

CSC 402 - Data Structures II, Fall, 2010

Assignment #2, Skiplists in C++, Dr. Dale E. Parson, DUE 11:59 PM Oct. 21

I have already laid out the class declaration for C++ class `map_skiplistimpl` that implements interface `map_interface`, the same interface implemented by your hash table in assignment 1. You need to perform the following steps in order to start your work.

```
cd ~/AdvDataStructures
cp ~parson/skiplistsASSN2.zip skiplistsASSN2.zip
unzip skiplistsASSN2.zip
cd ./skiplists
gmake clean build
```

At this point the program compiles OK, but attempting `gmake test` would core dump because many of the methods including the constructor for class `map_skiplistimpl` need to be defined.

File `map_skiplistimpl.h` is completed and should not be modified. All of your changes go into file `map_skiplistimpl.cxx` in the methods tagged with “STUDENT” comment headers. I have started a few lines of code for some of those methods. In such cases you should leave my code as it is, starting your code immediately after the STUDENT comment. We will go over this in class. The source code for `map_skiplistimpl.h` and the initial source code for `map_skiplistimpl.cxx` appear below.

When your tests works correctly within `gmake test`, execute `gmake turnitin` before the end of October 21. For this example your output files should match my reference files on bill because I am seeding the random number generator with a fixed seed (12345), so we should get identical distributions of key -> value mappings within the skiplists. You may get different results on other machines because of differences in the `srandom()` and `random()` library functions, so make sure to perform your testing on bill.

`/export/home/faculty/parson/AdvDataStructures/skiplists/map_skiplistimpl.h`

```
1 /* map_skiplistimpl.h -- Concrete implementation of map_interface
2 using a skiplist, see William Pugh's
3 A Probabilistic Alternative to Balanced Trees, CACM, 1990
4
5 CSC402, Fall, 2010, Dr. Dale Parson
6
7 A prototype implementation is in my file skiplists.py.
8 You can use that as executable pseudocode.
9 */
10
11 #ifndef MAP_SKIPLISTIMPL_H
12 #define MAP_SKIPLISTIMPL_H
13 #include "map_interface.h"
```

```
14 #include <list>
15
16 class map_skiplistimpl_iterator ; // forward declaration
17
18 /* Class: map_skiplistimpl
19
20 Concrete class implementing map_interface by using a skiplist.
21 */
22 class map_skiplistimpl : public map_interface {
23
24 friend class map_skiplistimpl_iterator ;
25
26 private:
27 struct node { // Implementation node that occupies a skiplist.
28 string key ; // This could be a template type that supports <,,=,>
29 int value ; // This could be any template value type.
30 node **forward ; // Variant length array of skiplist pointers.
31 int level ; // key's maximal level in skiplist
32 // Note: There are level+1 elements in forward, [0]..[level]
33 node(const string &keyparam, const int &valueparam, int levelparam) {
34 key = keyparam ;
35 value = valueparam ;
36 level = levelparam ;
37 forward = new node * [level+1] ;
38 for (int i = 0 ; i <= level ; i++) {
39 forward[i] = 0 ; // Initialize with NULLs
40 }
41 }
42 ~node() {
43 delete [] forward ;
44 }
45 };
46 int maxlevel ; // See Pugh's paper.
47 node **forward ; // Variant length array of skiplist pointers.
48 int *listlength ; // Variant length array of list lengths.
49 node *NIL ; // End-of-list sentinel marker.
50 int level ; // Pugh uses 1, C++, Java and Python start at 0.
51 int degree ; // Degree of equivalent tree, e.g., 2 for binary tree.
52 double probability ; // Inverse of degree.
53 bool isgrow ; // Whether to grow maxlevel in a crowded tree.
54 // Pugh's original paper does not support growth of maxlevel.
55 // This implementation does. See the Python pseudocode.
56 // See "man random" -- use the functions "void random(unsigned int seed);"
57 // and "long random(void);" as the C random number generator.
58 // A seed value of -1 means "no seed," else invoke srandom.
59
```

```

60 // map_skiplistimpl(); // 0 parameter constructor is not implemented.
61
62
63 // iterators tracks the current, non-destructed iterators
64 list<map_skiplistimpl_iterator *> iterators ;
65 void markIteratorsAsDone() ; // mark all my iterators as done
66
67 // Return a random level 0 through maxlevel-1, with probability
68 // 1 / degree of advancing at each level, yielding
69 // 1 / degree^level for any level. See Pugh's paper.
70 int randomLevel();
71
72 public:
73 /**
74  * Construct a skiplist with the given maxlevel; 4 matches Pugh's paper.
75  * Degree must be a small int > 1; 1/degree gives level probability.
76  * An integer seed parameter seeds the random number generator;
77  * a seed argument of -1 says to ignore the seed value.
78  * The isgrow parameter when true allows level growth above maxlevel.
79  * Pugh's paper says, "Initialization"
80  * An element NIL is allocated and given a key greater than any
81  * legal key. All levels of all skip lists are terminated with NIL.
82  * A new list is initialized so that the the level of the list is equal
83  * to 1 and all forward pointers of the lists header point to NIL."
84  * This implementation indexes 0..maxlevel-1 as in Python, C++ and Java.
85  * This implementation puts sentinel NIL node obejects at the front
86  * and the back of each linked list.
87  */
88 map_skiplistimpl(int maxlevel=4, int degree=2, int seed=-1,
89                 bool isgrow=false);
90 virtual ~map_skiplistimpl();
91
92 /* Operation: put, see map_interface.h.
93 */
94 virtual
95 void
96 put(const string &key, const int &value) ;
97
98 /* Operation: remove, see map_interface.h.
99 */
100 virtual
101 bool
102 remove(const string &key) ;
103
104 /* Operation: get, see map_interface.h.
105 */

