**DAY 13 LAB:**

**1.Write a C program to implement hashing using Separate chaining method**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define TABLE\_SIZE 10

typedef struct Node {

int key;

struct Node \*next;

} Node;

Node \*hashTable[TABLE\_SIZE];

int hash(int key) {

return key % TABLE\_SIZE;

}

Node\* createNode(int key) {

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

newNode->key = key;

newNode->next = NULL;

return newNode;

}

void insert(int key) {

int index = hash(key);

Node \*newNode = createNode(key);

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 1;

current = current->next;

}

return 0;

}

void delete(int key) {

int index = hash(key);

Node \*current = hashTable[index];

Node \*prev = NULL;

while (current != NULL && current->key != key) {

prev = current;

current = current->next;

}

if (current == NULL) {

printf("Key %d not found.\n", key);

return;

}

if (prev == NULL) {

hashTable[index] = current->next;

} else {

prev->next = current->next;

}

free(current);

printf("Key %d deleted.\n", key);

}

void printHashTable() {

for (int i = 0; i < TABLE\_SIZE; i++) {

Node \*current = hashTable[i];

printf("Bucket %d: ", i);

while (current != NULL) {

printf("%d -> ", current->key);

current = current->next;

}

printf("NULL\n");

}

}

int main() {

for (int i = 0; i < TABLE\_SIZE; i++)

hashTable[i] = NULL;

insert(10);

insert(20);

insert(15);

insert(25);

insert(30);

insert(5);

insert(17);

printf("Hash Table:\n");

printHashTable();

printf("Searching for 15: %s\n", search(15) ? "Found" : "Not Found");

printf("Searching for 100: %s\n", search(100) ? "Found" : "Not Found");

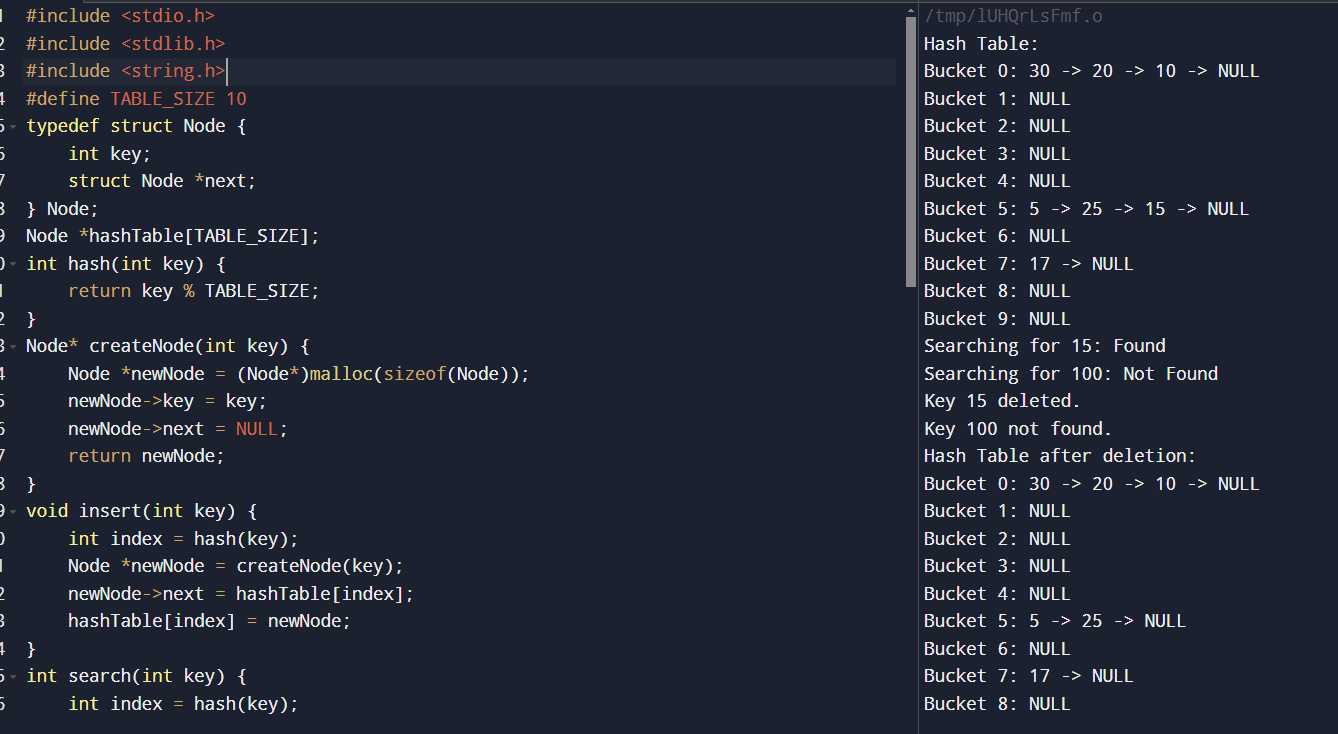
delete(15);

