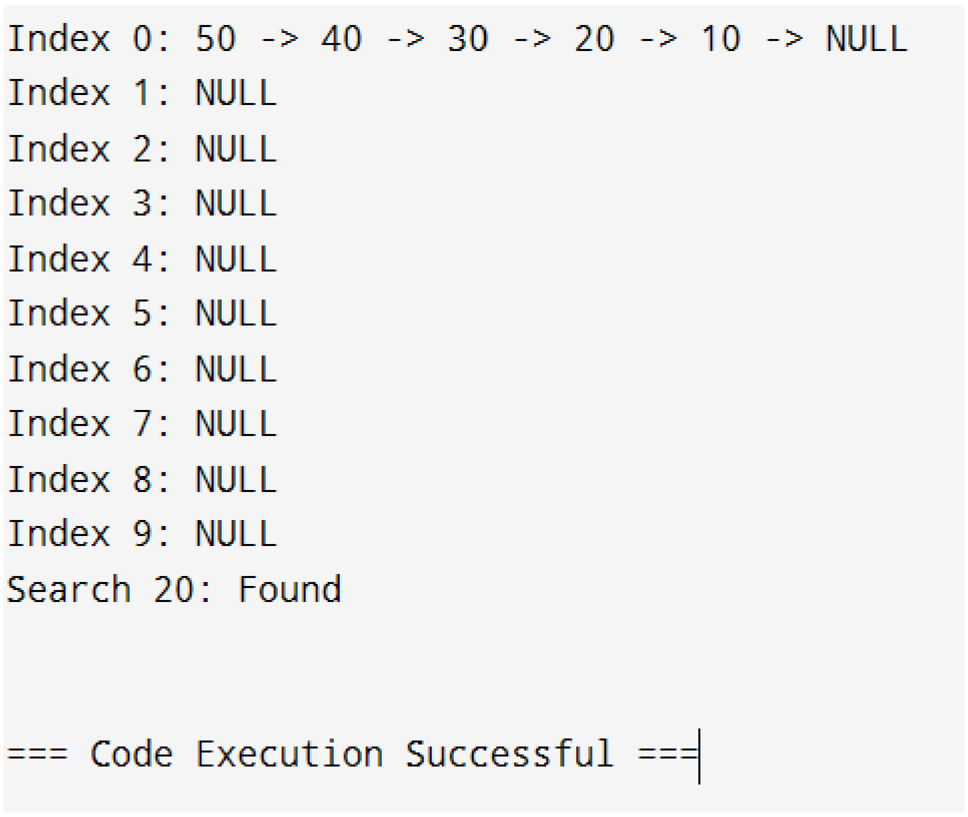
DAY-13-LAB

1.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 data;

    struct Node* next;

} Node;


Node* hashTable[TABLE_SIZE];


int hashFunction(int key) {

    return key % TABLE_SIZE;

}


void insert(int key) {

    int index = hashFunction(key);

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

    newNode->data = key;

    newNode->next = hashTable[index];

    hashTable[index] = newNode;

}
```

```
int search(int key) {
    int index = hashFunction(key);
    Node* temp = hashTable[index];
    while (temp) {
        if (temp->data == key) return 1;
        temp = temp->next;
    }
    return 0;
}

void display() {
    for (int i = 0; i < TABLE_SIZE; i++) {
        Node* temp = hashTable[i];
        printf("Index %d: ", i);
        while (temp) {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}
int main() {
    insert(10);
    insert(20);
    insert(30);
    insert(40);
    insert(50);
    display();

    printf("Search 20: %s\n", search(20) ? "Found" : "Not Found");

    return 0;

}
```

OUTPUT:



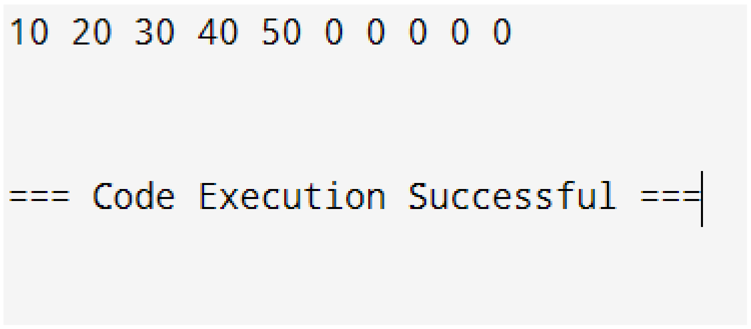
```
Index 0: 50 -> 40 -> 30 -> 20 -> 10 -> NULL
Index 1: NULL
Index 2: NULL
Index 3: NULL
Index 4: NULL
Index 5: NULL
Index 6: NULL
Index 7: NULL
Index 8: NULL
Index 9: NULL
Search 20: Found


=== Code Execution Successful ===
```

2.C program to implement hashing using Linear Probing method.

