**Program 2: Skip Lists**

**Definitions**

**Skip List:** a data structure that uses probabilistic balancing and is somewhat similar to a balanced binary tree. It is a doubly linked list with multiple layers

Below is a representation of a skip list.

Graphical user interface

Description automatically generated

**Specification**

**Purpose:** The purpose of the program is to create a skip list that can add and remove objects that hold integers. It will also be able to return true or false if the list contains a certain integer. Level 0 contains all of the list’s elements, level 1 contains half of the elements from level 0, level 2 contains half of the elements from level 1 and so on. There is a 50% probability of an element being inserted into the level above it. Each level is implemented as a doubly linked list with head and tail pointer.

Below is another representation of a skip list with four levels (0-3).

Diagram, engineering drawing

Description automatically generated

**Assumptions:** The program will use nodes and pointers in order to create the skip list. The data held within the nodes will be integers. Each level will have a node for each integer, even if the integer exists in the level below it (this does not mean there are duplicates within the list). This would mean that in order to remove an integer, you need to erase every instance of that integer.

**Special Cases:** There will be no duplicate integers allowed in the list. If all head pointers point to nullptr, the insert method will be a special case for the first node.

**Input/Output:** The input will consist of integers that will be added to the list for testing. The output will be the list as formatted below via the ostream operator function:

After adding 7 and 72

Level: 4 -- empty

Level: 3 -- empty

Level: 2 -- empty

Level: 1 -- empty

Level: 0 -- 7, 72

**Testing:** Every method will be tested with various inputs and there will be no integration testing performed.

**Error handling:** The pre-conditions for methods should be enforced and invariants will be managed, but other than that there will be no exceptions thrown.

**Design**

SkipList will consist of a list of linked nodes with a nested SkipListNodes struct.

**SkipList:**

**Private:**

**struct SkipListNode** {

**explicit SkipListNode(int data):** constructor that initializes a node with the integer data and sets next, prev, upLevel, and downLevel to nullptr

**int data:** holds integer value

**SkipListNode\* next:** pointer to the next node

**SkipListNode\* prev:** pointer to the previous node

**SkipListNode\* upLevel:** pointer to the node above, which will be holding the same integer

**SkipListNode\* downLevel**: pointer to the node below, which will be holding the same integer

};

**int maxLevels**: maximum number of levels in the skip list

**SkipListNode\*\* heads\_:** dynamically allocated array that holds the head pointers

**SkipListNode\*\* tails\_:** dynamically allocated array that holds the tail pointers

**void addBefore(SkipListNode\* newNode, SkipListNode\* nextNode, int level):** this creates a pointer to a new Node which points to the next Node, works as a helper function for the insert() method. Also incremenets numberOfElements

**Static bool alsoHigher() const:** a static method that uses the STL random number generator to “toss a coin”

and if it return true, the node inserted at the insert() method will be moved up to the next level.

**Public:**

**explicit SkipList(int maxLevels = 1):** constructor that sets the default value of maxLevels to 1 if not given a value or given a value greater than levelsLimit and initializes heads\_ and tails\_

**virtual ~SkipList():** destructor that calls the clear() method

**bool insert(int item):** adds an item to the SkipList.

**bool erase(int item):** removes item with int item in the list.

**bool contains(int item) const:** returns true if the int item is found in the list

**void clear():** clears the entire list

**bool isEmpty():** returns true if the list is empty

**int getNumberOfElements():** returns the number of elements in the list

**Friend: this function can access all private members and functions of SkipList**

**ostream& operator<<(ostream& os, const SkipList& list):** displays the SkipList in a certain format

**Non-trivial methods implementation:**

**bool insert(int item):**

if (contains(item))

return false

else

create vector with pointers, insertionPoints

create current node pointer, curr

for i= maxLevels\_-1 down to 0 do

curr = heads\_[i]

while (curr next is not null and curr is greater than curr next)

curr = curr next

insertionPoints[i] = curr //this will hold places of insertion for each level

int level = 0

create new node with data

call addBefore( newNode, insertionPoints[size of vector -1], level) which inserts new node into that level

level ++

while level < maxLevels\_

if alsoHigher is true

create a new node with data

call addBefore( newNode, insertionPoints[level], level)

level++

else

return;

return true;

**bool clear():**

if (isEmpty)

return false

else

for: level 0 to maxLevel

curr ptr = head[i]

while (head[i] does not equal null)

head[i] = head next

curr downLevel = null

curr upLevel = null

curr next = null

curr prev = null

delete curr

curr = head[i]

while (tail[i] != null)

tail[i] = null

return true;

**bool erase(int item):**

if (isEmpty)

return false

else if (contains(item) == false)

return false;

else

for i= maxLevels\_-1 down to 0 do

curr ptr = head ptr at top level

while (curr next does not equal null and curr data does not equal item)

curr = curr next

curr prev = curr next

curr next = curr prev

curr equals null

delete curr

return true;

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

if (isEmpty)

return false

for i= maxLevels\_-1 down to 0 do

curr ptr = head ptr at top level

while (curr next does not equal null)

if (curr data equals item)

return true

else

curr = curr next

return false;

**void addBefore(SkipListNode\* newNode, SkipListNode\* nextNode, int level):**

if (head ptr at level != null)

newNode previous = nextNode previous

newNode next = nextNode

nextNode prev next = new Node

nextNode prev = new Node

else

head ptr at new level = newNode

tail ptr at new level = newNode

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

for i:= 0 up to maxLevel

if heads[i] equals null

print Level: i – empty

else

curr = head[i]

while (curr does not equal null)

print Level: i –

for i:= 0 up to maxLevel

if (curr next does not equal null)

print on same line as above: head[i] data,

else

print on same line head[i]

**Implementation Plan**

**Unit Test:**

Individual tests will be performed on the following functions in the order of:

**SkipList:**

1. SkipList()
   1. Constructor will be initialized with maxLevels
2. isEmpty()
   1. should return true
3. Insert()
   1. Insert 5 integers (1-5)
4. isEmpty()
   1. should return false
5. erase()
   1. remove one integer (5)
6. getNumberOfElements()
   1. should return 4
7. contains()
   1. enter in an integer that is already in the list (4)
      1. this should return true
   2. enter in an integer that is not in the list (0)
      1. this should return false
8. ostream operator<<()
   1. should print out the contents of the list
9. clear()
10. isEmpty()
    1. should return true after the list has been cleared via the clear() function

**Works Cited:**

https://canvas.uw.edu/courses/1494706/pages/program-2-description-skip-lists?module\_item\_id=13974787