**GROUP - A**

**Consider telephone book database of N clients . Make use of a hash table implement to quickly look up client telephone number make use of two collison handling techniques and compare them using number of comparison requird to find a set of telephone number**

#include <iostream>

#include <string>

#include <list>

#include <vector>

using namespace std;

class Client {

public:

string name;

string phoneNumber;

Client(const string& name, const string& phoneNumber)

: name(name), phoneNumber(phoneNumber) {}

};

class HashTable {

private:

static const int TABLE\_SIZE = 100;

vector<list<Client>> table; // Using a vector of lists for chaining

vector<Client\*> linearProbingTable; // Using a vector for linear probing

vector<bool> linearProbingTableFlags; // Flags to indicate if a slot is occupied

public:

HashTable() {

table.resize(TABLE\_SIZE);

linearProbingTable.resize(TABLE\_SIZE, nullptr);

linearProbingTableFlags.resize(TABLE\_SIZE, false);

}

~HashTable() {

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

delete linearProbingTable[i];

}

}

int hashFunction(const string& key) {

int sum = 0;

for (char ch : key) {

sum += ch;

}

return sum % TABLE\_SIZE;

}

void insertChaining(const string& name, const string& phoneNumber) {

int index = hashFunction(name);

table[index].push\_back(Client(name, phoneNumber));

}

void insertLinearProbing(const string& name, const string& phoneNumber) {

int index = hashFunction(name);

int i = index;

bool inserted = false;

while (!inserted) {

if (!linearProbingTableFlags[i]) {

linearProbingTable[i] = new Client(name, phoneNumber);

linearProbingTableFlags[i] = true;

inserted = true;

}

i = (i + 1) % TABLE\_SIZE; // Linear probing

if (i == index) {

cerr << "Hash table is full!" << endl;

return;

}

}

}

int findChaining(const string& name) {

int index = hashFunction(name);

int comparisons = 0;

for (const Client& client : table[index]) {

comparisons++;

if (client.name == name) {

return comparisons;

}

}

return comparisons;

}

int findLinearProbing(const string& name) {

int index = hashFunction(name);

int i = index;

int comparisons = 0;

while (linearProbingTableFlags[i]) {

comparisons++;

if (linearProbingTable[i]->name == name) {

return comparisons;

}

i = (i + 1) % TABLE\_SIZE; // Linear probing

if (i == index) {

break;

}

}

return comparisons;

}

};

int main() {

HashTable phoneBook;

// Inserting clients' telephone numbers using chaining

phoneBook.insertChaining("John Doe", "1234567890");

phoneBook.insertChaining("Jane Smith", "9876543210");

phoneBook.insertChaining("Alice Johnson", "5678901234");

// Looking up telephone numbers and comparing the collision handling techniques

cout << "Comparison of Collision Handling Techniques:" << endl;

cout << "Name\t\tChaining\tLinear Probing" << endl;

cout << "----------------------------------------------" << endl;

// Set of names to search

vector<string> names = {"John Doe", "Jane Smith", "Alice Johnson", "Bob Brown"};

// Perform lookups and print the number of comparisons

for (const string& name : names) {

int chainingComparisons = phoneBook.findChaining(name);

int linearProbingComparisons = phoneBook.findLinearProbing(name);

cout << name << "\t\t" << chainingComparisons << "\t\t" << linearProbingComparisons << endl;

}

return 0;

}

OUTPUT : -

Comparison of Collision Handling Techniques:

Name Chaining Linear Probing

----------------------------------------------

John Doe 1 0

Jane Smith 1 0

Alice Johnson 1 0

Bob Brown 0 0

--------------------------------