# C File Solution Explanation

The provided C file implements a simple hash table with linear probing for collision resolution. Here is a detailed explanation of the code:

1. Headers and Constants:

```
#include <stdio.h>
#define TABLE_SIZE 11
```

Includes the standard input-output library and defines a constant TABLE_SIZE with a value of 11, representing the size of the hash table.

2. Hash Function:

```
// Hash function
int h1(int key) {
    int x = (key + 7) * (key + 7);
    x = x / 16;
    x = x + key;
    x = x % TABLE_SIZE;
    return x;
}
```

The hash function h1 takes an integer key as input and returns a computed hash value. It modifies the key, performs arithmetic operations, and takes the modulus with TABLE_SIZE to ensure the result fits within the table size.

3. Insertion Function:

```
// Function to insert a key using linear probing
void insert(int hashTable[], int key) {
```

```
    int homeSlot = h1(key);
    int slot = homeSlot;
    while (hashTable[slot] != -1) {
        slot = (slot + 1) % TABLE_SIZE;
    }
    hashTable[slot] = key;
}
```

The insert function inserts a key into the hash table. It uses the hash function to compute the home slot and linear probing to resolve collisions.

4. Main Function:

```
int main() {
    // Initialize the hash table with -1 to indicate empty slots
    int hashTable[TABLE_SIZE];
    for (int i = 0; i < TABLE_SIZE; i++) {
        hashTable[i] = -1;
    }
    // List of keys to insert
    int keys[] = {43, 23, 1, 0, 15, 31, 4, 7, 11, 3};
    int numKeys = sizeof(keys) / sizeof(keys[0]);
    // Insert each key into the hash table
    for (int i = 0; i < numKeys; i++) {
        insert(hashTable, keys[i]);
    }
    // Print the final contents of the hash table
    printf("Final Hash Table:\n");
```

```
    for (int i = 0; i < TABLE_SIZE; i++) {
        printf("Slot %d: %d\n", i, hashTable[i]);
    }
    return 0;
}
```

The main function initializes the hash table, inserts a list of keys, and prints the final contents of the hash table.