**1NAME** :- Manish Shashikant Jadhav

**UID** :- 2023301005.

**BRANCH** :- Comps -B. **BRANCH:** B.

**EXPERIMENT 9:** Implement Hashing using Quadratic Probing.

**SUBJECT** :- DS (DATA STRUCTURES).

**CODE** :-

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define TABLE\_SIZE 23

typedef struct KeyValue {

    char \*key;

    char \*value;

    bool isDeleted;

} KeyValue;

typedef struct {

    KeyValue \*\*array;

    int size;

    float load\_factor;

    int num\_keys;

    int num\_occupied\_indices;

    int num\_ops;

} HashTable;

KeyValue \*createKeyValue(char \*key, char \*value) {

    KeyValue \*newKeyValue = (KeyValue \*)malloc(sizeof(KeyValue));

    if (newKeyValue != NULL) {

        newKeyValue->key = strdup(key);

        newKeyValue->value = strdup(value);

        newKeyValue->isDeleted = false;

    }

    return newKeyValue;

}

HashTable \*createHashTable() {

    HashTable \*newTable = (HashTable \*)malloc(sizeof(HashTable));

    newTable->array = (KeyValue \*\*)malloc(TABLE\_SIZE \* sizeof(KeyValue \*));

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

        newTable->array[i] = NULL;

    newTable->size = TABLE\_SIZE;

    newTable->load\_factor = 0;

    newTable->num\_keys = 0;

    newTable->num\_occupied\_indices = 0;

    newTable->num\_ops = 0;

    return newTable;

}

int key\_to\_int(char \*key) {

    int sum = 0;

    int index = 0;

    while (\*key) {

        sum += (\*key) \* (index + 1);

        key++;

        index++;

    }

    return sum;

}

int hash\_function(char \*key, int size) {

    return key\_to\_int(key) % size;

}

int insert\_key\_value(HashTable \*ht, char \*key, char \*value) {

    if (ht == NULL || key == NULL || value == NULL)

        return -1;

    int index = hash\_function(key, ht->size);

    int original\_index = index;

    int i = 1; // Quadratic Probing Counter

    while (ht->array[index] != NULL) {

        if (strcmp(ht->array[index]->key, key) == 0 && !ht->array[index]->isDeleted)

            return -1; // Similar Key

        index = (original\_index + i \* i) % ht->size;

        i++;

        // Table Traversed Completely or Not

        if (index == original\_index)

            return -1; // Table Failed

    }

    // Insert the key-value pair

    ht->array[index] = createKeyValue(key, value);

    ht->num\_keys++;

    ht->num\_occupied\_indices++;

    ht->num\_ops++;

    return index;

}

char \*search\_key(HashTable \*ht, char \*key) {

    if (ht == NULL || key == NULL)

        return NULL;

    int index = hash\_function(key, ht->size);

    int original\_index = index;

    int i = 1; // Quadratic Probing Counter

    while (ht->array[index] != NULL) {

        if (strcmp(ht->array[index]->key, key) == 0 && !ht->array[index]->isDeleted) {

            return ht->array[index]->value;

        }

        index = (original\_index + i \* i) % ht->size;

        i++;

        // Table Traversed Completely or Not

        if (index == original\_index)

            break;

    }

    return NULL; // Key not found

}

int delete\_key(HashTable \*ht, char \*key) {

    if (ht == NULL || key == NULL)

        return -1;

    int index = hash\_function(key, ht->size);

    int original\_index = index;

    // Quadratic probing to handle collisions

    while (ht->array[index] != NULL) {

        if (strcmp(ht->array[index]->key, key) == 0 && !ht->array[index]->isDeleted) {

            // Mark it Deleted

            ht->array[index]->isDeleted = true;

            ht->num\_keys--;

            ht->num\_ops++;

            return index;

        }

        index = (index + 1) % ht->size;

        // Table Traversed Completely or Not

        if (index == original\_index)

            break;

    }

    return -1; // Key not found

}

int get\_load\_factor(HashTable \*ht) {

    if (ht == NULL || ht->size == 0)

        return -1;

    return (float)ht->num\_keys / ht->size;

}

int get\_avg\_probes(HashTable \*ht) {

    if (ht == NULL || ht->num\_ops == 0)

        return -1;

    return ht->num\_ops / ht->num\_keys;

}

void display(HashTable \*ht) {

    if (ht == NULL)

        return;

    printf("Hash Table:\n");

    printf("| %-10s | %-15s | %-15s |\n", "Index", "Key", "Value");

    printf("|------------|-----------------|-----------------|\n");

    for (int i = 0; i < ht->size; i++) {

        printf("| %-10d |", i);

        if (ht->array[i] != NULL) {

            if (ht->array[i]->isDeleted) {

                printf(" %-15s | %-15s |", "(Deleted)", "(Deleted)");

            } else {

                printf(" %-15s | %-15s |", ht->array[i]->key, ht->array[i]->value);

            }

        } else {

            printf(" %-15s | %-15s |", "(Empty)", "(Empty)");

