```
2 // Program #1 Hashing Experiment
3 // Name: Ben Diekhoff
4 // CMPS 5243 Algorithms
5 // Dr. Halverson
6 // Date: 03/24/2020
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.
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);
      void Print_Table(int);
30
31 public:
32
                                       // constructor
    HASH(int);
33
      ~HASH();
                                       // destructor
34
      int Mod_Hash(int, int);
35
      void Clean_Table_Pub(int);
                                       // caller function
                                      // caller function
36
      int Lin_Probe_Pub(int, int, int);
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 dyanimcally allocated table to a sentinel value.
49 void HASH::Clean_Table(int table_size) {
50
      for (int i = 0; i < table_size; i++) {</pre>
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
       Inserts a key into an "empty" slot in the table (orig_loc by default),
60
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
       is found. It returns the number of probes it takes for successful
63
       insertion, as well as storing that value in table[oirg loc].second.
64
    65
66 int HASH::Lin_Probe(int key, int orig_loc, int table_size) {
67
       int count = 1;
68
       //table[i].first holds keys
69
70
       //table[i].second holds a probe count
       while (table[orig_loc].first != -9999) {
71
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
       Inserts a key into an "empty" slot in the table (orig_loc by default),
85
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
       empty slot the key is inserted. It returns the number of probes it takes
88
89
       for successful insertion, as well as storing that value in
90
       table[oirg loc].second.
   91
   int HASH::Double_Probe(int key, int orig_loc, int table_size) {
       int increment = (key % 10) + 1;
93
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
100
           orig_loc = (orig_loc + increment) % table_size;
101
102
       table[orig_loc].first = key;
       table[orig loc].second = count;
103
       return count;
104
```

```
...ithm Analysis\Assignments\5243-AlgAn\A01-Hashing\hash.cpp
```

```
3
```

```
105 }
106
107
109
     Print Table
     Parameters: One int
110
     Prints the table in a nice, readable format to a .txt file.
111
     -9999 as a value under KEY means there was nothing inserted into
112
113
     that location. The same is true whe PROBES is 0.
114 ******************************
115 void HASH::Print_Table(int table_size) {
     outfile << setw(18) << left << "LOCATION" << setw(10) << "KEY"
116
117
       << setw(10) << "PROBES" << "\n";
118
119
     outfile << "\n";</pre>
120
     //table[i].first holds keys
121
     //table[i].second holds a probe count
122
123
     for (int loc = 0; loc 
       outfile << left << setw(18) << loc << setw(10) << table[loc].first
124
125
          << setw(10) << table[loc].second << "\n";</pre>
126
     }
127 }
128
129
135
136
138
     HASH
139
     Parameters: One int
140
     Constructor for the HASH class.
142 HASH::HASH(int table_size) {
143
     table = new pair<int, int>[table_size];
     //table[i].first holds keys
144
     //table[i].second holds a probe count
145
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;
```

```
...ithm Analysis\Assignments\5243-AlgAn\A01-Hashing\hash.cpp
```

```
157 }
158
159
   160
161
      Mod Hash
162
      Parameters: Two ints
      Calculates and returns the initial hash location for a given key.
163
165 int HASH::Mod_Hash(int key, int table_size) {
      return (key % table_size);
166
167 }
168
169
171
      Clean_Table_Pub
172
      Parameters: Three ints
      Caller function for Clean_Table, which:
173
      Sets every value in the dyanimcally allocated table to -9999.
175
176 void HASH::Clean_Table_Pub(int table_size) {
      Clean_Table(table_size);
177
178 }
179
180
   181
182
      Lin_Probe_Pub
183
      Parameters: Three ints
184
      Caller function for Lin_Probe, which:
      Inserts a key into an "empty" slot in the table (orig loc by default),
      where -9999 is considered empty. If the orig_loc isn't empty, it increments
186
      through the table one index at a time (mod table size) until an empty slot
187
      is found. It returns the number of probes it takes for successful
188
189
      insertion, as well as storing that value in table[oirg_loc].second.
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
      Caller function for Double Probe, which:
199
200
      Inserts a key into an "empty" slot in the table (orig_loc by default),
      where -9999 is considered empty. If the orig_loc isn't empty, it increments
201
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[oirg_loc].second.
           *************************
206
207
   int HASH::Double_Probe_Pub(int key, int orig_loc, int table_size) {
      return Double_Probe(key, orig_loc, table_size);
208
```

```
...ithm Analysis\Assignments\5243-AlgAn\A01-Hashing\hash.cpp
```

```
209 }
210
211
213
      Print_Table_Pub
214
      Parameters: One int
      Caller function for Print Table, which:
215
      Prints the table in a nice, readable format to a .txt file.
216
217
      -9999 as a value under KEY means there was nothing inserted into
      that location. The same is true whe PROBES is 0.
218
219 *****************************
220 void HASH::Print_Table_Pub(int table_size) {
      Print_Table(table_size);
221
222 }
223
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
      double n; // number of items to insert
238
      double alpha; // Load factor
239
240
                   //The value to be hashed
      int key;
241
      HASH h(table_size);
242
243
      // Run all experiments twice
244
          // Load the first half of the dataset into randData on iteration 0
          // Load the second half on iteration 1
245
246
      for (int iter = 0; iter < 2; iter++) {</pre>
247
                for (int i = 0; i < randData_size; i++){</pre>
248
                   datafile >> randData[i];
249
          // Loop through the 4 experiments
250
         for (int i = 0; i < 4; i++) {
251
252
             //make sure variables are reset
253
             probe_count = 0;
254
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
             //determine how much data to insert into the hash table
260
```

```
...ithm Analysis\Assignments\5243-AlgAn\A01-Hashing\hash.cpp
```

```
6
```

```
if (i == 0 || i == 2)
261
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
                     outfile << "LINEAR PROBE WITH ALPHA = " << alpha
278
279
                          << "\nIteration: " << iter + 1 << "\n=======\n";</pre>
280
281
                 }
282
283
                 // Double Hash
                 else {
284
                     for (int i = 0; i < n; i++) {</pre>
285
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
                     outfile << "DOUBLE HASH WITH ALPHA" << alpha
293
294
                          << "\nIteration: " << iter + 1 << "\n=======\n";</pre>
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)</pre>
301
                     << setw(21) << avg_probes << "\n\n";</pre>
302
             }
303
         }
304
         outfile.close();
305
         datafile.close();
306
         return 0;
307 }
308
309
```