Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are integers. Design and develop a Program in C that uses Hash function H: K -> L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

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

#define MAX_EMPLOYEES 100 // Maximum number of employees
#define HASH_TABLE_SIZE 10 // Size of the hash table

// Structure for employee record
struct Employee {
    int key; // 4-digit key
    // Other employee details can be added here
};

// Function prototypes
int hashFunction(int key);
void insertEmployee(struct Employee employees[], int hashTable[], struct
Employee emp);
void displayHashTable(int hashTable[]);

int main() {
    struct Employee employees[MAX_EMPLOYEES]; // Array to hold employee
records
    int hashTable[HASH_TABLE_SIZE] = {0}; // Hash table initialized with 0
    int n, m, i;

    // Input the number of employees
    printf("Enter the number of employees: ");
    scanf("%d", &n);
```

```
    // Input employee records
    printf("Enter employee records:\n");
    for (i = 0; i < n; ++i) {
        printf("Employee %d:\n", i + 1);
        printf("Enter 4-digit key: ");
        scanf("%d", &employees[i].key);
        // Additional details can be input here
        insertEmployee(employees, hashTable, employees[i]);
    }

    // Display the hash table
    printf("\nHash Table:\n");
    displayHashTable(hashTable);

    return 0;
}

// Hash function: H(K) = K mod m
int hashFunction(int key) {
    return key % HASH_TABLE_SIZE;
}

// Function to insert an employee into the hash table
void insertEmployee(struct Employee employees[], int hashTable[], struct Employee
emp) {
    int index = hashFunction(emp.key);

    // Linear probing to resolve collisions
    while (hashTable[index] != 0) {
        index = (index + 1) % HASH_TABLE_SIZE;
    }

    // Insert the employee key into the hash table
    hashTable[index] = emp.key;
}

// Function to display the hash table
void displayHashTable(int hashTable[]) {
    int i;
```

```
    for (i = 0; i < HASH_TABLE_SIZE; ++i) {
        printf("%d -> ", i);
        if (hashTable[i] == 0) {
            printf("Empty\n");
        } else {
            printf("%d\n", hashTable[i]);
        }
    }
}
```

```
Enter the number of employees: 5
Enter employee records:
Employee 1:
Enter 4-digit key: 1234
Employee 2:
Enter 4-digit key: 3456
Employee 3:
Enter 4-digit key: 9087
Employee 4:
Enter 4-digit key: 4567
Employee 5:
Enter 4-digit key: 1009

Hash Table:
0 -> Empty
1 -> Empty
2 -> Empty
3 -> Empty
4 -> 1234
5 -> Empty
6 -> 3456
7 -> 9087
8 -> 4567
9 -> 1009
```