```
// Lina Kang
// CS1D MW 2:30 - 5:00 PM
// Assignment 7 - Hashing
// This program exercises the hashing algorithms using a map

/* OUTPUT
Lina Kang
CS1D MW 2:30-5:00PM
Assignment 7 - Hashing
This program exercises the hashing algorithms using a map
Data Structure Used: map
--------------------------
Part One:
--------------------------
0 ---> ( 29, Dana Point )
1 ---> ( 88, El Segundo )
2 ---> ( 31, Los Angeles )
3 ---> ( 32, San Diego )
4 ---> ( 62, Laguna Hills )
5
6 ---> ( 35, Tustin )
7 ---> ( 60, Santa Ana )
8
9
10
11 ---> ( 11, La Jolla )
12 ---> ( 99, San Juan )
13 ---> ( 42, Vista )
14
15 ---> ( 44, Irvine )
16 ---> ( 103, Oceanside )
17 ---> ( 17, Orange )
18 ---> ( 18, Del Mar )
19 ---> ( 19, Brea )
20 ---> ( 49, San Diego )
21
22 ---> ( 22, Aliso Viejo )
23 ---> ( 41, San Clemente )
24
25
26
27
28
--------------------------
Part Two:
--------------------------
0 ---> ( 31, Los Angeles )
1 ---> ( 32, San Diego )
2
3
4 ---> ( 35, Tustin )
5
6 ---> ( 99, San Juan )
7
8
9 ---> ( 62, Laguna Hills )
10 ---> ( 103, Oceanside )
```

```
11 ---> ( 42, Vista )
12 ---> ( 11, La Jolla )
13 ---> ( 44, Irvine )
14 ---> ( 41, San Clemente )
15
16
17 ---> ( 17, Orange )
18 ---> ( 18, Del Mar )
19 ---> ( 49, San Diego )
20 ---> ( 19, Brea )
21
22 ---> ( 22, Aliso Viejo )
23
24
25
26 ---> ( 88, El Segundo )
27
28
29 ---> ( 60, Santa Ana )
30 ---> ( 29, Dana Point )

 */

#include <iostream>
#include <fstream>
#include <string>
#include <map>
#include "hashTable.h"

using namespace std;

int main()
{
    cout << "Lina Kang\n"
            "CS1D MW 2:30-5:00PM\n"
            "Assignment 7 - Hashing\n"
            "This program exercises the hashing algorithms using a map\n";
    cout << "Data Structure Used: map  \n"
            "--------------------------\n"
            "Part One: \n"
            "--------------------------\n";

    hashmap citiesDouble(29);

    citiesDouble.hashingDouble();

    cout << "--------------------------\n"
            "Part Two: \n"
            "--------------------------\n";

    hashmap citiesQuadratic(31);

    citiesQuadratic.hashingQuadratic();

}

/* input.txt
```

```

18 Laguna Niguel
41 Mission Viejo
22 San Clemente
44 Irvine
Delete key 41
58 Lake Forest
32 San Diego
49 Anaheim
Delete key 58
31 Los Angeles
17 Orange
72 Palms Springs
41 Riverside
Delete key 72
19 Brea
60 Santa Ana
35 Tustin
103 Oceanside
11 La Jolla
18 Del Mar
22 Aliso Viejo
49 Laguna Beach
Delete key 41
42 Vista
49 San Diego
99 San Juan
29 Dana Point
88 El Segundo
41 San Clemente
62 Laguna Hills

 */

#ifndef HASHTABLE_H_
#define HASHTABLE_H_

#include <vector>
#include <iostream>

using namespace std;

struct node
{
    string city = "";
    int key = -1;
    int index = -1;
};

class hashmap
{
public:
    hashmap(int);
    ~hashmap();

    // Double Hashing Algorithm
    void hashingDouble();
    int hashOne(int key, int i);
    void insertDouble(int key, string line);
    void eraseDouble(int key);

    // Quadratic Hashing Algorithm
    void hashingQuadratic();
    int hashTwo(int key, int i);
    void insertQuadratic(int key, string line);
    void eraseQuadratic(int key);

    void print();

private:
    node *map;
    int size;
};

hashmap::hashmap(int n)
{
    size = n;
    map = new node[n];
}
hashmap::~hashmap()
{
    delete [] map;
}

// Part 1. Double Hashing

void hashmap::hashingDouble()
{
    ifstream input ("input.txt");   // read in input from a separate txt file

    string line;
    int key;

    while(input >> line)
    {
        if(line[0] == 'D') // if the line begins with "Delete" - erase key
        {
            input >> line;
            input >> key;
            input.ignore(100,'\n');

