```
 1  /*****************************************************************************
 2   * AUTHOR           : Nick Reardon
 3   * Assignment #8    : Skip Lists
 4   * CLASS            : CS1D
 5   * SECTION          : MW - 2:30p
 6   * DUE DATE         : 03 / 25 / 20
 7   *****************************************************************************/
 8  #ifndef _SKIPLIST_H_
 9  #define _SKIPLIST_H_
10  #include <iostream>
11  #include <iomanip>
12  #include <vector>
13  #include <stdlib.h>
14  #include <time.h>
15  #include <string>
16
17
18  class SkipList
19  {
20
21  private:
22
23      struct Node
24      {
25          int key;
26          std::string value;
27
28          std::vector<Node*> next;
29
30          Node(int k, const std::string& v, int level) :
31              key(k), value(v), next(level, nullptr)
32          {
33          }
34      };
35
36      Node* head;
37      Node* tail;
38
39      int currentSize;
40
41      float probability;
42      int maxLevel;
43
44
45      int randomLevel() const
46      {
47          int newLevel = 1;
48          while ( ( (double)std::rand() / RAND_MAX) < probability && (newLevel <
              maxLevel) )
49          {
50              newLevel++;
51          }
```

```
52          return newLevel;
53      }
54
55
56      static int nodeLevel(const Node* v)
57      {
58          return v->next.size();
59      }
60
61
62      static Node* makeNode(int key, std::string val, int level)
63      {
64          return new Node(key, val, level);
65      }
66
67      Node* lower_bound(int searchKey) const
68      {
69          Node* node = head;
70
71          for (unsigned int i = nodeLevel(head); i-- > 0;)
72          {
73              while (node->next[i]->key < searchKey)
74              {
75                  node = node->next[i];
76              }
77          }
78          return node->next[0];
79      }
80
81
82      std::vector<SkipList::Node*> getPreviousNodes(int searchKey) const
83      {
84          std::vector<Node*> result(nodeLevel(head), nullptr);
85          Node* node = head;
86
87          for (unsigned int i = nodeLevel(head); i-- > 0;)
88          {
89              while (node->next[i]->key < searchKey)
90              {
91                  node = node->next[i];
92              }
93              result[i] = node;
94          }
95          return result;
96      }
97
98
99  protected:
100
101
102 public:
103     SkipList(float newProbability = 0.5f, int newMaxLevel = 16)
```

```
104     {
105         probability = newProbability;
106         maxLevel = newMaxLevel;
107 
108         std::srand(time(0));
109 
110         int headKey = std::numeric_limits<int>::min();
111         head = new Node(headKey, "head", maxLevel);
112 
113         int tailKey = std::numeric_limits<int>::max();
114         tail = new Node(tailKey, "tail", maxLevel);
115 
116         std::fill(head->next.begin(), head->next.end(), tail);
117     }
118 
119 
120     ~SkipList()
121     {
122         Node* node = head;
123         while (node->next[0])
124         {
125             Node* temp = node;
126             node = node->next[0];
127             delete temp;
128         }
129         delete node;
130     }
131 
132 
133     void print() const
134     {
135 
136 
137         Node* list = head->next[0];
138 
139         std::cout << "\n----------\n";
140         std::cout << std::left;
141         while (list != tail)
142         {
143             std::cout << "value: " << std::setw(20) << list->value
144                 << "key: " << std::setw(10) << list->key
145                 << "level: " << std::setw(10) << nodeLevel(list);
146 
147             for (int i = 0; i < nodeLevel(list); i++)
148             {
149                 std::cout << "[]";
150             }
151 
152             list = list->next[0];
153 
154 
155             std::cout << "\n";
```

```
156         }
157         std::cout << "\n----------\n";
158 
159     }
160 
161 
162     void printLevels() const
163     {
164         std::cout << "\n----------\n";
165 
166         Node* list;
167         for (int i = maxLevel - 1; i >= 0; i--)
168         {
169             list = head->next[i];
170 
171             std::cout << "level: " << std::setw(10) << i << '\n';
172 
173             while (list != tail)
174             {
175                 std::cout << std::string(10, ' ') << "value: " << std::setw(30)
                      << list->value
176                       << "key: " << std::setw(10) << list->key;
177 
178                 list = list->next[i];
179 
180 
181                 std::cout << "\n";
182             }
183         }
184         std::cout << "\n----------\n";
185     }
186 
187 
188     std::string* find(int searchKey) const
189     {
190         std::string* result = nullptr;
191         if (auto node = lower_bound(searchKey))
192         {
193             if (node->key == searchKey && node != tail)
194             {
195                 result = &(node->value);
196             }
197         }
198         return result;
199     }
200 
201 
202 
203     void insert(int searchKey, const std::string& newValue)
204     {
205         std::vector<Node*> prevNodes = getPreviousNodes(searchKey);
206 
```

```
207         int newNodeLevel = randomLevel();
208         Node* newNode = makeNode(searchKey, newValue, newNodeLevel);
209 
210         for (int i = 0; i < newNodeLevel; ++i)
211         {
212             newNode->next[i] = prevNodes[i]->next[i];
213             prevNodes[i]->next[i] = newNode;
214         }
215 
216         currentSize++;
217     }
218 
219 
220     void erase(int searchKey)
221     {
222         std::vector<Node*> prevNodes = getPreviousNodes(searchKey);
223 
224         Node* node = prevNodes[0]->next[0];
225         if (node->key != searchKey || node == tail)
226         {
227             return;
228         }
229 
230         for (size_t i = 0; i < nodeLevel(node); ++i)
231         {
232             prevNodes[i]->next[i] = node->next[i];
233         }
234         delete node;
235 
236         currentSize--;
237 
238     }
239 
240 
241     bool empty() const
242     {
243         return (currentSize == 0);
244     }
245 
246 
247     int size() const
248     {
249         return currentSize;
250     }
251 
252 
253 };
254 
255 
256 
257 
258 
```

```
259  #endif // !_SKIPLIST_H_
```