        }

        printf("\n");

    }

}

int main() {

    HashTable \*ht = createHashTable();

    // Insert key-value pairs

    insert\_key\_value(ht, "first name", "Manish");

    insert\_key\_value(ht, "last name", "Jadhav");

    insert\_key\_value(ht, "uid", "2023301005");

    insert\_key\_value(ht, "sport", "Cricket");

    insert\_key\_value(ht, "food", "Burger");

    insert\_key\_value(ht, "holiday", "Maldives");

    insert\_key\_value(ht, "role\_model", "Chhatrapati Shivaji Maharaj");

    insert\_key\_value(ht, "subject", "Python");

    insert\_key\_value(ht, "song", "Aarambh");

    insert\_key\_value(ht, "movie", "Farjand");

    insert\_key\_value(ht, "colour", "Orange");

    insert\_key\_value(ht, "book", "The Hidden Hindu");

    // Test search and delete operations

    char \*search\_result = search\_key(ht, "sport");

    if (search\_result != NULL) {

        printf("> Search Result for 'sport': %s\n", search\_result);

    } else {

        printf("> Key 'sport' not found\n");

    }

    int delete\_result = delete\_key(ht, "holiday");

    if (delete\_result != -1) {

        printf("> Deleted key 'holiday' at index [%d]\n", delete\_result);

    } else {

        printf("> Key 'holiday' not found for deletion\n");

    }

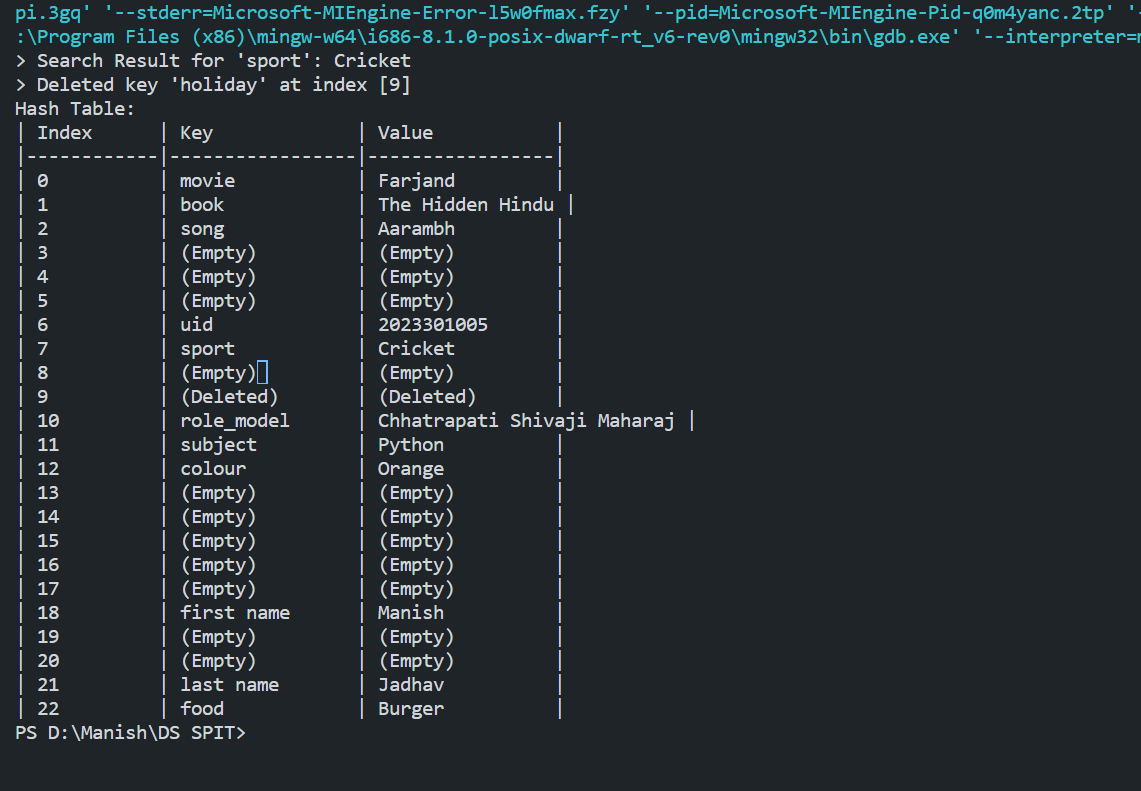
    // Display the hash table

    display(ht);

    return 0;

}

**Output:**

****

**Algorithm:**

**1. KeyValue Structure:**

- Contains a key (string), value (string), and a boolean flag to indicate if the entry has been deleted.

**2. HashTable Structure:**

- An array of pointers to KeyValue structures.

- Size of the array (`size`), load factor, number of keys (`num\_keys`), number of occupied indices (`num\_occupied\_indices`), and number of operations (`num\_ops`) are tracked.

**Functions:**

**1. createKeyValue:**

- Dynamically allocates memory for a new KeyValue structure and initializes its fields.

**2. createHashTable:**

- Dynamically allocates memory for a new HashTable structure and initializes its fields.

