```
* AUTHOR
                    : Nick Reardon
2
   * Assignment #7
3
                    : Hashing Algorithms
   * CLASS
                    : CS1D
5
   * SECTION
                    : MW - 2:30p
6
   * DUE DATE
                    : 03 / 04 / 20
   *******************************
8 #ifndef _ARRAYHASHMAP_H_
9 #define _ARRAYHASHMAP_H_
10 #include <string>
11 #include <iostream>
12 #include <iomanip>
13
14 namespace errorType
15 {
16
      enum errors
17
      {
18
          DEFUALT,
19
          EMPTY,
20
          FULL
21
22
      };
23
24
      std::string errorString[]
25
      {
26
          "Error - An error occured",
          "Error - Map is empty",
27
28
          "Error - Map is full"
29
      };
30 }
31
32 enum indexLabel
33 {
34
      EMPTY,
      FULL,
35
      AVAILABLE
36
37 };
38
39 template <class T_key, class T_value>
40 struct T_struct
41 {
42
      T_key key;
43
      T_value value;
44
45
      enum indexLabel label = EMPTY;
46
47
      T_struct<T_key, T_value>()
48
      {
49
          key = -1;
50
          label = EMPTY;
51
      }
52
```

```
53
        T_struct<T_key, T_value>(const T_key& key, const T_value& value)
 54
        {
 55
             this->key = key;
 56
             this->value = value;
 57
 58
            label = EMPTY;
 59
        }
 60
 61
        T_struct<T_key, T_value>(const T_struct<T_key, T_value>& rhs)
 62
 63
             this->key = rhs.key;
 64
             this->value = rhs.value;
 65
 66
            this->label = rhs.label;
 67
        }
 68
 69
        T_struct<T_key, T_value>& operator=(const T_struct<T_key, T_value>& rhs)
 70
 71
             this->key = rhs.key;
 72
            this->value = rhs.value;
 73
 74
             this->label = rhs.label;
 75
            return *this;
 76
 77
        }
 78 };
 79
 80 template <class T_key, class T_value>
 81 T_struct<T_key, T_value> make_struct(T_key newKey, T_value newValue)
 82 {
 83
        return T_struct<T_key, T_value>(newKey, newValue);
 84 }
 85
 86
 87
 88 template <class T_key, class T_value>
 89 class DoubleHashMap
 90 {
 91 private:
 92
 93
        T_struct<T_key, T_value>* map;
 94
 95
        int currentSize;
 96
        int capacity;
 97
        // ostream member? Assign it in constructor or method???
 98
        // set to NULL?
99
100
101 protected:
102
        int DoubleHash(const int givenKey, const int collisionCount) const
103
104
```

```
... shing Algorithms \verb|\CS1D-AS7-Hashing Algorithms \verb|\Array Hash Map.h| \\
```

```
3
```

```
105
             int hashKey;
106
             int j = collisionCount;
107
108
             int k = givenKey;
109
             int N = capacity;
110
             /*
111
112
             int hk;
113
             int hk2;
114
             hk = (k \% N);
115
             hk2 = (k \% 13);
116
             hk2 = 13 - hk2;
117
118
             hk2 = j * hk2;
119
120
             hashKey = hk + hk2;
121
122
             hashKey = hashKey % N;
123
124
125
             hashKey = (((k \% N) + (j * (13 - (k \% 13)))) \% N);
126
127
             return hashKey;
128
         }
129
130
         int DoubleHash(const T_struct<T_key, T_value>& toInsert, const int
           collisionCount) const
131
         {
132
             DoubleHash(toInsert.key, collisionCount);
133
         }
134
135 public:
136
137
         DoubleHashMap(const int newCapacity)
138
         {
             map = new T_struct<T_key, T_value>[newCapacity];
139
140
141
             currentSize = 0;
142
143
             capacity = newCapacity;
144
         }
145
146
         ~DoubleHashMap()
147
         {
148
             delete[] map;
149
         }
150
151
         void insert(const T_struct<T_key, T_value>& toInsert)
152
         {
153
             if (full())
154
             {
                 throw(errorType::FULL, errorType::errorString[FULL], 5);
155
```

```
...shingAlgorithms\CS1D-AS7-HashingAlgorithms\ArrayHashMap.h
```

```
4
```

```
156
157
158
             int hashKey;
159
             std::string output = std::to_string(toInsert.key);
160
             int collisionCount = 0;
161
             bool stopHash = false;
162
             bool success = false;
163
164
             while (stopHash == false)
165
