Skip Lists

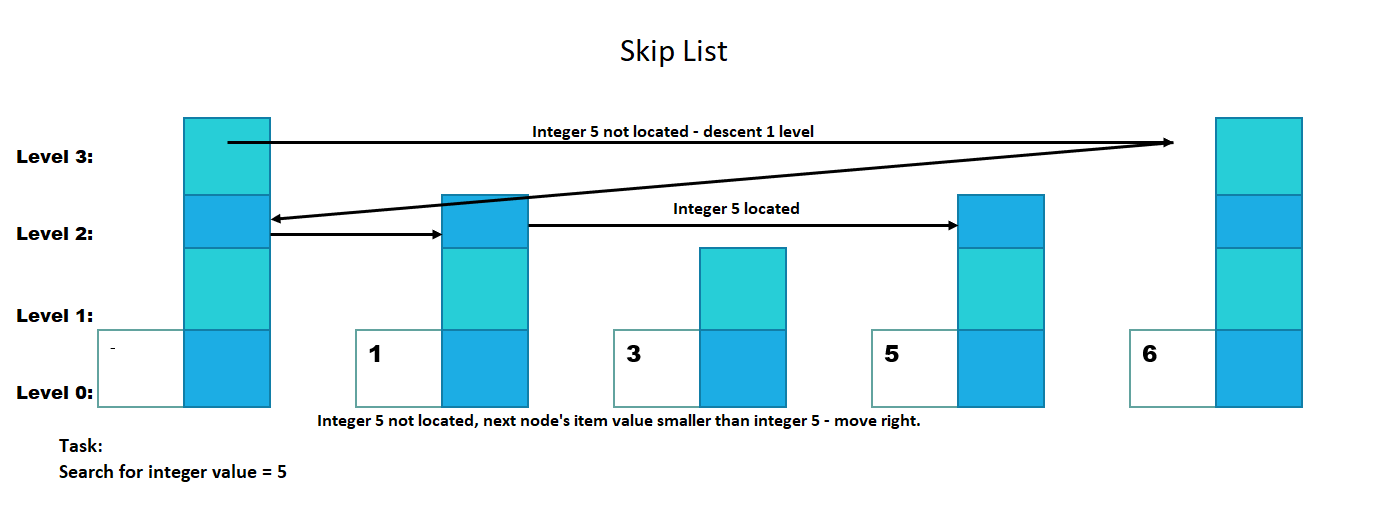
Caroline El Jazmi

Contents

**No table of contents entries found.**

# Specifications

This program implements a doubly linked list structure that will hold integer nodes and execute a skip list strategy. The skip list is a probabilistic data structure that behaves as a sorted list and has a logarithmic running-time of O(log(n)) for inserting, locating and deletion of data inside the list. The general strategy of a skip uses subsequent layers of doubly linked lists built upon a base list. Additional nodes to the skip inserted at the base-level of the list and are then consulted by a random generator at insertion time. The random generator executes a “coin-toss” which gives each node a 50% chance of being inserted into the next upper level. Generally, nodes inserted into level *i* have a 50% chance of being inserted into level *i*+1. Each additional layers above the base layer contains fewer elements but no new node. Skip lists maintain a linked hierarchy between the base layer’s individual nodes and each additional layer stacked on top of it to better assist in fast locating, insertion and deletion of nodes.



# I/O

## Input

The program will implement *cin* from the iostream header file to insert data into our list in the form of integers to:

* Initialize total # of levels in list structure
* Input new items into list structure
* Remove items from list structure
* Search for a specific item inside list structure

## Output

The program will implement an overloaded output stream *cout* from the iostream header file and cassert from the to display:

* Individual levels inside list structure.
* Integer items inside each individual levels
* If list structure is empty – outputs “Empty”
* If list contains items – displays each individual item inside each level
* Results of static\_assets while unit testing boolean methods inside SkipList class

# Error Handling

Error handing methods within the program:

* Dynamic memory management to avoid memory leaks
* Implement a custom destructor
* Null pointers set to nullptr to avoid dangling pointers
* Perform compile-time assertion checking

# Design

This program is made up of two classes. SkipList class and Main class. Main class is the driver used to perform integrated testing of our SkipList class.

