```
// Lina Kang
// CS1D MW 2:30 - 5:00 PM
// Assignment 8 - Skip List
// This program exercises the skip list method through insertions,
// deletions, searches, and other standard functions of skip list

/*
Lina Kang
CS1D MW 2:30 - 5:00 PM
Assignment 8 - Skip List
This program exercises the skip list method through insertions,
deletions, searches, and standard functions of skip list

Print all levels and items:
----------------------------
Level: 0
Laguna Niguel 18

Level: 1
Laguna Niguel 18

Level: 2
Laguna Niguel 18

Level: 3
Laguna Niguel 18 -> Aliso Viejo 22 -> Tustin 35 -> San Juan 99

Level: 4
Laguna Niguel 18 -> Aliso Viejo 22 -> Dana Point 29 -> San Diego 32 -> Tustin 35 -> San Juan
99

Level: 5
Laguna Niguel 18 -> Aliso Viejo 22 -> Dana Point 29 -> San Diego 32 -> Tustin 35 -> Irvine 44
-> Anaheim 49 -> Laguna Beach 49 -> San Diego 49 -> San Juan 99

Level: 6
La Jolla 11 -> Orange 17 -> Laguna Niguel 18 -> Del Mar 18 -> Brea 19 -> San Clemente 22 ->
Aliso Viejo 22 -> Dana Point 29 -> Los Angeles 31 -> San Diego 32 -> Tustin 35 -> San Clemente
41 -> Vista 42 -> Irvine 44 -> Anaheim 49 -> Laguna Beach 49 -> San Diego 49 -> Santa Ana 60
-> Laguna Hills 62 -> El Segundo 88 -> San Juan 99 -> Oceanside 103

Print values and keys:
----------------------------

La Jolla 11
Orange 17
Laguna Niguel 18
Del Mar 18
Brea 19
San Clemente 22
Aliso Viejo 22
Dana Point 29
Los Angeles 31
San Diego 32
Tustin 35
San Clemente 41
Vista 42
```

```
Irvine 44
Anaheim 49
Laguna Beach 49
San Diego 49
Santa Ana 60
Laguna Hills 62
El Segundo 88
San Juan 99
Oceanside 103


Find values and keys:
----------------------------
Found: 49 Anaheim

Found: 32 San Diego

Not found: -1


Test empty function:
----------------------------
Testing this skip list: The list is NOT empty
Testing a completely NEW skip list: The list is empty
Testing NEW skip list with 1 item added & removed immediately: The list is empty


Test size function:
----------------------------
Size of "skip": 22
Size of "skipEmpty": 0
Size of "skipEmpty2": 0
*/

#include "header.h"

int main()
{
    skipList skip;

    skip.generateRandom();

    // ---------------- Test "put" and "erase" function ------------------//

    skip.put(18, "Laguna Niguel"); //
    skip.put(41, "Mission Viejo");
    skip.put(22, "San Clemente"); //
    skip.put(44, "Irvine"); //
    skip.erase(41);
    skip.put(58, "Lake Forest");
    skip.put(32, "San Diego"); //
    skip.put(49, "Anaheim"); //
    skip.erase(58);
    skip.put(31, "Los Angeles"); //
    skip.put(17, "Orange"); //
    skip.put(72, "Palms Springs");
    skip.put(41, "Riverside");
```

```
110     skip.erase(72);
111     skip.put(19, "Brea"); //
112     skip.put(60, "Santa Ana"); //
113     skip.put(35, "Tustin"); //
114     skip.put(103, "Oceanside"); //
115     skip.put(11, "La Jolla"); //
116     skip.put(18, "Del Mar"); //
117     skip.put(22, "Aliso Viejo"); //
118     skip.put(49, "Laguna Beach"); //
119     skip.erase(41);
120     skip.put(42, "Vista"); //
121     skip.put(49, "San Diego"); //
122     skip.put(99, "San Juan"); //
123     skip.put(29, "Dana Point"); //
124     skip.put(88, "El Segundo"); //
125     skip.put(41, "San Clemente"); //
126     skip.put(62, "Laguna Hills"); //
127
128     cout << "Lina Kang\n"
129             "CS1D MW 2:30 - 5:00 PM\n"
130             "Assignment 8 - Skip List\n"
131             "This program exercises the skip list method through insertions,\n"
132             "deletions, searches, and standard functions of skip list\n\n";
133
134     // ---------------- Output each level of the skip list ------------------//
135
136     cout << "Print all levels and items: \n"
137             "----------------------------\n";
138
139     skip.printLevels();
140
141     // -------------- Output the dictionary keys and values -----------------//
142
143     cout << "\n\nPrint values and keys: \n"
144             "----------------------------\n\n";
145
146     skip.printDictionary();
147
148     // --------------------- Test "find" function ------------------------//
149
150     cout << "\n\nFind values and keys: \n"
151             "----------------------------";
152
153     int key = 49;
154     Node * found = skip.find(key);
155
156     if(found != NULL)
157         cout << "\nFound: " << found->key << " " << found->name << endl;
158     else
159         cout << "\nNot found: "<< key;
160
161     key = 32;
162     found = skip.find(key);
163
164     if(found != NULL)
165         cout << "\nFound: " << found->key << " " << found->name << endl;
166     else
```

```
167         cout << "\nNot found: " << key;
168 
169     key = -1;
170     found = skip.find(key);
171 
172     if(found != NULL)
173         cout << "\nFound: " << found->key << " " << found->name << endl;
174     else
175         cout << "\nNot found: " << key;
176 