- Initializes the array of KeyValue pointers to `NULL`.

**3. key\_to\_int:**

- Converts a string key to an integer value.

**4. hash\_function:**

- Uses the key\_to\_int function to calculate the hash index for a given key.

**5. insert\_key\_value:**

- Inserts a key-value pair into the hash table using quadratic probing for collision resolution.

**6. search\_key:**

- Searches for a key in the hash table and returns its corresponding value.

**7. delete\_key:**

- Deletes a key from the hash table and marks it as deleted.

**8. get\_load\_factor:**

- Calculates and returns the load factor of the hash table.

**9. get\_avg\_probes:**

- Calculates and returns the average number of probes per operation.

**10. display:**

- Displays the contents of the hash table

**11. Main Function:**

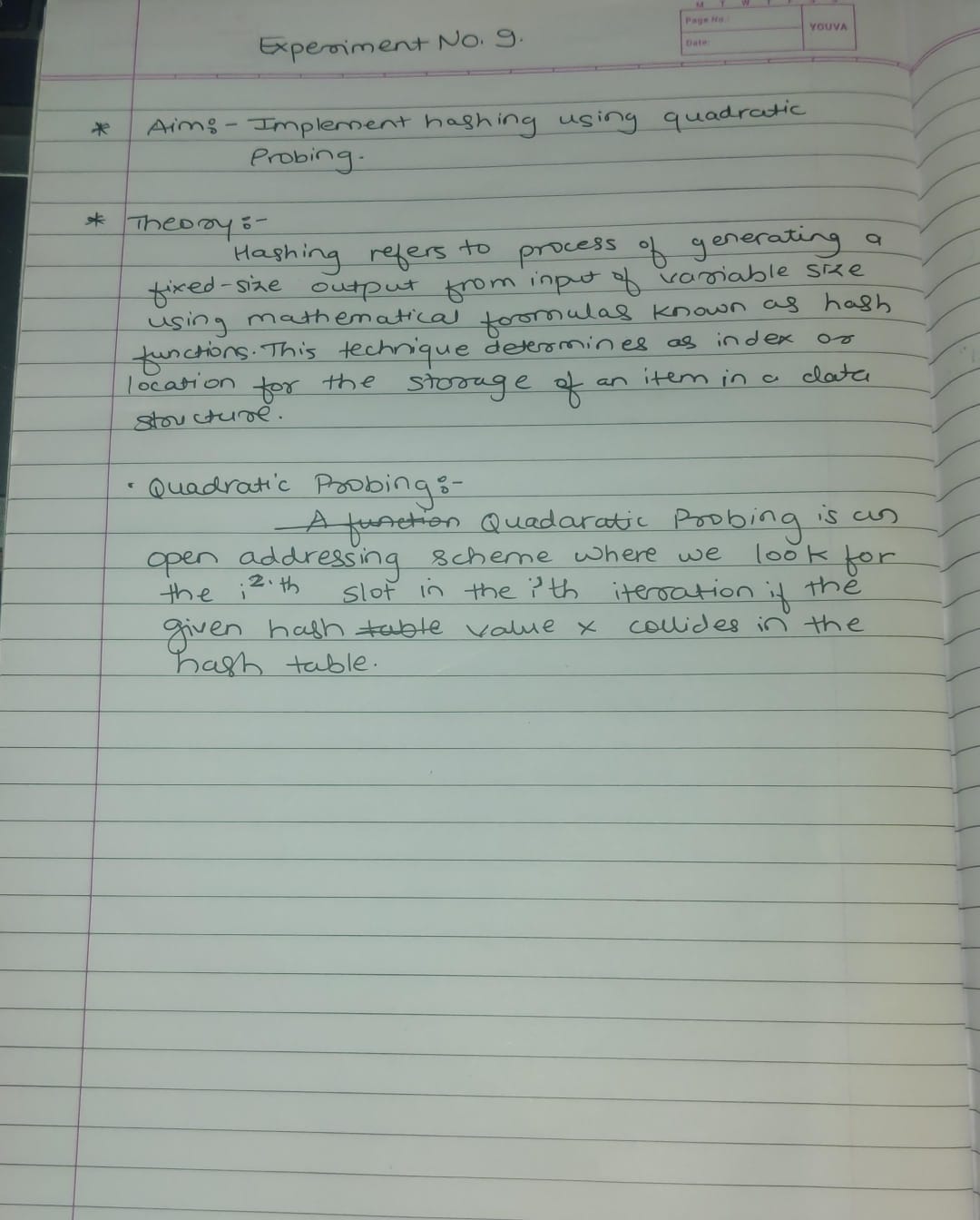
- Creates a hash table using `createHashTable`.

- Inserts several key-value pairs into the hash table using `insert\_key\_value`.

- Searches for the value of a specific key using `search\_key`.

- Deletes a specific key using `delete\_key`.

- Displays the contents of the hash table using `display`.



**Conclusion:**

Hence, by completing this experiment I came to know about implementing Hashing using Quadratic Probing.