

```

1  //*****
2  // Program #1 Hashing Experiment
3  // Name: Ben Diekhoff
4  // CMPS 5243 Algorithms
5  // Dr. Halverson
6  // Date: 03/24/2020
7  //*****
8  /*
9  This program takes a set of keys, hashes them using linear probing and double
10 hashing, and returns the complete hash tables for each CRP as well as the
11 average number of probes for each method.
12 /*****/
13 #include <iostream>
14 #include <iomanip>
15 #include <fstream>
16 #include <utility>
17 #include <ctime>
18 using namespace std;
19 ofstream outfile("output.txt");
20 ifstream datafile("datafile.txt");
21
22
23
24 class HASH {
25 private:
26     pair<int, int>* table;           // the hash table
27     void Clean_Table(int);
28     int Lin_Probe(int, int, int);
29     int Double_Probe(int, int, int);
30     void Print_Table(int);
31 public:
32     HASH(int);                     // constructor
33     ~HASH();                       // destructor
34     int Mod_Hash(int, int);
35     void Clean_Table_Pub(int);      // caller function
36     int Lin_Probe_Pub(int, int, int); // caller function
37     int Double_Probe_Pub(int, int, int); // caller function
38     void Print_Table_Pub(int);      // caller function
39
40 };
41
42
43 /*****
44     Clean_Table
45     Parameters: One int
46
47     Sets every value in the dynamically allocated table to a sentinel value.
48 *****/
49 void HASH::Clean_Table(int table_size) {
50     for (int i = 0; i < table_size; i++) {
51         table[i].first = -9999; //table[i].first holds keys
52         table[i].second = 0;    //table[i].second holds a probe count

```

```
53     }
54 }
55
56
57 /*****
58     Lin_Probe
59     Parameters: Three ints
60     Inserts a key into an "empty" slot in the table (orig_loc by default),
61     where -9999 is considered empty. If the orig_loc isn't empty, it increments
62     through the table one index at a time (mod table_size) until an empty slot
63     is found. It returns the number of probes it takes for successful
64     insertion, as well as storing that value in table[oorig_loc].second.
65 *****/
66 int HASH::Lin_Probe(int key, int orig_loc, int table_size) {
67     int count = 1;
68
69     //table[i].first holds keys
70     //table[i].second holds a probe count
71     while (table[orig_loc].first != -9999) {
72         count++;
73         orig_loc = (orig_loc + 1) % table_size;
74     }
75     table[orig_loc].first = key;
76     table[orig_loc].second = count;
77     return count;
78 }
79
80
81
82 /*****
83     Double_Probe
84     Parameters: Three ints
85     Inserts a key into an "empty" slot in the table (orig_loc by default),
86     where -9999 is considered empty. If the orig_loc isn't empty, it increments
87     through the table based on the last digit of the key + 1. When it finds an
88     empty slot the key is inserted. It returns the number of probes it takes
89     for successful insertion, as well as storing that value in
90     table[oorig_loc].second.
91 *****/
92 int HASH::Double_Probe(int key, int orig_loc, int table_size) {
93     int increment = (key % 10) + 1;
94     int count = 1;
95
96     //table[i].first holds keys
97     //table[i].second holds a probe count
98     while (table[orig_loc].first != -9999) {
99         count++;
100         orig_loc = (orig_loc + increment) % table_size;
101     }
102     table[orig_loc].first = key;
103     table[orig_loc].second = count;
104     return count;
```

```

105 }
106
107
108 /*****
109     Print_Table
110     Parameters: One int
111     Prints the table in a nice, readable format to a .txt file.
112     -9999 as a value under KEY means there was nothing inserted into
113     that location. The same is true whe PROBES is 0.
114 *****/
115 void HASH::Print_Table(int table_size) {
116     outfile << setw(18) << left << "LOCATION" << setw(10) << "KEY"
117         << setw(10) << "PROBES" << "\n";
118
119     outfile << "\n";
120
121     //table[i].first holds keys
122     //table[i].second holds a probe count
123     for (int loc = 0; loc < table_size; loc++) {
124         outfile << left << setw(18) << loc << setw(10) << table[loc].first
125             << setw(10) << table[loc].second << "\n";
126     }
127 }
128
129
130 //////////////////////////////////////
131 //////////////////////////////////////
132 ////////////////////////////////////// PUBLIC FUCNTIONS //////////////////////////////////////
133 //////////////////////////////////////
134 //////////////////////////////////////
135
136
137 /*****
138     HASH
139     Parameters: One int
140     Constructor for the HASH class.
141 *****/
142 HASH::HASH(int table_size) {
143     table = new pair<int, int>[table_size];
144     //table[i].first holds keys
145     //table[i].second holds a probe count
146 }
147
148
149 /*****
150     HASH
151     Parameters: None
152     Destructor for the HASH class.
153 *****/
154 HASH::~HASH() {
155     delete[] table;
156     table = nullptr;