177     // --------------------- Test "empty" function ------------------------//
178 
179     cout << "\n\n\nTest empty function: \n"
180             "----------------------------\n";
181 
182 
183     cout << "Testing this skip list: ";
184     if(skip.empty())
185         cout << "The list is empty" << endl;
186     else
187         cout << "The list is NOT empty" << endl;
188 
189 
190     skipList skipEmpty;
191 
192     cout << "Testing a completely NEW skip list: ";
193     if(skipEmpty.empty())
194         cout << "The list is empty" << endl;
195     else
196         cout << "The list is NOT empty" << endl;
197 
198 
199     skipList skipEmpty2;
200     skipEmpty2.put(100, "Testing");
201     skipEmpty2.erase(100);
202 
203     cout << "Testing NEW skip list with 1 item added & removed immediately: ";
204     if(skipEmpty2.empty())
205         cout << "The list is empty" << endl;
206     else
207         cout << "The list is NOT empty" << endl;
208 
209     // --------------------- Test "size" function --------------------------//
210 
211     cout << "\n\nTest size function: \n"
212             "----------------------------";
213 
214     cout << "\nSize of \"skip\": " << skip.getSize();
215     cout << "\nSize of \"skipEmpty\": " << skipEmpty.getSize();
216     cout << "\nSize of \"skipEmpty2\": " << skipEmpty2.getSize();
217 
218 }
219 
220 #ifndef HEADER_H_
221 #define HEADER_H_
222 
223 #include <iostream>
```

```cpp
#include <iomanip>
#include <vector>
#include <time.h>

using namespace std;

const int MIN = 0;      // sentinel values
const int MAX = 150;    // sentinel values

struct Node
{
    int key;
    string name;

    Node * next;
    Node * below;
};


class skipList
{
public:
    skipList();

    void put(int, string);
    Node* find(int);
    void erase(int);
    bool empty();
    int getSize();

    void printLevels();
    void printDictionary();
    void generateRandom();  // for randomized algorithm

private:
    int size;

    Node * top;             // points to top-left sentinel value
    Node * topLast;         // points to top-right sentinel value

    int randomNumbers[50];  // for randomized algorithm
    int randomCurr;         // for randomized algorithm

};

// skiplist constructor
skipList::skipList()
{
    top = new Node;
    top->key = MIN;
    top->name = "MIN";
    top->below = NULL;

    topLast = new Node;
    topLast->key = MAX;
    topLast->name = "MAX";
    topLast->below = NULL;
```

```
281
282     top->next = topLast;
283     topLast->next = NULL;
284
285     randomCurr = 0;
286     size = 0;
287 }
288
289 // insert an element into the skiplist
290 void skipList::put(int key, string name)
291 {
292     // list of nodes before the inserting item to update their "next" pointers
293     vector<Node *> nodesToUpdate;
294
295     Node * current = top;
296
297     // find the most appropriate location to insert the element
298     while(current != NULL)
299     {
300         if(current->next->key > key)
301         {
302             nodesToUpdate.push_back(current);
303             if(current->below == NULL)
304                 break;
305             else
306                 current = current->below;
307         }
308         else
309         {
310             current = current->next;
311         }
312     }
313
314 //  srand(time(NULL));
315
316     // insert the element and update all "next" pointers that stand before it
317     int random = 1;                         // random starts with 1 in order-
318     int i = nodesToUpdate.size() - 1;       // -to insert the item at least once
319     while(random == 1 && i >= 0)
320     {
321         Node * tempNext = nodesToUpdate[i]->next;
322         nodesToUpdate[i]->next = new Node;
323         nodesToUpdate[i]->next->key = key;
324         nodesToUpdate[i]->next->name = name;
325         nodesToUpdate[i]->next->next = tempNext;
326
327         if(i == nodesToUpdate.size() -1)
328             nodesToUpdate[i]->next->below = NULL;
329         else
330             nodesToUpdate[i]->next->below = nodesToUpdate[i+1]->next;
331
332         i--;
333         random = randomNumbers[randomCurr]; // get 0or1 from the randomized list
334         randomCurr++;
335     }
336
337     // If current level of element has exceeded beyond highest level AND
```

```
338     // random number is still "heads", continue to add until "tails" or 0 comes
339     current = nodesToUpdate[0]->next;
340     while(random == 1)
341     {
342         Node * tempBelow = current;
343         current = new Node;
344 
345         Node * topCurrent = new Node;
346         topCurrent->below = top;
347         topCurrent->next = current;
348         topCurrent->key = MIN;
349         topCurrent->name = "MIN";
350 
351         Node * topLastCurrent = new Node;
352         topLastCurrent->below = topLast;
