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

#define MAX_SIZE 1000 // Define a maximum size for the hash table

// Structure for the hash table
typedef struct {
    int key;
    int value;
} HashEntry;

typedef struct {
    HashEntry entries[MAX_SIZE];
```

```
    int size;
} HashTable;

// Function to create a hash table
HashTable* createHashTable() {
    HashTable* ht =
(HashTable*)malloc(sizeof(HashTa
ble));
    ht->size = 0;
    return ht;
}

// Hash function
int hash(int key) {
```

```
    return abs(key) % MAX_SIZE;
}

// Function to insert into the hash table
void insert(HashTable* ht, int key, int value) {
    int index = hash(key);
    while (ht->entries[index].key != 0) {
        index = (index + 1) % MAX_SIZE; // Linear probing
    }
    ht->entries[index].key = key;
```

```
    ht->entries[index].value = value;
    ht->size++;
}

// Function to search in the hash table
int search(HashTable* ht, int key) {
    int index = hash(key);
    while (ht->entries[index].key != 0) {
        if (ht->entries[index].key == key) {
            return ht->entries[index].value;
```

```
        }
        index = (index + 1) %
MAX_SIZE; // Linear probing
    }
    return -1; // Not found
}

// Function to find two indices that
sum to target
void twoSum(int* nums, int
numsSize, int target) {
    HashTable* ht =
createHashTable();
    for (int i = 0; i < numsSize; i++) {
```

```
        int complement = target - nums[i];
        int foundIndex = search(ht, complement);
        if (foundIndex != -1) {
            printf("[%d, %d]\n", foundIndex, i);
            free(ht); // Free the hash table
            return;
        }
        insert(ht, nums[i], i);
    }
    free(ht); // Free the hash table if no solution is found
```

```
}

int main() {
    int nums[] = {2, 7, 11, 15};
    int target = 9;
    int numsSize = sizeof(nums) / sizeof(nums[0]);

    printf("Input: nums = [2, 7, 11, 15], target = 9\n");
    printf("Output: ");
    twoSum(nums, numsSize, target);
```

```
    return 0;
}
```