinkedList *list);
Node *Search(const LinkedList *list, int val);

void Append(LinkedList *list, int val);
/**
 * The caller is responsible for ensuring that
 * *prev is a pointer to a node in the list.
 *
 * @param list
 * @param prev
 * @param val
 */
void Insert(LinkedList *list, Node *prev, int val);
void Delete(LinkedList *list, Node *prev);

void Free(LinkedList *list);
void Print(const LinkedList *list);

#endif //C_PL_CODING_0_11_LINKEDLIST_LL_LL_H_
```

```c
/**
 * file: llist.c
 *
 * Created by hengxin on 12/19/21.
 */

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

Node *head = NULL;
Node *tail = NULL;

void Init() {
  head = tail = NULL;
}

int IsEmpty() {
  return (head == NULL);
}

int IsSingleton() {
  return (head == tail) && (head != NULL);
}

int Head() {
  return head->val;
}

void Push(int val) {
  Node *node = malloc(sizeof(*node));
  if (node == NULL) {
    printf("Error: malloc failed in Push()\n");
    exit(EXIT_FAILURE);
  }
  node->val = val;

  node->next = head;
  head = node;

  if (tail == NULL) {
    tail = head;
  }
  tail->next = head;
}

void Delete(Node *prev) {
  Node *cur = prev->next;
  Node *next = cur->next;
  prev->next = next;

  if (cur == prev) {
    head = tail = NULL;
    free(cur);
    return;
  }

  if (cur == head) {
    head = next;
  }

  if (cur == tail) {
    tail = prev;
  }

  free(cur);
}

void Print() {
  if (IsEmpty()) {
    return;
  }

  Node *iter = head;
  do {
    printf("%d\t", iter->val);
    iter = iter->next;
  } while (iter != head);
}

void Free() {
  // TODO:
}
```

```c
/**
 * file: llist.h
 *
 * Created by hengxin on 12/19/21.
 */

#ifndef C_PL_CODING_0_11_LINKEDLIST_LLIST_H_
#define C_PL_CODING_0_11_LINKEDLIST_LLIST_H_

#include <stddef.h>

// recursive declaration
typedef struct node {
  int val;
  struct node *next;
} Node;

extern Node *head;
extern Node *tail;

void Init(void);
int IsEmpty(void);
int IsSingleton(void);
int Head(void);
void Push(int val);
Node* Pop(void);
void Search(int val);
void Insert(Node *node, int val);

/**
 * Delete the node *after* this NODE.
 * The caller is responsible for ensuring that
 * node->next is a node in the list.
 *
 * @param node
 */
void Delete(Node *prev);

void Free(void);
void Print(void);

#endif //C_PL_CODING_0_11_LINKEDLIST_LLIST_H_
```

```c
/**
 * file: josephus-ll.c
 *
 * Created by hengxin on 12/19/21.
 */

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

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

int main(int argc, char *argv[]) {
  if (argc < 2) {
    printf("Usage: josephus num\n");
    return 0;
  }
  int num = strtol(argv[1], NULL, 10);

  LinkedList list;
  for (int i = 1; i <= num; i++) {
    Init(&list);
    SitInCircle(&list, i);
    KillUntilOne(&list);
    printf("%d: %d\n", i, GetSurvivor(&list));
  }

  Free(&list);

  return 0;
}

void SitInCircle(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)) {
    Delete(list, node);
    node = node ->next;
  }
}

int GetSurvivor(const LinkedList *list) {
  assert(IsSingleton(list));

  return Head(list);
}
```

```c
/**
 * file: josephus.h
 *
 * Created by hengxin on 12/19/21.
 */

#include <stdio.h>
#include <stdlib.h>
#include "llist/llist.h"

void SitInCircle(int num);
void KillUntilOne(void);

int main(int argc, char *argv[]) {
  if (argc < 2) {
    printf("Usage: josephus num\n");
    return 0;
  }

  int num = strtol(argv[1], NULL, 10);
//  SitInCircle(num);
//  KillUntilOne();
//  printf("%d: %d\n", num, Head());

  for (int i = 1; i <= num; i++) {
    Init();
    SitInCircle(i);
    KillUntilOne();
    printf("%d: %d\n", i, Head());
  }

  Free();

  return 0;
}

void SitInCircle(int num) {
  for (int i = num; i > 0; i--) {
    Push(i);
  }
}

void KillUntilOne() {
  Node *node = head;

  while (! IsSingleton()) {
    Delete(node);
    node = node ->next;
  }
}
```

```
# 11-linkedlist

## `ll`
- `ll/ll.h`
- `ll/ll.c`
- `josephus-ll.c`

## `llist`
- `llist/llist.h`
- `llist/llist.c`
- `josephus-llist.c`
```