/\* Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client‘s telephone number. Make use of two collision handling techniques and compare them using number of comparisons required to find a set of telephone numbers \*/

#include <iostream>

#define MAX 10

using namespace std;

struct node

{

string name;

long long int telephone\_number;

};

class hash\_functions

{

private:

node data[MAX];

public:

hash\_functions()

{

for (int i = 0; i < MAX; i++)

{

data[i].name = " ";

data[i].telephone\_number = 0;

}

}

void insert();

int hash(string client\_name);

void search();

void display();

};

int main()

{

std::cout << "\n--------------------------------------------------" << std::endl;

std::cout << "--------------------------------------------------" << std::endl;

hash\_functions obj;

int choice;

do

{

std::cout << "1.Insert" << std::endl;

std::cout << "2.Search" << std::endl;

std::cout << "3.Display" << std::endl;

std::cout << "4.Exit" << std::endl;

std::cout << "--------------------------------------------------" << std::endl;

std::cout << "Enter your choice: ";

cin >> choice;

std::cout << "--------------------------------------------------" << std::endl;

switch (choice)

{

case 1:

obj.insert();

break;

case 2:

obj.search();

break;

case 3:

obj.display();

break;

case 4:

std::cout << "Thank You" << std::endl;

break;

default:

std::cout << "Invalid Choice" << std::endl;

std::cout << "--------------------------------------------------" << std::endl;

break;

}

} while (choice != 4);

std::cout << "--------------------------------------------------" << std::endl;

std::cout << "--------------------------------------------------" << std::endl;

return 0;

}

void hash\_functions::insert()

{

int pos;

string client\_name;

long int telephone\_number;

std::cout << "Enter the name of client: ";

cin >> client\_name;

std::cout << "Enter the client telephone number: ";

cin >> telephone\_number;

pos = hash(client\_name);

int c = pos;

bool flag = false;

while (flag != true)

{

if (data[pos].name == " " && data[pos].telephone\_number == 0)

{

data[pos].name = client\_name;

data[pos].telephone\_number = telephone\_number;

flag = true;

cout << "Hash is: " << pos << endl;

break;

}

pos = (pos + 1) % 10;

if (c == pos)

{

std::cout << "Hash Table is Full." << std::endl;

break;

}

}

std::cout << "--------------------------------------------------" << std::endl;

}

int hash\_functions::hash(string client\_name)

{

int sum = 0;

for (int i = 0; i < client\_name.length(); i++)

{

sum += (int)client\_name[i];

}

return (sum / client\_name.length()) % 10;

}

void hash\_functions::search()

{

int pos;

string client\_name;

std::cout << "Enter client name to be searched: ";

cin >> client\_name;

pos = hash(client\_name);

int c = pos;

bool flag = false;

while (flag != true)

{

if (data[pos].name == client\_name)

{

std::cout << "Data Found. Telephone number is: " << data[pos].telephone\_number << std::endl;

flag = true;

break;

}

pos = (pos + 1) % 10;

if (c == pos)

{

std::cout << "Data Not Found" << std::endl;

break;

}

}

std::cout << "--------------------------------------------------" << std::endl;

}

void hash\_functions::display()

{

for (int i = 0; i < MAX; i++)

{

std::cout << i << ". Name: " << data[i].name << "\n Telephone\_number: " << data[i].telephone\_number << std::endl;

}

std::cout << "--------------------------------------------------" << std::endl;

}

/\*Hash tables are an efficient implementation of a keyed array data structure, a structure sometimes known as an associative array or map. If you're working in C++, you can take advantage of the STL map container for keyed arrays implemented using binary trees, but this article will give you some of the theory behind how a hash table works.

Hashing Functions

A hashing function can be just about anything. How the hashing function is actually coded depends on the situation, but generally the hashing function should return a value based on a key and the size of the array the hashing table is built on. Also, one important thing that is sometimes overlooked is that a hashing function has to return the same value every time it is given the same key.

Basic Operations

Following are the basic primary operations of a hash table.

Search − Searches an element in a hash table.

Insert − inserts an element in a hash table.

delete− Deletes an element from a hash table.\*/