```
#include <stdio.h>

#define SIZE 10



int hashTable[SIZE] = {0};



int hashFunction(int key) {

    return key % SIZE;
```

```
}

void insert(int key) {
    int index = hashFunction(key);
    while (hashTable[index] != 0) {
        index = (index + 1) % SIZE;
    }
    hashTable[index] = key;
}

void display() {
    for (int i = 0; i < SIZE; i++) {
        printf("%d ", hashTable[i]);
    }
    printf("\n");
}

int main() {
    insert(10);
    insert(20);
    insert(30);
    insert(40);
    insert(50);
    display();
    return 0;
}
```

OUTPUT:

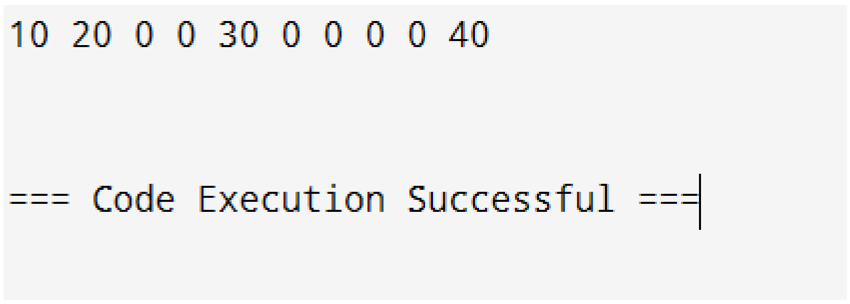


```
10 20 30 40 50 0 0 0 0 0


=== Code Execution Successful ===
```

3. C program to implement hashing using Quadratic Probing method.

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

#define TABLE_SIZE 10

int hash(int key) {
    return key % TABLE_SIZE;
}

int quadraticProbing(int hashTable[], int key) {
    int index = hash(key);
    int i = 0;
    while (hashTable[(index + i * i) % TABLE_SIZE] != 0) {
        i++;
    }
    return (index + i * i) % TABLE_SIZE;
}

void insert(int hashTable[], int key) {
    int index = quadraticProbing(hashTable, key);
    hashTable[index] = key;
}

void display(int hashTable[]) {
    for (int i = 0; i < TABLE_SIZE; i++) {
        printf("%d ", hashTable[i]);
    }
    printf("\n");
}

int main() {
    int hashTable[TABLE_SIZE] = {0};
    insert(hashTable, 10);
    insert(hashTable, 20);
    insert(hashTable, 30);
    insert(hashTable, 40);
    display(hashTable);
    return 0;
}
```

OUTPUT:

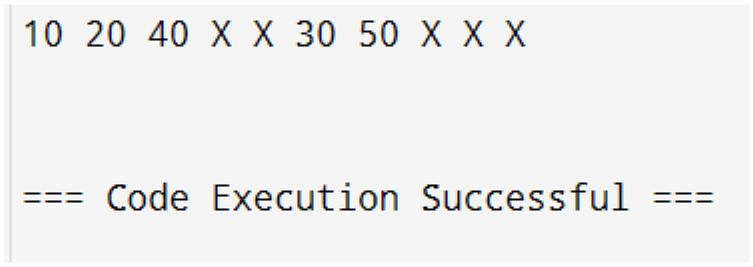


```
10 20 0 0 30 0 0 0 0 40


=== Code Execution Successful ===
```

4.C program to implement hashing using Double hashing method.

```
#include <stdio.h>

#include <stdlib.h>



#define TABLE_SIZE 10



int hash1(int key) {

    return key % TABLE_SIZE;

}



int hash2(int key) {

    return 7 - (key % 7);

}



void insert(int table[], int key) {

    int index = hash1(key);

    int step = hash2(key);

    while (table[index] != -1) {

        index = (index + step) % TABLE_SIZE;

    }
    table[index] = key;
}

void display(int table[]) {
    for (int i = 0; i < TABLE_SIZE; i++) {
        if (table[i] != -1)
            printf("%d ", table[i]);
        else
            printf("X ");
    }
    printf("\n");
}

int main() {
    int hashTable[TABLE_SIZE];
    for (int i = 0; i < TABLE_SIZE; i++) hashTable[i] = -1;

    insert(hashTable, 10);
    insert(hashTable, 20);
    insert(hashTable, 30);
    insert(hashTable, 40);
    insert(hashTable, 50);

    display(hashTable);
    return 0;

}
```

OUTPUT:



```
10 20 40 X X 30 50 X X X


=== Code Execution Successful ===
```