

main.cpp

```
1// Lina Kang
2// CS1D MW 2:30 - 5:00 PM
3// Assignment 7 - Hashing
4// This program exercises the hashing algorithms using a map
5
6/* OUTPUT
7Lina Kang
8CS1D MW 2:30-5:00PM
9Assignment 7 - Hashing
10This program exercises the hashing algorithms using a map
11Data Structure Used: map
12-----
13Part One:
14-----
150 ---> ( 29, Dana Point )
161 ---> ( 88, El Segundo )
172 ---> ( 31, Los Angeles )
183 ---> ( 32, San Diego )
194 ---> ( 62, Laguna Hills )
205
216 ---> ( 35, Tustin )
227 ---> ( 60, Santa Ana )
238
249
2510
2611 ---> ( 11, La Jolla )
2712 ---> ( 99, San Juan )
2813 ---> ( 42, Vista )
2914
3015 ---> ( 44, Irvine )
3116 ---> ( 103, Oceanside )
3217 ---> ( 17, Orange )
3318 ---> ( 18, Del Mar )
3419 ---> ( 19, Brea )
3520 ---> ( 49, San Diego )
3621
3722 ---> ( 22, Aliso Viejo )
3823 ---> ( 41, San Clemente )
3924
4025
4126
4227
4328
44-----
45Part Two:
46-----
470 ---> ( 31, Los Angeles )
481 ---> ( 32, San Diego )
492
503
514 ---> ( 35, Tustin )
525
536 ---> ( 99, San Juan )
547
558
569 ---> ( 62, Laguna Hills )
5710 ---> ( 103, Oceanside )
```

main.cpp

```
58 11 ---> ( 42, Vista )
59 12 ---> ( 11, La Jolla )
60 13 ---> ( 44, Irvine )
61 14 ---> ( 41, San Clemente )
62 15
63 16
64 17 ---> ( 17, Orange )
65 18 ---> ( 18, Del Mar )
66 19 ---> ( 49, San Diego )
67 20 ---> ( 19, Brea )
68 21
69 22 ---> ( 22, Aliso Viejo )
70 23
71 24
72 25
73 26 ---> ( 88, El Segundo )
74 27
75 28
76 29 ---> ( 60, Santa Ana )
77 30 ---> ( 29, Dana Point )
78
79 */
80
81 #include <iostream>
82 #include <fstream>
83 #include <string>
84 #include <map>
85 #include "hashTable.h"
86
87 using namespace std;
88
89 int main()
90 {
91     cout << "Lina Kang\n"
92            "CS1D MW 2:30-5:00PM\n"
93            "Assignment 7 - Hashing\n"
94            "This program exercises the hashing algorithms using a map\n";
95     cout << "Data Structure Used: map  \n"
96            "-----\n"
97            "Part One:  \n"
98            "-----\n";
99
100     hashmap citiesDouble(29);
101
102     citiesDouble.hashingDouble();
103
104     cout << "-----\n"
105            "Part Two:  \n"
106            "-----\n";
107
108     hashmap citiesQuadratic(31);
109
110     citiesQuadratic.hashingQuadratic();
111
112 }
113
114 /* input.txt
```

```

115
116 18 Laguna Niguel
117 41 Mission Viejo
118 22 San Clemente
119 44 Irvine
120 Delete key 41
121 58 Lake Forest
122 32 San Diego
123 49 Anaheim
124 Delete key 58
125 31 Los Angeles
126 17 Orange
127 72 Palms Springs
128 41 Riverside
129 Delete key 72
130 19 Brea
131 60 Santa Ana
132 35 Tustin
133 103 Oceanside
134 11 La Jolla
135 18 Del Mar
136 22 Aliso Viejo
137 49 Laguna Beach
138 Delete key 41
139 42 Vista
140 49 San Diego
141 99 San Juan
142 29 Dana Point
143 88 El Segundo
144 41 San Clemente
145 62 Laguna Hills
146
147 */
148
149 #ifndef HASHTABLE_H_
150 #define HASHTABLE_H_
151
152 #include <vector>
153 #include <iostream>
154
155 using namespace std;
156
157 struct node
158 {
159     string city = "";
160     int key = -1;
161     int index = -1;
162 };
163
164 class hashmap
165 {
166 public:
167     hashmap(int);
168     ~hashmap();
169
170     // Double Hashing Algorithm
171     void hashingDouble();

