CS515 Lab 11

The objective of this lab is to become familiar with implementing and testing a data structure that has randomized behavior: a skip list data structure. To start, download all files from `~cs515/public/11L`

Starter files needed in this task: `skiplist.h skiplist.cpp skiptest0.cpp skiptest.cpp`

Recall that sorted *arrays* are nice for binary search, but not for adding or removing elements. On the other hand, sorted *linked lists* support dynamic size and localized insertion/deletion, but search can be done only sequentially so its complexity is $\mathbf{O}(n)$.

A *skip list* is a probabilistic extension of the sorted linked list in order to speed up the searching process. The idea is to have *n*-levels of linked lists, with each level having probability *p* that a node exists at that level. To build a skip list, we start with all nodes in the bottom level, and then use a random number generator to determine if a node will be promoted to the next level up. To search for an element ***k*** in a skip list, we start at the top level of the skip list and follow the steps below:

- *Scan Forward Step*: move a pointer along the current level until it is at the right-most position on the present level with a key ***info*** < ***k***.
- *Drop Down Step*: if you are not yet in the bottom level, move down one level in the skip list. If you are already at the bottom level, then check if in next node's if ***info*** == ***k***. Search returns true if it matches, otherwise false.

The starting code provides a partial implementation of a skip list that stores only integer keys. You can eventually compile it and run the test drivers, including `skiptest0.cpp`. When completed, you may run the test drivers multiple times and see the different skip lists built during each run for the same test. This non-determinism is due to the random level generated for each node at runtime.

You are provided with a makefile to help you compile the test driver. Below is a sample run to compile and run the test driver `skiptest0.cpp`:

```
$ make
g++ -c -g skiplist.cpp
g++ -c -g skiptest0.cpp
g++ -g skiplist.o skiptest0.o -o skiptest0
$ ./skiptest0
Test insert
Test find
100    100    100    100
150
200    200
240    240
250    250
255    255    255    255
265
275
299    299    299    299
520    520    520
$ ./skiptest0
Test insert
Test find
100  100
150
200
240
250
255
265
275    275    275
299    299
520    520    520    520
```

Task: Complete the following two methods in the `skiplist.cpp`:

- **`insert()`** method that returns false if the given key already exists, but otherwise inserts a node of random height in the ordered lists and returns true.
- **`erase()`** method that will erase a node with the given key if it exists and return true. If the key does not exist, the method will return false.

From the test run of `skiptest0`, you see that each run will produce a skip list of different structures. This is due to the fact that the skip list is constructed with a random number generator. However, having a different skip list at each run won't help you to test or debug. ***Instead, you should fake the random number generation during the debug process*** with a sequence of pre-determined numbers (see the comment in `randomLevel()`). *Just be sure to put the "real" random number generation back before you submit!*

You need to only modify the `skiplist.cpp` file. Do **not** modify the current methods `skiplist.h`, but you may add new methods. Once you finish the implementation of the two methods, test your code with the test driver `skiptest.cpp`. You need to modify the makefile to compile new test drivers.

You must run the program using Valgrind to make sure there are no any memory errors in your programs.

Submission:

Submit the source file `skiplist.cpp`, header file `skiplist.h`, and a `README`. To submit the files in Mimir, follow the link for this lab in MyCourses. Here is a list of files you are expected to submit:

```
skiplist.h skiplist.cpp README
```

As always, do not turn in executables or object code, and make sure your submission compiles successfully on Agate. Programs that produce compile time errors or warnings will receive a zero mark (even if it might work perfectly on your home computer.) To check this, use the make command with the provided makefile, and check that your submission passes tests on Mimir.

Important:

You must include the standard comment block in each of your source files, including your name, section, date and collaboration details. You must also finish the README file along with your programs.

You should also include detailed collaboration declaration information in your comments. If you worked in pairs in this lab, each of you must include the partner's name in your program comment. Both you and your partner must complete an individual submission in order to earn a grade. If you work by yourself, you must indicate in the program comments that you have worked on your own independently.

You can resubmit as needed—your last submission will be graded. Here is a tentative grading scheme:

insert test run	30
erase test run(s)	40
Valgrind check	30