The SkipList class will implement a doubly list. The list will store integer items into nodes, and a variety of methods to manipulate the list and its behavior. The SkipList class will reference to iostream, string, random, vector, and cassert located inside the Standard C++ Library**.**

## SkipList

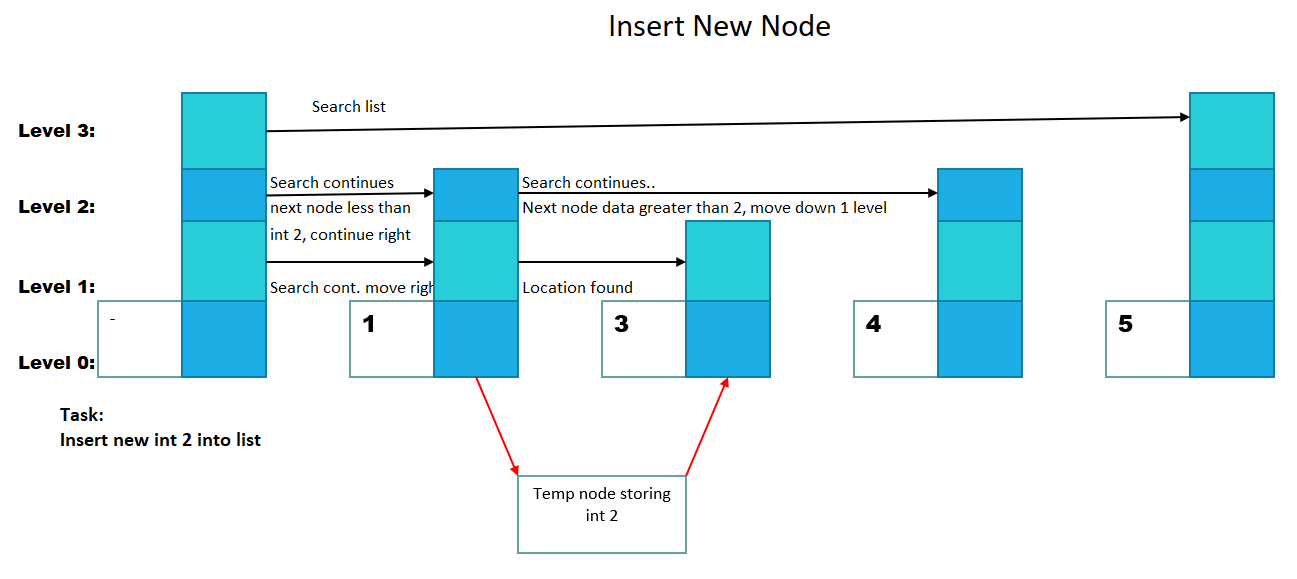
SkipList class is made up of these *public* methods:

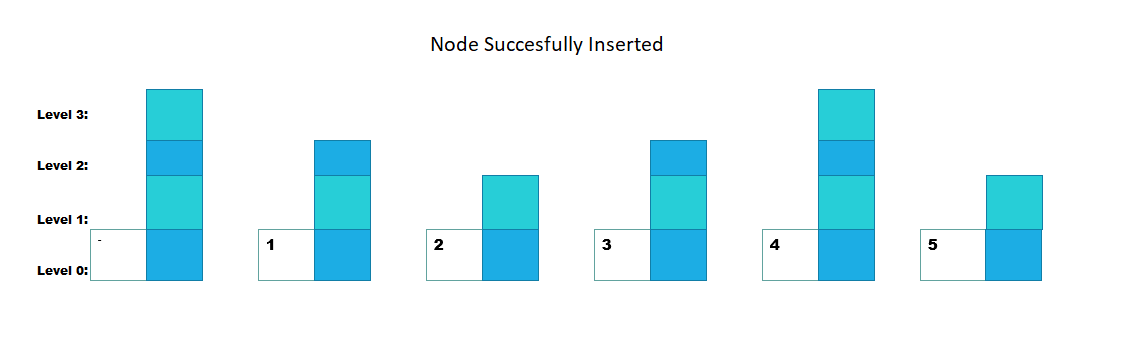
**Constructor/Destructor**

* Constructor will be made explicit to prevent implicit conversion errors. The constructor will allocate memory for new items added to the list and will initialize the maximum level in the list to 1 level.
* Destructor to deallocate storage that has been dynamically allocated.