                 hashKey = DoubleHash(toInsert.key, collisionCount);
166
167
168
                  if (map[hashKey].label == EMPTY ||
169
                      map[hashKey].label == AVAILABLE)
170
                  {
171
                      stopHash = true;
172
                      success = true;
173
174
                  }
175
                 else
176
                  {
                      if (map[hashKey].key == toInsert.key)
177
178
                      {
179
                          stopHash = true;
180
                          success = true;
181
                      }
182
183
                      collisionCount++;
184
                  }
185
186
                  output += "->" + std::to_string(hashKey);
187
188
             }
189
190
             if (success)
191
192
             {
193
                  if (map[hashKey].label == FULL)
194
                  {
                      std::cout << "Updating: " << '(' << map[hashKey].key << ", " <<</pre>
195
                        map[hashKey].value << ')' << " to "</pre>
                          << '(' << toInsert.key << ", " << toInsert.value << ')'
196
197
                          << '\n' << "Hashed Key: " << output << '\n' << '\n';
198
                  }
                  else
199
200
                  {
                      std::cout << "Inserting: " << '(' << toInsert.key << ", " <<</pre>
201
                        toInsert.value << ')'
                          << '\n' << "Hashed Key: " << output << '\n' << '\n';
202
203
                  }
204
205
                 map[hashKey] = toInsert;
```

```
... shing Algorithms \verb|\CS1D-AS7-Hashing Algorithms \verb|\Array Hash Map.h| \\
```

```
5
```

```
206
                 map[hashKey].label = FULL;
207
208
                 currentSize++;
209
             }
210
         }
211
212
213
         void remove(const T_key key)
214
         {
215
             if (empty())
216
             {
217
                 throw(errorType::FULL, errorType::errorString[FULL], 5);
218
             }
219
220
             int hashKey;
221
             std::string output = std::to_string(key);
222
             int collisionCount = 0;
223
             bool stopHash = false;
224
             bool success = false;
225
             while (stopHash == false)
226
             {
                 hashKey = DoubleHash(key, collisionCount);
227
228
229
                 if (map[hashKey].label == FULL ||
                     map[hashKey].label == AVAILABLE)
230
231
                 {
232
                     if (map[hashKey].key == key)
233
                     {
234
                         stopHash = true;
235
                         success = true;
236
                     }
237
                     else
238
                     {
239
                         collisionCount++;
240
                     }
241
242
                 }
243
                 else
244
                 {
245
                     stopHash = true;
246
247
                 output += "->" + std::to_string(hashKey);
248
             }
249
250
             if (success)
251
             {
252
                 std::cout << "Removing key: " << key << " (" << map[hashKey].key << >
253
                   ", " << map[hashKey].value << ')' << '\n'
254
                     << "Hashed Key: " << output << '\n' << '\n';
255
256
                 map[hashKey].key = -1;
```

```
...shingAlgorithms\CS1D-AS7-HashingAlgorithms\ArrayHashMap.h
```

```
6
```

```
map[hashKey].value = "";
257
258
                 map[hashKey].label = AVAILABLE;
259
260
                 currentSize--;
261
262
263
             }
264
265
266
         }
267
         bool full()
268
269
         {
270
             return currentSize == capacity;
271
         }
272
273
         bool empty()
274
         {
275
             return currentSize == 0;
276
         }
277
278
         int size()
279
280
             return currentSize;
281
         }
282
         void printAll(std::ostream& output)
283
284
         {
285
             if (empty())
286
287
                 throw(errorType::EMPTY, errorType::errorString[EMPTY], 5);
288
             }
289
             output << " Index | LABEL |
                                                       Value" << '\n'
