```cpp
/*****************************************************************************
 * ASN #8 - Skip Lists
 * ---------------------------------------------------------------------------
 * This program will output the class heading
 * ---------------------------------------------------------------------------
 *
 *****************************************************************************/

#include <iostream>
#include <bits/stdc++.h>
using namespace std;

    class Node
    {
    public:
        int key;            // value to hold for the key of the dictionary
        string value;       // value to hold for the string of the dictionary

        Node **forward;     // value to hold the skip list level
        Node(int, string, int);
    };

    /*****************************************************************************
    * CONSTRUCTOR Node
    * ___________________________________________________________________________
    *   This constructor is to initialize the key and value and develop the
    *   skip list level. The class variable forward is set equal to a new
    *   class variable with the condition of skip list level incremented. The
    *   function memset is used to set the first num bytes of the block of
    *   memory pointed by ptr to the specified value.
    * ___________________________________________________________________________
    *
    * ***************************************************************************/
    Node::Node(int key, string, int level)
    {
        this->key = key;

        value = "";

        forward = new Node*[level + 1];

        memset(forward, 0, sizeof(Node*)*(level+1));
    };

    class SkipList
    {
        int MAXLVL;         // value to hold the maximum level of the skip list
        float fracNode;     // value to hold the fraction of the nodes
        string input;       // value to hold the string of the dictionary
        int level;          // value to hold the current level of the skip list
        Node * header;      // value to hold the header of the node
        int count;          // value to hold the size of the skip list

    public:

        SkipList(int, string, float);
        int randomLevel();
```

```
58          Node * createNode(int, string, int);
59          int size();
60          void empty();
61          void put(int, string);
62          void erase(int, string);
63          void find(int, string);
64          void displayList();
65
66      };
67
68      /*************************************************************************
69      * CONSTRUCTOR SkipList
70      * ---------------------------------------------------------------------------
71      *   This constructor is to initialize the max level of the skip list, value
72      *   for the dictionary and the fraction of the nodes. The current level
73      *   of the skip list and the size of the skip list is also initialized.
74      *   Thus, the header of the node is set equal to a new node with the
75      *   condition of -1, dictionary value and the fraction of the node.
76      * ---------------------------------------------------------------------------
77      *
78      * ************************************************************************/
79      SkipList::SkipList(int MAXLVL, string input, float fracNode)
80      {
81          this->MAXLVL = MAXLVL;
82          this->fracNode = fracNode;
83          input = "";
84          level = 0;
85          count = 0;
86
87          header = new Node(-1, input, MAXLVL);
88      };
89
90      /*************************************************************************
91      * FUNCTION randomLevel
92      * ---------------------------------------------------------------------------
93      *   This function is to create a random level for the node. To do so, the
94      *   random level is set equal to a random float value divided by the
95      *   random max level for the skip list. Thus, the level of the tree
96      *   is set equal to 0. Therefore, while the random level is less than
97      *   the fraction of the node and level of the skip list is less than
98      *   the max level of the skip list, the level of the skip list will
99      *   increment and the random level will be set equal to a random
100     *   float value divided by the random max level for the skip list. Lastly,
101     *   the level of the skip list will return.
102     * ---------------------------------------------------------------------------
103     *
104     * ************************************************************************/
105     int SkipList::randomLevel()
106     {
107         float random = (float)rand()/RAND_MAX;
108         int lvl = 0;
109         while(random < fracNode && lvl < MAXLVL)
110         {
111             lvl++;
112             random = (float)rand()/RAND_MAX;
113         }
114         return lvl;
```

```
115     };
116
117     /*************************************************************************
118     * FUNCTION createNode
119     * ____________________________________________________________________________
120     *   This function is to create a new node for the skip list. The node
121     *   class variable pointer n is set equal to a new node with the condition
122     *   of the dictionary key and value with the level of the skip list. Lastly,
123     *   the class variable n is returned.
124     * ____________________________________________________________________________
125     *
126     * ************************************************************************/
127     Node * SkipList::createNode(int key, string input, int level)
128     {
129         Node * n = new Node(key,input, level);
130         return n;
131     };
132
133     /*************************************************************************
134     * FUNCTION size
135     * ____________________________________________________________________________
136     *   This function is to return the overall size of the skip list.