```

```

157 }
158
159
160 /*****
161     Mod_Hash
162     Parameters: Two ints
163     Calculates and returns the initial hash location for a given key.
164     *****/
165 int HASH::Mod_Hash(int key, int table_size) {
166     return (key % table_size);
167 }
168
169
170 /*****
171     Clean_Table_Pub
172     Parameters: Three ints
173     Caller function for Clean_Table, which:
174     Sets every value in the dynamically allocated table to -9999.
175     *****/
176 void HASH::Clean_Table_Pub(int table_size) {
177     Clean_Table(table_size);
178 }
179
180
181 /*****
182     Lin_Probe_Pub
183     Parameters: Three ints
184     Caller function for Lin_Probe, which:
185     Inserts a key into an "empty" slot in the table (orig_loc by default),
186     where -9999 is considered empty. If the orig_loc isn't empty, it increments
187     through the table one index at a time (mod table_size) until an empty slot
188     is found. It returns the number of probes it takes for successful
189     insertion, as well as storing that value in table[oorig_loc].second.
190     *****/
191 int HASH::Lin_Probe_Pub(int key, int orig_loc, int table_size) {
192     return Lin_Probe(key, orig_loc, table_size);
193 }
194
195
196 /*****
197     Double_Probe_Pub
198     Parameters: Three ints
199     Caller function for Double_Probe, which:
200     Inserts a key into an "empty" slot in the table (orig_loc by default),
201     where -9999 is considered empty. If the orig_loc isn't empty, it increments
202     through the table based on the last digit of the key + 1. When it finds an
203     empty slot the key is inserted. It returns the number of probes it takes
204     for successful insertion, as well as storing that value in
205     table[oorig_loc].second.
206     *****/
207 int HASH::Double_Probe_Pub(int key, int orig_loc, int table_size) {
208     return Double_Probe(key, orig_loc, table_size);

```

```

209 }
210
211
212 /*****
213     Print_Table_Pub
214     Parameters: One int
215     Caller function for Print_Table, which:
216     Prints the table in a nice, readable format to a .txt file.
217     -9999 as a value under KEY means there was nothing inserted into
218     that location. The same is true whe PROBES is 0.
219 *****/
220 void HASH::Print_Table_Pub(int table_size) {
221     Print_Table(table_size);
222 }
223
224 //////////////////////////////////////
225 //////////////////////////////////////
226 //////////////////////////////////////    MAIN FUCNTION    //////////////////////////////////////
227 //////////////////////////////////////
228 //////////////////////////////////////
229
230 int main() {
231
232     const int table_size = 311;
233     const int randData_size = 250;
234     int probe_count = 0; // 0 is a sentinel value
235     int orig_loc = -1; // Location a key is initially hashed toS
236     double avg_probes = -1.1; // Avg # of probes for inserting into a table
237     int randData[randData_size] = { 0 }; // holds random numbers to be hashed
238     double n; // number of items to insert
239     double alpha; // Load factor
240     int key; //The value to be hashed
241     HASH h(table_size);
242
243     // Run all experiments twice
244     // Load the first half of the dataset into randData on iteration 0
245     // Load the second half on iteration 1
246     for (int iter = 0; iter < 2; iter++) {
247         for (int i = 0; i < randData_size; i++){
248             datafile >> randData[i];
249         }
250         // Loop through the 4 experiments
251         for (int i = 0; i < 4; i++) {
252
253             //make sure variables are reset
254             probe_count = 0;
255             orig_loc = -1;
256             avg_probes = -1.1;
257             probe_count = 0;
258             h.Clean_Table_Pub(table_size); // Set all values in h.table to -9999
259
260             //determine how much data to insert into the hash table

```

```
261     if (i == 0 || i == 2)
262         n = 205;
263     else
264         n = 250;
265
266     alpha = n / table_size;
267
268     // Linear Probe
269     if (i < 2) {
270         for (int i = 0; i < n; i++) {
271             key = randData[i];
272             // determines hash location for key
273             orig_loc = h.Mod_Hash(key, table_size);
274             // accumulator for the number of probes required to insert
275             // all values into the table
276             probe_count += h.Lin_Probe_Pub(key, orig_loc, table_size);
277         }
278         outfile << "LINEAR PROBE WITH ALPHA = " << alpha
279             << "\nIteration: " << iter + 1 << "\n=====\\n";
280
281     }
282
283     // Double Hash
284     else {
285         for (int i = 0; i < n; i++) {
286             key = randData[i];
287             // determines hash location for key
288             orig_loc = h.Mod_Hash(key, table_size);
289             // accumulator for the number of probes required to insert
290             // all values into the table
291             probe_count += h.Double_Probe_Pub(key, orig_loc, table_size);
292         }
293         outfile << "DOUBLE HASH WITH ALPHA " << alpha
294             << "\nIteration: " << iter + 1 << "\n=====\\n";
295     }
296
297     // Print information to outfile
298     avg_probes = probe_count / n;
299     h.Print_Table_Pub(table_size);
300     outfile << "\\nAvg # of probes: " << fixed << setprecision(3)
301         << setw(21) << avg_probes << "\\n\\n";
302 }
303 }
304 outfile.close();
305 datafile.close();
306 return 0;
307 }
308
309
```