290
                                              Key
                    << "
291
                 << '\n';
292
293
             for (int i = 0; i < capacity; i++)</pre>
294
295
             {
296
                 output << std::right
297
                     << " [" << std::setw(5) << i << "] | ";</pre>
298
                 switch (map[i].label)
299
300
                 case EMPTY:
                      output << "EMPTY |";
301
302
                     break;
303
304
                 case FULL:
                      output << "FULL |";
305
306
                     break;
307
308
                 case AVAILABLE:
```

```
...shingAlgorithms\CS1D-AS7-HashingAlgorithms\ArrayHashMap.h
```

```
7
```

```
309
                   output << "AVAIL |";
310
                   break;
311
312
313
               output << ' ' << std::setw(4) << map[i].key << " |";
314
315
               output << std::left
                   << ' ' << map[i].value
316
                   << '\n';
317
318
            }
           output << "\n\n";</pre>
319
320
        }
321 };
322
324
325
326 template <class T_key, class T_value>
327 class QuadraticHashMap
328 {
329 private:
330
        T struct<T_key, T_value>* map;
331
332
        int currentSize;
333
334
        int capacity;
335
336
        // ostream member? Assign it in constructor or method???
337
        // set to NULL?
338
339 protected:
340
        int QuadraticHash(const int givenKey, const int collisionCount) const
341
342
        {
343
            int hashKey;
344
345
            int j = collisionCount;
346
            int k = givenKey;
            int N = capacity;
347
348
            /*
349
350
            int hk;
351
            int hk2;
352
353
            hk = (k \% N);
            hk2 = (k \% 13);
354
            hk2 = 13 - hk2;
355
356
            hk2 = j * hk2;
357
358
            hashKey = hk + hk2;
359
           hashKey = hashKey % N;
360
```

```
...shingAlgorithms\CS1D-AS7-HashingAlgorithms\ArrayHashMap.h
```

```
361
362
363
364
365
             if(j>0)
366
             {
367
                 hashKey = (((k \% N) + (j * j)) \% N);
368
             }
369
             else
370
             {
371
                 hashKey = (k \% N);
372
             }
373
374
             return hashKey;
375
         }
376
377
         int QuadraticHash(const T_struct<T_key, T_value>& toInsert, const int
           collisionCount) const
378
379
             QuadraticHash(toInsert.key, collisionCount);
380
         }
381
382
     public:
383
         QuadraticHashMap(const int newCapacity)
384
385
             map = new T_struct<T_key, T_value>[newCapacity];
386
387
388
             currentSize = 0;
389
390
             capacity = newCapacity;
391
         }
392
393
         ~QuadraticHashMap()
394
         {
395
             delete[] map;
396
         }
397
398
         void insert(const T_struct<T_key, T_value>& toInsert)
399
             if (full())
400
401
             {
402
                 throw(errorType::FULL, errorType::errorString[FULL], 5);
403
             }
404
405
             int hashKey;
             std::string output = std::to_string(toInsert.key);
406
407
             int collisionCount = 0;
408
             bool stopHash = false;
409
             bool success = false;
410
             while (stopHash == false)
411
```

```
...shingAlgorithms\CS1D-AS7-HashingAlgorithms\ArrayHashMap.h
```

```
9
```

```
412
413
                 hashKey = QuadraticHash(toInsert.key, collisionCount);
414
415
                 if (map[hashKey].label == EMPTY ||
416
                      map[hashKey].label == AVAILABLE)
417
                 {
418
                      stopHash = true;
419
                      success = true;
420
421
                 }
422
                 else
423
                 {
                      if (map[hashKey].key == toInsert.key)
424
425
                      {
426
                          stopHash = true;
427
                          success = true;
428
                      }
429
430
                      collisionCount++;
431
                 }
432
433
                 output += "->" + std::to_string(hashKey);
434
435
             }
436
437
438
             if (success)
439
             {
440
