**K230607 BCS-3F 28 Nov,2024**

**TASK 01**

#include <iostream>

using namespace std;

struct Node {

int key;

Node\* next;

Node(int k) : key(k), next(nullptr) {}

};

class HashTable {

private:

static const int TABLE\_SIZE = 10;

Node\* table[TABLE\_SIZE];

int hashFunction(int key) {

return key % TABLE\_SIZE;

}

public:

HashTable() {

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

table[i] = nullptr;

}

}

void insert(int key) {

int index = hashFunction(key);

Node\* newNode = new Node(key);

newNode->next = table[index];

table[index] = newNode;

cout << "Inserted key " << key << " at index " << index << endl;

}

bool search(int key) {

int index = hashFunction(key);

Node\* current = table[index];

while (current) {

if (current->key == key) {

return true;

}

current = current->next;

}

return false;

}

void remove(int key) {

int index = hashFunction(key);

Node\* current = table[index];

Node\* prev = nullptr;

while (current && current->key != key) {

prev = current;

current = current->next;

}

if (!current) {

cout << "Key " << key << " not found!" << endl;

return;

}

if (!prev) {

table[index] = current->next;

} else {

prev->next = current->next;

}

delete current;

cout << "Key " << key << " deleted!" << endl;

}

void display() {

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

cout << "Index " << i << ": ";

Node\* current = table[i];

while (current) {

cout << current->key << " -> ";

current = current->next;

}

cout << "NULL" << endl;

}

}

};

int main() {

HashTable ht;

ht.insert(20);

ht.insert(34);

ht.insert(45);

ht.insert(70);

ht.insert(56);

cout << "Hash Table:" << endl;

ht.display();

cout << "Search for key 45: " << (ht.search(45) ? "Found" : "Not Found") << endl;

cout << "Search for key 99: " << (ht.search(99) ? "Found" : "Not Found") << endl;

ht.remove(34);

ht.remove(99);

cout << "Hash Table after deletions:" << endl;

ht.display();

return 0;

}

A screenshot of a computer program

Description automatically generated

**TASK 02**

#include <iostream>

using namespace std;

int countPairsWithSum(int arr[], int N, int K) {

int hashTable[10000]= {0};

int count= 0;

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

int complement= K - arr[i];

if (complement >= 0 && hashTable[complement] > 0) {

count += hashTable[complement];

}

hashTable[arr[i]]++;

}

return count;

}

int main() {

int N= 4, K= 6;

int arr[]= {1, 5, 7, 1};

cout<<"Number of pairs with sum "<<K<<" is: " <<countPairsWithSum(arr, N, K)<<endl;

return 0;

}

A close up of a sign

Description automatically generated

**TASK 03**

#include <iostream>

using namespace std;

int hasTripletWithZeroSum(int arr[], int n) {

for (int i= 0; i < n - 1; i++) {

bool hashTable[10000]= {0};

for (int j= i + 1; j < n; j++) {

int x= -(arr[i] + arr[j]);

if (x >= 0 && hashTable[x]) return 1;

hashTable[arr[j]]= true;

}

}

return 0;

}

int main() {

int n= 5;

int arr[]= {0, -1, 2, -3, 1};

cout << hasTripletWithZeroSum(arr, n) << endl;

return 0;

}

**TASK 04**

#include <iostream>

using namespace std;

struct HashNode {

char key;

bool isPresent;

HashNode() : key(0), isPresent(false) {}

};

class CustomHashTable {

private:

static const int TABLE\_SIZE= 128;

HashNode table[TABLE\_SIZE];

int hashFunction(char key) {

return key % TABLE\_SIZE;

}

public:

void insert(char key) {

int index= hashFunction(key);

while (table[index].isPresent) {

index= (index + 1) % TABLE\_SIZE;

}

table[index].key= key;

table[index].isPresent= true;

}

bool contains(char key) {

int index= hashFunction(key);

int startIndex= index;

while (table[index].isPresent) {

if (table[index].key== key) return true;

index= (index + 1) % TABLE\_SIZE;

if (index== startIndex) break;

}

return false;

}

};

void matchNutsAndBolts(char nuts[], char bolts[], int N) {

char order[]= {'#', '$', '%', '&', '\*', '@', '^', '~'};

CustomHashTable hashTable;

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

hashTable.insert(nuts[i]);

}

int index= 0;

for (int i= 0; i < 8; i++) {

if (hashTable.contains(order[i])) {

nuts[index]= order[i];

bolts[index]= order[i];

index++;

}

}

}

int main() {

int N= 5;

char nuts[]= {'@', '%', '$', '#', '^'};

char bolts[]= {'%', '@', '#', '$', '^'};

matchNutsAndBolts(nuts, bolts, N);

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

cout<<nuts[i]<<" ";

}

cout<<endl;

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

cout<<bolts[i]<<" ";

}

cout<<endl;

return 0;

}

A group of symbols on a dark background

Description automatically generated