COLLISION

LINEAR PROBING:

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define TABLE_SIZE 10
typedef struct {
    int key;
    int value;
    int isOccupied;
} HashEntry;
HashEntry hashTable[TABLE_SIZE];
int hash(int key) {
    return key % TABLE_SIZE;
}
void initTable() {
    for (int i = 0; i < TABLE_SIZE; i++) {
        hashTable[i].isOccupied = 0;
    }
}
void insert(int key, int value) {
    int index = hash(key);
    int startIndex = index;
    while (hashTable[index].isOccupied && hashTable[index].key != key) {
        index = (index + 1) % TABLE_SIZE;
        if (index == startIndex) {
            printf("Hash table is full.\n");
            return;
        }
    }
    hashTable[index].key = key;
    hashTable[index].value = value;
    hashTable[index].isOccupied = 1;
}
int search(int key) {
```

```
    int index = hash(key);
    int startIndex = index;
    while (hashTable[index].isOccupied) {
        if (hashTable[index].key == key) {
            return hashTable[index].value;
        }
        index = (index + 1) % TABLE_SIZE;
        if (index == startIndex) {
            break;
        }
    }
    printf("Key not found.\n");
    return -1;
}
void delete(int key) {
    int index = hash(key);
    int startIndex = index;
    while (hashTable[index].isOccupied) {
        if (hashTable[index].key == key) {
            hashTable[index].isOccupied = 0;
            return;
        }
        index = (index + 1) % TABLE_SIZE;
        if (index == startIndex) {
            break;
        }
    }
    printf("Key not found.\n");
}
void printTable() {
    for (int i = 0; i < TABLE_SIZE; i++) {
        if (hashTable[i].isOccupied) {
            printf("Index %d: Key %d, Value %d\n", i, hashTable[i].key, hashTable[i].value);
        } else {
            printf("Index %d: Empty\n", i);
        }
    }
}
int main() {
```

```
    initTable();
    insert(1, 100);
    insert(11, 200);
    insert(21, 300);
    printf("Hash Table:\n");
    printTable();
    printf("Value for key 11: %d\n", search(11));
    printf("Value for key 2: %d\n", search(2));
    delete(11);
    printf("After deleting key 11:\n");
    printTable();
    return 0;
}
```

OUTPUT:

```
Hash Table:
Index 0: Empty
Index 1: Key 1, Value 100
Index 2: Key 11, Value 200
Index 3: Key 21, Value 300
Index 4: Empty
Index 5: Empty
Index 6: Empty
Index 7: Empty
Index 8: Empty
Index 9: Empty
Value for key 11: 200
Key not found.
Value for key 2: -1
After deleting key 11:
Index 0: Empty
Index 1: Key 1, Value 100
Index 2: Empty
Index 3: Key 21, Value 300
Index 4: Empty
Index 5: Empty
Index 6: Empty
Index 7: Empty
Index 8: Empty
```

Index 9: Empty

SEPARATE HASHING:

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define TABLE_SIZE 10
typedef struct Node {
    int key;
    int value;
    struct Node* next;
} Node;
Node* hashTable[TABLE_SIZE];
int hash(int key) {
    return key % TABLE_SIZE;
}
void initTable() {
    for (int i = 0; i < TABLE_SIZE; i++) {
        hashTable[i] = NULL;
    }
}
Node* createNode(int key, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->key = key;
    newNode->value = value;
    newNode->next = NULL;
    return newNode;
}
void insert(int key, int value) {
    int index = hash(key);
    Node* newNode = createNode(key, value);
    newNode->next = hashTable[index];
    hashTable[index] = newNode;
}
int search(int key) {
    int index = hash(key);
    Node* current = hashTable[index];
```

```
    while (current != NULL) {
        if (current->key == key) {
            return current->value;
        }
        current = current->next;
    }

    printf("Key not found.\n");
    return -1;
}
void delete(int key) {
    int index = hash(key);
    Node* current = hashTable[index];
    Node* prev = NULL;
    while (current != NULL) {
        if (current->key == key) {
            if (prev == NULL) {
                hashTable[index] = current->next;
            } else {
                prev->next = current->next;
            }
            free(current);
            return;
        }
        prev = current;
        current = current->next;
    }
    printf("Key not found.\n");
}
void printTable() {
    for (int i = 0; i < TABLE_SIZE; i++) {
        Node* current = hashTable[i];
        printf("Index %d:", i);
        while (current != NULL) {
            printf(" -> (Key %d, Value %d)", current->key, current->value);
            current = current->next;
        }
        printf(" -> NULL\n");
    }
```

```
}
int main() {
    initTable();
    insert(1, 100);
    insert(11, 200);
    insert(21, 300);
    printf("Hash Table:\n");
    printTable();
    printf("Value for key 11: %d\n", search(11));
    printf("Value for key 2: %d\n", search(2));
    delete(11);
    printf("After deleting key 11:\n");
    printTable();
    return 0;
}
```

OUTPUT:

```
Hash Table:
Index 0: -> NULL
Index 1: -> (Key 21, Value 300) -> (Key 11, Value 200) -> (Key 1, Value 100) -> NULL
Index 2: -> NULL
Index 3: -> NULL
Index 4: -> NULL
Index 5: -> NULL
Index 6: -> NULL
Index 7: -> NULL
Index 8: -> NULL
Index 9: -> NULL
Value for key 11: 200
Key not found.
Value for key 2: -1
After deleting key 11:
Index 0: -> NULL
Index 1: -> (Key 21, Value 300) -> (Key 1, Value 100) -> NULL
Index 2: -> NULL
Index 3: -> NULL
Index 4: -> NULL
Index 5: -> NULL
Index 6: -> NULL
```

Index 7: -> NULL

Index 8: -> NULL

Index 9: -> NULL