# **Advanced Data Structures**

## **Assignment-9**

Name: Ayush Sanjay Dhangar

Roll no: 42

Prn:12210406

Div: IT-A (Batch 2)

Problem statement:Write C/C++ program to demonstrate Hashing using linear or quadratic probing

* **Hash Function**: The hash function calculates the hash value for a given key based on the size of the hash table. It simply takes the modulo of the key with the size of the hash table.
* **Linear Probing**: The linearProbing function handles collisions in the hash table by linearly searching for the next available slot. It iterates through the hash table until an empty slot (-1) is found or until it has searched the entire table. If the table is full, it returns -1.
* **Quadratic Probing**: Similar to linear probing, the quadraticProbing function handles collisions, but it uses a quadratic function (i \* i) to calculate the next probing index. This helps to reduce clustering that can occur with linear probing.
* **Insertion**: The insert function inserts a key into the hash table using either linear or quadratic probing technique based on the user's choice. It first calculates the index using the chosen probing method and then inserts the key into the hash table at that index.
* **Search**: The search function searches for a key in the hash table using either linear or quadratic probing technique based on the user's choice. It calculates the index where the key is expected to be found and iterates through the table until it either finds the key or determines that it's not present.
* **Display**: The display function simply prints out the contents of the hash table.
* **Main Function**:

1. It first prompts the user to input the size of the hash table and the probing technique to be used.
2. Then, it enters a loop where the user can choose to insert a key, search for a key, display the hash table, or exit the program.
3. Depending on the user's choice, it calls the corresponding function (insert, search, display) or exits the loop.

CODE:

#include <stdio.h>

#include <stdlib.h>

// Hash function

int hash(int key, int size) {

    return key % size;

}

// Linear probing

int linearProbing(int key, int size, int hashTable[]) {

    int index = hash(key, size);

    int i = 0;

    while (hashTable[(index + i) % size] != -1) {

        i++;

        if (i == size) {

            return -1; // Hash table is full

        }

    }

    return (index + i) % size;

}

// Quadratic probing

int quadraticProbing(int key, int size, int hashTable[]) {

    int index = hash(key, size);

    int i = 0;

    while (hashTable[(index + i \* i) % size] != -1) {

        i++;

        if (i == size) {

            return -1; // Hash table is full

        }

    }

    return (index + i \* i) % size;

}

// Insert

void insert(int key, int size, int hashTable[], int technique) {

    int index;

    switch (technique) {

        case 1:

            index = linearProbing(key, size, hashTable);

            break;

        case 2:

            index = quadraticProbing(key, size, hashTable);

            break;

        default:

            printf("Invalid technique\n");

            return;

    }

    if (index == -1) {

        printf("Hash table is full, cannot insert key %d\n", key);

    } else {

        hashTable[index] = key;

    }

}

// Search

int search(int key, int size, int hashTable[], int technique) {

    int index = hash(key, size);

    int i = 0;

    switch (technique) {

        case 1:

            // Linear probing

            while (hashTable[(index + i) % size] != key) {

                if (hashTable[(index + i) % size] == -1 || i == size) {

                    return -1; // Key not found

                }

                i++;

            }

            return (index + i) % size;

        case 2:

            // Quadratic probing

            while (hashTable[(index + i \* i) % size] != key) {

                if (hashTable[(index + i \* i) % size] == -1 || i == size) {

                    return -1; // Key not found

                }

                i++;

            }

            return (index + i \* i) % size;

        default:

            printf("Invalid technique\n");

            return -1;

    }

}

void display(int size, int hashTable[]) {

    printf("\nHash Table:\n");

    for (int i = 0; i < size; i++) {

        if (hashTable[i] == -1) {

            printf("-\t");

        } else {

            printf("%d\t", hashTable[i]);

        }

    }

    printf("\n");

}

int main() {

    int size;

    printf("Enter the size of the hash table: ");

    scanf("%d", &size);

    int hashTable[size];

    for (int i = 0; i < size; i++) {

        hashTable[i] = -1;

    }

    int technique;

    printf("Enter the hashing technique to use:\n1. Linear Probing\n2. Quadratic Probing\nEnter your choice: ");

    scanf("%d", &technique);

    int choice;

    do {

        printf("\n1. Insert\n2. Search\n3. Display\n4. Exit\nEnter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1: {

                int key;

                int n;

                printf("Enter the number of keys to insert: ");

                scanf("%d", &n);

                for (int i = 0; i < n; i++) {

                    printf("Enter the key to be inserted: ");

                    scanf("%d", &key);

                    insert(key, size, hashTable, technique);

                }

                break;

            }

            case 2: {

                int key;

                printf("Enter the key to be searched: ");

                scanf("%d", &key);

                int index = search(key, size, hashTable, technique);

                if (index == -1) {

                    printf("Key not found\n");

                } else {

                    printf("Key found at index %d\n", index);

                }

                break;

            }

            case 3: {

                display(size, hashTable);

                break;

            }

            case 4: {

                printf("Exiting...\n");

                break;

            }

            default: {

                printf("Invalid choice\n");

                break;