137     * ____________________________________________________________________________
138     *
139     * ************************************************************************/
140     int SkipList::size()
141     {
142         return count;
143     }
144
145     /*************************************************************************
146     * FUNCTION empty
147     * ____________________________________________________________________________
148     *   This function is to return the skip list size as 0.
149     * ____________________________________________________________________________
150     *
151     * ************************************************************************/
152     void SkipList::empty()
153     {
154         if(count == 0)
155         {
156             cout << "Empty!" << endl;
157
158         }
159         else
160         {
161             cout << "Not Empty!" << endl;
162         }
163
164     }
165
166     /*************************************************************************
167     * FUNCTION put
168     * ____________________________________________________________________________
169     *   This function is to insert the given key into the skip list. Firstly,
170     *   the function creates an update array which is initialized. The for
171     *   loop starts from the highest level of the skip list which then moves
```

```
172     *   the current pointer forward under the condition where the key
173     *   is greater than the key of the node next to the current pointer. If this
174     *   is not the case, then it will default to inserting the current pointer
175     *   of the list in the update array and ultimately move the pointer one
176     *   level down. If it reaches level 0, then the forward pointer is pointed
177     *   to the right. If the current pointer is NULL then that will indicate
178     *   the end of the level. If the random level is greater than the list's
179     *   current maximum level, then the update value will be initialized to the
180     *   header pointer. Lastly, the node will output.
181     * ------------------------------------------------------------------------
182     *
183     * ************************************************************************/
184    void SkipList::put(int key, string otherKey)
185    {
186        Node * current = header;
187
188        Node * update[MAXLVL + 1];
189        memset(update, 0, sizeof(Node*)*(MAXLVL + 1));
190
191        for(int i = level; i >=0; i--)
192        {
193            while(current->forward[i] != NULL && current->forward[i]->key < key)
194            {
195                current = current->forward[i];
196            }
197            update[i] = current;
198        }
199
200        current = current->forward[0];
201
202        if(current == NULL || current->key != key)
203        {
204            int rlevel = randomLevel();
205
206            if(rlevel > level)
207            {
208                for(int i = level + 1;i < rlevel + 1; i++)
209                {
210                    update[i] = header;
211                }
212                level = rlevel;
213
214            }
215
216            Node * n = createNode(key, otherKey, rlevel);
217
218            for(int i = 0; i <= rlevel; i++)
219            {
220                n->forward[i] = update[i]->forward[i];
221                update[i]->forward[i] = n;
222            }
223            n ->value = otherKey;
224            cout << "Successfully Inserted key " << key << " " << otherKey
225                 << "\n";
226
227        }
228
```

```
229         if (key > count)
230         {
231             count = key;
232         }
233 
234     };
235 
236     /*************************************************************************
237     * FUNCTION erase
238     * ___________________________________________________________________________
239     *   This function is to delete the key from the skip list.  Firstly,
240     *   the function creates an update array which is initialized. The for
241     *   loop starts from the highest level of the skip list which then moves
242     *   the current pointer forward under the condition where the key
243     *   is greater than the key of the node next to the current pointer. If this
244     *   is not the case, then it will default to inserting the current pointer
245     *   of the list in the update array and ultimately move the pointer one
246     *   level down. If it reached level 0, then the forward pointer will point
247     *   to the right. If the current node is the desired node from the user,
248     *   then it will start from the lowest level of the skip list and
249     *   rearrange the pointers. If the key is at the level increment number,
250     *   then the next node is not the desired node. Lastly, it will remove
251     *   the key from the skip list.
252     * ___________________________________________________________________________
253     *
254     * ************************************************************************/
255     void SkipList::erase(int key, string otherKey)
256     {
257         Node * current = header;
258 
259         Node * update[MAXLVL + 1];
260         memset(update, 0, sizeof(Node*)*MAXLVL + 1);
261 
262         for(int i = level; i >= 0; i--)
263         {
264             while(current->forward[i] != NULL && current->forward[i]->key < key)
265             {
266                 current = current->forward[i];
267             }
268             update[i] = current;
269         }
270 
271         current = current->forward[0];
272 
273         if(current != NULL and current->key == key)
274         {
275             for(int i = 0; i >= 0; i++)
276             {
277                 if(update[i]->forward[i] != current)
278                 {
279                     break;
280                 }
281                 update[i]->forward[i] = current->forward[i];
282 
283             }
284 
285             while(level > 0 && header->forward[level] == 0)
```

```
286             {
287                 level--;
288             }
289 
290             cout << "Successfully delete key " << key << " " << otherKey
291                  << "\n";
292 
293         }
294         count--;
295     };
296 
297     /*************************************************************************
298     * FUNCTION find
299     * ___________________________________________________________________________
300     *   This function is to search for the key from the skip list. The for
301     *   loop starts from the highest level of the skip list which then moves
302     *   the current pointer forward under the condition where the key
303     *   is greater than the key of the node next to the current pointer. If this
304     *   is not the case, then it will default to inserting the current pointer
305     *   of the list in the update array and ultimately move the pointer one
306     *   level down. If it reached level 0, then the forward pointer will point
307     *   to the right. Lastly, it will find the key.