                 if (map[hashKey].label == FULL)
441
                      std::cout << "Updating: " << '(' << map[hashKey].key << ", " <<</pre>
442
                        map[hashKey].value << ')' << " to "</pre>
443
                          << '(' << toInsert.key << ", " << toInsert.value << ')'
                          << '\n' << "Hashed Key: " << output << '\n' << '\n';
444
445
                 }
                 else
446
447
                 {
                      std::cout << "Inserting: " << '(' << toInsert.key << ", " <<</pre>
448
                        toInsert.value << ')'
449
                          << '\n' << "Hashed Key: " << output << '\n' << '\n';
450
                 }
451
452
                 map[hashKey] = toInsert;
453
                 map[hashKey].label = FULL;
454
455
                 currentSize++;
456
             }
457
         }
458
459
460
         void remove(const T_key key)
461
```

```
462
             if (empty())
463
             {
464
                 throw(errorType::FULL, errorType::errorString[FULL], 5);
465
             }
466
467
             int hashKey;
468
             std::string output = std::to_string(key);
             int collisionCount = 0;
469
470
             bool stopHash = false;
471
             bool success = false;
472
             while (stopHash == false)
473
474
                 hashKey = QuadraticHash(key, collisionCount);
475
476
                 if (map[hashKey].label == FULL ||
477
                     map[hashKey].label == AVAILABLE)
478
                 {
479
                     if (map[hashKey].key == key)
480
                     {
481
                          stopHash = true;
482
                         success = true;
483
                     }
484
                     else
485
                     {
486
                         collisionCount++;
487
488
                     }
489
                 }
490
                 else
491
                 {
492
                     stopHash = true;
493
494
                 output += "->" + std::to_string(hashKey);
495
             }
496
             if (success)
497
498
             {
499
                 std::cout << "Removing key: " << key << " (" << map[hashKey].key << >
500
                   ", " << map[hashKey].value << ')' << '\n'
501
                     << "Hashed Key: " << output << '\n' << '\n';
502
503
                 map[hashKey].key = -1;
                 map[hashKey].value = "";
504
505
                 map[hashKey].label = AVAILABLE;
506
507
                 currentSize--;
508
509
510
             }
511
512
```

```
513
514
515
         bool full()
516
517
             return currentSize == capacity;
518
         }
519
520
         bool empty()
521
         {
522
             return currentSize == 0;
523
         }
524
         int size()
525
526
         {
527
             return currentSize;
528
         }
529
530
         void printAll(std::ostream& output)
531
532
             if (empty())
533
             {
534
                 throw(errorType::EMPTY, errorType::errorString[EMPTY], 5);
535
536
             output << " Index | LABEL | Key | Value" << '\n'
537
                 << "
538
                 << '\n';
539
540
541
             for (int i = 0; i < capacity; i++)</pre>
542
543
                 output << std::right
                   << " [" << std::setw(5) << i << "] | ";</pre>
544
545
                 switch (map[i].label)
546
                 case EMPTY:
547
                     output << "EMPTY |";
548
549
                     break;
550
                 case FULL:
551
552
                     output << "FULL |";
553
                     break;
554
555
                 case AVAILABLE:
556
                     output << "AVAIL |";
557
                     break;
558
559
                 output << ' ' << std::setw(4) << map[i].key << " |";
560
561
562
                 output << std::left</pre>
                     << ' ' << map[i].value
563
                     << '\n';
564
```

```
...shingAlgorithms\CS1D-AS7-HashingAlgorithms\ArrayHashMap.h
```

```
12
```

```
565
566
             output << "\n\n";</pre>
567
        }
568 };
569
570
571
572
573
574
575
576 #endif //!_ARRAYHASHMAP_H_
577
578
579
580
581
```