delete(100);

printf("Hash Table after deletion:\n");

printHashTable();

}



**2.Write a C program to implement hashing using Linear Probing method.**

#include <stdio.h>

#include <stdlib.h>

#define TABLE\_SIZE 10

#define EMPTY\_SLOT -1

#define DELETED\_SLOT -2

int hashTable[TABLE\_SIZE];

int hash(int key) {

return key % TABLE\_SIZE;

}

void initializeHashTable() {

for (int i = 0; i < TABLE\_SIZE; i++) {

hashTable[i] = EMPTY\_SLOT;

}

}

void insert(int key) {

int index = hash(key);

int startIndex = index;

while (hashTable[index] != EMPTY\_SLOT && hashTable[index] != DELETED\_SLOT) {

index = (index + 1) % TABLE\_SIZE;

if (index == startIndex) {

printf("Hash table is full!\n");

return;

}

}

hashTable[index] = key;

}

int search(int key) {

int index = hash(key);

int startIndex = index;

while (hashTable[index] != EMPTY\_SLOT) {

if (hashTable[index] == key)

return 1;

index = (index + 1) % TABLE\_SIZE;

if (index == startIndex)

return 0;

}

return 0;

}

void delete(int key) {

int index = hash(key);

int startIndex = index;

while (hashTable[index] != EMPTY\_SLOT) {

if (hashTable[index] == key) {

hashTable[index] = DELETED\_SLOT;

printf("Key %d deleted.\n", key);

return;

}

index = (index + 1) % TABLE\_SIZE;

if (index == startIndex)

return;

}

printf("Key %d not found.\n", key);

}

void printHashTable() {

for (int i = 0; i < TABLE\_SIZE; i++) {

if (hashTable[i] == EMPTY\_SLOT)

printf("Slot %d: EMPTY\n", i);

else if (hashTable[i] == DELETED\_SLOT)

printf("Slot %d: DELETED\n", i);

else

printf("Slot %d: %d\n", i, hashTable[i]);

}

}

int main() {

initializeHashTable();

insert(10);

insert(20);

insert(15);

insert(25);

insert(30);

insert(5);

insert(17);

printf("Hash Table:\n");

printHashTable();

printf("Searching for 15: %s\n", search(15) ? "Found" : "Not Found");

printf("Searching for 100: %s\n", search(100) ? "Found" : "Not Found");

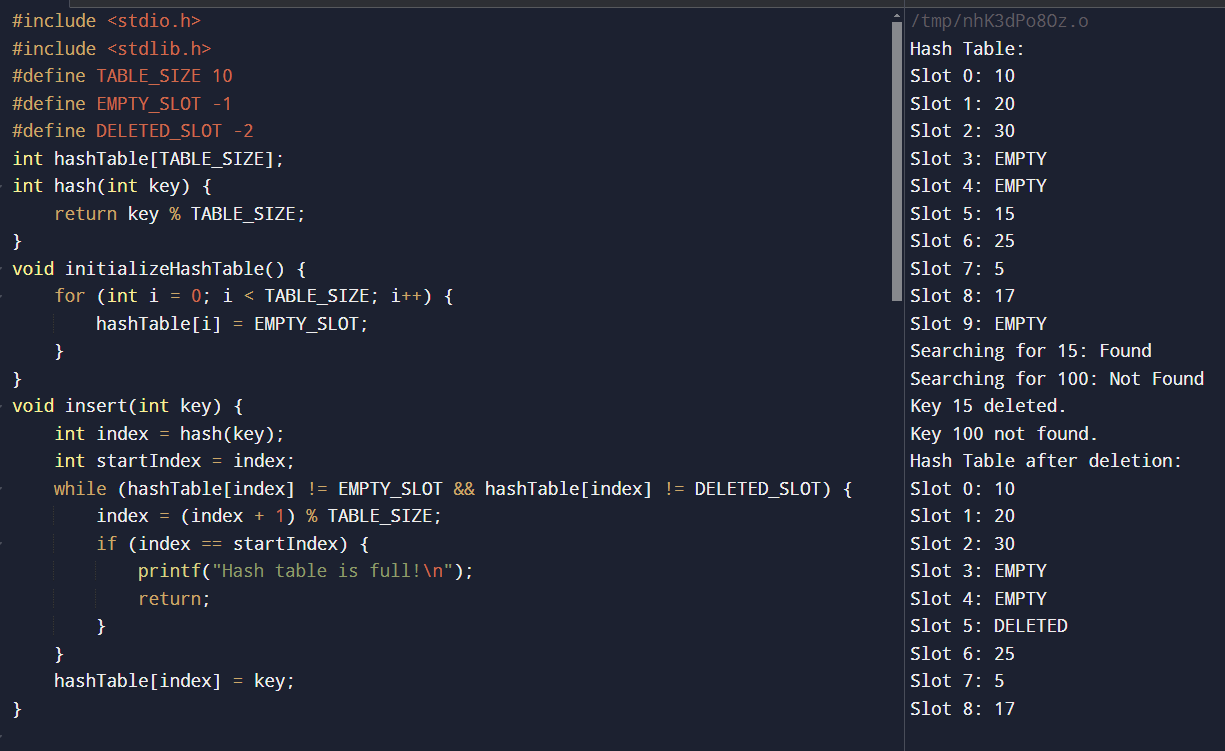
delete(15);

delete(100);

printf("Hash Table after deletion:\n");

printHashTable();

}



**3.Write a C program to implement hashing using Quadratic Probing method.**

#include <stdio.h>

#include <stdlib.h>

#define TABLE\_SIZE 10

typedef struct {

int key;

} HashTableElement;

HashTableElement\* hashTable[TABLE\_SIZE];

int hashFunction(int key) {

return key % TABLE\_SIZE;

}

void insert(int key) {

int index = hashFunction(key);

int i = 0;

while (hashTable[(index + i \* i) % TABLE\_SIZE] != NULL && i < TABLE\_SIZE) {

if (hashTable[(index + i \* i) % TABLE\_SIZE]->key == key) {

printf("Key already exists in the table\n");

return;

}

i++;

}

if (i == TABLE\_SIZE) {

printf("Hash table is full\n");

return;

}

hashTable[(index + i \* i) % TABLE\_SIZE] = (HashTableElement\*) malloc(sizeof(HashTableElement));

hashTable[(index + i \* i) % TABLE\_SIZE]->key = key;

}

void search(int key) {

int index = hashFunction(key);

int i = 0;

while (hashTable[(index + i \* i) % TABLE\_SIZE] != NULL && i < TABLE\_SIZE) {

if (hashTable[(index + i \* i) % TABLE\_SIZE]->key == key) {

printf("Key found in the table\n");

return;

}

i++;

}

printf("Key not found in the table\n");

}

void display() {

for (int i = 0; i < TABLE\_SIZE; i++) {

if (hashTable[i] != NULL) {

printf("Index %d: Key = %d\n", i, hashTable[i]->key);

} else {

printf("Index %d: Empty\n", i);

}

}

}

int main() {

for (int i = 0; i < TABLE\_SIZE; i++) {

hashTable[i] = NULL;

}

insert(5);

insert(15);

insert(25);