308     * ___________________________________________________________________________
309     *
310     * ************************************************************************/
311     void SkipList::find(int key, string otherKey)
312     {
313         Node * current = header;
314 
315         for(int i = level; i >= 0; i--)
316         {
317             while(current->forward[i] && current->forward[i]->key < key)
318             {
319                 current = current->forward[i];
320             }
321         }
322 
323         current = current->forward[0];
324 
325         if(current and current->key == key)
326         {
327             cout << "Found key: " << key << " " << otherKey << "\n";
328         }
329         else
330         {
331             cout << key << " " << otherKey << " does not exist on"
332                  << " the skip list!" << "\n";
333         }
334     };
335 
336     /*************************************************************************
337     * FUNCTION displayList
338     * ___________________________________________________________________________
339     *   This function is to output the skip list in its entirety. The header
340     *   of the node points the increment number of forward. Lastly, this will
341     *   output the dictionary key.
342     * ___________________________________________________________________________
```

```
343     *
344     * ************************************************************************/
345     void SkipList::displayList()
346     {
347         cout<<"\n*****Skip List*****"<<"\n";
348         for(int i=0;i<=level;i++)
349         {
350             Node *node = header->forward[i];
351             cout << "Level "<< i << ": ";
352             while(node != NULL)
353             {
354                 cout << node->key << " " << node->value << " ";
355                 node = node->forward[i];
356             }
357             cout << "\n";
358         }
359     };
360
361
362
363
364
365 int main()
366 {
367     /*****************************************************************************
368     * CONSTANT
369     * ---------------------------------------------------------------------------
370     * OUTPUT - USED FOR CLASS HEADING
371     * ---------------------------------------------------------------------------
372     * PROGRAMMER : Wesley Chok
373     * CLASS      : CS 1D
374     * SECTION    : MW 2:30p - 4:50p
375     * ASN_NUM    : 8
376     * ASN_NAME   : Skip Lists
377     *****************************************************************************/
378
379     srand((unsigned)time(0));
380
381     SkipList address(18, "San Clemente", 0.5);
382
383     address.put(41, "Mission Viejo");
384     address.put(22, "Laguna Niguel");
385     address.put(44, "Irvine");
386     address.erase(18, "San Clemente");
387     address.put(58, "Lake Forest");
388     address.put(32, "San Diego");
389     address.put(49, "Anaheim");
390     address.erase(58, "Lake Forest");
391     address.put(31, "Los Angeles");
392     address.put(17, "Orange");
393     address.put(72, "Palms Springs");
394     address.put(41, "Riverside");
395     address.erase(49, "Anaheim");
396     address.put(19, "Brea");
397     address.put(60, "Santa Ana");
398     address.put(35, "Tustin");
399     address.put(103, "Oceanside");
```

```
400     address.put(11, "La Jolla");
401     address.put(18, "Del Mar");
402     address.put(22, "Alista Viejo");
403     address.put(49, "Laguna Beach");
404     address.erase(41, "Riverside");
405     address.put(42, "Vista");
406     address.put(49, "San Diego");
407     address.put(99, "San Juan");
408     address.put(29, "Dana Point");
409     address.put(88, "El Segundo");
410     address.put(41, "San Clemente");
411     address.put(62, "Laguna Hills");
412 
413     address.displayList();
414 
415     address.find(62, "Laguna Hills");
416     address.find(105, "Kentucky");
417 
418     cout << "Size: " << address.size() << endl;
419 
420     address.empty();
421 
422 
423 
424     return 0;
425 }
426 
```