AIM: **HASH TABLE IMPLEMENTATION**

Implement a hash table data structure using different hash function and collision resolution techniques such as chaining and open addressing.

PROGRAM:

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

#define SIZE 10

// Node for chaining
struct Node {
    int data;
    struct Node* next;
};

struct Node* chainTable[SIZE]; // Hash table for chaining
int openAddressingTable[SIZE];  // Hash table for open addressing

// Hash function
int hashFunction(int value) {
    return value % SIZE;
}

// Chaining
void insertChaining(int value) {
    int index = hashFunction(value);
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = chainTable[index];
    chainTable[index] = newNode;
}
```

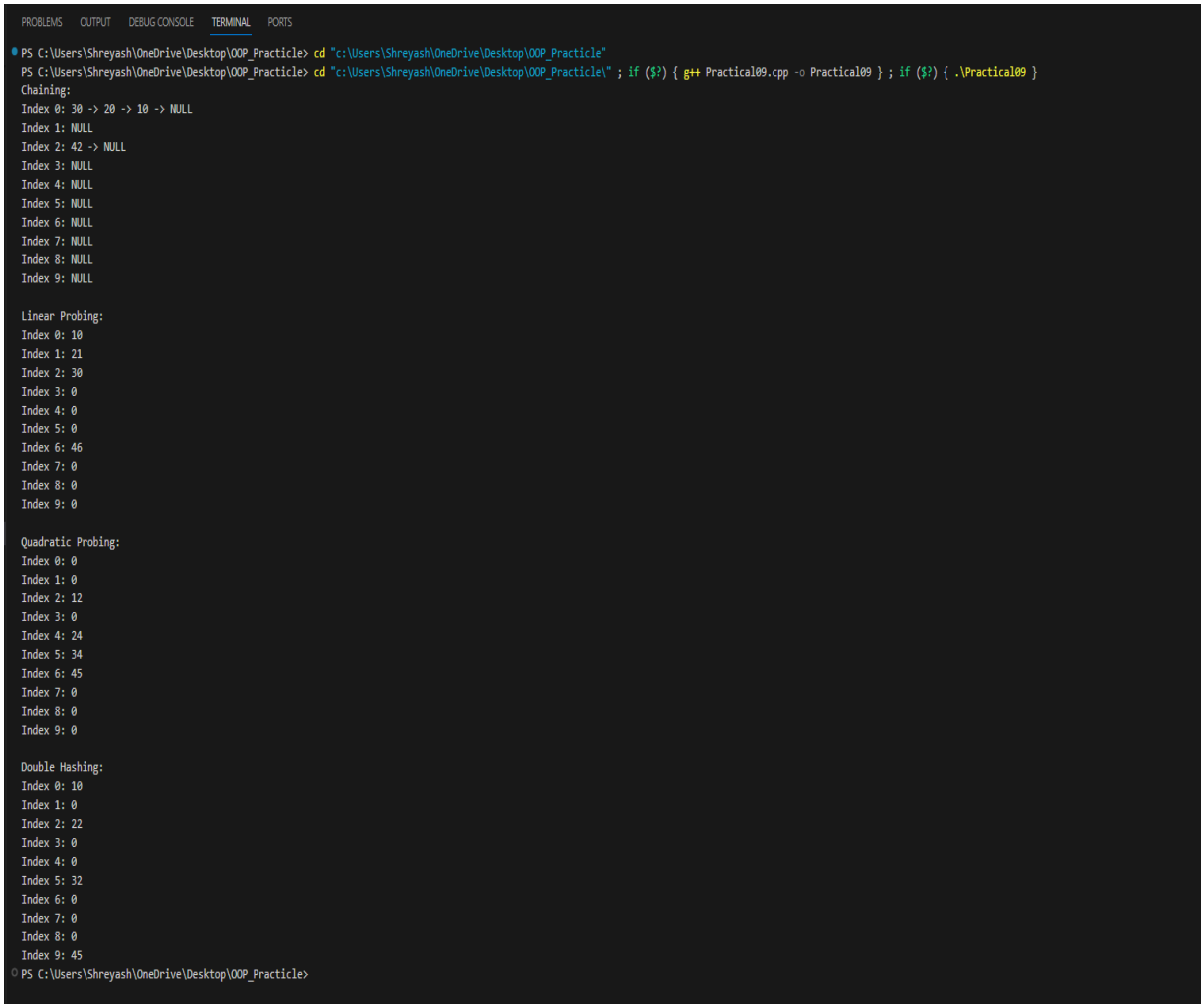
```
void displayChaining() {
    for (int i = 0; i < SIZE; i++) {
        struct Node* temp = chainTable[i];
        printf("Index %d: ", i);
        while (temp) {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

// Linear Probing
void insertLinearProbing(int value) {
    int index = hashFunction(value);
    while (openAddressingTable[index] != 0) {
        index = (index + 1) % SIZE;
    }
    openAddressingTable[index] = value;
}

void displayLinearProbing() {
    for (int i = 0; i < SIZE; i++) {
        printf("Index %d: %d\n", i, openAddressingTable[i]);
    }
}

// Quadratic Probing
void insertQuadraticProbing(int value) {
    int index = hashFunction(value);
```

```
    for (int i = 0; i < SIZE; i++) {
        int newIndex = (index + i * i) % SIZE;
        if (openAddressingTable[newIndex] == 0) {
            openAddressingTable[newIndex] = value;
            return;
        }
    }
}

// Double Hashing
int secondHashFunction(int value) {
    return 7 - (value % 7); // Secondary hash function
}

void insertDoubleHashing(int value) {
    int index = hashFunction(value);
    int stepSize = secondHashFunction(value);
    while (openAddressingTable[index] != 0) {
        index = (index + stepSize) % SIZE;
    }
    openAddressingTable[index] = value;
}

// Main function
int main() {
    // Chaining
    printf("Chaining:\n");
    insertChaining(10);
    insertChaining(20);
    insertChaining(30);
    insertChaining(42);
```

```
displayChaining();

// Open Addressing
printf("\nLinear Probing:\n");
for (int i = 0; i < SIZE; i++) {
   openAddressingTable[i] = 0; // Initialize the table
}
insertLinearProbing(10);
insertLinearProbing(21);
insertLinearProbing(30);
insertLinearProbing(46);
displayLinearProbing();

// Quadratic Probing
printf("\nQuadratic Probing:\n");
for (int i = 0; i < SIZE; i++) {
   openAddressingTable[i] = 0; // Initialize the table
}
insertQuadraticProbing(12);
insertQuadraticProbing(24);
insertQuadraticProbing(34);
insertQuadraticProbing(45);
displayLinearProbing(); // Reusing display function

// Double Hashing
printf("\nDouble Hashing:\n");
for (int i = 0; i < SIZE; i++) {
   openAddressingTable[i] = 0; // Initialize the table
}
insertDoubleHashing(10);
insertDoubleHashing(22);
```

```
    insertDoubleHashing(32);
    insertDoubleHashing(45);
    displayLinearProbing(); // Reusing display function


    return 0;
}
```

OUTPUT



GITHUB LINK : https://github.com/ShreyashGajbhiye453/Data-Structure-Practical-No.-01