**Write a C program to create a hash table and perform collision resolution using the following techniques.**

1. **Open addressing**
2. **Closed Addressing**
3. **Rehashing**

**Algorithm:**

**CODE:**

**i)Open Addressing:**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <stdbool.h>**

**#define SIZE 10**

**struct HashNode {**

**int key;**

**int value;**

**bool occupied;**

**};**

**struct HashNode\* createHashNode(int key, int value) {**

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

**newNode->key = key;**

**newNode->value = value;**

**newNode->occupied = true;**

**return newNode;**

**}**

**struct HashNode\*\* createHashTable() {**

**struct HashNode\*\* table = (struct HashNode\*\*)malloc(SIZE \* sizeof(struct HashNode\*));**

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

**table[i] = NULL;**

**}**

**return table;**

**}**

**int hashCode(int key) {**

**return key % SIZE;**

**}**

**void insertOpenAddressing(struct HashNode\*\* table, int key, int value) {**

**int index = hashCode(key);**

**while (table[index] != NULL && table[index]->occupied) {**

**index = (index + 1) % SIZE;**

**}**

**table[index] = createHashNode(key, value);**

**}**

**void displayHashTable(struct HashNode\*\* table) {**

**printf("Hash Table:\n");**

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

**if (table[i] != NULL && table[i]->occupied) {**

**printf("Index %d: Key = %d, Value = %d\n", i, table[i]->key, table[i]->value);**

**} else {**

**printf("Index %d: Empty\n", i);**

**}**

**}**

**}**

**int main() {**

**struct HashNode\*\* hashTable = createHashTable();**

**insertOpenAddressing(hashTable, 5, 10);**

**insertOpenAddressing(hashTable, 15, 20);**

**insertOpenAddressing(hashTable, 25, 30);**

**insertOpenAddressing(hashTable, 35, 40);**

**insertOpenAddressing(hashTable, 45, 50);**

**insertOpenAddressing(hashTable, 55, 60);**

**insertOpenAddressing(hashTable, 65, 70);**

**insertOpenAddressing(hashTable, 75, 80);**

**insertOpenAddressing(hashTable, 85, 90);**

**insertOpenAddressing(hashTable, 95, 100);**

**displayHashTable(hashTable);**

**return 0;**

**}**

**2. Closed Addressing:**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define SIZE 10**

**struct HashNode {**

**int key;**

**int value;**

**struct HashNode\* next;**

**};**

**struct HashNode\* createHashNode(int key, int value) {**

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

**newNode->key = key;**

**newNode->value = value;**

**newNode->next = NULL;**

**return newNode;**

**}**

**struct HashNode\*\* createHashTable() {**

**struct HashNode\*\* table = (struct HashNode\*\*)malloc(SIZE \* sizeof(struct HashNode\*));**

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

**table[i] = NULL;**

**}**

**return table;**

**}**

**int hashCode(int key) {**

**return key % SIZE;**

**}**

**void insertClosedAddressing(struct HashNode\*\* table, int key, int value) {**

**int index = hashCode(key);**

**if (table[index] == NULL) {**

**table[index] = createHashNode(key, value);**

**} else {**

**struct HashNode\* newNode = createHashNode(key, value);**

**newNode->next = table[index];**

**table[index] = newNode;**

**}**

**}**

**void displayHashTable(struct HashNode\*\* table) {**

**printf("Hash Table:\n");**

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