```

```

172     int hashOne(int key, int i);
173     void insertDouble(int key, string line);
174     void eraseDouble(int key);
175
176     // Quadratic Hashing Algorithm
177     void hashingQuadratic();
178     int hashTwo(int key, int i);
179     void insertQuadratic(int key, string line);
180     void eraseQuadratic(int key);
181
182     void print();
183
184 private:
185     node *map;
186     int size;
187 };
188
189 hashmap::hashmap(int n)
190 {
191     size = n;
192     map = new node[n];
193 }
194 hashmap::~hashmap()
195 {
196     delete [] map;
197 }
198
199 // Part 1. Double Hashing
200
201 void hashmap::hashingDouble()
202 {
203     ifstream input ("input.txt");    // read in input from a separate txt file
204
205     string line;
206     int key;
207
208     while(input >> line)
209     {
210         if(line[0] == 'D') // if the line begins with "Delete" - erase key
211         {
212             input >> line;
213             input >> key;
214             input.ignore(100, '\n');
215
216             eraseDouble(key);
217         }
218         else // insert key and its info
219         {
220             key = stoi(line);
221             input.ignore(100, ' ');
222             getline(input, line);
223
224             insertDouble(key, line);
225         }
226     }
227
228     print();

```

```

229 }
230
231 int hashmap::hashOne(int key, int i)
232 {
233     return ( key % size + i*( 13- key % 13) ) % size;
234 }
235
236 void hashmap::insertDouble(int key, string line)
237 {
238     int i = 0;
239     int index = 0;
240
241     // 1. Run it through hash function
242     index = hashOne(key,i);
243
244     // 2. If index is EMPTY or has SAME KEY, insert/replace element
245     if(map[index].city == "" || key == map[index].key)
246     {
247         map[index].city = line;
248         map[index].key = key;
249         map[index].index = index;
250     }
251     // 3. If index is occupied...
252     else
253     {
254         // 4. Until an EMPTY index or element with SAME KEY is found...
255         while(map[index].city != "" && map[index].key != key)
256         {
257             i++;
258             index = hashOne(key, i);
259         }
260         map[index].city = line;
261         map[index].key = key;
262         map[index].index = index;
263     }
264
265 }
266
267 void hashmap::eraseDouble(int key)
268 {
269     int i = 0;
270     int index = 0;
271
272     // Run until element with SAME KEY is found
273     do
274     {
275         index = hashOne(key, i);
276         i++;
277     } while(map[index].key != key && i < size);
278     // i < size to prevent infinite loop
279
280     if(map[index].key == key)
281     {
282         map[index].city = "";
283         map[index].key = -1;
284         map[index].index = -1;
285     }

```

```

286 }
287
288 // Part 2. Quadratic Hashing
289
290 void hashmap::hashingQuadratic()
291 {
292     ifstream input ("input.txt");    // read in input from a separate txt file
293
294     string line;
295     int key;
296
297     while(input >> line)
298     {
299         if(line[0] == 'D') // if the line begins with "Delete" - erase key
300         {
301             input >> line;
302             input >> key;
303             input.ignore(100, '\n');
304
305             eraseQuadratic(key);
306         }
307         else    // insert key and its info
308         {
309             key = stoi(line);
310             input.ignore(100, ' ');
311             getline(input, line);
312
313             insertQuadratic(key, line);
314         }
315     }
316     print();
317 }
318
319 int hashmap::hashTwo(int key, int i)
320 {
321     return (key % size + i*i) % size;
322 }
323
324 void hashmap::insertQuadratic(int key, string line)
325 {
326     int i = 0;
327     int index = 0;
328
329     // 1. Run it through hash function
330     index = hashTwo(key,i);
331
332     // 2. If index is EMPTY or has SAME KEY, insert/replace element
333     if(map[index].city == "" || key == map[index].key)
334     {
335         map[index].city = line;
336         map[index].key = key;
337         map[index].index = index;
338     }
339     // 3. If index is occupied...
340     else
341     {
342         // 4. Until an EMPTY index or element with SAME KEY is found...

```

```

343     while(map[index].city != "" && map[index].key != key)
344     {
345         i++;
346         index = hashTwo(key, i);
347     }
348     map[index].city = line;
349     map[index].key = key;
350     map[index].index = index;
351 }
352 }
353
354 void hashmap::eraseQuadratic(int key)
355 {
356     int i = 0;
357     int index = 0;
358
359     // Run until element with SAME KEY is found
360     do
361     {
362         index = hashTwo(key, i);
363         i++;
364     }while(map[index].key != key && i < size);
365     // i < size to prevent infinite loop
366
367     if(map[index].key == key)
368     {
369         map[index].city = "";
370         map[index].key = -1;
371         map[index].index = -1;
372     }
373 }
374
375 void hashmap::print()
376 {
377     // print the map
378     for(int i = 0; i < size; i++)
379     {
380         cout << i;
381         if(map[i].city != "")
382             cout << " ---> ( " << map[i].key << ", " <<
383                 map[i].city << " )";
384         cout << endl;
385     }
386 }
387
388 #endif /* HASHTABLE_H_ */
389

```