            }

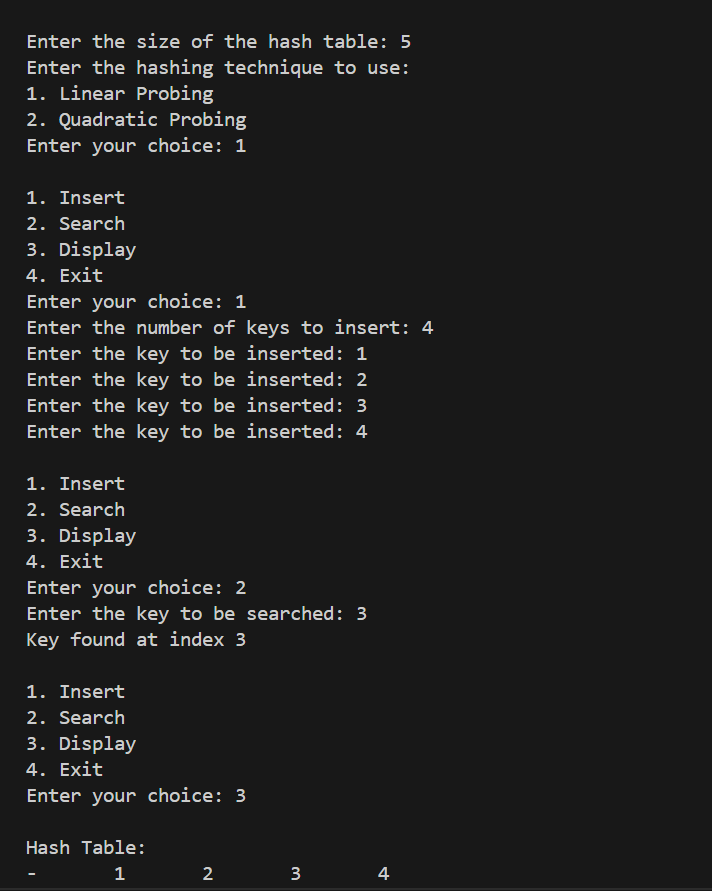
        }

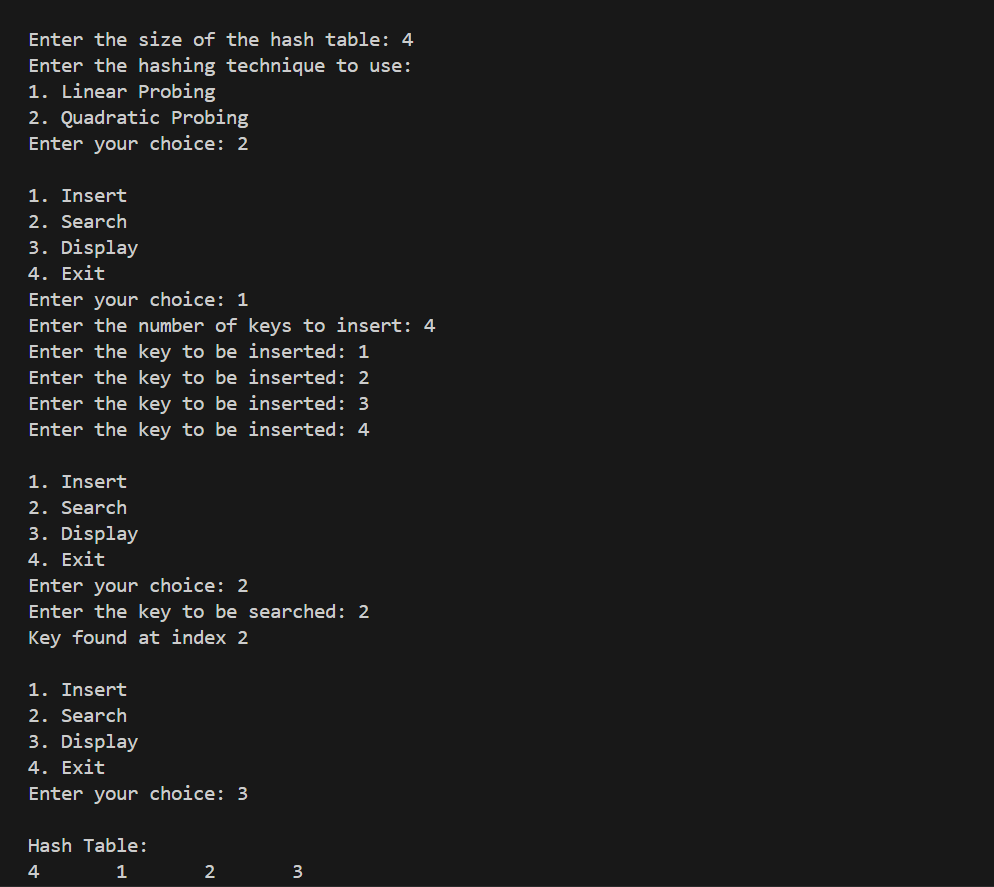
    } while (choice != 4);

    return 0;

}

**Output:**





**Conclusion:**

This code presents a practical implementation of a hash table with two different probing techniques: linear probing and quadratic probing. By employing these methods to handle collisions, the code ensures efficient storage and retrieval of key-value pairs in the hash table. The user interface allows for easy interaction, enabling users to insert keys, search for specific keys, and display the contents of the hash table. With clear prompts and error handling for invalid inputs, this code serves as a versatile tool for managing data using hash tables, offering flexibility in choosing the preferred probing technique based on specific requirements.