```

```

106 virtual
107 int
108 get(const string &key, bool &isvalid) const ;
109
110 /* Operation: size, see map_interface.h.
111 */
112 virtual
113 int
114 size() const ;
115
116 /* Operation: print, see map_interface.h.
117 The skiplistimpl supports only forward
118 walking, prints key -> value in key sorted order.
119 */
120 void
121 print(bool forward=true) const ;
122
123 /* dump operates like print, except that it dumps the entire skiplist.
124 */
125 void
126 dump() const ;
127
128 /* Operation: make_new_map_interface_iterator,
129 see map_interface.h.
130 Client code must eventually delete an iterator returned by
131 make_new_map_interface_iterator.
132 */
133 virtual
134 map_interface_iterator *
135 make_new_map_interface_iterator() ;
136
137 };
138
139 /* Class: map_skiplistimpl_iterator
140
141 Concrete class that implements an iterator for
142 (string -> integer) mapping class map_skiplistimpl.
143 */
144 class map_skiplistimpl_iterator : public map_interface_iterator {
145 friend class map_skiplistimpl ;
146 private:
147 // The constructors are private because only function
148 // map_skiplistimpl::make_new_map_interface_iterator can construct a
149 // map_skiplistimpl_iterator object. The default order of keys and values
150 // returned by the iterator is key sorted.
151 map_skiplistimpl_iterator(); // not implemented

```

```

152 map_skiplistimpl_iterator(map_skiplistimpl *thetable);
153
154 map_skiplistimpl *mytable; // the table over which I am iterating
155 map_skiplistimpl::node *curnode; // current node, NULL for end
156
157 void markAsDone() { // iterator no longer isvalid
158     mytable = NULL;
159     curnode = NULL;
160 }
161 public:
162
163 /* Operation:    hasNext, see map_interface.h.
164 */
165 virtual
166 bool
167 hasNext() const;
168
169 /* Operation:    nextKey, see map_interface.h.
170 */
171 virtual
172 string
173 nextKey(bool isadvance, bool &isvalid);
174
175 /* Operation:    nextValue, see map_interface.h.
176 */
177
178 virtual
179 int
180 nextValue(bool isadvance, bool &isvalid);
181
182 virtual
183 ~map_skiplistimpl_iterator();
184 };
185 #endif

```

/export/home/faculty/parson/AdvDataStructures/skiplists/map_skiplistimpl.cxx

```

1 /* map_skiplistimpl.cxx -- Method implementations for
2 map_skiplistimpl.h.
3
4 CSC402, Fall, 2010, Dr. Dale Parson., Assignment 1
5
6 STUDENTS: See map_interface.h for instructions map_interface
7 and map_skiplistimpl.h for this class.
8 */
9

```

```

10 #include "map_skiplistimpl.h"
11 #include <iostream>
12 using namespace std;
13 #include <stdlib.h> // random() and srandom() (see "man random")
14
15 map_skiplistimpl::map_skiplistimpl(int maxlevel, int degree, int seed,
16     bool isgrow) {
17     string snil("NIL"); // Use this as key for trailing NIL sentinel.
18     string sdummy("SENTINEL"); // Use as key for each list's leading sentinel.
19     int idummy = 0; // use as value for any sentinel object.
20     // STUDENT WRITE CONSTRUCTOR, note that when parameter and field
21     // have the same name, you must use "this->" to access the field.
22 }
23
24 map_skiplistimpl::~map_skiplistimpl() {
25     markIteratorsAsDone();
26     // STUDENT -- Write destructor -- make sure to free all objects
27     // allocated via "new" in correct order. Note that all nodes
28     // appear in the forward[0] list.
29 }
30
31 static double
32 randomHelper() {
33     // DO NOT CHANGE THIS HELPER FUNCTION.
34     // random() returns values in a range of 0 to 2**31 -1
35     // We need values in a range of [0.0, 1.0) for Pugh's algorithm,
36     // so we divide the output of random() by twoToTheThirtyOne.
37     // This implementation adds in some paranoia in case the man page
38     // for random() is wrong by looping until we have a value in range.
39     const static double twoToTheThirtyOne = 2147483648.0;
40     double result = random() / twoToTheThirtyOne;
41     while (result < 0.0 || result >= 1.0) {
42         result = random() / twoToTheThirtyOne;
43     }
44     return result;
45 }
46
47 /**
48     Return a random level 0 through maxlevel-1, with probability
49     1 / degree of advancing at each level, yielding
50     1 / degree^level for any level. See Pugh's paper.
51 */
52 int
53 map_skiplistimpl::randomLevel() {
54     // DO NOT CHANGE THIS METHOD.
55     int lvl = 0;

