**NAME :** MIRUTHULA.B

**REG NO** 231501100

**EX NO16** IMPLEMENTATION OF COLLISION RESOLUTION TECHNIQUE

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define SIZE 10 // Size of the hash table

// Structure to represent a node in the hash table

struct Node {

int key;

int value;

};

// Structure to represent the hash table

struct HashTable {

struct Node\* array[SIZE];

};

// Function to create a new node

struct Node\* createNode(int key, int value) {

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

newNode->key = key;

newNode->value = value;

return newNode;

}

// Function to create a hash table

struct HashTable\* createHashTable() {

struct HashTable\* hashTable = (struct HashTable\*)malloc(sizeof(struct HashTable));

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

hashTable->array[i] = NULL;

}

return hashTable;

}

// Function to calculate the hash index

int hash(int key) {

return key % SIZE;

}

// Function to perform open addressing (linear probing) for collision resolution

void linearProbing(struct HashTable\* hashTable, int key, int value) {

int index = hash(key);

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

index = (index + 1) % SIZE; // Linear probing

}

hashTable->array[index] = createNode(key, value);

}

// Function to perform closed addressing (chaining) for collision resolution

void chaining(struct HashTable\* hashTable, int key, int value) {

int index = hash(key);

struct Node\* newNode = createNode(key, value);

if (hashTable->array[index] == NULL) {

hashTable->array[index] = newNode;

} else {

// Adding to the end of the linked list at the index

struct Node\* temp = hashTable->array[index];

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

}

// Function to display the hash table

void display(struct HashTable\* hashTable) {

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

printf("%d: ", i);

struct Node\* temp = hashTable->array[i];

while (temp != NULL) {

printf("(%d, %d) ", temp->key, temp->value);

temp = temp->next;

}

printf("\n");

}

}

int main() {

struct HashTable\* hashTable\_linear = createHashTable();

struct HashTable\* hashTable\_chaining = createHashTable();

// Inserting elements using linear probing

linearProbing(hashTable\_linear, 10, 20);

linearProbing(hashTable\_linear, 21, 30);

linearProbing(hashTable\_linear, 22, 40);

linearProbing(hashTable\_linear, 23, 50);

linearProbing(hashTable\_linear, 33, 60);

// Inserting elements using chaining

chaining(hashTable\_chaining, 10, 20);

chaining(hashTable\_chaining, 21, 30);

chaining(hashTable\_chaining, 22, 40);

chaining(hashTable\_chaining, 23, 50);

chaining(hashTable\_chaining, 33, 60);

printf("Hash Table with Linear Probing:\n");

display(hashTable\_linear);

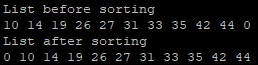
printf("\nHash Table with Chaining:\n");

display(hashTable\_chaining);

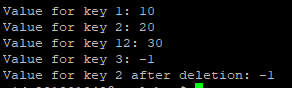
return 0;

}

OUTPUT:



OUTPUT 2:



OUTPUT 3:

