Assignment 4: SkipList

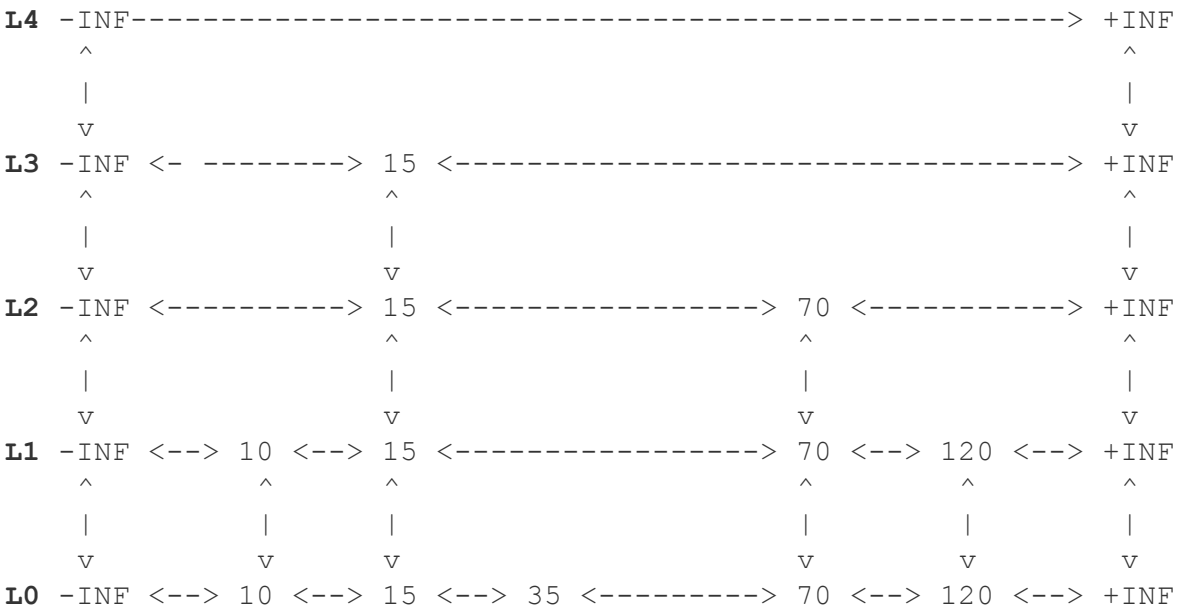
Goals: Working with dynamic arrays, pointers, doubly linked lists

For this assignment, you will write a Skip List data structure to store integers. When searching a Skip List, items can be found in `O(log n)` time. No duplicates are allowed.

A `SkipList` can have multiple levels. `SkipList` of depth 1 is similar to a doubly linked list. All elements are inserted into a single doubly linked list.

When a `SkipList`has multiple levels, all elements are inserted at level = 0. 50% of those inserted at level = 0, are also inserted at level = 1. 50% of those inserted at level = 1 are also inserted at level = 2, and so on.

There are multiple different types of implementations of Skip List data structure that you might find on the internet, make sure you are following the assignment specifications.



The above picture shows a `SkipList` with a depth of 5, Level-0 to Level-4

At **L0**, all items are inserted: 10, 15, 35, 70, 120

Some of those items (based on a coin toss, 50% chance using `(rand() % 2) == 1`) are inserted at level = 1, i.e. at L1. Some of those items are inserted at L2, and so on.

`SkipList` has front and rear guards, in our case `INT_MIN` and `INT_MAX` as the first and the last items at each of the levels. This makes it easier to add and find items.

Skip List is made up of private `SNode` elements. Each `SNode` has `SNode*` of `next`, `prev`, `upLevel` and `downLevel`, as well as `data` which holds the actual integer value stored at the node.

front and rear guards are special `SNode*` objects to make the implementation easier. The pointers to these guards are kept in arrays, named `frontGuards` and `rearGuards`, see below for details.

The public functions of `SkipList` are:

```
// default SkipList has depth of 1, just one doubly-linked list
explicit SkipList(int depth = 1);
// destructor
virtual ~SkipList();
// return true if successfully added, no duplicates
bool Add(int data);
// return true if successfully removed
bool Remove(int data);
// return true if found in SkipList
bool Contains(int data);
```

The private variables and functions of `SkipList` are:

```
class SNode {
public:
    // SNoide stores int as data
    explicit SNode(int data);
    // data for SNode
    int data;
    // link to next SNode
    SNode *next;
    // link to prev SNode
    SNode *prev;
    // link to up one level
    SNode *upLevel;
    // link to down one level
    SNode *downLevel;
};

// depth of SkipList, levels are 0 to depth-1
int depth;
// array of depth SNode* objects as frontGuards linking levels
// if depth = 2, we'd have frontGuards[0] and frontGuards[1]
SNode **frontGuards;
// array of depth SNode* objects as rearGuards linking levels
SNode **rearGuards;
```

```
    // given a SNode, place it before the given NextNode
    void addBefore(SNode *newNode, SNode *nextNode);

    // return true 50% of time,
    // each node has a 50% chance of being at higher level
    bool alsoHigher() const;
};
```

The friend function is used to display all the elements of `SkipList`

```
friend ostream &operator<<(ostream &os, const SkipList &list)
```

Make sure your implementation of `operator<<` matches the output provided below.

You may choose to have additional private functions as needed. The depth of the SkipList is given in the constructor, so frontGuards and rearGuards will have to be dynamically allocated arrays, using `frontGuards = new SNode*[depth];` where the for level L0, the frontGuard will be `frontGuards[0]` and rear guard `frontGuards[0]`

Constructor for `SNode`: set the data, set all other pointers to nullptr

SNode does not have any other functions. The data members of `SNode` are directly accessed from `SkipList`.

Constructor for `SkipList`: Dynamically allocate frontGuards and rearGuards arrays, create the special SNode* objects as guards, tie all the SNode objects together (both prev-next and up-down)

`SkipList::Add`: return false if the given value is already in level-0. For initial insertion:

1. Set SNode*, such `nextNode` to be `frontGuards[0]->next` We know that nothing can come before the f`frontGuards[0]`, so the `nextNode` from the one we will insert should be at least `frontGuards[0]->next`
2. As long as `nextNode->next` is not null and `nextNode->data < data`, keep moving nextNode to the right
3. If `nextNode->data == data`, return false stating duplicates are not allowed
4. If not, create a new SNode called `newNode`, put the data in it and call addBefore(`newNode`, nextNode); to connect all the pointers together.

After inserting into level-0, toss a coin to check if it should be inserted at higher level. If inserting at a higher level:

1. Create another node to be inserted at the higher level, `newUpper`
2. Connect `newNode` and `newUpper` (up-down)

3. Starting from `newNode` that was inserted, keep going back towards INT_MIN until you find a node that has a valid upLevel link
4. Go up one level
5. Go right towards INT_MAX
6. Call addBefore to insert `newUpper` to come before the node you just got to
7. Set `newNode` to be `newUpper`
8. Toss a coin to decide if you need to insert it at a higher level again

`SkipList::Contains`: return true if the given value is in `SkipList`. You do not want to search at level-0, that will be O(n), but you might want to start there as you build your program. Assuming you are looking for `value` (to make the explanation easier and not to confuse it with node data)

1. Start at top-left, highest level `frontGuards[depth-1]`, let's call this node `current`
2. As long as `current` is not nullptr
   a. Keep moving right as long as the `current->next->data` is less than `value`
   b. If `current->next->data` is equal to `value`,
      return `true`
      else
      set `current` to be the node down one level from `current` node
3. Go to step 2
4. If `current` is nullptr, the item is not in `SkipList`

`SkipList::~SkipList`: You will have to test your program under valgrind in CSS Linux Lab to make sure there is no memory leak. For debugging purposes, you may also want to add a static counter to increment/decrement SNode as it is getting created and deleted. You should remove the static counter before submitting your program.

Make sure you delete each of the nodes at each level AND the dynamically allocated arrays `frontGuards` and `rearGuards`.

When deleting an object, use `delete objectPtr`
When deleting an array, use `delete[] array`

Under unix, compile your code using

```
g++  -g -Wall -Wextra assignment4.cpp skiplist.cpp -o assignment4
```

You need to submit `assignment4.zip` with the following files in it. See course assignments page for instructions on how to create it

`skiplist.h` - the prototypes of `SkipList` class functions

`skiplist.cpp` - the implementation of `SkipList` class functions
`assignment4.cpp` - **extensive** tests demonstrating working of `SkipList` with different depths `output.txt` - See course assignments page for instructions on how to create it `selfassessment.txt` - See course assignments page for the template

Tips and Hints:

- Get things working for Skiplist(1) with a depth of 1, there is only a single level to work with. Make sure displaying, adding and contains works for this single level SkipList before moving on.
- Use assert statements to confirm assumptions and stop execution of the program. For example:

```
void SkipList::addBefore(SNode * newNode, SNode * nextNode) {
      assert(newNode != nullptr && nextNode != nullptr &&
            newNode->data < nextNode->data);
…
      // after nodes have been linked
      assert (newNode->next == nextNode &&
            nextNode->prev == newNode);
      assert (newNode->prev != nullptr &&
            newNode->prev->data < newNode->data);
}
```

Sample debugging output, produced by:

```
void test04() {
    SkipList s(5);
    for (int i = 0; i < 20; ++i) {
        int number = rand() % 100;
        s.Add(number);
        cout << "After adding " << number << endl;
        cout << s << endl;
    }
}
```

```
After adding 7
Level: 4 -- -2147483648, 2147483647,
Level: 3 -- -2147483648, 2147483647,
Level: 2 -- -2147483648, 2147483647,
Level: 1 -- -2147483648, 2147483647,
Level: 0 -- -2147483648, 7, 2147483647,
```

```
After adding 72
Level: 4 -- -2147483648, 2147483647,
Level: 3 -- -2147483648, 2147483647,
Level: 2 -- -2147483648, 2147483647,
Level: 1 -- -2147483648, 2147483647,
Level: 0 -- -2147483648, 7, 72, 2147483647,

After adding 23
Level: 4 -- -2147483648, 2147483647,
Level: 3 -- -2147483648, 2147483647,
Level: 2 -- -2147483648, 2147483647,
Level: 1 -- -2147483648, 23, 2147483647,
Level: 0 -- -2147483648, 7, 23, 72, 2147483647,

After adding 87
Level: 4 -- -2147483648, 2147483647,
Level: 3 -- -2147483648, 2147483647,
Level: 2 -- -2147483648, 2147483647,
Level: 1 -- -2147483648, 23, 2147483647,
Level: 0 -- -2147483648, 7, 23, 72, 87, 2147483647,

After adding 3
Level: 4 -- -2147483648, 2147483647,
Level: 3 -- -2147483648, 2147483647,
Level: 2 -- -2147483648, 2147483647,
Level: 1 -- -2147483648, 3, 23, 2147483647,
Level: 0 -- -2147483648, 3, 7, 23, 72, 87, 2147483647,

After adding 35
Level: 4 -- -2147483648, 2147483647,
Level: 3 -- -2147483648, 2147483647,
Level: 2 -- -2147483648, 2147483647,
Level: 1 -- -2147483648, 3, 23, 2147483647,
Level: 0 -- -2147483648, 3, 7, 23, 35, 72, 87, 2147483647,

After adding 67
Level: 4 -- -2147483648, 2147483647,
Level: 3 -- -2147483648, 2147483647,
Level: 2 -- -2147483648, 2147483647,
Level: 1 -- -2147483648, 3, 23, 67, 2147483647,
Level: 0 -- -2147483648, 3, 7, 23, 35, 67, 72, 87, 2147483647,

After adding 93
Level: 4 -- -2147483648, 2147483647,
Level: 3 -- -2147483648, 93, 2147483647,
Level: 2 -- -2147483648, 93, 2147483647,
Level: 1 -- -2147483648, 3, 23, 67, 93, 2147483647,
Level: 0 -- -2147483648, 3, 7, 23, 35, 67, 72, 87, 93, 2147483647,

After adding 68
Level: 4 -- -2147483648, 2147483647,
Level: 3 -- -2147483648, 93, 2147483647,
Level: 2 -- -2147483648, 93, 2147483647,
Level: 1 -- -2147483648, 3, 23, 67, 93, 2147483647,
```

```
Level: 0 -- -2147483648, 3, 7, 23, 35, 67, 68, 72, 87, 93, 2147483647,

After adding 96
Level: 4 -- -2147483648, 2147483647,
Level: 3 -- -2147483648, 93, 2147483647,
Level: 2 -- -2147483648, 93, 2147483647,
Level: 1 -- -2147483648, 3, 23, 67, 93, 96, 2147483647,