353         topLastCurrent->next = NULL;
354         topLastCurrent->key = MAX;
355         topLastCurrent->name = "MAX";
356 
357         current->next = topLastCurrent;
358         current->below = tempBelow;
359         current->key = key;
360         current->name = name;
361 
362         top = topCurrent;
363         topLast = topLastCurrent;
364 
365         random = randomNumbers[randomCurr];
366         randomCurr++;
367     }
368     size++;
369 }
370 
371 // returns a Node that contains the "key"
372 Node* skipList::find(int key)
373 {
374     Node * current = top;
375     bool found = false;
376 
377     while(current != NULL)
378     {
379         if(current->next->key > key)        // go down a level
380         {
381             if(current->below == NULL)
382                 break;
383             else
384                 current = current->below;
385         }
386         else if(current->next->key == key)  // if key is found
387         {
388             found = true;
389             break;
390         }
391         else                                // keep going to the right
392         {
393             current = current->next;
394         }
```

```
395     }
396
397     if(found)
398         return current->next;
399     return NULL;
400 }
401
402 // removes an element with the indicated key
403 void skipList::erase(int key)
404 {
405     vector<Node *> nodesToUpdate;           // list of nodes that stand right
406                                             // before the key element(s)
407                                             // for the purpose of updating
408                                             // their "next" pointers
409     Node * current = top;
410
411     while(current != NULL)
412     {
413         if(current->next->key > key)        // go down a level
414         {
415             if(current->below == NULL)
416                 break;
417             else
418                 current = current->below;
419         }
420         else if(current->next->key == key)  // if key is found
421         {
422             nodesToUpdate.push_back(current);
423             if(current->below == NULL)
424                 break;
425             else
426                 current = current->below;
427         }
428         else                                // keep going to the right
429         {
430             current = current->next;
431         }
432     }
433
434     // update "next" pointers of nodes that stood before the deleted element
435     int i = nodesToUpdate.size() - 1;
436     while(i >= 0)
437     {
438         Node * tempNext = nodesToUpdate[i]->next->next;
439         Node * tempDelete = nodesToUpdate[i]->next;
440
441         nodesToUpdate[i]->next = tempNext;
442
443         i--;
444         delete tempDelete;
445     }
446     size--;
447 }
448
449 // determines whether the list is empty or not
450 bool skipList::empty()
451 {
```

```
452     if(size <= 0)
453     {
454         return true;
455     }
456     return false;
457 }
458 
459 // returns the number of elements in the skip list
460 int skipList::getSize()
461 {
462     return size;
463 }
464 
465 // prints the skiplist by levels
466 void skipList::printLevels()
467 {
468     Node * current = top->next;
469     Node * topCurrent = top;
470 
471     int level = 0;
472 
473     cout << left << "Level: " << level << endl;
474     while(current != NULL)
475     {
476         cout << current->name << " " << current->key;
477 
478         if(current->next->key == 150 && current->below == NULL)
479         {
480             break;
481         }
482         if(current->next->key == 150)
483         {
484             current = topCurrent->below;
485             topCurrent = current;
486             current = current->next;
487             level++;
488             cout << endl << endl << "Level: " << level << endl;
489         }
490         else
491         {
492             current = current->next;
493             cout << " -> ";
494         }
495     }
496 }
497 
498 // prints the values and the keys within the skiplist
499 void skipList::printDictionary()
500 {
501     Node * current = top;
502 
503     while(current->below != NULL)
504     {
505         current = current->below;
506     }
507     current = current->next;
508     while(current->next != NULL)
```

```
509     {
510         cout << current->name << " " << current->key << endl;
511         current = current->next;
512     }
513 }
514 
515 // generate a list of randomized 0's and 1's
516 // (using srand(time(NULL)) and rand() in one go tends to show same results)
517 //  everytime)
518 void skipList::generateRandom()
519 {
520     srand(time(NULL));
521     for(int i = 0; i < 50; i++)
522     {
523         randomNumbers[i] = rand() % 2;
524     }
525 }
526 
527 #endif /* HEADER_H_ */
528 
```