**struct HashNode\* current = table[i];**

**printf("Index %d: ", i);**

**while (current != NULL) {**

**printf("Key = %d, Value = %d -> ", current->key, current->value);**

**current = current->next;**

**}**

**printf("NULL\n");**

**}**

**}**

**int main() {**

**struct HashNode\*\* hashTable = createHashTable();**

**insertClosedAddressing(hashTable, 5, 10);**

**insertClosedAddressing(hashTable, 15, 20);**

**insertClosedAddressing(hashTable, 25, 30);**

**insertClosedAddressing(hashTable, 35, 40);**

**insertClosedAddressing(hashTable, 45, 50);**

**insertClosedAddressing(hashTable, 55, 60);**

**insertClosedAddressing(hashTable, 65, 70);**

**insertClosedAddressing(hashTable, 75, 80);**

**insertClosedAddressing(hashTable, 85, 90);**

**insertClosedAddressing(hashTable, 95, 100);**

**displayHashTable(hashTable);**

**return 0;**

**}**

**3. Rehashing:**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <stdbool.h>**

**#define SIZE 10**

**#define LOAD\_FACTOR\_THRESHOLD 0.7**

**struct HashNode {**

**int key;**

**int value;**

**bool occupied;**

**};**

**struct HashNode\* createHashNode(int key, int value) {**

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

**newNode->key = key;**

**newNode->value = value;**

**newNode->occupied = true;**

**return newNode;**

**}**

**struct HashNode\*\* createHashTable(int size) {**

**struct HashNode\*\* table = (struct HashNode\*\*)malloc(size \* sizeof(struct HashNode\*));**

**for (int i = 0; i < size; i++) {**

**table[i] = NULL;**

**}**

**return table;**

**}**

**int hashCode(int key, int size) {**

**return key % size;**

**}**

**void insertOpenAddressing(struct HashNode\*\* table, int key**

**, int value, int size) {**

**int index = hashCode(key, size);**

**while (table[index] != NULL && table[index]->occupied) {**

**index = (index + 1) % size;**

**}**

**table[index] = createHashNode(key, value);**

**}**

**void displayHashTable(struct HashNode\*\* table, int size) {**

**printf("Hash Table:\n");**

**for (int i = 0; i < size; i++) {**

**if (table[i] != NULL && table[i]->occupied) {**

**printf("Index %d: Key = %d, Value = %d\n", i, table[i]->key, table[i]->value);**

**} else {**

**printf("Index %d: Empty\n", i);**

**}**

**}**

**}**

**double getLoadFactor(int count, int size) {**

**return (double)count / size;**

**}**

**void rehash(struct HashNode\*\* oldTable, int oldSize, struct HashNode\*\* newTable, int newSize) {**

**for (int i = 0; i < oldSize; i++) {**

**if (oldTable[i] != NULL && oldTable[i]->occupied) {**

**int index = oldTable[i]->key % newSize;**

**while (newTable[index] != NULL && newTable[index]->occupied) {**

**index = (index + 1) % newSize;**

**}**

**newTable[index] = oldTable[i];**

**}**

**}**

**free(oldTable);**

**}**

**int main() {**

**int initialSize = SIZE;**

**struct HashNode\*\* hashTable = createHashTable(initialSize);**

**int count = 0;**

**insertOpenAddressing(hashTable, 5, 10, initialSize);**

**insertOpenAddressing(hashTable, 15, 20, initialSize);**

**insertOpenAddressing(hashTable, 25, 30, initialSize);**

**insertOpenAddressing(hashTable, 35, 40, initialSize);**

**insertOpenAddressing(hashTable, 45, 50, initialSize);**

**insertOpenAddressing(hashTable, 55, 60, initialSize);**

**insertOpenAddressing(hashTable, 65, 70, initialSize);**

**insertOpenAddressing(hashTable, 75, 80, initialSize);**

**insertOpenAddressing(hashTable, 85, 90, initialSize);**

**insertOpenAddressing(hashTable, 95, 100, initialSize);**

**count = 10;**

**double loadFactor = getLoadFactor(count, initialSize);**

**if (loadFactor > LOAD\_FACTOR\_THRESHOLD) {**

**int newSize = initialSize \* 2; // Double the size**

**struct HashNode\*\* newTable = createHashTable(newSize);**

**rehash(hashTable, initialSize, newTable, newSize);**

**hashTable = newTable;**

**initialSize = newSize;**

**}**

**displayHashTable(hashTable, initialSize);**

**return 0;**

**}**

**OUTPUT:**

**1)insert**

**2)search**

**3)exit**

**Enter your choice: 1**

**Enter key:1**

**Enter value:10**

**Inserted key-value pair(1,10)**

**1)insert**

**2)search**

**3)exit**

**Enter your choice:2**

**Enter key to search:1**

**Value for key1:10**