**bool insert(int item)**

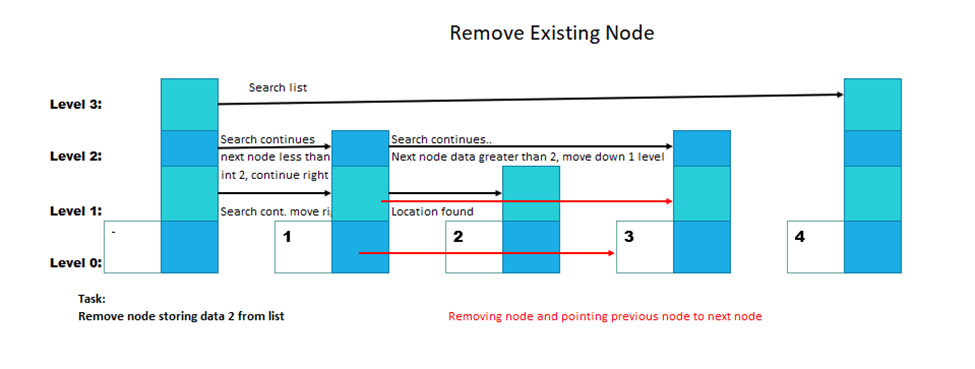
Checks if list is empty, inserts new node as new head of list structure. Checks for replicated nodes – returns false. If list is not empty, inserts a new node into the doubly linked list. Calls upon alsoHigher() to decide if node should be randomly generated into level i + 1. Returns true if new node has successfully been inserted into list structure.

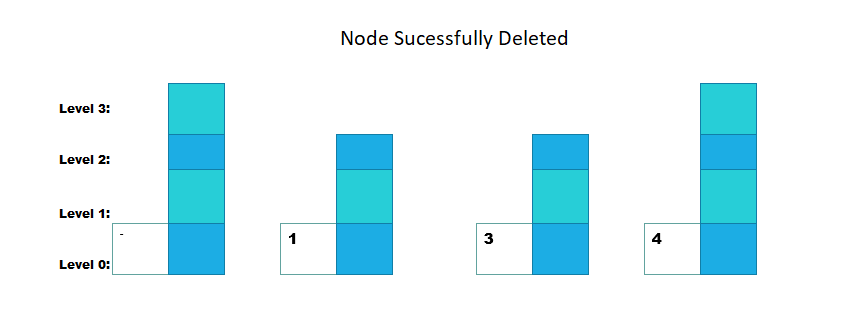




**bool erase(int item) const**

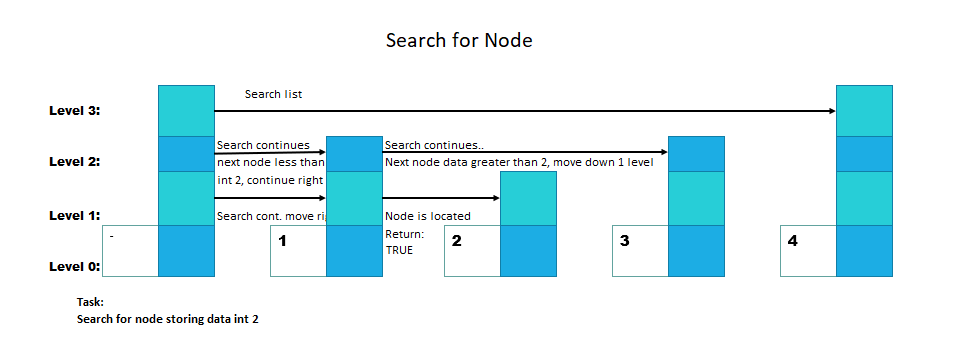
Checks if list is empty – return false if empty. If list is not empty, list is search beginning at the highest level until item is found. When item is found, item is removed from the list structure -returns true if item has been successfully removed from list structure





**bool contains(int item) const**

Checks if list is empty – returns false. If list is not empty, searches for item inside list structure beginning at the highest level (level i). If item is not found at the highest level, search continues at level i – 1. If node is found – return true. If nose is not found – return false.