            eraseDouble(key);
        }
        else    // insert key and its info
        {
            key = stoi(line);
            input.ignore(100, ' ');
            getline(input, line);

            insertDouble(key, line);
        }
    }

    print();
```

```cpp
}

int hashmap::hashOne(int key, int i)
{
    return ( key % size + i*( 13- key % 13) ) % size;
}

void hashmap::insertDouble(int key, string line)
{
    int i = 0;
    int index = 0;

    // 1. Run it through hash function
    index = hashOne(key,i);

    // 2. If index is EMPTY or has SAME KEY, insert/replace element
    if(map[index].city == "" || key == map[index].key)
    {
        map[index].city = line;
        map[index].key = key;
        map[index].index = index;
    }
    // 3. If index is occupied...
    else
    {
        // 4. Until an EMPTY index or element with SAME KEY is found...
        while(map[index].city != "" && map[index].key != key)
        {
            i++;
            index = hashOne(key, i);
        }
        map[index].city = line;
        map[index].key = key;
        map[index].index = index;
    }

}

void hashmap::eraseDouble(int key)
{
    int i = 0;
    int index = 0;

    // Run until element with SAME KEY is found
    do
    {
        index = hashOne(key, i);
        i++;
    }while(map[index].key != key && i < size);
    // i < size to prevent infinite loop

    if(map[index].key == key)
    {
        map[index].city = "";
        map[index].key = -1;
        map[index].index = -1;
    }
```

```
}

// Part 2. Quadratic Hashing

void hashmap::hashingQuadratic()
{
    ifstream input ("input.txt");   // read in input from a separate txt file

    string line;
    int key;

    while(input >> line)
    {
        if(line[0] == 'D') // if the line begins with "Delete" - erase key
        {
            input >> line;
            input >> key;
            input.ignore(100,'\n');

            eraseQuadratic(key);
        }
        else    // insert key and its info
        {
            key = stoi(line);
            input.ignore(100, ' ');
            getline(input, line);

            insertQuadratic(key, line);
        }
    }
    print();
}

int hashmap::hashTwo(int key, int i)
{
    return (key % size + i*i) % size;
}

void hashmap::insertQuadratic(int key, string line)
{
    int i = 0;
    int index = 0;

    // 1. Run it through hash function
    index = hashTwo(key,i);

    // 2. If index is EMPTY or has SAME KEY, insert/replace element
    if(map[index].city == "" || key == map[index].key)
    {
        map[index].city = line;
        map[index].key = key;
        map[index].index = index;
    }
    // 3. If index is occupied...
    else
    {
        // 4. Until an EMPTY index or element with SAME KEY is found...
        while(map[index].city != "" && map[index].key != key)
        {
            i++;
            index = hashTwo(key, i);
        }
        map[index].city = line;
        map[index].key = key;
        map[index].index = index;
    }
}

void hashmap::eraseQuadratic(int key)
{
    int i = 0;
    int index = 0;

    // Run until element with SAME KEY is found
    do
    {
        index = hashTwo(key, i);
        i++;
    }while(map[index].key != key && i < size);
    // i < size to prevent infinite loop

    if(map[index].key == key)
    {
        map[index].city = "";
        map[index].key = -1;
        map[index].index = -1;
    }
}

void hashmap::print()
{
    // print the map
    for(int i = 0; i < size; i++)
    {
        cout << i;
        if(map[i].city != "")
            cout << " ---> ( " << map[i].key << ", " <<
                map[i].city << " )";
        cout << endl;
    }
}

#endif /* HASHTABLE_H_ */

```