```c
// Make this Hash Table implementation to be thread-safe!

#define SKIP_MAIN
#include "threads-list-simple.c"

#define HASH_BUCKETS (7)
typedef struct __hash_t {
  list_t hlists[HASH_BUCKETS];
  pthread_mutex_t m[HASH_BUCKETS];
} hash_t;

void Hash_Init(hash_t *H) {
  int i;
  for (i = 0; i < HASH_BUCKETS; i++) {
    List_Init(&H->hlists[i]);
  }
}

void Hash_Insert(hash_t *H, int key) {
  int b = key % HASH_BUCKETS;
  List_Insert(&H->hlists[b], key);
}

void Hash_Print(hash_t *H) {
  int i;
  for (i = 0; i < HASH_BUCKETS; i++) {
    printf("LIST %d: ", i);
    List_Print(&H->hlists[i]);
  }
}

int main(int argc, char *argv[]) {
  hash_t myhash;
  Hash_Init(&myhash);
  Hash_Insert(&myhash, 10);
  Hash_Insert(&myhash, 5);
  Hash_Insert(&myhash, 30);
  Hash_Print(&myhash);
  return 0;
}
```

```c
// Make this Linked List implementation to be thread-safe!

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

typedef struct __node_t {
  int key;
  struct __node_t *next;
} node_t;

typedef struct __list_t {
  node_t *head;
} list_t;

void List_Init(list_t *L) { L->head = NULL; }

void List_Insert(list_t *L, int key) {
  node_t *new = malloc(sizeof(node_t));
  if (new == NULL) {
    perror("malloc");
    return;
  }
  new->key = key;
  new->next = L->head;
  L->head = new;
}

int List_Lookup(list_t *L, int key) {
  node_t *tmp = L->head;
  while (tmp) {
    if (tmp->key == key) return 1;
    tmp = tmp->next;
  }
  return 0;
}

void List_Print(list_t *L) {
  node_t *tmp = L->head;
  while (tmp) {
    printf("%d ", tmp->key);
    tmp = tmp->next;
  }
  printf("\n");
}

int main(int argc, char *argv[]) {
  list_t mylist;
  List_Init(&mylist);
  List_Insert(&mylist, 10);
  List_Insert(&mylist, 30);
  List_Insert(&mylist, 5);
  List_Print(&mylist);
  printf("In List: 10? %d 20? %d\n", List_Lookup(&mylist, 10),
         List_Lookup(&mylist, 20));
  return 0;
}
```