Lab Exercise #7
Due Date: Apr 9th
Skip Lists

Description:
For this lab exercise, you will implement a **Skip List.** A skip list is a probabilistic data structure used for maintaining a sorted list of elements. It allows for efficient search, insertion, and deletion operations with an average time complexity of O(log n). It employs multiple layers of linked lists, with each layer representing a different level of granularity for faster traversal. See lecture and lab presentations for more details on the concept.

starter.zip (Starter Code):
The following is a brief description of the starter code provided to you:

- **skiplist.h**
  - **class Node:**
    - **int key:** Stores the integer element associated with this node object
    - **vector<Node*> forward:** Stores a vector of pointers to the next node on each level.
      - For example:
        - node->forward[0] stores the pointer to the next node at level 0
        - node->forward[2] stores the pointer to the next node at level 2
      - The size of the forward vector of a particular node depends on the highest level at which this node is inserted. This is because the node must store the pointer to the next node at each level from level 0 to the highest level at which it is inserted.
  - **class SkipList:**
    - The SkipList class contains the following private member variables:
      - **double P:** During insertion, you must generate a random probability (between 0.0 and 1.0) - (i.e. similar to a coin flip). If the random probability < P, you can insert the node at the next level (and continue checking the probability of inserting at the level above). If the random probability >= P, then you must not insert the node at any further levels.
      - **int MAXLVL:** This member variable stores the highest level at which a node can be inserted. Even if the coin flips (i.e. random probability check mentioned above) result in the node being able to be inserted at levels > MAXLVL, you should not insert the node at levels > MAXLVL.
      - **Node* header:** This is the dummy header node (with a key of -1) that will mark the start of each level. We can assume for the purposes of this assignment that all keys that will be inserted into the skip list

will be non-negative. Hence using the key -1 will satisfy the skip list property.

- **int level**: This member variable keeps track of the highest level at which any key has been inserted (Note that level should always be <= MAXLVL).

- The SkipList declares the following public methods:
  - **Constructor**: The Constructor has been implemented for you. The constructor sets the MAXLVL and the P members based on the parameters taken in. Notice that the header is allocated with a key = -1 and allocates a vector of Node* (forward) of MAXLVL + 1 size. This would result in the header's forward pointer (at each level) pointing to nullptr (sentinel node) at the initial state of the skipped list. The level member variable is initialized to 0 as the skip list is empty.
  - **Destructor**: You must implement the destructor by deallocating all the nodes in the skiplist.
  - **void insertElement(int key)**: Inserts the key into the skip list such that the keys are maintained in sorted order. If the key already exists in the skiplist, we can leave the skip list undisturbed. The insertElement function must use the **int randomLevel()** function to decide the furthest level at which the node must be inserted
    - The int randomLevel() helper function must simulate the random probability generator and compare that to P in order to decide which is the highest level up till which the node with this key must be inserted.
  - **void deleteElement(int search_key)**: Deletes the key from the skip list such that the other keys are maintained in sorted order. If the key does not exist, we leave the skip list undisturbed.
  - **bool searchElement(int key)**: Returns true if the key exists in the skip list and false if it does not exist.
  - **void displayList()**: This method has been implemented for you for debugging purposes. You can call this method to print the current state of the skip list.

- **main.cpp**
  - You can use this file to instantiate SkipList objects and test your code.

Contract (TASK):
Your task is to implement the methods of the SkipList class. You have been provided with non-functional definitions of these methods. **You must implement these method definitions in the given skiplist.h file**, by replacing the current non-functional definitions with your functional

implementations. **Do not create an additional cpp file to define these functions, make sure that they are part of the header file.**

Deliverables and Submissions:

In a zip folder named using the format **<FirstName>-<LastName>-<UIN>-<LE7>.zip** you must submit the following file(s) to Canvas:

- **skiplist.h** (containing the functional method definitions of the SkipList class)

Do not use angular brackets in the name of the submission folder.

Grading:
Coding (20 points)
The methods of the SkipList will be tested using an automatic testing infrastructure. You will be provided with the testing infrastructure which you can use to evaluate your implementations.

The grading rubric is as follows:

- Hard Coded Small Skip List
  - Insert: 5 points
  - Delete: 5 points
    - Both insert and delete depend on search
- Random Skip List (Depends on insert, delete and search)
  - 10 points

The hard coded small skip list is primarily to check the functionality of insert and delete. The randomized skip list will be timed in order to check its efficiency. If you are implementing the skip list using the levels correctly and not relying on just a single level for your operations, you should be ok with respect to the timing; we are not providing exact runtimes to check for, as your machines may be slower/faster than ours (our Mac M1 system recorded a runtime of ~1300 ms, and our linux system on Docker recorded a runtime of ~10200 ms for the random simulation). The timing of this random simulation will be tested on our machines. The test script will output the time your implementation took to run the random simulation.

Note that we are not testing you on memory leaks on this assignment, but you are encouraged to implement the destructor to take care of the memory leaks

testInfrastructure.zip (Test Methodology):
In order to familiarize you with the auto-grading environment on our end we are providing you with an automated test infrastructure that can be used to check the correctness of your code. The testing infrastructure is constructed as follows:

- **skiplist_test.cpp**: This cpp file defines all the test functions that have been mentioned in the grading rubric above. It also runs these tests and outputs your final score.

- **test.py:** This file compiles skiplist_test.cpp in order to produce the executable. It then runs this executable which will indicate your score (as well as the test cases you have passed or failed).

Note: There is a possibility that one or more of your methods causes a segmentation fault which in turn causes the test script to crash when calling that particular method. In such a case, we will not be able to calculate your score for the assignment and you will be assigned a default grade of 0. After the grades are posted, you will be given 1 week to rectify your error(s) and resubmit the assignment. You will be subject to a 25% penalty on the grade obtained from the resulting resubmission of the assignment.

How to test your program natively (run your own test cases):
You can use main.cpp provided in the starter.zip folder to instantiate SkipList objects. You can then test the functionality of your methods by calling these methods and checking whether what they return is what you expect.
To compile your own tests in main.cpp, **you can use the makefile provided within starter.zip to compile your program.** In your terminal:

- Run **make** on the terminal (this will create an executable named main)
- Next, run **./main** on your terminal to run your native tests

How to use testInfrastructure:
To check your score using testInfrastructure, you can follow either of the two options mentioned below:

- **Option 1:** Move your skiplist.h file into the testInfrastructure directory. After this, run **python3 test.py**
- **Option 2:** Move your skiplist.h file into the testInfrastructure directory. **Ensure that the makefile in this directory is the one given to you originally in the testInfrastructure.zip folder and not in the starter.zip folder.** In your terminal:
    - Run **make** in order to compile the program (which will compile skiplist_test.cpp and create an executable named skiplist).
    - Next, run **./skiplist** on your terminal to actually run the tests and output your score.