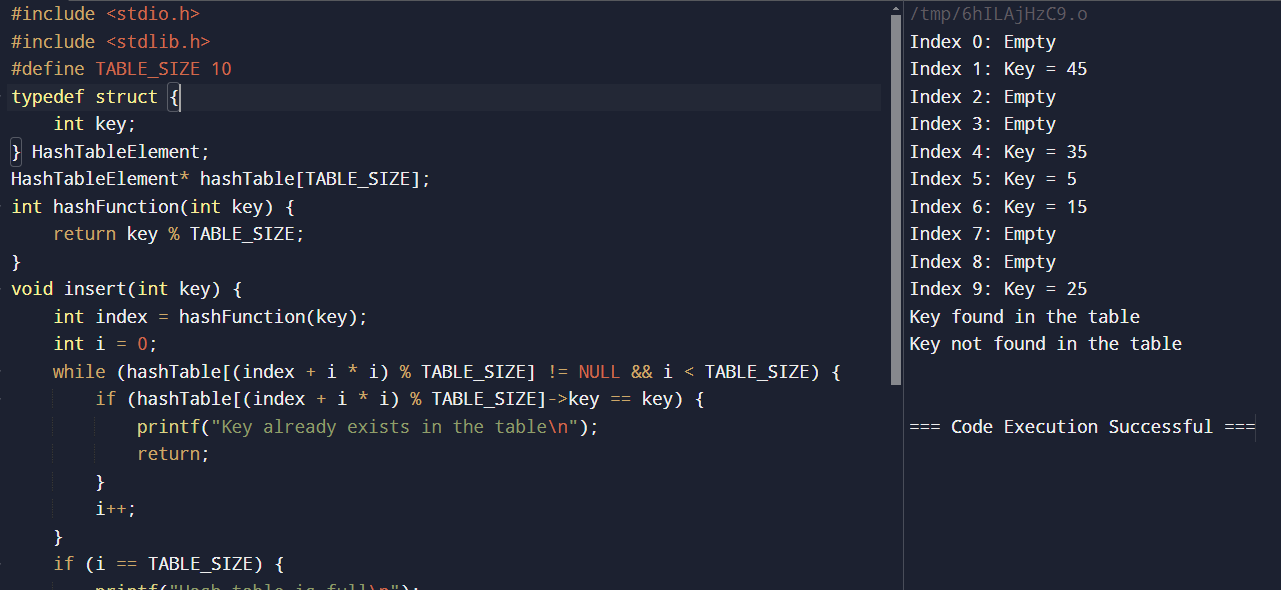
insert(35);

insert(45);

display();

search(25);

search(30);

}

**4.Write a C program to implement hashing using Double hashing method.**

#include <stdio.h>

#include <stdlib.h>

#define TABLE\_SIZE 10

typedef struct {

int key;

} HashTableElement;

HashTableElement\* hashTable[TABLE\_SIZE];

int hashFunction1(int key) {

return key % TABLE\_SIZE;

}

int hashFunction2(int key) {

return (key / TABLE\_SIZE) % (TABLE\_SIZE - 1) + 1;

}

void insert(int key) {

int index = hashFunction1(key);

int probe = hashFunction2(key);

int i = 0;

while (hashTable[(index + i \* probe) % TABLE\_SIZE] != NULL && i < TABLE\_SIZE) {

if (hashTable[(index + i \* probe) % TABLE\_SIZE]->key == key) {

printf("Key already exists in the table\n");

return;

}

i++;

}

if (i == TABLE\_SIZE) {

printf("Hash table is full\n");

return;

}

hashTable[(index + i \* probe) % TABLE\_SIZE] = (HashTableElement\*) malloc(sizeof(HashTableElement));

hashTable[(index + i \* probe) % TABLE\_SIZE]->key = key;

}

void search(int key) {

int index = hashFunction1(key);

int probe = hashFunction2(key);

int i = 0;

while (hashTable[(index + i \* probe) % TABLE\_SIZE] != NULL && i < TABLE\_SIZE) {

if (hashTable[(index + i \* probe) % TABLE\_SIZE]->key == key) {

printf("Key found in the table\n");

return;

}

i++;

}

printf("Key not found in the table\n");

}

void display() {

for (int i = 0; i < TABLE\_SIZE; i++) {

if (hashTable[i] != NULL) {

printf("Index %d: Key = %d\n", i, hashTable[i]->key);

} else {

printf("Index %d: Empty\n", i);

}

}

}

int main() {

for (int i = 0; i < TABLE\_SIZE; i++) {

hashTable[i] = NULL;

}

insert(5);

insert(15);

insert(25);

insert(35);

insert(45);

display();

search(25);

search(30);

}