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

Overloaded output stream of the friend type to the SkipList class. Friend class gives overloaded output stream access to SkipList class’s private methods. Overloaded stream operator displays individual levels inside list structure. Each levels will display its stored integer items. If level is empty, output “empty’.

**bool alsoHigher() const:**

STL random number generation capability to toss a coin. While true, gives integer a 50/50 percent chance to move up 1 level until max level is reached.

SkipList class is made up of these *private* methods:

**explicit SkipListNode(int data)**

Constructor to create node, set integer data, and declare previous and next node pointers, as well as node pointers pointing to upper and lower levels. All pointers are set to nullptr.

## Main class

The main class is the driver class of the program. The Main class will include the SkipList.h file. and will evaluate the program’s performance through integral testing of the methods inside the SkipList class. The main class will reference to iostream, string, random, vector, and cassert located inside the Standard C++ Library**.**

# Implementation

Designing, testing and debugging will take place in the Visual Studio Code - remote (C++) development on Linux lab machines

The strategy to building the SkipList program will consists of constructing the SkipList class and then the driver class Main.cpp to test the methods inside the SkipList Class. Unit testing will take place after each method implemented in the SkipList class. After every successful unit tests, commits to Github will occur.

Implementation Steps:

1. Create the SkipList class: Implement public constructor/deconstruct methods. Implement private struct SkipListNode, pointers and data members.
2. Implement individual methods,

**bool insert(int item):**

* If head pointer at base level is null, list is empty and new node becomes new head of list structure.
* If list is not empty, search through each level starting at the highest level inside the list structure, moving to in the right direction if current node’s integer item is smaller than the new integer item.
* If current node’s integer item is larger than the new integer item, search descents into level i – 1, and continues search until a sorted-location can be found.
* Dynamically inserts a new node into memory.
* Item is stored into the new nodes memory.
* Call upon addBefore function to insert new node before smaller integer item.
* After inserting new node into its proper location, call upon alsoHigher() to perform a “coin toss”
* If alsoHigher() returns true and we haven’t reached the highlest level in the list, iterate from level 0 to the highest level, insert new node and connect lower level to the upLevel\_ pointer to new node insert and this new nodes downLevel\_ pointer to the lowest level node.
* As long as the coin toss is true and you haven't reached the highest level in the skip list, you iterate through the insertion point vector from lower levels to higher levels, inserting a new node, connecting the lower level upLevel\_ pointer to this new node and this new node's downLevel\_ pointer to the lower-level node.
* Repeat alsoHigher() until returns false.
* Unit test with assert() – if successful, commit to Github.

**bool erase(int item) const:**

* If head pointer is null, list is empty, moves down 1 level.
* If list is not empty
* Searches for item begins. Pointer search to the right if item is not found and next node data is smaller than the item. If item not found, pointer moves down 1 level.
* If item is found - return true
* If item is not found - return false
* Unit test with assert()– if successful, commit to Github.

**bool contains(int item) const:**

* If head pointer at highest level(level i) s null, list is empty and search continues in level i – 1.
* If list is not empty, and next item is smaller than item being searched more, search continues in the right direction.
* If current node’s integer item is larger than the item being search for. search descents into level i – 1, and continues search until a item is found.
* Returns true if item is found – returns false if item is not found inside of list structure.
* Unit test with assert() – if successful, commit to Github

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

* Overloaded output stream declared as a friend type to SkipList class giving it access to its private members.
* For every level, from Level 0 to maximum level, outputs individual Level.
* For each level, outputs integer inside the level separated with a comma. If level is empty, outputs “empty”.

1. Create a driver class- main.cpp and make it an extension of the SkipList Class
   * Performs an integration test for each method in SkipList class as follows:

* Call SkipList constructor and input integer num. of levels I want to create in this doubly linked list.
* Add individual integer items into the list using the insert() function.
* Search for an item inside the lists using the contain() function. Test bool with static\_assert.
* Search for an item that is not located inside the list using contain() function. Test bool with static\_assert.
* Remove an item from the list using erase(). Test bool with static\_assert.
* Remove an item not located inside the list using erace(). Test bool with static\_asser.

1. Run and debug
2. Make final commit to Github