1.Give a c program for separate chaining.

Program:

```
#include <stdio.h>

#include <stdlib.h>


// Define the structure for the linked list node

struct Node {

   int data;

   struct Node* next;

};


// Define the structure for the hash table

struct HashTable {

   struct Node** table;

   int size;

};


// Function to create a new node

struct Node* createNode(int data) {

   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

   newNode->data = data;

   newNode->next = NULL;

   return newNode;

}


// Function to create a hash table

struct HashTable* createHashTable(int size) {

   struct HashTable* hashTable = (struct HashTable*)malloc(sizeof(struct HashTable));

   hashTable->size = size;

   hashTable->table = (struct Node**)malloc(size * sizeof(struct Node*));
```

```
    for (int i = 0; i < size; i++) {
        hashTable->table[i] = NULL;
    }
    return hashTable;
}

// Hash function
int hashFunction(struct HashTable* hashTable, int key) {
    return key % hashTable->size;
}

// Function to insert a key into the hash table
void insert(struct HashTable* hashTable, int key) {
    int index = hashFunction(hashTable, key);
    struct Node* newNode = createNode(key);
    newNode->next = hashTable->table[index];
    hashTable->table[index] = newNode;
}

// Function to display the hash table
void displayHashTable(struct HashTable* hashTable) {
    for (int i = 0; i < hashTable->size; i++) {
        struct Node* temp = hashTable->table[i];
        printf("Bucket %d: ", i);
        while (temp) {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}
```

```
// Main function
int main() {
    int size, n, key;

    printf("Enter the size of the hash table: ");
    scanf("%d", &size);

    struct HashTable* hashTable = createHashTable(size);

    printf("Enter the number of keys to be inserted: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        printf("Enter key %d: ", i + 1);
        scanf("%d", &key);
        insert(hashTable, key);
    }

    displayHashTable(hashTable);

    return 0;
}
```

Input:

```
Enter the size of the hash table: 10
Enter the number of keys to be inserted: 5
Enter key 1: 5
Enter key 2: 15
Enter key 3: 25
Enter key 4: 35
Enter key 5: 45
```

Output:

```
Bucket 0: NULL
Bucket 1: NULL
Bucket 2: NULL
Bucket 3: NULL
Bucket 4: NULL
Bucket 5: 45 -> 35 -> 25 -> 15 -> 5 -> NULL
Bucket 6: NULL
Bucket 7: NULL
Bucket 8: NULL
Bucket 9: NULL
```

2.Give a c program for open addressing.

Program:

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

#define EMPTY -1
#define DELETED -2

// Define the structure for the hash table
struct HashTable {
   int* table;
   int size;
};

// Function to create a hash table
struct HashTable* createHashTable(int size) {
   struct HashTable* hashTable = (struct HashTable*)malloc(sizeof(struct HashTable));
   hashTable->size = size;
```

```
    hashTable->table = (int*)malloc(size * sizeof(int));
    for (int i = 0; i < size; i++) {
        hashTable->table[i] = EMPTY;
    }
    return hashTable;
}

// Hash function
int hashFunction(struct HashTable* hashTable, int key) {
    return key % hashTable->size;
}

// Linear probing function to find the next available slot
int linearProbe(struct HashTable* hashTable, int key) {
    int index = hashFunction(hashTable, key);
    int i = 0;
    while (hashTable->table[(index + i) % hashTable->size] != EMPTY &&
           hashTable->table[(index + i) % hashTable->size] != DELETED) {
        i++;
    }
    return (index + i) % hashTable->size;
}

// Function to insert a key into the hash table
void insert(struct HashTable* hashTable, int key) {
    int index = linearProbe(hashTable, key);
    hashTable->table[index] = key;
}

// Function to delete a key from the hash table
void delete(struct HashTable* hashTable, int key) {
    int index = hashFunction(hashTable, key);
    int i = 0;
    while (hashTable->table[(index + i) % hashTable->size] != EMPTY) {
        if (hashTable->table[(index + i) % hashTable->size] == key) {
            hashTable->table[(index + i) % hashTable->size] = DELETED;
            return;
        }
        i++;
    }
    printf("Key %d not found\n", key);
}

// Function to search for a key in the hash table
int search(struct HashTable* hashTable, int key) {
    int index = hashFunction(hashTable, key);
    int i = 0;
    while (hashTable->table[(index + i) % hashTable->size] != EMPTY) {
        if (hashTable->table[(index + i) % hashTable->size] == key) {
            return (index + i) % hashTable->size;
        }
        i++;
    }
    return -1; // Key not found
}

// Function to display the hash table
void displayHashTable(struct HashTable* hashTable) {
    for (int i = 0; i < hashTable->size; i++) {
        if (hashTable->table[i] == EMPTY) {
            printf("Bucket %d: EMPTY\n", i);
        } else if (hashTable->table[i] == DELETED) {
            printf("Bucket %d: DELETED\n", i);
        } else {
            printf("Bucket %d: %d\n", i, hashTable->table[i]);
        }
    }
}

// Main function
int main() {
    int size, n, key, choice;

    printf("Enter the size of the hash table: ");
    scanf("%d", &size);

    struct HashTable* hashTable = createHashTable(size);

    while (1) {
        printf("\n1. Insert\n2. Delete\n3. Search\n4. Display\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter key to insert: ");
                scanf("%d", &key);
                insert(hashTable, key);
                break;
            case 2:
                printf("Enter key to delete: ");
                scanf("%d", &key);
                delete(hashTable, key);
                break;
            case 3:
                printf("Enter key to search: ");
                scanf("%d", &key);
                int index = search(hashTable, key);
                if (index != -1) {
                  printf("Key %d found at index %d\n", key, index);
                } else {
                  printf("Key %d not found\n", key);
                }
                break;
            case 4:
                displayHashTable(hashTable);
                break;
            case 5:
                free(hashTable->table);
                free(hashTable);

                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

Input:

```
Enter the size of the hash table: 10

1. Insert
2. Delete
```

```
3. Search
4. Display
5. Exit
Enter your choice: 1
Enter key to insert: 5

1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 1
Enter key to insert: 15

1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 4
```

Output:

```
Bucket 0: EMPTY
Bucket 1: EMPTY
Bucket 2: EMPTY
Bucket 3: EMPTY
Bucket 4: EMPTY
Bucket 5: 5
Bucket 6: 15
Bucket 7: EMPTY
Bucket 8: EMPTY
Bucket 9: EMPTY
```