Level: 0 -- -2147483648, 3, 7, 23, 35, 67, 68, 72, 87, 93, 96, 2147483647,

After adding 49
Level: 4 -- -2147483648, 2147483647,
Level: 3 -- -2147483648, 93, 2147483647,
Level: 2 -- -2147483648, 93, 2147483647,
Level: 1 -- -2147483648, 3, 23, 49, 67, 93, 96, 2147483647,
Level: 0 -- -2147483648, 3, 7, 23, 35, 49, 67, 68, 72, 87, 93, 96, 2147483647,

After adding 33
Level: 4 -- -2147483648, 33, 2147483647,
Level: 3 -- -2147483648, 33, 93, 2147483647,
Level: 2 -- -2147483648, 33, 93, 2147483647,
Level: 1 -- -2147483648, 3, 23, 33, 49, 67, 93, 96, 2147483647,
Level: 0 -- -2147483648, 3, 7, 23, 33, 35, 49, 67, 68, 72, 87, 93, 96,
2147483647,

After adding 51
Level: 4 -- -2147483648, 33, 2147483647,
Level: 3 -- -2147483648, 33, 93, 2147483647,
Level: 2 -- -2147483648, 33, 93, 2147483647,
Level: 1 -- -2147483648, 3, 23, 33, 49, 67, 93, 96, 2147483647,
Level: 0 -- -2147483648, 3, 7, 23, 33, 35, 49, 51, 67, 68, 72, 87, 93, 96,
2147483647,

Duplicates not allowed: 23
After adding 23
Level: 4 -- -2147483648, 33, 2147483647,
Level: 3 -- -2147483648, 33, 93, 2147483647,
Level: 2 -- -2147483648, 33, 93, 2147483647,
Level: 1 -- -2147483648, 3, 23, 33, 49, 67, 93, 96, 2147483647,
Level: 0 -- -2147483648, 3, 7, 23, 33, 35, 49, 51, 67, 68, 72, 87, 93, 96,
2147483647,

After adding 5
Level: 4 -- -2147483648, 33, 2147483647,
Level: 3 -- -2147483648, 33, 93, 2147483647,
Level: 2 -- -2147483648, 33, 93, 2147483647,
Level: 1 -- -2147483648, 3, 23, 33, 49, 67, 93, 96, 2147483647,
Level: 0 -- -2147483648, 3, 5, 7, 23, 33, 35, 49, 51, 67, 68, 72, 87, 93, 96,
2147483647,

After adding 66
Level: 4 -- -2147483648, 33, 2147483647,
Level: 3 -- -2147483648, 33, 93, 2147483647,
Level: 2 -- -2147483648, 33, 93, 2147483647,
Level: 1 -- -2147483648, 3, 23, 33, 49, 66, 67, 93, 96, 2147483647,
```

```
Level: 0 -- -2147483648, 3, 5, 7, 23, 33, 35, 49, 51, 66, 67, 68, 72, 87, 93,
96, 2147483647,

After adding 1
Level: 4 -- -2147483648, 33, 2147483647,
Level: 3 -- -2147483648, 33, 93, 2147483647,
Level: 2 -- -2147483648, 1, 33, 93, 2147483647,
Level: 1 -- -2147483648, 1, 3, 23, 33, 49, 66, 67, 93, 96, 2147483647,
Level: 0 -- -2147483648, 1, 3, 5, 7, 23, 33, 35, 49, 51, 66, 67, 68, 72, 87,
93, 96, 2147483647,

After adding 9
Level: 4 -- -2147483648, 33, 2147483647,
Level: 3 -- -2147483648, 33, 93, 2147483647,
Level: 2 -- -2147483648, 1, 33, 93, 2147483647,
Level: 1 -- -2147483648, 1, 3, 23, 33, 49, 66, 67, 93, 96, 2147483647,
Level: 0 -- -2147483648, 1, 3, 5, 7, 9, 23, 33, 35, 49, 51, 66, 67, 68, 72, 87,
93, 96, 2147483647,

After adding 99
Level: 4 -- -2147483648, 33, 2147483647,
Level: 3 -- -2147483648, 33, 93, 2147483647,
Level: 2 -- -2147483648, 1, 33, 93, 2147483647,
Level: 1 -- -2147483648, 1, 3, 23, 33, 49, 66, 67, 93, 96, 2147483647,
Level: 0 -- -2147483648, 1, 3, 5, 7, 9, 23, 33, 35, 49, 51, 66, 67, 68, 72, 87,
93, 96, 99, 2147483647,

After adding 60
Level: 4 -- -2147483648, 33, 2147483647,
Level: 3 -- -2147483648, 33, 93, 2147483647,
Level: 2 -- -2147483648, 1, 33, 93, 2147483647,
Level: 1 -- -2147483648, 1, 3, 23, 33, 49, 66, 67, 93, 96, 2147483647,
Level: 0 -- -2147483648, 1, 3, 5, 7, 9, 23, 33, 35, 49, 51, 60, 66, 67, 68, 72,
87, 93, 96, 99, 2147483647,
```