**DATASTRUCTURE**

**PROGRAMS:**

**1.Separate Chaining**

*#include <stdio.h>*

*#include <stdlib.h>*

*typedef struct Node {*

*int key;*

*int value;*

*struct Node\* next;*

*} Node;*

*typedef struct HashTable {*

*Node\*\* buckets;*

*int size;*

*} HashTable;*

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

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

*newNode->key = key;*

*newNode->value = value;*

*newNode->next = NULL;*

*return newNode;*

*}*

*HashTable\* createHashTable(int size) {*

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

*table->size = size;*

*table->buckets = (Node\*\*) malloc(size \* sizeof(Node\*));*

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

*table->buckets[i] = NULL;*

*}*

*return table;*

*}*

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

*return key % size;*

*}*

*void insert(HashTable\* table, int key, int value) {*

*int index = hash(key, table->size);*

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

*if (table->buckets[index] == NULL) {*

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

*} else {*

*Node\* temp = table->buckets[index];*

*while (temp->next != NULL) {*

*temp = temp->next;*

*}*

*temp->next = newNode;*

*}*

*}*

*int search(HashTable\* table, int key) {*

*int index = hash(key, table->size);*

*Node\* temp = table->buckets[index];*

*while (temp != NULL) {*

*if (temp->key == key) {*

*return temp->value;*

*}*

*temp = temp->next;*

*}*

*return -1; // Key not found*

*}*

*void delete(HashTable\* table, int key) {*

*int index = hash(key, table->size);*

*Node\* temp = table->buckets[index];*

*Node\* prev = NULL;*

*while (temp != NULL && temp->key != key) {*

*prev = temp;*

*temp = temp->next;*

*}*

*if (temp == NULL) {*

*printf("Key not found\n");*

*return;*

*}*

*if (prev == NULL) {*

*table->buckets[index] = temp->next;*

*} else {*

*prev->next = temp->next;*

*}*

*free(temp);*

*}*

*int main() {*

*HashTable\* table = createHashTable(10);*

*insert(table, 1, 10);*

*insert(table, 2, 20);*

*insert(table, 11, 110);*

*insert(table, 21, 210);*

*printf("Value for key 1: %d\n", search(table, 1));*

*printf("Value for key 2: %d\n", search(table, 2));*

*printf("Value for key 11: %d\n", search(table, 11));*

*printf("Value for key 21: %d\n", search(table, 21));*

*delete(table, 11);*

*printf("Value for key 11 after deletion: %d\n", search(table, 11));*

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

*Node\* temp = table->buckets[i];*

*while (temp != NULL) {*

*Node\* toFree = temp;*

*temp = temp->next;*

*free(toFree);*

*}*

*}*

*free(table->buckets);*

*free(table);*

*return 0;*

*}*

**OUTPUT:**

*Value for key 1: 10*

*Value for key 2: 20*

*Value for key 11: 110*

*Value for key 21: 210*

*Value for key 11 after deletion: -1*

**2.Linear Probing**

*#include <stdio.h>*

*#include <stdlib.h>*

*typedef struct {*

*int key;*

*int value;*

*int isOccupied; // 0: empty, 1: occupied, 2: deleted*

*} HashEntry;*

*typedef struct {*

*HashEntry\* table;*

*int size;*

*} HashTable;*

*HashTable\* createHashTable(int size) {*

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

*hashTable->size = size;*

*hashTable->table = (HashEntry\*) malloc(size \* sizeof(HashEntry));*

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

*hashTable->table[i].isOccupied = 0;*

*}*

*return hashTable;*

*}*

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

*return key % size;*

*}*

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

*int index = hash(key, hashTable->size);*

*int originalIndex = index;*

*while (hashTable->table[index].isOccupied == 1) {*

*index = (index + 1) % hashTable->size;*

*if (index == originalIndex) {*

*printf("Hash table is full\n");*

*return;*

*}*

*}*

*hashTable->table[index].key = key;*

*hashTable->table[index].value = value;*

*hashTable->table[index].isOccupied = 1;*

*}*

*int search(HashTable\* hashTable, int key) {*

*int index = hash(key, hashTable->size);*

*int originalIndex = index;*

*while (hashTable->table[index].isOccupied != 0) {*

*if (hashTable->table[index].isOccupied == 1 && hashTable->table[index].key == key) {*

*return hashTable->table[index].value;*

*}*

*index = (index + 1) % hashTable->size;*

*if (index == originalIndex) {*

*break;*

*}*

*}*

*return -1;*

*}*

*void delete(HashTable\* hashTable, int key) {*

*int index = hash(key, hashTable->size);*

*int originalIndex = index;*

*while (hashTable->table[index].isOccupied != 0) {*

*if (hashTable->table[index].isOccupied == 1 && hashTable->table[index].key == key) {*

*hashTable->table[index].isOccupied = 2;*

*return;*

*}*

*index = (index + 1) % hashTable->size;*

*if (index == originalIndex) {*

*break;*

*}*

*}*

*printf("Key not found\n");*

*}*

*int main() {*

*HashTable\* hashTable = createHashTable(10);*

*insert(hashTable, 1, 10);*

*insert(hashTable, 2, 20);*

*insert(hashTable, 11, 110);*

*insert(hashTable, 21, 210);*

*printf("Value for key 1: %d\n", search(hashTable, 1));*

*printf("Value for key 2: %d\n", search(hashTable, 2));*

*printf("Value for key 11: %d\n", search(hashTable, 11));*

*printf("Value for key 21: %d\n", search(hashTable, 21));*

*delete(hashTable, 11);*

*printf("Value for key 11 after deletion: %d\n", search(hashTable, 11));*

*// Free memory*

*free(hashTable->table);*

*free(hashTable);*

*return 0;*

*}*

**OUTPUT:**

*Value for key 1: 10*

*Value for key 2: 20*

*Value for key 11: 110*

*Value for key 21: 210*

*Value for key 11 after deletion: -1*