```

```

56 while (randomHelper() < probability and lvl < (maxlevel-1)) {
57     lvl++;
58 }
59 return lvl ;
60 }
61
62 void
63 map_skiplistimpl::put(const string &key, const int &value) {
64     markIteratorsAsDone();
65     // STUDENT Write put according to the Python pseudocode.
66 }
67
68 bool
69 map_skiplistimpl::remove(const string &key) {
70     bool result = false ;
71     markIteratorsAsDone();
72     // STUDENT Write remove according to the Python pseudocode.
73 }
74
75 int
76 map_skiplistimpl::get(const string &key, bool &isvalid) const {
77     // STUDENT Write get according to the Python pseudocode.
78 }
79
80 int
81 map_skiplistimpl::size() const {
82     // DO NOT CHANGE THIS METHOD.
83     return listlength[0];
84 }
85
86 void
87 map_skiplistimpl::print(bool ignore) const {
88     // DO NOT CHANGE THIS METHOD.
89     node *x = forward[0]->forward[0] ; // first node beyond lead sentinel
90     while (x != NIL) { // while not at trailing sentinel
91         cout << x->key << " -> " << x->value << endl ;
92         x = x->forward[0];
93     }
94 }
95
96 void
97 map_skiplistimpl::dump() const {
98     // DO NOT CHANGE THIS METHOD.
99     for (int lvl = level ; lvl >= 0 ; lvl--) {
100         cout << "\nLEVEL " << lvl << " (count "
101             << listlength[lvl] << ") " << endl ;

```

```

102     node *x = forward[lvl];
103     while (x != 0) { // print sentinels as well as mappings
104         cout << "\t" << x->key << " -> " << x->value << endl ;
105         // Minor design flaw -- NIL should have its forward
106         // array grown whenever isgrow-based growth of maxlevel
107         // occurs. Alternately, just be careful never to walk
108         // beyond the NIL node.
109         if (x == NIL) {
110             break ;
111         }
112         x = x->forward[lvl];
113     }
114 }
115 }
116
117 // STUDENTS: Below this point the code is completed. The iterator-related
118 // methods are correct. If they crash, then there is a bug in your skiplist
119 // data structure.
120
121 map_interface_iterator *
122 map_skiplistimpl::make_new_map_interface_iterator() {
123     map_skiplistimpl_iterator *result
124     = new map_skiplistimpl_iterator(this);
125     iterators.push_back(result); // add latest iterator to back of list
126     return result ;
127 }
128
129 void
130 map_skiplistimpl::markIteratorsAsDone() {
131     // Mark all as done and then clear the list. This is called on mutators.
132     list<map_skiplistimpl_iterator *>::iterator liter ;
133     for (liter = iterators.begin(); liter != iterators.end(); liter++) {
134         map_skiplistimpl_iterator *nextone = *liter ;
135         nextone->markAsDone();
136     }
137     iterators.clear();
138 }
139
140 map_skiplistimpl_iterator::map_skiplistimpl_iterator(
141     map_skiplistimpl *thetable) {
142     mytable = thetable ;
143     curnode = mytable->forward[0]->forward[0] ; // first non-sentinel if any
144     if (curnode == mytable->NIL) {
145         markAsDone();
146     }
147 }

```

```

148
149 map_skiplistimpl_iterator::~map_skiplistimpl_iterator() {
150     if (mytable != NULL) {
151         mytable->iterators.remove(this); // Take me off the list of iterators.
152     }
153 }
154
155 bool
156 map_skiplistimpl_iterator::hasNext() const {
157     return(mytable != 0 && curnode != 0 && curnode != mytable->NIL);
158 }
159
160 string
161 map_skiplistimpl_iterator::nextKey(bool isadvance, bool &isvalid) {
162     string result("");
163     isvalid = false ;
164     if (hasNext()) {
165         result = curnode->key ;
166         isvalid = true ;
167         if (isadvance) {
168             curnode = curnode->forward[0];
169             if (curnode == mytable->NIL) {
170                 markAsDone();
171             }
172         }
173     }
174     return result ;
175 }
176
177 int
178 map_skiplistimpl_iterator::nextValue(bool isadvance, bool &isvalid) {
179     int result = 0 ;
180     isvalid = false ;
181     if (hasNext()) {
182         result = curnode->value ;
183         isvalid = true ;
184         if (isadvance) {
185             curnode = curnode->forward[0];
186             if (curnode == mytable->NIL) {
187                 markAsDone();
188             }
189         }
190     }
191     return result ;
192 }

```